Dynamic Car-following Model Calibration with Deep Reinforcement Learning

Htet Naing[1], Wentong Cai[1], Tiantian Wu[2] and Liang Yu[2]

***Abstract*— In microscopic traffic simulation, it is apparent that there is a dilemma between physics-based and learning-based models for modelling car-following behaviours. The former can offer analytical insights with low simulation accuracy while the latter functions like a black-box, but offers high simulation accuracy. Thus, a new perspective on combining physics-based car-following models (CFM) and learning-based methods is given in this paper by integrating the two approaches through "model calibration". The CFM calibration is formulated as a sequential decision-making process via deep reinforcement learning (DRL), and a general framework is provided to achieve this purpose. To the best of our knowledge, this is the first time in literature that formulates dynamic CFM calibration using DRL. The experiment results have shown that our DRL-based dynamic calibration method, d-DDPG, has outperformed more conventional method such as Genetic Algorithm with 26.80% and 23.16% reduction in terms of root mean squared spacing error and position trajectory deviation error, respectively. Therefore, our work could serve as a promising initiative towards this new research direction.**

I. INTRODUCTION

In microscopic traffic simulation, an increasing number of works have adopted data-driven approaches: [1]–[5], for modelling car-following behaviours since they can achieve higher simulation accuracy than conventional physics-based models [6]–[8]. This is due to both their high-dimensional modelling capability and flexibility, which allows them to incorporate more relevant input features into the models [9]. More recent data-driven models are formulated based on deep learning [1]–[3] and deep reinforcement learning (DRL) [4], [5]. One major drawback of these learning-based models is their black-box nature because they are usually composed of a large number of parameters with many hidden layers, making it difficult to interpret their results analytically to draw useful insights with respect to traffic flow theory unlike physics-based models.

To resolve the above dilemma, it is desirable to formulate a new approach that does not undermine model analytical explainability while leveraging the strengths of learning-based models. One viable integration of the two (physics-based and learning-based) modelling paradigms, which can address the trade-off between model analytical explainability and simulation accuracy, can be achieved through "model calibration". In other words, to maintain the analytical explainability, the underlying model should remain physics-based in nature. However, physics-based models usually employ constant parameters to account for missing physics in the model formulation [10]. Thus, before using physics-based models, these constant parameters should be calibrated based on observation data to attain the best possible simulation accuracy. Inadequate parameter calibration can have negative impact on the accuracy of a physics-based model [10]. This creates an opportunity to utilize learning-based models for estimating the parameter values more accurately.

Among learning-based methods, supervised deep learning-based models are not ideal for model calibration task because they require a large amount of ground-truth labelled parameter values, which can be difficult to derive from observation data. This is a critical limitation of using deep learning models for model calibration task [11]. Alternatively, it can be overcome by DRL which only requires a simulation environment and the observation data, thus removing the necessity of the ground-truth labelled parameter values [11]. Due to this advantage of DRL models, it has been already adopted for dynamic model calibration in a few science and engineering applications [11]–[14].

Despite the benefits of DRL, to the best of our knowledge, there is no systematic framework that describes how to dynamically calibrate a car-following model (CFM) parameters using DRL. Furthermore, due to the exploratory nature involved in the learning mechanism of DRL, it could result in implausible or invalid parameter values during the learning process. A naive way of dealing with this issue is to clip such a parameter value within its valid interval. However, this does not prevent the DRL agent from extreme over- or under-estimation again since there is no feedback (reward/penalty) signal associated with this behaviour. It is even more important, in the deployment stage, to avoid this behaviour if the DRL agent is to be used for dynamic model calibration. To fill these research gaps, we have proposed a general CFM calibration framework based on DRL. Our main contributions can be summarised as follows:

1) a general dynamic CFM calibration framework is developed based on DRL.
2) an improvement to model parameter exploration process is made by penalising the DRL agent for causing invalid parameter values.
3) a set of both qualitative and quantitative experiments were conducted to demonstrate the advantages of the proposed framework.

The rest of the paper is organized as follows: Section II describes related works in existing literature. Our proposed framework is presented in Section III. The experiment results are then described and analyzed in Section IV. Finally, the

[1]Htet Naing and Wentong Cai are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore.
[2]Tiantian Wu and Liang Yu are with City Brain Lab, Alibaba Cloud, China.

paper concludes with a summary in Section V.

II. RELATED WORKS

A. Model Calibration with Deep Reinforcement Learning

It is an important task to estimate the parameters of a physics-based model, as accurately as possible, from observation data in many scientific and engineering applications. As mentioned in [11], [12], manual parameter calibration (such as the Design of Experiments and classic search algorithms) do not scale well when the number of parameters grows larger. This is resolved via automated calibration techniques that are based on optimisation and sampling such as Genetic Algorithm (GA) or Particle Filters, which are, however, still not ideal for dynamic model calibration due to their high computational cost. Another group of existing works uses deep learning as surrogate models to approximate the behavior of a simulation model to save computational efforts required for large simulation runs. However, in the case of parameter calibration, it could be very challenging to obtain reliable ground-truth labelled data that are required for training these deep learning models.

To tackle the above limitations inherent in existing methods, a few relatively new research works have adopted DRL for model calibration tasks [11]–[14]. For the reason already discussed in Section I, DRL is more suitable for the model calibration task than deep learning models because it does not require any labelled ground-truth parameter values. A summary of these works is tabulated in Table I, where N_{param} is the number of calibration parameters. Due to the paper length constraint, a review of these works is given in our online supplementary materials [15].

