A Graph-based Adversarial Imitation Learning Framework for Reliable & Realtime Fleet Scheduling in Urban Air Mobility

Prithvi Poddar [*], Steve Paul[†] and Souma Chowdhury [‡]
University at Buffalo, Buffalo, NY, 14260

The advent of Urban Air Mobility (UAM) presents the scope for a transformative shift in the domain of urban transportation. However, its widespread adoption and economic viability depends in part on the ability to optimally schedule the fleet of aircraft across vertiports in a UAM network, under uncertainties attributed to airspace congestion, changing weather conditions, and varying demands. This paper presents a comprehensive optimization formulation of the fleet scheduling problem, while also identifying the need for alternate solution approaches, since directly solving the resulting integer nonlinear programming (INLP) problem is computationally prohibitive for daily fleet scheduling. Previous work has shown the effectiveness of using (graph) reinforcement learning (RL) approaches to train real-time executable policy models for fleet scheduling. However, such policies can often be brittle on out-of-distribution scenarios or edge cases. Moreover, training performance also deteriorates as the complexity (e.g., number of constraints) of the problem increases. To address these issues, this paper presents an imitation learning approach where the RL-based policy exploits expert demonstrations yielded by solving the exact optimization using a Genetic Algorithm. The policy model comprises Graph Neural Network (GNN) based encoders that embed the space of vertiports and aircraft, Transformer networks to encode demand, passenger fare, and transport cost profiles, and a Multi-head attention (MHA) based decoder. Expert demonstrations are used through the Generative Adversarial Imitation Learning (GAIL) algorithm. Interfaced with a UAM simulation environment involving 8 vertiports and 40 aircrafts, in terms of the daily profits earned reward, the new imitative approach achieves better mean performance and remarkable improvement in the case of unseen worst-case scenarios, compared to pure RL results.

I. Nomenclature

N	=	total number of vertiports/nodes in the UAM network
V	=	set of all vertiports/nodes
E	=	set of all vertiport connections/edges
A	=	connectivity matrix for the vertiports/nodes
G_V	=	(V_V, E_V, A_V) Vertiport network expressed as a graph
G_e	=	(V_e, E_e, A_e) eVTOL network expressed as a graph
K	=	set of all eVTOLs
N_K	=	number of eVTOLs
k	=	index to denote eVTOLS in K
x_{kt}	=	vertiport where eVTOL k is at time t
S_k^{jour}	=	set of all journey of eVTOL $k \in K$
$V_{k,l}^{start}$	=	starting vertiport of the journey $j \in S_k^{jour}$ for eVTOL $k \in K$
$V_{k,l}^{end}$	=	ending vertiport of the journey $j \in S_k^{jour}$ for eVTOL $k \in K$
C	=	maximum passenger capacity for the eVTOLs
T	=	total time window for daily operation
$T_{\text{start}}^k(jk)$	=	start time of journey $j \in S_k^{jour}$ for eVTOL $k \in K$
$T_{\text{end}}^k(jk)$	=	end time of journey $j \in S_k^{jour}$ for eVTOL $k \in K$

[*]Graduate Student, Department of Mechanical and Aerospace Engineering, University at Buffalo, AIAA student member.
[†]Graduate Student, Department of Mechanical and Aerospace Engineering, University at Buffalo, AIAA student member.
[‡]Associate Professor, Department of Mechanical and Aerospace Engineering, University at Buffalo, AIAA Associate Fellow.

T^{wait}	=	waiting time
t	=	representation of a time instant (t $\in T$)
$C^{\text{park}}_{\max}$	=	max eVTOL parking capacity of a vertiport
$C^{\text{charge}}_{\max}$	=	max eVTOL charging capacity of a vertiport
$B^{k}_{\max}$	=	maximum battery capacity of eVTOL k
B^{k}_{t}	=	battery charge of eVTOL k at time instant t
$B^{charge}_{i,j}$	=	battery charge required to travel from vertiport i to vertiport j
Q	=	demand forecast model
$Q(i,j,t)$	=	demand forecast from node i to node j at time t
$F^{passenger}_{ijt}$	=	fare charged from passengers travelling from node i to j at time t
R_{ij}	=	cost of transporting 1 passenger from vertiport i to node j
$N^{passengers}_{i,l}$	=	number of passengers transported by an eVTOL on journey $j \in S^{jour}_{k}, i = V^{start}_{k,l}$
$Price^{elec}$	=	price of electricity
T^{TOD}_{i}	=	expected take-off delay at vertiport i
$P^{closure}_{i,j}$	=	probability of route closure between vertiports i and j
P^{fail}_{k}	=	probability of eVTOL $k \in K$ failing

II. Introduction

In the modern landscape of urban transportation, the emergence of Urban Air Mobility (UAM) introduces a revolutionary dimension to the concept of commuting. As cities grapple with increasing populations and traffic congestion, the concept of UAM proposes the use of electric vertical take-off and landing (eVTOL) aircraft [1] as an alternate form of automated air transportation. With a projected market size of \$1.5 trillion by 2040 [2], its economic viability will be driven by the ability to operate a sufficiently large number of eVTOLs in any given market (high-penetration), placing the domain of UAM fleet scheduling at the forefront of innovations. Following the air-space and aircraft safety constraints while being robust against dynamic environmental changes, mitigating energy footprint, and maximizing profitability makes developing an optimal scheduling policy challenging. The complexities inherent in UAM fleet scheduling are characterized by:

Dynamic environments where real-time factors such as airspace congestion, weather conditions, demand uncertainty characterized by dynamic traffic patterns, changing weather conditions, and regulatory constraints demand intelligent scheduling solutions that can swiftly adapt to real-time challenges [1].

State information sharing among all vertiports and eVTOLs is necessary for trajectory and speed adjustments to ensure the safety of the operations [3].

Scarce resources such as vertiport parking lots, charging stations, and air corridors must be optimally allocated to prevent unnecessary delays and conflicts [4].

Contemporary solutions to such scheduling problems take the form of complex nonlinear Combinatorial Optimization (CO) problems [5], which can be addressed through classical optimization, heuristic search, and learning-based approaches. While these approaches provide local optimal solutions for small UAM fleets [6–8], they often present computational complexities that render them impractical for online decision-making and larger fleets. Furthermore, despite the computational expenses, these methods would still be viable options if we disregard uncertainties in the eVTOL's operations (e.g. occasional failures of the aircraft) and environmental constraints (like the closure of air corridors due to bad weather conditions). However, these factors cannot be disregarded because of the safety concerns they impose, making it especially challenging to use such approaches.

In the pursuit of addressing these complexities and presenting an efficient online scheduler, we extend our previous work in [9] by proposing a specialized Graph Neural Networks (GNN) [10, 11] based adversarial imitation learning framework that learns from the optimal schedules (expert data) generated by classical optimization algorithms and builds upon existing multi-agent task allocation methods for scaling up to larger UAM fleets. We begin by posing the UAM fleet scheduling problem as an Integer Non-Linear Programming (INLP) problem with a small number of eVTOLs and vertiports and generate optimal solutions using an elitist Genetic Algorithm. These form the expert demonstrations for the imitation learning policy. We use the Generative Adversarial Imitation Learning (GAIL) [12] algorithm to train the GNN policy. Then, we establish how a graph representation efficiently encodes all the state-space information of the vertiports and the eVTOLs, thus motivating the use of GNNs over regular neural networks. We also present a comparative study between a pure reinforcement learning-based method and a Genetic Algorithm, that demonstrates edge cases where standard RL methodologies fail to generate an optimal solution and require access to expert data to train an optimal policy. Finally, we compare our results with standard RL-based and Genetic Algorithm based solutions and demonstrate that our method performs better than RL-based methods, in terms of profitability and while it performs equally to a Genetic Algorithm, it is significantly faster in it terms of computational times when compared to the Genetic Algorithm. Fig.1 presents a diagrammatic representation of the UAM fleet management problem.

