# CAC: Enabling Customer-Centered Passenger-Seeking for Self-Driving Ride Service with Conservative Actor-Critic


Palawat Busaranuvong
*Worcester Polytechnic Institute*
pbusaranuvong@wpi.edu

Xin Zhang
*San Diego State University*
xzhang19@sdsu.edu

Yanhua Li
*Worcester Polytechnic Institute*
yli15@wpi.edu

Xun Zhou*
*University of Iowa*
xun-zhou@uiowa.edu

Jun Luo
*Logistics and Supply Chain MultiTech R&D Centre, Hong Kong*
jluo@lscm.hk



***Abstract*—Rapid advances in perception, planning, and decision-making areas for self-driving vehicles have led to great improvements in their function and capabilities and enabled several prototypes to be driving on the roads and streets, such as Waymo Driver, TuSimple, Nuro, *etc*. Among various applications of self-driving vehicles, a promising one is the ride service as it has the potential to improve service quality and productivity and to provide service to anyone at any time. Extensive studies have been conducted on self-driving planning and safety, but few works focus on self-driving ride service decision-making and routing. In this work, we take the lead to study self-driving ride service planning and decision-making problem leveraging human-generated spatial-temporal data, and propose the data-driven Conservative Actor-Critic approach – CAC – based on offline reinforcement learning. Our CAC is able to make conservative decisions in a complicated environment with multiple goal states, and avoid dangerous and overly optimistic behaviors by exploiting human decisions. Extensive experiments with real-world data demonstrate that our CAC-learned policies are able to improve taxi service operation efficiency and quality drastically in terms of shortening passenger waiting time and improving service revenue.**

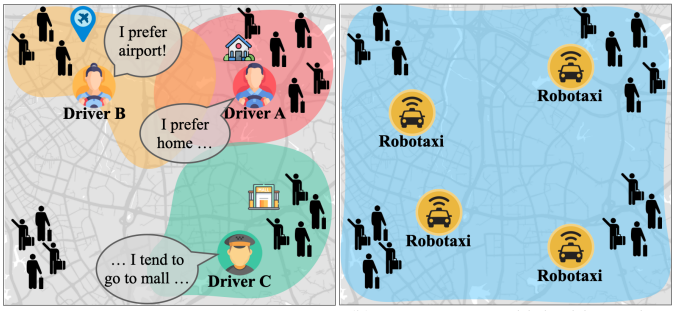


***Index Terms*—offline reinforcement learning, spatial-temporal data mining, actor-critic, conservative Q-learning**


## I. Introduction

The advancements in perception, planning, and decision-making have revolutionized self-driving vehicles or autonomous vehicles [1]. Various prototypes, such as Waymo Driver [2], TuSimple [3], Nuro [4], *etc*., have been successfully developed and can be observed operating on roads and streets. Self-driving vehicles have extensive commercial applications, including freight delivery [3], order fulfillment and logistics [5], and even farming machinery [6]. However, the development of self-driving ride services, or robotaxis, holds significant importance. Robotaxis have the potential to enhance road safety by reducing the number of accidents, provide convenient and equal access to transportation for all passengers at any time, and improve service quality and efficiency. Achieving these goals necessitates advancements in various aspects of vehicle autonomy, especially, efficient and intelligent passenger-seeking automation strategies.


*Corresponding author.


(a) Human drivers with diverse working zones and preferences.

(b) Autonomous vehicle ride service covering all zones.

Fig. 1: Diverse decision preferences from human agents *vs*. self-driving vehicles.

**Differences of self-driving *vs*. human ride services.** Ride services serve passengers and transport passengers from an origin to a destination. Traditional ride services operated by human agents are greatly influenced by personal constraints for example, working preferences and proficiency. These constraints put limits on the ride service quality and efficiency. Fig. 1(a) shows three human drivers' familiar working areas or "comfort zones", where driver A prefers to work close to home locations, driver B would like to serve longer trips and seek passengers around the airport, and driver C tends to serve around the downtown area close to shopping malls. Due to the biased service preferences, passengers in the southwestern location fail to be well-served. None of the three human agents focus purely on shortening passenger waiting time and improving the ride service quality. Unlike human agents, ride service operated by autonomous vehicles has no prejudice, preferences, or constraints on serving passengers. It has the capability to provide customer-centered and equal ride services to all passengers in all areas at any time shown in Fig. 1(b).

**Literature on passenger-seeking strategy learning.** Due to

a lack of historical passenger-seeking data from autonomous vehicles, researchers turn to human passenger-seeking data to learn decision-making strategies [7]–[10]. There has been rich literature on learning passenger-seeking strategies utilizing imitation learning (IL) and inverse reinforcement learning (IRL) techniques [7]–[10]. These works model passenger-seeking behaviors using the Markov Decision Process (MDP), and learn human decision-making policies from the trajectories of experienced drivers with high earning efficiency. However, these approaches have limitations if applied to self-driving ride service scenarios. They require significant interactions with the real-world environment while learning, which can be expensive and impractical. In addition, human decision preferences vary greatly among diverse decision-makers, in various geographical locations, at different time [7]–[9]. Though experienced, human decision-makers and their objectives can be influenced by various factors and do not necessarily prioritize passenger-serving and profit-earning. Consequently, IRL and IL-learned policies are adversely affected by diverse decision preferences and fail to target improving service quality and efficiency. Moreover, IRL or IL-learned policies have a performance ceiling where it is unlikely to exceed the performance of the demonstrated human agents. Therefore, IRL or IL-based approaches cannot help design passenger-seeking plans and strategies for self-driving passenger services.

Other approaches based on offline RL [11]–[18] try to avoid expensive environment interactions from previously collected data and learn policies with one unique goal. One line of works [12], [15]–[17] value the importance of exploration for more robust policies and apply state-independent noises to boost exploration. Though demonstrate remarkable performance, policies learned from them likely lead to reckless behaviors and impair self-driving passenger service quality and safety. Therefore, another line of works [13], [14], [18] solve the extrapolation error problem, *i.e.*, overestimating out-of-distribution actions, and fully leverage given data to generate safe and conservative behaviors. However, none of them target the passenger-seeking problem in the self-driving ride service setting. Therefore, we plan to leverage offline RL and design a safe and data-driven passenger-seeking strategy for providing high-quality self-driving passenger services in near future.

In this work, we are motivated to design a data-driven automatic passenger-seeking strategy prioritizing improving service quality in terms of shortening passenger waiting time and increasing service revenue. We make the first attempt to learn unique decision-making policies to efficiently find a passenger leveraging a massive amount of human-generated spatial-temporal data without interactions with the real-world environment, and propose the innovative Conservative Actor-Critic approach, in short CAC, based on offline reinforcement learning (offline RL) [11]. CAC has the capability to extract useful decision-making knowledge from diverse human behaviors and demonstrations. It thus allows a robotaxi to adapt to different environments and service scenarios to produce optimal behaviors. *Our contributions* are summarized as follows:

- To the best of our knowledge, we are the first to bring up the autonomous vehicle passenger-seeking problem in a spatial-temporal decision-making environment to improve self-driving ride service quality and productivity in shortening passenger waiting time and increasing operation earning efficiency.
- We propose an innovative data-driven Conservative Actor-Critic (CAC) approach based on offline RL which leverages human-generated spatial-temporal data and learns conservative passenger-seeking strategies to avoid dangerous and reckless behaviors without environment interactions.
- We validate our framework on real-world passenger-seeking trajectory data, and our CAC shows remarkable performance in reducing the average passenger waiting time by half, when compared with baselines and human strategies. *We made our code and unique dataset available to contribute to the research community Github [19].*

## II. Overview

In this section, we define the robotaxi passenger-seeking problem and highlight the research challenges.

