Accelerating High-Capacity Ridepooling in Robo-Taxi Systems

Xinling Li[1], Daniele Gammelli[2], Alex Wallar[3], Jinhua Zhao[4], and Gioele Zardini[1]

***Abstract*—Rapid urbanization has increased demand for customized urban mobility, making on-demand services and robo-taxis central to future transportation. The efficiency of these systems hinges on real-time fleet coordination algorithms. This work accelerates the state-of-the-art high-capacity ridepooling framework by identifying its computational bottlenecks and introducing two complementary strategies: (i) a data-driven feasibility predictor that filters low-potential trips, and (ii) a graph-partitioning scheme that enables parallelizable trip generation. Using real-world Manhattan demand data, we show that the acceleration algorithms reduce the optimality gap by up to 27% under real-time constraints and cut empty travel time by up to 5%. These improvements translate into tangible economic and environmental benefits, advancing the scalability of high-capacity robo-taxi operations in dense urban settings.**

***Index Terms*—Robo-Taxi, Ridepooling, Fleet coordination**

I. Introduction

By 2050, 68% of the global population is projected to live in urban areas [1]. This rapid urbanization is driving demand for flexible, customized mobility solutions, while worsening congestion and greenhouse gas emissions. On-demand mobility services offer a promising alternative to private vehicle ownership, combining flexibility with the potential to easy pressure on infrastructure. Their success, however, depends on effective large-scale vehicle coordination: without it, the inherent spatial and temporal imbalance of travel demand leads to excessive empty trips, eroding efficiency and sustainability gains [2]. Autonomous Mobility-on-Demand (AMoD) systems, also referred to as robo-taxi services, address this challenge via centralized fleet control enabled by algorithmic coordination [3]. At the core lies the vehicle-to-request(s) assignment problem. Yet, most existing work has focused on *single-passenger* assignments, which simplify the optimization but increase inefficiencies such as empty repositioning. Moreover, single-passenger services often compete with mass transit, undermining sustainability objectives and worsening congestion in the long term [4]. Ridepooling, the practice of matching and merging multiple passenger trips, offers a compelling alternative. By enabling shared rides, AMoD fleets can reduce fleet size, cut empty mileage, and mitigate congestion and emissions. Yet, despite such benefits, high-capacity ridepooling remains underexplored, primarily due to its combinatorial complexity. Even single-passenger assignments are NP-hard, and allowing shared rides adds further structure that makes real-time fleet control more challenging. The efficiency of ridepooling in AMoD systems therefore depends critically on the scalability and real-time performance of assignment algorithms. The state-of-the-art algorithm of [5] addresses this complexity by leveraging shareability networks, retaining optimality while improving tractability. However, despite its practical success, it still struggles to deliver high-quality assignments under strict real-time constraints, where the strict real-time constraints are required by the tight limits on the computation time available to construct routes and confirm assignments in response to live requests. Improving its computational efficiency is therefore essential, but its bottlenecks remain poorly understood and strategies for improvement largely unexplored.

Statement of contribution: This letter makes four key contributions. (1) We formally reformulate the high-capacity ridepooling algorithm of [5], clarifying its structure and systematically identifying its computational bottlenecks. (2) Building on this analysis, we propose two accelerations: a data-driven feasibility predictor that filters low-potential trips before optimization, and a shareability graph partitioning scheme that enables parallelization and reduces problem size. (3) We validate these strategies using large-scale, real-world data from Manhattan, NYC, demonstrating substantial real-time gains, including reductions in optimality gap up to 27% and empty travel time of up to 5%. (4) We distill practical insights on when to deploy each strategy, offering actionable guidance for scalable, high-capacity ridepooling.

The remainder of the paper reviews related work (Section II), formulates the baseline (Section III), introduces our accelerations (Section IV), evaluates them (Section V), and concludes with a discussion of future directions (Section VI).

II. Related work

The vehicle-passenger assignment problem has received significant attention with the rise of mobility-on-demand services. Early work focused on the single-passenger setting, where a vehicle serves one passenger at a time, as in traditional taxis. The problem has been extensively studied via optimization-based model predictive control [6], [7], learning [8], [9], and hybrid optimization-learning frameworks [10]–[13], offering different trade-offs between computational efficiency and solution

Manuscript received: August 27, 2025; Revised: December 23, 2025; Accepted:

This paper was recommended for publication by Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers' comments.

This work is supported by the US Department of Energy/Office of Energy Efficiency and Renewable Energy, Grant Agreement DE-EE0011186, and by the Sidara Urban Seed Grant Program at the Norman B. Leventhal Center for Advanced Urbanism, Massachusetts Institute of Technology.

[1]Laboratory for Information & Decision Systems, Massachusetts Institute of Technology {xinli831,gzardini}@mit.edu

[2]Department of Aeronautics and Astronautics, Stanford University gammelli@stanford.edu

[3]The Routing Company alex@theroutingcompany.com

[4]Department of Urban Studies and Planning, Massachusetts Institute of Technology jinhua@mit.edu

Digital Object Identifier (DOI): see top of this page.

quality. Studies of Mobility-on-Demand (MoD) systems have shown that such services can increase congestion and induce longer empty travel miles due to cruising [14]. This motivated the study of *ridepooling*, where rides are shared to improve utilization. Most work has focused on *low-capacity* ridepooling, typically two-passenger trips [15]–[17], where formulations are only marginally more complex than single-passenger models [18]. While such restrictions simplify optimization, they limit the efficiency gains achievable through higher degrees of sharing [5].

High-capacity ridepooling can deliver larger efficiency and sustainability benefits but introduces substantial combinatorial complexity. Even for small fleet sizes, real-time optimization remains challenging. Existing approaches address this trade-off either by integrating learning-based components to accelerate optimization while preserving feasibility [19], [20], or by using heuristics that simplify sharing scenarios at the cost of generality [21], [22]. The leading high-capacity approach is [5], which leverages shareability networks [23] to prune the search space while retaining optimality. This method has been widely adopted in practice and extended to non-myopic coordination [19], simplified single-assignment [22], and multi-objective formulations [24]. However, despite its success, it remains difficult to deploy at scale under strict real-time constraints, and little work has targeted its efficiency.

In the remainder of the paper, we refer to [5] as the "baseline". While our analysis centers on ridepooling, the proposed ideas apply broadly to graph-based resource allocation problems that require solving many small optimization instances, including, e.g., robotics,and smart grids.

III. Problem formulation and baseline

We formulate the dynamic high-capacity ridepooling problem, and make its structural elements explicit, enabling a systematic analysis of computational bottlenecks and the development of targeted acceleration strategies.

a) Network, fleet, and requests: We represent the on-demand mobility network as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, t, o, d)$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of directed edges. The maps $o : \mathcal{E} \to \mathcal{V}$ and $d : \mathcal{E} \to \mathcal{V}$ assign an edge to its source and sink nodes, respectively. This graph represents a driving network where only one edge $e \in \mathcal{E}$ exists from $o(e)$ to $d(e)$. The map $t : \mathcal{E} \times \mathcal{E} \to \mathbb{R}_{\geq 0}$ maps each edge e to a corresponding travel time $t(e)$. Let $t_{\min} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$ denote a function that returns the shortest travel time between a pair of nodes on $\mathcal{G}$.[1] We assume that $\mathcal{G}$ is strongly connected.