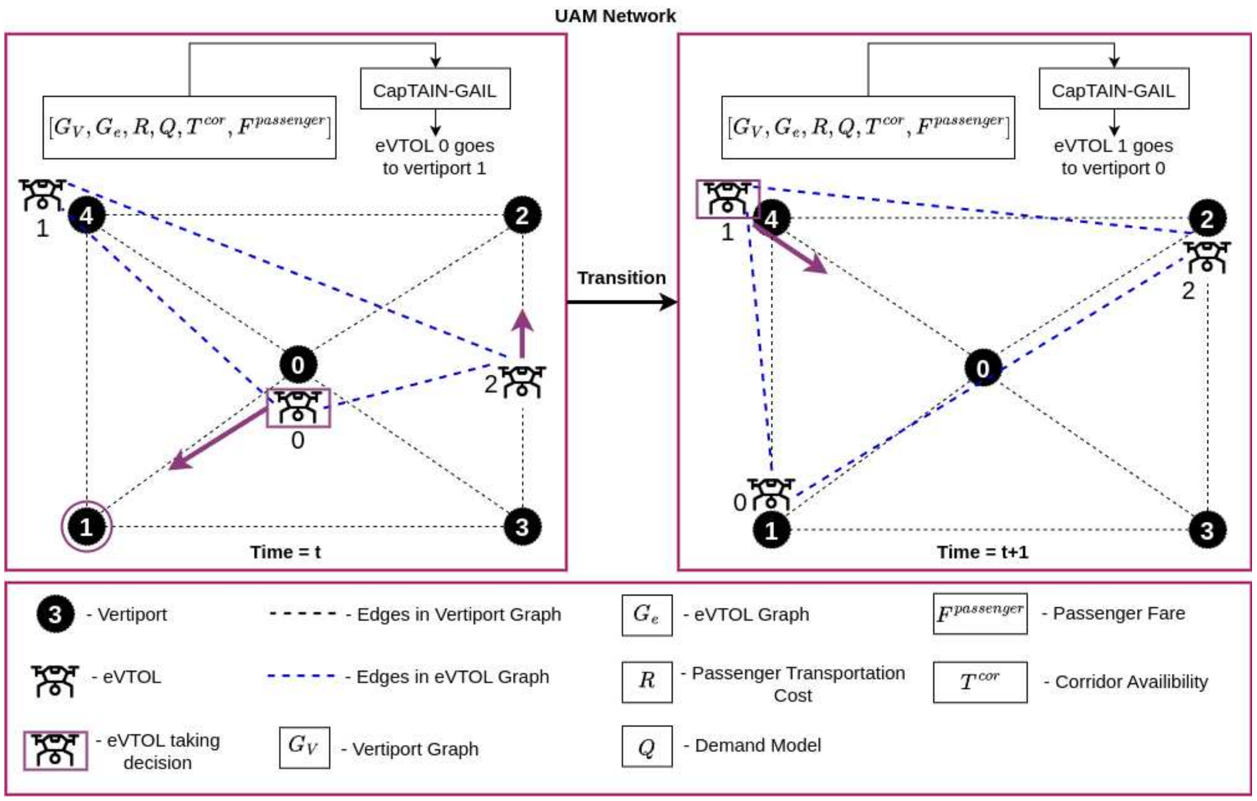
UAM Network
$[G_V, G_e, R, Q, T^{cor}, F^{passenger}]$
CapTAIN-GAIL
eVTOL 0 goes to vertiport 1
Time = t
Transition
eVTOL 1 goes to vertiport 0
Time = t+1
- Vertiport
- Edges in Vertiport Graph
G_e - eVTOL Graph
$F^{passenger}$ - Passenger Fare
- eVTOL
- Edges in eVTOL Graph
R - Passenger Transportation Cost
T^{cor} - Corridor Availibility
- eVTOL taking decision
G_V - Vertiport Graph
Q - Demand Model

Fig. 1 A visual representation of the UAM fleet management problem. CapTAIN-GAIL is the imitation learning policy that takes as input, the information from the vertiport and eVTOL graphs along with operational costs, passenger demands, and air-corridor availabilities and decides the next vertiport that the decision-making agent should go to.

III. Related Works

The existing body of work in Urban Air Mobility (UAM) fleet planning has witnessed notable growth, propelling the optimization and learning formalism to advance this emerging concept. However, a considerable portion of this literature neglects crucial guidelines outlined by the FAA [1] concerning UAM airspace integration, particularly in aspects such as air corridors, range/battery constraints, and unforeseen events like route closures due to adverse weather or eVTOL malfunctions [7, 13, 14]. Traditional methods, including Integer Linear Programming (ILP) and metaheuristics, prove inadequate for efficiently solving related NP-hard fleet scheduling problems [15–20]. For context, we focus on hour-ahead planning to allow flexibility in adapting to fluctuating demand and route conditions affected by weather and unforeseen aircraft downtime. While learning-based methods have shown promise in generating policies for combinatorial optimization problems with relatable characteristics [11, 21–27], their current forms often oversimplify complexities or tackle less intricate problem scenarios. In response, our paper introduces a centralized learning-based approach tailored to address the complexities and safety guidelines inherent in UAM fleet scheduling. Our proposed approach involves the creation of a simulation environment that encapsulates these complexities, facilitating the training of a policy network for generating hour-ahead sequential actions for eVTOLs across a generic 12-hour operational day. The policy network integrates a Graph Capsule Convolutional Neural Network (GCACPN) [28] for encoding vertiport and eVTOL state information presented as graphs, a Transformer encoder network for assimilating time-series data on demand and fare, and a feedforward network for encoding passenger transportation cost. Additionally, a Multi-head Attention mechanism is employed to fuse the encoded information and problem-specific context, enhancing sequential decision-making [25, 29].

IV. Problem Description and Formulation

A. UAM Fleet Scheduling as an Integer Non-Linear Programming Problem

We begin by posing the UAM fleet scheduling problem as an optimization problem [9], enabling us to use a Genetic Algorithm to generate the expert solution. Consider a UAM network comprising N vertiports and N_K number of eVTOLs, each with a maximum passenger carrying capacity of $C(=4)$. Let V and K denote the set of all vertiports and eVTOLs, respectively, with each vertiport $i \in V$ capable of accommodating a maximum of $C_{max}^{park}(=10)$ number of eVTOLs while also having charging stations for $C_{max}^{charge}(=6)$ eVTOLS. Some vertiports might not have charging stations and are called vertistops ($V_S \subset V$). We consider there to be 4 air corridors between two vertiports with two corridors for each direction. Each vertiport $i \in V$ has an expected take-off delay T_i^{TOD} that affects every eVTOL taking off, and we consider this estimated take-off delay to be less than 6 minutes at each vertiport. The probability of route closure between any two vertiports i and j is considered to be $P_{i,j}^{closure}(\leq 0.05)$ while the probability of an eVTOL k becoming dysfunctional is considered to be $P_k^{fail}(\leq 0.005)$. Every episode considers a randomly assigned take-off delay ($\leq$ 30 mins) that is drawn from a Gaussian distribution with mean T_i^{TOD} and a standard deviation of 6 minutes. Additional assumptions that are made regarding the problem setup are: 1) The decision-making is done by a central agent that has access to the full observation of the states of the eVTOLs and the vertiports, and 2) An eVTOL can travel between any two vertiports given it has sufficient battery charge and the resistive loss of batteries is negligible.

We define R_{ij} as the operation cost of transporting a passenger from vertiport i to j while $F_{ijt}^{passenger}$ is the price a passenger is charged for traveling from i to j at time t, that is based on passenger demand. We follow the demand model as mentioned in our previous work [9]. Let $Q(I, j, t)$ represent the demand between vertiports i and j at time t. A subset of vertiports $V_B \subset V$ are considered to be high-demand vertiports and we account for two peak hours of operation: $8:00-9:00$ am (T^{peak1}) and $4:00-5:00$ pm (T^{peak2}) such that V_B experience high demands during both peak hours. The demand is high from $V - V_B$ to V_B during T^{peak1} and vice-versa during T^{peak2}. The passenger fare is computed by adding a variable fare $F^{passenger}$ to a fixed base fare of $F^{base} = \$5$. The variable fare between two vertiports i and j is dependent on the passenger demand $Q(i, j, t)$ and the operational cost $R_{i,j}$ and is computed as $F_{i,j,t}^{passenger} = R_{i,j} \times Q_{factor}(i, j, t)$, where $Q_{factor}(i, j, t) = \max(\log(Q(i, j, t)/10), 1)$. A constant electricity pricing, $Price^{elec} = \$0.2$/kWh is used [30].

The eVTOL vehicle model is considered to be the City Airbus eVTOL [31] with a battery model similar to the one described in [8]. The aircraft has a maximum cruising speed of 74.5 mph and a passenger seating capacity of 4. The operating cost of this vehicle is \$0.64 per mile [8]. It has a maximum battery capacity of $B_{max} = 110$ kWh. Let B_t^k represent the charge left in eVTOL k at time t and if the eVTOL travels from vertiport i to j, then the charge at the next time step is $B_{t+1}^k = B_t^k - B_{i,j}^{charge}$, where $B_{i,j}^{charge}$ is the charge required to travel from i to j.

We consider an operating time horizon of T hours (with start time T^{start} and end time T^{end}) and the aim of the UAM scheduling problem is to maximize the profits earned in the T hours. This is achieved by smartly assigning (based on the demand, battery charge, operation cost, etc.) a journey to an eVTOL during each decision-making instance $t \in T$. A journey is defined as the commute of an eVTOl between two vertiports i and j. If $i = j$, the eVTOL has to wait for $T^{wait}(=15)$ minutes at the vertiport before assigning a journey in the next decision-making instance. We assume that each eVTOL can take off at any time within the T horizon, and we consider the schedule for a single-day operation where the operational time begins at 6:00 AM (T^{start}) and ends at 6:00 PM (T^{end}).

