# Appendix

## A Full Algorithms

In this supplementary section, we explicitly define how to train an AV by SAPO in Algorithm 1. During the training process, we reset the environment and initialize the global state (Line 1). For each AV $m$, we first check if the status is active (Line 4-5). Then, through Interactive Attention, it observes other AVs and focuses on the interactive AV $p$ as the interacting object (Line 6-7). Based on the Dec-POMDP, the environment transits to the next step when all of AVs have made decisions according to their policies (Line 8-10). Meanwhile, the AV $m$ receives its individual reward $r_t^m$ and generates the social-aware reward by estimating the current SP to the AV $p$ (Line 11-12). The episode will be terminated if: (a) all of the AVs have arrived at their destinations, (b) some of them collide with each others, or (c) the timestep has reached the given maximum length $T$. Note that we can arbitrarily set a finite horizon to accelerate the exploring process. This is because our policy actually do not have any time dependencies (i.e., $T = \infty$), which is suitable in general. Finally, based on PPO, we sample mini-batches of experience data and update parameters of the policy and value networks (Line 13).

**Algorithm 1:** Socially-Attentive Policy Optimization (for an AV $m$)

```
1  while not coveraged do
2      Reset the environment and initialize the global state as $\boldsymbol{s}_0$
3      for $t \leftarrow 1$ to $T$ do
4          if collide or reach the goal then
5              continue
6          Get observation $\boldsymbol{o}_t^m$.
7          Focus on the interactive AV $p$ by Eqn. (7)
8          Sample action: $\boldsymbol{a}_t^m \sim \pi^m(\cdot|\boldsymbol{o}_t^m)$.
9          Synchronize actions of active AVs.
10         Step environment: $\boldsymbol{s}_{t+1} \sim \Pr(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$
11         Get individual reward $r_t^m$.
12         Generate social-aware reward by Eqn. (9).
13     Update parameters of policy $\pi^m$ and value $V^m$ networks, using Eqn. (4) and Eqn. (5).
```

For implementation that matters, we utilize RLlib [33], a powerful distributed training architecture for RL training. To make sure our trained policy can adapt a diverse set of environmental settings, we set different random seed in each actor.

## B Experimental Settings in SMARTS

Implemented in SMARTS, the detailed formulation of the Dec-POMDP in our MASD system is given as follows:

**State and observation space.** The global state $\boldsymbol{s}_t$ is a vector which concatenates information of each AV, including their 2-D position, heading direction, moving speed, yaw rate and steering angle at timestep $t$. Then, each AV $m$'s observation $\boldsymbol{o}_t^m$ is a kinematic observation of nearby vehicles, which has the identical size as the global state $s_t$. That is, when other AVs are out of the observable range of an AV $m$, their corresponding information in $\boldsymbol{o}_t^m$ will becomes zero as blind.

**Action space.** Based on the basic low-level controller [4] of the vehicle kinematics (i.e., throttle and steering angle), we focus on the available meta-actions in MASD systems. For example, when AVs are moving in different lanes at crossroads, the action space $\mathcal{A}$ consists in speeding up, keeping the speed and slowing down.

**Reward function.** $r_t^m$ measures the performance of the AV $m$ at timestep $t$, represented by the closeness to the given destination. The occurrence of collision and timeout will bring a large penalty to AV $m$.

**Randomness.** The randomness of the simulation environment is an important issue for RL training. If the initial state keeps the same during the training and testing process, it is easy to search how to achieve the optimal policy, which is far from real-world application. To this end, we utilize a random seed to represent a unique permutation of several important settings, including: (a) the entrance from which the vehicle enter; (b) the initial speed of the vehicle; and (c) the given route which the vehicle is following, such as moving straight, changing lanes and turning left/right. We choose several valid settings where vehicles obey the road rules and save the relevant seeds as the training and testing cases.

## C Implementation Details of Baselines

### C.1 Iterative Best Response

Inspired by the concept of *best response*, Iterative BR [34] is a numerical method to compute a Nash equilibrium in a multi-player game, where players' strategies are sequentially and iteratively updated until all players have converged on their BR. In our experiments, we define an AV's BR by choosing the best action from its action space, which achieves the maximum individual reward. Note that calculating the reward function needs to access the simulation of other AVs' behaviors. Iterative BR is commonly used in solving finite-time horizon differential games. However, one drawback of this method is that the iterative optimization process may be computationally inefficient with long solution times when dealing with too many players. Thus, for fair comparison, we also add a baseline which converts multi-player games to two-player games and takes advantages of BR.

