DJGRAD: Sparse Gradients Protocol for Distributed Assisted Learning in CAVs

Matt Raymond
Computer Science and Engineering
University of Michigan
Ann Arbor, MI, USA
mattrmd@umich.edu

Brian Tang
Computer Science and Engineering
University of Michigan
Ann Arbor, MI, USA
bjaytang@umich.edu

Matt Wilmes
Automotive Engineering
University of Michigan
Ann Arbor, MI, USA
mwilmes@umich.edu

***Abstract*—Performance of connected and autonomous vehicles (CAVs) is limited by the amount of data that can be collected for training their machine learning (ML) models. Current paradigms require expensive fleets of training vehicles, or sharing customer data with original equipment manufacturers (OEMs). These methods are expensive and lead to privacy concerns. We propose DJGRAD, a novel distributed ML protocol for distributed, asynchronous, consensus-free, real-time training on consumer CAVs. Empirical results show that our method is capable of learning with strict real-time constraints, improves performance over individually-trained models, and is resistant to black-box adversarial attacks.**

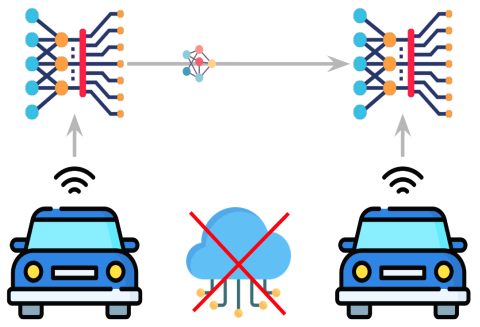


Fig. 1: DJGRAD is a real-time distributed learning protocol that utilizes DSRC to communicate directly with neighboring vehicles, as opposed to leveraging cell towers or the cloud. This avoids the need to share data with OEMs and allows for consensus-free models customized to each driver.

I. INTRODUCTION

A. Overview

Machine learning's (ML) usage in modern vehicles is becoming increasingly ubiquitous, especially with the emergence of deep neural networks (DNNs). Currently, original equipment manufactures (OEMs) such as Tesla, GM, BMW, Nissan, and Volkeswagen are under constant pressure from competitors, and are therefore incentivized to constantly improve their ML models for connected and autonomous vehicles (CAVs). Traditionally, OEMs collect data from private CAV fleets or simulations [1, 2]. However, it is impossible to find every possible corner case, so models will eventually meet a condition that falls out of their training distribution. Current ML theory assumes that data is independent and identically distributed (i.i.d.), meaning that samples are drawn from the same distribution and the value of the current sample has no effect on the next sample drawn. Since extrapolation violates this assumption (because the data is outside of the training distribution) models cannot reliably extrapolate to handle anomalous conditions.

Recent works have developed models that can handle situations outside of its training distribution, so-called zero-shot methods [3] or bio-inspired networks [4], but these methods perform worse in other conditions or require significantly more computing power. Furthermore, it is unclear how these methods adapt to the reliability needed in autonomous vehicle settings. Therefore, OEMs are incentivized to collect data from consumer vehicles to expand their datasets [5] to include millions of kilometers of roads. For example, CAVs could continually send driving data to a cloud server to train a centralized model. However, privacy-conscious users may be reluctant to share their data with an OEM. Additionally, a centralized computing center would require significant upgrades to communication infrastructure to handle massive data transmissions. Utilizing a centralized model may reduce a CAV's performance in niche environments in favor of generality.

Providing privacy-preserving, cost-efficient, and customized user experiences with centralized models is difficult. While federated learning (FL) seeks to alleviate privacy concerns [6], this comes at the cost of accuracy when training on heterogeneous data. Other work [7] has shown that original data can still be reconstructed from privacy-aware collaborative training schemes such as FL.

Alternatively, CAVs can improve their models in a distributed learning setting. These models are trained on the user's vehicle and share model parameters using vehicle-to-vehicle (V2V) communication protocols. Cellular networks rely on a centralized and trusted entity, exhibit high latency, and lack ubiquity, so dedicated short-range communications (DSRC) are preferable. The trade-off is that cellular networks have a wide area of coverage, whereas vehicles with DSRC can only broadcast or send unicast messages to vehicles within a range of about 300 m [8]. Since vehicles are geographically distributed non-uniformly, we cannot guarantee that all updates to model parameters will be shared with all other vehicles. Therefore, the performance of a distributed protocol must not rely on a consensus model, and decentralized FL paradigms [9] cannot be used.

