Autonomous Driving Decision Making and Controlling Based on Federated Deep Reinforcement Learning

Zhejian Xu
Faculty of Information Technology
Beijing University of Technology
Beijing, China
azha@azha.fun

Zhiqing Huang
Faculty of Information Technology
Beijing University of Technology
Beijing, China
zqhuang@bjut.edu.cn

Abstract—Deep reinforcement learning has shown remarkable decision-making and control capabilities in the field of autonomous driving, delivering impressive performance in real-world scenarios. However, constructing high-quality policies with deep reinforcement learning becomes challenging when dealing with small state feature spaces and limited training data. In privacy-sensitive applications, the exchange of data between agents is restricted due to privacy concerns, necessitating the exploration of alternative approaches. This paper proposes a federated deep reinforcement learning framework by combining federated learning with deep reinforcement learning. We discuss the learning strategy and federated policy within this framework. Furthermore, we investigate the improvement of multi-scenario adaptation by training models with different agents using federated learning across multiple scenarios, compared to traditional deep reinforcement learning models trained in a single scenario. Through experiments conducted on the TORCS platform, we validate the adaptability of the federated deep reinforcement learning model to complex scenarios. Additionally, we demonstrate enhanced driving performance by training vehicles in the task of lane keeping on a race track while ensuring data privacy.

Index Terms—autonomous driving, deep reinforcement learning, federated learning.

I. Introduction

The advancements in deep reinforcement learning (DRL) have greatly enhanced decision-making and control technologies in autonomous driving. However, effective training of DRL agents often requires a significant amount of high-quality scenario data. In commercial and industrial scenarios, privacy constraints prevent DRL agents from using private data for training. While encrypting the training data can protect privacy, it often leads to a loss of relevance to the specific scenario, resulting in inaccuracies in trained models.

To overcome these challenges, this paper proposes an efficient, privacy-secure, and accurate end-to-end training model that leverages federated learning. By preserving the privacy of training data in autonomous driving cars while making optimal use of computational resources, this approach improves adaptation to complex scenarios and enhances driving performance. Consequently, this research has broad real-life applications. Multiple autonomous driving clients can collaborate, sharing their resources without exchanging private data, to train more powerful deep reinforcement learning models.

II. Background and Related Work

Numerous studies have been conducted on implementing deep reinforcement learning for end-to-end autonomous driving decision-making and control models. For instance, Loiacono et al. achieved overtaking maneuvers on the TORCS platform using the Q-Learning algorithm [1]. W. Xia et al. improved the Q-Learning algorithm by incorporating demonstration data, resulting in a 72% reduction in training time and a 32% stability improvement on the TORCS platform [2]. H. Chae et al. employed the Deep Q-Learning Network algorithm to train vehicles for braking in urban roads under the PreScan platform [3]. A. E. Sallab et al. used the DQN algorithm to make driving action decisions and simple interactions with other vehicles in the TORCS environment [4]. However, most of the current reinforcement learning research focuses on discrete action output algorithms, which differ significantly from continuous driving actions in real-world scenes. In 2016, Google proposed the deep deterministic policy gradient algorithm [5] which is more suitable for decision-making in scenarios involving continuous actions (e.g., actions within the interval [0, 1] with smooth continuous variations), resembling real-world driving actions. Nonetheless, considering the robust requirements of autonomous driving decision-making and control models, extensive experimentation in different scenarios is necessary to gain sufficient experience. In complex scenarios, the number of training steps for the model often exceeds millions, and setting super parameters or defining reward functions may take several days [6].

The concept of federated learning was introduced by Google in 2016 [7] to address the challenge of updating models locally by Android phone end-users. In recent years, federated learning has gained prominence in artificial intelligence research due to its ability to preserve data owner privacy and aggregate models from feder-

ated participants into more effective federated models. Yang et al. categorized federated learning models into horizontal federated learning, vertical federated learning, and federated transfer learning, exploring their respective application scenarios [8]. A. Hard et al. proposed joint training of an input method prediction model using local data from multiple Android cell phone terminals [9]. Y. Zhao et al. developed a federated model for scenarios where data from different owners are not independent and identically distributed, proposing a strategy of sharing a portion of the data among federated servers to improve model accuracy [10]. The concept of federated learning also finds relevance in autonomous driving and machine automation fields. Microsoft suggested setting up parameter servers on cloud servers and distributing training tasks to agent nodes for separate training, significantly reducing training time while ensuring accuracy [11]. X. Liang et al. proposed a federated transfer reinforcement learning model that addresses the practical requirements of transferring autonomous driving simulation environments to real environments, resulting in improved driving distance and reduced collisions of remote control cars by transferring the model's state space and action space [12]. B. Liu et al. proposed a lifelong learning system architecture for cloud robots based on federated learning, analyzing knowledge fusion and transfer schemes for different data types, effectively improving the learning efficiency of client robots [13] [14].

The aforementioned studies demonstrate the progress made by existing deep reinforcement learning algorithms in autonomous driving decision-making and control. However, to achieve self-driving action decisions in multiple scenarios and safer action decisions within the same scenario, deep reinforcement learning-based action decision-making and control methods require a larger amount of data. Federated learning has emerged as a promising approach that can leverage data without compromising privacy, enable knowledge sharing, improve the accuracy of federated models, and enhance training efficiency. The combination of deep reinforcement learning and federated learning in the context of autonomous driving holds great significance. Collaborative efforts among multiple clients offer the potential to train a powerful model with adaptability across various scenarios and improved driving performance.