Optimization formulation: Based on the notations described above, the optimization problem is formulated as follows (similar to our previous work [9]). For every eVTOL $k \in V_e$, let S_k^{jour} be the set of journeys taken during time period of T and, $N_{i,l}^{passengers}$ be the number of passengers transported during the trip, $l \in S_k^{jour}$ by eVTOL k. Let B_l^k be the battery charge of eVTOL k just before its l'th journey, $V_{k,l}^{start}$ and $V_{k,l}^{end}$ be the respective start and end vertiports of eVTOL k during it l'th journey, $T_{k,l}^{takeoff}$ and $T_{k,l}^{landing}$ be the corresponding takeoff time and landing time. Then the objective of the optimization problem is to maximize the net profit z that can be computed by subtracting the operational costs C^O and the cost of electricity C^E from the revenue generated. These values are computed as:

$$C^O = \sum_{k \in K} \sum_{l \in S_k^{jour}} N_{i,l}^{passengers} \times R_{i,j} \quad , i = V_{k,l}^{start}, j = V_{k,l}^{end} \tag{1}$$

$$C^E = \sum_{k \in K} \sum_{l \in S_k^{jour}} Price^{elec} \times B_{i,j}^{charge} \quad , i = V_{k,l}^{start}, j = V_{k,l}^{end} \tag{2}$$

$$\text{Revenue} = \sum_{k \in K} \sum_{l \in S_k^{jour}} N_{i,l}^{passengers} \times F_{i,j,t}^{passenger} \quad , i = V_{k,l}^{start}, j = V_{k,l}^{end} \tag{3}$$

Therefore, the objective function can be formulated as:

$$\max z = \text{Revenue} - C^O - C^C \tag{4}$$

With the following constraints:

$$N_{i,l}^{passengers} = min(C, Q_{act}(I, j, t)) \quad , i = V_{k,l}^{start}, j = V_{k,l}^{end}, t = T_{k,l}^{takeoff} \forall l \in S_k^{jour} \tag{5}$$

$$B_{T_{k,l}^{takeoff}}^{k} > B_{i,j}^{charge} \quad , i = V_{k,l}^{start}, j = V_{k,l}^{end}, \forall k \in K, \forall l \in S_k^{jour} \tag{6}$$

$$C_{i,t}^{park} \leq C_{max}^{park} \quad , \forall i \in V, \forall t \in [T^{start}, T^{end}] \tag{7}$$

$$V_{l,k}^{end} \neq i \quad , \text{if} A^{i, V_{l,k}^{end}} = 0, \forall i \in V, \forall k \in K, \forall l \in S_k^{jour} \tag{8}$$

Where $Q_{act}(I, j, t)$ is the actual demand, and A is a matrix that represents the availability of routes between vertiports. $A_{ij} = A_{ji} = 1$ if the route between i and j is open and $= 0$ if the route is closed.

B. Generating the Expert Demonstrations

Given the INLP problem formulation in Sec.IV.A, we use an elitist Genetic Algorithm (GA) to generate the expert solutions for a small number of eVTOLs ($N_k = 40$) and vertiports ($N = 8$). The GA uses a population size of 100, max iterations of 100, mutation probability of 0.1, elite ratio of 0.01, and cross-over probability of 0.5. The GA is implemented in batches of 60 decision variables which are then simulated sequentially and the objective functions are computed. Upon simulating a batch of the GA solution, we achieve an updated state of the environment which is then used to compute the next 60 decision variables. This process is repeated until the episode is over. Each decision variable takes an integer value between 1 and N. We generate the GA solutions for 100 different scenarios (for each scenario, the locations of the vertiports stay fixed) which are then used as expert demonstrations for the imitation learning algorithm.

C. MDP Formulation

Having expressed the UAM fleet scheduling problem as an INLP problem, we now define it as a Markov Decision Process (MDP) that sequentially computes the action for each eVTOL at the decision-making time instant $t \in T$. The state, action, reward, and the transitions are described below:

State Space: The state-space at time t consists of information about **1)** the vertiports, represented as a graph G_V, **2)** the eVTOLs, represented as a graph G_e, **3)** the passenger demand model Q, **4)** passenger fare $F^{passenger}$, **5)** operational costs R (computed based on the per mile operational cost of the eVTOL, the passenger demands and the price for electricity), and **6)** time when it is safe to launch an eVTOL to the corridors $T^{cor} \in \mathbb{R}^{N \times N \times 2}$.

At any decision-making time step, the state information is processed by a transformer-aided multihead-attention graph capsule convolutional neural network (that was presented in [9]) that acts as the policy network for the imitation learning algorithm. Details about the policy network have been discussed in V.A. Further details on the state space are as follows:

Graph Representation of the Vertiport Network: $G_V = (V_v, E_v, A_v)$ represents the vertiport network as a graph where $V_v (= V)$ is the set of vertiports, E_v represents the set of edges/routes between the vertiports, and A_v represents the adjacency matrix of the graph. To consider the route closure probability, a weighted adjacency matrix is computed as $A_v = (1_{N \times N} - P^{closure}) \times A$. The node properties δ_i^t of the vertiport $i \in V_v$ at time t are defined as $\delta_i^t = [x_i, y_i, C_{i,t}^{park}, T_i^{charge}, T_i^{TOD}, I_i^{vstop}]$, where (x_i, y_i) are the x-y coordinates of the vertiport, $C_{i,t}^{park}$ is the number of eVTOLs currently parked, T_i^{charge} is the earliest time when a charging station will be free, T_i^{TOD} is the expected take-off delay, and I_i^{vstop} is a variable that takes the value 1 if the node is a vertistop and 0 if the node is a vertiport.

Graph Representation of the eVTOLs: $G_e = (V_e, E_e, A_e)$ represents the graph that encodes the information about the eVTOL network. V_e represents the set of eVTOLs, E_e represents the set of edges, and A_e represents the adjacency matrix. G_e is considered to be fully connected and each eVTOL $k \in V_e$ is represented by it properties i.e. $\psi_k^t = \left[x_k^d, y_k^d, B_t^k, T_k^{\text{flight}}, T_k^{\text{dec}}, P_k^{\text{fail}}\right]$, $\psi_i^t \in \mathbb{R}^6$. Here, $x_k^d, y_k^d)$ represent the coordinates of the destination vertiport, B_t^k is the current battery level, T_k^{flight} is the next flight time, T_k^{dec} is the next decision making time, and P_k^{fail} is the probability of failure.

Action Space: At each decision-making time instance, each agent takes an action from the available action space. The action space consists of all the available vertiports. Therefore the action space will be of size N_K. During a decision-making step, if an agent chooses the vertiport at which it currently is, then it waits for 15 minutes in the vertiport, until it makes a new decision.

Reward: We consider a delayed reward function where the agent gets a reward only at the end of an episode. The reward is ratio of the profit earned to the maximum possible profit in an episode, i.e. $\sum_{i \in V, j \in V, t \in [T^{start}, T^{end}]} (Q(i, j, t) \times F_{i,j,t}^{passenger})$.

Transition Dynamics: Since demand and electricity pricing can vary from that of the forecasted values, the transition of the states is considered to be stochastic. The transition is an event-based trigger. An event is defined as the condition that an eVTOL is ready for takeoff. As environmental uncertainties and communication issues (thus partial observation) are not considered in this paper, only deterministic state transitions are allowed.

V. Proposed Graph-based Adversarial Imitation Learning Approach

We propose an adversarial imitation learning formulation to train a transformer-aided GNN-based policy network called CapTAIN [9] that will assign actions to the eVTOLs. The policy network takes the state information from all the eVTOLs and vertiports as input and assigns an action to the eVTOLs that are ready for take-off. The policy is trained using the generative adversarial imitation learning (GAIL) [12] algorithm that uses the solutions generated by the GA as the expert demonstrations. The following section presents further information about the GAIL algorithm and the state encoding and decoding.

A. Policy Network

We use the **Cap**sule **T**ransformer **A**ttention-mechanism **I**ntegrated **N**etwork (CapTAIN) [9] as the policy network for GAIL. CapTAIN combines graph neural networks and transformers to encode the state-space information and used a multi-head attention-based decoder to generate the action.