TABLE I
SUMMARY OF DRL-BASED MODEL CALIBRATION APPROACHES

Ref	Year	DRL Model Structure	Application	N_{param}
[14]	2019	DQN [16]	Incomplete vehicle trajectory reconstruction	1
[11]	2020	SAC [17]	Diagnostics-related parameter calibration for turbofan engines	4
[12]	2020	DDPG [18]	Power system model parameter calibration	2
[13]	2021	SAC [17]	Battery model parameter calibration	2

Compared to the existing DRL-based calibration works, ours is different in two aspects. Firstly, in these works, no clear emphasis was given towards extreme parameter changes that could result in invalid parameter values due to a DRL agent calibration action. It may be because in their applications, either the impact of such extreme over- or under- estimation is insignificant (towards the underlying physics model performance), or such possibility is simply ignored. However, this could lead to unstable agent exploration behaviour during the training process as observed in our experiment results (see Section IV-D). Consequently, it could also affect the training convergence towards the agent optimal behaviour. Hence, in our work, we have designed an improved reward function that penalizes such undesired calibration actions of the agent. Secondly, the number of parameters calibrated by DRL in these works only range from one parameter in [14] to four parameters in [11]. In this paper, by calibrating up to six parameters of a CFM through DRL, we have shown the capability of our calibration framework that could potentially scale to larger number of parameter dimensions.

B. Existing Works in Car-following Model Calibration

Among the existing CFM calibration works, two approaches are popular – 1) a gradient-free GA [19]–[23], and 2) a descent method, Simultaneous Perturbation Stochastic Approximation (SPSA) [24], [25] which is known to be capable of finding global minima. One major disadvantage that is common in both GA and SPSA is their slow convergence issue [25]. Hence, it becomes even more difficult to formulate dynamic CFM calibration with these methods. As a result, very few works on dynamic CFM calibration exist although an initiative has been made in [25] to encourage this research direction. Therefore, this motivates us to offer a new perspective on CFM calibration by using DRL. Despite its promising results, (discussed in Section II-A), to the best of our knowledge, since there is no general framework that provides how to calibrate a CFM both offline and online via DRL, we take the first initiative to propose such a framework for dynamic CFM calibration in this paper.

III. DYNAMIC CAR-FOLLOWING MODEL CALIBRATION

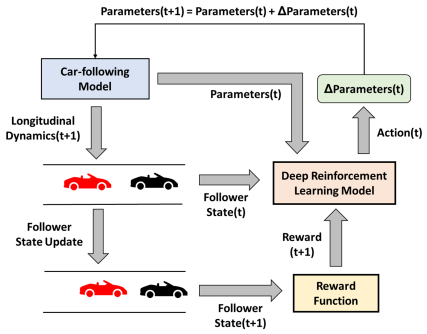

Fig. 1. Dynamic car-following model calibration framework based on DRL

The overview of our proposed framework is shown in Figure 1. In this figure, a red vehicle (follower) is following a black vehicle (leader) in a car-following trajectory simulation. In such a simulation, the leader usually moves according to the actual ground-truth data while the follower is simulated by a car-following model (CFM). This model determines the longitudinal dynamics for the follower based on the follower's current state and its own constant parameters. The role of a DRL model (agent) is to determine how much current parameter values should be changed so that the response of the CFM matches more closely to the observed data. Hence, at each simulation step, the CFM constant parameters are updated with these parameter changes determined by the DRL model. With the newly updated parameter values, CFM produces the longitudinal dynamics

for the follower simulation towards the next time step. After updating the follower state using the predicted dynamics, the new follower state is observed, and then, the reward can be calculated based on this new state to evaluate the agent calibration action, which can be used as a feedback signal to guide the agent behaviour towards the desired optimal behaviour. In our case, the optimal behaviour for the DRL agent is to estimate the most accurate parameter changes at each simulation step in order to achieve the best possible simulation accuracy through maximizing the cumulative sum of rewards. Each component involved in the framework is described in more details in the following sections.

A. Problem Formulation

The parameter calibration problem can be formulated as a sequential decision making process where the goal of our DRL agent is to learn the best possible sequence of calibration actions, which are to be taken at each time step throughout the trajectory of a simulated follower, so that the expected future cumulative reward can be maximized. More formally,

$$Traj_{sim} = \{veh_t | t \in 1, ..., L\} \tag{1}$$

$$veh_{t+1} = CFM(veh_t, \lambda_{t+1}) \tag{2}$$

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\Big[\sum_{k=t+1}^{L} \gamma^{k-(t+1)} r_k \Big| s_t = s, a_t = a, \pi \Big] \tag{3}$$

$$\lambda_{t+1} = \lambda_t + \Delta\lambda_t; \quad LB_\lambda \leq \lambda \leq UB_\lambda \tag{4}$$

where $\Delta\lambda_t = a_t;\ s_t = \{veh_t, veh_t^a, \lambda_t\};\ s_t \in S;\ a_t \in A$

