Leveraging Multi-Agent Reinforcement Learning to Enhance E-AMoD Fleet Control

Ayush Agrawal
ayushag@cs.stanford.edu

Aaryan Singhal
aaryan04@cs.stanford.edu

Justin Wu
justwu@cs.stanford.edu

1 Extended Abstract

As the need for mass urban transportation grows, Electric Autonomous Mobility on Demand (E-AMoD) systems are poised to play a critical role in meeting passenger demand across cities. Current frameworks to design E-AMoD systems, however, focus purely on maximizing profits and only use linear or single-agent methods to solve a multi-step non-linear problem.

In this project, building off of Gammelli et al.'s [1] tri-level framework, we developed a novel re-formulation for E-AMoD fleet control that can maximize profits while simultaneously considering charging of electric vehicles as a key consideration in its decision-making, which [1] fails to do. In addition to the first incorporation of charge as a variable, we also implemented the first approach to Sequential Multi-Agent Reinforcement Learning (MARL) in the context of fleet control, with four strategies formulated and tested. While previous approaches have only used singular RL agents, we devised a novel multi-agent coordination mechanism ("one-time freeze") to train sequential heterogeneous RL agents with changing global state spaces and non-goal conditioned joint policies, which involved optimized gradient-freezing cycles between the two RL agents.

Our methods make use of a graph-based state space that encodes multi-vehicle moving capabilities in a single timestep, and our algorithms optimally account for both passenger-matching in the current timestep and vehicle rebalancing/recharging for future timestep optimization using two GNN-parametrized Actor-Critic RL agents. Each agent is built off of a similar underlying structure and rewarded output, thus forming a broader framework that can be used to incorporate other agents with similar state spaces and functionality if necessary.

Overall, although resource-constrained to toy problems that a combination of linear optimization and singular RL agent can near-perfectly solve, our novel "one-time freeze" method is able to provide over 38% improvements compared to standard MARL strategies, and still able to achieve 93% the performance of a near-perfect solution. While these methods exhibit more stochasticity over our limited episodic training runs, the best policies are still able to perform similarly on new testing environments in the toy problem. Furthermore, we are able to informally demonstrate that our approach is likely to be more generalizable to more complex scenarios such as real city-wide transportation data, and are able to characterize flaws in the original paper's methods, particularly the solely profit-driven reward mechanisms and inflexible graph-based architecture.

Overall, we develop a promising baseline of multi-agent reinforcement learning approaches for the highly relevant E-AMoD fleet control application, one that can potentially be combined with linear optimization and off-policy methods to be implemented in real city settings. Future work aims to explore promising new transformer-based sequential architecture, more robust reward formulations, and increased stochasticity of E-AMoD system dynamics. A link to our GitHub repository is here: https://github.com/Aaryan0404/cs224rfinalProject.

2 Introduction and Motivation

Electric-Autonomous-Mobility-on-Demand (E-AMoD) systems are set to play a critical role in addressing the challenges associated with the rapid growth of urban transportation demand. Controlling a fleet of autonomous electric vehicles traditionally consists of addressing three priorities: matching vehicles according to current passenger demand, rebalancing idle vehicles to areas with high expected passenger demand, and recharging vehicles as necessary. In the real-world, such a system is likely to be controlled in real-time by a centralized operator, who hopes to maximize generated profits.

To understand the importance of using reinforcement-learning (RL), it is necessary to evaluate the various approaches that have been tested in previous works. While traditional linear optimization is able to always find the optimal solution to the fleet control problem, it is far too computationally expensive to be suitable for applications of the scales that real-world deployment requires [2]. Furthermore, several solutions dependent on linear optimization assume optimization is being done in a static environment and are therefore not suitable for real-time fleet control. With this understanding, several research teams have focused their efforts on utilizing RL pipelines as a tool to develop efficient fleet control algorithms, including cascaded Q-learning, Deep Q-Networks, and Proximal Policy Optimization ([3], [4], [5], [6]). Yet despite improving upon linear optimization, these approaches lack the abilities to take into account temporal demand and dynamic conditions, coordinate vehicles cleverly within the fleet, and handle computational inefficiency caused by per-vehicle rebalancing algorithms, restricting their ability to scale well with increasing fleet size [1].

Attempting to address these flaws, a tri-level framework was recently used to break the fleet control problem down into a series of steps that could be represented as actions being taken in a graph [1]. Given the success of graph-based RL with this particular problem, our team decided to extend its application to E-AMoD systems, where vehicles now not only have variable charge levels, but also have to be actively charged at certain time steps to cater to long-term passenger demand. To this end, we reformulated the tri-level framework: (1) assign vehicles with varying charge levels to passengers, (2) rebalance vehicles spatially and recharge vehicles as necessary, and (3) achieve the optimal desired idle vehicle distribution.

Extending graph-RL work for AMoD systems to E-AMoD systems is non-trivial, for the effects of charge are no longer a feasible sub-problem to solve with linear optimization in step 1 due to the lack of associated heuristics. Additionally, the second step of the tri-level framework now has to take into consideration vehicle charge levels, instead of only spatial positions.