Let the fleet be given by $\mathcal{N} = \{n_i\}_{i=1}^{N}$, and a capacity map $c : \mathcal{N} \to \mathbb{N}$ assign each vehicle to its capacity. Let $\mathcal{R}$ denote the set of requests, where each $r \in \mathcal{R}$ is a tuple $(t_r, o_r, d_r, t_w^r, t_d^r, \alpha)$, where $t_r \in \mathbb{R}_{\geq 0}$ is the request time, $o_r, d_r \in \mathcal{V}$ are the pickup and dropoff locations with $r_o \neq r_d$, $t_w^r \in \mathbb{R}_{\geq 0}$ is the maximum allowable pickup waiting time, $t_d^r \in \mathbb{R}_{\geq 0}$ is the maximum allowable dropoff delay time, and α is the number of passengers requesting this particular ride. The corresponding pickup and dropoff time windows are: $w_p^r = [t_r, t_r + t_w^r]$ and $w_d^r = [t_r + t_{\min}(r_o, r_d), t_r + t_{\min}(r_o, r_d) + t_d^r]$. Given any requests with the same attributes, i.e., $r_1 = (t, o, d, t_w, t_d, \alpha_1)$ and $r_2 = (t, o, d, t_w, t_d, \alpha_2)$, one can aggregate them into $r = (t, o, d, t_w, t_d, \alpha_1 + \alpha_2)$.

b) Discretization and state maps: Fleet $\mathcal{N}$ serves requests $\mathcal{R}$ over a time horizon $[t_{\text{start}}, t_{\text{end}}]$, discretized into T windows $t_1, \ldots, t_T$ of equal length $t_{\text{fix}} > 0$, such that $t_{\text{end}} = t_{\text{start}} + T \cdot t_{\text{fix}}$. Each discrete time index t_i corresponds to the time interval $[t_{\text{start}} + t_{i-1} \cdot t_{\text{fix}}, t_{\text{start}} + t_i \cdot t_{\text{fix}}]$. We denote the discrete time window by t and continuous time by $\tilde{t} \in [t_{\text{start}}, t_{\text{end}}]$. We define the following maps to track requests and fleet status over time. The active request aggregator $u\colon \{t_1, \ldots, t_T\} \to \mathcal{P}(\mathcal{R})$, $t_i \mapsto \{r \in \mathcal{R} \mid t_{\text{start}} + t_i \cdot t_{\text{fix}} < t_r + t_w^r\}$, returns the requests that are waiting for service at time window t_i. Further, the vehicle state maps $s\colon \mathcal{N} \times [t_{\text{start}}, t_{\text{end}}] \to \mathcal{V} \times \mathbb{R}_{\geq 0}$, which maps each vehicle to its location at a given time, indicated by the node to which it is traveling and the time remaining to reach that node, and $b\colon \mathcal{N} \times [t_{\text{start}}, t_{\text{end}}] \to \mathcal{P}(\mathcal{R})$, which maps each vehicle to requests onboard at any time.

c) Trips and feasibility: To formally present the assignment problem, we first introduce the notion of feasible trip.

Definition 1 (Trip)**.** A *trip* is a tuple $\tau = (n, l, \mathcal{R}^\tau)$, where $n \in \mathcal{N}$ is a vehicle, l is a finite, ordered sequence of visited nodes $(v_1, \ldots, v_K)$ on $\mathcal{G}$, and $\mathcal{R}^\tau \subseteq \mathcal{R}$ is the set of new request served in the trip. A trip may also drop off existing onboard requests, while not exceeding the vehicle's capacity $c(n)$ at any point along l.

Definition 2 (Feasible trip)**.** A trip $\tau = (n, l, \mathcal{R}^\tau)$ is *feasible* if: i) Each $r \in \mathcal{R}^\tau$ is picked up at o_r within w_p^r and dropped off at d_r within w_d^r, ii) Each onboard request r is dropped off at d_r within w_d^r.

Let $t_{\min}^m : \mathcal{P}(\mathcal{R}) \times \mathcal{N} \to \mathbb{R}_{\geq 0}$ return the minimal travel time of any feasible trip serving given requests with a given vehicle.

d) Dynamic assignment problem:

Problem 1 (Assignment Problem)**.** At time window t, given fleet $\mathcal{N}$ and active requests $u(t)$, one represents *assignment* as an optimization problem with the following decision variables:

- Assignment variables: $x_{r,n} \in \{0, 1\},\ \forall r \in \mathcal{R}, n \in \mathcal{N}$, indicating if request r is assigned to vehicle n;
- Feasible trip variables: $\tau_n = (n, l, \mathcal{R}^{\tau_n}),\ \forall n \in \mathcal{N}$, deciding a feasible trip for each vehicle n.

Constraints include $\sum_{n \in \mathcal{N}} x_{r,n} \leq 1, \forall r \in u(t)$, ensuring that each request is served by *at most* one vehicle, and $\mathcal{R}^{\tau_n} = \{r | x_{r,n} = 1\}, \forall n \in \mathcal{N}$, ensuring that the feasible trip for a vehicle includes all its assigned requests. The objective is:

$$\min_{x, \mathcal{T}} \quad M \cdot \sum_{r \in u(t)} \sum_{n \in \mathcal{N}} x_{r,n} + \sum_{n} t_{\min}^m(R^{\tau_n}, n).$$

The problem is solved for each of the T time windows on a rolling horizon [25]. It is known to be NP-hard [26].

e) Baseline algorithm: Fig. 1 illustrates the baseline algorithm for the assignment problem. In [5], a subsequent rebalancing step uses a simple heuristic to allocate idle vehicles to unserved requests. While effective, the computational bottleneck lies in the assignment optimization itself. Since our focus is on improving this stage, we omit rebalancing here, though it can be incorporated as a final step once assignments are determined. The baseline algorithm improves tractability

[1] This is equivalent to solving a single vehicle routing problem on $\mathcal{G}$.

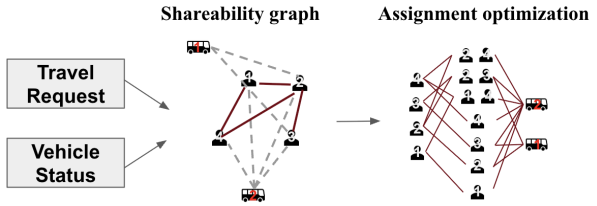

Fig. 1: Assignment algorithm for each decision window.

by reducing the search space (Fig. 1). We next introduce key concepts underpinning its operation.

