Physics-guided Graph Convolutional Network for Modeling Car-following Behaviors under Distributional Shift

Htet Naing[1]

I. INTRODUCTION

This document contains the supplementary materials of our paper, *"Physics-guided Graph Convolutional Network for Modeling Car-following Behaviors under Distributional Shift"*, which has been accepted for publication in the IEEE International Conference on Intelligent Transportation Systems (IEEE ITSC 2023). The paper provides all related supplementary information regarding the ablation study, data pre-processing, model implementation, model architectures, model training parameters (or hyperparameters), and model training algorithm. All the data-driven CFMs used in our paper and the car-following simulation environment were implemented in Python with the help of Pytorch [1] and PyTorch Geometric [2]. Genetic Algorithm (GA), was implemented using PyGAD [3].

II. ABLATION STUDY

On top of the analysis already given in our conference paper, the complete ablation study results using all three datasets are tabulated in Tables I, II, and III. Three PGML aspects introduced in our model are — 1) physics-guided loss, 2) physics-guided features, and 3) physics-guided edges. Accordingly, different model variants involved in this study are:

- PG-GCN-GRU-0: Our proposed model without physics-guided edges.
- PG-GCN-GRU-1: Our proposed model without physics-guided loss.
- PG-GCN-GRU-2: Our proposed model without physics-guided features.

We hope that these results can be helpful for future works interested in the improvement of our proposed model. More analysis of the results when testing with Dataset 3 (which exhibits the most severe distributional shift) is given below.

Interestingly, when disabling physics-guided edges, *CPGE* has improved slightly by a *relative error decrease* (RED) of about 0.97% whereas an improvement of about 0.5% was observed when removing physics-guided features. It should be noted that this pattern does not hold true for the ablation experiment results across all three datasets. Despite these marginal changes, these counter-intuitive results might imply that the calibrated IDM, which plays a crucial role in creating physics-guided features and edges, is itself affected by the distributional shift (as there is a slight error increase in its results when testing with *Dataset 3* as compared to *Dataset 1*). This issue will be scrutinized in our future works via more distributional shift scenarios such as testing with data collected from completely different study areas.

TABLE I

ABLATION STUDY USING DATASET 1

Model	CPGE	FC	RC	Collisions (%)
PG-GCN-GRU-0	7.4951	**0**	9.2	6.39
PG-GCN-GRU-1	**6.7185**	0.6	**7.4**	**5.56**
PG-GCN-GRU-2	7.1589	**0**	8.4	5.83
PG-GCN-GRU	7.4034	**0**	8.6	5.97

TABLE II

ABLATION STUDY USING DATASET 2

Model	CPGE	FC	RC	Collisions (%)
PG-GCN-GRU-0	6.2264	**0**	**18.4**	**2.62**
PG-GCN-GRU-1	9.8613	23.4	108.6	18.78
PG-GCN-GRU-2	**6.207**	**0**	19	2.7
PG-GCN-GRU	6.2095	**0**	18.6	2.65

TABLE III

ABLATION STUDY USING DATASET 3

Model	CPGE	FC	RC	Collisions (%)
PG-GCN-GRU-0	**7.4655**	**0**	**62.6**	**7.91**
PG-GCN-GRU-1	10.3523	48.8	167.2	27.31
PG-GCN-GRU-2	7.5012	**0**	72.6	9.18
PG-GCN-GRU	7.5386	**0**	64.4	8.14

III. DATA PRE-PROCESSING

The pre-processed data from our previous work[1] [4] was used as our starting point. Apart from the extra pre-processing procedure already described in our conference paper, the surrounding vehicle information were also extracted as follows. For a given vehicle, the longitudinal gap distances in six directions – 1) front gap, 2) rear gap, 3) left front gap, left rear gap, right front gap, and right rear gap – were extracted from the trajectory data. The vehicle sensing range was assumed to be 100m in both front and rear directions. If no neighbor was found within the range, the default gap of 100m was used with the default velocity set to 33.3333 m/s. Negative gap distances w.r.t the neighbors from the same lane were not allowed, and such data points were removed. There can be negative gap distances w.r.t the neighbors from adjacent lanes. Our proposed model depends only on front and rear gap information since the vehicle platoon graph (VPG) w.r.t the ego's current lane is considered. The rest of the gaps are used by our baselines such as variants of [5] that consider surrounding vehicle information as part of their input graphs.