The information from the veriports and eVTOLs (G_V and G_e) are first passed through a Graph Capsule Convolutional Network (GCAPCN), introduced in [28], which takes the graphs as the inputs and generates the feature embeddings for the vertiports and eVTOLs. Simultaneously, a transformer architecture is used to compute learnable feature vectors for the passenger demand model and the passenger fares (information that can be represented as time-series data). Additionally, a simple feed forward network computes the embeddings for the passenger transportation cost R which can be represented as a $N \times N$ matrix, and another feed forward network processes the corridor availability. We keep a track of the time at which it is safe for a new eVTOL to enter a corridor, subject to various safety restrictions and minimum separation, using a $N \times N \times 2$ tensor, T^{cor}, which is flattened and fed into the feed forward network. The outputs from the transformer, GCAPCN, and the feed forward networks form the encoded embedding which is then passed through a multi-head attention [29] decoder to generate the probabilities of choosing each action, for the decision-making agent. Further technical details about CapTAIN have been discussed in [9].

B. Training with Generative Adversarial Imitation Learning

The policy network π is trained on the expert demonstrations generated by the Genetic Algorithm, using Generative Adversarial Imitation Learning (GAIL), an imitation learning approach introduced in [12]. The policy is learnt using a two-player zero-sum game which can be defined as:

$$\underset{\pi}{\text{argmin}} \underset{D \in (0,1)}{\text{argmax}} \, \mathbb{E}_{\pi} \left[\log D(s, a)\right] + \mathbb{E}_{\pi_E} \left[\log\left(1 - D(s, a)\right)\right] - \lambda H(\pi) \tag{9}$$

Where π_E is the policy followed by the expert demonstration. D is a discriminator that solves a binary classification problem $D : \mathcal{S}\times\mathcal{A} \to (0, 1)$, where $\mathcal{S}$ represents the state-space and $\mathcal{A}$ represents the action-space. D tries to distinguish between the distribution of data generated by π from the expert data generated by π_E, When D cannot distinguish data generated by the policy π from the true data, then π has successfully matched the true data. $H(\lambda) = \mathbb{E}_\pi [-\log \pi(a|s)]$ is the entropy. Figure 2 shows the overall learning framework.

The discriminator (approximated by a neural network) tries to maximize the objective, while the policy π tries to minimize it. GAIL alternates between an Adam gradient [6] step on the parameters w of D_w with the gradient:

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_w \log D_w(s, a)] + \hat{\mathbb{E}}_{\mathcal{D}} [\nabla_w \log 1 - D_w(s, a)] \tag{10}$$

and a Trust Region Policy Optimization (TRPO) [7] gradient step on the parameters θ of $\pi_t heta$ which minimizes a cost function $c(s, a) = \log D_{w_{i+1}}(s, a)$ with the gradient:

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_\theta \log \pi_\theta(a|s)Q(s, a)] - \lambda\nabla_\theta H(\pi_\theta) \tag{11}$$

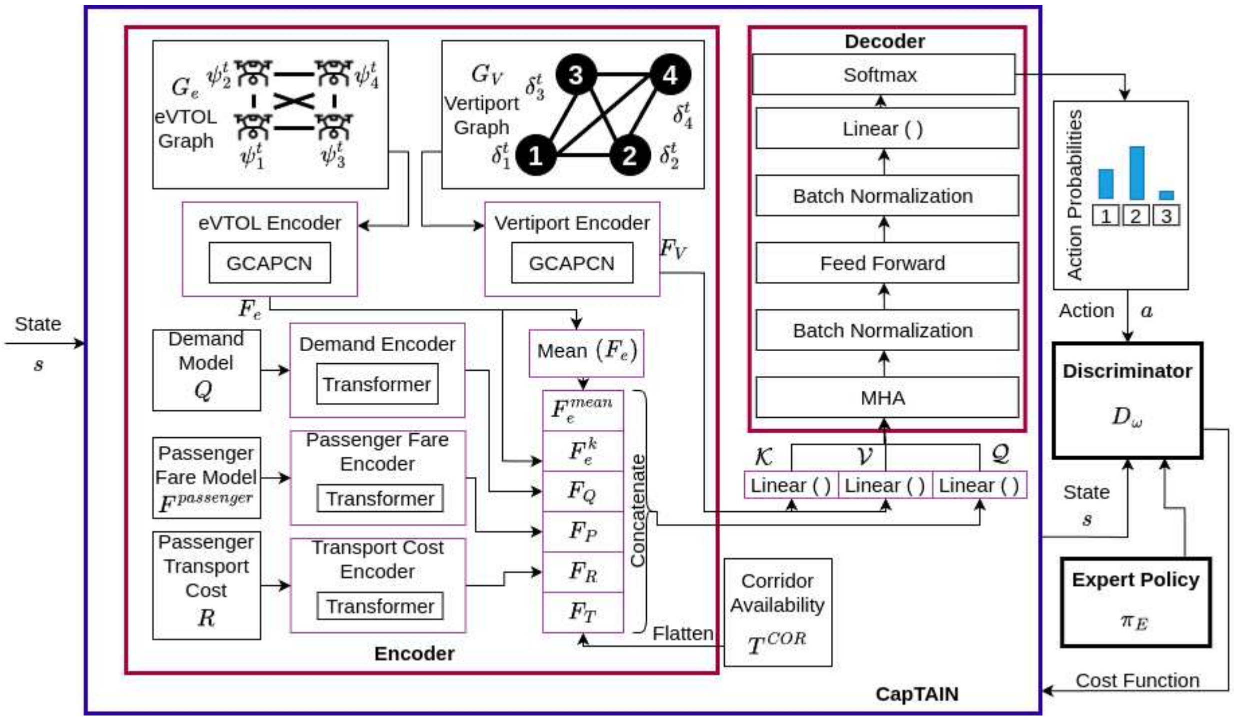

Fig. 2 The learning framework for CapTAIN-GAIL

VI. Experiments and Results

A. Simulation Details

We use the simulation environment presented in [9] that is implemented in Python and uses the Open AI Gym environment interface. We consider a hypothetical city covering an area of 50×50 sq. miles with 8 vertiports (2 of which are vertistops) and 40 eVTOLs. The locations of the vertiports remain the same throughout training and testing the algorithm. The rest of the operational details stay the same, as discussed above. To train the CapTAIN policy network, we use the GAIL implementation from **imitation** [32], a python package for imitation learning algorithms, and PPO [33] from **stable-baselines3** [34].

B. Training Details

We begin by training the **CapTAIN** policy using just PPO, for 1.5 million steps. Next, we generate the expert demonstrations for 100 scenarios (un-seen by CapTAIN during training) using a standard elitist Genetic Algorithm

(**GA**) (using the parameters as described in Sec.IV.B) and compare the performance of CapTAIN against GA. These test scenarios are generated by fixing the seed values for the random number generators in the simulation. Finally, we train the imitation learning policy (**CapTAIN-GAIL**) on the expert demonstration, with a soft start by beginning the training with the pre-trained weights from CapTAIN, and finally compare the performance of CapTAIN-GAIL against CapTAIN and GA. All the training, experiments, and evaluations are computed on a workstation running Ubuntu 22.04.4 LTS and equipped with an Intel Core i9-12900K processor and Nvidia GeForce RTX 3080 Ti graphics processor. Further evaluation details and results are discussed in the sections ahead.

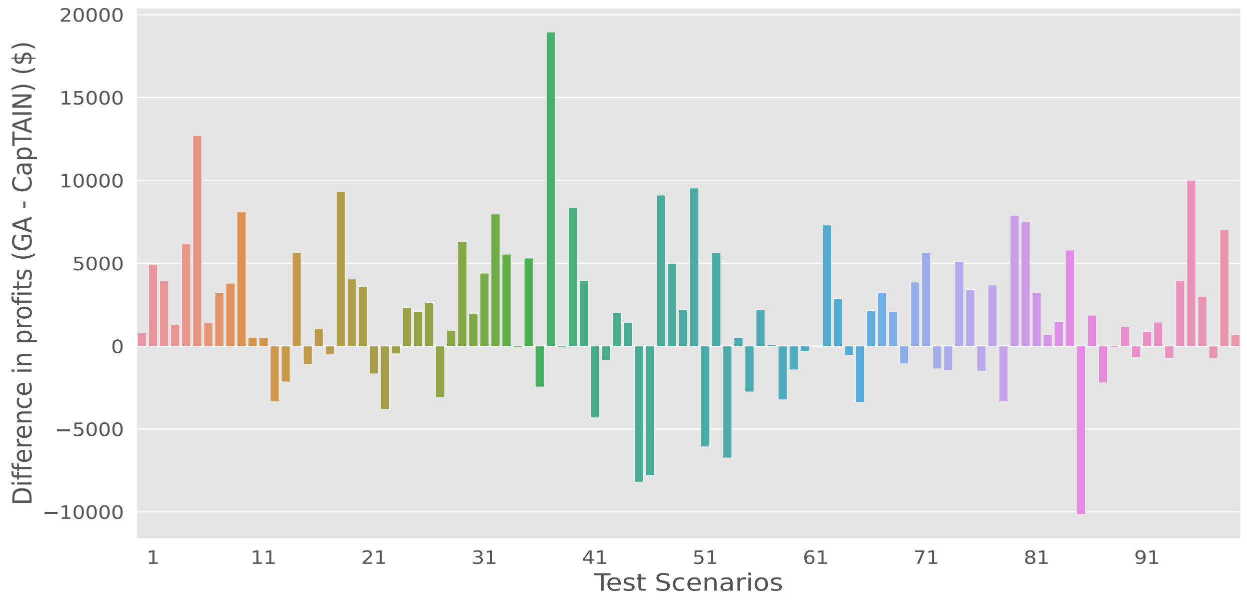

Fig. 3 Difference in the profits earned by GA and CapTAIN, Positive values indicate better performance by GA and negative values indicate better performance by CapTAIN.