For a follower trajectory to be simulated ($Traj_{sim}$) with the trajectory length L, the follower state at the next simulation step veh_{t+1} is governed by a car-following model CFM which depends on the current follower state veh_t and the estimated model parameter values λ_{t+1}. Based on these states with additional relevant features veh_t^a (which are not part of the CFM inputs veh_t) represented altogether as s_t, the goal of our DRL agent is to learn, for each simulation step, how to take the best calibration action a_t which determines the most favorable change $\Delta\lambda_t$ for each parameter value subject to their lower and upper parameter bounds $[LB_\lambda, UB_\lambda]$. This so-called "best calibration action" can be achieved by learning how to approximate the optimal action-value function $Q^*(s,a)$ as defined in Equation 3. This is the standard formulation in DRL literature which can be found in [16], [18], [26] for more details. The intuition behind this formulation is that if the agent has learnt how to approximate this optimal Q-value (which estimates the future cost due to current decisions), it will be able to decide what kind of action should be taken, for a given state, in order to obtain the maximum achievable sum of rewards. The meaning of the rest of the symbols are: r_{t+1} is the reward for taking a_t at s_t, and r_k where $k \in t+2, ..., L$ are the future rewards which are discounted by their corresponding discount factor $\gamma^{k-(t+1)}$. π is a behaviour policy that maps states to actions. S and A are the state and action spaces of the agent, respectively.

B. Car-following Model (CFM)

In this paper, for demonstration of our proposed framework, Intelligent Driver Model (IDM) [6] is selected as our choice of physics-based CFM due to both its widespread popularity and longitudinal collision-free property. Without loss of generality, our framework is applicable to parameter calibration of any CFM. IDM is formulated as follows:

$$acc_{t+1} = acc_{max} \left[1 - \Big(\frac{v_t}{v_0}\Big)^{\delta} - \Big(\frac{gap^*(v_t, v_t^{rel})}{gap_t}\Big)^2 \right], \tag{5}$$

$$acc_e \leq acc_{t+1} \leq acc_{max}$$

where $gap^*(v_t, v_t^{rel})$

$$= gap_{jam} + max\Big\{0, v_t\, T + \frac{v_t\, v_t^{rel}}{2\sqrt{acc_{max}\, dacc_{max}}}\Big\}$$

The IDM produces a follower's acceleration in the next time step (acc_{t+1}) as its output based on a set of inputs: the follower's current speed (v_t); front gap distance to the leader (gap_t); the relative speed with respect to the leader (v_t^{rel}). Finally, $gap^*(.)$ is the minimum desired gap distance which depends on v_t and v_t^{rel}. The predicted acceleration is bounded between the emergency braking acceleration acc_e and the maximum desired acceleration acc_{max}. The value of acc_e is often set to be within the range of $[-9.5, -9]$, which can be referred to [5]. Hence, the value -9.5 is used in this paper. As a result, the follower state in Equation 1 can be defined as:

$$veh = \{v, v^{rel}, gap\} \tag{6}$$

The IDM also uses constant parameters that are maximum desired speed (v_0); jam spacing distance (gap_{jam}); maximum desired acceleration and deceleration: (acc_{max}) and ($dacc_{max}$), respectively; desired time headway (T), and configurable exponent parameter (δ). Therefore, the set of model calibration parameters λ introduced in Equation 2 can be defined as:

$$\lambda = \{v_0, T, gap_{jam}, acc_{max}, dacc_{max}, \delta\} \tag{7}$$

Using the predicted acceleration in Equation 5, the follower position and speed can be updated using the ballistic update scheme as defined below.

$$x_{t+1} = x_t + v_t \Delta t + \frac{1}{2} acc_t (\Delta t)^2;\ \ x_{t+1} \geq x_t; \tag{8}$$

$$v_{t+1} = v_t + acc_t \Delta t;\ \ v_{t+1} \geq 0;\ \ \Delta t = 1\, sec \tag{9}$$

C. Deep Reinforcement Learning for Model Calibration

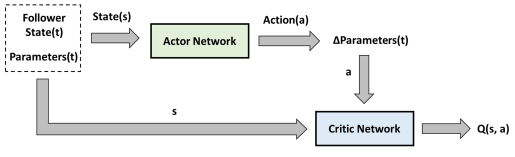

Fig. 2. DDPG for dynamic car-following model calibration

Inspired by [14], Deep Q-Network (DQN) was selected, as our starting point, to be used in our framework. Due to the

paper length constraint, the complete description about our model calibration framework using DQN is provided in our supplementary materials [15]. As discussed in [18], DQN has its weakness in terms of dealing with continuous action spaces since it can only produce a discrete set of action choices. As originally proposed in [18], Deep Deterministic Policy Gradient (DDPG) is an extension of DQN so that it can handle continuous action spaces with higher model complexity since it employs two networks namely, "Actor" and "Critic" Networks, grounded on the classic actor-critic reinforcement learning. Since DDPG (as illustrated in Figure 2) can handle high-dimensional continuous action spaces, it is adopted in this paper to calibrate all IDM parameters λ defined in Equation 7.

1) Actor Network: The symbol notations used in the original DDPG paper [18] are adopted here to avoid any confusion. The actor network $\mu(s|\theta^{\mu})$ with internal parameters θ^{μ} takes the follower state veh and IDM parameters λ (shown in Equations 6 and 7, respectively) as its input state (s). In order to keep our follower state representation generic, the set of additional features veh_t^a was defined to be empty. However, future works may consider adding more relevant features as necessary according to their application context. Based on these states, the actor produces the corresponding changes for each parameter ($\Delta\lambda$) as its output action (a). Since the last layer of the actor network is activated by $\tanh$ function, the amount of $\Delta\lambda$ is bounded between -1 and 1. These changes are then used to update the current λ.