B. Goals

The goal of this project is to create a network protocol capable of sending machine ML model updates between CAVs that are level 2-4 (human-in-the-loop), without the need for cloud communication or cellular networks (Fig. 1). This protocol will ensure ML models in CAVs will continuously perform well and adapt to local driving environments. Parameter updates will propagate between nearby CAVs while driving; thus, the proposed protocol must be lightweight and runtime efficient.
The contributions of our work are as follows:

- We design and evaluate the efficacy of DJGRAD, a real-time distributed learning protocol, in the context of V2V communications. Specifically, we leverage the WAVE (wireless access in vehicular environments) short message protocol (WSMP) to advertise, request, and send gradients between vehicles via DSRC. Our evaluation finds that gradients can spread to over 90% of vehicles in both a highway and downtown scenario, even with a 10% packet drop rate.
- We evaluate the holdout performance of DJGRAD on a regression task as a surrogate for Deep Q-Learning. We show that DJGRAD improves performance over independent models, even with imperfect communication. The statistical significance of DJGRAD's performance is similar to that of FL.
- We perform a security evaluation on DJGRAD and find that it is resistant against attacks in the black-box setting. The security evaluation consists of adapting three types of attacks to the distributed learning setting. These attacks seek to harm model performance, inject hidden backdoors, and reconstruct training data. While DJGRAD sacrifices some performance when compared to federated learning, DJGRAD provides higher security and privacy guarantees.

II. BACKGROUND

A. Machine Learning Primer

Consider a space $\mathcal{Z}$ of the form $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the input/sample space and $\mathcal{Y}$ is the output space. For example, in Section IV-C our experiments utilized inputs of attributes such as weather, date, temperature, etc. and outputs of number of bikes being used. In Section IV-D, we consider the task of image classification where inputs are images and outputs are labels categorizing these images. Let $\mathcal{H}$ be a hypothesis space (e.g., the weights of a DNN). We can create a loss function $L : \mathcal{H} \times \mathcal{Z} \mapsto \mathbb{R}$ that measures the closeness of a hypothesis to a ground truth. This learning algorithm results in a classifier F which accepts an input $x \in \mathcal{X}$, and outputs $y \in \mathcal{Y}$.

B. Reinforcement Learning

Reinforcement learning (RL) is an AI technique in which an agent learns how to interact with an environment based on the rewards it receives for its actions. These rewards typically correspond with an action that results in a positive outcome (positive reward) or a negative outcome (negative reward) for the agent in question. Typically, RL requires a "model" of its environment so it can make predictions and maximize its expected reward. However, Q-Learning enables model-free RL by learning a real-valued Q-function to optimize its prediction of expected rewards. *Deep Q-Learning* uses a DNN to estimate the Q-function.

C. Related Works

1) Federated Learning: Recent advances in machine learning systems allow multiple data owners to engage in collaborative training, or FL [10]. FL's decentralized training and global parameter aggregation provide data owners with privacy guarantees not possible with traditional "data-aggregating" ML approaches. However, most FL paradigms assume a centralized server to aggregate parameter updates [11], and reliable communication with edge devices. Although FL is more secure than direct data transmission (because it does not explicitly aggregate data), there are various exploits that extract data from model parameters [12, 13], and vehicular networks are notoriously unreliable [14]. There is recent work on distributed FL [15] that address distributed communication with intermittent connections, and asynchronous FL [16] that allows for asynchronous parameter updates over unreliable networks. However, even in optimal conditions, a decentralized consensus model will not be customized to individual user experiences.

2) Assisted Learning: Assisted learning [17] (AL) is similar to FL, but does not intend to learn a consensus model. Instead, each agent a_i has its own goal $\mathcal{G}_i$ and data $\mathcal{D}_i$ that must remain private. Instead of sharing data or model parameters, they share data IDs and residuals, which allow them to train on an arbitrary $\mathcal{D}_j$ without having direct access to it, or sharing $\mathcal{G}_i$. This is excellent for data privacy, but requires reliable connections and long-term, full-duplex communication, which is not realistic in CAVs. Also, note that in our setting $\forall i,j\ \mathcal{G}_i = \mathcal{G}_j$, as all vehicles want to improve driving performance. Therefore, AL enforces higher privacy constraints than we need.