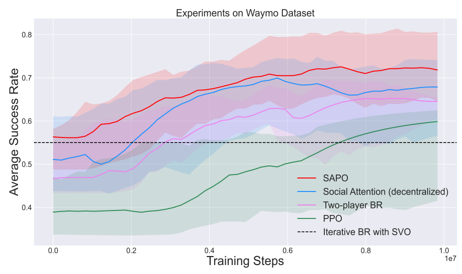
### C.2 Online Estimation of SVO

Our reference baseline [20] has been widely used in self-driving scenarios. It estimates SVOs at each timestep by sampling from a histogram filter, based on the principle of maximum entropy, i.e., the probability of one candidate reward parameter is proportional to the probability of the observed trajectories under this reward parameter. To implement Iterative BR with SVO, we follow [20] and give the following algorithm routine of how to make social-aware decisions and estimate SVOs in real time, executed by an AV in an MASD system. First, we give the SVO of the ego AV (i.e., it only knows the ground truth of its own SVO) and initialize the estimation of other AVs' SVOs with a uniform distribution over the SVO ring. Then, at each timestep, the ego AV perceives its own observation and observes other AVs' trajectories. Next, based on maximum entropy likelihood function, we get a likelihood distribution over SVOs from the observed trajectories. We integrate the likelihood function into a recursive filtering framework (e.g., Histogram filter used in our work), updating the current posterior distribution over SVOs. Finally, given an estimation over the SVOs of other AVs in the MASD system, the ego AV can use the KKT approach to predict other AVs' behaviors and take the BR. After interaction, each AV steps into the next timestep and do online estimation again before making decisions.

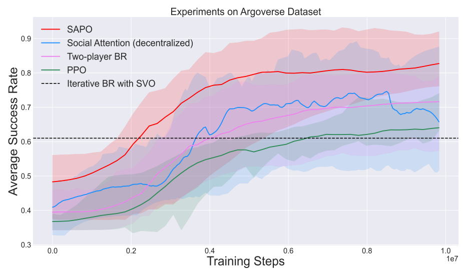
## D Self-Driving Application in Environment with Real Data

In this section, we underline the relevance of this paper to robotics by showing the perspective of integrating SAPO into a more realistic autonomous driving system." Similar to CoPO [15] using Metadrive [3], this paper mainly evaluate the performance of SAPO and other baselines in an MASD system, implemented by SMARTS [4]. However, CoPO is mainly used for centralized training and decentralized execution of multiple AVs, based on the total reward of all controllable vehicles. This limits the application of driving AVs in real traffic of uncontrollable vehicles. Differently, SAPO is based on decentralized training, where we do not utilize any global information like the states or the total of individual reward. That is, SAPO can be appplicable not only in some MASD systems where AVs are trained together, but also in several single-agent self-driving environments where an ego AV is controlled to learn better interaction with all real vehicles.

To this end, we evaluate SAPO in two self-driving scenarios[1][2], where the traffics are based on Waymo Motion Dataset [31] and Argoverse Dataset [32], respectively. Supported by Metadrive [3], we replay the trajectory of each uncontrollable vehicle based on the dataset but marginalize them with the timeliness of the ego AV's movement, within several timesteps in an episode. The problem formulation is modified from our MASD system based on SMARTS (see Appendix B), where the episode will be terminated if: (a) the ego AV has arrived at its destination, (b) the ego AV collides with other vehicles. We compare SAPO with several baselines based on independent control and the results are shown in Figure 8. We observe that SAPO can still outperform the baselines with the help of Interactive Attention and SPs. However, compared to the results in MASD systems (see Fig. 7), the performance are more unstable. This is partially because of the diversity and complexity of the real trajectories. Besides, there exist some open questions of replaying the RL-based vehicle and data-driven vehicles together in an simulation environment. For example, the success rate may be improved if we consider the mixed policy of SAPO and other conventional driving model like IDM, which is a promising research direction but beyond the scope of this paper.



(a) Waymo



(b) Argoverse

Figure 8: Self-driving environment of driving in real traffic: (a) Waymo Motion Dataset. (b) Argoverse Dataset.

[1]See: https://github.com/metadriverse/metadrive/blob/main/metadrive/envs/real_data_envs/waymo_env.py
[2]See: https://github.com/metadriverse/metadrive/blob/main/metadrive/envs/real_data_envs/argoverse_env.py

## E Hyperparameter Settings

In Table 1, Table 2 and Table 3, we report all hyperparameters used in the Bottleneck, SMARTS and real environment, respectively.

Table 1: Hyperparameter settings in Bottleneck.

| **Hyperparameter** | **Value** |
|---|---|
| Clip Gradient Norm | 1 |
| $\gamma$ | 0.99 |
| $\lambda$ | 0.95 |
| Learning rate | 1x$10^{-4}$ |
| Number of minibatches | 4 |
| Number of optimisation epochs | 4 |
| Number of parallel actors | 4 |
| Optimisation algorithm | ADAM |
| Rollout length | 64 |
| Use Generalized Advantage Estimation | True |

Table 2: Hyperparameter settings in SMARTS.

| **Hyperparameter** | **Value** |
|---|---|
| Clip Gradient Norm | 1 |
| $\gamma$ | 0.99 |
| $\lambda$ | 0.95 |
| Learning rate | 1x$10^{-5}$ |
| Number of minibatches | 10 |
| Number of optimisation epochs | 4 |
| Number of parallel actors | 16 |
| Optimisation algorithm | ADAM |
| Rollout length | 100 |
| Use Generalized Advantage Estimation | True |

Table 3: Hyperparameter settings in real environment with datasets.

| **Hyperparameter** | **Value** |
|---|---|
| Clip Gradient Norm | 1 |
| $\gamma$ | 0.99 |
| $\lambda$ | 0.95 |
| Learning rate | 1x$10^{-5}$ |
| Number of minibatches | 10 |
| Number of optimisation epochs | 4 |
| Number of parallel actors | 32 |
| Optimisation algorithm | ADAM |
| Rollout length | 250 |
| Use Generalized Advantage Estimation | True |