2) Critic Network: To estimate the Q-value for a given state-action pair, the critic network $Q(s, a|\theta^{Q})$ with internal parameters θ^{Q} is used. It takes both state and action of the actor as its inputs, and produces the corresponding Q-value as its output.

3) Improved Reward Function: The reward function used for the DQN training process is inadequate since it only accounts for the deviation between the simulated and observed spacing data. Gap distance (spacing) is chosen as our performance measure because Punzo and et al. has proved empirically in [19] and theoretically in [27] that a solution given by the optimization based on spacing is indeed more optimal than that of velocity or acceleration. Due to the exploratory nature of DRL, it could lead to implausible or invalid parameter values during this training process. This problem is naively bypassed in our DQN calibration by clipping each "out-of-bounds (OOB)" parameter to be within their valid intervals. However, this simple technique does not prevent the DRL agent from extremely over- or under-estimating $\Delta\lambda$ again. This is because our DRL agent does not receive any feedback (reward/penalty) signal associated with this undesired behaviour. Thus, before clipping the extreme parameter values, we have calculated penalty for every OOB parameter change in order to discourage our DRL agent from committing such action in the future. Accordingly, our improved reward function is defined as follows:

$$r_t = e^{-(Error_{sum})} \tag{10}$$

The intuition behind this reward function is that the lower the sum of errors, the higher the reward value. The maximum achievable reward for our agent in a given time step is a scalar 1, which occurs when there is neither any gap distance squared error nor any OOB penalty.

$$Error_{sum} = Error_{data} + Error_{oob} \tag{11}$$

$$Error_{data} = (gap_t^{sim} - gap_t^{truth})^2 \tag{12}$$

$$Error_{oob} = \sum_{\lambda_i \in \lambda} f(\lambda_i); \tag{13}$$

$$f(\lambda_i) = \begin{cases} |LB_{\lambda_i} - \lambda_i| & , \text{ if } \lambda_i < LB_{\lambda_i}; \\ |UB_{\lambda_i} - \lambda_i| & , \text{ if } \lambda_i > UB_{\lambda_i}; \\ 0 & , \text{ otherwise} \end{cases}$$

For any exceeded amount (either $|LB_{\lambda_i} - \lambda_i|$ or $|UB_{\lambda_i} - \lambda_i|$) resulted due to $\Delta\lambda$, it is added as the penalty for the agent action. If it is within $[LB_{\lambda}, UB_{\lambda}]$, no penalty is added.

4) Agent Training: With a batch of transition experiences sampled from the replay memory which stores all agent transition experiences (s_t, a_t, s_{t+1}, r_{t+1}), the critic network parameters are updated by minimising the following training error loss.

$$Loss_{critic} = \mathbb{E}\Big[(Q^*(s,a) - Q(s,a|\theta^{Q}))^2\Big] \tag{14}$$

where $Q^*(s,a)$ (also defined in Equation 3) is approximated by $\approx r + \gamma Q^{'}(s', a'|\theta^{Q\prime})$; $a' = \mu'(s'|\theta^{\mu\prime})$; s' and a' are potential next state and action, respectively, after taking action a at the state s. $Q^{'}(.|\theta^{Q\prime})$ and $\mu'(.|\theta^{\mu\prime})$ are softly updated copies of the actor and critic networks that are used for approximating the optimal target Q-values, thus, namely "target networks". They are required to help the training of the critic network more consistent by providing stable target estimates. As for the actor network, it aims to maximize the expected return $\mathbb{E}\Big[Q(s,a|\theta^{Q})\Big]$ by applying the chain rule to this return with respect to θ^{μ} (see [18] for more details about the derivations). The actor optimization problem can be turned into minimization by placing negative sign (-) as follows:

$$Loss_{actor} = -\mathbb{E}\Big[Q(s,a|\theta^{Q})\Big] \tag{15}$$

Finally, the target network parameters are gradually copied from the main networks using the following equations:

$$\theta^{Q\prime} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q\prime} \tag{16}$$

$$\theta^{\mu\prime} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu\prime}$$

where τ is usually set to value much less than 1 for changing the target network parameters slowly. By following the original implementation of DDPG, the noise-based exploration strategy is also adopted in this paper by adding Ornstein–Uhlenbeck process (OU-Noise) to the action generated by the actor network during the training process.

Our DDPG training algorithm is provided in Algorithm 1. At the start, IDM parameters λ were initialised to default values. During the training process, 100 uniformly sampled trajectories (*D=100*) were simulated one after another. It can

be imagined that calibrating λ over one sampled trajectory is equivalent to playing one round of video game by the DRL agent in its original formulation [26]. Thus, the end of one trajectory marks the end of one training iteration. At each episode, the agent goes through 100 training iterations according to our algorithm. It is important to note that the CFM parameters λ are constantly being calibrated. After one training iteration, the previous calibrated state of λ is carried over to the next trajectory run and so on. When the agent behaviour has converged to optimal strategy, the set of calibrated parameters is also assumed to converge towards an optimal solution. This convergence process is usually monitored in DRL training process by keeping track of agent scores (accumulated rewards) for a given episode (for example, see Figure 3). When there is very insignificant increase in terms of the agent scores for the last few episodes, the algorithm can be considered for termination.