Definition 3 (Shareable requests)**.** At time $\tilde{t}$, requests $r, p \in \mathcal{R}$ are *shareable* if a feasible trip $(n, l, \{r, p\})$ exists for n such that: i) $b(n, \tilde{t}) = \emptyset$, ii) $s(n, \tilde{t}) \in \{(r_0, 0), (p_o, 0)\}$.

Definition 4 (Shareability graph)**.** The *shareability graph* is an undirected graph $\mathcal{S} = (\mathcal{V}^s, \mathcal{E}^s, m)$ with node set $\mathcal{V}^s$, edge set $\mathcal{E}^s$, and a map $m\colon \mathcal{V}^s \to \mathcal{N} \cup \mathcal{R}$ that maps each node to a vehicle or request. An edge $(x, y) \in \mathcal{E}^s$ exists if one of the following conditions holds: a) $m(x), m(y) \in \mathcal{R}$, and the two requests $m(x)$ and $m(y)$ are shareable; b) $m(x) \in \mathcal{N}, m(y) \in \mathcal{R}$, and a feasible trip exists for $m(x)$ serving $m(y)$. These conditions provide a constructive way to build a shareability graph given sets of vehicles and requests.

Definition 5 (Clique)**.** A *clique* on a shareability graph $\mathcal{S}$ is a fully connected subgraph $\mathcal{S}' = \left(\mathcal{V}^{s'}, \mathcal{E}^{s'}, m\right)$ such that: a) $\mathcal{S}' \in \mathcal{S}$, b) $|\{x \in \mathcal{V}^{s'} : m(x) \in \mathcal{N}\}| = 1$, and c) $|\{y \in \mathcal{V}^{s'} : m(y) \in \mathcal{R}\}| \geq 1$. Conditions b) and c) indicate that a clique comprises one vehicle node and at least one request node.

At each time window t, the baseline algorithm solves the assignment problem with the following procedure:
(1) Get vehicle set $\mathcal{N}$ and request set $u(t)$. Construct a shareability graph $\mathcal{S}$ based on them.
(2) Enumerate all cliques on $\mathcal{S}$ and keep track of the ones that give feasible trips. Denote feasible trips as $\mathcal{T} = \{\tau_1, ..., \tau_P\}$.
(3) Formulate and solve a global assignment problem as a Integer Linear Program (ILP) based on $\mathcal{T}$. Take a feasible trip $\tau_p = (n_p, l_p, \mathcal{R}^p)$. Define indicator functions $\mathbf{1}^n(n', \tau_p) = 1$ if $n' = n_p$, and $\mathbf{1}^r(i, \tau_p) = 1$ if $i \in \mathcal{R}^p$. Decision variable $x_i \in \{0, 1\}$ indicates whether passenger i is assigned to a feasible trip, and $y_p \in \{0, 1\}$ whether a feasible trip τ_p is assigned to its corresponding vehicle n_p. Following Problem 1, the constraints of the ILP are:

$$\sum_{\tau_p \in \mathcal{T}} y_p \cdot \mathbf{1}^n(n', \tau_p) \leq 1 \quad n' \in \mathcal{N}, \tag{1}$$

$$\sum_{\tau_p \in \mathcal{T}} y_p \cdot \mathbf{1}^r(i, \tau_p) \leq x_i \quad i \in \mathcal{R}. \tag{2}$$

Eq. (1) says that one vehicle can serve at most one feasible trip, and Eq. (2) says that a passenger can be served by at most one vehicle. The objective is as in Problem 1.

IV. Methodology

We first leverage the formulation in Section III to reveal the computational bottlenecks of the baseline algorithm. We then introduce two targeted acceleration strategies, one data-driven, one graph-structural, designed to improve real-time performance without sacrificing assignment quality.

A. Baseline analysis and bottleneck identification

Although not explicitly mentioned in [5], the baseline approach is analogous to the concept of column generation and cutting plane algorithms in integer optimization [27].

a) Column generation analogy: The assignment optimization step in the baseline can be viewed as a set-partitioning/packing formulation [28] whose set elements are requests. Let $\mathcal{T}$ be the set of all feasible trips obtained from clique enumeration. Define a binary incidence matrix $\boldsymbol{A}$ with one row per vehicle $n \in \mathcal{N}$ and one row per request $r \in \mathcal{R}$; each column $j \in \mathcal{T}$ corresponds to a feasible trip τ_j. Set $a_{ij} = 1$ if row i is a vehicle n and τ_j uses n or if row i is a request r and $r \in \mathcal{R}^{\tau_j}$, and $a_{ij} = 0$ otherwise. With binary variables y_j selecting feasible trips, the master problem takes the form $\min \sum_{j \in \mathcal{T}} c_j y_j$ s.t. $\boldsymbol{A}y \leq b,\ y \in \{0, 1\}^{|\mathcal{T}|}$, where $b_i = 1$ enforces constraints in Problem 1, and c_j aggregates the objectives. In classical column generation, one solves a restricted master problem with a subset $\mathcal{T}_0 \subset \mathcal{T}$, derives dual prices, and then solves a pricing problem, here a VRP with time windows instance with dual-weighted costs, to find a trip $\tau \in \mathcal{T} \backslash \mathcal{T}_0$ improving columns. The process repeats until no improving column exists. By contrast, the baseline fully enumerates $\mathcal{T}$ via clique enumeration and feasibility checks before solving the master ILP. This generates-all-then-optimize approach creates the bottleneck: exhaustive upfront column generation incurs high computation.

b) Cutting plane analogy: Enumerating all vehicles-passenger combinations is intractable. The baseline mitigates this with a shareability graph (Definition 4), which encodes pairwise feasibility and filters infeasible combinations before trip generation. This is analogous to cutting plane methods in integer optimization, where valid inequalities iteratively shrink feasible regions. Here, the absence of an edge serves as an implicit cut, forbidding certain pairs from appearing in the same trip and drastically reducing the search space.

c) Bottlenecks: The baseline pipeline has two stages. First, the shareability graph is constructed in parallel over active vehicles and requests: for each vehicle-request (n, r) or request-request (r_1, r_2) pair, one solves a constant-size routing problem. Since each edge cost is bounded and independent, this step scales linearly with parallelization. Second, feasible trip generation checks each clique $\mathcal{S}'$ with $|\mathcal{V}^{s'}| = 1$ by solving a small capacity-constraint Vehicle Routing Problem (VRP) with time windows. While individual instances are small, the number of cliques grows combinatorially with graph density, $\mathcal{O}(2^d)$ for average degree d. The baseline enumerates cliques sequentially by size to reduce exploration, but clique counts remain large and this step dominates runtime. The solid arrows in Fig. 2 illustrate the process of the latter: each edge added to the assignment graph requires solving a VRP for the corresponding candidate trip. The exponential growth of candidate cliques, combined with the inherently sequential enumeration step, makes this stage the computational bottleneck of the baseline.