3) Gradient Updates: Many self-driving systems rely on DNNs [18], which exhibit state-of-the-art performance in many data-heavy ML applications. Their expressiveness comes from a chain of linear transformations and non-linear "activation functions," which warp a given dataset into a linearly-separable representation. These linear transformations are expressed as $m \times n$ matrices, where m and n are consecutive layers. Finding analytical solutions to a network is impossible because they are typically highly over-parameterized. Therefore, models are usually updated using stochastic gradient descent. During training, a model is used to generate predictions using inputs. Residuals, an approximation for error, can be calculated from the margin between the predicted values and the true values. These residuals are used to calculate gradients for each of the model weights. These gradients are then used by the model to learn the dataset and hypothesis classes. Gradient updates scale as DNNs grow larger, so sharing updates between CAVs in real-time becomes a challenging task.

Some distributed DNNs use aggregated gradients, where the gradients are accumulated over several epochs before being shared [19]. This can reduce networking overhead, but can also lead to training instability and suboptimal performance. Although this method decreases total network traffic, it still requires full gradient updates to be transmitted. However, with DSRCs, our main concern is the maximum transfer time, since two vehicles are not guaranteed to remain in communication range for more than a few seconds. Therefore, gradient caching is not applicable here.

Previous work has attempted to sparsify gradient matrices [20], as sparse matrices can be significantly compressed. Early works drop gradients below a certain threshold, but they tend to converge with suboptimal performance [21]. Recent work has proposed a combination of gradient caching and sparsified gradients, where small terms are added to the next gradient matrix rather than being dropped [20]. However, current thresholding techniques do not guarantee a consistent number of non-zero terms [20] and therefore is not conducive to real-time communication.

III. Proposed Design

A. Transmission Requirements

To ensure rapid propagation of model updates, V2V transmission must be reliable in many driving conditions. To keep the packet drop rate of DSRC below 10%, we limited our transmission radius to 100 m [22]. DSRC can transmit at rates anywhere between 6 and 27 Mbps [23], with a maximum transmission latency of 150 ms [8]. Because of packet drops and transmission rates that can vary based on vehicle proximity, speed, and occlusions, we did not assume a fixed transmission rate of data. Instead, we parameterized based on the successful transmission time (STT).

If a vehicle must send a total of N valid gradient messages and it takes t_i seconds to send message i, the STT can be calculated as the discontinuous sum of the time it takes to send these packets.

$$STT = \sum_{i=1}^{N} t_i \tag{1}$$

For example, we might require a vehicle to have an STT of 1 second, meaning the total transmission time could be greater than 1 due to dropped packets, but the total discontinuous sending time of valid gradient messages packets is exactly 1 second.

Furthermore, DSRC uses WSMP messages for communication, so any transmitted messages must use this same protocol. A very common message that uses WSMP is the basic safety message (BSM), which is used for safety-critical communication such as sharing vehicle position and velocity. WSMP also supports non-critical message types, such as the WAVE service advertisement (WSA) [24]. Finally, for sending generic data, WSMP supports the standardized WAVE short message (WSM) [25]. Since our protocol is not critical to the safety of the driver and we need to send non-standard gradient data, we limited ourselves to WSAs and WSMs.

B. *Network Protocol*

Since our setting assumes level 2-4 CAVs, we assume that our ML model is an RL algorithm [26], ideally using Deep Q-Learning with user interaction as negative reinforcement[1]. User intervention triggers local model updates, generating a new set of gradients θ_δ which are sparsified and stored for transmission. Whenever the CAV receives a sparse gradient matrix $\hat{\theta}_\delta$ from another vehicle, it applies the updates and stores them in a queue for transmission. Our protocol uses the following features to ensure model performance:

1) Sparse Gradients: As θ_δ is calculated, it is split into a disjoint set of sparse gradients (henceforth simply referred to as gradients) such that[2]:

$$\theta_\delta = \sum_i^m \theta_\delta^{(i)} \mid \forall i,j,\ i \neq j \iff \left(\theta_\delta^{(i)} \odot \theta_\delta^{(j)} = 0\right)$$

For example:

$$\underbrace{\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}}_{\theta_\delta} = \underbrace{\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}}_{\theta_\delta^{(1)}} + \underbrace{\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}}_{\theta_\delta^{(2)}}$$

The cardinality m of the partition for each full gradient is a tunable hyperparameter. Each time CAV a_t interacts with CAV a_r, a random $\theta_\delta^{(i)}$ is transmitted. Since these gradients will be forwarded by other CAVs, the full gradients will eventually propagate through the environment and all CAVs will theoretically apply the full update θ_δ.