C. Comparing CapTAIN vs. Genetic Algorithm

We begin by testing the performance of a policy trained purely using reinforcement learning (i.e. CapTAIN trained using PPO) against GA. The daily profit earned by both the algorithms in each test scenario, is used to compare their performance and Fig.3 plots the difference in the profits earned by GA and CapTAIN in each of the 100 test scenarios.

It can be seen that GA generates more profits in the majority of scenarios (**GA performs better in 66 out of 100 cases**), with **GA generating an average profit of $17603 while CapTAIN generating an average profit of $15727.** To test the significance of this difference, we perform a statistical T-test with the null-hypothesis being that both methods generate the same amount of profit. The **p-value of the test turns out to be 3.08×10^{-5} (< 0.05)** which means that GA has a significant statistical advantage over CapTAIN. This motivates the use of imitation learning for training a policy that can mimic GA while being significantly more efficient than GA in terms of computational time.

D. Evaluating CapTAIN-GAIL

We train the imitation learning policy with a soft start, i.e. we use the pre-trained CapTAIN policy from Sec.VI.C as the initial policy for GAIL. The generator policy in GAIL is trained for 20 iterations with 20000 steps in each iteration. The trained imitation learning policy is then tested on the 100 test cases and its performance is compared against CapTAIN and GA. We further test the generalizability of CapTAIN-GAIL on a new set of 100 unseen test cases that were not a part of the expert demonstrations.

1. Average Profits

Fig.4 plots the average profits earned by all three methods, in the test scenarios that were a part of the expert demonstrations. It can be noticed that CapTAIN generates the least profit while CapTAIN-GAIL and GA almost earn equal profits on average. A more detailed analysis of the performance of the three methods is discussed below.

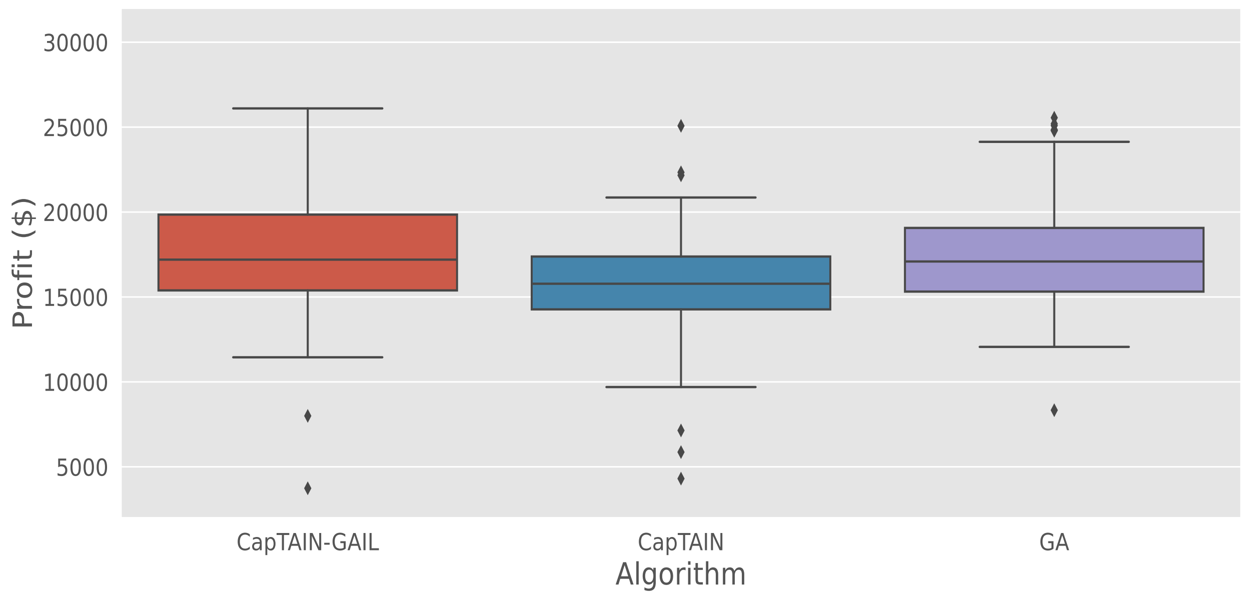

Fig. 4 Comparing the average profits earned by all the 3 methods

2. CapTAIN-GAIL vs. CapTAIN

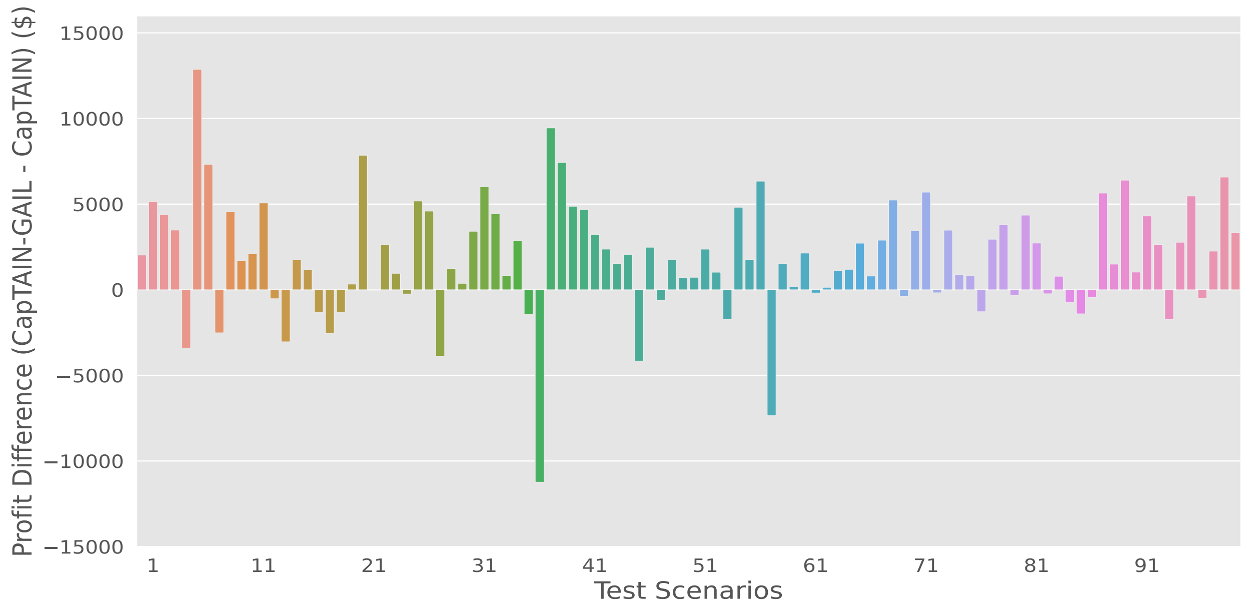

Fig. 5 Difference in the profits earned by CapTAIN-GAIL and CapTAIN, Positive values indicate better performance by CapTAIN-GAIL and negative values indicate better performance by CapTAIN.

Fig.5 compares the difference in the profits earned by CapTAIN-GAIL and CapTAIN. CapTAIN-GAIL generates more profits than CapTAIN in **73 out of 100 cases**, with the **average profit of CapTAIN-GAIL being $17587 while the average profit of CapTAIN is $15727**. To statistically evaluate the significance of this difference, we conduct a T-test with the null-hypothesis being that both the algorithms generate equal average profits. With a **p-value of**

$\mathbf{6.55 \times 10^{-5} (< 0.05)}$, we have statistical evidence that CapTAIN-GAIL performs significantly better than CapTAIN.

3. CapTAIN-GAIL vs. GA

Next, we evaluate the performance of CapTAIN-GAIL against GA. With Fig.6 plotting the difference in the profits earned by these two methods, we can see that CapTAIN-GAIL performs better than GA in **55 out of the 100 cases**, which is roughly half the number of test cases. The **average profit earned by GA is \$17603 while the average profit earned by CapTAIN-GAIL is \$17587**. Upon performing a T-test with the null hypothesis being that both methods earn the same average profit, we get a **p-value of** $\mathbf{0.97 (> 0.05)}$ which indicates that statistically, both methods have a similar performance. This result matches our expectations since the imitation learning policy should at least perform equal to the expert when tested on the expert demonstrations.