Finally, after data pre-processing, the summary statistics of CF duration in the three datasets based on the format of *(min, median, max)* are — *(25, 32, 55)* sec for *Datasets 1*,

[1]https://github.com/Javelin1991/dynamicDRLcalibrationForCFM

(25, 35, 83) sec for *Datasets 2*, and *(25, 46, 78)* sec *Datasets 3*, respectively.

IV. Model Training Algorithm

Algorithm 1 is used to train our proposed model. The final result of the algorithm is our physics-guided data-driven CFM, f whose parameters Θ have been optimized with PGML. All the required training constant/model parameters and hyperparameters are initialized as described in lines 2-6. The weight parameters of GCN θ_S and GRU θ_T are jointly trained in this paper. It will be interesting for future works to consider spatial graph representation learning and downstream prediction tasks separately by optimizing based on different objectives. The maximum number of batches that can be created depending on the size of D_{tr} is calculated in line 6. D_{tr} and D_{val} contain all valid training and validation trajectories, respectively.

The training process begins in line 7, which lasts for E epochs. The training trajectories ζ are always shuffled at the beginning of each epoch in line 8. The model parameters are updated in a batch-by-batch manner as iterated in line 9. Since the batch size is defined as B, each batch of training data contains B number of trajectories. For each trajectory ζ_j in B, all valid input-output sample pairs $\mathcal{P}$: 1) *input*: ego-centered VPG from time $t-k+1$ to t, and 2) *output*: ground-truth and IDM output acceleration at $t+1$, are retrieved in lines 12-13. Then, in line 14, our *SIP* layer GCN is fed with the input feature matrix X along with its weighted adjacency matrix $\hat{A}$. After processing through the GCN layer, the ego vehicle's spatial contexts C are produced. These extracted spatial contexts from time $t-k+1$ to t are sequentially processed with the GRU layer that is also succeeded with another post-processing linear layer. Eventually, the final output acceleration for the ego vehicle is obtained in line 15. The current model parameters Θ are then updated by using the gradient ∇_Θ of the physics-guided loss $\mathcal{L}(\Theta)$ defined in Equation 1 in line 16.

For a more robust validation process that can help improve the model generalization, the CF simulation-based validation is performed by using the updated Θ in line 19. This is more robust than conventional one-step prediction validation because it can measure the model performance with the consideration of error accumulation during the simulation. Based on the simulation results, the validation error is calculated using Equation 2 for keeping track of the best model throughout the training process.

$$\mathcal{L}(\Theta) = \frac{1}{|\mathcal{P}|}\sum_{i=1}^{|\mathcal{P}|}\left[a_i^{data} - a_i\right]^2 + \lambda\left[a_i^{phy} - a_i\right]^2 \quad (1)$$

$$\sqrt{\frac{1}{V}\sum_{j=1}^{V}\left[\left(\frac{1}{L_j}\sum_{t=1}^{L_j}(s_{j,t}^{data} - s_{j,t})^2\right) + \mathbb{1}_{[L_j<N_j]}\gamma P_j\right]} \quad (2)$$

V. Proposed Model Implementation

In this section, implementation details about our proposed *Physics-guided Graph Convolutional Network with Gated Recurrent Unit (PG-GCN-GRU)* are provided.

Algorithm 1: Model Training Algorithm

Result: CFM f with trained parameters Θ;
1 **Initialization**:
2 Initialize training and validation data D_{tr} and D_{val}, respectively, containing $X, \hat{A}, a_{data}, a_{phy}$;
3 Initialize hyperparameters for f;
4 Initialize network weights $\theta_S, \theta_T \in \Theta$ randomly;
5 Initialize a lower and upper bounds: $[a_{LB}, a_{UB}]$;
6 Initialize maximum batch count $B_{max} = \lfloor \frac{|D_{tr}|}{B} \rfloor$;
7 **for** *epoch = 1, ..., E* **do**
8 Shuffle trajectories ζ in D_{tr};
9 **for** *batch = 1, ..., B_{max}* **do**
10 Get B trajectories from D_{tr};
11 **for** ζ_1, ..., ζ_B **do**
12 Get all valid training input and output sample pairs $\mathcal{P}$ from ζ_j;
13 Get $X_\mathcal{P}, \hat{A}_\mathcal{P}, a_\mathcal{P}^{data}, a_\mathcal{P}^{phy}$ from $\mathcal{P}$;
14 $C_\mathcal{P} \leftarrow GCN(X_\mathcal{P}, \hat{A}_\mathcal{P}|\theta_S)$ where $C_\mathcal{P} = [C_{t-k+1}, ..., C_t]_\mathcal{P}$;
15 $a_\mathcal{P} \leftarrow GRU(C_\mathcal{P}|\theta_T)$;
16 $\Theta \leftarrow \Theta - \alpha\nabla_\Theta\mathcal{L}(\Theta)$;
17 **end**
18 **end**
19 Perform simulation-based validation using D_{val};
20 Keep $f(\Theta)$ with the lowest validation error;
21 **end**