To address this multi-factor problem of E-AMoD coordination, we sought to do the following:

1. Alter the tri-level graph-based framework developed for AMoD systems so that they are suitable for E-AMoD systems.
2. Leverage sequential multi-agent reinforcement learning (MARL) to learn optimal policies for both passenger-vehicle assignment (step 1) and vehicle rebalancing/recharging (step 2).

3 Related Work

To motivate our approach to develop a Multi-Agent Reinforcement Learning (MARL) solution to the E-AMoD fleet control problem, we first investigated classical MARL approaches. Most progress in MARL has been toward building solutions for independent, homogenous RL-agents that need to cooperate and/or compete in a simulated environment. A common approach for encouraging interactions amongst independent, homogenous RL-agents is enabling communication, specifically to allow the exchange of information about state, action, and reward [7], with each RL agent learning an individual policy in a decentralized manner [8].

Our scenario's unique tri-level framework, encompassing two interdependent Reinforcement Learning (RL) agents, also possesses a sequential problem, making immediate good/bad action differentiation impossible until the reward from the third step is received. Given the sequence and heterogeneity, the two RL agents cannot be trained independently, with the state space and actions of the second agent being contingent on the first agent's actions. This leads to a "moving targets" challenge in MARL, where each agent's optimal policy hinges on the other's learning process.

Given that most literature in MARL is constrained to environments where agents are assumed to independent and homogeneous, recent work has focused on centralized training/learning and

decentralized execution (CTDE), which enables interaction between heterogeneous agents with distinctive reward functions in the shared state space [9]. CTDE-based approaches also help address the challenge of partial observability [10]. However, despite the encouraging results achieved, CTDE makes an independence assumption on agent policies, which limits agents from adopting global cooperative information from each other during centralized training as is critical for the sequential nature of our problem. Additionally, the centralized training part of CTDE usually involves using off-policy algorithms that can learn from the experiences of all agents, regardless of the current policy. Applying CTDE with on-policy algorithms such as our Actor-Critic can be challenging due to the issue of non-stationarity, where the environment changes as other agents learn and change their policies and the global state at every iteration.

Given the non-stationary nature of our environment, another alternative involves MARL policy-coordination and hierarchical MARL. With regards to the former, researchers have found baseline success in improving coordination between heterogeneous, interdependent RL agents by enabling each agent to predict the actions being taken by the other [11]. Furthermore, hierarchical MARL helps address the challenge of non-stationarity and delayed rewards via incorporation of traditional experience replay and temporal abstraction [12]. However, while hierarchical MARL could potentially provide some benefits for our scenario, like allowing agents to specialize in their respective sub-tasks, designing and implementing the "managing" agent and determining how it should set sub-goals for the other agents (without the explicit structure provided by the tri-level framework) is unnecessarily complex, especially with the variations in the state space, passenger demand, and resource costs. Furthermore, there is the issue of credit assignment: it could be challenging to determine how to distribute rewards or penalties between the manager and worker agents, and which agent to iterate upon if there is a plateau in performance.

Fortunately, the structure of our tri-level framework uniquely enables us to use a linear optimizer (adapted from classical model predictive control algorithms) to train our two heterogenous RL agents via alternative freezing. In order to address both non-stationarity and delayed rewards, we propose training two RL agents in a sequential manner, hypothesizing that each agent will be incentivized to learn the best it can, given the other agent's best learned policy so far. To the best of our knowledge, such an approach (further detailed below) has not been tested in our domain space previously.

4 Technical Approach

Environment We introduce a strategic graph-based model for efficient coordination of Electric Autonomous Mobility-on-Demand (E-AMoD) systems, which learns from historical data to optimize for both passenger-vehicle matching and rebalancing/recharging of idle vehicles. Our model considers a transportation network depicted as a fully connected graph $G = (V, E)$, with vertices V denoting stations (including pick-up and drop-off points) at six levels of charge (chosen arbitrarily) and edges E signifying the shortest routes between these stations (with certain rules placed on moving between spatial-charge nodes - for example, there is no edge that allows a vehicle to gain charging while moving from one spatial node to another). Within this network, there are M single-occupancy autonomous vehicles and $N_v = \frac{|V|}{6}$ stations.

In our model, time is discretized into specific intervals $\mathcal{T} = \{1, 2, ..., T\}$, each having a fixed duration ΔT. The travel duration between any two stations is calculated in these units of time intervals, with $\tau_{ij} \in \mathbf{Z}^+$ symbolizing the number of time steps necessary to travel from station i to station j.

At each time step, passengers arrive at their respective stations, awaiting transportation to their desired destinations. Each trip is characterized by the demand and the price that the passengers are willing to pay. Given a passenger departing from station i at time t, they are expected to arrive at their destination station j at the time $t + \tau_{ij}$.

The primary responsibility of the AMoD operator is to effectively match passengers with available autonomous vehicles, facilitating their journeys to the respective destinations and maximizing profit generated. The quantity of passengers that successfully depart from station i and arrive at station j at time t, having been matched with a vehicle, is referred to as the *passenger flow*. It is important to note that, unlike in the classical AMoD problem, working with E-AMoD systems now means that the available autonomous vehicles now have varying charge levels.