### A. Human-Generated Spatial-Temporal Data for Passenger-seeking

Human-generated spatial-temporal data (HSTD), *e.g.*, taxi GPS traces, records human mobility from their sequential decisions. We leverage HSTD to learn how taxi drivers make decisions when they are seeking passengers. HSTD from taxis consists of multiple sequences of passenger-seeking trajectories. Each trajectory is a series of *spatial-temporal states* by following a sequence of decisions as *actions*. We give a formal definition of the terms we use below.

**Definition 1: A state** $s$ represents a spatial-temporal location determined by latitude $lat$, longitude $lng$, and time stamp $t$, *i.e.* $s = \langle lat, lng, t \rangle$. The set of all states is denoted as $\mathcal{S}$, with $\mathcal{S} = \{s\}$. Note that each (spatial-temporal) state is associated with a set of decision-making-related features, *e.g.*, traffic speed and volume of the nearby area.

**Definition 2: An action** $a$ is a decision that a human makes to move from one state $s$ to another state $s'$ in order to complete a task. For example, a taxi driver would choose different directions to go as an action for searching for a passenger. They may also choose to terminate a passenger-seeking trajectory as an action to pick up a passenger. The space of all actions is denoted as $\mathcal{A}$, *i.e.*, $\mathcal{A} = \{a\}$.

**Definition 3: A trajectory** $tr$ is a sequence of spatial-temporal state-action pairs that an agent takes when completing a task in a geographic region, *i.e.*, $tr = (s_1, a_1, s_2, a_2, \cdots, s_T, a_T)$, where $T$ is the length of trajectory $tr$. Moreover, $Tr = \{tr_1, tr_2, \cdots, tr_m\}$ denotes a trajectory set with $m$ trajectories generated by diverse taxi drivers.

Note that we focus on learning from taxi drivers' "seeking" trajectories to acquire a good decision-making strategy, *i.e.*, *policy*, to enable shorter passenger-waiting time and higher income than human agents. Once a passenger is found and picked up, a passenger-seeking process ends with an immediate *reward* based on the revenue from the subsequent service of

the passenger. Hence, the passenger-seeking strategy is closely related to two functions (formally defined below): (i) *reward function* (evaluating the outcome of an action) and (i) *policy function* (likelihood of choosing a particular action at a state).

**Definition 4: A reward** $r$ is assigned at the final state, or goal state when a passenger is found and picked up. It is zero along the search trajectories. Since in the self-driving taxi service setting, service providers prioritize service quality and efficiency, the reward has two parts: it is inversely proportional to the time spent on looking for passengers and is proportional to the service income. The service income is computed based on the distance from the passenger pickup state to the passenger drop-off state. Note that the reward $r$ defined here does not reflect human decision preferences, but guides a robotaxi towards improving self-driving ride service quality and efficiency only.

**Definition 5: A policy** $\pi$ characterizes the probability distribution to choose an action $a$ given the current state $s$.

Based on the above definitions, we aim to train a robotaxi agent with an optimal policy that is able to get the highest reward via interacting with the environment.

### B. Autonomous Vehicle Passenger-seeking Problem

*Problem Definition.* Given a set of passenger-seeking trajectories of diverse drivers $Tr$, we aim to learn a passenger-seeking policy $\pi$ achieve the highest reward $r$ in terms of minimizing passenger-seeking time and maximizing service revenue.

*Challenges.* The proposed autonomous vehicle passenger-seeking problem is challenging in two aspects: (**C1**) Given a complex passenger-seeking environment with multiple goal states (*i.e.*, states with passengers), how to effectively learn a policy that is able to identify an appropriate goal state that shortens passenger waiting time in general? (**C2**) How to leverage HSTD to learn good policies without environmental interactions while avoiding overly optimistic estimations of unseen actions at unseen states in the dataset?

To tackle these challenges, we propose our solution in the following sections as is shown in Fig. 2.

## III. Methodology

### A. Stage 1: Passenger Seeking Decision-Making Modeling

We model the taxi-driver passenger-seeking process as a Markov decision process (MDP) following [7]–[9], [20]. An MDP is a mathematical framework to model stochastic decision-making processes where outcomes are uncertain. It is defined as a 5-tuple: $\langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P$ is the transition function, $r$ is the reward function, and $\gamma$ is the discount factor. The transition function $P(s'|s,a)$ tells the probability of reaching state $s'$ by taking action $a$ at state $s$. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function that outputs a reward given the current decision $a$ at a state $s$. The discount factor $\gamma$ ranges between 0 and 1, which discounts the future reward exponentially.

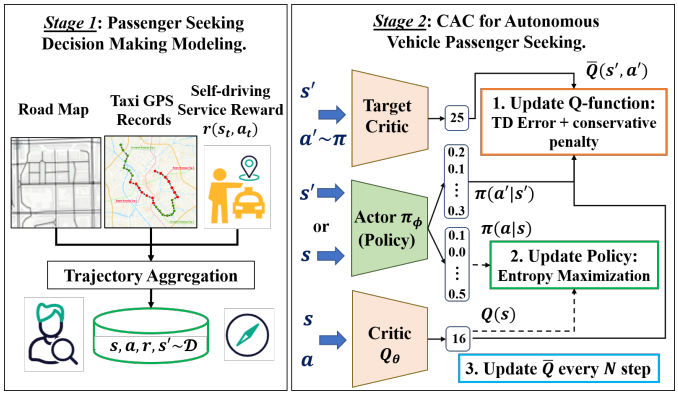



Fig. 2: Solution Framework.

**Self-driving service reward.** To prioritize service quality and efficiency, we specify the reward as a combination of passenger-seeking time and service income, *i.e.*,

$$r(s_t, a_t) = \begin{cases} (2-\alpha) f(s_t, a_t) & \text{if } a_t \text{ is terminate;} \\ -\alpha & \text{otherwise.} \end{cases}$$

Here, the $f$ function reflects the charging criterion in a service city. For example, in Boston, MA in the States, a taxi costs \$2.60 basic fee and the per kilometer price for traveling is \$1.75 [21]. The $f(s_t, a_t)$ value then evaluates the charge for a pick up at state $s_t$ in a specific city [1]. In the reward function, we penalize each time step for seeking passengers by $\alpha$ which has scalar ranges between 0 and 1 to balance the influence between passenger-seeking time and service revenue. In this paper, we use a fixed value of $\alpha$ to 0.8 for training RL models as experiment demonstrate the best model performance. We omit the experiment results of different $\alpha$ due to limited space.