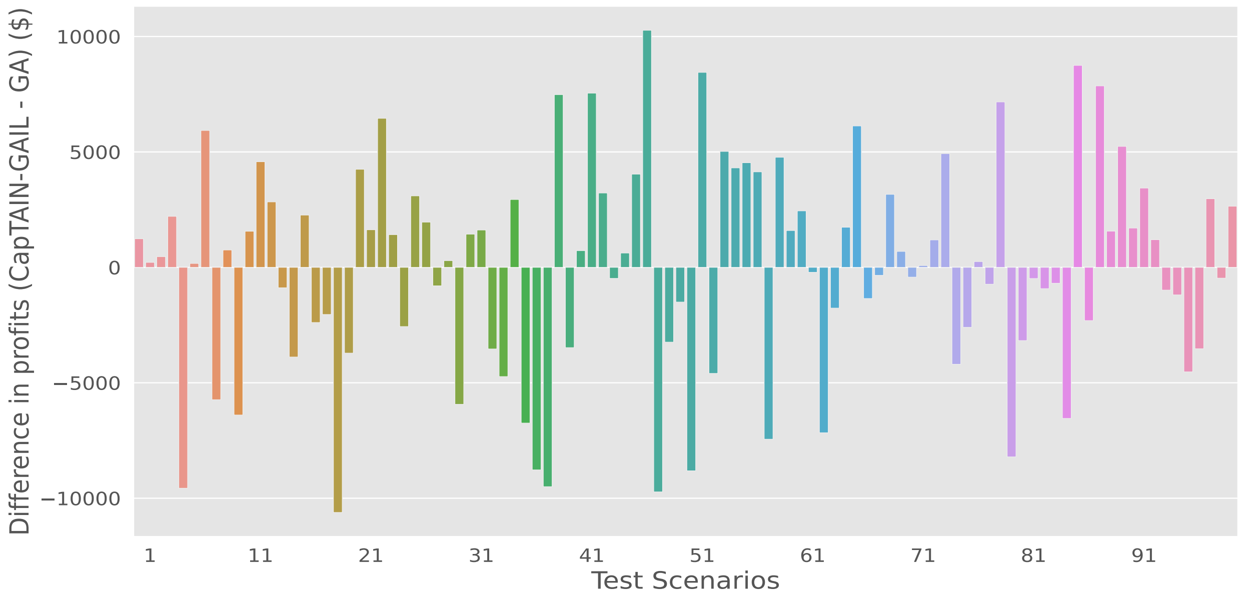

Fig. 6 Difference in the profits earned by CapTAIN-GAIL and GA, Positive values indicate better performance by CapTAIN-GAIL and negative values indicate better performance by GA.

Even though both CapTAIN-GAIL and GA perform equally in terms of the average profits they generate, it shall be noticed in Fig.6 that the difference in the profit values are quite large. This indicates that in cases where GA performs better, it outperforms CapTAIN-GAIL by a large margin while the opposite is also true, i.e. CapTAIN-GAIL outperforms GA by a large margin, in the cases where it performs better. One would ideally assume the difference in the profits to be close to 0, given that both the methods generate similar amount for profits, and this discrepancy in the performance of these two algorithms shall remain the subject of future research.

4. Generalizability Analysis

To test the generalizability of CapTAIN-GAIL, we test it on a new set of 100 unseen scenarios that were not a part of the expert demonstrations on which it was trained, and compare it's performance against CapTAIN. Fig.7 plots the difference in the profits earned by CapTAIN-GAIL and CapTAIN. When compared to Fig.5, we notice that CapTAIN-GAIL shows improvement in terms of performing better in cases where CapTAIN generates more profits (which can be seen by the reduced negative values in Fig.7).

Further analysing the mean profits earned by both the methods in the unseen scenarios (as shown in Fig.8) and comparing it to Fig.4, we notice that CapTAIN-GAIL achieves better mean performance and remarkably better bounding of the unseen worst-case scenarios (that can be seen in the reduce standard deviation between Fig.4 and Fig.8), when compared to CapTAIN. In the unseen scenarios, **CapTAIN-GAIL generates an average profit of \$17813** as compared to \$17587 in the expert demonstration cases while **CapTAIN generates \$15902**. Additionally, the standard deviation of the profits earned, goes down from \$3363 in the expert demonstration cases to \$3161 in the unseen cases. This

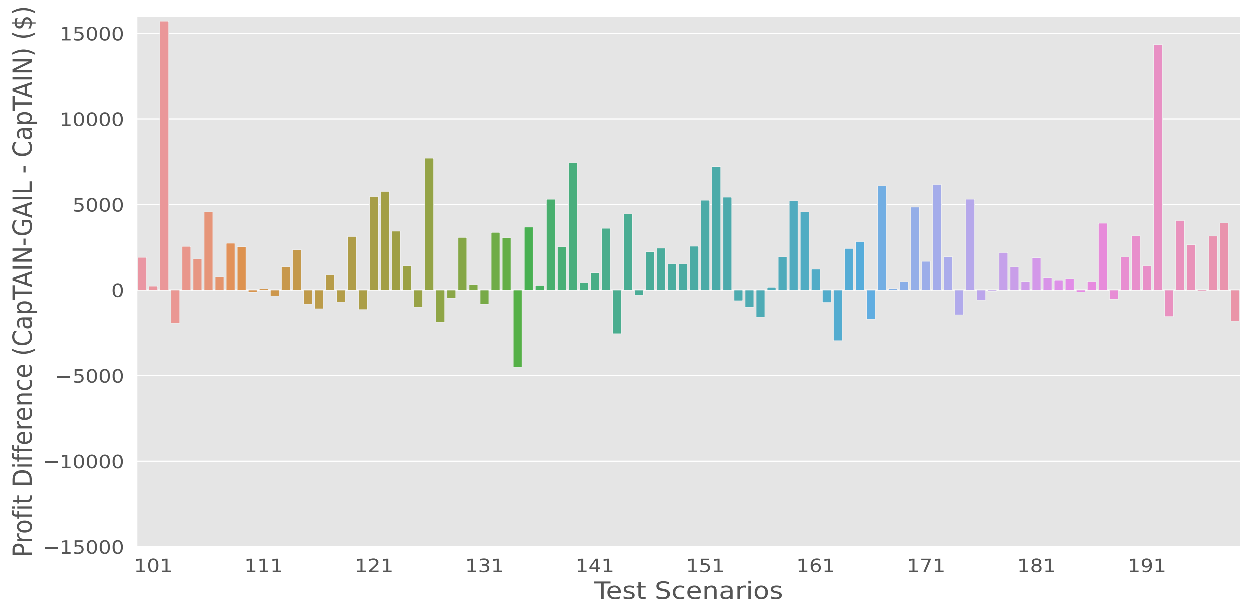

Fig. 7 Difference in the profits earned by CapTAIN-GAIL and CapTAIN in completely unseen scenarios. Positive values indicate better performance by CapTAIN-GAIL and negative values indicate better performance by GA.

shows that CapTAIN-GAIL is generalizable across scenarios that are unseen in the expert demonstrations. Conducting a T-test with the null-hypothesis being that both the methods generate the same average profits, gives a p-value of $3.09^{-5} (< 0.05)$. Which means that CapTAIN-GAIL performs statistically better than CapTAIN.

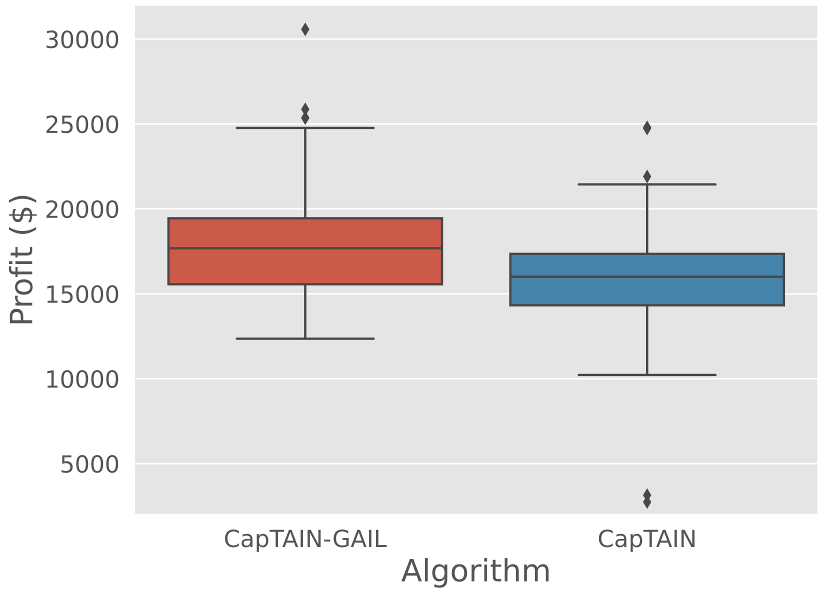

Fig. 8 Comparing the average profits earned by CapTAIN-GAIL and CapTAIN in the unseen test cases.