Passengers who fail to find a match with any vehicle will exit the system. Conversely, vehicles that aren't matched with passengers can choose to remain at their current location or be relocated to other stations with potentially higher demand. This process of vehicle repositioning is called 'rebalancing'. The number of vehicles re-positioned from station i to station j at a particular time t is described as the *rebalancing flow*.

Through this approach, our aim is to develop an effective control system that adapts to demand patterns, thereby reducing passenger wait times and maximizing the utilization of autonomous vehicles. Following this setup, we adopt a tri-level framework, similar to the three-step framework proposed in [1]. In particular, we'll (1) use an RL agent to derive passenger flow by solving a matching problem (essentially dispatching passengers to given destinations, while maximizing profits and accounting for variably charged vehicles), (2) compute the desired distribution of idle/unused vehicles according to forecasted costs (including costs associated to charge), and (3) convert the desired distribution to rebalancing flow by solving a minimal rebalancing-cost problem. We refer to Figure 1 for a visual. We'll formalize this by describing the Markov Decision Process in the following paragraphs.

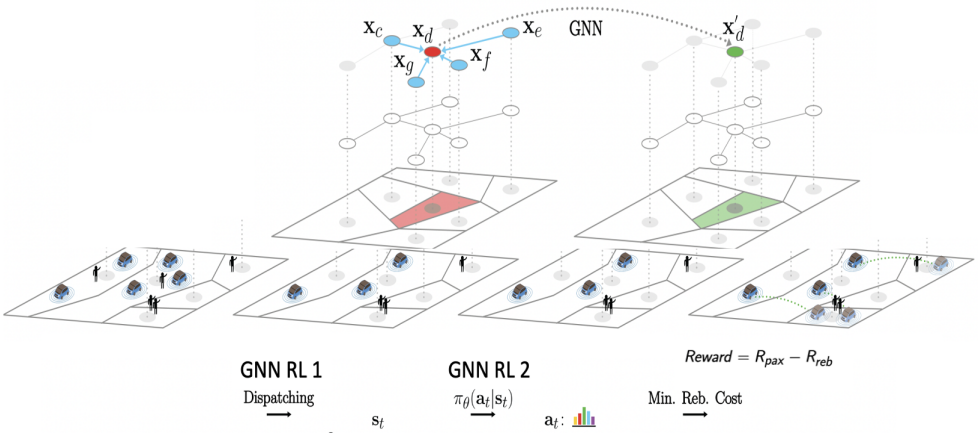

Figure 1: For each time-step, a tri-level framework is utilized to (1) assign passengers to vehicles, (2) rebalance idle vehicles across the space-charge graph, and (3) optimally achieve the desired rebalancing of vehicles

Framework Let's formalize the rebalancing issue (handled by the secondary RL agent in our problem set-up) of the E-AMoD system as a Markov Decision Process (MDP). We can express this MDP as $\mathcal{M}_{reb} = (\mathcal{S}_{reb}, \mathcal{A}_{reb}, P_{reb}, r_{reb}, \gamma)$. This formulation includes the state space $\mathcal{S}_{reb}$, action space $\mathcal{A}_{reb}$, state transition probability function P_{reb}, reward function r_{reb}, and a discount factor γ.

The *action space* is defined by considering a certain number of available vehicles $M_a \leq M$ and the current distribution of these vehicles among the N_v stations. Our goal is to decide the most effective distribution of idle vehicles a^t_{reb} across the stations at any given time step t. Each action represents the decision of how many idle vehicles should be relocated from one station to another.

The *behavior policy* we propose is aimed at generating a probability distribution across the stations. This distribution provides a percentage indication of idle vehicles that should be rebalanced to each station. Essentially, the behavior policy guides the distribution of idle vehicles based on the current state of the AMoD system, striving for an efficient balance that ensures availability of vehicles where they are most needed. It is important to note here that our RL agent is to return a distribution of actions given the state it observes.

We define the reward function for the MDP in terms of the AMoD system operator's perspective. The goal here is to devise behavior policies that optimize the satisfaction of travel demand and the operator's profits, while minimizing unnecessary vehicle trips. The net reward in this scenario is the balance between the gains from distributing vehicles to meet demand and the incurred costs from rebalancing operations.

In this calculation, C_{oper} denotes the operational cost per timestep, T_{ij} represents the time required to travel from node i to node j, P_{avg} stands for the average charging price, and Q_{ij} describes the charge difference when moving vehicles from node i to node j. T_{norm} serves as a time normalizer.

The rebalancing action for the edge from i to j is given by R_{ij}, and the passenger action for the same edge is symbolized as S_{ij}. Finally, P_{ij} represents the trip price from region i to region j at time t.

The total reward combines the rewards from dispatching vehicles and the cost of rebalancing:

$$\textbf{Reward} = \mathbf{R}_{\text{pax}} - \mathbf{R}_{\text{rebalancing}}$$

where

$$\mathbf{R}_{\text{pax}} = \sum_{i,j} S_{ij}(P_{ij}(T_{ij} + T_{norm})C_{oper})$$

represents the reward from dispatching vehicles and

$$\mathbf{R}_{\text{rebalancing}} = \sum_{i,j} R_{ij}((T_{ij} + T_{norm})C_{oper} + P_{avg}Q_{ij})$$

denotes the cost of rebalancing vehicles.