**Trajectory aggregation.** Given the above reward function $r$, we aggregate the trajectories $tr \in Tr$ from HSTD with rewards. At a specific state $s_t$ with action $a_t$ at time step $t$, the reward for such a state-action pair is evaluated with the reward function $r$, *i.e.*, $r_t = r(s_t, a_t)$. Then we aggregate each decision at $(s_t, a_t)$ with its reward $r_t$ to get a state-action-reward triple $(s_t, a_t, r_t)$. A sequence of trajectories $tr$ as a sequence of state-action pairs can thus be processed as a sequence of state-action-reward triple, *i.e.*, $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T)$ where $T$ is the trajectory length. The set of all aggregated trajectories is denoted as $\mathcal{T} = \{\tau\}$. We further extract the trajectories into a collection of transitions in a data buffer $\mathcal{D}$, *i.e.*, $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{T-1}$.

**Autonomous vehicle passenger seeking problem.** Given a set of HSTD processed as a collection of aggregated trajectories $\mathcal{T}$ and a reward function favoring high service quality, the autonomous vehicles in a passenger-seeking scenario then strive to learn a good policy $\pi^*$ that is able to attain the highest reward when interacting with the environment, *i.e.*, searching

[1]We are aware of the uncertainty in service destination and request, and view the service distance at a state $s_t$ as a static value for ease of evaluation. This approach can also be applied in general settings considering various service requests and destinations.

for and serving passengers. That is, the optimal policy is able to maximize the accumulated reward, *i.e.*,

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t, a_t)\right]. \quad (1)$$

Note that during training, the learner policy $\pi$ has no access to the environment.

### *B. Stage 2: Conservative Actor-Critic (CAC) for Autonomous Vehicle Passenger Seeking*

Leveraging HSTD and the decision-making model with the objective in Eq. (1), we target to learn a passenger-seeking strategy that only focuses on improving self-driving ride service quality without environmental interactions. For this, we turn to offline RL for solutions.

**Offline RL for Autonomous Vehicle Passenger Seeking.** Various offline RL models [13], [14], [18], [22] have been proposed to train their decision-making policies from previously collected data, rather than through trial and error in the environment. This approach can be helpful when the collected data is diverse and large, or when the environment is too dangerous or costly for the agent to explore. Therefore, we propose to apply offline RL to tackle the autonomous vehicle passenger seeking problem. Below is a SOTA offline RL model that can be applied to learn passenger-seeking strategies.

*Conservative Q-Learning (CQL) [18]* is an offline RL approach to solve the extrapolation error problem, *i.e.*, learning overly optimistic policies which leads to dangerous decisions. It learns a conservative action-value estimation on out-of-distribution decisions. Unlike traditional Q-learning [23], [24], CQL uses a modified update rule of the Q-value function by adding a conservative regularizer into the Bellman equation,

$$Q^* \leftarrow \beta\,\underbrace{\mathbb{E}_{s,a\sim\mathcal{D}}\left[\log\sum_{a}\exp(Q(s,a)) - Q(s,a)\right]}_{\text{Conservative penalty}} + \min_{Q}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[r(s,a) + \gamma\mathbb{E}_{a'\sim\pi(a'|s')}[\bar{Q}(s',a')] - Q(s,a)\right]^2. \quad (2)$$

The conservative regularizer allows it to learn a conservative lower bound of the true value of a policy, ensuring that Q-values on unseen actions are not overestimated. Here, $\beta$ is a hyperparameter controlling the degree of "conservativeness", *i.e.*, how much to stick to offline demonstrations, during $Q$ function learning. The conservative penalty encourages the learned action-value function $Q$ to favor observed state-action pairs $(s, a)$, and discourages reckless trial and error which likely leads to dangerous and overly optimistic behaviors. Note that since the CQL is based on DQN [23], [24], it has a target action value function denoted as $\bar{Q}$ in the above equation.

*Limitation of SOTA works.* The CQL approach has limitations if directly applied to our problem. The CQL solution is based on the Deep Q-Network (DQN) model [18], which optimizes the policy solely based on the $Q$-value maximization, leading to a deterministic policy. The action for each state is determined by selecting the action with the highest $Q$ value. However, we desire a stochastic policy. Since the passenger-seeking environment is highly dynamic with multiple goal states having potential passengers, a deterministic policy likely leads to autonomous passenger service congestion in the same state neglecting potential services elsewhere.

**CAC: Conservative Actor-Critic.** Given the above limitations to our passenger-seeking problem, we propose a new offline RL framework, called the Conservative Actor-Critic (CAC) model, to improve passenger service quality and efficiency.

The CAC model is an actor-critic-based approach. It consists of two functions to be learned, *i.e.*, an actor or policy function $\pi$ and a critic or $Q$ function, which are updated iteratively towards an optimal policy $\pi^*$. The offline RL CAC model seeks to maximize the entropy of the policy, represented by $\pi$, encouraging the exploitation of different actions in different states observed in the large dataset. This is particularly useful in passenger-seeking scenarios where there may be more than one optimal action in a given state as mentioned in the challenge (**C1**). Additionally, the CAC model addresses the challenge (**C2**) of extrapolation errors by incorporating a conservative regularization term into the Bellman optimal equation when updating the critic function $Q$. Below we detail our design choices for CAC.

*Entropy Maximization to Encourage Exploring the Data.* To attain a stochastic policy catering to the multiple goal-states scenario, we incorporate an entropy maximization term while learning a policy $\pi$ on top of the objective in Eq. (1), *i.e.*,

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t, a_t) + \zeta\mathcal{H}(\pi(\cdot|s_t))\right].$$

Here, $\mathcal{H}(\pi(\cdot|s_t)) = \mathbb{E}_{a\sim\pi(s_t)}[-\log\pi(a|s_t)]$ is the entropy of the policy $\pi$ at state $s_t$, and the scalar $\zeta$ controls the level of entropy maximization. The entropy term plays an important role, as it is a measure of the randomness or uncertainty of the policy. By maximizing the entropy of the policy, the agent is encouraged to explore different actions in different states observed in the dataset, rather than being overly deterministic and sticking to a small set of actions. In addition, considering we do not have an interactive environment, the proposed CAC solution can only leverage given human decision data $\mathcal{D}$. Thus, in our CAC approach, the policy is optimized by maximizing the soft-state value function of the policy following equation (See Appx for detailed derivation),

$$\pi^* \leftarrow \max_{\pi}\mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}\left[Q(s,a) - \zeta\log(\pi(a|s))\right]\right]. \quad (3)$$

Based on the policy entropy maximization principle, the state-value function and the action-value function of a policy $\pi$ at state $s$ and action $a$, that is, $V(s)$ and $Q(s, a)$ are changed to include the entropy term as well. We denote them as $V^\pi(s)$ and $Q^\pi(s, a)$ respectively to make this change clear, *i.e.*,

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}\left[Q(s,a) - \beta\log(\pi(a|s))\right],$$
$$Q^\pi(s,a) = \mathbb{E}_{s'\sim\mathcal{D}}\left[r(s,a) + \gamma V^\pi(s')\right].$$

*Conservative Regularization to Reduce Extrapolation Errors.* To optimize the action-value function $Q^\pi(s, a)$, we apply

a temporal-difference (TD) approach. We leverage a target action-value function of the policy $\pi$, *i.e.*, $\bar{Q}^{\pi}(s,a)$, and try to minimize the TD error between the target action-value function and the action-value function at the current state $s$ with action $a$, *i.e.* $Q^{\pi}(s,a)$. The TD error $TD$ can be expressed as,

$$TD = \mathbb{E}_{s,a,s' \sim \mathcal{D}} \left[ (\bar{Q}^{\pi}(s,a) - Q(s,a))^2 \right]. \tag{4}$$

Here, we consider a conservative regularization term when learning the $Q$ function in CAC to encourage a conservative learner policy $\pi$. Specifically, we apply the TD error in Eq. (4) and incorporate the conservative penalty term in Eq. (2) to learn and update the action value function $Q$. This results in learning a conservative lower bound of the critic function [18]. Therefore, the action-value function objective becomes,

$$\begin{aligned} Q^* \leftarrow \min_{Q} & \mathbb{E}_{s,a,s' \sim \mathcal{D}} \left[ (r(s,a) + \gamma \bar{V}^{\pi}(s') - Q(s,a))^2 \right] \\ & + \beta \mathbb{E}_{s,a \sim \mathcal{D}} \left[ \log \sum_{a} exp(Q(s,a)) - Q(s,a) \right], \end{aligned} \tag{5}$$

and $\beta$ is a hyper-parameter controlling how much the conservative term will affect the Q function optimization.

