Dynamic Vehicle Routing Problem with Prompt Confirmation of Advance Requests

Amutheezan Sivagnanam*, Ayan Mukhopadhyay†, Samitha Samaranayake‡, Abhishek Dubey§ and Aron Laszka*
*Department of Informatics and Intelligent Systems
Pennsylvania State University, University Park, Pennsylvania
Emails: amutheezan@psu.edu and laszka@psu.edu
†Department of Computer Science
College of William & Mary, Williamsburg, Virginia
Email: amukhopadhyay@wm.edu
‡School of Civil and Environmental Engineering
Cornell University, Ithaca, New York
Email: samitha@cornell.edu
§College of Connected Computing
Vanderbilt University, Nashville, Tennessee
Email: abhishek.dubey@vanderbilt.edu

Published in the Proceedings of the 17th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS 2026).

***Abstract*—Transit agencies that operate on-demand transportation services have to respond to trip requests from passengers in real time, which involves solving *dynamic vehicle routing problems with pick-up and drop-off constraints*. Based on discussions with public transit agencies, we observe a real-world problem that is not addressed by prior work: when trips are booked in advance (e.g., trip requests arrive a few hours in advance of their requested pick-up times), the agency needs to *promptly confirm whether a request can be accepted or not*, and ensure that accepted requests are served as promised. State-of-the-art computational approaches either provide prompt confirmation but lack the ability to *continually optimize and improve routes for accepted requests*, or they provide continual optimization but cannot guarantee serving all accepted requests. To address this gap, we introduce a novel problem formulation of *dynamic vehicle routing with prompt confirmation and continual optimization*. We propose a novel computational approach for this vehicle routing problem, which integrates a quick insertion search for prompt confirmation with an anytime algorithm for continual optimization. To maximize the number requests served, we train a non-myopic objective function using reinforcement learning, which guides both the insertion and the anytime algorithms towards optimal, non-myopic solutions. We evaluate our computational approach on a real-world microtransit dataset from a public transit agency in the U.S., demonstrating that our proposed approach provides prompt confirmation while significantly increasing the number of requests served compared to existing approaches.**

***Index Terms*—Dynamic Vehicle Routing Problem, Public Transportation, Microtransit Service, Anytime Algorithm, Machine Learning**

I. Introduction

The Dynamic Vehicle Routing Problem (DVRP) is a sequential decision-making problem that models the operation of an on-demand transportation service, which serves requests for transportation between requested locations within requested time windows using a given set of vehicles with limited passenger capacities (e.g., ride-sharing, paratransit, or microtransit services). In a DVRP, requests are received sequentially while the vehicles are in operation, serving requests that were received and accepted earlier. When a request is received, acceptance and routing decisions have to be made under uncertainty since future requests are stochastic, i.e., only the distribution of future requests is known. A natural objective for this problem is to maximize the service rate, i.e., to maximize the number of requests that are accepted and served within the requested time windows.

Existing computational approaches for this problem can be divided into two categories. One category of algorithms decides whether to accept or reject a request when the request is received, and immediately assigns an accepted request to a vehicle manifest [1]–[3], removing any uncertainty for the passengers. The other category of algorithms delays the confirmation of acceptance to continuously optimize the assignments as new requests arrive [4], [5], increasing the malleability of vehicle manifests, which can lead to higher service rates. There is a gap between these two categories: an approach that provides *prompt confirmation of acceptance or rejection for every request*, while also enabling the *continuous improvement of vehicle manifests*. Both of these requirements are important from a practical perspective. On the one hand, people are more likely to use a service that provides prompt confirmation, since it enables them to plan ahead and be certain that they will reach their destination on time if their requests are accepted. On the other hand, the ability to continuously optimize assignments and vehicle manifests provides greater flexibility to service providers, which enables them to improve their service rates through optimization. In many practical applications, especially in emerging on-demand microtransit services, these advantages are crucial.

To address this gap, we propose a novel approach that combines a quick search algorithm for request confirmation

with an anytime algorithm for improving vehicle manifests *between* the arrival of requests, a period of time when existing computational approaches are idle. The quick search algorithm provides prompt confirmation, accepting or rejecting each request within a fraction of a second, which improves usability for passengers. Between the arrival of consecutive requests, the anytime algorithm continuously works on finding better manifests, which increases services rate. A key question faced by this approach is choosing an appropriate *objective function for the anytime algorithm*. While maximizing service rate may seem like a trivial choice for this objective, it is actually meaningless since the set of confirmed (i.e., accepted) requests is fixed between the arrival of consecutive requests. In fact, this objective function must be non-myopic: it must maximize the chance of accepting future requests and future improvements to manifests, thereby maximizing service rate on the long term.

To address this question, we formulate the problem of request confirmation and manifest optimization as a sequential decision-making problem under uncertainty, specifically, as a Markov decision process (MDP). The state of this MDP represents the state of the transportation service, i.e., vehicle locations, passengers on board each vehicle, set of accepted requests, current vehicle manifests, and the most recently received trip request. An action represents the choice of accepting or rejecting the most recent request (by the search algorithm) as well as the choice of the new manifests (by the anytime algorithm). We apply reinforcement learning to approximate the action-value function of the optimal policy in this MDP, which is the objective function that maximizes service rate in the long term in our model. We demonstrate using real-world and synthetic problem instances that our approach provides significantly better trade-off between confirmation time and service rate than existing approaches.

The remainder of this paper is organized as follows. Section II describes our formal model of the dynamic VRP with prompt request confirmation and continual manifest optimization. Section III introduces our proposed computational approach that integrates a quick search with an anytime algorithm and trains their objective function using reinforcement learning. Section IV presents numerical results, demonstrating that our approach achieves higher service rates with lower confirmation times compared to existing approaches. Section V discusses prior work on dynamic VRP with stochastic requests. Finally, Section VI provides concluding remarks.

II. Model and Problem Formulation

We begin by introducing our model of DVRP with prompt and guaranteed confirmation. Table I provides a list of key symbols used in the paper.

A. Dynamic Vehicle Routing Model

We let $\mathcal{L}$ denote the set of locations (e.g., pickup locations, dropoff locations, vehicle depots), which represents all points in the road network. For a pair of locations $l_1, l_2 \in \mathcal{L}$, we let $D(l_1, l_2)$ denote the travel time from l_1 to l_2. Note that our approach makes no assumptions about D, and it may incorporate a variety of travel-time models.

TABLE I
List of Symbols

Symbol	Description
	Variables
$\mathcal{T}_t$	Set of requests that arrived as of time t
$\mathcal{T}_t^{accept}$	Set of requests that are part of route plans at time t
	Constants
$\mathcal{L}$	Set of locations
$\mathcal{V}$	Set of vehicles
C	Maximum passenger capacity of the vehicles
l_k^{pickup}	Pickup location of request $T_k \in \mathcal{T}_t$
$l_k^{dropoff}$	Dropoff location of request $T_k \in \mathcal{T}_t$
$\tau_k^{earliest}$	Earliest pickup time of request $T_k \in \mathcal{T}_t$
τ_k^{latest}	Latest dropoff time of request $T_k \in \mathcal{T}_t$
$\tau_k^{arrival}$	Arrival time of request $T_k \in \mathcal{T}_t$
n_k	Number of passengers in request $T_k \in \mathcal{T}_t$
	Markov Decision Process
s_t	State of the environment at time t
$\mathcal{P}_t$	Set of vehicle locations for every vehicle $\mathcal{V}$ at time t
$\mathcal{R}_t^{pre}$	Set of route plans for every vehicle $\mathcal{V}$ at time t before making a decision
$\mathcal{R}_t^{post}$	Set of route plans for every vehicle $v \in \mathcal{V}$ after making a decision at time t
a_t	Action of the decision agent at time t
$\mathbf{r}(s, a)$	Reward for taking action a in state s
$Q(s, a)$	State-action value of taking action a in state s

Throughout the day, the transit agency receives requests one-by-one. [1] We let $\mathcal{T}_t = \{T_1, T_2, \ldots\}$ denote the set of requests that have arrived up to time t. Each request $T_k \in \mathcal{T}_t$ specifies a pickup location $l_k^{pickup} \in \mathcal{L}$; a dropoff location $l_k^{dropoff} \in \mathcal{L}$; an earliest pickup time $\tau_k^{earliest} \in \mathbb{R}^+$; a latest dropoff time $\tau_k^{latest} \in \mathbb{R}^+$; and the number of passengers $n_k \in \mathbb{N}^+$. Each request is drawn from a known distribution $\mathbf{D}$, i.e., $\langle l_k^{pickup}, l_k^{dropoff}, \tau_k^{earliest}, \tau_k^{latest}, n_k \rangle \sim \mathbf{D}$. As is usual in the DVRP literature, we assume that $\mathbf{D}$ can be modeled or simulated based on historical request data. We let $\tau_k^{arrival} \in \mathbb{R}^+$ ($\leq \tau_k^{earliest}$) denote the arrival time of request $T_k \in \mathcal{T}_t$, i.e., the time when the transit agency receives request T_k.

The transit agency uses a set of vehicles $\mathcal{V}$ to serve the requests, each vehicle having a fixed passenger capacity $C \in \mathbb{N}^+$ (for ease of presentation, we assume uniform capacity, but our approach can be applied naturally to vehicles with heterogeneous capacities). We let $R_v = \{(l_j, \tau_j, n_j) \mid j = 1, 2, \ldots, |R_v|\}$ denote the route plan (i.e., route manifest) for vehicle $v \in \mathcal{V}$, where $l_j \in \mathcal{L}$ is either the pickup or dropoff location of a request, $\tau_j \in \mathbb{R}^+$ is the time when vehicle v will depart from location l_j, and $n_j \in \{\pm 1, \pm 2, \ldots, \pm C\}$ is the change in the number of passengers on-board vehicle v due to stopping at location l_j. Note that a valid route plan includes two stops for each request T_k that is assigned to the vehicle

[1] Multiple requests arriving at the same time can be modeled simply as sequential arrivals with zero time gap between them.