5) Agent Testing: At the end of each training episode, an updated set of IDM parameters are saved so that they can be selected later for testing. The agent performance for CFM calibration can be tested in two ways as follows:

- Static (offline) calibration: The final set of calibrated IDM parameters is used for individual follower trajectory simulation. No pre-trained actor network is deployed during the test stage.
- Dynamic (online) calibration: Each test follower is initialized with the final set of offline calibrated parameters. The pre-trained actor network is deployed during the test stage in order to perform online calibration of the IDM parameters for each follower trajectory. No learning (training) is involved during the test stage.

This dynamic model calibration capability of DRL, without relying on any online optimization or learning during the deployment stage, is the major advantage of DRL over other widely used optimisation algorithms such as GA. Although the DRL training process may take longer than others due to its inherent trial-and-error learning approach, the computational speed of the DRL model during the inference stage is considerably fast subject to the model complexity.

Algorithm 1: DDPG Training Algorithm

Result: Actor network $\mu(s|\theta^{\mu})$, critic network $Q(s,a|\theta^{Q})$ with trained parameters θ^{μ}, θ^{Q}, respectively; Calibrated CFM parameters λ

Initialization:
Initialize hyperparameters for $\mu(s|\theta^{\mu})$ and $Q(s,a|\theta^{Q})$;
Initialize network weights θ^{μ} and θ^{Q} randomly;
Initialize target network Q' and μ' with weights $\theta^{Q\prime} \leftarrow \theta^{Q}$ and $\theta^{\mu\prime} \leftarrow \theta^{\mu}$;
Initialize λ using predefined values;
Initialize λ lower and upper bounds: $[LB_{\lambda}, UB_{\lambda}]$;
Initialize a random process (*Noise*) for exploration;
for *epoch = 1, ..., E* **do**
  Uniformly sample D unique car-following trajectories from training data;
  for *Traj d = 1, ..., D* **do**
    Initialize follower and leader using ground-truth $Traj_t^d$ at time step t = 0;
    Get the agent environment state s_t where $veh_t, \lambda_t \in s_t$;
    for *t = 1, ..., L* **do**
      Get the agent action $a_t = \mu(s_t|\theta^{\mu})$;
      $\Delta\lambda_t \leftarrow a_t + Noise$;
      $\lambda_{t+1} = \lambda_t + \Delta\lambda_t$ subject to $[LB_{\lambda}, UB_{\lambda}]$;
      Update leader state using $Traj_{t+1}^d$;
      Update follower state using λ_{t+1} with Equations 5, 8, and 9;
      Observe new state s_{t+1};
      Calculate reward r_{t+1} using Equation 10, 11, 12, 13;
      Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory;
      Uniformly sample a batch of transition experiences B from replay memory;
      Calculate critic and actor losses using B with Equations 14 and 15 respectively;
      Update θ^{Q} using $Loss_{critic}$ gradient;
      Update θ^{μ} using $Loss_{actor}$ gradient;
      Update target networks using Equation 16;
    end
  end
end

IV. EXPERIMENTS

A. Experiment Setup

The experiments were conducted by using a publicly available real-world dataset, namely Next Generation Simulation (NGSIM) US Highway 101 dataset [28]. The dataset was collected from a segment of a highway road with length about 640 meters. It consists of five lanes on the main road (lanes 1-5), an auxiliary road (lane 6), an on-ramp (lane 7) and an off-ramp (lane 8). This dataset contains three sub-trajectory datasets recorded for 15 mins period each, *Dataset 1* - 07:50-08:05 AM, *Dataset 2* - 08:05-08:20 AM, and *Dataset 3* - 08:20-08:35 AM. The vehicle state trajectories were pre-processed from recorded videos in the original dataset.

1) Data Sampling: Each of the above datasets has a temporal resolution of 100 milliseconds with a total of 1,048,575 data instances. In this paper, the simulation time increment of 1 sec is adopted for drivers are unable to perform too many actions during a shorter period of time [1]. Hence, data instances were downsampled with 1-sec resolution.

2) Data Smoothing: As highlighted in existing studies [29], [30], the NGSIM dataset contains unrealistic pre-processed vehicle velocity and acceleration. Thus, a classic signal processing method, Savitzky-Golay filter [31] (with smoothing window and polynomial order set to 3 and 1, respectively) was adopted to smooth vehicle trajectories so that noise involved in the pre-processed data was alleviated. With reference to [30], the position data were smoothed firstly, then the velocity data were derived using the smoothed positions via differentiation. Secondly, the derived velocity data were smoothed, and differentiated to derive acceleration data. Finally, the acceleration data were smoothed again to further mitigate the noise amplification due to differentiation.

Since the original measurement units were in feet, they were converted to meters.

3) Data Filtering: Three types of vehicles were observed in data: (i) normal car, (ii) truck, and (iii) motorcycle, and very large proportion of the data belongs to normal car. All motorbike trajectories were removed. Only those data associated with main lanes 1-5 were selected. Moreover, from each dataset, only data instances between Frame 1500 and Frame 6000 were used since other frames contain unstable trajectories due to camera transitions. According to [4], 15-sec trajectory length requirement was adopted for selecting unique follower-leader pairs. Finally, a few outlier data points with negative preceding distances were also eliminated.