**Our CAC Algorithm.** We model our policy or actor function $\pi$ and action-value or critic function $Q$ as two neural networks parameterized by $\phi$ and $\theta$ respectively, *i.e.*, $\pi_{\phi}$ and $Q_{\theta}$. Instead of using only one critic function, we follow the rich literature [22], [25] to use two copies of the critic networks $Q_{\theta_1}$ and $Q_{\theta_2}$ to avoid overestimating Q-values over actions by selecting the one that estimates lower Q-values. Additionally, using two critic networks can improve the stability of the learning process and accelerate the convergence of the algorithm. Therefore, our framework adopts this technique and employs two critic networks to improve the performance. The CAC training procedure is shown in Alg. 1.

In Alg. 1, there are 5 neural networks to be trained, *i.e.*, a policy ($\pi$), 2 Q-functions $Q_{\theta_1}$ and $Q_{\theta_2}$, and 2 target Q-functions $\bar{Q}_{\bar{\theta_1}}$ and $\bar{Q}_{\bar{\theta_2}}$. During training, we apply batch gradient descent to update all networks. For each gradient step, we sample a batch of transitions $B = \{(s,a,r,s')\}$ from our static training buffer $\mathcal{D}$ (Line 2). We then compute target Q functions at states $s'$ and $a'$, *i.e.*, $\bar{Q}_{\bar{\theta_1}}(s',a')$ and $\bar{Q}_{\bar{\theta_2}}(s',a')$. Then we select one target Q network that obtains a lower Q value for computing the soft target Q function of the policy $\bar{Q}^{\pi}(s,a)$ at the current state $s$ (Line 3). This network will be used for updating the conservative Q-functions via minimizing the TD error and the conservative penalty term through gradient descent using the Adam [26] optimizer (Lines 4). With two Q-networks, the total loss is the summation of both $Q_{\theta_1}$ and $Q_{\theta_2}$ losses. After that, we update the policy $\pi_{\phi}$ by maximizing the summation of the expected return and the expected entropy of the policy at the current state $s$ with action $a \sim \pi_{\phi}(a|s)$ using gradient ascent (Line 5). Adam optimizer is used for updating this gradient as well. Now, after every $N$ gradient step, we update the parameters of both target Q networks $\bar{\theta}_1$ and $\bar{\theta}_2$ by replacing them with the current parameters of Q functions $\theta_1$ and $\theta_2$, respectively (Line 6).

**Algorithm 1** Conservative Actor-Critic

**Require:** Initial policy $\pi_{\phi}$, Q-functions $Q_{\theta_1}$, $Q_{\theta_2}$, target Q-functions $\bar{Q}_{\bar{\theta_1}}$, $\bar{Q}_{\bar{\theta_2}}$, and dataset $\mathcal{D}$. Target parameters are assigned as $\bar{\theta}_1 = \theta_1$, $\bar{\theta}_2 = \theta_2$.

**Ensure:** A learned policy $\pi_{\phi}$.

1: **for** $t$ in $\{1, ..., K\}$ **do**

2: Sample a transitions batch $B_t = \{(s,a,r,s')\}$ from $\mathcal{D}$.

3: Compute soft target Q function following

$$\bar{Q}^{\pi}(s,a) = r + \gamma \left[ \min_{i=1,2} (\bar{Q}_{\bar{\theta}_i}(s',a')) - \zeta \log(\pi_{\phi}(a'|s')) \right].$$

4: Update conservative Q-functions by gradient descent as

$$\nabla_{\theta_i} \sum_{(s,a,r,s') \in B_t} (\bar{Q}^{\pi}(s,a) - Q_{\theta_i}(s,a))^2 + reg_i$$

for $i = 1, 2$, where

$$reg_i = \beta \mathbb{E}_{s,a \sim B_t} \left[ \log \sum_{a} \exp(Q_{\theta_i}(s,a)) - Q_{\theta_i}(s,a) \right].$$

5: Update policy by gradient ascent with

$$\nabla_{\phi} \sum_{s \in B_t} \left( \left[ \min_{i=1,2} (Q_{\theta_i}(s,a)) - \zeta \log(\pi_{\phi}(a|s)) \right] \right).$$

6: Update target parameters every $N$ steps, *i.e.*, $\bar{\theta}_i \leftarrow \theta_i$ for $i = 1, 2$.

7: **end for**

8: Return the learned policy $\pi_{\phi}$.

## IV. Experiment

We evaluate our CAC performance using the taxi trajectory dataset representing the taxi driver's passenger-seeking decision process. In this section, we show that our CAC approach is able to learn passenger-seeking strategies that minimize the passenger-seeking time (or passenger-waiting time) and maximize ride service earning efficiency.

### A. Data Description and Preparation

Taxi trajectory data were collected in July 2016 in Shenzhen, China, which contains GPS records from 8,572 taxis. Every GPS record includes five attributes: a unique plate ID, longitude, latitude, time stamp, and passenger indicator. Passenger indicator contains binary values, in which 1 indicates that a taxi is serving a passenger and 0 otherwise.

*1) Map Gridding and Time Quantization.:* We follow [7]–[9] to first process GPS data into spatial-temporal grid cells. Specifically, we partition the Shenzhen map into equal side-length grid cells and split the time in a day into five-minute intervals for a total of 288 intervals per day. Therefore, we represent a taxi position in the spatial-temporal state as the spatial-temporal grid cell.

*2) State Feature Extraction & Action Specification:* We extract decision-related spatial-temporal features to represent a state, including *distance to points of interests (PoIs)* and *traffic* information. We specify different directions to go as actions.

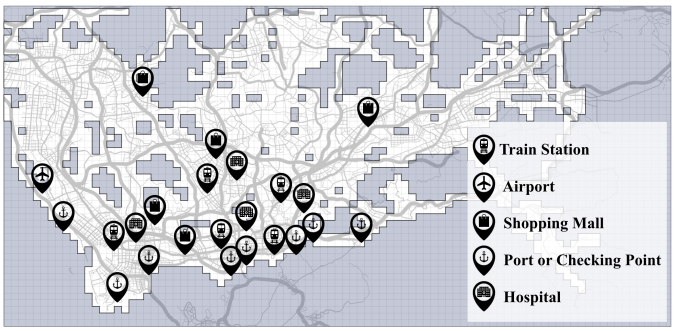



Fig. 3: Points of Interests (PoIs) in Shenzhen, China.