For the *state Space*, we consider the information aggregated at each station. For each station, we track the current availability of idle vehicles at each station and the projected availability of idle vehicles which is estimated based on previously assigned passenger and rebalancing trips. Our rebalancing strategy must also consider both the current and forecasted transportation demand between all stations. Thus we represent the travel behavior as the rate of a time-dependent Poisson process. The dynamics in the MDP encapsulate the stochastic evolution of travel demand patterns and the effect of rebalancing decisions on future states of the system. Travel demand between stations evolves stochastically, following a time-dependent Poisson process (independent of rebalancing actions). Note that the availability and distribution of idle vehicles are directly influenced by the rebalancing decisions.

The AMoD rebalancing problem, having been formally modeled as an MDP, can be solved using an Advantage Actor-Critic (A2C) approach. The A2C algorithm uses a specific neural network architecture to define both the policy $\pi_\theta(a_t|s_t)$ and the value function estimator $V_\phi(s_t)$. During the rebalancing stage, we update the node features with graph convolutions and non-linearities. This helps compute parameters α for a Dirichlet policy, which serves as a probability distribution over stations, indicating what percentage of idle vehicles should be rebalanced in each station. Furthermore, it assists in estimating the value function $V(s_t)$.

The policy network is devised such that its output represents the concentration parameters α of a Dirichlet distribution, ensuring that $\pi_\theta(a_t|s_t)$ forms a valid probability density over actions. The neural network utilized in our implementation comprises a layer of graph convolution (equipped with skip-connections and ReLU activations). The output of this layer is aggregated across neighboring nodes using a sum-pooling function and is then passed through three MLP layers with 32 hidden units to generate the Dirichlet concentration parameters. The value function $V_\phi(s_t)$ has a similar architecture to that of the policy with an additional global sum-pooling step carried out on the graph convolution output.

In our approach, we also consider a separate RL agent for the dispatching step. Our RL-1 is a modified version of the RL-2.

Unlike RL-2, which has partial observability of the state, RL-1 observes the entire state of the AMoD environment. Specifically, RL-1, through the graph network representation of the AMoD environment it receives, is able to observe the per-node accumulation of vehicles in the system and passenger demand at each node. Both of these pieces of information are observed by RL-1 in addition to what is observed by RL-2. It is worth noting that this is because there is no concept of idle vehicles in the environment in which the RL-1 is acting and current per-node passenger demand is information only relevant and available in the state in the which RL-1 is acting.

Given the state it observes, RL-1 implicitly returns a passenger flow action by returning the number of desired vehicles at each node. While the exact output of the RL-1 and RL-2 are identical in that they are a distribution of desired vehicles across the space-charge graph, these outputs are converted into actions in distinct ways.

For RL-1, an explanation of how the passenger flows are calculated from a distribution of vehicles across the space-charge graph now follows. A single passenger flow action consists of a tuple equal to the number of edges in the graph, where the ith element in the tuple represents the number of vehicles flowing through the ith edge of the environment. Given that RL-1 returns a desired distribution of vehicles across the space-charge graph, passenger flows are calculated by looking at the pairwise difference of desired vehicles between relevant pairs of nodes in the space-charge graph. Specifically, the passenger flow for the ith edge, connecting origin node o and destination node d in the AMoD environment, is equal to the difference between the desired number of vehicles at d and the desired number of vehicles at o.

While the state space and realization of actions differ for RL-1 and RL-2, the formulation of the reward (calculated at the end of the tri-level framework) is the same.

Training vs. Evaluation: In order to properly contextualize the results of our work, it is important to note that both RL1 and RL2 return parameters for distributions of their respective actions (which will help account for training stochastically, as described in the next section). During training, the actual passenger-to-vehicle assignment done in the first step of the tri-level framework is based on a single action randomly sampled from the Dirichlet distribution of passenger-to-vehicle assignment actions outputted by RL1. Similarly, the actual desired distribution of idle vehicles across the space-charge graph in the second step of the tri-level framework is based on a single desired distribution randomly sampled from the Dirichlet distribution of desired idle vehicle distributions by RL2. During training, we keep track of our best policy so far. To fairly evaluate the best policy found once training is complete, we then proceed to roll-out the policy in question in the same environment, but select the actions by taking the average of the two Dirichlet distributions outputted by the RL agents (as compared to sampling randomly from the two distributions during training).

5 Experiments and Results

For our experiments, we focus on using toy problems that we constructed. Though we had access to larger datasets like NYC and SF networks, we could not train on these for the scope of this project due to compute resources and time constraints.