III. PROPOSED FRAMEWORK

This section presents the application of the Federated Deep Reinforcement Learning (FDRL) framework for decision-making and control of autonomous driving cars on the TORCS platform. It is important to highlight that the FDRL framework proposed in this paper can be applied to various deep reinforcement learning methods.

A. DDPG Algorithm

The Deep Deterministic Policy Gradient (DDPG) algorithm is well-suited for obtaining continuous action outputs based on state information, making it suitable for real-world driving actions. In this paper, we leverage the DDPG algorithm to develop an autonomous driving system that encompasses decision-making and control. The system takes pivotal information into account, including vehicle status and environmental data, as input during training on the TORCS platform. It then generates continuous control actions to enable driving decisions. Fig. 1 illustrates the structure of our autonomous driving decision and control system based on the DDPG algorithm.

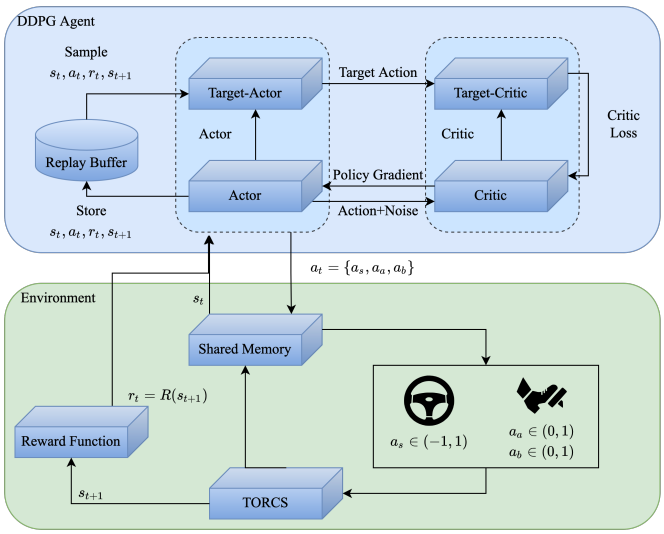


Fig. 1. The autonomous driving decision and controlling system structure based on DDPG

Deep reinforcement learning (DRL) formulates the problem as a Markov Decision Process (MDP). The DRL agent collects information such as the current state observation s_t, action a_t, reward r_t, and the next state observation s_{t+1} as a tuple (s_t, a_t, r_t, s_{t+1}) which is then organized into a set (S, A, R, S'). The agent selects actions based on a deterministic policy $\pi : S \rightarrow A$. The cumulative reward value R_t at the current state observation s_t is define as

$$R_t = \sum_{0}^{T} \gamma^t r(s_t, a_t) \tag{1}$$

where $\gamma \in (0, 1)$ is the discount factor that accounts for the influence of future rewards.

The Q-value function, denoted as

$$Q^{\pi}(s, a) = \mathbb{E}[R_t | s_t, a_t] \tag{2}$$

The optimization objective is the maximization of the cumulative reward value R_t by finding the optimal policy π^*, as expressed in Eq. (3), where $\forall(s, a) \in (S, A)$ implies

that the optimal Q-value is greater than or equal to the Q-value of any other policy.

$$\forall(s,a) \in (S,A),\ Q^{\pi^*}(s_t,a_t) \geq Q^{\pi}(s_t,a_t) \tag{3}$$

The DDPG algorithm is a model-free deep reinforcement learning approach that generates continuous action outputs. It utilizes an actor-critic structure, comprising the Actor network $\mu(s|\theta^{\mu})$ and the Critic network $Q(s,a|\theta^{Q})$. In addition, there are Target-Actor and Target-Critic networks denoted by $\mu(s|\theta^{\mu'})$ and $Q(s,a|\theta^{Q'})$ respectively. These networks play a pivotal role in the algorithm's overall functioning.

In order to promote exploration in the environment, the DDPG algorithm incorporates random noise $\mathcal{N}$. This noise aids in diversifying the actions taken by the agent and facilitates a more comprehensive learning process. Another crucial component of the DDPG algorithm is the experience replay buffer $\mathcal{R}$, which serves as a storage mechanism for training data. By randomly sampling and replaying past experiences during network training, the algorithm can effectively learn from its history and improve its performance.

To strike a balance between exploiting learned knowledge and exploring new possibilities during the model training process, it is necessary to introduce noise to the actions generated by the policy network. This noise serves the purpose of facilitating exploration. In our approach, we adopted the Ornstein-Uhlenbeck process, which generates noise that gradually converges over time. By applying this technique, we can transform the deterministic behavior of the policy network into stochastic behavior, enabling a more diverse exploration of the environment.