**State features:** Distance to PoIs: We consider 21 important PoIs in Shenzhen, China, including train stations, airports, popular shopping malls, ports & checking points, and major hospitals shown in Fig. 3. We evaluate the distances from a grid cell $s$ to PoIs as distance to PoIs features. Traffic features: In the passenger-seeking process, many factors are considered, for example, the current traffic speed, traffic volume, travel demand, and waiting time. In addition, the traffic conditions in the near regions also affect passenger-seeking decision-making. Given a spatial-temporal grid cell, we consider this traffic-related information for the current grid cell and its neighboring $5 \times 5$ grid cells. Here, the traffic speed in a state $s$ is calculated from an average speed of all trajectories passing the state $s$, the traffic volume estimates the average number of taxis in a state $s$, the travel demand estimates the number of passengers looking for taxis. Therefore, after performing state feature extraction, we get the *state feature* vector of size $[5 \times 5 \times 5]$. From now on, we call a spatial-temporal grid cell as well as its related state features as a state for convenience.
**Action space**: When a taxi is at a state $s$, the driver can choose an action $a$ from 10 possible options. Each action, $a \in \mathcal{A}$, is to move from the current grid to one of the eight neighboring grid cells (*i.e.*, $\{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow\}$), stay in the current one (*i.e.*, $\circlearrowleft$), or to pick up a passenger (*i.e.*, $pickup$).

*3) Passenger-seeking simulation.:* We created a deterministic passenger-seeking environment based on the above taxi trajectory data for the comparison of approaches requiring environmental interactions. The simulation is based on the OpenAI FrozenLake environment [27] except that it has multiple goal states (*i.e.* passenger-pickup states). In the goal states, the rewards are calculated based on the passenger-seeking time and service distance from all taxi-trajectory records. However, to get the reward from one of the goal states, the agent must visit that goal state and chooses the action to terminate and pick up in this state, then the agent will get a reward. If other actions are selected, the agent will move to the next state and must continue searching for passengers. We open source this simulation environment via an anonymous link [19].

### *B. Experiment Settings*

In this section, we detail the experiment settings including the evaluation metrics and baseline methods. We randomly split the taxi trajectory dataset into two parts: *training set* (one million trajectories), *validation set* (10,000 trajectories), and *testing set* (10,000 trajectories). For testing, we extract and fix the starting states (*i.e.*, the first states from each validation/test trajectory), and let the learned policies from each experiment approach to generate passenger-seeking trajectories using and starting from the extracted starting states with the simulated environment. If a learner policy takes longer than 60-time steps (*i.e.*, 300 minutes) [2] while searching for a passenger, the seeking process is stopped with zero earning. If an inaccessible state is reached, the seeking agent will fail whose seeking time is added with 300 minutes as a penalty. In terms of the reward function $r$ used during training, we set $\alpha$ to be 1 as we value shortening passenger-seeking time and serving passengers as soon as possible.
**Evaluation metrics.** In order to assess the effectiveness of our proposed CAC framework in reducing passenger-seeking times and improving ride service quality and efficiency, we consider the following three scores:

- **Average seeking time (in minutes)** measures the time elapsed from the point at which the agent begins seeking a passenger at the initial grid cell until the point at which it chooses to pick up a passenger.
- **Earning Efficiency,** $EE$ **(Yuan/hour)** reflects the efficiency of passenger-seeking processes. It is calculated as the total income (estimated from historical data in Chinese Yuan) divided by the total seeking time.
- **Success Rate** evaluates the portion of generated passenger-seeking trajectories that successfully finds a passenger. It is calculated based on the fraction between the number of successful trajectories over the total number of trajectories.

**Baseline Methods** To compare our proposed CAC with baselines, we consider the IL approaches targeting human behavior analysis, and offline RL approaches that learn passenger-seeking strategies without environmental interactions. Note that IL approaches require learning from demonstrations of experienced drivers instead of random drivers in our approach, so we prepared different training data for these approaches accordingly [3]. The details of the baselines are as below:

- **Behavior Cloning (BC) [28]** is a supervised learning approach to solving the IL problem. Given human demonstration data as a set of state-action pairs, the learner policy is trained by viewing states as inputs and actions as labels.
- **Generative Adversarial Imitation Learning (GAIL) [10]** is an IL method consisting of a policy network mimicking human behaviors, and a reward network distinguishing between learner and human behaviors. The policy net and reward net are adversarially and iteratively trained to obtain an optimal learner policy producing human-like behaviors.
- **Conditional Generative Adversarial Imitation Learning (cGAIL) [8]** is an IL approach based on GAIL. Unlike GAIL which models each human agent's decision strategy

[2]We chose 60 steps, *i.e.*, 300 minutes because most of the passenger-seeking trajectories in our taxi trajectory data end within 300 minutes.

[3]We follow the experiment settings in cGAIL [8] and TrajGAIL [9] to first select 50 "expert" taxi drivers' trajectories and utilize their demonstrations to train the IL approaches including BC, GAIL, cGAIL, and TrajGAIL for a fair comparison. Here "expert" taxi drivers have top earnings among all drivers in the taxi trajectory data.

TABLE I: Performance comparison with baseline models.

| | Ave. Seek Time (min) | $EE$ (Yuan/hour) | Success Rate |
|---|---|---|---|
| Humans | 23.69 | 54.70 | 1.00 |
| BC | 28.14 $\pm$0.42 | 47.66$\pm$1.06 | 0.98 |
| GAIL | 27.01 $\pm$0.53 | 50.30$\pm$0.98 | **0.99** |
| cGAIL | 26.20 $\pm$0.62 | 60.33$\pm$1.03 | 0.95 |
| TrajGAIL | 22.17 $\pm$0.28 | 62.38$\pm$0.48 | **0.99** |
| CQL | 18.22 $\pm$0.30 | 68.48 $\pm$1.31 | 0.98 |
| SAC | 15.51 $\pm$0.24 | 79.74 $\pm$1.33 | 0.97 |
| CAC (Ours) | **12.54 $\pm$ 0.19** | **100.75 $\pm$ 1.39** | **0.99** |

with a pair of policy-reward nets, it unifies and combines each human agent's decision strategy in one model (*i.e.*, one policy-reward nets pair ) via a condition latent. It provides a unifying framework to enable knowledge sharing among multiple drivers across various geographical locations.

- **Trajectory Generative Adversarial Imitation Learning (TrajGAIL) [9]** is also an IL method similar to GAIL; however, instead of modeling the human decision-making process as an MDP, TrajGAIL models it as a variable length MDP. It thus models the policy and reward networks to consider historical decision sequences as input.
- **Conservative $Q$ Learning (CQL) [18]** is an offline RL approach that leverages the Q-value to learn which state-action pair can maximize Q. The input to the Q network is a state $s$ and the output is the Q vector of size $\mathbb{R}^{|\mathcal{A}|}$.
- **Soft Actor-Critic [22]** is an actor-critic model in RL that composes of a policy network and a Q network. These two networks are iteratively updated during the training process so that the Q network can help the policy network to better understand the action probabilities given an input state $s$.