A. GCN Layer(s) for Spatial Information Processing

Our spatial information processing function is implemented using GCN layers as follows.

- Input: An input graph with each node having a feature vector of size 5, with their connections defined in a weighted adjacency matrix.
- Layer 1: *GCNConv* layer with 128 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 2: *GCNConv* layer with 128 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 3: *GCNConv* layer with 32 neurons, activated by *ReLU* function.
- Layer 4: *GRU-based Aggregation* layer from [6] with 32 neurons.
- Ego Feature Addition: The output *Hidden Platoon State* from the GRU layer is concatenated with an ego's original five features. Thus, the input size to the final *Linear* layer is 37.
- Layer 5: *Linear* layer with 16 neurons, activated by *ReLU* function.
- Output: Spatial context vector C of size 16.

B. GRU Layer(s) for Temporal Information Processing

Our temporal information processing function is implemented using GRU layers as follows The past trajectory (sequence) length k is set to 10.

- Input: A sequence of spatial context vectors C_t from time $t-k+1$ to t.
- Layer 1: *GRU* layer with 16 neurons.
- Layer 2: *GRU* layer with 10 neurons.

- Layer 3: *GRU* layer with 10 neurons.
- Layer 4: *Linear* layer with 1 neuron, activated by *Tanh* function.
- Scaling: The output is scaled based on the range $[a_{LB}, a_{UB}]$ set to [-9.5, 3.41].
- Output: Longitudinal acceleration.

C. Physics-guided Learning

- IDM parameters: The IDM constants parameters were pre-calibrated with GA (with the settings described in Section VI-B), and the calibrated values are [17.369, 1.0038, 2.1154, 0.9026, 0.5043] for $[v_0, T, s_0, a_{max}, b_{max}]$, respectively.
- Physics-guided loss: The weight parameter λ to control the physics loss contribution was set to 1.

D. Hyperparameters

The hyperparameters used in the training process are — batch size B = 8; training epochs E = 500; learning rate α = 0.0005; optimizer = Adam. The same hyperparameters were used to train all the models.

VI. Baseline Model Implementation

A. Intelligent Driver Model (IDM)

For default IDM, the constant parameter values were configured as those reported in [7]. They are [33.3, 1.6, 2, 0.73, 1.67] for $[v_0, T, s_0, a_{max}, b_{max}]$, respectively.

B. Genetic Algorithm (GA-IDM)

The GA training parameters used for IDM calibration are described below. The IDM parameter ranges defined for the feasible search region were — [10, 33.3333]m/s for maximum desired velocity v_0; [0.3, 6]s for desired time headway T; [1, 5]m for jam spacing distance s_0; [0.28, 3.41]m/s^2 for maximum desired acceleration a_{max}; [0.47, 3.41]m/s^2 for maximum desired deceleration b_{max}.

The GA training parameters were:

- Maximum number of generations was set to 100.
- Population size was set to 100.
- Mutation method was *uniform* random.
- Mutation probability was set to 0.3.
- Elite ratio for the proportion of elites kept in the population was set to 0.01.
- Crossover method was *uniform* crossover.
- Crossover probability was set to 0.5.
- Proportion of mating parents in the population was set to 0.5.
- Parent selection type was set to *tournament selection*.
- Stopping criterion was set to *none* (i.e., run for all complete generations).

After running five times with different random seeds, the calibrated values for different runs are given in Table IV.

