Optimizing Task Offloading Migration Decisions: An Advantage Actor-Critic Approach

Jin Zhu, Amiya Nayak
School of Electrical Engineering and Computer Science
University of Ottawa, Canada

***Abstract*—The paper presents a study on task offloading migration decisions. The aim is to minimize the total latency of the offloaded computational tasks. Actor-Critic algorithm was employed to solve the problem, and real world taxi traces were used to train and test the model. The results indicate that the scheme of only migrating to neighboring servers performs well compared with the scheme of only migrating to any server. The model has good average rewards and converges faster, and has a degree of generalization ability.**

***Index Terms*—Mobile edge computing, service offloading, deep reinforcement learning, actor-critic approach.**

I. Introduction

Mobile user equipments (UEs), such as smartphones and laptops evolves rapidly, so do the applications running on them. Mobile applications are consuming more and more computational resources, which quickly drain the battery. One viable way of solving this problem is Mobile Cloud Computing (MCC), which offload the compute-intensive tasks to cloud servers [1]. Since the power supply of mobile devices is very limited, this technology reduces power consumption and extends battery life. However, offloading tasks to a conventional centralized cloud may result in significant network delay. In order to minimize the network delay, a technology called Mobile Edge Computing (MEC) was introduced. MEC moves servers from centralized data centers to the network edge, bringing computational resources closer to end-users, which greatly reduces the network delay and improves the Quality-of-Service (QoS).

Although MEC reduces the computational overhead of UEs, it also introduces some other expenses, such as the time and energy consumption for data transmission, the time consumption for remote task execution. It is necessary to weigh the trade-off between energy and time consumption. To address these issues, researchers have developed various algorithms to generate efficient offloading decisions [2].

UEs might be moving at high speed while using the offloading service, and they need to switch to another base station in the cellular network. In addition to that, although edge servers are closer to end-users, they have relatively limited computational resources compared to powerful distant centralized clouds that are located far away. This raises another question, computational tasks or their results have to be forwarded to other base stations. When, where and how to migrate the tasks is also a fundamental problem which have a significant impact on the system's performance [3] [4].

II. Related Work

A joint task offloading and migration approach based on reinforcement learning was presented by Wang et al. [5] for mobility-aware MEC networks. Their network model is a two-layer cellular network with a macro base station and many small cell network, each equipped with a small-cell base station (SBS), tasks can be locally calculated by mobile equipments or offloaded to the MEC server at SBS side. The goal is to minimize a utility function of time and energy cost of MEs. They solved the problem using Q-learning algorithm, whose performance is better than that of the genetic algorithm, randomly offloading algorithm and fully offloading algorithm.

Ding et al. [6] formulated two overhead optimization problems in a MEC network with mobile UEs: the problem of minimizing energy cost with constraints on resources and latency, and the problem of minimizing latency with constraints on resources and energy. Then they proposed and proved three optimal task offloading and service migration strategies when the service deployment strategy is given.

Han et al. [7] proposed a deep Q-learning network (DQN) algorithm to achieve fast and sub-optimal dynamic task offloading and migration solution. Tasks can be migrated to other edge servers or the centralized cloud. The goal is to minimize a weighted sum of the time and energy cost of MEs. The DQN algorithm was compared with three greedy decision-making algorithms, which demonstrated that DQN outperformed greedy algorithms.

Wang et al. [8] introduced a strategy for task offloading and migration in a generic multi-layer fog system which minimizes the probability of migration. The authors proposed a Gini coefficient-based algorithm for optimizing the offloading decisions and a genetic algorithm for optimizing computation resource allocation.

Wang et al. [4] modeled the service migration problem as a partially observable Markov decision process and proposed a deep recurrent off-policy actor-critic based algorithm. Experiments using real-world taxi traces demonstrated that the algorithm had an exceptionally good performance, much better than five different online baseline algorithms.

III. Methodology

A. Introduction to the A2C Algorithm

The sequential decision making problem is what reinforcement learning (RL) seeks to resolve. In RL, intelligent agents can learn from interactions with the environment. In each round, an agent would make a decision based on the current

state of the environment, then perform an action on the environment and receive a reward. The environment would transform to the next state. Reinforcement learning encompasses both policy-based techniques like REINFORCE and value-based techniques like Q-learning. Value-based methods focus on learning the value function, which takes a state as input and outputs the value of that state. Policy-based methods directly learn the policy function, which specifies the best action in different states [9] [10].