### C. Performances Comparison with Baselines

Tab. I shows the performance comparison, where the mean values and standard deviations are calculated from 10 trials with 10 different random seeds. Note that since we only select successful passenger-seeking trajectories for training and testing purposes, the human success rate is 100%. For all approaches, a higher success rate indicates a better model.

In Tab. I, our CAC model outperforms baselines in all evaluation aspects. The reason that IL approaches do not compete is that their performance is upper-bounded by human demonstrations [8]–[10], and are likely to be influenced by human constraints in terms of working preferences and proficiency. In general, offline RL algorithms, including CQL, SAC, and our CAC, outperform human strategies in both shortening average seeking time and improving earning efficiency. Specifically, the SAC model only achieves about 97% success rate with a longer seeking time and a lower earning efficiency in picking up a passenger. The reason is that SAC is vulnerable to extrapolation errors, and tends to choose out-of-distribution actions and produce reckless behaviors. Due to the reckless behavior, SAC trajectories are more likely to reach unseen states during training that are inaccessible in reality (*e.g.*, in the sea or outside of Shenzhen, China). On the other hand, the CQL model is able to deal with an extrapolation error problem resulting in a higher success rate. However, it learns a deterministic policy and only focuses on going to a specific goal state overlooking potential passengers in other goal states. It, therefore, struggles with decreasing the seeking time and improving the earning efficiency. Unlike them, CAC-learned strategies are conservative to avoid inaccessible and dangerous states, and are open to possibilities from various goal states.

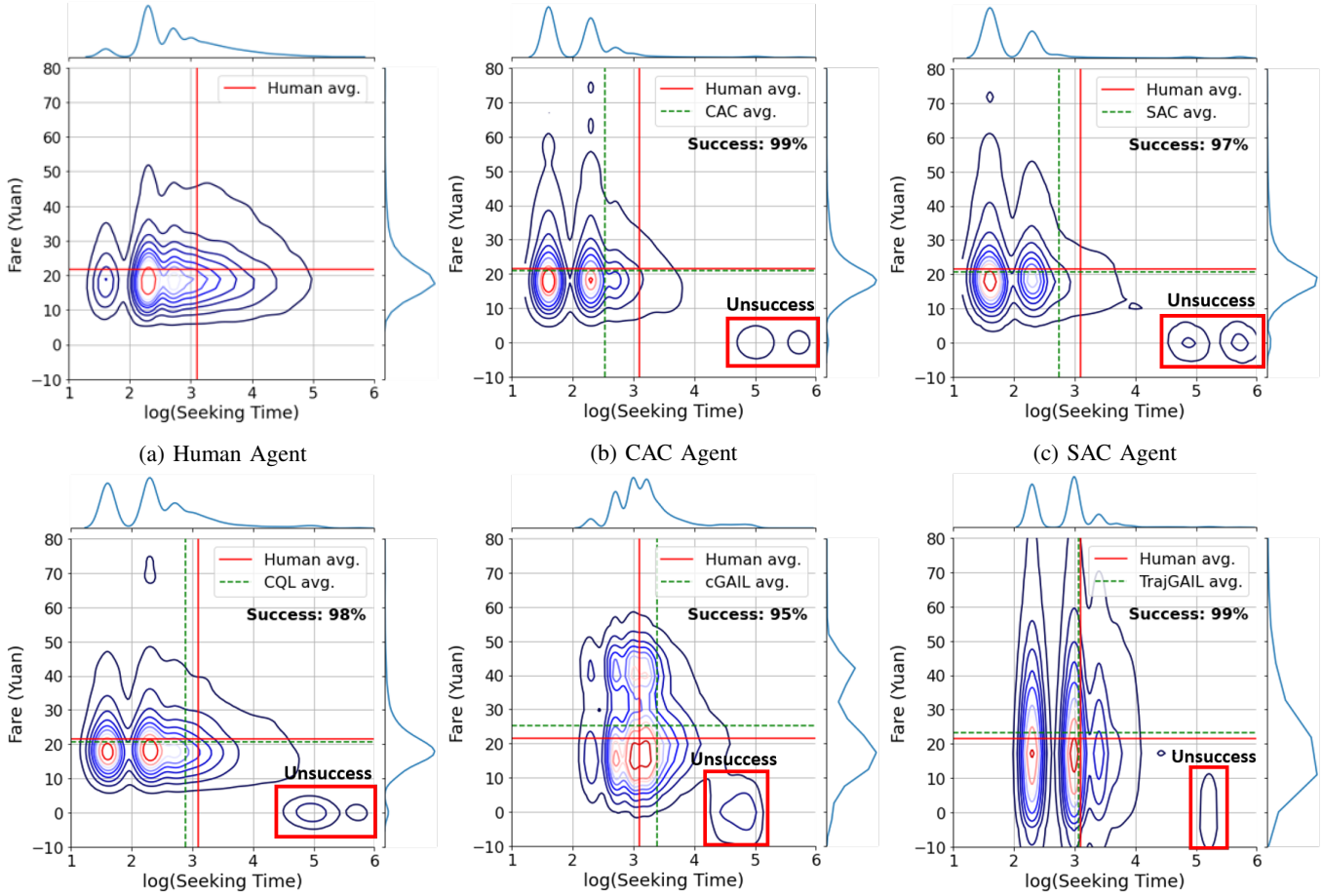
The CAC model stands out with its ability to reduce the passenger-seeking time by half, which in turn leads to a doubling of the earning efficiency in comparison to human agents. The great performance of CAC is due to the addition of a conservative regularizer to the TD error of the Discrete SAC algorithm. This regularization helps the model to learn the lower bound of the Q-value functions, leading to more stabilized action sampling in test time. In addition, the success rate of the CAC model is about 99% which is very close to the TrajGAIL model that is allowed to interact with the environment while learning expert drivers' strategies.

### D. Analysis of *Earnings* vs. *Seeking Time*

In this part, we give a detailed analysis of the generated trajectories from baseline models and ours with Fig. 4 in terms of earnings (or trip fare) and seeking time (in a $\log$ scale). Fig. 4 shows the distributions of fare (y-axis) and seeking time (x-axis) for all approaches, where the square plots show the joint distribution of fare and seeking time, the upper plots show the distribution of trajectory seeking times, and the right plots show the distribution of trajectory fares. All the distributions are estimated using the kernel density estimation [29]. We also plot the average seeking time and average fare as vertical and horizontal lines in the plots, where red lines represent average values from human demonstrations, and the green dashed lines represent those from baseline models and our CAC model. In general, a higher average fare with lower seeking time (*i.e.*, a higher horizontal line and a more left vertical line) represents a better model. In addition, in the square plots of Fig. 4b 4f, there are noticeable clusters shown in the red boxes. These clusters indicate unsuccessful trajectories where no passengers are found with 0 trip fare. Here, the left clusters in each image indicate unsuccessful trajectories that are truncated before a passenger is found. The right cluster centering around 5.7 in the x-axis (*i.e.*, seeking time reaches 300 minutes with $\log(300) \approx 5.7$) in each image represents unsuccessful trajectories that reach an inaccessible state.