Toy problem 1 is a simple set-up consisting of 2 spatial nodes, each with 6 charge levels. The scenario contains 20 time steps, the first 10 of which have passenger demand (manually constructed in a list d_1) to travel from spatial node A to spatial node B, and the last 10 of which have passenger demand flowing in the opposite direction. As a vehicle moves from one spatial node to the other, it drops one charge level. In a single time-step, if a vehicle is recharging (the cost of which is also manually constructed in a list c_1), it stays at the same spatial nodes and can move up a maximum of 4 charge levels. For this first toy problem, we ran MPC, baseline, and standard MARL, but quickly realized that the dispatching problem (step 1) was too easily solvable by a linear optimizer, even with the constraint of charge, which would make our experiments with RL techniques meaningless.

As such, we built off this toy problem 1 to construct a more complex one ("toy problem 2") that the linear optimizer would have more difficulty solving. In particular, we:

1. Vary the demand by taking $d_2 = d_1 \log^2(t)$,
2. Vary the charge by taking $c_2 = c_1 \sin(2\pi t/n)$,
3. Increase the number of spatial nodes (i.e. possible passenger locations) from two to three. Note that while we would like the latter to be much larger in order to allow more non-linearity, this scales exponentially with compute resources needed,
4. Increase the previous ratio of the number of vehicles relative to passenger demand by two orders of magnitude. We hypothesize that this will force the model to make more strategic long-term decisions in managing costs and passenger demand.

For the rest of this section, we'll be focusing on this toy problem 2.

Baselines To formulate a theoretical best-case solution to our toy problem, we built a model predictive control (MPC) algorithm that determines the optimal passenger-to-vehicle matching and vehicle rebalancing across the spatial-charge graph for each time-step, such that the total profit generated across the 20 time steps is maximized. The objective of the MPC optimization problem is the revenue generated by serving passengers minus the cost of spatial rebalancing, vehicle charging, and vehicle operations. Running our MPC algorithm on Toy Problem 2, we found that our theoretical maximum reward is 77718, with optimal served demand and rebalancing costs being 9194 and 9632, respectively.

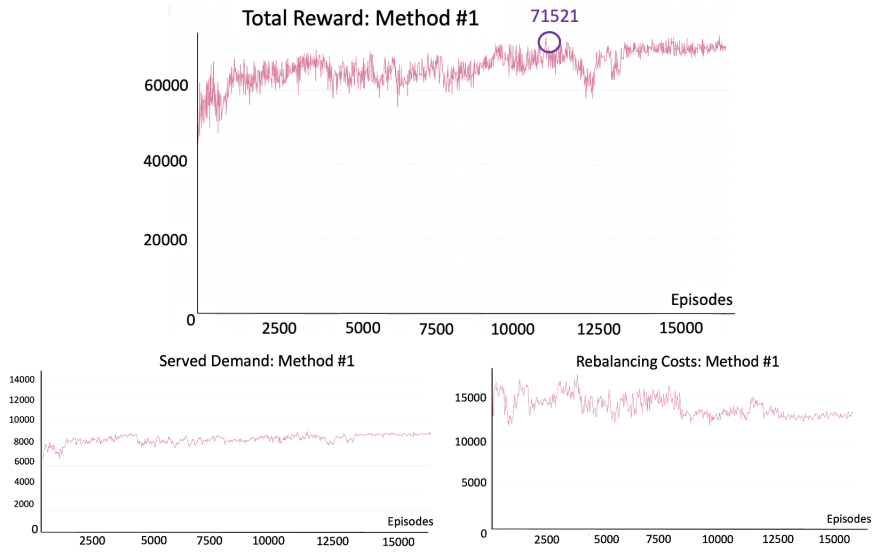

Figure 2: Reward, Served Demand, and Rebalancing Cost results of the baseline single-RL method.

As our standard baseline for RL performance, we re-implemented Gammelli et al.'s [1] RL-pipeline that consists of a single agent to perform vehicle rebalancing across the spatial-charge graph by outputting a desired distribution of idle vehicles, which is then realized to the best of a linear optimizer's ability. It is important to note that passenger-to-vehicle matching is done purely from a profit-seeking perspective with an implicit bias against unrealistic charge levels, given that there is no obvious heuristic or optimal solution to assign vehicles of varying charge levels and spatial locations to passengers wanting to travel from one spatial location to the other. We used an MPNN (Message-Passing Graph Neural Network) architecture and were able to achieve a maximum reward of 71521, which is 92% of the theoretical maximum. Unfortunately, as aforementioned, this means that our baseline is able to near-perfectly achieve the theoretical maximum for our harder toy problem (though less so compared to the 99.5% it achieved under toy problem 1), which means toy problem 2 was still not hard enough.

Standard MARL Our second experiment was to implement a standard Multi-Agent RL (MARL) strategy, in which we use an RL agent (RL1) to replace the linear optimizer in the dispatching step. Since the RL agent for rebalancing remains unchanged, this defines a sequential learning problem, in which RL2 depends on the output of RL1 in order to improve its policy and produce a reward that the linear optimizer in the third step realizes. This approach was only able to achieve maximal reward of $46,434$: about 65% that of our baseline single-RL-agent method above (see Figure 3 and Table 1). Additionally, our served demand and rebalance costs over episodes were much different than the values from MPC and single-RL-agent baselines, which was concerning as it indicated that the RL agents were likely jointly learning a sub-optimal policy.