2) Vehicle Handshaking: Broadcasting full gradients would be data intensive and inefficient. Instead, a vehicle with gradients (the ego vehicle) periodically broadcasts a WSA, indicating it has gradients to share. The message contains a unique identifier that other vehicles can use when requesting. Once a vehicle receives the WSA, it may send a request message (in the form of a WSM) back to the ego vehicle; the WSM contains the original unique identifier. Finally, the ego vehicle transmits $\theta_\delta^{(i)}$ as a unicast message back to each vehicle i that requested gradients.

3) Gradient Forwarding: Once vehicle i has received the gradients, it can start broadcasting WSAs of its own. $\theta_\delta^{(i)}$ is also added to a forwarding queue Q_f, which holds outgoing gradients. If Q_f is full, the first element is popped off. This provides a delay before transmitting gradients, so that CAV a_r will forward to CAVs that have not already received $\hat{\theta}_\delta$ from a_t.

[1]In order to meet the scope of the project, we assume that all CAVs have the same ML model architecture.

[2]Note that $\odot$ represents the Hadamard product.

4) Interaction List: Each CAV a_i maintains a queue $Q_v^{(i)}$ of vehicles that it will not communicate with. Every time CAVs a_t, a_r interact, a_t is added to $Q_v^{(r)}$ and vice versa. If $Q_v^{(i)}$ is full, the top vehicle will be removed. This prevents two CAVs from engaging in an infinite loop of gradient forwarding, since they will only communicate once. The length of this queue is a tunable hyperparameter.

5) Model Update: Gradient updates must happen quickly for safety reasons. Therefore, updates will be applied to a duplicated set of weights held in memory, and the model will load the new set of weights in between execution cycles. The duplicate weights will double the amount of space required for storing model weights (2GB of space in the most extreme cases). This overhead may cause the model to miss a frame. However, a CAV's automatic braking system (ABS) is not connected to its self-driving system and will not be impacted.

A flowchart of the full protocol is shown in Appendix A (Fig. 9).

IV. Methodology

A. *Experiment Designs*

We are interested in understanding the following aspects of our protocol:

1) **Networking: How well do model gradients spread in realistic driving environments?** Veins (Vehicles in Network Simulation) is a bidirectionally coupled simulation framework integrating both a traffic simulator and a network simulator [27]. We implemented the V2V communication aspect of our protocol and simulated it to quantify how gradients propagate throughout the network of CAVs.
2) **Learning: Does our distributed learning protocol positively impact learning?** We constructed a composite model in TensorFlow [28], providing each sub-model with a non-i.i.d. subset of training data and simulating sparse gradient transfers during the centralized training step. In Section V-B, we demonstrate that this results in improved individual and generalized performance over individual models with non-i.i.d. datasets. We use low-dimensional regression datasets (non-vehicular) to shorten simulation times and emulate Deep Q-Learning.
3) **Security & Privacy: How can a malicious vehicle attack other vehicles by exploiting this distributed learning protocol?** More specifically, we are interested in learning how to harm the performance of other

vehicles' models, inject backdoors into other vehicles' models, and steal data from received gradients. This was evaluated by performing existing attacks on our protocol in a setting where the adversary does not have access to other vehicles' models (black-box setting) [12, 13].

B. Networking

To understand the V2V communication aspect of our protocol, we leveraged simulation since it was not feasible to study it with actual vehicles. Two main components were required to replicate our protocol: a traffic simulator and a networking simulator. For the traffic simulator we used SUMO (Simulation of Urban Mobility) [29]. For the network simulator, we used OMNeT++ (Objective Modular Network Testbed in C++) [30]. These two simulators cannot communicate directly, but Veins is specifically designed for this [27]. Therefore, we leveraged Veins to help us assess how well gradients could propagate throughout the vehicular network.