The Actor-Critic algorithm incorporates aspects of both policy-based and value-based approaches. Actor-Critic is a category of algorithms which is composed of two parts: the actor and the critic. The actor develops a better policy through interacting with the environment. The information gathered by the actor's interactions with the surroundings is used by the critic to learn a value function. Actor-Critic algorithms can achieve better sample efficiency and converge faster than pure policy-based and value-based methods. The state space of the migration decision problem is continuous, so tabular methods are unable to solve the problem. The Actor-Critic methods employ two neural networks, allowing it to handle problems with continuous state space. Actor-Critic algorithm is model-free; thus, it is very suitable for complex and unknown environments and more efficient.

The naive Actor-Critic algorithm is used in this research because of its simplicity, training stability and robustness. Advantage Actor-Critic (A2C) and other more complex forms of Actor-Critic are not the same. Actor-Critic has fewer hyperparameters and requires less parameter tuning techniques. It is also easy to implement. Despite its simplicity, it has good returns and convergence speed. We attempted to implement the A2C algorithm, and made an effort to tune the hyperparameters, but it didn't converge well. Moreover, it suffered from the problems of vanishing gradients and exploding gradients.

The actor's policy function's parameter update formula is:

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t)\delta(t)$$

where θ is the parameters of the policy function, α is the learning rate, $\pi_\theta(s_t, a_t)$ is the policy function, $\delta(t)$ is the Temporal Difference (TD) error [10]. The update formula for the parameters of the critic's model is:

$$\delta = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

where R_{t+1} is the reward, γ is the discount factor, $Q(S_{t+1}, A_{t+1})$ is the estimated value of the state-action pair at the next time step, $Q(S_t, A_t)$ is the estimated value of the state-action pair at the current time step.

B. Task Offloading and Migration Model

The model is adopted from Wang et al.'s work [4]. consists of U mobile user equipments and M base stations, each base station is equipped with a MEC server. Only offloaded tasks are taken into consideration in this model. The user equipment is always connected to the nearest base station and the offloaded task is transmitted to that base station. The associated MEC server is required to make a decision about whether and where to migrate the task. The local server of a UE is defined as the nearest server to the UE and the serving server is defined as the server handling the offloaded task.

The QoS is measured in terms of latency which includes computation, communication and migration delays. The migration delay is defined as:

$$B(u; d_t) = \begin{cases} 0, & \text{if } dt = 0 \\ \frac{data_t^s(u)}{\eta_t^m} + \sigma_t^m dt, & \text{if } dt \neq 0 \end{cases}$$

where t is the current time slot, d_t is the distance of migration, σ_t^m is the migration delay coefficient, $data_t^s(u)$ denotes the size of service, and η_t^m is the migration bandwidth. The computation delay is given by:

$$D(a_t) = \frac{c_t}{\frac{c_t}{w_t(a_t)+c_t} \cdot f(a_t)} = \frac{w_t(a_t) + c_t}{f(a_t)}$$

where c_t is the CPU cycles needed for processing offloaded task, $w_t(a_t)$ represents the serving server's workload, and $f(a_t)$ denotes the server's computational capacity. The MEC servers all employ a weighted resource allocation strategy. The communication delay is given by:

$$\rho_t = \omega \log_2 \left(1 + \text{SNR}(u, m_t^l(u))\right)$$

where ρ_t denotes the uplink transmission rate between UE and its local server, ω represents the uplink bandwidth, and $\text{SNR}(u, m_t^l(u))$ corresponds to the SNR. The SNR is defined as:

$$\text{SNR}(u, m_t^l(u)) = \frac{p_u \left| g_t(u, m_t^l(u)) \right|^2}{\omega \mathcal{N}}$$

where p_u is the transmission power of UE, $\mathcal{N}$ is the power spectral density of white Gaussian noise, and $g_t(u, m_t^l(u))$ corresponds to the channel gain. The access delay at time t is:

$$R(u) = \frac{data_t^o(u)}{\rho_t}$$

The backhaul delay at time t is:

$$P(u, y_t) = \begin{cases} 0, & \text{if } y_t = 0 \\ \frac{\text{data}_t^o(u)}{\eta_t^{bh}} + \sigma_t^{bh} y_t, & \text{if } y_t \neq 0 \end{cases}$$

where y_t is the hop distance between local server and serving server, σ_t^{bh} is the backhaul delay coefficient, and η_t^{bh} is the backhaul bandwidth.The communication delay is defined as:

$$E(u, y_t) = P(u, y_t) + R(u)$$

The utility function is defined as:

$$\min_{a_0, a_1, \ldots, a_T} \sum_{t=0}^{T} \left(E(u, y_t) + B(u, d_t) + D(a_t) \right),$$
$$\text{s.t. } a_t \in \{1, 2, \ldots, M\},$$
$$\forall u \in \{1, 2, \ldots, U\}$$

C. Application of Actor-Critic in Task Migration

1) Environment: The environment is mostly adopted from the work of Wang et al. [4] with some modifications and improvements. In this environment, a total of 64 base stations are arranged in an 8 by 8 matrix, with each base station spaced 1 kilometer apart. This matrix is placed at the downtown area of Rome and San Francisco, where the datasets come from. In each time slot, every MEC server randomly generates a CPU cycle required for the workload of that server, and every taxi

cab also randomly generates a task data volume and required CPU cycle in a given range. The CPU frequencies of all servers are same and constant. The tasks are offloaded to the nearest server, waiting to be migrated or computed. Wireless upload rate is calculated according to distance between the UE and its base station. The backhaul delay coefficient is constant, but migration delay coefficients are randomly generated in a given range. The hop number between servers is defined as the Manhattan distance. By reason of the large data volume and irregular trace length of the datasets, all traces are truncated to a fixed length, which makes the data of each trajectory neat and tidy.

2) Action: In RL, action denotes the action an agent takes which affects the environment. The action in this model is to choose which server to migrate the tasks to. In Wang et al.'s research [4], the tasks can be migrated to any server, thus the action space is 64. We used a simplified migration scheme: servers periodically exchange their workloads with one-hop neighbors, and reduce the action space to three, which include 'do not migrate', 'migrate to local server', and 'migrate to neighboring server with least workload'.

Algorithm 1 ActionSpace_3

procedure MIN_WORKLOAD_NBR
  $nbrs \leftarrow \{my\_id + base\_station\_column, my\_id - base\_station\_column, my\_id + 1, my\_id - 1\}$
  for each $element$ **in** $nbrs$ **do**
    if $element < 0$ **or** $element > base\_station\_count$ **or** $ManhattanDistance(my_id, element) > 1$ **then**
      remove $element$ from $nbrs$
    end if
  end for
  $min\_workload \leftarrow INF$
  for each $element$ **in** $nbrs$ **do**
    if $element.workload < min\_workload$ **then**
      $min\_workload \leftarrow element.workload$
      $min\_nbr \leftarrow element$
    end if
  end for
  return min_nbr
end procedure

procedure STEP($action$)
  if $action = no\_migrate$ **then**
    pass
  else if $action = migrate\_min\_nbr$ **then**
    $serving\_server \leftarrow$ min_workload_nbr()
  else if $action = migrate\_local$ **then**
    $serving\_server \leftarrow local\_server$
  end if
  ...
end procedure

3) State: In RL, state is a complete description of the world and does not hide any information, observation is a partial description of the state and may omit some information [9]. Using full state as observation is too complex and often not realistic. When the agent can only see partial observations, this environment is called partially observable. In such cases, RL is often modeled as a partially observable Markov decision process [9]. In this migration decision problem, collecting the full state from the whole system is also unrealistic. Moreover, this environment demands high timeliness for data. Data transmitted over the network may be outdated. As a result, the observation only contains a few data items that are easily accessible from the local server. The observed data items of each UE are: local server index, transmission rate, data volume and required CPU cycles of tasks.

4) Reward: As stated in Section III, the utility function is to minimize the sum of migration, computation, and communication delays. We define the reward as the negative of sum of delays. To improve the training efficiency, 100 environments are run concurrently.