N-alternate freezing After noticing that Standard MARL performs relatively poorly, we hypothesized that the *moving targets* problem might be the primary cause. As each agent learns and changes its own policy, the environment from the perspective of other agents (or the global state) is also changing. To address this, we devised the strategy of alternate freezing of our multiple RL agents. The concept of freezing the weights, or "parameters", in a neural network comes from the desire

to maintain some layers or parts of the model while allowing others to learn and adapt [13]. In reinforcement learning, the idea of freezing weights has been previously implemented, but typically within a single agent framework to stabilize learning [4].

In our project, we applied this concept across multiple agents. The idea was to have each agent learn its policy in a relatively static environment, allowing it to better estimate the value function and update its policy. In particular for our approach, we first froze (i.e. stopped the gradient updates of) RL1 and let RL2 train with the linear optimizer for $2,000$ episodes (since we must provide some sort of step 1 input for RL2 to function). Then, we froze RL2 and let RL1 train, also for 2000 episodes. We iteratively repeated this process every $n = 2000$ episodes until the end of our 16000 episodes (i.e. a total of 8 times). This sequential/alternate freezing reduces the effect of the moving targets problem by ensuring the environment, from each agent's perspective, stays relatively consistent during its learning phase.

To the best of our knowledge, this alternate freezing strategy has not been attempted for sequential MARL problems in previous literature. Unfortunately, the results we observed were unpromising: N-alternate freezing performed slightly worse than even Standard MARL, achieving maximal reward of $42,139$ (see Figure 3).

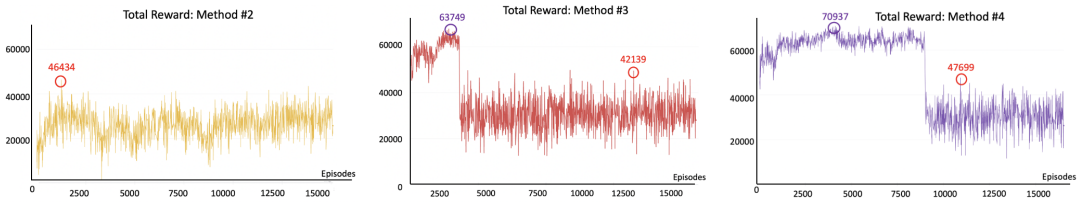

Figure 3: Reward graphs for methods 2, 3, and 4, with MARL reward represented by the red circle. Each plot represents a sub-optimal iteration of MARL techniques that eventually built up to our method 5.

Warm-start MARL Seeing that the alternate freezing did not do nearly as well as we had predicted either, we hypothesized that the RL agents lacked a sufficiently solid baseline to improve upon, thus getting stuck in local minima. Thus, for our fourth experiment, we decided to provide a warm-start to RL2 in order to create a baseline of good "expected" results for our toy problem. We perform this warm-start by training RL2 with the linear optimizer for the dispatching step (step 1) for $7,000$ epsiodes, before then unfreezing RL1. From our previous trials, we estimated that $7,000$ episodes was sufficient time for our baseline to arrive at a reasonably consistent answer to our toy problem 2. With an optimized RL2, we then proceed with the standard MARL approach. Unfortunately, this method 4 too performs only slightly better than methods 2 or 3.

One-time freeze For our final experiment, we let the RL2 converge on a linear optimizer for $7,000$ episodes as in method 4. But rather than switch to standard MARL afterwards, we freeze RL2 and let RL1 train on the remaining $9,000$ episodes. From methods 2-4, we hypothesized that standard MARL and even alternate freezing with large intervals such as $n = 2,000$ were likely not providing enough time for either of our RL agents to sufficiently learn the complexity of the environment and obtain a reasonably optimal policy. Given our compute constraints, since we couldn't perform freeze cycles of anything greater than $n = 10,000$, we chose instead to try one-time freezing: allowing RL2 to nearly converge, and then allowing RL1 to try to determine a best policy on a frozen RL2.

Since we are only freezing each agent once, Method 5 is able to find a middle ground between the moving targets problem that plagues our other methods and being forced to rely on linear optimizers for problems of increasingly non-linear complexity. As was our expectation, this method does indeed perform the best out of all of our MARL strategies, achieving a best-policy (which we then save for later use, as shown in Table 1) reward of $66,213$: which is 38% better than any of the other MARL strategies (methods 2-4) and does 93% as well as our (near-perfect) baseline single-RL method (Method 1).