4) Training & Test Trajectories: After performing the above data-preprocessing steps, a total of: 1063; 1148; and 1224, unique follower-leader trajectories were extracted from *Dataset 1*, *Dataset 2*, and *Dataset 3*, respectively. The trajectories from *Dataset 1* and *Dataset 2* were used for model training (and offline calibration) whereas *Dataset 3* was used for model testing (and online calibration).

5) Performance Evaluation Metrics: As discussed in Section III-C.3, since we have focused on optimising the parameters based on gap distance, the following two error metrics that are associated with gap distance are used to assess the model calibration performance.

- $RMSE_{gap}$: For *N* test data instances, the root mean squared error (RMSE) between the actual ground-truth gap (gap^{truth}) and simulated gap (gap^{sim}) is defined as:

$$RMSE_{gap} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(gap_i^{truth} - gap_i^{sim})^2} \quad (17)$$

- $PTDE$: The position trajectory deviation error (PTDE) is defined in Equation 18 by calculating the deviation between the observed and simulated positions throughout the entire trajectory of each individual follower.

$$PTDE = \sqrt{\frac{1}{V}\sum_{j=1}^{V}\Big[\frac{1}{L_j}\sum_{i=1}^{L_j}(x_{ji}^{truth} - x_{ji}^{sim})^2\Big]} \quad (18)$$

where *L* is the trajectory length (in sec) for j^{th} vehicle; *V* is the total number of followers simulated during the test stage; x_{ji} is the position of a j^{th} vehicle at i^{th} time, and *truth* and *sim* refer to actual and simulated measurements, respectively.

B. Car-following Model Calibration Methods

The following methods are considered for performance comparison. As described in Section III-B, IDM was adopted in this paper for experiments, and different calibration methods were used to calibrate IDM parameters either offline or online. Due to the length constraint, the training hyperparameters and settings, and other implementation details for different methods are thoroughly reported in our online supplementary materials [15]. The final calibrated parameters from each method are provided in this respective order: $\{v_0, T, gap_{jam}, acc_{max}, dacc_{max}, \delta\}$.

(i) **Default IDM** is the IDM with default parameters inferred from those values reported in existing studies [6], [32]. These values are: {33.3, 1.6, 2, 0.73, 1.67, 4}.

(ii) **GA** is the Genetic Algorithm used for static (offline) model calibration only. It was used to calibrate all IDM parameters. The fitness scores for GA were calculated based on the entire simulated trajectories of followers, instead of one-step prediction. The final GA solution is {13.3969, 0.3649, 1.8921, 0.9125, 3.3515, 3.1234}.

(iii) **s-DQN** is the static DQN-based calibration method formulated in [15]. In this case, IDM parameters were only calibrated offline and no trained Q-Network was used during the test stage. The calibrated parameters are {33.3, 1.1, 2, 1.34, 1.67, 8}.

(iv) **s-DDPG** is the static DDPG-based calibration method formulated in Section III-C. Similarly, no actor network was used during the test stage. The calibrated parameters are {22.7567, 1.4878, 2.9848, 1.9287, 1.0782, 6.1835}.

(v) **d-DQN** is the dynamic version of s-DQN. Each follower trajectory was initialised using the final set of offline-calibrated IDM parameters (by s-DQN), and then, throughout the test trajectory simulation, the IDM parameters were dynamically calibrated with the help of pre-trained Q-network.

(vi) **d-DDPG** is the dynamic version of s-DDPG. Each follower trajectory was initialised using the final set of offline-calibrated IDM parameters (by s-DDPG). After that, pre-trained actor network was used to dynamically calibrate the IDM parameters throughout the trajectory simulation.

C. Static Calibration Performance Assessment

TABLE II
CAR-FOLLOWING MODEL CALIBRATION PERFORMANCE COMPARISON

	$\mathbf{RMSE_{gap}}$	**PTDE**
IDM	10.9717	11.1216
GA	10.5127	10.5255
s-DQN	8.0482	8.5274
s-DDPG	7.9757	8.435
d-DQN	7.8805	8.3329
d-DDPG	**7.6949**	**8.0882**

In Table II, the CFM calibration performance of different methods are given. Since default IDM parameters were not specifically calibrated based on the training datasets, its performance is the worst among all the methods compared. Thus, future works should be aware of the underperformance of this default model when left uncalibrated, since in literature, only a small number of papers (nine percent of 994 CFM-related articles, gathered since 2005) attempted to perform parameter calibration using observation data [20].

By using the GA-calibrated parameter set, the error has been reduced by 4.18% and 5.36% in terms of $RMSE_{gap}$ and $PTDE$, respectively as compared against default IDM.

The offline calibration performance of both s-DQN and s-DDPG are competitive to each other. Their results are much superior to that of GA with a relative error decrease of about 24.13% and about 19.86% in terms of $RMSE_{gap}$ and $PTDE$, correspondingly. This is because GA suffers from premature convergence as we observed during its optimization process, and the best set of parameters found could turn out to be locally optimal solution instead of global one. Unlike GA, DRL-based methods: both s-DQN and s-DDPG, perform quite well in producing more optimal solution than GA, without getting stuck at local optimum. Due to their trial-and-error exploration nature during the training process, DRL-based methods are more robust against early convergence towards local solution.