The policy of adding noise to promote exploration is commonly known as the behavior policy, denoted as β. It is worth noting that during the testing or evaluation phase of the model, a deterministic policy is typically employed instead of the behavior policy. This deterministic policy allows us to focus on exploiting the learned knowledge and making precise decisions when assessing the model's performance under controlled conditions. To assess the effectiveness of policy π, we utilize the function $J_{\beta}(\pi)$, which is defined as:

$$J_{\beta}(\pi) = \int_s \rho^{\beta}(s) Q^{\pi}(s,\mu(s))ds = \mathbb{E}_{s\sim\rho^{\beta}}[Q^{\pi}(s,\mu(s))] \tag{4}$$

where s denotes state of the environment. Additionally, the distribution of state is subject to behavior policy , i.e., ρ^{β}.

1) Training the Actor Network: The Actor network is a fully connected network that takes the state s_t at time t as input and processes it through two fully connected layers. Between the hidden layers, the ReLU activation function is applied to introduce non-linearity. The output layer of the network consists of two activation functions: the tanh function, which normalizes the steering action within the range $(-1,1)$, and the sigmoid function, which normalizes the braking and acceleration actions within the range $(0,1)$. The structure of the Target-Actor network mirrors that of the Actor network, ensuring consistency in their architectures. Fig. 2 provides a visual representation of the Actor network's structure.

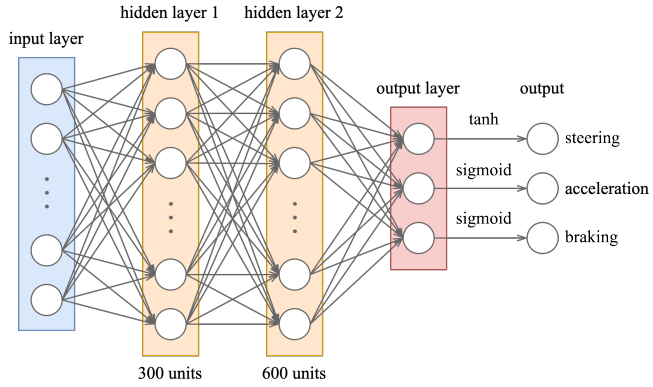


Fig. 2. Actor network

The gradient of the function J can be approximated by considering the expectation over the state s_t drawn from the behavior policy ρ^{β}. This can be expressed as:

$$\begin{aligned}\nabla_{\theta^{\mu}} J &\approx \mathbb{E}_{s_t\sim\rho^{\beta}}[\nabla_{\theta^{\mu}} Q(s,a|\theta^{Q})|_{s=s_t,a=\mu(s_t|\theta^{\mu})}] \\ &= \mathbb{E}_{s_t\sim\rho^{\beta}}[\nabla_a Q(s,a|\theta^{Q})|_{s=s_t,a=\mu(s_t)} \cdot \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s=s_t}]\end{aligned} \tag{5}$$

This gradient can be used to update the parameters θ^{μ} of the Actor network in order to maximize the expected return Q^{π}. We have chosen to utilize the Adaptive Moment Estimation (Adam) optimization algorithm to perform the parameter updates.

2) Training The Critic Network: The Critic network in the DDPG algorithm receives two inputs: the state s_t and the action a_t. The state s_t is initially processed through two fully connected layers, resulting in the extraction of features l_s^1 and l_s^2. Similarly, the action a_t undergoes a single fully connected layer to obtain features l_a^1, which has the same dimension as l_s^2 (600 in this case). These features are then integrated to form l_{sa}.Subsequently, the integrated features l_{sa} are passed through an additional fully connected layer to compute the Q-value. The input layer s_t and the fourth hidden layer are activated using the ReLU activation function, while the other hidden layers do not employ any activation function.It is worth noting that the structure of the Target-Critic network mirrors that of the Critic network.The structure of the Critic network is shown in Fig. 3.

The loss of Critic network is defined as

$$L(\theta^{Q}) = \mathbb{E}_{s_t\sim\rho^{\beta},a_t\sim\beta,r_t\sim E}[(Q(s_t,a_t|\theta^{Q}) - y_t)^2] \tag{6}$$

where

$$y_t = r(s_t,a_t) + \gamma Q\prime(s_{t+1},\mu\prime(s_{t+1}|\theta^{\mu'})|\theta^{Q\prime}) \tag{7}$$

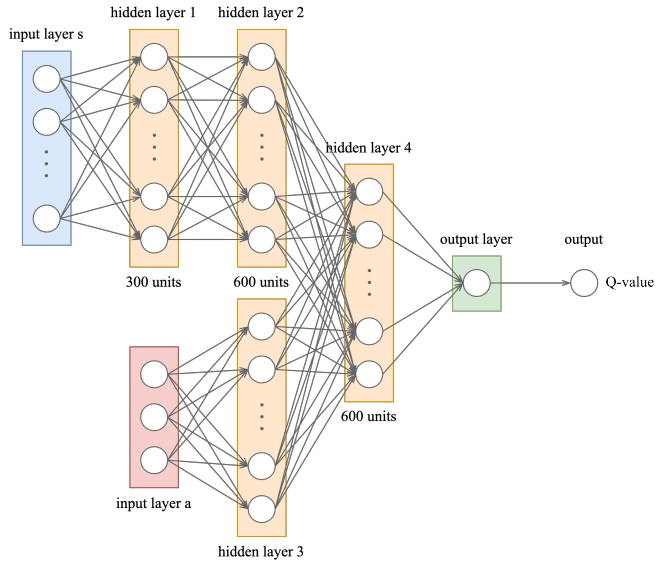


Fig. 3. Critic network

Thus, the gradient of $L(\theta^Q)$ is given by

$$\nabla L(\theta^Q) = \mathbb{E}[2(y - Q(s_t, a_t|\theta^Q)) \cdot \nabla Q(s_t, a_t)] \quad (8)$$

To minimize $L(\theta^Q)$, the Critic network parameter θ^Q can be updated by gradient descent. We chose the Adaptive Moment Estimation (Adam) as our optimization algorithm.