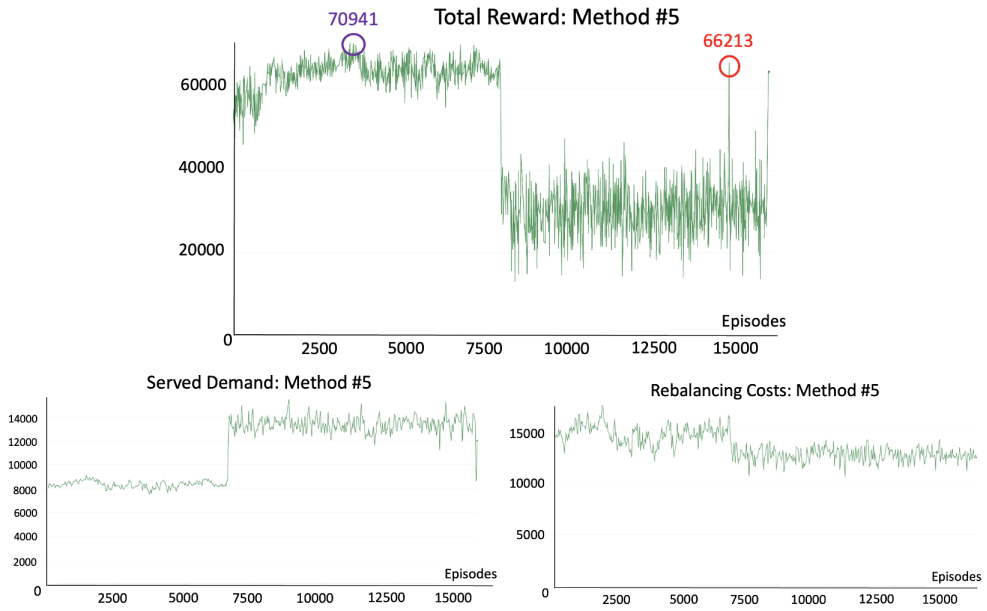

Figure 4: Reward, Served Demand, and Rebalancing Cost results of the best-performing MARL method: one-time freeze.

Method	Best Reward (Train)	Reward (Eval)	Served Demand	Rebalance Cost
Baseline	71521	68865	8965	12597
Standard MARL	46434	45459	11594	10731
N-alternate freezing	42139	41422	13221	8136
Warm-start MARL	47699	46788	12585	10638
One-time freeze	66213	65797	11205	12134

Table 1: Breakdown of best reward, highest served demand, and optimal rebalancing cost for different tri-level configurations.

6 Discussion

Table 1 summarizes the results obtained from all five tested methods in the project. Particularly, we include previously not mentioned evaluation results here that test out a method's best policy (obtained from non-warm-start parts, if necessary) on a new subset of the toy problem. We note that the baseline serves much lower overall demand than all the MARL strategies, and how this is inversely correlated with rebalance costs, the implications of both of which we will discuss below. One interesting thing to note is the better apparent generalization capabilities of MARL strategies, particularly Method 5, as compared to the baseline. While this could just be due to random chance and/or error, one-time freeze is able to achieve five times better performance on the evaluation setting than the baseline, which might be an indication of overfitting in the latter. This remains a key topic to be explored in future studies.

In our Multi-Agent Reinforcement Learning (MARL) experiments, the convergence of our four methods did not meet the levels attained by the baseline single-RL-agent, yielding oscillatory reward values throughout the 16,000 episodes. Limited by computational resources and time constraints, we were unable to extend our experiments to observe more explicit signs of convergence. Notably, in the one-time freeze (Method 5), the model showed a propensity for generating an optimized policy every 2,000 to 3,000 episodes, including one policy which led to our maximum reward of $66,213$ that was very close to baseline. Our observations suggest that extended training duration may foster improved policies and potential weak convergence of the agents towards a unified joint policy. Future research will aim to explore this proposition.

We would like to briefly note that the results in this section all utilize an optimal set of hyperparameters, determined after hyperparameter tuning using a combination of results from Methods 1-5. Of note, we experimented with three different Graph Neural Network architectures for each of the two graph-based MDP state spaces (GCN, MPNN, and GAT); five different periods of n-rotational freezing, as in Method 3 ($n = 200, 500, 1000, 2000, 4000$); and three different learning rates ($10^{-2}, 10^{-3}, 10^{-4}$) of the neural networks for each of the parametrized actors and critics for the two RL agents. Overall, we found best performance with a GCN + MPNN structure for RL1 and RL2 respectively, with learning rates across agents and actors/critics set to 10^{-3}. Finally, as aforementioned, freeze cycles of $n = 2000$ gave the best performance, though the improvements over $n = 1000$ and $n = 4000$ were marginal.

A deeper analysis of our served demand and rebalancing cost graphs in Figures 2 and 4 prompted us to question the optimality of the most numerically robust policies (i.e., the MPC and single-RL-agent baselines) from a real-world application standpoint. The linear optimizer, as per these figures, serves approximately 40% fewer passengers and incurs a $15 - 20\%$ higher rebalancing cost per episode relative to the MARL policies. This paradox arises from our reward function, which primarily values maximizing profit and neglects aspects like customer satisfaction or maximizing passenger service. The profit-oriented policies consequently opt to minimize operational costs, which are generally more substantial in the toy problem than the idle vehicle rebalancing/recharge costs or the revenue from serving a passenger. Consequently, these baseline policies assign fewer cars, leading to the need for increased vehicle rebalancing, which manifests as higher rebalancing costs. Conversely, the "sub-optimal" MARL strategies prioritize serving more passengers, incurring higher operational costs, and consequently a lower reward. This trade-off is most directly observable in Figure 4, where the peak of the best-performing policy is accompanied by a decline in served demand and a rise in rebalancing costs.