1) Traffic Simulator: The traffic simulator is necessary to allow the vehicles in the simulation to drive on roads just like they would in the physical world. Roads can be created using nodes and edges, but instead we leveraged OpenStreetMap data for more realistic driving scenarios [31]. For variability, we chose two maps, one of a highway and one of a downtown area, both in the greater Minneapolis area of Minnesota. The highway map we used is shown in Fig. 2.

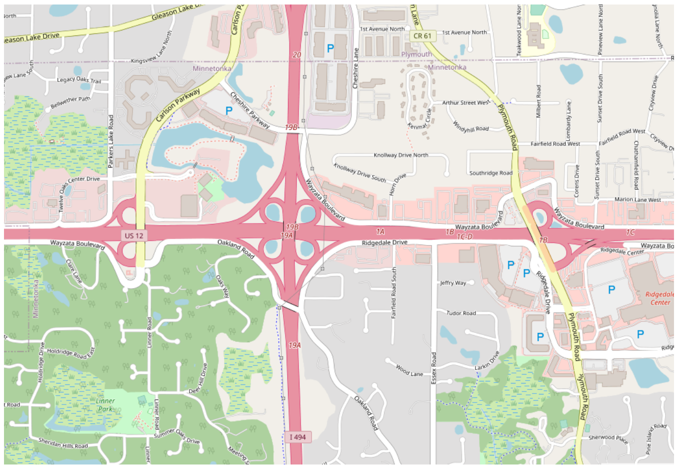


Fig. 2: OpenStreetMap map used for highway scenario.

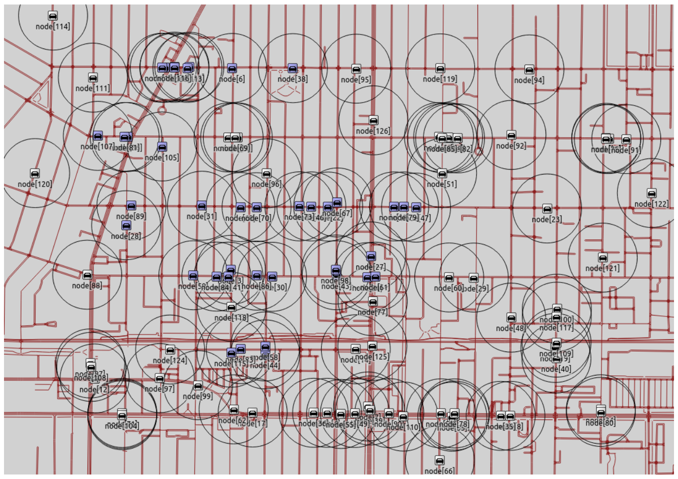


Fig. 3: Simulation of downtown scenario. Circles represent 100 m communication range. Vehicles shaded blue have gradients and vehicles shaded gray do not.

The advantage of the highway scenario is that vehicles are near one another for longer, giving them more time to communicate with each other. However, due to the linear nature of highways, a given vehicle may not interact with as many different vehicles, potentially limiting its ability to obtaining gradients. The advantage of the downtown scenario is that a given vehicle will pass by many other vehicles, giving it more chances to pass a vehicle with gradients. However, due to the variety of paths any given vehicle may take, this can limit how long a vehicle is next to another vehicle.

Once we imported the OpenStreetMap data into SUMO, random vehicle routes were created using SUMO's `randomTrips.py` and `duarouter` scripts. These scripts output XML files that are parseable by OMNeT++.

2) Integrating with Network Simulator: We coded our protocol in C++ using libraries provided by Veins. Their libraries include models of IEEE 802.11p and IEEE 1609.4 DSRC/WAVE [27], which were essential for mimicking the behavior of DSRC in our vehicular network. Classes are included that implement the DSRC/WAVE stack, but we had to implement functions to define the behavior of vehicles every time they entered the scene, their position changed, they received a WSA, they received a WSM, or they received a signal to send a message. Furthermore, we wrote specific message structures for our custom WSMs: one for the request message and one for the gradient message. The behavior of the vehicle when receiving a request WSM, naturally, had to be programmed differently than the behavior when receiving a gradient WSM.

We created two applications, one which required vehicles to have an STT of 1 second and another which required vehicles to have an STT of 2 seconds.