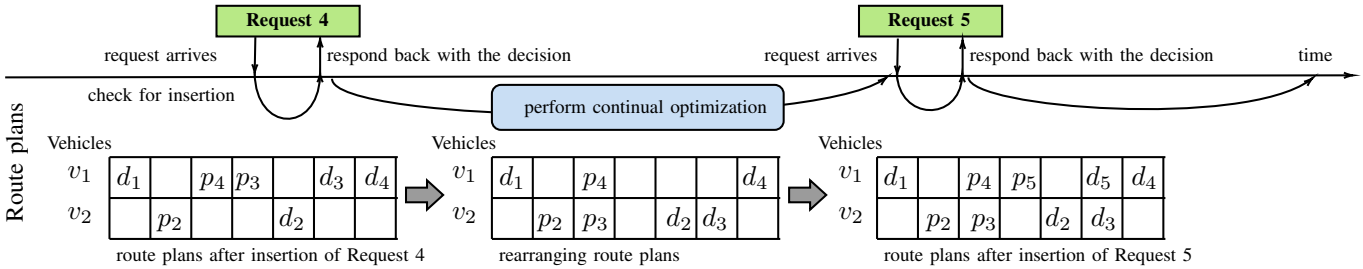

Fig. 1. During service, (1) requests arrive following a known distribution (e.g., based on historical data); (2) upon the arrival of each request, we decide whether to accept or reject the request given the current state of the service; (3) we promptly notify the passenger of our decision; (4) until the next request arrives, we continuously optimize route plans to better accommodate future requests. Symbols p_k and d_k represent the pickup and dropoff of request k, respectively.

but has not been picked up yet: a pickup stop at l_k^{pickup} and a dropoff stop at $l_k^{dropoff}$ (and if the request has already been picked up, only a dropoff stop at $l_k^{dropoff}$). At the beginning of the day, the route plan for each vehicle $v \in \mathcal{V}$ is an empty set. Vehicle $v \in \mathcal{V}$ starts its shift at time $v^{start} \in \mathbb{R}^+$ at the depot of the transit agency $l^{depot} \in \mathcal{L}$; it serves requests following its route plan R_v, which is updated in real time as requests are received by the transit agency; and it returns to the depot by the end of its shift $v^{end} \in \mathbb{R}^+$.

1) Operational Constraints: While operating the service, the transit agency must satisfy the following constraints: (1) each incoming request $T_k \in \mathcal{T}_t$ must be added to either the set of accepted requests $\mathcal{T}_t^{accept} \subseteq \mathcal{T}_t$ or rejected requests $\mathcal{T}_t^{reject} = \mathcal{T}_t \setminus \mathcal{T}_t^{accept}$; (2) every accepted request $T_k \in \mathcal{T}_t^{accept}$ that has not been picked up yet must be assigned to exactly one vehicle, satisfying the time-window constraints (i.e., request T_k must be picked up after $\tau_k^{earliest}$ and dropped off before τ_k^{latest}); (3) every accepted request $T_k \in \mathcal{T}_t^{accept}$ that has already been picked up must keep its vehicle assignment unchanged and be dropped off before τ_k^{latest}; (4) every route plan R_v must satisfy travel-time constraints (i.e., $D(l_j, l_{j+1}) \leq \tau_{j+1} - \tau_j$), starting from the current location of vehicle v (note that this constraint implies that the stops j are ordered by their departure times τ_j); (5) and the number of passengers on-board vehicle $v \in \mathcal{V}$ must never exceed the passenger capacity C.

B. Computational Assumptions

When the transit agency receives a request $T_k \in \mathcal{T}_t$ at time $t = \tau_k^{arrival}$, it must decide whether to accept or reject the request *within a matter of seconds*. To accept request T_k, the agency must assign the request to a vehicle $v \in \mathcal{V}$ and compute a feasible route plan R_v for the selected vehicle that satisfies all operational constraints (Section II-A1). Note that finding a feasible route plan is crucial to guaranteeing that the agency will be able to serve all accepted requests; if the agency accepted a request without having a feasible route plan, it would face the risk of never finding a feasible route plan and, hence, failing to serve some of the accepted requests. Between the arrival of two consecutive requests (i.e., between $\tau_k^{arrival}$ and $\tau_{k+1}^{arrival}$), the transit agency may update the route plan of each and every vehicle, as long as the updated route plans satisfy all operational constraints (Section II-A1). Note that the computational time available for updating route plans is stochastic since the arrival time $\tau_{k+1}^{arrival}$ of the next request is stochastic.

C. Metrics

To measure the performance of the transit service up to a given time t, we use *service rate* and *rejection rate* as our key metrics, which we define as the ratio of accepted requests $|\mathcal{T}_t^{accept}| \,/\, |\mathcal{T}_t|$ and the ratio of rejected requests $|\mathcal{T}_t^{reject}| \,/\, |\mathcal{T}_t|$, respectively.

D. Problem Formulation

Our problem formulation is a *two-stage optimization for a DVRP with advance requests*. First, upon the arrival of each request, we have to promptly decide whether to accept or reject the request (and if we decide to accept the request, we have to find a feasible route plan). Second, once we have made a decision, we may optimize the route plans to increase the chances of accepting future requests (we may continuously optimize routes plans until the arrival of the next request). The objective is to *maximize service rate* (or, equivalently, to *minimize rejection rate*) over a long time horizon ($t \to \infty$), satisfying the operational constraints of the DVRP (Section II-A1) and the computational constraints of decision making and optimization (Section II-B). Figure 1 shows an illustration of our problem setting and our proposed computational approach.

III. Computational Approach

We propose to solve this computational problem by introducing a (1) quick insertion algorithm for promptly deciding whether to accept or reject a request and for finding a feasible route plan and an (2) anytime algorithm for continual optimization of route plans between the arrival of consecutive requests. For both of these algorithms, we must specify an objective function, which enables us to choose between acceptance and rejection and enables us to choose between different route plans. We propose to learn a non-myopic objective function, which maximizes service rate on the long term, using reinforcement learning.

A. Markov Decision Process

To apply reinforcement learning, we model our DVRP problem as a Markov decision process.

a) Decision Epoch: A decision epoch occurs at each $t = \tau_k^{arrival}$, when request T_k is received. Between the arrival of consecutive requests, the state of the environment evolves in continuous time (e.g., vehicles drive around, picking up and dropping off passengers, following their route plans).

b) State: The state s_t of the environment at decision epoch $t = \tau_k^{arrival}$ includes the new request T_k, the current locations $\mathcal{P}_t$ of all the vehicles, the set of accepted requests $\mathcal{T}_t^{accept}$, and the current set of feasible route plans $\mathcal{R}_t^{pre}$ for all the vehicles.

c) Action: The action a_t taken at decision epoch t includes the decision whether to *accept* or *reject* request T_k as well as a new set of feasible route plans $\mathcal{R}_t^{post}$ for the vehicles. An action a_t can *accept* the new request T_k only if the new set of feasible route plans $\mathcal{R}_t^{post}$ serves all previously accepted requests $\mathcal{T}_t^{accept}$ as well as the new request T_k. If action a_t *rejects* the new request, then the new set of feasible route plans $\mathcal{R}_t^{post}$ has to serve only the previously accepted requirements $\mathcal{T}_t^{accept}$ (note that in this case, the new set of route plans $\mathcal{R}_t^{post}$ can be the same as the current set of route plans $\mathcal{R}_t^{pre}$).

d) State Transition: At each decision epoch t, the environment first transitions from state s_t to a post-decision state s_t^{post}. As part of this transition, the route plans of the vehicles $\mathcal{R}_t^{pre}$ are updated to the new, post-decision route plans $\mathcal{R}_t^{post}$. If action a_t *accepts* the new request T_k, then the new request is added to the set of accepted requests: $\mathcal{T}_t^{accept,post} = \mathcal{T}_t^{accept} \cup \{T_k\}$. Finally, the environment transitions to its next state $s_{t'}$ upon the arrival of the next request T_{k+1} at time $t' = \tau_{k+1}^{arrival}$.

e) Reward: The reward is $\mathbf{r}(s_t, a_t) = 1$ if action a_t *accepts* the new request, and $\mathbf{r}(s_t, a_t) = 0$ otherwise. Note that in the long term, maximizing rewards is equivalent to maximizing service rate (or minimizing rejection rate).

f) Action Value: For any state-action pair (s_t, a_t), the action value $Q(s_t, a_t)$ of the optimal policy is defined as

$$Q(s_t, a_t) = \mathbf{r}(s_t, a_t) + \gamma \mathbb{E}_{s_{t'}} \left[\max_{a'} Q(s_{t'}, a') \right], \qquad (1)$$

where $\gamma \in (0, 1)$ is a temporal discount factor, and $s_{t'}$ is the stochastic next state after taking action a_t. For $\gamma \approx 1$, an action a_t that maximizes the action-value Q also maximizes the long-term service rate; therefore, we will employ action-value Q as our objective function. In the following subsections, we will first discuss the optimization problems of finding optimal actions for prompt confirmation (Section III-B) and continual optimization (Section III-C), and then discuss how to learn an action-value estimator (Section III-D).