**Earnings *vs*. seeking time for offline RL models.** Fig. 4 demonstrates that the passenger-seeking time for offline RL models is lower compared to that from human-generated trajectories. However, it is observed that the fare distributions, shown on the right side of Fig. 4a- 4d, are almost identical, with an average fare of approximately 21 Yuan. This is consistent with the objective function in Eq. (1) which seeks to maximize the cumulative discounted reward defined by prioritizing shortening passenger-seeking time.

The seeking-time distributions for SAC, CQL, and our CAC, models show two peaks, with the left peak of the SAC distribution slightly higher than that of the CAC and CQL algorithms. This suggests that the SAC policy is capable of finding passengers slightly faster than the other policies.




(a) Human Agent (b) CAC Agent (c) SAC Agent

(d) CQL Agent (e) cGAIL Agent (f) TrajGAIL Agent

Fig. 4: Distribution plots of seeking time (in log scale, upper plots), trip fare (right plots), and seeking time (in log scale) *vs.* trip fare for human agents, baseline models, and our CAC model. The red solid lines represent human agents' average seeking time and trip fare while the green dashed lines represent those of each algorithm. All distributions are estimated based on Kernel Density Estimation (KDE) [29].

However, the right cluster (centering around 5.7 in log seeking time) for the SAC model is denser than others. This highlights the inability of the SAC model to handle the extrapolation error problem which leads to reckless behaviors in inaccessible regions. This leads to a higher average seeking time and lower earning efficiency compared to the CAC policy.

We further analyze the results of CQL in Fig. 4d. Unlike the SAC model, which learns the Q-value function from the critic networks, the CQL model learns a conservative lower bound of Q-values for actions. As a result, we observe that in its unsuccessful box, the left cluster is much smaller than the one of SAC in Fig. 4c. This means that the CQL model could avoid unsafe or dangerous states and stay in the Shenzhen area. Moreover, the CQL model has a higher chance of waiting for a passenger in a grid cell similar to the human agent. That is, both the CQL and the human agent have a similar second peak in their seeking time distribution if Fig. 4b and 4a. This indicates that CQL learns and picks up the human trick to stay around the same location over time, *e.g.*, driving around a landmark like a park, shopping mall, or a transportation hub so that they are likely to pick up passengers and save fuel consumption at the same time. Therefore, the results in Fig. 4b indicate that the CAC model is successful in balancing seeking time and safety. By incorporating a conservative regularizer in the TD error minimization process, the CAC model is able to avoid visiting unsafe and inaccessible states while also reducing passenger-seeking time.

**Earnings *vs.* seeking time for IL models.** In terms of IL-based approaches, we only show the joint distribution plots of the cGAIL and TrajGAIL frameworks in Fig. 4e and 4f as other IL-based approaches demonstrate similar distribution patterns. Overall, IL-based approaches mimic human demonstrations and are able to pick up passengers with higher trip fares, *i.e.*, having higher average trip fares than offline RL approaches and average human agents. The reason is that IL models learn from selected human agents with high earnings (*i.e.*, top 50 among over 8,000 driers), and they are more likely to go to goal states with higher income. However, considering various human constraints in terms of decision preferences and proficiency, the passenger-seeking time does not necessarily be the shortest. Therefore, cGAIL and TrajGAIL learned policies have similar or even longer passenger-seeking time.

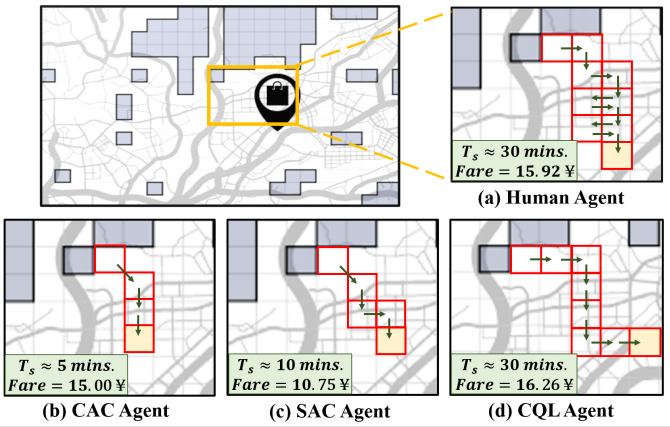



Fig. 5: Decision-making strategies by a human agent vs. offline RL policies.

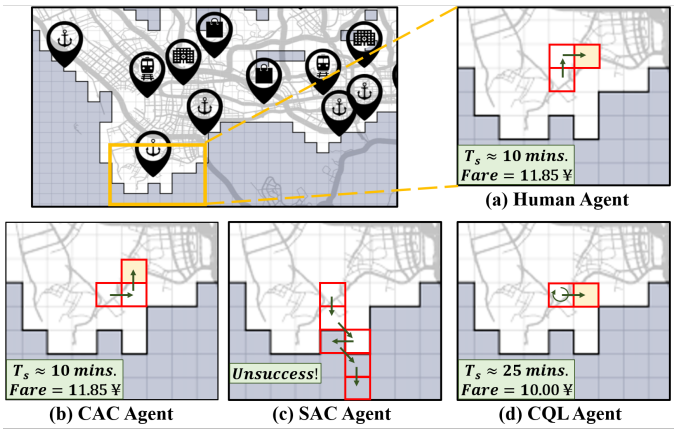



Fig. 6: Exploring the limitations of offline RL through examples of decision-making strategies

### *E. Understanding CAC Passenger-seeking Trajectories*

To better understand the learned passenger-seeking strategies from CAC, we present two example cases below.

**Decisions around a random location.** Fig. 5 shows the decision-making trajectories of a human agent and offline RL models. It can be observed that the human agent in Fig. 5(a) demonstrates a time-consuming back-and-forth movement, while the policies in Fig. 5(b)-(d) exhibit more direct and efficient strategies to reach a pickup state. The comparative analysis between offline RL models (*i.e.*, Fig. 5(b)-(d)) reveals that the CAC and SAC agents exhibit similar behavior for the initial three steps, with SAC subsequently selecting an additional action that results in reaching the other pickup state and longer seeking time than the CAC model that could successfully pick up a passenger within 5 minutes. Additionally, the results indicate that the CQL agent takes a comparable amount of time (*i.e.*, 30 minutes) as the human agent in seeking a passenger. Furthermore, the initial grid representation highlights that none of the policies choose to move towards the gray grid cell, which is inaccessible in the woods, and instead opt to head to the shopping mall.

**Decisions around a POI.** At another starting state close to a large area of inaccessible regions (specifically, sea-colored grey) and a harbor in Fig. 6, the SAC model struggles to locate a passenger and strays from the Shenzhen region due to extrapolation errors. Conversely, the CQL policy avoids this issue, but it moves to a terminal grid in 25 minutes. This implies that the CQL strategy favors remaining at the starting location until the state features change, which is a good and commercial strategy, considering its proximity to a harbor (as shown in Fig. 6). However, faster passenger acquisition could be achieved if the CQL moves to other states. Our CAC model successfully picks up a passenger in the same spatial-temporal grid as a human agent, while only making different intermediate actions.

## V. Related Work

**Urban computing** studies urban problems, such as traffic congestion, energy consumption, and pollution through data and analytics [30]. In taxi management, research works focus on dispatching and passenger-seeking strategies to improve driver performance and revenue [31]–[35]. A study [36] addresses passenger-seeking with direction recommendations. Multiple works [37]–[39] focus on reasonable order-dispatching strategies from the business platform level. All existing research focuses on optimal deterministic strategies, while our work focuses on practical, adaptive stochastic strategies.