Furthermore, the Target-Actor network parameter $\theta^{\mu'}$ and the Target-Critic network parameter $\theta^{Q\prime}$ can be "soft" target updated according to Eq. (9)-(10)

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \quad (9)$$

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \quad (10)$$

where τ is super parameter of "soft" target update, particularly, $\tau \ll 1$.

The complete autonomous driving decision and controlling algorithm for FDRL is presented in Algorithm 1.

B. DRL Reward Function Settings

In the context of the TORCS autonomous driving task, the state space is defined as a set of variables collected from sensors. The variables and their descriptions are presented in Eq. (11) and Table I.

$$s_t = \{t_p, \upsilon_x, \upsilon_y, \upsilon_z, t, \imath, \omega, a\} \quad (11)$$

Similarly, the action space is presented in Eq. (12) and TABLE II.

$$a_t = \{a_a, a_b, a_s\} \quad (12)$$

The reward value R_t is generated according to reward function. The goal of autonomous driving is a well-trained DDPG agent that is able to output actions to drive in the

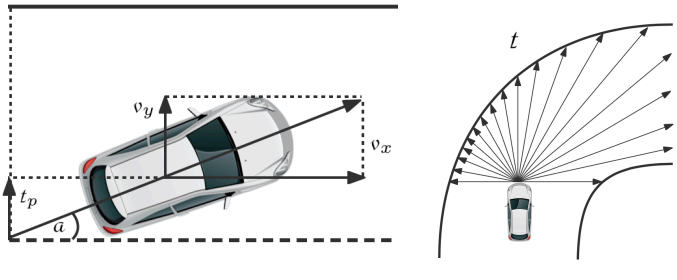
Algorithm 1 The DDPG Based Autonomous Driving Decision and Controlling Algorithm

1: Randomly initialize the online Actor network parameters θ^{μ}, the online Critic network parameters θ^{Q}. Set the target Actor network parameters $\theta^{\mu'} \leftarrow \theta^{\mu}$, and set the target Critic network parameters $\theta^{Q'} \leftarrow \theta^{Q}$.
2: Initialize the replay buffer $\mathcal{R}$.
3: for each episode do
4: Initialize the Ornstein-Uhlenbeck process.
5: for each time slot t do
6: According to behavior policy, the Actor network chooses an action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$.
7: Executes action a_t in the environment and observes the reward r_t and the next state s_{t+1}.
8: Store the transition $(s_t, a_t, r_t, s_{t+1}) into \mathcal{R}$.
9: Sample a random mini-batch of transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$.
10: Update the Critic network parameter θ^{Q} by minimizing $L(\theta^{Q})$ according to Eq. (6)-(8).
11: Update the Actor network parameter θ^{μ} by maximizing Q^{π} according to Eq. (1)-(5).
12: Update the target network according to Eq. (9)-(10).
13: end for
14: end for

TABLE I
State space properties

Symbol	Description
t_p	Percentage of deviation from the center of the track
υ_x	Longitudinal velocity
υ_y	Lateral velocity
υ_z	Vertical velocity
t	Distance to edge of track collected from sensors on front half of car every 10 degree
$\imath$	Current engine speed
ω	Angular velocity of the four wheels
a	Angle between car and track axis ($\pi > a > \pi$)

Note that the dimension of t is 19, and the dimension of ω is 4, therefore, the dimension of s_t is 29.



(a) (b)

Fig. 4. State Space Properties

TABLE II
Action space properties

Symbol	Description
a_a	Acceleration volume. The values range from $(0, 1)$
a_b	Braking volume. The values range from $(0, 1)$
a_s	Steering volume. The values range from $(-1, 1)$

Note that the negative value of a_s indicates a right turn, and the positive value of a_s indicates a left turn.

center of a lane at a target speed of 80 km/h in TORCS simulator. Hence, we conceived a reward function based on the next state observed in the environment after taking action outputted by the agent combined with action taken by agent.

We took longitudinal velocity, angle between car and track axis, percentage of deviation from the center of the track, distance to edge of track collected from sensors on front half of car every 10 degree, braking volume, steering volume i.e., v_x, a, t_p, t, a_b, a_s into account.

We specified the reward function with bonus term in the following cases:

- the car is closer to the target speed;
- the angle between car and track axis is smaller;
- the car is closer to the center of the track;
- the car moves toward the center of track and on the same side as the previous state.

However. A penalty term is applied in the following cases:

- brakes are still applied when the speed is too low;
- the car still turns in the same direction of offset when the offset is too large.