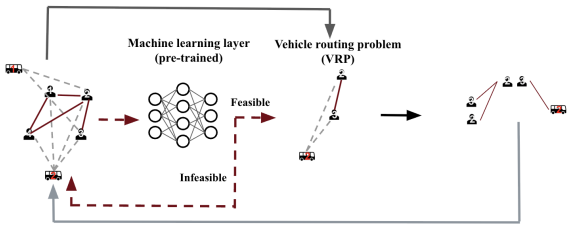

Fig. 2: Building assignment ILP from shareability graph. The iteration ends after all cliques have been iterated over. Solid arrows: baseline. Dashed arrows: the data-driven step.

We propose two accelerations: i) data-driven feasibility screening to discard low-quality cliques before optimization, and ii) graph partitioning to parallelize the feasibility checks.

B. Data-driven ILP construction

To accelerate the construction of the assignment ILP, we introduce a data-driven *feasibility prediction layer* that precedes the evaluation of each VRP. The accelerated algorithm is reported in Algorithm 1, and a schema in Figure 2.

a) Representation: Given a clique c containing exactly one vehicle n and a set of requests $\mathcal{R}_c$, we build a *tour graph* $\mathcal{G}_{\text{tour}}(c) = (\mathcal{V}_{\text{tour}}, \mathcal{E}_{\text{tour}})$, where every node can be mapped to types $\{\text{PU}(r), \text{DO}(r)\}_{r\in\mathcal{R}_c}$ (i.e., pickup and dropoff), and every vehicle to its location node $\text{VO}(n)$. The edges of this graph encode admissible precedence/transition constraints, including i) $\text{VO}(n) \to \text{PU}(r)$ for all $r \in \mathcal{R}_c$ (i.e., vehicle picking up request), ii) $\text{PU}(r) \to \text{DO}(r)$ (i.e., pickup must precede dropoff), and iii) $\text{DO}(r) \to \text{PU}(r')$ (i.e., serving r' immediately after dropping off r) and $\text{PU}(r') \to \text{DO}(r)$ (i.e., picking up r' while r is still onboard).

We will consider node features such as the type of node (i.e., VO, PU, or DO), the rate of request α, and time-window slack, and edge features such as shortest-path travel time and time-window slack of arriving at the sink node. This graph representation "flattens" the clique so that hard constraints (i.e., precedence, time windows, capacity) become explicit features on nodes and edges.

b) Predictor: A neural predictor $\hat{f}_\theta(c) \in [0,1]$ estimates $\Pr[c \text{ is feasible under the VRP} \mid \mathcal{G}_{\text{tour}}(c)]$ (i.e., the probability of the clique being feasible given a tour graph). We leverage a Structure2Vect-style [29] message passing on $\mathcal{G}_{\text{tour}}(c)$, where the node embedding update μ_v^{l+1} is:

$$\text{ReLU}\left(\theta_1 x_v + \theta_2 \sum_{u\in\text{ne}(v)} \mu_u^{(l)} + \theta_3 \sum_{u\in\text{ne}(v)} \text{ReLU}(\theta_4 \omega(v,u))\right)$$

where x_v is the original feature of node v, $\text{ne}(v)$ denotes the neighbors of node v, and $\theta_{1,2,3,4}$ are weights parameterized by neural networks. We then pool $\{\mu_v^{(L)}\}$ and pass through a sigmoid to obtain $\hat{f}_\theta(c)$. If the model is trained with class-balanced data and subsequently calibrated (e.g., via Platt scaling [30]), then $\hat{f}_\theta(c)$ can be directly interpreted as a calibrated probability of feasibility, making $\hat{f}_\theta(c)$ interpretable in probabilistic terms. One chooses a feasibility threshold β and only examines cliques with $\hat{f}_\theta(c) \geq \beta$. In practice, one can also perform stochastic selection: for each clique c, draw $\beta \sim \text{Uniform}(0,1)$ and proceed with exact VRP feasibility checking only if $\beta \leq \hat{f}_\theta(c)$. Under calibration, this means that a clique with predicted feasibility α will be checked with probability α. This method is reported in Algorithm 1, merging clique flattening and feasibility prediction. This process, along with the neural network, is illustrated in Fig. 3.

Algorithm 1 Data-Driven ILP Construction

Require: Shareability graph $\mathcal{S}$; prediction model for trip feasibility
Trips $\leftarrow \emptyset$
for $i = 1, \ldots, k$ **do**
  Round $\leftarrow \emptyset$
  // Union feasible trips of size $i-1$
  $C \leftarrow$ GetPotentialFeasible$(i, \mathcal{S})$
  for $c \in C$ **do**
    $p \leftarrow \hat{f}_\theta(c)$
    $\beta \sim$ Uniform$(0,1)$
    if $\beta \leq p$ **then**
      feasible $\leftarrow$ Routing(c)
      if feasible **then**
        Round $\leftarrow$ Round $\cup \{c\}$
      end if
    end if
  end for
  Trips $\leftarrow$ Trips $\cup$ Round
end for
AssignmentILP $\leftarrow$ BuildILP(Trips)

c) Properties: As we will show in the case studies, while simple in principle, this filter can be very effective. We make explicit what it guarantees, and what it does not. Let $\mathcal{C}$ be the set of enumerated cliques, and $\mathcal{F} \subseteq \mathcal{C}$ the set of truly VRP-feasible cliques. Let $\mathcal{F}_\beta = \{c \in \mathcal{C} \mid (\hat{f}_\theta(c) \geq \beta) \wedge (c \text{ is feasible})\}$ be the feasible set that survives filtering.

Lemma 1 (Preservation of feasibility)**.** The filtering process is sound, i.e., filtering never introduces infeasible trips.

Proof: Every candidate that passes the predictor still undergoes exact VRP checking. Thus, $\mathcal{F}_\beta \subseteq \mathcal{F}$. ■

Lemma 2 (Preservation of optimality)**.** Let $\mathcal{F}^\star \subseteq \mathcal{F}$ denote the set of trips used by an optimal solution to the baseline ILP (at most $|\mathcal{N}|$ columns). If the predictor has no false negatives on $\mathcal{F}^\star$ at threshold β (i.e., $\mathcal{F}^\star \subseteq \{c : \hat{f}_\theta(c) \geq \beta\}$, high recall condition), then the optimal solution of the filtered ILP equals that of the baseline ILP.

Lemma 2 relies on the assumption that the predictor produces no false negatives, which ensures that feasible trips are never excluded from consideration. In practice, this assumption may be mildly violated, as learned predictors can exhibit a low but nonzero false-negative rate.