In a real-world context, this approach might not be most effective. For instance, if Uber adopted the baseline's policy and Lyft embraced the MARL strategy in a competitive market like New York City, Lyft could quickly establish dominance despite incurring initially lower profits. By serving a larger passenger base, Lyft could ensure long-term supremacy as more people choose their service and advertise their reliability and consistency in providing vehicles on demand. To address these real-world nuances, our future work will explore reformulating the reward function, possibly by introducing a regularization penalty against strategies that deprioritize passengers to maximize profit.

7 Conclusion & Future Work

As demand grows for sustainable, autonomous urban transportation, E-AMoD systems will play a critical role in meeting passenger demand across cities. In this project, we developed a comprehensive tri-level framework for E-AMoD fleet control that can maximize profits while simultaneously considering charging of electric vehicles as a key consideration in its decision-making. Starting off with a single-RL agent baseline, we implemented the first approach to sequential Multi-Agent Reinforcement Learning (MARL) in this problem space and formulated a novel strategy to train such sequential heterogeneous RL agents in a self-devised, non-linear toy setting. Our experiments make use of a graph-based state space that encodes multi-vehicle moving capabilities in a single timestep, and optimally accounts for both passenger-matching and vehicle rebalancing/recharging using two Actor-Critic single-reward RL agents.

In future work, beyond exploring more robust reward functions and longer training times, we plan to train our best models (baseline single-RL-agent and one-time freeze MARL) on real-world, real-time datasets, which we have available from ten cities in the United States and China including New York City and San Francisco. Gammelli et al. [1] have shown in their research that the baseline linear optimization methods (such as MPC) scale very poorly to more complex datasets like the ones from actual cities, so we are curious to see if the MARL strategies' somewhat better generalizability will hold up in performance on much more complex scenarios. Another promising avenue of research is to experiment with recently released MARL-Transformer and Centralized Advising and Decentralized Pruning (CADP) frameworks [14]. Because transformers are well-suited to tackle sequential problems, owing to their origin in RNN sequence-to-sequence modeling approaches, promising initial results have been shown by researchers that allow for more explicit communication and coupling of objectives in these sequential MARL problems.

8 Acknowledgments & Group Contributions

We'd like to thank Stanford's Autonomous Systems Lab, led by Professor Marco Pavone, for providing inspiration around the project idea and the initial codebase that we built off of for our methods, experiments, and results.

All three group members contributed equally to the project. Work among the group was split in the following way:

1. **Ayush:** Developed different flavors of alternative freezing to experiment with. Collected/documented/presented data generated during training and evaluation of RL agents.
2. **Aaryan:** Worked on adapting RL2 to RL1 so that it fits into the tri-level framework. Came up with the Warm Start Strategy.
3. **Justin:** Conducted a literature review of MARL techniques to identify which are applicable and flawed, analyzed where there is a gap in model research. Came up with the Alternate Freezing approach.

All three of us worked together to understand and add to the existing codebase. All code implementation of new experiments (including baselines) was done together, with equal contributions. The work for the poster and the paper was divided equally among the three of us.

Some of this work deviates from the original proposal because our project direction changed. For one, we discarded the idea of offline RL because we reasoned that using offline RL is not practical given the sequential nature of the problem (specifically, the tri-level framework). As such, a multitude of new experiments that were not mentioned in the proposal was conducted.

References

[1] Daniele Gammelli, Kaidi Yang, James Harrison, Filipe Rodrigues, Francisco C. Pereira, and Marco Pavone. Graph neural network reinforcement learning for autonomous mobility-on-demand systems. In *2021 60th IEEE Conference on Decision and Control (CDC)*, page 2996–3003. IEEE Press, 2021.

[2] Maxime Guériau, Federico Cugurullo, Ransford A. Acheampong, and Ivana Dusparic. Shared autonomous mobility on demand: A learning-based approach and its performance in the presence of traffic congestion. *IEEE Intelligent Transportation Systems Magazine*, 12, 01 2020.

[3] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1090–1095, Los Alamitos, CA, USA, nov 2019. IEEE Computer Society.

[4] Christian Fluri, Claudio Ruch, Julian Zilly, Jan Hakenberg, and Emilio Frazzoli. Learning to operate a fleet of cars. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2292–2298, 2019.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 07 2017.

[7] Ming Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *International Conference on Machine Learning*, 1997.

[8] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 2085–2087, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[9] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5872–5881. PMLR, 10–15 Jul 2018.

[10] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2681–2690. JMLR.org, 2017.

[11] Paul Barde, Julien Roy, Félix G. Harvey, Derek Nowrouzezahrai, and Christopher Pal. Promoting coordination through policy regularization in multi-agent deep reinforcement learning, 2020.

[12] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Changjie Fan, and Li Wang. Hierarchical deep multiagent reinforcement learning, 09 2018.

[13] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[14] Jiangxing Wang, Deheng Ye, and Zongqing Lu. More centralized training, still decentralized execution: Multi-agent conditional policy factorization. In *The Eleventh International Conference on Learning Representations*, 2023.