Furthermore, the training episode is terminated and a greater penalty term is applied in the following cases:

- a collision occurs or the car goes out of track;
- the progress of agent is too small;
- the car runs backward.

The specified reward function is presented in Algorithm 2.

Algorithm 2 Reward Function of DDPG Agent

Require: the next state observed including longitudinal velocity v_x, angle between car and track axis a, percentage of deviation from the center of the track t_p, distance to edge of track collected from sensors on front half of car every 10 degree t, current percentage of deviation from the center of the track t'_p, braking volume a_b and steering volume a_s.

Require: judgment steps j_s=200, minimum speed limit l_s=8.

if $v_x < 63$ then
  $sp = v_x$
else if $v_x < 80$ then
  $sp = 1.22^{\frac{v_x}{3}}$
else if $v_x < 97$ then
  $sp = 1.22^{\frac{160 - v_x}{3}}$
else
  $sp = 160 - v_x$
end if
$re = sp \times \cos(a) - |sp \times \sin(a)| - sp \times |t_p|$
if $t_p \times t'_p > 0$ and $t'_p \neq 0$ then
  $re = re + sp \times \frac{|t'_p - t_p|}{|t'_p - 0|}$
end if
if $sp < 50$ and $a_b > 0$ then
  $re = re - 100$
end if
if $\min(t) < 0$ then
  $re = re - 200$
end if
if $steps > j_s$ and $sp \times \cos(a) < l_s$ then
  $re = re - 200$
end if
if $\cos(a) < 0$ then
  $re = re - 200$
end if
return reward value re

C. FDRL Framework

We proposed a FDRL framework in this paper. The basic architecture of the FDRL framework is presented in Fig. 5. Different autonomous car agents are trained separately under different scenario maps, each one of them is not allowed to communicate with each other privately to prevent privacy breaches. Each agent shares the same the neural network structure so that our federal aggregation strategy can be processed.

1) Communication Strategy: The communication strategy of the federation server in the FDRL framework is depicted in Fig. 6. The strategy involves three communication commands: Send, Pending, and Call, which serve different purposes in the communication process between the server and the clients.

1. Send: The Send command is used by the federation server to transmit the federated model network parameters to the client. This command allows the server to distribute the updated model parameters to the clients for training.

2. Pending: The Pending command is set up to indicate that the model is updated asynchronously between the federation server and the clients. It signifies that the server is awaiting the completion of the client's local training process. Setting a suitable time interval ensures that the client has sufficient time to train locally without being overly influenced by frequent model updates from the federated aggregation process.

3. Call: The Call command is utilized by the federation server to request the model network parameters from the clients. This command prompts the clients to transmit their locally trained model parameters back to the server, enabling the aggregation of knowledge from multiple clients.