D. Dynamic Calibration Performance Assessment

To quantify the performance of dynamic calibration, the best performing offline method, s-DDPG, is selected as the baseline in this section. The dynamic calibration performance of d-DQN is slightly better than that of s-DDPG, and its offline counterpart, s-DQN, achieving about 1.2% relative error decrease with the former, and about 2.28% relative error decrease with the latter in terms of $PTDE$.

Finally, d-DDPG has produced the best results among all models, resulting in an relative $PTDE$ decrease of about 4.11% when compared against offline s-DDPG. This higher performance gain (than DQN) could be due to the following reasons – (i) DDPG can handle high-dimensional continuous action spaces unlike DQN, and hence, in our experiments, while DDPG was used to calibrate all IDM parameters, DQN was only designed to calibrate three IDM parameters due to this limitation; and (ii) DDPG was trained using the improved reward function defined in Equation 10 that penalizes extreme parameter changes. Therefore, this feedback signal encourages DDPG for more stable exploration process whereas the DQN agent does not receive this negative feedback ($Error_{oob}$). This is illustrated in Figure 3 where the training performance of DQN and DDPG are compared. It appears that the training strategy of DDPG is more stable than that of DQN. Consequently, during the test stage, it has learnt to perform dynamic calibration better than d-DQN. In the figure, the DQN receives higher scores at some episodes merely due to the absence of this $Error_{oob}$ penalty.

Fig. 3. Training performance of DDPG and DQN

E. Qualitative Performance Assessment

A test follower trajectory was randomly selected for qualitative assessment. The simulated gaps produced by different calibration methods throughout the test follower trajectory are compared in Figure 4. Since this test trajectory has the length of 91 seconds, it is also helpful towards assessing how different calibration methods perform in a long car-following period. It is evident that when using calibrated parameters given by either default IDM or GA, the simulated gaps deviate a lot from the ground-truth gaps. The simulated gaps become much closer to the ground-truth values when the IDM parameters are dynamically calibrated by d-DQN. Finally, in alignment with the above quantitative assessment, d-DDPG has outperformed the rest of the methods because its simulated gaps match those of the ground-truth data more closely than others.

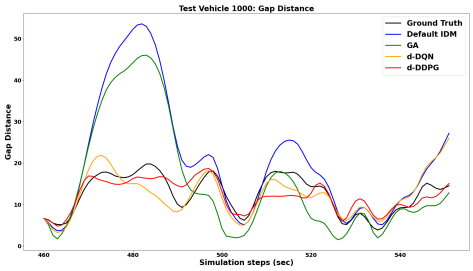

Fig. 4. Simulated gaps throughout a test follower trajectory

F. Visual Performance Assessment

As for visual assessment, SUMO [33] was used for animating the same test vehicle trajectory generated by different calibration methods. The trajectories were animated in parallel as shown in Figure 5. Throughout different parts of the trajectory, that are at (i) beginning (t=12 sec), (ii) middle (t=45 sec), and (iii) end (t=90 sec), it is obvious that the follower positions which were simulated by IDM which was dynamically calibrated by d-DDPG, are very similar to the observed positions. Thus, this indicates that dynamic CFM calibration with DRL can produce promising outcome which seems difficult to achieve with traditional offline calibration approaches.

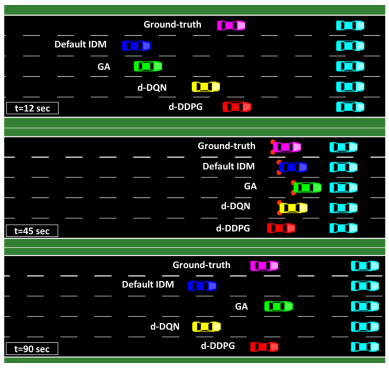

Fig. 5. Visual performance comparison for a test follower with ID 1000

V. CONCLUSIONS

In this paper, a new perspective on combining physics-based and learning-based models is given via "model calibration". Such a combination not only accounts for model analytical explainability, but also improves the model simulation accuracy by calibrating a physics-based CFM with DRL. Our experiment results have shown that a dynamic DRL-based calibration method can be an effective technique that offers promising results although more robust validation is still needed. Most importantly, when compared to a more conventional gradient-free optimisation algorithm such as GA, our dynamic calibration method, d-DDPG, has outperformed GA with a relative error decrease of 26.8% and 23.16% in terms of $RMSE_{gap}$ and $PTDE$, respectively. Therefore, our work could serve as an initial attempt towards this new research direction for CFM calibration.

ACKNOWLEDGMENT

This work was supported by Alibaba Group through Alibaba Innovative Research (AIR) Program and Alibaba-NTU Singapore Joint Research Institute (JRI), Nanyang Technological University, Singapore.

REFERENCES

[1] X. Wang, R. Jiang, L. Li, Y. Lin, X. Zheng, and F. Y. Wang, "Capturing Car-Following Behaviors by Deep Learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 3, pp. 910–920, 2018.

[2] R. Shi, Z. Mo, K. Huang, X. Di, and Q. Du, "A physics-informed deep learning paradigm for traffic state and fundamental diagram estimation," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[3] H. Naing, W. Cai, N. Hu, T. Wu, and L. Yu, "Data-driven microscopic traffic modelling and simulation using dynamic lstm," in *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2021, pp. 1–12.