B. Prompt Confirmation

We first discuss the problem of prompt confirmation, which entails deciding whether to accept or reject a new request based on the current state of the environment (i.e., route plans, accepted requests, and vehicle locations), and if the decision is to accept, finding a feasible route plan that serves the new request.

a) Problem Input: The input of prompt confirmation for the decision epoch at time t consists of the new request T_k, the current location of each vehicle $\mathcal{P}_t = \{l^v \in \mathcal{L} \mid v \in \mathcal{V}\}$, the set of previously accepted requests $\mathcal{T}_t^{accept}$, and the current set of route plans $\mathcal{R}_t^{pre}$ for the vehicles, which serve all the previously accepted requests $\mathcal{T}_t^{accept}$.

b) Solution Space: A solution represents the decision whether to accept or reject the new request T_k, that is, whether to include the new request T_k in the set of accepted requests $\mathcal{T}_t^{accept,post}$, and a new set of feasible route plans $\mathcal{R}_t^{insert}$ (which may be the same as $\mathcal{R}_t^{pre}$ if the decision is to reject the new request).

c) Objective: We can formally express the objective of the problem as maximizing the action-value function Q of the MDP:

$$\max_{\mathcal{R}_t{}^{insert}} Q(\langle \mathcal{T}_t^{accept}, T_k, \mathcal{P}_t, \mathcal{R}_t^{pre} \rangle, \mathcal{R}_t{}^{insert}) \qquad (2)$$

where $\langle \mathcal{T}_t^{accept}, T_k, \mathcal{P}_t, \mathcal{R}_t^{pre} \rangle$ is the state at time t. Note that we omit $\mathcal{T}_t^{accept,post}$ from the action representation since the inclusion (or exclusion) of request T_k in the route plans $\mathcal{R}_t^{insert}$ indicates acceptance (or rejection).

d) Quick Insertion Algorithm: To ensure that we can solve this optimization promptly, we restrict our search space for $\mathcal{R}_t{}^{insert}$ to *searching for a simple insertion* of request T_k into the current route plans $\mathcal{R}_t{}^{pre}$, that is, adding T_k to $\mathcal{R}_t^{pre}$ without changing the vehicle assignments or the order of previously accepted requests. To find a simple insertion, we can exhaustively search over each stop of each vehicle manifest $R_v \in \mathcal{R}_t^{pre}$, test if we can insert the pickup and dropoff stops of T_k without violating the time or capacity constraints (Section II-A1), and select the feasible insertion that maximizes the Q value. Since the stops of a route plan are always ordered by their departure times, the computational complexity of exhaustively searching for feasible insertions of both the pickup and dropoff stops is $O(|\mathcal{T}_t^{accept}|^2)$. First, slacks in the schedule (i.e., additional time that may be spent between two consecutive stops without violating the time constraints) can be calculated by one forward and one backward pass over the route plan (linear time $O(|\mathcal{T}_t^{accept}|)$). Then, for each possible insertion of the pickup stop, a forward pass can check all possible insertions of the dropoff stop (quadratic time $O(|\mathcal{T}_t^{accept}|^2)$). However, the search space is typically much smaller in practice. Our experimental results demonstrate that (1) we can perform an exhaustive search over all possible insertions in a fraction of a second in practice, which means that restricting the search space to insertions enables prompt confirmation, and (2) our insertion-based prompt confirmation combined with the continual optimization attains higher services rates than existing approaches.

C. Continual Optimization

Second, we discuss the problem of continual optimization, which entails improving route plans after each confirmation.

a) Problem Input: The input of continual optimization consists of the current location of each vehicle $\mathcal{P}_t = \{l^v \in \mathcal{L} \mid v \in \mathcal{V}\}$, the set of accepted requests $\mathcal{T}_t^{accept,post}$, and the set of feasible route plans output by the prompt confirmation $\mathcal{R}_t^{insert}$.

b) Solution Space: A solution is an updated set of feasible route plans $\mathcal{R}_t^{post}$ for the vehicles, which serve all requests in $\mathcal{T}_t^{accept,post}$.

c) Objective: We can formally express the objective of the problem as maximizing the action-value function Q of the MDP:

$$\max_{\mathcal{R}_t^{post}} Q(\langle \mathcal{T}_t^{accept,post}, \mathcal{P}_t, \mathcal{R}_t^{insert} \rangle, \mathcal{R}_t^{post}) \tag{3}$$

where $\langle \mathcal{T}_t^{accept,post}, \mathcal{P}_t, \mathcal{R}_t^{insert} \rangle$ is the state of the environment at time t after the prompt confirmation.

d) Anytime Optimization Algorithm: Solving the above problem poses two challenges. First, similar to other formulations of vehicle routing, this problem is NP-hard [6]; even with a single vehicle, the problem is more general than the classical NP-hard Traveling Salesman Problem. Second, we do not know in advance how much computational time $\tau_{k+1}^{arrival} - t$ we will have to solve the problem, since the arrival time $\tau_{k+1}^{arrival}$ of the next request is stochastic. To address these challenges, we propose to apply an *anytime metaheuristic algorithm*. The advantage of employing an anytime algorithm is that we can terminate the algorithm whenever the next request T_{k+1} arrives, and we can let $\mathcal{R}_t^{post}$ be the best feasible solution found by the algorithm up that point.

e) Simulated Annealing: We propose a *simulated-annealing algorithm* as the anytime metaheuristic for continual optimization [7]. We start the simulated-annealing algorithm with $\mathcal{R}_t^{insert}$ as the initial feasible solution. Starting from this initial feasible solution, the algorithm will try to improve the solution in iterations. In each iteration of the algorithm, we perform a set of random mutation operations, including *Swap*, which randomly chooses two route plans and a swappable request from each route plan (i.e., a request that is on the route but not yet picked up), and swaps these requests between the two route plans; *Move*, which randomly chooses two route plans and a swappable request from one of these route plans, and moves this request to the other route plan; *Shift*, which randomly chooses one route plan and one stop (either a pickup or dropoff) and changes the order of the stop within the route; and *Reverse*, which selects two or more subsequent stops in a route plan and reverses their order (note that these mutation operations are commonly used for solving VRPs [8]–[11]). If the mutation results in an infeasible solution, then we revert it; if the mutation results in a feasible solution with a lower Q value, then we revert it with some probability; otherwise, we continue the next iteration from the mutated solution. When the next request T_{k+1} arrives, we terminate the simulated-annealing algorithm, and we select the best feasible solution found by the algorithm up to that point as the updated set of route plans $\mathcal{R}_t^{post}$. This way, simulated annealing acts as an anytime algorithm.

D. Reinforcement Learning for Value Estimation

To learn the action-value function Q, we employ reinforcement learning. During learning, we simulate the environment, and at each decision epoch t (i.e., at the arrival of each request T_k), we find an action a_t that maximizes our current estimate of function Q by solving the optimization problems in Sections III-B and III-C for state s_t using the proposed algorithms. We record the state transition as an experience, i.e., a tuple that consists of the state s_t at time t, the chosen action a_t (i.e., decision to accept or reject T_k and the updated set of route plans $\mathcal{R}_t^{post}$), the next state $s_{t'}$ (where $t' = \tau_{k+1}^{arrival}$), and the reward $\mathbf{r}(s_t, a_t)$. From the recorded experiences, we can learn the action-value function Q using Q-learning [12], [13] based on the standard Bellman equation:

$$Q(s_t, a_t) \leftarrow \mathbf{r}(s_t, a_t) + \gamma \max_{a'} Q(s_{t'}, a') \tag{4}$$

Note that Q-learning is a value-based reinforcement learning approach, which means that there is no explicit policy π^* mapping from states to actions; the optimal policy π^* is implicitly defined as the solution to the maximization $\operatorname{argmax}_a Q(s_t, a)$ in prompt confirmation and continual optimization. Next, we discuss how to design a machine learning model that can take as input a state-action pair (s_t, a_t) and output an estimated Q value.

a) Feature Vectors: Although prompt confirmation and continual optimization have different action spaces, the result of an action is always an updated set of routes, serving a set of requests. So, we can learn a single Q-function, whose input is a set of feasible routes (which implicitly specifies the set of accepted requests). Accordingly, we transform the complex state-action pair representation (s_t, a_t) into a *fixed-length feature vector* and feed it into a *neural network* Q. For the neural network, we explore fully-connected feed-forward networks (MLP), Kolmogorov-Arnold networks (KAN) [14], and convolutional neural networks (CNN); we present experimental results for these various architectures as part of our numerical evaluation. Here, we describe three different feature-vector representations that we can feed into the various neural network architectures. We present a more detailed discussion of hyperparameter search and feature selection in Appendix A.

Total idle time: First, we introduce a simple feature $\boldsymbol{x}$ that measures the total idle time (i.e., time when a vehicle is idle between serving two requests) for all route plans for the next h hours:

$$\boldsymbol{x} = \sum_{R_v \in \mathcal{R}_t} \sum_{j=0}^{|R_v|} \mathbf{I}(R_v{}^+, j) \tag{5}$$

where $R_v{}^+ = \{(l_j, \tau_j, n_j) \mid j = 0, 1, 2, \ldots, |R_v + 1|\}$ denotes the complete route plan, including the start (i.e., $j = 0$) and end stops (i.e., $j = |R_v + 1|$) of the route, where the start stop is either the initial location (i.e., stationed at a depot) or the current location of the vehicle, and the end stop is stationed back at the depot. $\mathbf{I}(R_v{}^+, j)$ denotes the idle time between

two consecutive stops j and $j+1$ in the route plan. We can formally express $\mathbf{I}({R_v}^+, j)$ as follows:

$$\mathbf{I}({R_v}^+, j) = \max\{\min\{\tau_{j+1}, t+h\} - \tau_j - D(l_j, l_{j+1}), 0\}$$

Temporal availability: Next, we consider a feature vector $\boldsymbol{x}_w$ that captures the fine-grained availability of vehicles for next h hours. Specifically, we divide the next h hours into time intervals of length w, resulting in $\frac{h}{w}$ time intervals. For time interval $i \in \{0, \ldots, \frac{h}{w} - 1\}$, we count the number of vehicles that are idle during time interval i, and we let the feature value $(\boldsymbol{x}_w)_i$ be this number.

Spatio-temporal availability: Finally, we introduce a feature vector $\boldsymbol{x}_a$, which represents summarized vehicle availability over the next h hours with fine granularity. To construct this feature, we first partition the spatial area of the service into a two-dimensional grid of size ($g \times g$). Each grid cell $[c_p, c_q]$ (where $c_p, c_q \in \{1, \ldots, g\}$) is further divided into time intervals of length w, yielding a total of $\frac{h}{w}$ intervals. For each interval $i \in \{0, \ldots, \frac{h}{w} - 1\}$, we count the number of idle vehicles and denote this count as $(\chi_a)_i^{pq}$. Next, for each grid cell $[c_p, c_q]$, we compute the minimum number of idle vehicles in the continuous subarray of size $N_{wm} \times N_{ci}$, where N_{wm} represents the multiple of the window to be considered, and N_{ci} represents the number of such continuous intervals to be evaluated. Accordingly, we compute the feature vector for all possible combinations of $[c_p, c_q] \in \{1, \ldots, g\} \times \{1, \ldots, g\}$, $n_{ci} \in \{1, \ldots, N_{ci}\}$, and $n_{mv} \in \{1, \ldots, N_{mv}\}$:

$$\boldsymbol{x}_a = \sum_{i=\kappa}^{\frac{h}{w}} \mathbf{1} \mid \{\min\{(\chi_a)_{i-\kappa}^{pq}, \ldots, (\chi_a)_{i-1}^{pq}\} \geq n_{mv}\}$$

where $\kappa = N_{wm} \times n_{ci}$. Accordingly, at the end we obtain a feature vector of size $g \times g \times N_{ci} \times N_{mv}$.