By employing this communication strategy, the federa-

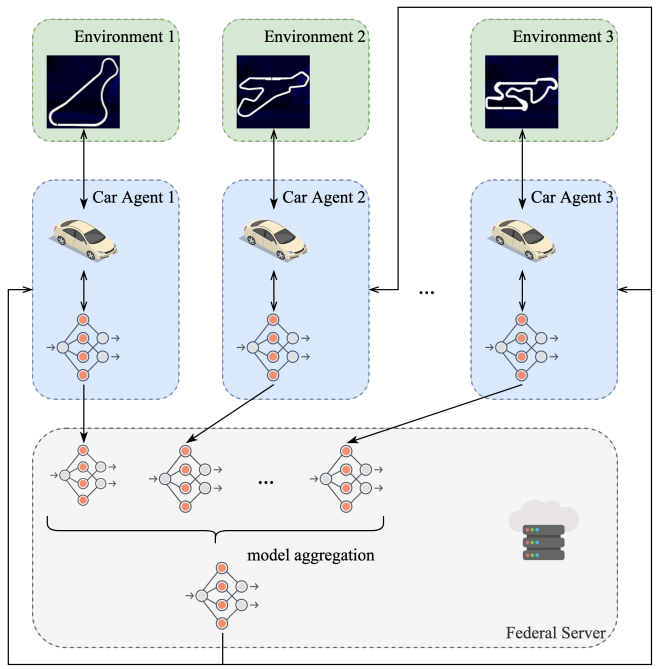


Fig. 5. FDRL Framework

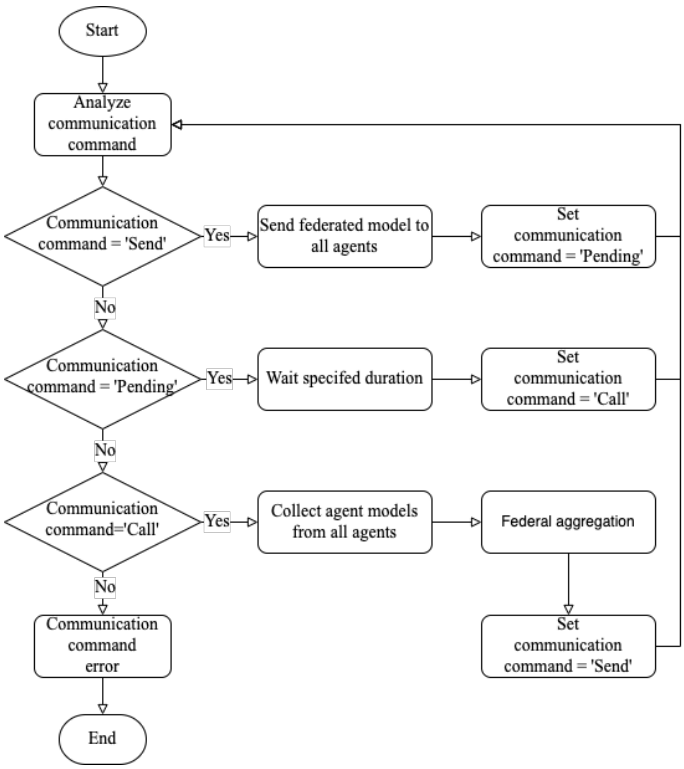


Fig. 6. Communication Strategy

tion server and the clients can effectively exchange model parameters and synchronize their training processes. The Pending command plays a crucial role in balancing the timing of model updates, allowing clients to train locally without significant disruptions caused by frequent federated aggregation updates.

2) Federal Aggregation Strategy: We leveraged the FedAvg algorithm as the federation aggregation algorithm. The federation server collects the model network parameters uploaded from all clients when the federation process happens. Since each client shares the same network structure, we are able to average them by parameter to synthesis the federation model. The process can be expressed as Eq. (13).

$$w_{fed}^{\theta} \leftarrow \frac{1}{N} \sum_{i=1}^{N} w_i^{\theta} \tag{13}$$

Here, w_{fed}^{θ} denotes the network parameters of federated model, similarly, w_i^{θ} denotes the network parameters of i-th client. N denotes the number of clients.

The federated learning process is presented in Algorithm 3.

IV. EXPERIMENT

This section describes our experimental session of the FDRL framework on multiple maps under the TORCS platform, aiming to validate the performance improvement of our proposed FDRL framework for autonomous driving decision and control models.

Algorithm 3 Federated Learning Process

Require: time interval t_s, start time $t_0 \leftarrow$ current time, the number of clients N.

Federation server sends initial federated model network to all clients.
while TRUE do
  Each client trains DDPG model locally according to Algorithm 1 and Algorithm 2.
  Get communication command c_c.
  if c_c ==‘Send’ then
    Federation server sends federated model to all clients.
    Set c_c = ‘Pending’.
  else if c_c ==‘Pending’ then
    Get current time t_c.
  else if then$t_c - t_0 > t_s$
    Set c_c = ‘Call’.
  else if c_c ==‘Call’ then
    Federation server requests model network from all clients.
    Federation server process federation aggregation leveraging FedAvg according to Eq. (13).
    Set c_c = ‘Send’.
  end if
end while

We selected three maps, namely CG Speedway number 1, E-Track 2, and E-Track 3 in the TORCS platform, for training and evaluation purposes, as shown in Fig. 7.

(a) CG Speedway number 1 (b) E-Track 2 (c) E-Track 3

Fig. 7. Training and Evaluation Maps

More specifically, CG Speedway number 1 is a relatively simple map characterized by flat routes, smaller curves, and low driving difficulty. In contrast, the map routes of E-Track 2 and E-Track 3 are more complex, featuring larger curves and sharp continuous curves, which pose significant challenges for unmanned car driving.

To evaluate the performance improvement of our proposed FDRL framework for autonomous driving decision and control models, we conducted four groups of experiments with different scenarios and training methods. These groups include:

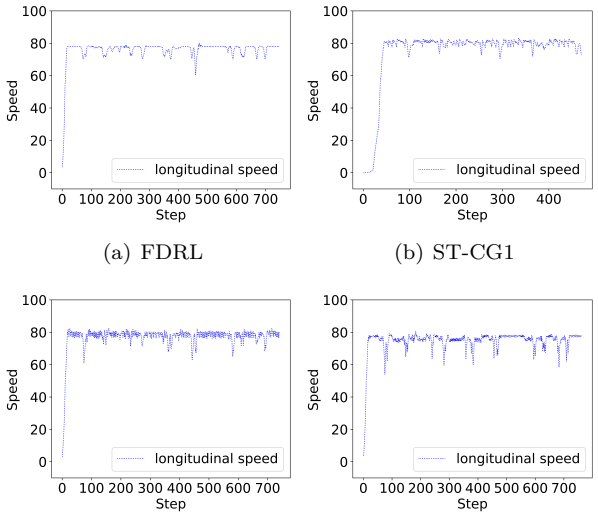
- FDRL: This group involves multiple clients training DRL agents separately in different scenarios and building a federated model, as described in Section III. For instance, in this study, we assigned two clients to train on CG Speedway number 1 and E-Track 3 individually, with communication between the clients and the federation server to synthesize an FDRL model.
- ST-CG1: In this group, a single client trains a DRL agent on CG Speedway number 1 in a stand-alone manner.
- ST-ET3: This group focuses on a single client training a DRL agent on E-Track 3 independently.
- MULTI: This group involves a single client training a DRL agent on both CG Speedway number 1 and E-Track 3 consecutively.