As demonstrated by Eq. (1), the longer the STT, the larger the gradients are assumed to be. Testing two STT values allowed us to test our hypothesis that gradients need to be kept as small as possible (i.e., sparsified). For further details of our applications, please see the `veins-dsrc/src/` folder on our GitHub page.[3]

3) Setting up the Simulations: Once our applications were written, an `omnetpp.ini` file was created for both the highway and downtown scenarios. This file allows for configuration of different settings for the simulation. For all of our simulations, one vehicle started with gradients, the WSA periodicity was set to 10 seconds, the communication range was set to 100 meters, channel switching was enabled, and the simulation time was set to 1,000 seconds. A launch configuration file determined which scenario to run. The packet drop rate was controlled by changing the `frameErrorRate` of the MAC layer. For our experiments, we used values of 0% and 10%. With two variables to study (STT and packet drop rate), each with two different values, this led to four experiments for each of our two scenarios. One of our simulations for the downtown scenario is shown in Fig. 3. Further details of our simulations can be found on our GitHub page in the `veins-dsrc/simulations/` folder.

C. Learning

1) Simplifications: It is common for CAVs to each have several GPUs for ML inference [32], making it impossible to simulate thousands of CAVs with realistic ML models. Therefore, we perform our learning simulations with four CAVs, which each have a simplified model. This allows us to run all simulations on a single NVIDIA RTX 3080. Real-time negative reinforcement from drivers is infeasible, so we treat each CAV as an unmoving graph node with a static dataset. Since they are not moving, each CAV will communicate with a predefined set of "neighbor" CAVs (see Fig. 4).

2) Datasets: Because we are running several ML models on a single consumer-level GPU, we need a low-dimensional dataset (few features) with patterns that can be learned by a shallow neural network (NN). Additionally, we are simulating Deep Q-Learning (real-valued output), so the dataset must contain real-valued labels. This dataset must also be large enough to split into four parts without impacting training. We used the UCI Bike Sharing Dataset [33], which has 14 features and 17,389 samples. Since this dataset is i.i.d., we impose a non-i.i.d. distribution by splitting the dataset based on the time of day each measurement was taken (0:00-5:00, 6:00-11:00, 12:00-17:00, and 18:00-23:00). This leaves us with approximately 4,000 samples per data subset.

3) Training: All simulations were implemented in TensorFlow using the Keras library [34]. We extended the base model class to contain four submodels, which each trained on their own data subset. On evaluation, each submodel was tested on holdout data that was representative of the original (non-split) dataset. We used this paradigm to implement independent models (no sharing), FL (sharing all gradients immediately), and DJGRAD (sharing sparse gradients with a delay). Our implementation of DJGRAD includes all features described in our gradient forwarding protocol, and includes a hyperparameter p that determines the probability that a sparsified gradient will *not* be dropped while being forwarded. During training, each model's "gradient sharing protocols" were executed after each batch. Each model was simulated for 4,000 epochs, taking approximately 6 hours per simulation.

4) Evaluation: We evaluated our model on an i.i.d. holdout dataset, and computed the holdout loss at every epoch.

D. Security & Privacy

1) General Assumptions: In the scenario where a malicious driver exploits the gradient sharing protocol to harm other models or steal data, we need to understand how our protocol functions under these attack assumptions. In the most realistic scenario, an adversary is

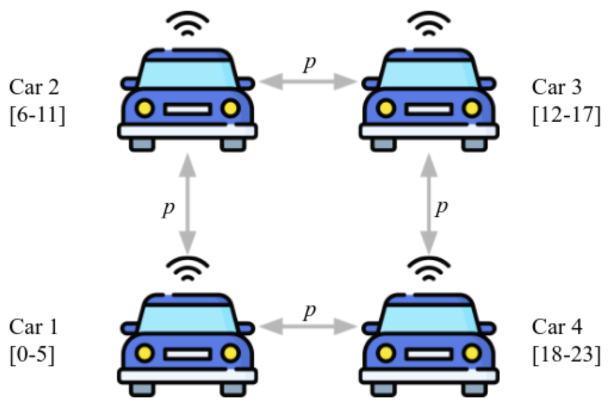


Fig. 4: Four simulated CAVs, each forwarding gradients to its two closest neighbors with a probability p (hyperparameter). Each vehicle has access to data from 1 of 4 different 6-hour time blocks (e.g., [0 – 5]).