The mapping from the complete state s_t to a fixed-length feature vector ($\boldsymbol{x}$, $\boldsymbol{x}_w$, or $\boldsymbol{x}_a$) is necessary because the complete state representation is highly unstructured and variable length. The mapping reduces this to a structured, fixed-length vector, which can be fed into a relatively simple neural network. Alternatively, we could try to employ a complex neural architecture to ingest the complete representation. However, a complex architecture would require substantially more training experiences. Our experimental results in Section IV demonstrate that our vector-representations are both computationally tractable and attain excellent performance.

E. Supervised Pre-Training

While reinforcement based training can lead to an optimal solution, the training process is computationally expensive. To tackle this challenge, we apply simple supervised learning for pre-training, where we first gather experiences by simulating the environment with a simple policy π^0. This simple policy π^0 always accepts the new request T_k if there is a feasible insertion, and it chooses action a_t by maximizing the total idle time for next h hours:

$$\pi^0(s_t) = \text{argmax}_{a_t} \sum_{R_v \in \mathcal{R}_t} \sum_{k=0}^{|R_v|} \mathbf{I}({R_v}^+, k) \tag{6}$$

where $\mathcal{R}_t$ is the set of route plan that we obtain from applying the action a_t to the state s_t.

To apply supervised learning, we need to provide ground truth; in this case, we must provide ground truth for the action values $Q^{\pi^0}(s, a)$ of the simple policy π^0, which we will use to pre-train our neural network Q. We estimate the ground truth $Q^{\pi^0}(s, a)$ based on the discounted sum of the future rewards for next k steps:

$$Q^{\pi^0}(s_t, a_t) \approx \sum_{i=0}^{k-1} \gamma^i \mathbf{r}(s_{t+i}, a_{t+i}) \tag{7}$$

This k-step Monte Carlo estimate of the action value $Q^{\pi^0}(s, a)$ strikes a balance between minimizing bias in the estimate (due to considering a limited number of steps k) and minimizing the variance in the estimate (due to the stochasticity of future state transitions and rewards).

Once the supervised pre-training of our neural network Q has converged, we can apply reinforcement learning with the Bellman equation to fine-tune the policy, as described in Section III-D. Once reinforcement learning has converged, we can deploy our approach with the learned Q; during execution, we only need to solve $\text{argmax}_a\, Q(s_t, a)$ for each decision.

IV. Numerical Evaluation

A. Datasets and Experimental Setup

We evaluate our proposed computational approach on two datasets: a real-world microtransit dataset from a mid-size U.S. city and the widely-used NYC taxi dataset. This novel microtransit dataset and all software artifacts, including the implementation of our simulation environment and our computational framework, are available at `https://github.com/aronlaszka/DVRP-AR`

a) Microtransit Data: We obtained real-world microtransit data from a public transit agency that serves a mid-size U.S. city. The dataset contains trip requests from August 2022 to October 2023. We model the arrival $\tau_k^{arrival}$ of requests T_k as a Poisson process, matching the average arrival rate of the dataset (i.e., 28 requests per hour). We model the time difference between request arrival $\tau_k^{arrival}$ and requested pickup time $\tau_k^{earliest}$ as an exponential distribution, matching the average time difference between request arrival and requested pickup time of the dataset (i.e., 2 hours). We consider a vehicle capacity of $c = 8$ and let the number of vehicles be $|\mathcal{V}| = 4$, matching the resources of the real-world service. We calculate travel times between all pairs of locations using real road-network data from OpenStreetMap. The road network contains 4,555 nodes (i.e., intersection) and 11,410 edges (i.e., road segments).

b) NYC Taxi Data: We also evaluate our approach on the NYC taxi dataset, which has been used very widely in VRP research [1], [15], [16]. The NYC taxi dataset consists of trip requests from January 2016. We model the arrival of requests as a Poisson process with an arrival rate of 60 requests per hour. Since this dataset contains requests with only a short lead time (i.e., short time between request

arrival and requested pickup time, since the service did not support advance booking), we adapted the dataset so that some requests are booked in advance. Specifically, we model the time difference between request arrival and requested pickup time as an exponential distribution, matching the average time difference between request arrival and requested pickup time in the microtransit dataset (i.e., 2 hours). Similarly, we again consider a vehicle capacity of $c = 8$ and let the number of vehicles be $|\mathcal{V}| = 4$. We again calculate travel times using real road-network data from OpenStreetMap. The road network contains 14,512 nodes (i.e., intersection) and 33,763 edges (i.e., road segments).

c) Evaluation Episodes: For evaluation, we consider each episode to be 1 day. For the microtransit data, we consider 5 different episodes with 616 requests in total, and for the NYC taxi data, we consider 5 different episodes with 1,140 requests in total. To make the comparison fair, we evaluate our proposed approach and all the baseline approaches on the exact same 5 + 5 sets of requests.

B. Baseline Approaches

We compare our proposed solution approach to the following baseline approaches.

- **Google OR-Tools:** OR-Tools Routing is a publicly available, state-of-the-art VRP solver, developed by Google [17]. We implemented a dynamic VRP solver based on the OR-Tools Routing API by using the *Local Cheapest Cost Insertion* algorithm to make the quick insertion decision at the arrival of a request and using the *Guided Local Search* algorithm to optimize the route plans between the arrival of the requests.
- **Rolling Horizon (RH):** Kim *et al.* [16] recently published a state-of-the-art VRP solver that supports pickups and dropoffs with time windows, based on a rolling-horizon temporal decomposition and the widely used RTV-ILP framework for dynamic VRP [1]. We apply the Rolling Horizon (RH) solver by running a complete RH optimization upon the arrival of each request to determine whether we can accept the request and to optimize route plans.
- **MC VRP:** Wilbur *et al.* [2] introduced a dynamic VRP solver that is non-myopic (i.e., considers future requests when making acceptance and route-optimization decisions) based on Monte Carlo tree search (MC VRP). We run a full MC VRP search on the arrival of each request.

Although prior work has explored learning-based solution approaches for dynamic VRP, most existing approaches consider simplified or abstract settings, and as a result, none of them can be applied to our problem setting without substantial changes. For example, Joe and Lau [3] consider a dynamic VRP with only dropoffs (no pickups); Zhang *et al.* [18] consider only a single vehicle. Due to these limitations, we cannot make a direct, fair comparison between our proposed approach and these existing learning-based approaches; we discuss the limitations of existing approaches in more detail in Section V.

C. Implementation and Hyperparameters

a) Software and Hardware: We implemented our framework using Python 3.11 and Keras 3.6 with PyTorch 2.5 backend [19]. We ran all algorithms on a server with 2 AMD EPYC 7763 64-core CPUs, 1TB of RAM, and 2 NVIDIA RTX A5000 GPUs with 24GB of VRAM.

b) Architectures and Hyperparameters: We represent our action-value function $Q(s, a)$ as a neural network. We performed an architecture search over different types of neural networks, including multi-layer perceptron (MLP), Kolmogorov-Arnold network (KAN), and convolutional neural network (CNN). We describe these architectures and their hyperparameters in detail in Appendix A. For MLP, we optimized the number of layers, the number of neurons in each layer, and the drop-out rates; for KAN, we optimized the number of layers and the width of each layer; and for CNN, we optimized the number of channels, the size of the kernel, the size of the feed-forward network, and the drop-out rates. In all cases, we used a learning rate of 0.001 for gradient descent, and a temporal discount factor of $\gamma = 0.9$.

c) Pre-Training: For both the microtransit dataset and the NYC taxi dataset, we gathered 1 million experiences for supervised pre-training by simulating the simple policy π^0 (i.e., always accepting requests and maximizing idle time). We performed the actual pre-training with 800,000 experiences, using the remaining 200,000 experiences as a hold-out test set. The trained Q-function achieves an R^2 score of 0.995 for microtransit data and 0.984 for NYC taxi data. With a batch size of 1024, each training epoch takes around 15 seconds for MLP, 33 seconds for CNN, and around 45 seconds and for KAN. On average, the supervised pre-training converged after 11 epochs (less than 3 minutes) for MLP, after 130 epochs (around 97 minutes) for KAN, and after 22 epochs (around 12 minutes) for CNN. Among the three neural-network architectures, MLP and KAN perform slightly better than CNN with respect to the rejection rate, on both the microtransit and the NYC taxi data. We provide results based on MLP models in the main text, and we provide results based on the other two architectures in Appendices B and C.

D. Numerical Results

1) Running Time of Confirmation: A key consideration is how quickly we can confirm the acceptance or rejection of a trip request. The average running time of our confirmation algorithm with the trained action-value function Q is around *0.2 seconds for microtransit data* and around *1 second for NYC data*. This is sufficiently low to accept or reject incoming requests in real time. **Google OR-Tools** takes around 0.1 seconds for microtransit data and around 0.5 seconds for NYC data, which is also sufficiently low (however, it underperforms our proposed approach in terms of rejection rate, as discussed below). For **RH**, we let the rolling-horizon factor be 2, and we let the solver perform RTV-graph generation for up to 1 second per batch, where each batch groups requests within a pre-defined interval (e.g., 60 seconds). For microtransit data, solving each decision epoch using the RH approach takes 50

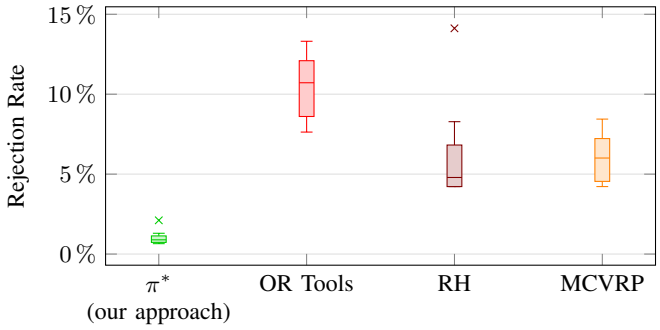

Fig. 2. Distribution of request rejection rates across 5 different episodes from the real-world microtransit data.