Lemma 3 (Probabilistic preservation of the baseline optimum)**.** Let $\mathcal{F}$ be the set of feasible trips for the baseline ILP, and $\mathcal{F}^\star \subseteq \mathcal{F}$ the subset of trips in a baseline-optimal solution. For each $f \in \mathcal{F}$, define $E_f =$ "filter keeps f", and let $\rho_f = \Pr(E_f) \in [0,1]$. Use $\star = \Pr(\text{optimum preserved})$. a) Assuming no independence, we have $\star \geq \max\{0, 1 - \sum_{f\in\mathcal{F}^\star}(1-\rho_f)\}$; b) If $\rho_{\min} = \min_{f\in\mathcal{F}^\star}\rho_f$, then $\star \geq \max\{0, 1 - |\mathcal{F}^\star|(1-\rho_{\min})\} \geq \max\{0, 1 - |\mathcal{N}|(1-\rho_{\min})\}$; c) If $\{E_f\}$ independent, then $\star = \prod_{f\in\mathcal{F}^\star}\rho_f \geq \rho_{\min}^{|\mathcal{F}^\star|} \geq \rho_{\min}^{|\mathcal{N}|}$.

Proof: The optimum is preserved if and only if all trips in $\mathcal{F}^\star$ survive the filter, i.e., if $\bigcap_{f\in\mathcal{F}^\star} E_f$ occurs. By the

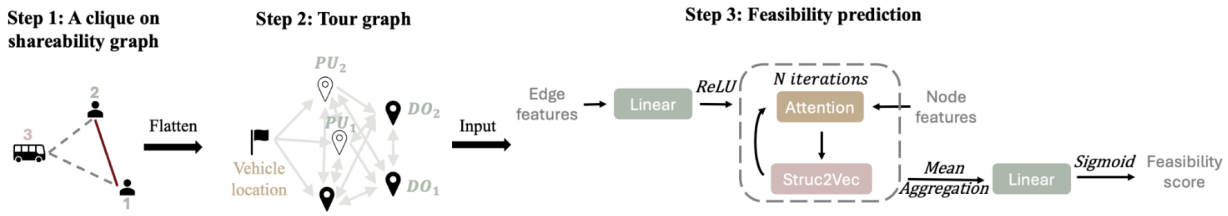

Fig. 3: Feasibility prediction in the data-driven ILP. (1) A clique is extracted from the shareability network. (2) Passenger nodes are expanded into pickup (PU) and dropoff (DO) nodes in a tour graph, where the vehicle connects only to the onboard passenger's dropoff and new PU nodes. (3) The tour graph is passed to the feasibility prediction module.

complement rule and Boole's inequality (union bound):

$$\Pr\left(\bigcap_f E_f\right) = 1 - \Pr\left(\bigcup_f E_f^c\right) \geq 1 - \sum_f \Pr(E_f^c) = 1 - \sum_f (1 - \rho_f).$$

Clipping to the interval $[0, 1]$ yields the first inequality, $\rho_f \geq \rho_{\min}$ the second, and independence gives third. ■

d) Practical aspects: First, if $\hat{f}_\theta$ is calibrated, β trades computation for optimality probability (Lemma 3). One can target a high-recall operating point to protect optimal columns, then adjust β to meet the real-time computational budget. Second, to protect against correlated false negatives, one can evaluate a small random fraction ϵ of filtered-out cliques with the exact VRP. Finally, the filter is order-agnostic for cliques with the same size; one can greedily prioritize cliques by $\hat{f}_\theta$ (and/or size) and stop when the time budget is exhausted, yielding a consistent anytime variant.

C. Shareability graph partition

We now address the clique-to-VRP bottleneck by reducing the size and density of the search space and enabling parallelization. We rely on partitioning the shareability graph into smaller, disjoint subgraphs on which feasible trip generation can be carried out *independently* and in parallel, before re-aggregating the results into a unified global ILP (Figure 4).

a) Partitioned construction: Let $\mathcal{S} = (\mathcal{V}^s, \mathcal{E}^s, m)$ be the shareability graph (Definition 4). Let $\mathcal{V}^{\text{req}} = \{v \in \mathcal{V}^s : m(v) \in \mathcal{R}\}$, and $\mathcal{V}^{\text{veh}} = \{v \in \mathcal{V}^s : m(v) \in \mathcal{N}\}$. Let $\mathcal{G}_{\text{RR}}$ be the request-request induced subgraph. We compute a K-way partition $P = \{P_1, \ldots, P_K\}$, $\bigcup_{k=1}^K P_k = \mathcal{V}^{\text{req}}$, with $P_i \cap P_j = \emptyset$ for $i \neq j$. For each block P_k we reattach all vehicles in the parallel computing process. Specifically, we build an induced subgraph $\mathcal{S}_k$ by taking all request nodes in partition P_k, and all vehicle nodes $\mathcal{V}^{\text{veh}}$ (to preserve every vehicle-request edge for that block). Formally, for $k = 1, \ldots, K$: $\mathcal{S}_k = \mathcal{S}[P_k \cup \mathcal{V}^{\text{veh}}]$, where $\mathcal{S}[X]$ denotes the induced subgraph of $\mathcal{S}$ on the node set X. This ensures that each block has full access to the fleet when generating feasible trips, even though the request-request part is restricted to P_k. Note that the reattachment step will not introduce duplicate trips since each subgraph has disjoint sets of requests, and it it can be fully parallelized among vehicles and subgraphs. Feasible trip enumeration is then run *in parallel* on each $\mathcal{S}_k$, and the union of all feasible trips feeds into a single global ILP, with the same constraints as defined above.

Algorithm 2 Partition-Based ILP

Require: Shareability graph $\mathcal{S}$; set of active requests $\mathcal{R}$
RR $\leftarrow \{x \in \mathcal{S} \mid m(x) \in \mathcal{R}\}$ ▷ exclude vehicle nodes
SPs $\leftarrow$ Partition($\mathcal{R}$)
Trips $\leftarrow \emptyset$
for all $sp \in$ SPs **do** ▷ executed in parallel
  $sp = sp \cup \{x \in \mathcal{S} \mid m(x) \in \mathcal{N}\}$
  Round $\leftarrow \emptyset$
  for $i = 1, \ldots, k$ **do**
    $C \leftarrow$ GetPotentialFeasible(i, sp)
    for $c \in C$ **do**
      feasible $\leftarrow$ Routing(c)
      if feasible **then**
        Round $\leftarrow$ Round $\cup \{c\}$
      end if
    end for
  end for
  Trips $\leftarrow$ Trips $\cup$ Round
end for
AssignmentILP $\leftarrow$ BuildILP(Trips)

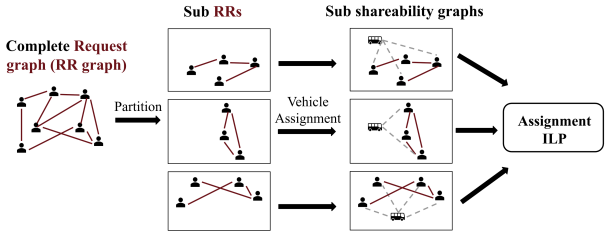

Fig. 4: Partition-based algorithm.

b) Partition algorithms: Graph partitioning, a classical NP-hard problem in parallel computing [31], is typically addressed with heuristics tuned to graph structure and objectives. The request-request graph here is undirected and homogeneous, with edges encoding binary shareability. We evaluate two well-established complementary partitioners. First, METIS [32]: A multilevel K-way method, producing balanced partitions with few edge cuts. Balanced workloads speed clique enumeration, but cuts may sever edges crucial for optimal trips. Second, modularity [33]: A community-detection approach maximizing within-group edge density. It better preserves dense local connectivity and high-quality trips, but can yield unbalanced partitions that slow the largest subproblems. In short, METIS favors uniform parallelism, while modularity favors feasibility. Both reduce complexity and enable parallel execution preserving the ILP formulation.

c) Properties: Let $\mathcal{T}$ the set of feasible trips (columns) produced by the baseline on $\mathcal{S}$, and $\mathcal{T}_{\text{part}} = \bigcup_{k=1}^K \mathcal{T}_k$ the union of feasible trips extracted from $\{\mathcal{S}_k\}_{k=1}^K$.