IV. Experimental Design

A. Datasets

Two real-world datasets are used to train and test the model. They are CRAWDAD ROMA/TAXI [11] and CRAWDAD EPFL/MOBILITY [12]. They include mobility traces of taxis in San Francisco, USA, and Rome, Italy, respectively. Each data item is composed of driver ID, time, and GPS coordinates. The time interval between adjacent data points may be too short, and the GPS coordinates' distance may be too close, so data items are taken at intervals. The data items located outside the 8 km × 8 km downtown area are removed. The trajectories of taxis reflect the true shape of urban roads and the real patterns of vehicle movement, making them highly suitable for this experiment. In both datasets, 120 traces have been used as training and 30 traces as testing sets.

B. Parameter Selection & Evaluation Metrics

The parameter values (in Table I) of the simulated environment are the same as [4]. The hyperparameters of A2C are given in Table II. The algorithm is evaluated with average reward, convergence speed and generalization ability.

TABLE I
Enviroment Parameters [4]

Simulation Parameter	Value
Total MEC servers (M)	64
Computational capacity (f) of each MEC server	128 GHz
Upload rate (ρ_t)	$\{60, 48, 36, 24, 12\}$ Mbps
Bandwidth (η_t) of backhaul network	500 Mbps
Coefficient (σ_t^{bh}) of backhaul delay	0.02 s/hop
Coefficient (σ_t^{m}) of migration delay	$U[1.0, 3.0]$ s/hop
Data size ($datas_t^s(u)$) of the service data	$U[0.5, 100]$ MB
Data size ($data_t^o(u)$) of each offloaded task	$U[0.05, 5]$ MB
Processing density of an offloaded task, κ	$U[200, 10000]$ cycles/bit
Task arrival rate (λ_p^u) for user	$p2$ tasks/slot
Task arrival rate (λ_p^s) for MEC server	$U[5, 20]$ tasks/slot
Trace interval	Rome:12, SF:3

TABLE II
A2C HYPERPARAMETERS

Hyperparameter	Value
Learning rate (actor)	1e-3
Learning rate (critic)	1e-2
Discount factor (gamma)	0.8
Number of steps (T)	100
Number of epochs	1000
Batch size	120
Neural network hidden dimension	256
Optimizer	Adam

V. RESULTS AND ANALYSIS

A. Performance Comparison

Figures 1 and 2 show the convergence curve with the setup mentioned above on Rome dataset. 'ma rewards' means moving average of 10 rewards. Figures 3 and 4 show the convergence curve with the setup mentioned above on San Francisco dataset. Figures 1 and 3 have action space 3, which means tasks can choose among migrating to local server, migrating to neighboring server with lease workload, and do not migrate. Figures 2 and 4 have action space 64, which means task can migrate to any server. Table III shows the episodes which the relative change rate of moving average rewards drops to less than 0.05%.

TABLE III
NUMBER OF EPISODES WHEN THE MODELS CONVERGE

Dataset	Action Space	Number of episode
Rome	3	32
Rome	64	53
SF	3	35
SF	64	48

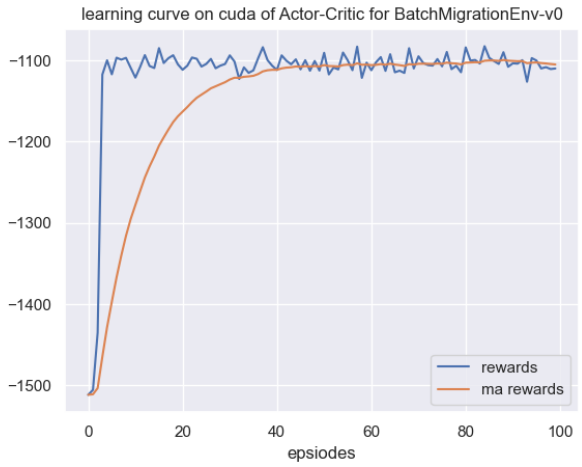

Fig. 1. Training on Rome Data, Action Space = 3

Figure 5 shows the comparison of average rewards of migration with different action schemes and that of no migration scheme (tasks are always computed at the server they are