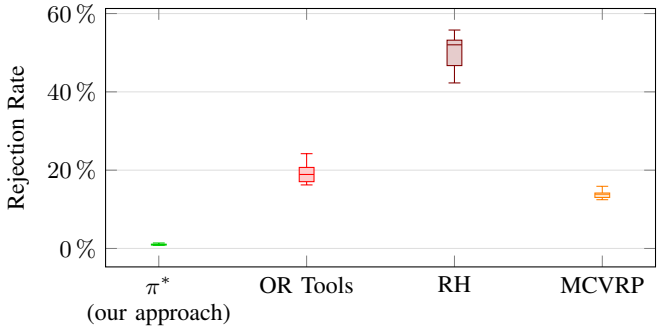

Fig. 3. Distribution of request rejection rates across 5 different episodes from the NYC taxi data.

seconds on average, with a minimum of 7 seconds and a maximum of 122 seconds. In Appendix H, we discuss the results of additional experiments where we let the RTV-graph generation take up to 5, 10, and 15 seconds with various rolling horizon factors, such as 1 and 2. For **MC VRP**, we let the running time at each decision epoch be 30 seconds. We provide additional experimental results with different running times, ranging from 1 to 30 seconds per decision epoch, in Appendix G.

2) Rejection Rates: The second key consideration is what fraction of requests we have to reject. Figure 2 shows the distribution of rejection rates over 5 different episodes from the microtransit data. Our proposed computational approach π^* significantly outperforms the baselines, and it is able to reduce rejection rates to around 1%. Figure 3 shows the distribution of rejection rates over 5 different episodes from NYC data. Again, our computational approach π^* significantly outperforms all baselines. These results demonstrate that our proposed approach, which combines a quick insertion for confirmation with an anytime metaheuristic for continual optimization, provides by far the best trade-off: it significantly outperforms Google OR-Tools in terms of rejection rate, and it significantly outperforms the other two baselines in terms of both rejection rate and confirmation time.

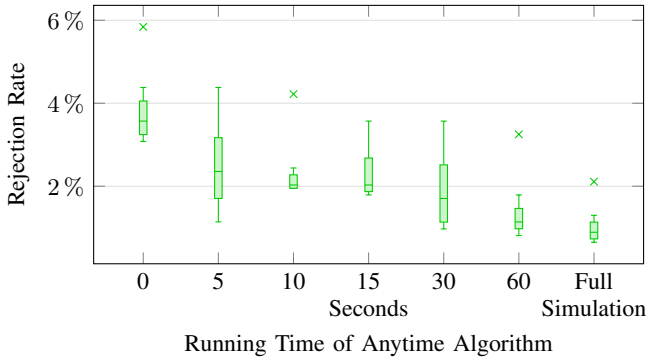

Fig. 4. Distribution of rejection rates across 5 different episodes from the real-world microtransit data, with various running-time limits for the anytime algorithm (in seconds).

E. Ablation Studies

Finally, we present ablation studies that investigate the benefits of continual optimization and employing the learned objective Q.

1) Varying the Running Time of the Anytime Algorithm: First, we investigate the benefits of continual optimization, i.e., the benefit of running the anytime algorithm between the arrival of consecutive requests to improve route plans. Figure 4 shows how the rejection rate depends on the running time of the anytime algorithm (on the microtransit data). In this experiment, we limit the running time of the anytime algorithm to a given time allotment instead of running it until the arrival of the next request. The results demonstrate the importance of taking advantage of the time between consecutive request arrivals since the rejection rate decreases significantly as the running time increases.

2) Learned Q-Function: Second, we investigate the benefits of employing the learned action-value function Q as a non-myopic objective. In these experiments, we replace the learned Q-function with the objective of the simple heuristic π^0 in various parts of our framework, including the acceptance/rejection decision, the quick insertion, and the continual optimization. We provide the results of these various ablation studies in Appendix E.

V. Related Work

Here, we provide an overview of computational approaches for dynamic VRP, focusing on recent learning-based approaches, which are most relevant to our work. For a more detailed overview of neural combinatorial optimization for vehicle routing problems, we refer the reader to the recent survey by Wu *et al.* [20].

A. Supervised Learning

Jungel *et al.* [21] propose learning-based online optimization for large-scale online dispatching and rebalancing. To find online, non-myopic dispatching and rebalancing actions, the approach formulates finding a decision as a combinatorial optimization problem that is parameterized by a machine

learning (ML) layer and trained using structured learning (SL). In contrast to our approach, Jungel et al. assume that rides cannot be shared. Time is discretized into periods, and at the end of each period, the system decides which requests to accept out of requests arrived within each period. Due to this, all the requests have to wait until the end of the period for confirmation of acceptance.

Baty *et al.* [22] consider the DVRP with Time Windows (DVRPTW), where requests must be served by strictly satisfying time windows. In contrast to our work, Baty et al. assume that there is an unlimited fleet of homogeneous vehicles and the goal is minimizing total travel costs. They propose a ML pipeline that incorporates a combinatorial optimization layer. This pipeline discretizes the planning horizon into a set of epochs: in each epoch, the fleet operator solves a dispatching and vehicle routing problem for the epoch-dependent request set (i.e., requests that have been received but not served yet), deciding which requests to serve in that epoch. In both [21], [22], once a request is assigned to a vehicle, the assignment cannot be changed. In contrast, our approach continually optimizes routes, providing a significant advantage (see Figure 4).

B. Reinforcement Learning and Neural Approximate Dynamic Programming

Joe and Lau [3] consider the DVRP with both known and stochastic requests, formulating it as a route-based MDP. In their model, the state of the MDP encompasses vehicle locations, manifests, and received requests; an action is updating the manifest of a vehicle; and rewards capture the objective of minimizing travel time and time-windows violations, while all the requests must be served. In contrast, the objective of our approach is to maximize service rate, while strictly satisfying time windows. Joe and Lau employ temporal-difference (TD) learning to learn an approximate value function of the optimal policy in this MDP and they propose a routing heuristic based on simulated annealing to find actions that maximize the value function. In contrast to our work, Joe and Lau do not consider running simulated annealing between the arrival of consecutive requests; they constrain the action space so that requests are always accepted and cannot be swapped between vehicles; and they consider delivery (i.e., drop-off only) instead of pick-up and drop-off problem.

Shah *et al.* [23] propose a neural approximate dynamic programming approach, called NeurADP, for the on-demand ride-pooling problem. They formulate the Ride-pool Matching Problem (RMP) with a given set of vehicles, stochastic requests, and the objective of maximizing the number of requests served. In this formulation, requests must be served in the near future, subject to delay constraints (maximum allowed pick-up delay, maximum allowed detour delay); in contrast, our approach considers arbitrary pick-up and drop-off time windows. NeurADP divides time into decision epochs, and at the end of each epoch, it assigns a subset of the requests received in the current epoch to the vehicles; once a request is assigned to a vehicle, its assignment cannot be changed. In contrast, our approach provides prompt confirmation for requests (instead of waiting until the end of an epoch) and continual optimization of route plans (including changing assignments).

Ma *et al.* [24] consider DVRP with both pickups and dropoffs. They solve the problem using hierarchal RL; when a request is received, it is first added to a buffer, and a high-level RL agent (based on DQN) decides when to release the requests. Once released the request are assigned to vehicles and manifests are iteratively optimized by a low-level RL agent (based on REINFORCE). In contrast to our approach, this framework runs iterative optimization intermittently (instead of continually) and for limited time, and it does not consider the choice of accepting or rejecting requests (minimizing time-window violations instead). Zhang *et al.* [18] consider the Dynamic Traveling Salesman Problem (DTSP) and the Dynamic Pickup and Delivery Problem (DPDP) with stochastic, time-dependent travel times and stochastic customers (i.e., customers may be dynamically added to or deleted from the pool of customer who have not been picked up yet). Note that this approach considers a single vehicle. Pan and Liu [25] consider the Dynamic and Uncertain Vehicle Routing Problem (DU-VRP), whose objective is to serve the uncertain demands of customers in a dynamic environment.

Azagirre *et al.* [26] present the RL-based platform that Lyft uses for the online rideshare matching problem, which matches trip requests with available drivers in batch. Since matching decisions affect the distribution of available drivers (and, hence, the number of requests that the platform will be able to fulfill in the future), the matching must be non-myopic. To this end, the platform models the online rideshare matching problem as a MDP, and it employs an online RL framework that learns and updates the value function completely online and on-policy while generating the matching decisions in real time.

Finally, Huang *et al.* [27] propose a heuristic based solution approach for a DVRP with pre-booked and on-demand requests. However, acceptance decisions are delayed until operational hours begin, and the approach considers only single-capacity vehicles without ride-sharing.

VI. Conclusions

Motivated by the emergence of on-demand microtransit services, we introduced a novel problem formulation, the *dynamic vehicle routing problem with advance booking*, and a novel computational approach that provides prompt confirmation of trip requests while enabling the continual improvement of route plans. Our experimental results demonstrate that our approach can make decisions on whether to accept or reject a request *in a fraction of a second* and that our anytime optimization of route plans leads to *substantially lower rejection rates* than existing approaches. The significance of these results lies in the ability to deploy on-demand microtransit services that (1) allow passengers to request trips in advance, promptly confirming the acceptance (or rejection) of their requests and guaranteeing that accepted requests will be served,

and (2) achieve high service efficiency in terms of the fraction of trip requests that are accepted and served on time.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation (NSF) under Award No. CNS-1952011 and by the U.S. Department of Energy (DOE) under Award No. DE-EE0011188. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF and the U.S. DOE. We thank the anonymous reviewers for their feedback on our work and for their suggestions to improve our manuscript.

REFERENCES

[1] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017.

[2] M. Wilbur, S. U. Kadir, Y. Kim, G. Pettet, A. Mukhopadhyay, P. Pugliese, S. Samaranayake, A. Laszka, and A. Dubey, "An online approach to solve the dynamic vehicle routing problem with stochastic trip requests for paratransit services," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2022, pp. 147–158.

[3] W. Joe and H. C. Lau, "Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers," in *30th International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 30, 2020, pp. 394–402.