**Human decision learning** is the process of understanding individual drivers' unique decision-making preferences. Studies like cGAIL [8] and TrajGAIL [7] leverage these preferences to learn about human strategies, leading to the characterization of good driver behaviors and improved strategies over time. TrajGAIL [9] also models the human decision-making process, but as variable-length Markov decision processes without directly learning individual preferences. In our framework, we also do not learn about human preferences, but instead, use diverse strategies from human drivers to inform actions based on spatial-temporal information.

**Offline reinforcement learning** is to train RL agents using historical data without interactions with the environment [11]–[18]. A challenge in offline RL is to prevent extrapolation errors while training on static data. The Conservative Q-Learning (CQL) algorithm [18] addresses this by using a conservative penalty term to avoid overestimating Q values. However, CQL uses a deterministic policy while our framework deals with a more complex environment with multiple goal states and multi-modal actions by integrating the conservative term with a discrete SAC model for optimizing multi-modal policy.

## VI. Conclusion

Our work presents the Conservative Actor-Critic (CAC) framework to improve the efficiency and quality of taxi service operations in self-driving vehicles. CAC utilizes static passenger-seeking data from diverse taxis to reduce passenger wait time and increase service revenue. It addresses the extrapolaration error problem by adding a conservative regularizer to the action-value function objective and incorporating entropy maximization into the policy objective. CAC reduces average passenger waiting time by half and doubles earning efficiency compared to human drivers, outperforming other offline RL baselines with a higher success rate.

## VII. Acknowledgement

Palawat Busaranuvong and Yanhua Li were supported in part by NSF grants IIS-1942680 (CAREER), CNS-1952085, and DGE-2021871.

## References

[1] S. Vougioukas, "Annual review of control, robotics, and autonomous systems," *Agricultural robotics*, vol. 2, no. 1, pp. 365–392, 2019.
[2] Waymo, "Waymo driver services." https://waymo.com/waymo-driver/, 2023.
[3] TuSimple, "Tusimple services." https://www.tusimple.com, 2023.
[4] Nuro, "Nuro services." https://www.nuro.ai/vehicle, 2023.
[5] L. Robotics, "Locus robotics." https://locusrobotics.com, 2023.
[6] I. Technologies, "Innoviz technologies." https://innoviz.tech, 2023.
[7] M. Pan, Y. Li, X. Zhou, Z. Liu, R. Song, H. Lu, and J. Luo, "Dissecting the learning curve of taxi drivers: A data-driven approach," in *Proceedings of the 2019 SIAM International Conference on Data Mining*, pp. 783–791, SIAM, 2019.
[8] X. Zhang, Y. Li, X. Zhou, and J. Luo, "Cgail: Conditional generative adversarial imitation learning—an application in taxi drivers' strategy learning," *IEEE Transactions on Big Data*, vol. 8, no. 5, pp. 1288–1300, 2020.
[9] X. Zhang, Y. Li, X. Zhou, Z. Zhang, and J. Luo, "Trajgail: Trajectory generative adversarial imitation learning for long-term decision analysis," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 801–810, IEEE, 2020.
[10] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.
[11] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.
[12] T. Shi, D. Chen, K. Chen, and Z. Li, "Offline reinforcement learning for autonomous driving with safety and exploration enhancement," *arXiv preprint arXiv:2110.07067*, 2021.
[13] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*, pp. 2052–2062, PMLR, 2019.
[14] Y. Wu, G. Tucker, and O. Nachum, "Behavior regularized offline reinforcement learning," *arXiv preprint arXiv:1911.11361*, 2019.
[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[16] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," *arXiv preprint arXiv:1706.01905*, 2017.
[17] T. Xu, Q. Liu, L. Zhao, and J. Peng, "Learning to explore via meta-policy gradient," in *International Conference on Machine Learning*, pp. 5463–5472, PMLR, 2018.
[18] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.
[19] Anonymous, "Cac code." https://github.com/XinZhang525/CAC, 2023.
[20] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[21] T. Calculator, "Taxi calculator." https://www.taxi-calculator.com/taxi-fare-boston/280, 2023.
[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.
[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[24] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.
[25] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," Oct. 2018. arXiv:1802.09477 [cs, stat].
[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[27] OpenAI, "Openai frozenlake environment." https://gym.openai.com/envs/FrozenLake-v0/, 2021.
[28] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems*, vol. 1, 1988.
[29] S. J. Sheather and M. C. Jones, "A reliable data-based bandwidth selection method for kernel density estimation," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 53, no. 3, pp. 683–690, 1991.
[30] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 5, no. 3, pp. 1–55, 2014.
[31] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 410–421, IEEE, 2013.
[32] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie, "T-finder: A recommender system for finding passengers and vacant taxis," *IEEE Transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2390–2403, 2012.
[33] J. Yuan, Y. Zheng, L. Zhang, X. Xie, and G. Sun, "Where to find my next passenger," in *Proceedings of the 13th international conference on Ubiquitous computing*, pp. 109–118, 2011.
[34] Y. Ge, H. Xiong, A. Tuzhilin, K. Xiao, M. Gruteser, and M. Pazzani, "An energy-efficient mobile recommender system," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 899–908, 2010.
[35] Y. Ge, C. Liu, H. Xiong, and J. Chen, "A taxi business intelligence system," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 735–738, 2011.
[36] H. Rong, X. Zhou, C. Yang, Z. Shafiq, and A. Liu, "The rich and the poor: A markov decision process approach to optimizing taxi driver revenue efficiency," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pp. 2329–2334, 2016.
[37] X. Tang, F. Zhang, Z. Qin, Y. Wang, D. Shi, B. Song, Y. Tong, H. Zhu, and J. Ye, "Value function is all you need: A unified learning framework for ride hailing platforms," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 3605–3615, 2021.
[38] X. Tang, Z. Qin, F. Zhang, Z. Wang, Z. Xu, Y. Ma, H. Zhu, and J. Ye, "A deep value-network based approach for multi-driver order dispatching," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1780–1790, 2019.
[39] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. Ye, "Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 905–913, 2018.

## Appendix

Maximizing Eq. (3) is equivalent to the objective below,

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t) + \zeta\mathcal{H}(\pi(\cdot|s_t))\right]. \quad (6)$$

Derivation: Since $\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{a\sim\pi}\left[\log(\pi(a|s_t))\right]$, we can substitute the expanded entropy term back into Eq. (6):

$$\begin{aligned}\pi^* &= \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t) - \zeta\mathbb{E}_{a\sim\pi}\left[\log(\pi(a|s_t))\right]\right] \\ &= \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t) - \zeta\mathbb{E}_{s\sim\mathcal{D},a\sim\pi}\left[\log(\pi(a|s))\right]\right].\end{aligned}$$

Defining the state-value function $V(s)$, and the action-value function $Q(s,a)$ as $V(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\right]$, and $Q(s,a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\middle| s_0=s, a_0=a\right]$, we can rewrite the above expression of $\pi^*$ as,

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}\left[Q(s,a) - \zeta\log(\pi(a|s))\right]\right].$$