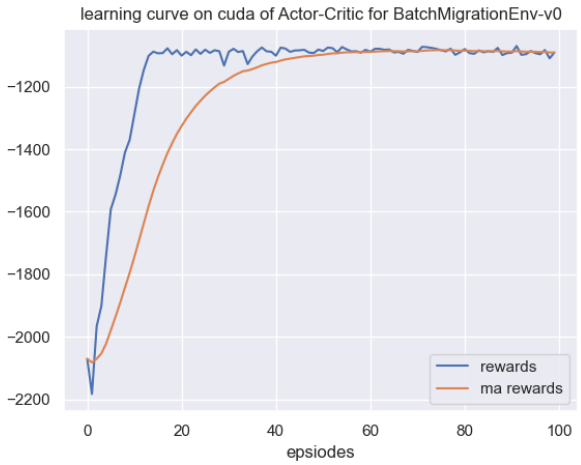

Fig. 2. Training on Rome Data, Action Space = 64

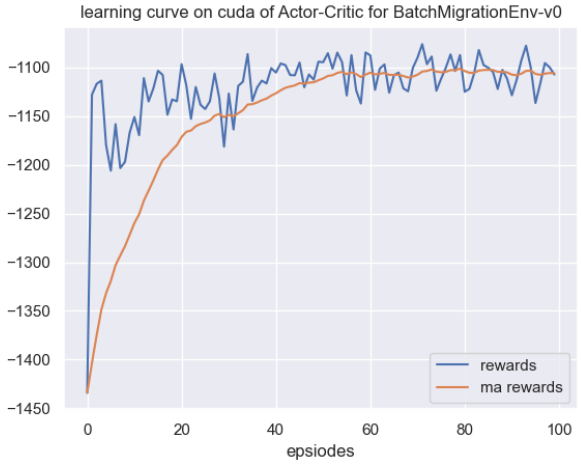

Fig. 3. Training on SF Data, Action Space = 3

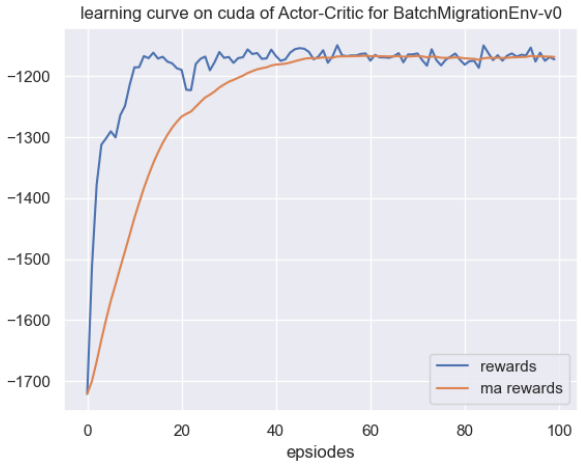

Fig. 4. Training on SF Data, Action Space = 64

offloaded to). Figure 5 also shows the generalization abilities of models trained with the two datasets. Model trained with Rome is tested using San Francisco dataset and vice versa.