5. Further Analysis

To further analyse the performance difference among the three methods, we look at their computation times as well as track the average number of idle decisions and flight decisions taken by the three methods.

The computing time for GA is calculated by adding the total time required to generate the solutions for an entire episode. Similarly, the computation times for CapTAIN and CapTAIN-GAIL are calculated based on the total forward-propagation time for the entire episode. It was found that GA took an average of 783.48 ± 267.54 seconds per episode while CapTAIN and CapTAIN-GAIL took 2.57 ± 2.24 seconds and 1.86 ± 2.25 seconds respectively. Thus, there is a significant advantage of using a policy trained using imitation learning when compared to an optimization solver like GA, in terms of computational time required.

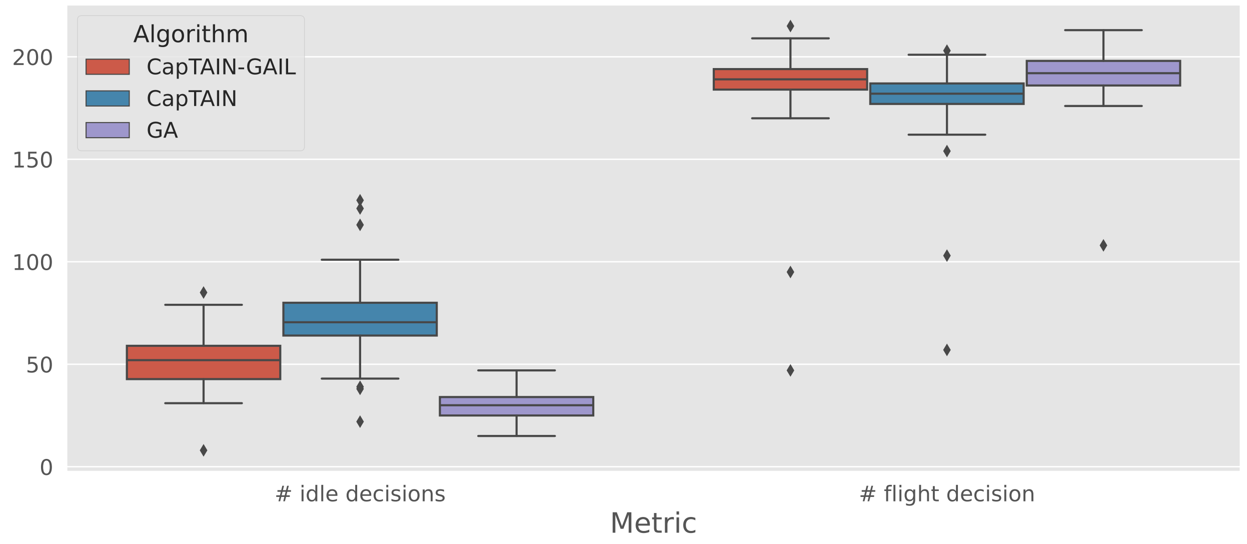

Fig. 9 Comparing the number of idle decisions and number of flight decisions taken by the 3 algorithms.

Additionally, Fig.9 shows the average idle decisions and flight decisions taken by all the three algorithms in the scenarios present in the expert demonstrations. It can be noted that GA take the most number of flight decisions and the least number of idle decisions where as CapTAIN takes the least number of flight decisions and most number of idle decisions. Meanwhile, the decisions taken by CapTAIN-GAIL closely resemble those taken by GA, which is expected given that we have already seen that CapTAIN-GAIL performs similar to the GA.

In comparison, Fig.10 show the decisions taken by CapTAIN-GAIL and CapTAIN in the unseen test cases. It can be noted that the decisions remain fairly the same for both the algorithms. Additionally, the number of outliers in the unseen cases for CapTAIN-GAIL are lower than that for the expert demonstration cases. On an average, CapTAIN-GAIL takes 51.84 idle decisions and 187.29 fight decisions in the seen cases and 52.36 idle decisions and 189.48 flight decisions in the unseen cases, while CapTAIN takes 72.11 idle decisions and 179.96 fight decisions in the seen cases and 71.8 idle decisions and 178.39 flight decisions in the unseen cases. This further demonstrates the generalizability of CapTAIN-GAIL.

VII. Conclusion and Future Work

In this paper, we present CapTAIN-GAIL, a graph neural network based policy that is trained on expert demonstrations from a Genetic Algorithm, using generative-adversarial imitation learning, for the Urban Air Mobility fleet scheduling. We demonstrate that pure RL-based methods are unable to achieve optimal solutions whereas classical and heuristic optimization techniques are computationally too expensive. This motivates the use of imitation learning and we demonstrate that CapTAIN-GAIL performs better than pure RL-based methods. We evaluated the three methods on 100 test scenarios and found that CapTAIN-GAIL generates near-optimal solutions (statistically equivalent to the solutions generated by the Genetic Algorithm) while doing so in a fraction of the time taken by the Genetic Algorithm. Additionally, we tested CapTAIN-GAIL and CapTAIN on 100 unseen test scenarios that were not a part of the expert demonstrations and found that our method showed an improvement in terms of the average profit earned while also

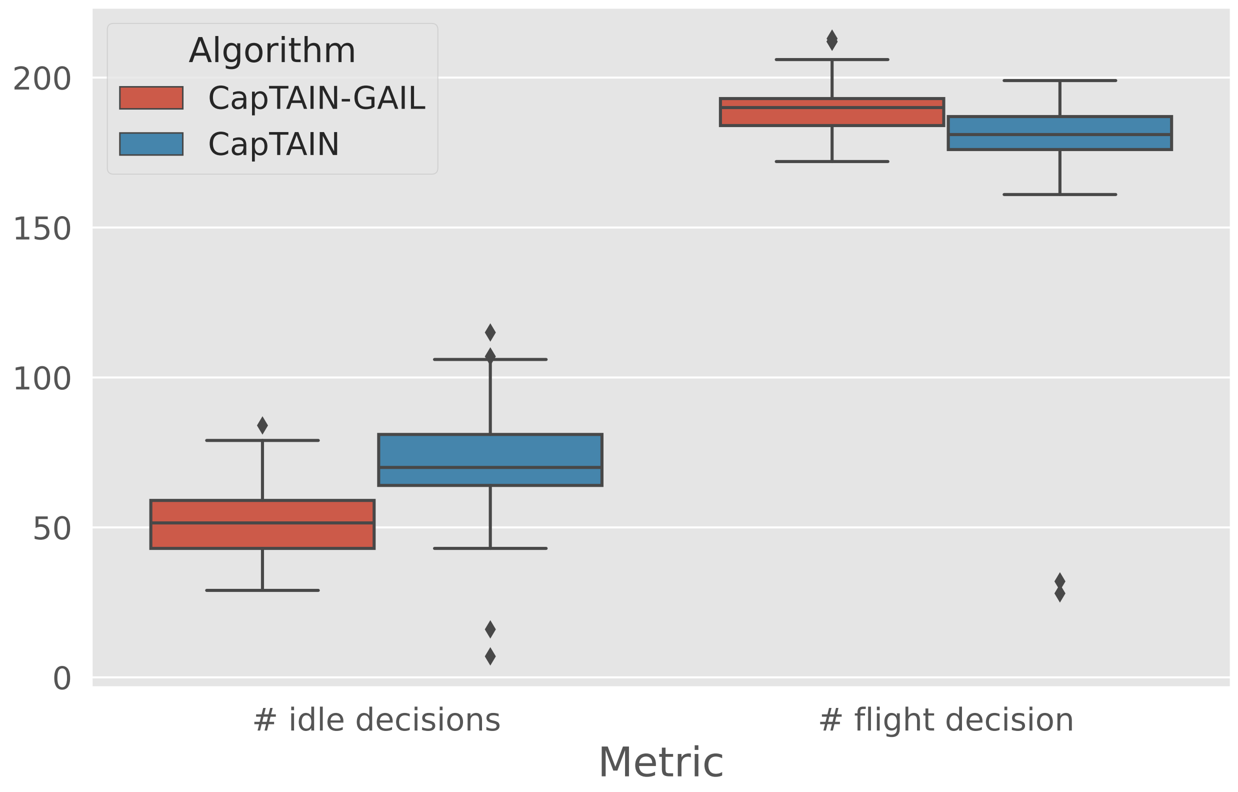

Fig. 10 Comparing the number of idle decisions and number of flight decisions taken by the 2 algorithms in the unseen scenarios.

reducing the standard deviation of the profits earned.