[4] A. Čivilis, A. Barauskas, A. Brilingaitė, L. Bukauskas, and S. Šaltenis, "Managing advanced and flexible ride-hailing requests," in *Proceedings of the 16th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, 2023, pp. 62–69.

[5] F. Karami, W. Vancroonenburg, and G. Vanden Berghe, "A periodic optimization approach to dynamic pickup and delivery problems with time windows," *Journal of Scheduling*, vol. 23, pp. 711–731, 2020.

[6] J. K. Lenstra and A. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[7] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.

[8] A. Mor and M. G. Speranza, "Vehicle routing problems over time: a survey," *Annals of Operations Research*, vol. 314, no. 1, pp. 255–275, 2022.

[9] F. Liu, C. Lu, L. Gui, Q. Zhang, X. Tong, and M. Yuan, "Heuristics for vehicle routing problem: A survey and recent advances," *arXiv preprint arXiv:2303.04147*, 2023.

[10] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of Operations Research*, vol. 41, no. 4, pp. 421–451, 1993.

[11] W.-C. Chiang and R. A. Russell, "Simulated annealing metaheuristics for the vehicle routing problem with time windows," *Annals of Operations Research*, vol. 63, no. 1, pp. 3–27, 1996.

[12] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[14] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacic, T. Y. Hou, and M. Tegmark, "KAN: Kolmogorov–arnold networks," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=Ozo7qJ5vZi

[15] NYC, "TLC Trip Record Data," https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page, 2016, online; accessed: August 15th, 2024.

[16] Y. Kim, D. Edirimanna, M. Wilbur, P. Pugliese, A. Laszka, A. Dubey, and S. Samaranayake, "Rolling horizon based temporal decomposition for the offline pickup and delivery problem with time windows," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 5151–5159.

[17] Google, "Google OR-Tools: Vehicle Routing Problem," https://developers.google.com/optimization/routing/vrp, 2024, online; accessed: August 15th, 2024.

[18] Z. Zhang, H. Liu, M. Zhou, and J. Wang, "Solving dynamic traveling salesman problems with deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 4, pp. 2119–2132, 2021.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and others, "PyTorch: An imperative style, high-performance deep learning library," in *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

[20] X. Wu, D. Wang, L. Wen, Y. Xiao, C. Wu, Y. Wu, C. Yu, D. L. Maskell, and Y. Zhou, "Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives," *arXiv preprint arXiv:2406.00415*, 2024.

[21] K. Jungel, A. Parmentier, M. Schiffer, and T. Vidal, "Learning-based online optimization for autonomous mobility-on-demand fleet control," *arXiv preprint arXiv:2302.03963*, 2023.

[22] L. Baty, K. Jungel, P. S. Klein, A. Parmentier, and M. Schiffer, "Combinatorial optimization-enriched machine learning to solve the dynamic vehicle routing problem with time windows," *Transportation Science*, 2024.

[23] S. Shah, M. Lowalekar, and P. Varakantham, "Neural approximate dynamic programming for on-demand ride-pooling," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 34, 2020, pp. 507–515.

[24] Y. Ma, X. Hao, J. Hao, J. Lu, X. Liu, T. Xialiang, M. Yuan, Z. Li, J. Tang, and Z. Meng, "A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 23 609–23 620, 2021.

[25] W. Pan and S. Q. Liu, "Deep reinforcement learning for the dynamic and uncertain vehicle routing problem," *Applied Intelligence*, vol. 53, no. 1, pp. 405–422, 2023.

[26] X. Azagirre, A. Balwally, G. Candeli, N. Chamandy, B. Han, A. King, H. Lee, M. Loncaric, S. Martin, V. Narasiman *et al.*, "A better match for drivers and riders: Reinforcement learning at Lyft," *INFORMS Journal on Applied Analytics*, vol. 54, no. 1, pp. 71–83, 2024.

[27] T. Huang, B. Fang, X. Bei, and F. Fang, "Dynamic trip-vehicle dispatch with scheduled and on-demand requests," in *Uncertainty in artificial intelligence*. PMLR, 2020, pp. 250–260.

APPENDIX

A. Neural-Network Architecture Search

a) MLP: For MLP-based architecture, we optimize the number of hidden layers, the number of neurons for each hidden layer, and the dropout rate. We consider hyperparameters in the following search space:

- Number of Hidden Layers: 1 / 2 / 3 layers
- Number of Neurons per Hidden Layers: 32 / 64 / 128 neurons per layer
- Dropout Rate (after each hidden layer): 0.0 / 0.1 / 0.2

During the hyperparameter search, we consider features based on the following two sets of features and their combinations $\boldsymbol{x}, \boldsymbol{x}_a$. For $\boldsymbol{x}_a$ and combination of $\boldsymbol{x}, \boldsymbol{x}_a$, we consider the following search space:

- Grid dimension: 1 / 2 / 3
- Multiple of window: 1 / 2 / 4
- Number of subsequent intervals: 1 / 2 / 4
- Number of idle vehicle at an interval: 1 / 2 / 4

On both microtransit and NYC taxi data, the MLP-based model performs better when using both features combined (see Table III).

b) KAN: For KAN, we optimize the number of KAN layers and the width of each KAN layer, considering the following search space:

- Number of KAN Layers: 1 / 2 layers
- Width of KAN Layers: 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9

During the hyperparameter search, we consider features based on the following two sets of features and their combinations $\boldsymbol{x}, \boldsymbol{x}_a$. For $\boldsymbol{x}_a$ and combination of $\boldsymbol{x}, \boldsymbol{x}_a$, we consider the following search space:

- Grid dimension: 1 / 2 / 3
- Multiple of window: 1 / 2 / 4
- Number of subsequent intervals: 1 / 2 / 4
- Number of idle vehicle at an interval: 1 / 2 / 4

On both microtransit and NYC taxi data, the MLP-based model performs better when using both features combined (see Table III).

c) CNN: For CNN, we consider two 1-dimensional convolution layers followed by a single layer of feed-forward network. We optimize the number of output channels for each layer, the kernel size, the number of neurons in the feed-forward layer, and the dropout rate, considering the following search space:

- Number of Output Channels: 4, 8, 16
- Kernel Size: 2, 3, 4
- Number of Neurons in the Feed-Forward Layer: 64 / 128 neurons
- Dropout Rate (after each hidden layer): 0.0 / 0.1

For CNN-based architecture, we consider a feature vector $\boldsymbol{x}_w$ that captures the fine-grained availability of vehicles for next h hours. In our experiments, we let $h = 4$ hours and $w = 30$ seconds, which results in a feature vector $\boldsymbol{x}$ of length 480.

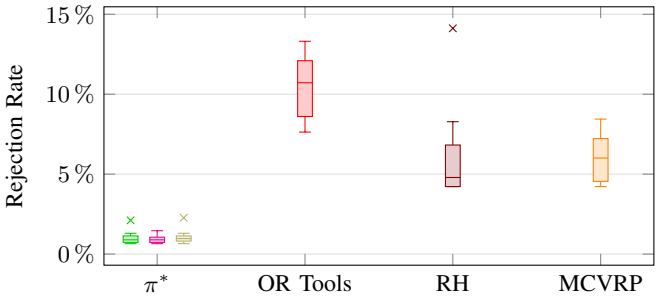

Fig. 5. Distribution of rejection rates for the trained policy π^* with neural-network architectures based on MLP (■), KAN (■), and CNN (■), across 5 different episodes from the **microtransit data**.

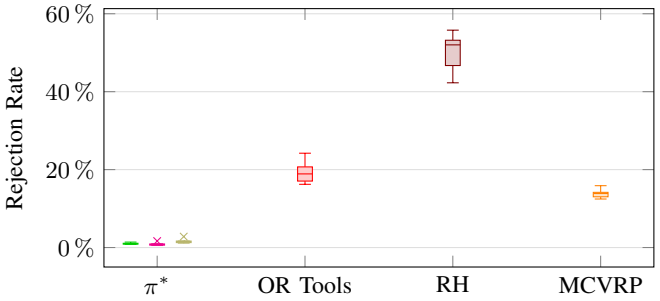

Fig. 6. Distribution of rejection rates for the trained policy π^* with neural-network architectures based on MLP (■), KAN (■), and CNN (■), across 5 different episodes from the **NYC taxi data**.

Table II shows the best hyperparameters for each of the neural-network architectures (i.e., MLP, KAN, and CNN) for both microtransit and NYC taxi data. Table III shows the best feature combination for each neural-network architecture (i.e., MLP, KAN) for both microtransit and NYC taxi data.

B. Rejection Rates with Various Neural-Network Architectures

Figure 5 shows the distribution of rejection rates for the trained policy π^* with various neural-network architectures (MLP, KAN and CNN), evaluated over 5 different episodes from the microtransit data. We observe that the MLP-based policy π^* performs slightly better than policies based on KAN and CNN. Further, all three trained policies outperform all of the baselines.

Figure 6 shows the distribution of rejection rates for the trained policy π^* with various neural-network architectures (MLP, KAN and CNN), evaluated over 5 different episodes from the NYC taxi data. We again observe that the MLP-based trained policy π^* performs slightly better than policies trained with KAN and CNN. Further, all three trained policies outperform all of the baselines.