[4] M. Zhu, X. Wang, and Y. Wang, "Human-like autonomous car-following model with deep reinforcement learning," *Transportation research part C: emerging technologies*, vol. 97, pp. 348–368, 2018.

[5] F. Hart, O. Okhrin, and M. Treiber, "Formulation and validation of a car-following model based on deep reinforcement learning," *arXiv preprint arXiv:2109.14268*, 2021.

[6] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[7] P. G. Gipps, "A behavioural car-following model for computer simulation," *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, 1981.

[8] S. Krauss, "Microscopic modeling of traffic flow: investigation of collision free vehicle dynamics," *Forschungsbericht - Dtsch. Forschungsanstalt fuer Luft - und Raumfahrt e.V.*, no. 98-8, 1998.

[9] V. Papathanasopoulou and C. Antoniou, "Towards data-driven car-following models," *Transportation Research Part C: Emerging Technologies*, vol. 55, pp. 496–509, 2015.

[10] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating physics-based modeling with machine learning: A survey," *arXiv preprint arXiv:2003.04919*, vol. 1, no. 1, pp. 1–34, 2020.

[11] Y. Tian, M. A. Chao, C. Kulkarni, K. Goebel, and O. Fink, "Real-time model calibration with deep reinforcement learning," *arXiv preprint arXiv:2006.04001*, 2020.

[12] L. Lin, W. Wu, Z. Shangguan, S. Wshah, R. Elmoudi, and B. Xu, "Hpt-rl: Calibrating power system models based on hierarchical parameter tuning and reinforcement learning," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 1231–1237.

[13] A. Unagar, Y. Tian, M. A. Chao, and O. Fink, "Learning to calibrate battery models in real-time with deep reinforcement learning," *Energies*, vol. 14, no. 5, p. 1361, 2021.

[14] X. Tang, B. Gong, Y. Yu, H. Yao, Y. Li, H. Xie, and X. Wang, "Joint modeling of dense and incomplete trajectories for citywide traffic volume inference," in *The World Wide Web Conference*, 2019, pp. 1806–1817.

[15] H. Naing. (2022) Supplementary materials: Dynamic car-following model calibration with deep reinforcement learning. [Online]. Available: https://github.com/Javelin1991/dynamicDRLcalibrationForCFM

[16] 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.

[17] 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*. PMLR, 2018, pp. 1861–1870.

[18] 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.

[19] V. Punzo, B. Ciuffo, and M. Montanino, "Can results of car-following model calibration based on trajectory data be trusted?" *Transportation research record*, vol. 2315, no. 1, pp. 11–24, 2012.

[20] V. Punzo, Z. Zheng, and M. Montanino, "About calibration of car-following dynamics of automated and human-driven vehicles: Methodology, guidelines and codes," *Transportation Research Part C: Emerging Technologies*, vol. 128, p. 103165, 2021.

[21] A. Sharma, Z. Zheng, and A. Bhaskar, "Is more always better? the impact of vehicular trajectory completeness on car-following model calibration and validation," *Transportation research part B: methodological*, vol. 120, pp. 49–75, 2019.

[22] M. Pourabdollah, E. Bjärkvik, F. Fürer, B. Lindenberg, and K. Burgdorf, "Calibration and evaluation of car following models using real-world driving data," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6.

[23] L. Li, X. M. Chen, and L. Zhang, "A global optimization algorithm for trajectory data based car-following model calibration," *Transportation Research Part C: Emerging Technologies*, vol. 68, pp. 311–332, 2016.

[24] D. Sha, K. Ozbay, and Y. Ding, "Applying bayesian optimization for calibration of transportation simulation models," *Transportation Research Record*, vol. 2674, no. 10, pp. 215–228, 2020.

[25] I. Markou, V. Papathanasopoulou, and C. Antoniou, "Dynamic car–following model calibration using spsa and isres algorithms," *Periodica Polytechnica Transportation Engineering*, vol. 47, no. 2, pp. 146–156, 2019.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[27] V. Punzo and M. Montanino, "Speed or spacing? cumulative variables, and convolution of model errors and time in traffic flow models validation and calibration," *Transportation Research Part B: Methodological*, vol. 91, pp. 21–33, 2016.

[28] J. Colyar and J. Halkias, "Us highway 101 dataset," Federal Highway Admin. (FHWA), Washington, DC, USA, Tech. Rep. FHWA-HRT-07-030, 2007.

[29] B. Coifman and L. Li, "A critical evaluation of the next generation simulation (ngsim) vehicle trajectory dataset," *Transportation Research Part B: Methodological*, vol. 105, pp. 362 – 377, 2017.

[30] C. Thiemann, M. Treiber, and A. Kesting, "Estimating acceleration and lane-changing dynamics from next generation simulation trajectory data," *Transportation Research Record*, vol. 2088, no. 1, pp. 90–101, 2008.

[31] R. W. Schafer, "What is a savitzky-golay filter?[lecture notes]," *IEEE Signal processing magazine*, vol. 28, no. 4, pp. 111–117, 2011.

[32] Z. Mo, R. Shi, and X. Di, "A physics-informed deep learning paradigm for car-following models," *Transportation research part C: emerging technologies*, vol. 130, p. 103240, 2021.

[33] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using sumo," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575–2582.