TABLE IV
IDM parameters calibrated with GA

Run	v_0	T	s_0	a_{max}	b_{max}
1	18.0934	1.0576	1.0999	0.8178	0.5024
2	16.9161	0.9251	1.4074	0.8230	0.5083
3	16.9178	0.9127	2.1809	0.8905	0.5667
4	17.8402	1.0766	1.8882	0.8864	0.5873
5	17.7691	1.0802	1.2605	0.8759	0.5820

C. Feed-forward Neural Network (FFN)

The FFN architecture was implemented as follows.

- Input: A feature vector of size 11.
- Layer 1: *Linear* layer with 128 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 2: *Linear* layer with 128 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 3: *Linear* layer with 128 neurons, activated by *ReLU* function.
- Layer 4: *Linear* layer with 1 neuron, activated by *Tanh* function.
- Scaling: The output is scaled based on the range $[a_{LB}, a_{UB}]$ set to [-9.5, 3.41].
- Output: Longitudinal acceleration.

D. Graph Convolutional Network (GCN)

The GCN architecture was implemented as follows.

- Input: An input graph with each node having a feature vector of size 11, with their connections defined in an adjacency matrix.
- Layer 1: *GCNConv* layer with 128 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 2: *GCNConv* layer with 128 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 3: *GCNConv* layer with 128 neurons.
- *Global mean pooling*: Used to aggregate all node embeddings.
- Ego Feature Addition: The aggregated embedding is concatenated with an ego's original 11 features. Thus, the input size to the next layer is 139.
- Layer 4: *Linear* layer with 22 neurons, activated by *ReLU* function.
- Layer 5: *Linear* layer with 1 neuron, activated by *Tanh* function.
- Scaling: The output is scaled based on the range $[a_{LB}, a_{UB}]$ set to [-9.5, 3.41].
- Output: Longitudinal acceleration.

E. Graph Convolutional Network with Gated Recurrent Unit (GCN-GRU)

The GCN-GRU architecture was implemented as follows.

- Input: An input graph with each node having a feature vector of size 11, with their connections defined in an adjacency matrix.
- Layer 1: *GCNConv* layer with 128 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 2: *GCNConv* layer with 128 neurons, followed by *layer normalization*, activated by *ReLU* function.

- Layer 3: *GCNConv* layer with 128 neurons.
- *Global mean pooling*: Used to aggregate all node embeddings.
- Ego Feature Addition: The aggregated embedding is concatenated with an ego's original 11 features. Thus, the input size to the next layer is 139.
- Layer 4: *Linear* layer with 11 neurons, activated by *ReLU* function. The output from this layer is the spatial context vector C. A sequence of spatial context vectors C_t from time $t-k+1$ to t is fed to the next *GRU* layer.
- Layer 5: *GRU* layer with 30 neurons.
- Layer 6: *GRU* layer with 10 neurons.
- Layer 7: *GRU* layer with 10 neurons.
- Layer 8: *Linear* layer with 1 neuron, activated by *Tanh* function.
- Scaling: The output is scaled based on the range $[a_{LB}, a_{UB}]$ set to [-9.5, 3.41].
- Output: Longitudinal acceleration.

F. Improved GCN-GRU: GCN-GRU++

The GCN-GRU++ architecture was implemented the same as our proposed model (described in Section V) except that it does not incorporate any PGML aspects. Thus, it has an input graph with each node having a feature vector of size 3. It only uses a (0,1) adjacency matrix and does not use the physics loss as part of its loss function.

REFERENCES

[1] M. (a.k.a. Facebook), "Pytorch," https://pytorch.org, 2023.

[2] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[3] A. F. Gad, "Pygad: An intuitive genetic algorithm python library," 2021.

[4] H. Naing, W. Cai, T. Wu, and L. Yu, "Dynamic car-following model calibration with deep reinforcement learning," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 959–966.

[5] J. Su, P. A. Beling, R. Guo, and K. Han, "Graph convolution networks for probabilistic modeling of driving acceleration," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–8.

[6] D. Buterez, J. P. Janet, S. J. Kiddle, D. Oglic, and P. Liò, "Graph neural networks with adaptive readouts," *arXiv preprint arXiv:2211.04952*, 2022.

[7] V. Punzo, M. Montanino, and B. Ciuffo, "Do we really need to calibrate all the parameters? variance-based sensitivity analysis to simplify microscopic traffic flow models," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 184–193, 2014.