Lemma 4 (Preservation of feasible trips)**.** $\mathcal{T}_{\text{part}} \subseteq \mathcal{T}$.

Proof: Each $\mathcal{S}_k$ is an induced subgraph of $\mathcal{S}$; every

candidate clique that passes VRP feasibility in $\mathcal{S}_k$ is a valid feasible trip in $\mathcal{S}$. The re-attachment of vehicles preserves all vehicle-request edges; we never create edges not in $\mathcal{S}$. ■

Lemma 5 (Preservation of optimality)**.** Let $\mathcal{T}^\star \subseteq \mathcal{T}$ be a set of trips used by a baseline-optimal ILP solution. If every $\tau \in \mathcal{T}^\star$ satisfies $\mathcal{R}^\tau \subseteq P_k$ for some k (i.e., no optimal trip crosses a partition), then the partitioned ILP achieves the same optimal objective as the baseline.

Proof: $\mathcal{T}^\star \subseteq \mathcal{T}_{\text{part}}$ by construction, so the baseline-optimal solution is feasible in the restricted master. ■

Lemma 5 establishes optimality under the assumption that no optimal trip crosses graph partitions. When this assumption is violated, excluding cross-partition trips from the partitioned search space can lead to a nonzero optimality gap, whose magnitude depends on the prevalence of such trips.

Lemma 6 (Preservation of optimality)**.** Let $\mathcal{F}$ be the feasible-trip set produced by the baseline on $\mathcal{S}$ and let $\mathcal{B}^\star \subseteq \mathcal{F}$ denote any set of trips used by a baseline-optimal ILP solution. If for every $\tau \in \mathcal{B}^\star$ there exists a block P_k such that $R^\tau \subseteq P_k$, then the partitioned construction recovers all trips in $\mathcal{B}^\star$.

Remark 1 (Speedup)**.** Let $C(\mathcal{S})$ be the number of candidate cliques tested in the baseline and $C(\mathcal{S}_k)$ those in block k. If one VRP feasibility call costs T and blocks run in parallel, the expected wall-clock VRP time is $T_{\text{part}} \approx \max_k C(\mathcal{S}_k)T$, vs. $T_{\text{bas}} = C(\mathcal{S})T$. Balanced partitions (e.g., METIS) reduce $\max_k C(\mathcal{S}_k)$. High-modularity reduces $\sum_k C(\mathcal{S}_k)$ by pruning sparse cross-block cliques.

d) Practical aspects: First, "halo" blocks can be introduced by expanding each P_k with nearby requests, e.g., $\tilde{P}_k = P_k \cup \{r \in \mathcal{V}^{\text{req}} \colon \text{distance}_{\text{RR}}(r, P_k) \leq h\}$. With $h = 1$, any trip contained within a block plus its immediate neighbors is recoverable, and cross-block trips can be reconstructed by merging feasible trips with shared requests across partitions, capturing more high-value cliques at modest cost. Second, priority scheduling can allocate denser of higher-demand blocks to earlier or stronger processing cores, and may be combined with the data-driven filter by ranking cliques on predicted feasibility within each block. Finally, adaptive repartitioning in dense regimes, triggered every few decision windows based on current graph statistics (e.g., degree distribution), could maintain partition quality over time.

V. Numerical case study

We evaluate our accelerations against the baseline using the NYC TLC dataset [34], which captures real-world on-demand mobility. We focus on the morning peak (8-9 AM, May 15, 2024) with ∼13,000 trips over a network of ∼4,000 nodes and 10,000 edges from OpenStreetMap [35]. The decision window is one minute, with $M = 10^6$ in Problem 1 to prioritize passenger assignment over travel time. Vehicles have a capacity of 10 to reflect high-capacity service. We first show the scalability limits of the baseline and the reduction in optimality gap achieved by our accelerations, then compare service quality. All cases ran on a cluster with 2× Intel Xeon E5-2670 v2 CPUs (20 cores, 40 threads) and 128 GB RAM.

TABLE I: Optimality gap (%) across methods at a 60s aggregation interval. Bold represents decreased gaps.

Timeout [s]	Baseline	METIS	Modularity	Data-driven
15	46.0	24.8(**-21.2**)	27.2(**-18.8**)	46.8(+0.8)
30	46.0	24.0(**-22.0**)	26.8(**-19.2**)	39.7(**-6.3**)
45	45.6	23.2(**-21.8**)	25.6(**-20.0**)	35.5(**-10.1**)
60	44.0	16.8(**-27.2**)	24.8(**-19.2**)	29.6(**-14.4**)

A. Scalability issue

To showcase the complexity of ridepooling assignments, we study how solution quality scales with computation time. Demand is aggregated over 15s, 30s, 45s, and 60s intervals from 8 AM, yielding problems of increasing size. Using 100 vehicles, enough to serve over 95% of passengers in the 1-minute case, we run the baseline once at each scale. We then (i) solve to optimality for all intervals to compare runtimes, and (ii) for the 60s case, run baseline and accelerations under fixed time budgets to compare optimality gaps.

For the first set of experiments, solving to optimality required roughly 20, 80, 280, and 2000s for the 15s, 30s, 45s, and 60s aggregation windows, respectively, showing a sharp growth in complexity with scale. Thus, real-time applications cannot rely on optimal solutions within each decision window, and fixed time budgets are typically imposed. For the 60s window, results under fixed time budgets are reported in Table I. Because the objective (Problem 1) heavily weights passenger assignments via a large M, the reported gap reflects only the difference in assigned passengers between the optimal solution and the best solution found under timeout. Given the large optimality gap of the baseline under timeouts, improving efficiency is essential. The proposed accelerations reduce this gap by up to 27% under real-time constraints. Note that the 60 s aggregation window only includes requests made between 8:00-8:01. In practice, rolling-horizon operation also carries over unassigned requests. To better reflect real-world conditions, we therefore evaluate baseline and accelerated algorithms over a one-hour Manhattan simulation. For all subsequent experiments, we apply a 30 s timeout per decision window, leaving time for communication and dispatch, and vary fleet sizes from 400 to 800 vehicles. With 400 vehicles, the baseline serves about half of peak-hour demand; with 800, over 95% is served. This range creates diverse demand-supply balances to test the robustness of our methods.