C. Rejection Rate vs. Running Time for Anytime Algorithm with Various Neural-Network Architectures

Figures 7 and 8 show how the rejection rate depends on the running time of the anytime algorithm, on microtransit and NYC taxi data, respectively. The figures show results for

TABLE II
NEURAL-NETWORK HYPERPARAMETERS

Neural Network Architecture: MLP		
Name of the Hyperparameter	**Microtransit Data**	**NYC Taxi Data**
Number of Hidden Layers	3	2
Number of Neurons	[64, 128, 128]	[128, 128]
Dropout Rate	0.1	0.2
Neural Network Architecture: KAN		
Name of the Hyperparameter	**Microtransit Data**	**NYC Taxi Data**
Number of KAN Layers	2	2
Layer Widths	[9, 9]	[2, 1]
Neural Network Architecture: CNN		
Name of the Hyperparameter	**Microtransit Data**	**NYC Taxi Data**
Output Channels of 1^{st} Convolution Layer	8	8
Output Channels of 2^{nd} Convolution Layer	4	8
Kernel Size	3	3
Number of Neurons	64	128
Dropout Rate	0.1	0.1

TABLE III
FEATURE VECTOR COMBINATION

Feature Selection: MLP		
Name of the Parameter	**Microtransit Data**	**NYC Taxi Data**
Feature Combination	$\boldsymbol{x}, \boldsymbol{x}_a$	$\boldsymbol{x}, \boldsymbol{x}_a$
Grid dimension (g)	1	3
Multiple of window (N_{wm})	2	2
Number of subsequent intervals (N_{ci})	1	1
Number of idle vehicle at an interval (N_{mv})	1	1
Feature Selection: KAN		
Name of the Parameter	**Microtransit Data**	**NYC Taxi Data**
Feature Combination	$\boldsymbol{x}, \boldsymbol{x}_a$	$\boldsymbol{x}, \boldsymbol{x}_a$
Grid dimension (g)	3	3
Multiple of window (N_{wm})	2	2
Number of subsequent intervals (N_{ci})	1	1
Number of idle vehicle at an interval (N_{mv})	2	1

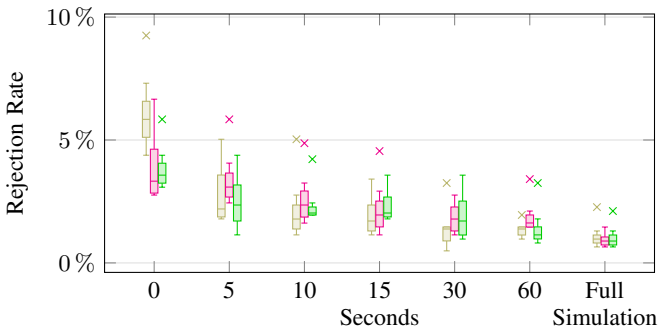

Fig. 7. Distribution of rejection rates for various running-time limits on the anytime algorithm (in seconds, horizontal axis) with trained policies π^* based on MLP (■), KAN (■), and CNN (■), across 5 different episodes from the **microtransit data**.

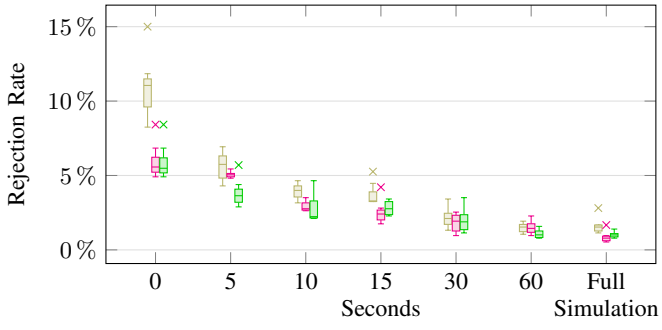

Fig. 8. Distribution of rejection rates for various running-time limits on the anytime algorithm (in seconds, horizontal axis) with trained policies π^* based on MLP (■), KAN (■), and CNN (■), across 5 different episodes from the **NYC taxi data**.

trained policies π^* based on various neural-network architectures (MLP, KAN, and CNN). The results of both figures demonstrate the importance of taking advantage of the time between request arrivals since the rejection rate decreases significantly as the running time increases, regardless of the dataset and the neural-network architecture.

D. Fairness of Distributing Requests between Route Plans

We assess the fairness of request distribution by analyzing the number of requests allocated to each driver (i.e., vehicle). Our findings indicate that the standard deviation of request

distribution is approximately 14 when using Google OR-Tools, π^0, and π^*, and around 13.5 with RH and MCVRP on the real-world microtransit dataset. Even though fairness is an important aspect, which ensures that all drivers are equally considered while serving requests, we primarily focus on maximizing the service rate in this work. As a result, requests may end up being distributed unevenly among drivers.

E. Ablation Study

We compared our complete solution approach (i.e., use the learned Q-function for choosing the insertion placement, accept or reject decisions, as well as performing continual optimization) against different variations such as:

A.) use the **simple heuristic objective** of maximizing idle time for the next 4 hours (i.e., policy π^0) when making the insertion decision, as well as during the continual optimization via an anytime algorithm.
B.) use the **simple heuristic objective** (i.e., π^0) to determine the insertion placement, while using the learned Q-function for making the insertion decision, as well as performing continual optimization.
C.) use the learned Q-function for choosing the insertion placement and the accept-or-reject decision, but use **simple heuristic objective** (i.e., π^0) to perform continual optimization, always accepting requests if a feasible insertion exists.
D.) use the learned Q-function for choosing the insertion placement and the accept-or-reject decision, but use **simple heuristic objective** (i.e., π^0) to perform continual optimization.
E.) use the learned Q-function for making the insertion decision, as well as performing continual optimization, but always accept requests if a feasible insertion exists.

Tables IV to VI show a comparison of the service rate using different ablation settings for the microtransit data, where the Q-function is learned using the neural network architectures of MLP, CNN, and KAN respectively. When using the Q-function learned with an MLP-based neural network architecture, the ablation setting "E" performs slightly better than our solution approach. Similarly, when using a CNN-based neural network architecture, the ablation setting "D" performs slightly better than our solution approach. On the other hand, when using a KAN-based neural network architecture, our proposed approach outperforms all other settings in the ablation study.

Tables VII to IX show a comparison of the service rate using different ablation settings for the NYC taxi data, where the Q-function is learned using MLP, CNN, and KAN neural network architectures, respectively. When using a Q-function learned with MLP or CNN based neural network architectures, the ablation setting "D" performs slightly better than our solution approach. On the other hand, when using the KAN-based neural network architecture, our proposed approach outperforms all other settings in the ablation study.

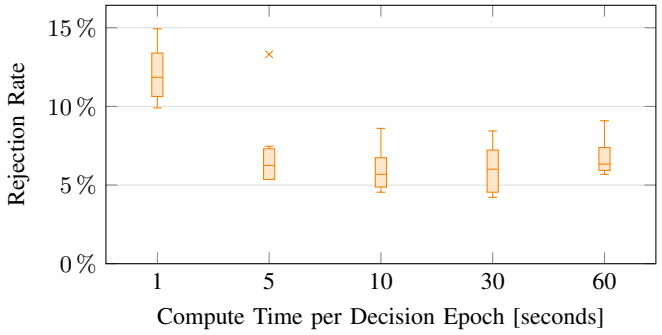

Fig. 9. Distribution of rejection rates when using MC VRP as the dynamic VRP solver with different running times per decision epoch, across 5 different episodes from the **microtransit data**.

F. Abstract Setting

In the abstract setting, we represent the city as a simple 2D square grid (2500 $\times$ 2500). For simplicity, we model the request arrivals as a stationary Poisson process with an average arrival rate of 20 requests per every 3600 units of time. We model the distribution of the time difference between request arrival and requested pick-up time as an exponential distribution with a mean of 7200 units in length. We consider a vehicle capacity of $c = 8$ and let the number of vehicles be $|\mathcal{V}| = 4$. We calculate travel times as the Euclidean distance between two location coordinates. For the evaluation, we consider each episode to last 12 hours. Accordingly, we examine 5 different episodes with a total of 240 requests. To ensure a fair comparison, we evaluate the proposed approach and ablation settings on the same 5 sets of requests.

Tables X and XI compares the service rate across different ablation settings for abstract problem setting, where the Q-function is learned using MLP- and CNN-based neural network architectures, respectively. Our proposed approach outperforms all other settings in the ablation study when using the learned Q-function at all the decision-making points. Specifically, the Q-function learned using CNN-based neural network achieves a 2.5% improvement over a simple heuristic policy (i.e., π^0) instead of learned Q-function (i.e., ablation setting "A").

G. Rejection Rate vs. Running Time for MC VRP

Figures 9 and 10 show how the rejection rate depends on the computation time at each decision epoch when using MC VRP as the dynamic VRP solver on the microtransit and NYC taxi datasets, respectively. We observe that as the time spent on each decision epoch increases, the rejection rate gradually decreases.

H. Rejection Rate vs. Computation Time for Rolling Horizon

Figures 11 and 12 show how the Rolling Horizon solver's rejection rate and computation time (per decision epoch) vary with the running time for RTV generation per batch on the microtransit data. While allowing the RH solver to perform RTV generation for 15 seconds per batch could achieve very

Settings	Insertion Decision	Acceptance Decision	Anytime Optimization	Service Rate
A	π^0	Always Accept (π^0)	π^0	98.64
B	π^0	Accept/Reject (π^*)	π^*	98.83
C	π^*	Always Accept (π^*)	π^0	98.70
D	π^*	Accept/Reject (π^*)	π^0	98.86
E	π^*	Always Accept (π^*)	π^*	**99.06**
Our	π^*	Accept/Reject (π^*)	π^*	98.83

TABLE IV

AVERAGE SERVICE RATE ACROSS 5 DIFFERENT EPISODES FROM **MICROTRANSIT DATA** EVALUATED USING THE DECISION MAKING CHOICES WHILE USING LEARNED Q-FUNCTION USING MLP BASED ARCHITECTURE.

Settings	Insertion Decision	Acceptance Decision	Anytime Optimization	Service Rate
A	π^0	Always Accept (π^0)	π^0	98.64
B	π^0	Accept/Reject (π^*)	π^*	98.25
C	π^*	Always Accept (π^*)	π^0	98.67
D	π^*	Accept/Reject (π^*)	π^0	**98.80**
E	π^*	Always Accept (π^*)	π^*	98.67
Our	π^*	Accept/Reject (π^*)	π^*	98.77

TABLE V

AVERAGE SERVICE RATE ACROSS 5 DIFFERENT EPISODES FROM **MICROTRANSIT DATA** EVALUATED USING THE DECISION MAKING CHOICES WHILE USING LEARNED Q-FUNCTION USING CNN BASED ARCHITECTURE.