As shown in Fig. 8, we selected E-Track 2 as the evaluation map for assessing the performance of the agents. Due to the fact that the reward value alone may not provide an intuitive reflection of the agent's driving performance on the map, we considered several parameters as indicators to evaluate the agents' driving capabilities. These parameters include v_x, v_y and t_p. v_x represents the driving speed performance of the agent. A higher speed value that approaches 80 km/h indicates better speed performance for the agent. Similarly, v_y and t_d are indicators of the agent's driving safety performance. A smaller lateral velocity and a smaller offset from the track center indicate better safety performance for the agent.

Fig. 8. Evaluation on E-Track 2

To obtain the track offset distance t_d of the vehicle from the track center, we multiply t_p by the road width, which is 12 meters.

By considering these parameters, we can comprehensively evaluate the driving performance of the agents on E-Track 2 and analyze their speed and safety capabilities.



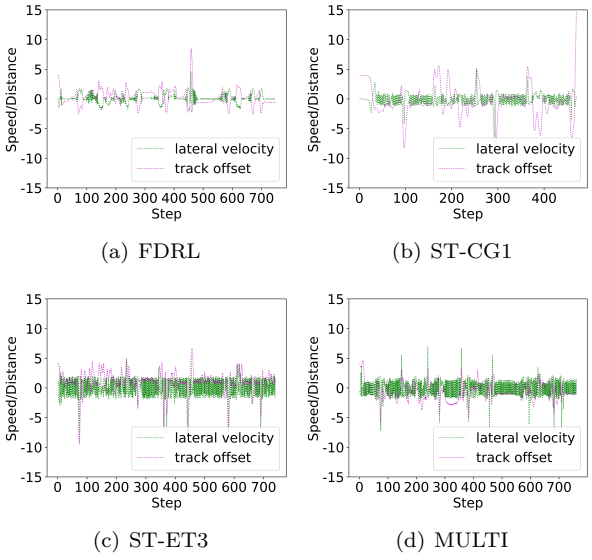
(a) FDRL (b) ST-CG1

(c) ST-ET3 (d) MULTI

Fig. 9. Speed Curve

Fig. 9. illustrates the speed curves of the four groups of agents driving one lap on the E-Track 2 map. Upon observation, it can be seen that the speed of all four groups is initially maintained at around 80 km/h. When comparing the curves of the four groups, it becomes apparent that FDRL exhibits the smoothest speed curve and is able to consistently maintain a stable speed throughout the lap. However, it is worth noting that the speed curve of ST-CG1 in Fig. 9(b) terminates at step 470, . This termination occurs because the agent of ST-CG1 encountered a continuous sharp turn, as shown in Fig. 8,

and failed to make an appropriate decision. Consequently, the vehicle veered off the lane, resulting in the termination of the experiment for this agent. Furthermore, the speed curve of ST-ET3 in Fig. 9(c) demonstrates intensive fluctuations with small changes. This indicates that the agent of ST-ET3 struggles to stabilize the speed, even when driving on a flat road. Lastly, the speed curve of MULTI in Fig. 9(d) exhibits the most significant changes in speed while passing through the curved section. At one point, the speed even drops below 60 km/h, indicating poor cornering performance for this agent. Based on these observations, it can be concluded that FDRL shows superior speed control and stability compared to the other groups. ST-CG1 encounters difficulties in handling sharp turns, ST-ET3 struggles to maintain a stable speed, and MULTI exhibits suboptimal cornering performance.



(a) FDRL (b) ST-CG1

(c) ST-ET3 (d) MULTI

Fig. 10. Offset Curve

Fig. 10 resents the offset curves of the four experimental groups of agents driving one lap on the E-Track 2 road. Analyzing the curves, several observations can be made: In Fig. 10(a), the lateral velocity curve of FDRL exhibits minimal fluctuations, mostly remaining above and below 0. The track offset curve of FDRL also shows relatively infrequent changes and less fluctuation, except for a larger track offset when passing through the continuous sharp turn road section. Overall, the track offset curve of FDRL remains within the interval [-4, 4]. Fig. 10(b) illustrates that the track offset curve of ST-CG1 terminates after reaching an extreme value at step 470. This termination occurs due to the vehicles controlled by this agent veering out of the lane. In Fig. 10(c), the lateral velocity curve of ST-ET3 appears dense and fluctuates significantly, failing to stabilize above and below 0. The track offset curve of ST-ET3 also exhibits substantial fluctuations, exceeding 5 meters multiple times. Examining Figure 10(d), it can be observed that the lateral velocity curve of MULTI is dense and displays significant fluctuations, surpassing 5 km/h on numerous occasions. However, the fluctuation of the track offset curve of MULTI occurs less frequently and can be maintained within a small range.Comparing the four sets of curves, it becomes evident that FDRL demonstrates smoother overall performance with smaller fluctuation amplitudes in both the lateral velocity and track offset curves. ST-CG1 experiences challenges in maintaining the vehicle within the lane, ST-ET3 struggles to stabilize lateral velocity and track offset, and MULTI exhibits considerable fluctuations in lateral velocity while managing to control track offset within a relatively small range.