B. Data-driven ILP construction

The data-driven method selectively skips cliques, prioritizing larger and more impactful ones. The data-driven method selectively skips cliques, prioritizing larger and more impactful ones. To train the feasibility predictor used for clique-skipping, we construct a labeled dataset by running the baseline algorithm and logging candidate cliques together with their feasibility outcomes determined by the baseline's feasibility check (solving a VRP subproblem for each clique). In total, 10,000 cliques are used for training, with balanced feasible and infeasible cliques. These cliques are encoded into the graph representation described in Fig. 3 and split into 75% training and 25% testing. On the held-out test set, the trained predictor achieves precision 0.87 and recall 0.86.

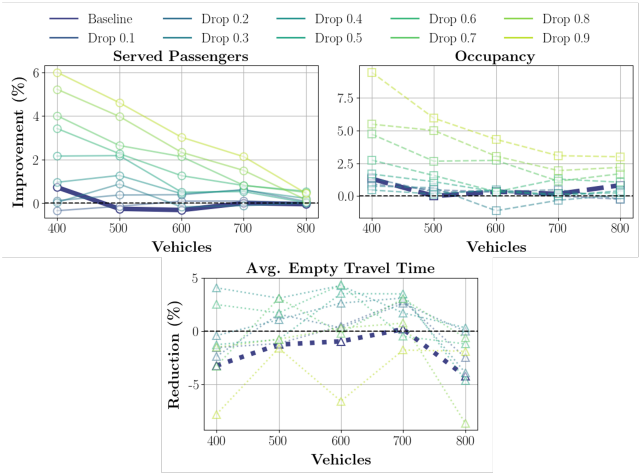

Fig. 5: Data-driven acceleration vs. baseline and various drop rates, with baseline comparison in bold.

With the trained feasibility predictor, we evaluate service performance of the data-driven method against the baseline in terms of served passengers, occupancy, and average Empty Travel Time (ETT). As a heuristic counterpart, we also test the baseline with random clique drops at various rates, enabling a direct comparison between ML-based filtering and naive pruning. Fig. 5 reports results across fleet sizes. Relative to the baseline, the data-driven method serves a similar number of passengers, slightly increases occupancy, and reduces ETT by up to 5%. This improvement has tangible consequences. A 5% cut in empty mileage for Manhattan's ride-hailing market (∼300M vehicle miles annually) corresponds to 15M fewer miles driven. For operators, this translates into millions of dollars saved in fuel, maintenance, and vehicle hours, while simultaneously increasing fleet availability for passengers. Environmentally, it reduces annual CO_2 emissions by an estimated 4,000-8,000 tons [36], [37], valued at $1-1.5M in avoided climate damages.

Random drop rates consistently yield fewer served passengers and lower occupancy than the data-driven one, and while ETT can occasionally be lower, performance is highly sensitive to chosen rates. Since the optimal rate depends on demand, supply, and network topology, it must be fine-tuned. In contrast, the data-driven method generalizes across settings, avoids parameter tuning, and outperforms the baseline.

C. Shareability graph partition

We evaluate partition-based acceleration using METIS [32], which balances subgraph sizes while minimizing edge cuts, and a modularity-based method [33], which favors dense communities over balance. Results are shown in Figure 6.

METIS outperforms the baseline at low and high fleet sizes but shows little or negative improvement at medium sizes. At low supply, dense shareability graphs make baseline clique exploration the bottleneck; METIS partitions the graph, enabling broader exploration and better assignments. At high supply, graphs are sparse, and partitions cut few edges, so METIS retains most connectivity while still gaining from parallelization. At intermediate supply, however, METIS may sever critical edges in pursuit of balance, reducing trip quality and offsetting parallelization gains. Modularity-based partition

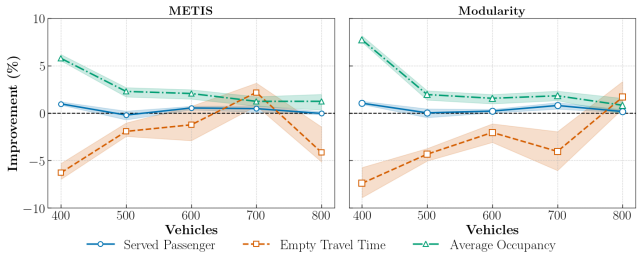

Fig. 6: Partition-based acceleration vs. baseline.

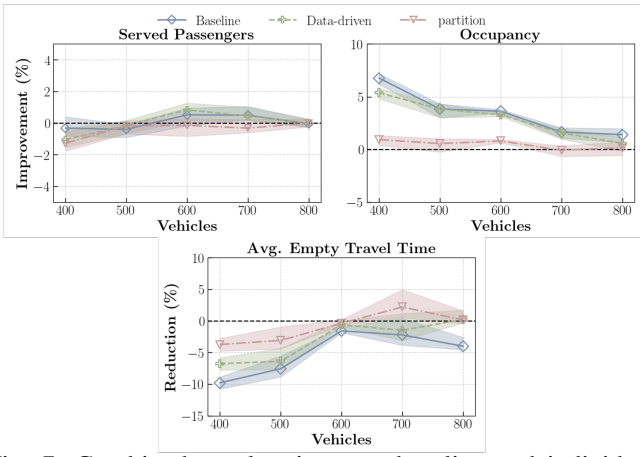

Fig. 7: Combined accelerations vs. baseline and individual methods. METIS for fleets of 400, 800; modularity for rest.

performs better at low supply but deteriorates as fleet size increases. With larger fleets, graphs are sparse, and it tends to preserve one large core with small peripheral clusters [38], yielding subproblems close in size to the baseline while still cutting edges, leading to worse solutions.

Overall, partitioning can accelerate the baseline, but effectiveness depends strongly on the demand-supply regime.

D. Data-driven ILP with partition-based parallelization

We now examine the combined use of data-driven ILP and partition-based parallelization, comparing it to the baseline and to each method individually (Fig. 7). As shown in Fig. 7, with 400-500 vehicles the combined approach further reduces ETT compared to either method alone, indicating their compatibility. These scenarios are especially challenging: low fleet supply produces large, dense shareability graphs, and unserved requests accumulate over time. Under such conditions, the synergy between data-driven filtering and parallelized exploration yields the largest benefits.

VI. Conclusion and Future Work

We studied the dynamic assignment problem with ridepooling, analyzed the state-of-the-art algorithm of [5], and proposed two accelerations to address real-time performance bottlenecks. Experiments on NYC data show that both methods reduce empty travel time under realistic time limits and can be combined for further gains, with direct benefits for congestion and sustainability. In practice, the choice of acceleration and performance improvement that it can achieve depends on the complexity of the assignment ILP. Partition-based strategies expand feasible trip exploration and improve solution

quality when complexity is manageable, while data-driven filtering is preferable when complexity is high, as it preserves problem size while pruning low-quality trips. Combining both is effective when additional performance is needed. In lower-density settings with reduced demand and supply, where the baseline method may already achieve near-optimal solutions within real-time constraints, the benefits of acceleration are expected to diminish. This work highlights general strategies for accelerating high-capacity ridepooling, rather than developing specialized ML solvers or graph partition methods. Future directions include integrating such advances into our framework, evaluating accelerations across diverse scenarios, and exploring co-design of fleet composition [39].