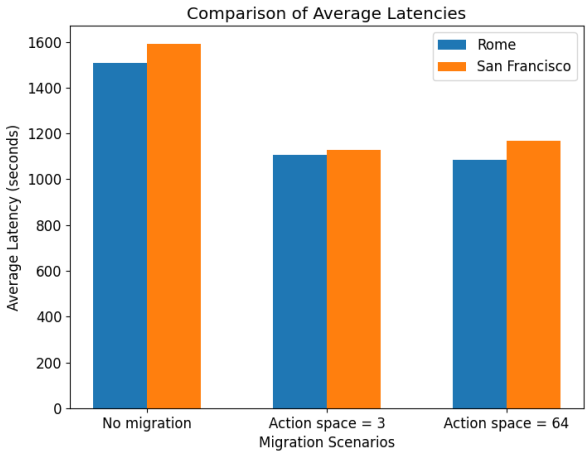

Fig. 5. Comparison of Average Latencies

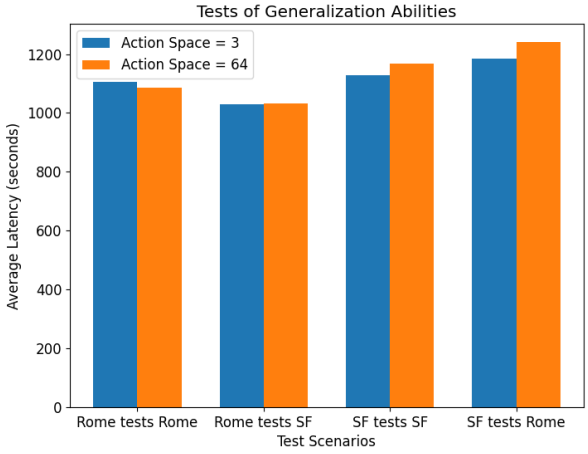

Fig. 6. Tests of Generalization Ability

B. Interpretation of Experimental Results

By comparing the convergence curve of schemes with different action spaces in Figure 1 to Figure 4 and Table III, we found that migrating only to neighboring servers makes the model to converge faster. This conclusion holds true for both datasets. However, the convergence curve of migrating only to neighboring servers exhibits more oscillations. This means reducing the action space to 3 speeds up convergence, saving computational resources and training time. By comparing the average latencies of no migration and schemes with different action spaces, we found that the average rewards of two schemes are nearly identical. This implies that we can maintain almost the same performance while reducing training overhead. Experiments of testing models trained on the other dataset demonstrate that these two models have a certain degree of generalization ability. Through this experiment, we found that the model does not have to be trained on the data of the city it is used in. Instead, we can train the model on a limited number of cities and apply the model to numerous other cities, which can greatly reduce the cost associated with training.

VI. Conclusion

In this research, we proposed a scheme of migrating to neighboring servers and train a Actor-Critic model to make the migration decisions. The findings are as follows: transferring the destination to 1-hop neighboring servers does not negatively effect the reward significantly, but it accelerates the convergence speed of the model and reduce the training overhead; therefore, it is a good a good alternative method. The Actor-Critic algorithm performs well in this environment, since it has good rewards, fast convergence speed and shows good degree of generalization ability.

References

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[2] W. Zhan, C. Luo, G. Min, C. Wang, Q. Zhu, and H. Duan, "Mobility-aware multi-user offloading optimization for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3341–3356, 2020.

[3] A. Nayak and I. Stojmenovic, *Handbook of applied algorithms: Solving scientific, engineering, and practical problems*. John Wiley & Sons, 2007.

[4] J. Wang, J. Hu, G. Min, Q. Ni, and T. El-Ghazawi, "Online service migration in mobile edge with incomplete system information: A deep recurrent actor-critic learning approach," *IEEE Transactions on Mobile Computing*, vol. 22, no. 11, pp. 6663–6675, 2023.

[5] D. Wang, X. Tian, H. Cui, and Z. Liu, "Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware mec network," *China Communications*, vol. 17, no. 8, pp. 31–44, 2020.

[6] Y. Ding, C. Liu, K. Li, Z. Tang, and K. Li, "Task offloading and service migration strategies for user equipments with mobility consideration in mobile edge computing," in *2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2019, pp. 176–183.

[7] Y. Han, X. Li, and Z. Zhou, "Dynamic task offloading and service migration optimization in edge networks," *International Journal of Crowd Science*, vol. 7, no. 1, pp. 16–23, 2023.

[8] D. Wang, Z. Liu, X. Wang, and Y. Lan, "Mobility-aware task offloading and migration schemes in fog computing networks," *IEEE Access*, vol. 7, pp. 43 356–43 368, 2019.

[9] J. J. Qi Wang, Yiyuan Yang, *Easy RL: Reinforcement Learning Tutorial*. Beijing: Posts Telecom Press, 2022. [Online]. Available: https://github.com/datawhalechina/easy-rl

[10] Y. Y. Weinan Zhang, Jian Shen, *Hands-on Reinforcement Learning*. Beijing: Posts Telecom Press, 2022. [Online]. Available: https://hrl.boyuai.com/

[11] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi, "Crawdad roma/taxi," 2022. [Online]. Available: https://dx.doi.org/10.15783/C7QC7M

[12] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "Crawdad epfl/mobility," 2022. [Online]. Available: https://dx.doi.org/10.15783/C7J010