Settings	Insertion Decision	Acceptance Decision	Anytime Optimization	Service Rate
A	π^0	Always Accept (π^0)	π^0	98.64
B	π^0	Accept/Reject (π^*)	π^*	98.44
C	π^*	Always Accept (π^*)	π^0	98.57
D	π^*	Accept/Reject (π^*)	π^0	98.80
E	π^*	Always Accept (π^*)	π^*	98.83
Our	π^*	Accept/Reject (π^*)	π^*	**98.99**

TABLE VI

AVERAGE SERVICE RATE ACROSS 5 DIFFERENT EPISODES FROM **MICROTRANSIT DATA** EVALUATED USING THE DECISION MAKING CHOICES WHILE USING LEARNED Q-FUNCTION USING KAN BASED ARCHITECTURE.

Settings	Insertion Decision	Acceptance Decision	Anytime Optimization	Service Rate
A	π^0	Always Accept (π^0)	π^0	98.70
B	π^0	Accept/Reject (π^*)	π^*	98.95
C	π^*	Always Accept (π^*)	π^0	99.00
D	π^*	Accept/Reject (π^*)	π^0	**99.29**
E	π^*	Always Accept (π^*)	π^*	98.84
Our	π^*	Accept/Reject (π^*)	π^*	98.93

TABLE VII

AVERAGE SERVICE RATE ACROSS 5 DIFFERENT EPISODES FROM **NYC TAXI DATA** EVALUATED USING THE DECISION MAKING CHOICES WHILE USING LEARNED Q-FUNCTION USING MLP BASED ARCHITECTURE.

Settings	Insertion Decision	Acceptance Decision	Anytime Optimization	Service Rate
A	π^0	Always Accept (π^0)	π^0	98.70
B	π^0	Accept/Reject (π^*)	π^*	98.47
C	π^*	Always Accept (π^*)	π^0	98.95
D	π^*	Accept/Reject (π^*)	π^0	**99.02**
E	π^*	Always Accept (π^*)	π^*	98.35
Our	π^*	Accept/Reject (π^*)	π^*	98.26

TABLE VIII

AVERAGE SERVICE RATE ACROSS 5 DIFFERENT EPISODES FROM **NYC TAXI DATA** EVALUATED USING THE DECISION MAKING CHOICES WHILE USING LEARNED Q-FUNCTION USING CNN BASED ARCHITECTURE.

low rejection rates, the overall computation time per decision epoch (i.e., computation time to perform RTV generation for all batches and to solve the ILP problem) varies between 85 to 225 seconds. Running the RH solver for this long to confirm an incoming request is unacceptable in practical applications that aim to provide prompt confirmation.

Settings	Insertion Decision	Acceptance Decision	Anytime Optimization	Service Rate
A	π^0	Always Accept (π^0)	π^0	98.70
B	π^0	Accept/Reject (π^*)	π^*	98.72
C	π^*	Always Accept (π^*)	π^0	99.00
D	π^*	Accept/Reject (π^*)	π^0	99.00
E	π^*	Always Accept (π^*)	π^*	98.84
Our	π^*	Accept/Reject (π^*)	π^*	**99.05**

TABLE IX

AVERAGE SERVICE RATE ACROSS 5 DIFFERENT EPISODES FROM **NYC TAXI DATA** EVALUATED USING THE DECISION MAKING CHOICES WHILE USING LEARNED Q-FUNCTION USING KAN BASED ARCHITECTURE.

Settings	Insertion Decision	Acceptance Decision	Anytime Optimization	Service Rate
A	π^0	Always Accept (π^0)	π^0	56.67
B	π^0	Accept/Reject (π^*)	π^*	57.91
C	π^*	Always Accept (π^*)	π^0	57.17
D	π^*	Accept/Reject (π^*)	π^0	56.83
E	π^*	Always Accept (π^*)	π^*	58.17
Our	π^*	Accept/Reject (π^*)	π^*	**58.33**

TABLE X

AVERAGE SERVICE RATE ACROSS 5 DIFFERENT EPISODES FROM **ABSTRACT** ENVIRONMENT EVALUATED USING THE DECISION MAKING CHOICES WHILE USING LEARNED Q FUNCTION USING MLP BASED ARCHITECTURE.

Settings	Insertion Decision	Acceptance Decision	Anytime Optimization	Service Rate
A	π^0	Always Accept (π^0)	π^0	56.67
B	π^0	Accept/Reject (π^*)	π^*	59.08
C	π^*	Always Accept (π^*)	π^0	56.84
D	π^*	Accept/Reject (π^*)	π^0	57.92
E	π^*	Always Accept (π^*)	π^*	58.00
Our	π^*	Accept/Reject (π^*)	π^*	**59.17**

TABLE XI

AVERAGE SERVICE RATE ACROSS 5 DIFFERENT EPISODES FROM **ABSTRACT** ENVIRONMENT EVALUATED USING THE DECISION MAKING CHOICES WHILE USING LEARNED Q FUNCTION USING CNN BASED ARCHITECTURE.

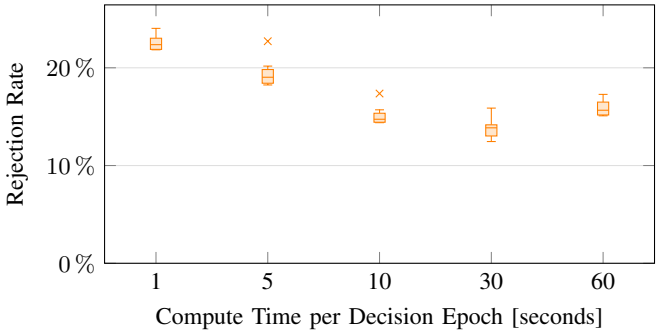

Fig. 10. Distribution of rejection rates when using MC VRP as the dynamic VRP solver with different running times per decision epoch, across 5 different episodes from the **NYC taxi data**.

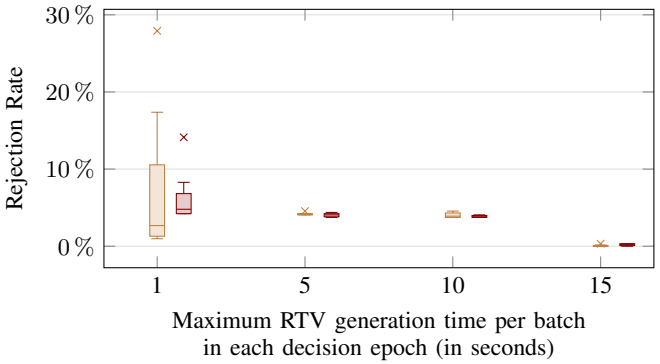

Fig. 11. Distribution of rejection rates when using Rolling Horizon solver at each decision epoch with rolling-horizon factors RH1 ( ) and RH2 ( ), across 5 different episodes from the **microtransit data**.

Figures 13 and 14 show how the Rolling Horizon solver's rejection rate and computation time (per decision epoch) vary with the running time for RTV generation per batch on the NYC taxi data. While spending more time on RTV generation per batch is able to reduce the rejection rate from 60% to 40%, the computation time increases drastically from 50 to 130 seconds per decision epoch. Therefore, our proposed approach provides better performance both in terms of acceptance rate and in terms of time required to confirm a request.

I. Statistical Tests of Significance

Finally, we present the results of statistical tests on the significance of our experimental results shown by Figures 2 and 3. We performed paired two-sample permutation tests comparing the service rates of our proposed approach to the service rates of each baseline approach on both datasets, based

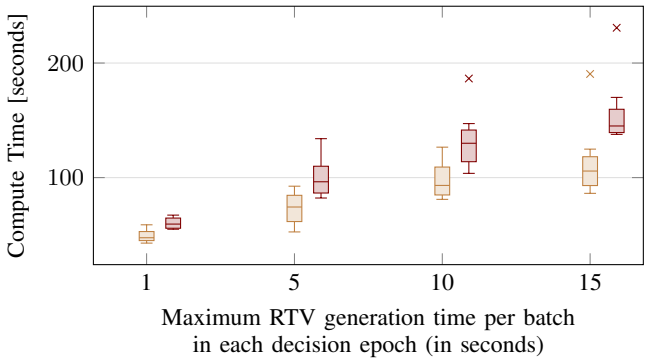

Fig. 12. Distribution of average computation time for each decision epoch when using Rolling Horizon solver with rolling-horizon factors RH1 (■) and RH2 (■), across 5 different episodes from the **microtransit data**.

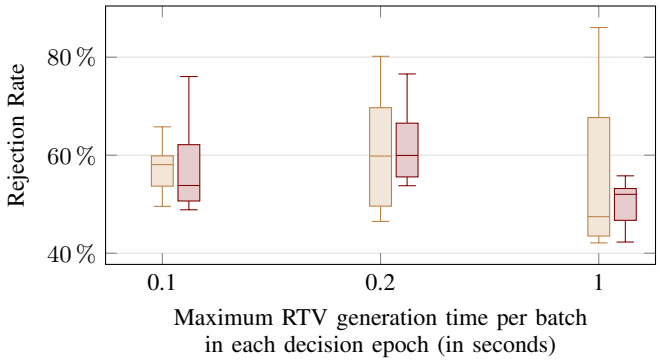

Fig. 13. Distribution of rejection rates when using Rolling Horizon solver at each decision epoch with rolling-horizon factors RH1 (■) and RH2 (■), across 5 different episodes from the **NYC taxi data**.

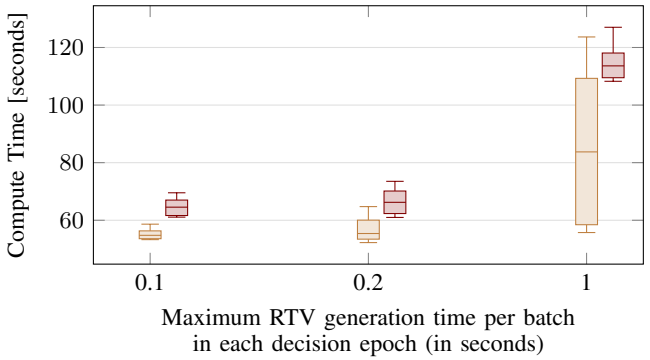

Fig. 14. Distribution of average computation time for each decision epoch when using Rolling Horizon solver with rolling-horizon factors RH1 (■) and RH2 (■), across 5 different episodes from the **NYC taxi data**.

on 5 different episodes in each case. We observe that for both microtransit data and NYC data, the service rates obtained using our trained policy π^* are significantly better than those of the baselines when we consider the significance threshold for the p-value to be 5%.