REFERENCES

[1] United Nations, Department of Economic and Social Affairs, Population Division, *World Urbanization Prospects: The 2018 Revision*. New York: United Nations, 2019. [Online]. Available: https://population.un.org/wup/assets/WUP2018-Report.pdf

[2] S. Oh, A. F. Lentzakis, R. Seshadri, and M. Ben-Akiva, "Impacts of Automated Mobility-on-Demand on traffic dynamics, energy and emissions: A case study of Singapore," *Simulation Modelling Practice and Theory*, vol. 110, p. 102327, 2021.

[3] G. Zardini, N. Lanzetti, M. Pavone, and E. Frazzoli, "Analysis and control of autonomous mobility-on-demand systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 633–658, 2022.

[4] S. Oh, R. Seshadri, C. L. Azevedo, N. Kumar, K. Basak, and M. Ben-Akiva, "Assessing the impacts of automated mobility-on-demand through agent-based simulation: A study of singapore," *Transportation Research Part A: Policy and Practice*, vol. 138, pp. 367–388, 2020.

[5] 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.

[6] A. Carron, F. Seccamonte, C. Ruch, E. Frazzoli, and M. N. Zeilinger, "Scalable model predictive control for autonomous mobility-on-demand systems," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 2, pp. 635–644, 2019.

[7] M. Tsao, R. Iglesias, and M. Pavone, "Stochastic model predictive control for autonomous mobility on demand," in *2018 21st International conference on intelligent transportation systems (ITSC)*. IEEE, 2018, pp. 3941–3948.

[8] S. Sadeghi Eshkevari, X. Tang, Z. Qin, J. Mei, C. Zhang, Q. Meng, and J. Xu, "Reinforcement learning in the wild: Scalable RL dispatching algorithm deployed in ridehailing marketplace," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 3838–3848.

[9] X. Li, C. Schmidt, D. Gammelli, and F. Rodrigues, "Learning joint rebalancing and dynamic pricing policies for autonomous mobility-on-demand," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–16, 2025.

[10] E. Liang, K. Wen, W. H. Lam, A. Sumalee, and R. Zhong, "An integrated reinforcement learning and centralized programming approach for online taxi dispatching," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 4742–4756, 2021.

[11] 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.

[12] Z. Woywood, J. I. Wiltfang, J. Luy, T. Enders, and M. Schiffer, "Multi-Agent Soft Actor-Critic with Global Loss for Autonomous Mobility-on-Demand Fleet Control," *arXiv preprint arXiv:2404.06975*, 2024.

[13] L. Tresca, C. Schmidt, J. Harrison, F. Rodrigues, G. Zardini, D. Gammelli, and M. Pavone, "Robo-taxi fleet coordination at scale via reinforcement learning," *arXiv preprint arXiv:2504.06125*, 2025.

[14] G. S. Nair, C. R. Bhat, I. Batur, R. M. Pendyala, and W. H. Lam, "A model of deadheading trips and pick-up locations for ride-hailing service vehicles," *Transportation Research Part A: Policy and Practice*, vol. 135, pp. 289–308, 2020.

[15] T. Enders, J. Harrison, M. Pavone, and M. Schiffer, "Hybrid multi-agent deep reinforcement learning for autonomous mobility on demand systems," in *Learning for Dynamics and Control Conference*. PMLR, 2023, pp. 1284–1296.

[16] I. Jindal, Z. T. Qin, X. Chen, M. Nokleby, and J. Ye, "Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 1417–1426.

[17] X. Yu and S. Shen, "An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, pp. 3811–3820, 2019.

[18] X. Li, M. Alharbi, D. Gammelli, J. Harrison, F. Rodrigues, M. Schiffer, M. Pavone, E. Frazzoli, J. Zhao, and G. Zardini, "Reproducibility in the control of autonomous mobility-on-demand systems," *arXiv preprint arXiv:2506.07345*, 2025.

[19] S. Shah, M. Lowalekar, and P. Varakantham, "Neural approximate dynamic programming for on-demand ride-pooling," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 507–515.

[20] Y. Kim, V. Jayawardana, and S. Samaranayake, "Learning-augmented vehicle dispatching with slack times for high-capacity ride-pooling," *Available at SSRN 4801437*, 2024.

[21] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, p. 1633, 2018.

[22] A. Simonetto, J. Monteil, and C. Gambella, "Real-time city-scale ridesharing via linear assignment problems," *Transportation Research Part C: Emerging Technologies*, vol. 101, pp. 208–232, 2019.

[23] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proceedings of the National Academy of Sciences*, vol. 111, no. 37, pp. 13 290–13 294, 2014.

[24] M. Cáp and J. Alonso-Mora, "Multi-objective analysis of ridesharing in automated mobility-on-demand," in *RSS 2018: Robotics-Science and Systems XIV*, 2018.

[25] S. Sethi and G. Sorger, "A theory of rolling horizon decision making," *Annals of operations research*, vol. 29, no. 1, pp. 387–415, 1991.

[26] Y. Dumas, J. Desrosiers, and F. Soumis, "The pickup and delivery problem with time windows," *European Journal of Operational Research*, vol. 54, no. 1, pp. 7–22, 1991.

[27] *Column (and Row) Generation Algorithms*. John Wiley & Sons, Ltd, 2020, ch. 11, pp. 213–233.

[28] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[29] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[30] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International conference on machine learning*. PMLR, 2017, pp. 1321–1330.

[31] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel computing*, vol. 26, no. 12, pp. 1519–1534, 2000.

[32] G. Karypis and V. Kumar, "Multilevelk-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed computing*, vol. 48, no. 1, pp. 96–129, 1998.

[33] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 70, no. 6, p. 066111, 2004.

[34] New York City Taxi and Limousine Commission, "Tlc trip record data," 2025, accessed: 2025-05-05. [Online]. Available: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[35] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[36] New York City Mayor's Office of Climate & Environmental Justice, "NYC 2023 Greenhouse Gas Inventory Methodology," City of New York, New York, NY, Tech. Rep., Mar. 2025. [Online]. Available: https://climate.cityofnewyork.us/wp-content/uploads/2025/03/GHG-Inventory-Methodology-2023.pdf

[37] U.S. Environmental Protection Agency. (2025, Jun.) Greenhouse gas emissions from a typical passenger vehicle. Last updated June 12, 2025. [Online]. Available: https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle

[38] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[39] G. Zardini, N. Lanzetti, A. Censi, E. Frazzoli, and M. Pavone, "Co-design to enable user-friendly tools to assess the impact of future mobility solutions," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 2, pp. 827–844, 2022.