While comparing the performance of our algorithm against GA, we notice that in some cases, GA performs significantly better while in other cases, CapTAIN-GAIL performs significantly better. Although both the methods generate similar average profits, one would expect the difference in their performance to be ideally minimal. Identifying the cause of this result shall be a focus for the future works.

References

[1] "Urban Air Mobility (UAM) Concept of Operations 2.0_0.pdf," , ???? URL `https://www.faa.gov/sites/faa.gov/files/Urban%20Air%20Mobility%20%28UAM%29%20Concept%20of%20Operations%202.0_0.pdf`.

[2] Reed, T., "INRIX Global Traffic Scorecard," , 2019.

[3] Krivonos, P. D., "Communication in Aviation Safety: Lessons Learned and Lessons Required," 2007.

[4] Sun, L., Wei, P., and Xie, W., "Fair and Risk-Averse Urban Air Mobility Resource Allocation Under Uncertainties," 2023. URL `http://dx.doi.org/10.2139/ssrn.4343979http://dx.doi.org/10.2139/ssrn.4343979`.

[5] Hwang, S.-I., and Cheng, S.-T., "Combinatorial Optimization in Real-Time Scheduling: Theory and Algorithms," *Journal of Combinatorial Optimization*, Vol. 5, No. 3, 2001, pp. 345–375. https://doi.org/10.1023/A:1011449311477, URL https://doi.org/10.1023/A:1011449311477.

[6] Pradeep, P., and Wei, P., "Heuristic Approach for Arrival Sequencing and Scheduling for eVTOL Aircraft in On-Demand Urban Air Mobility," *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, 2018, pp. 1–7. https://doi.org/10.1109/DASC.2018.8569225.

[7] Kim, S. H., "Receding Horizon Scheduling of On-Demand Urban Air Mobility With Heterogeneous Fleet," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 56, No. 4, 2020, pp. 2751–2761. https://doi.org/10.1109/TAES.2019.2953417.

[8] Shihab, S. A. M., Wei, P., Shi, J., and Yu, N., *Optimal eVTOL Fleet Dispatch for Urban Air Mobility and Power Grid Services*, ???? https://doi.org/10.2514/6.2020-2906, URL https://arc.aiaa.org/doi/abs/10.2514/6.2020-2906.

[9] Paul, S., Witter, J., and Chowdhury, S., "Graph Learning-based Fleet Scheduling for Urban Air Mobility under Operational Constraints, Varying Demand & Uncertainties," *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, Association for Computing Machinery, New York, NY, USA, 2024, p. 638–645. https://doi.org/10.1145/3605098.3635976, URL https://doi.org/10.1145/3605098.3635976.

[10] *A Scalable Graph Learning Approach to Capacitated Vehicle Routing Problem Using Capsule Networks and Attention Mechanism*, International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. Volume 3B: 48th Design Automation Conference (DAC), 2022. https://doi.org/10.1115/DETC2022-90123, URL https://doi.org/10.1115/DETC2022-90123.

[11] Paul, S., Ghassemi, P., and Chowdhury, S., "Learning Scalable Policies over Graphs for Multi-Robot Task Allocation using Capsule Attention Networks," *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 8815–8822. URL https://api.semanticscholar.org/CorpusID:248562542.

[12] Ho, J., and Ermon, S., "Generative Adversarial Imitation Learning," *Advances in Neural Information Processing Systems*, Vol. 29, edited by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf.

[13] Fernando, M., Senanayake, R., Choi, H. H., and Swany, M., "Graph Attention Multi-Agent Fleet Autonomy for Advanced Air Mobility," 2023. https://doi.org/10.15607/RSS.2023.XIX.105.

[14] Paul, S., and Chowdhury, S., "A Graph-based Reinforcement Learning Framework for Urban Air Mobility Fleet Scheduling," *AIAA AVIATION 2022 Forum*, ???? https://doi.org/10.2514/6.2022-3911, URL https://par.nsf.gov/biblio/10345357.

[15] Kamra, N., and Ayanian, N., "A mixed integer programming model for timed deliveries in multirobot systems," *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, 2015, pp. 612–617. https://doi.org/10.1109/CoASE.2015.7294146.

[16] Miller, C. E., Tucker, A. W., and Zemlin, R. A., "Integer Programming Formulation of Traveling Salesman Problems," *J. ACM*, Vol. 7, 1960, pp. 326–329. URL https://api.semanticscholar.org/CorpusID:2984845.

[17] Mühlenbein, H., "Parallel genetic algorithms, population genetics and combinatorial optimization," *Parallelism, Learning, Evolution*, edited by J. D. Becker, I. Eisele, and F. W. Mündemann, Springer Berlin Heidelberg, Berlin, Heidelberg, 1991, pp. 398–406.

[18] Peng, Y., Choi, B., and Xu, J., "Graph Learning for Combinatorial Optimization: A Survey of State-of-the-Art," *Data Science and Engineering*, Vol. 6, No. 2, 2021, pp. 119–141. https://doi.org/10.1007/s41019-021-00155-3, URL https://doi.org/10.1007/s41019-021-00155-3.

[19] Rizzoli, A. E., Montemanni, R., Lucibello, E., and Gambardella, L. M., "Ant colony optimization for real-world vehicle routing problems," *Swarm Intelligence*, Vol. 1, No. 2, 2007, pp. 135–151. https://doi.org/10.1007/s11721-007-0005-x, URL https://doi.org/10.1007/s11721-007-0005-x.

[20] Wang, X., Choi, T.-M., Liu, H., and Yue, X., "Novel Ant Colony Optimization Methods for Simplifying Solution Construction in Vehicle Routing Problems," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 17, No. 11, 2016, pp. 3132–3141. https://doi.org/10.1109/TITS.2016.2542264.

[21] "Exploratory Combinatorial Optimization with Reinforcement Learning," Vol. 34, 2020, pp. 3243–3250. https://doi.org/10.1609/aaai.v34i04.5723, URL https://ojs.aaai.org/index.php/AAAI/article/view/5723.

[22] Khalil, E. B., Dai, H., Zhang, Y., Dilkina, B. N., and Song, L., "Learning Combinatorial Optimization Algorithms over Graphs," *ArXiv*, Vol. abs/1704.01665, 2017. URL https://api.semanticscholar.org/CorpusID:3486660.

[23] Jacob, R. A., Paul, S., Li, W., Chowdhury, S., Gel, Y. R., and Zhang, J., "Reconfiguring Unbalanced Distribution Networks using Reinforcement Learning over Graphs," *2022 IEEE Texas Power and Energy Conference (TPEC)*, 2022, pp. 1–6. https://doi.org/10.1109/TPEC54980.2022.9750805.

[24] Kaempfer, Y., and Wolf, L., "Learning the Multiple Traveling Salesmen Problem with Permutation Invariant Pooling Networks," 2018.

[25] Kool, W., van Hoof, H., and Welling, M., "Attention, Learn to Solve Routing Problems!" *International Conference on Learning Representations*, 2018. URL https://api.semanticscholar.org/CorpusID:59608816.

[26] Li, Z., Chen, Q., and Koltun, V., "Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search," *Advances in Neural Information Processing Systems*, Vol. 31, edited by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/8d3bba7425e7c98c50f52ca1b52d3735-Paper.pdf.

[27] Nowak, A., Villar, S., Bandeira, A. S., and Bruna, J., "Revised Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks," , 2018.

[28] Verma, S., and Zhang, Z.-L., "Graph Capsule Convolutional Neural Networks," , 2018.

[29] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I., "Attention is All you Need," *Advances in Neural Information Processing Systems*, Vol. 30, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[30] "Electricity Rates (Updated June 2024) — Electric Choice," https://www.electricchoice.com/electricity-prices-by-state/, ????

[31] "CityAirbus NextGen - Urban Air Mobility - Airbus," https://www.airbus.com/en/innovation/low-carbon-aviation/urban-air-mobility/cityairbus-nextgen, ????

[32] Gleave, A., Taufeeque, M., Rocamonde, J., Jenner, E., Wang, S. H., Toyer, S., Ernestus, M., Belrose, N., Emmons, S., and Russell, S., "imitation: Clean Imitation Learning Implementations," arXiv:2211.11972v1 [cs.LG], 2022. URL https://arxiv.org/abs/2211.11972.

[33] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," *ArXiv*, Vol. abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

[34] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, Vol. 22, No. 268, 2021, pp. 1–8. URL http://jmlr.org/papers/v22/20-1364.html.