Based on the analysis of Fig. 9 and Fig. 10, it can be concluded that FDRL outperforms the other three groups in terms of stabilizing longitudinal speed and offset speed while maintaining the target speed. FDRL demonstrates a smoother driving performance by effectively controlling the vehicle. Additionally, the FDRL approach offers the advantage of improving the environmental adaptability of agents trained in simpler environments. This allows the agents to achieve good driving performance even in complex environments they have not encountered before. The federated model generated through FDRL enables agents to generalize their learned behaviors and effectively navigate challenging driving scenarios.

To facilitate a more intuitive comparison of the experimental results, we conducted data analysis on three key parameters: longitudinal velocity (v_x), lateral velocity (v_y), and track offset distance (t_d). Specifically, we recorded the mean values and standard deviations of these parameters for the different agent groups. Since both lateral velocity and track offset distance have directions, we recorded the standard deviation of their absolute values to assess their variability. The results of the analysis are presented in Table III, with the superior results highlighted in bold.

TABLE III
Numerical Analysis

	Avg v_x	SD v_x	Avg $\|v_y\|$	SD v_y	Avg $\|t_d\|$	SD t_d
FDRL	75.798	7.361	0.486	0.735	0.957	1.295
ST-CG1	74.255	19.552	0.990	0.606	1.950	2.732
ST-ET3	76.964	8.055	1.502	1.769	1.714	1.996
MULTI	74.406	7.708	1.336	1.591	1.295	1.511

Indeed, the analysis of Table III clearly demonstrates the superior performance of the FDRL group in five out of the six recorded data metrics, and the second-best performance in one metric. Specifically, FDRL showcases remarkable results by minimizing velocity variations while maintaining an average longitudinal velocity close to the target speed. Moreover, FDRL effectively minimizes the variations in both lateral velocity and track offset distance, indicating its ability to drive with stability and precision.Comparing FDRL with ST-CG1, we can conclude that the FDRL approach enhances the driving performance of models trained in simple scenarios,

enabling them to navigate complex scenarios that they have not previously encountered. This demonstrates the effectiveness of the FDRL framework in improving the adaptability and generalization capabilities of the trained models.Furthermore, when comparing FDRL with ST-ET3 and MULTI, it becomes evident that the FDRL-trained model aligns more closely with the objectives of our autonomous driving task, which include maintaining the target speed and driving steadily near the center of the lane. This emphasizes the superiority of FDRL in achieving the desired driving behavior.Overall, the results highlight the significant benefits of the FDRL framework in enhancing driving performance, adaptability, and stability, thereby advancing the capabilities of autonomous driving models in diverse and challenging scenarios.

V. Conclusion

In conclusion, this paper presents the FDRL framework, which addresses the challenges of preserving data privacy while enabling clients to obtain models adapted to diverse and complex scenarios. The experimental results demonstrate that the driving performance of the FDRL model surpasses that of traditional training methods.Nevertheless, it is important to note that the experiment conducted in this paper involved a relatively small number of clients. To further improve the performance and effectiveness of the FDRL framework, it is recommended to involve a larger number of clients representing different scenarios. By increasing the diversity of clients and incorporating their expertise, the FDRL framework has the potential to achieve even better results and enhance its overall performance.Overall, the FDRL framework presents a promising approach to federated learning in the field of autonomous driving, offering improved driving performance while addressing data privacy concerns. Future research and experiments with an expanded client base can further validate and enhance the capabilities of the FDRL framework in real-world autonomous driving applications.

References

[1] D. Loiacono, A. Prete, P. L. Lanzi, and L. Cardamone, "Learning to overtake in torcs using simple reinforcement learning," in IEEE Congress on Evolutionary Computation. IEEE, 2010, pp. 1–8.

[2] W. Xia, H. Li, and B. Li, "A control strategy of autonomous vehicles based on deep reinforcement learning," in 2016 9th International Symposium on Computational Intelligence and Design (ISCID), vol. 2. IEEE, 2016, pp. 198–201.

[3] H. Chae, C. M. Kang, B. Kim, J. Kim, C. C. Chung, and J. W. Choi, "Autonomous braking system via deep reinforcement learning," in 2017 IEEE 20th International conference on intelligent transportation systems (ITSC). IEEE, 2017, pp. 1–6.

[4] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," Electronic Imaging, vol. 2017, no. 19, pp. 70–76, 2017.

[5] 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.

[6] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," IEEE Transactions on Intelligent Transportation Systems, 2020.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Artificial intelligence and statistics. PMLR, 2017, pp. 1273–1282.

[8] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 13, no. 3, pp. 1–207, 2019.

[9] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," arXiv preprint arXiv:1811.03604, 2018.

[10] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," arXiv preprint arXiv:1806.00582, 2018.

[11] M. Spryn, A. Sharma, D. Parkar, and M. Shrimal, "Distributed deep reinforcement learning on the cloud for autonomous driving," in Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems, 2018, pp. 16–22.

[12] X. Liang, Y. Liu, T. Chen, M. Liu, and Q. Yang, "Federated transfer reinforcement learning for autonomous driving," arXiv preprint arXiv:1910.06001, 2019.

[13] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems," IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 4555–4562, 2019.

[14] B. Liu, L. Wang, M. Liu, and C.-Z. Xu, "Federated imitation learning: A novel framework for cloud robotic systems with heterogeneous sensor data," IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 3509–3516, 2020.