[3]https://github.com/m4ttr4ymond/DJGrad

assumed to be operating under a black-box attack setting where it has access to their own vehicle's models and any gradients sent or received. However, the adversary can still employ black-box attacks on other vehicles' models by performing attacks on their own model as a sort of "surrogate model". As a baseline comparison, experiments will be performed in the white-box setting as well, where the adversary has access to all data, gradients, and models. The attacks are formulated using the same 4-vehicle simulation as denoted in Section IV-C. Additionally, as we were unable to successfully attack our protocol when using disjoint gradient subsets, we assume that the full gradients are shared in each attack scenario. In the following sections, we detail the three types of attacks constructed for our protocol with the intention of: harming the performance of other vehicles' models, injecting backdoor triggers in other vehicles' models, and reconstructing training data from gradients received by the adversary's vehicle.

2) Models and Datasets: In our experiments, we explore the efficacy of these attacks in the context of the image classification task, using datasets such as MNIST [35] and CIFAR-100 [36]. Our protocol, DJGRAD, leverages small 4-layer (LeNet [37]) and 7-layer convolutional neural networks (CNNs).

3) Harm Performance: The subject of adversarial machine learning and robust classification has been studied extensively in the past [38]. An area of particular interest is attacking models by degrading performance. For example, Xiao et al. [39] perform label-flipping attacks on support vector machines (SVMs). Instead of flipping labels, we can similarly attack DNNs by computing gradients from inverted loss metrics $-L$. For this attack scenario, we train the DNNs of each vehicle using 8 epochs in our distributed learning setting, each achieving accuracies of more than 98%. Afterwards, we pollute and distribute gradients to other vehicle models in both the white-box and black-box settings. Reproducing this attack is available in our GitHub page in the `security/` folder.

4) Inject Backdoors: Here the adversary wishes to compute a poisoned set of gradients θ_δ that, when applied to a ML model F, result in maliciously injecting backdoors for certain triggers λ. The presence of the backdoor trigger in an input, $x_i + \lambda$ causes the model to output a particular target label y_t. An example of a backdoor trigger is given in Fig. 5. Previous work has explored poisoning data through backdoor attacks [40, 41] as well as data poisoning attacks in federated learning settings [42, 43]. Once again, after training for 8 epochs,

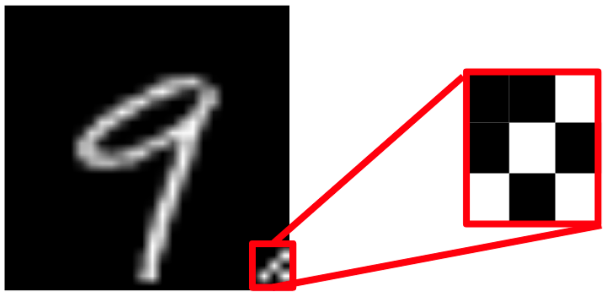


Fig. 5: An example of a backdoor trigger inserted into the input sample.

we poison inputs, compute gradients, and distribute the poisoned gradients to other vehicle models in both the white-box and black-box settings. Reproducing this attack is available in our GitHub page in the `security/` folder.

5) Steal Data: The final attack we are interested in is the case where an adversary reconstructs the original input data to a DNN using only the gradients and a model. This technique was discovered by Zhu et al. [7] and showed that data could still be leaked even when FL is employed. The algorithm to reconstruct data uses randomly initialized "dummy data", an optimization procedure known as gradient matching. Zhu et al. already evaluate their method on sparsified and compressed gradients and find that they are unable to recover data when sparsity reaches 20% or more. To convert this attack scenario into the black-box setting, we initialize the weights of multiple DNNs to the same randomized values (between -0.5 and 0.5). Afterwards, we perturb the weights of the "victim model" by a threshold ϵ. Finally, using 10 initializations, we attempt to reconstruct the original data from the received gradients using only the "surrogate model". Reproducing this attack is available in our GitHub page in the `security/dlg/` folder.

V. RESULTS

A. Networking

1) Highway Scenario: The results from our highway scenario simulations are shown in Table I. For plots of the percent of vehicles with gradients over the entire duration of the simulations, see Fig. 10 in Appendix B. For the simulation with a 1 second STT and 0% packet drop rate, 91.94% of the vehicles had gradients. This demonstrates that the gradients spread very well. We were not anticipating achieving 100% spread because some vehicles exited very early in the simulation before

Highway Scenario		**STT**	
		1 s	2 s
Packet Drop Rate	0%	91.94%	34.41%
	10%	90.86%	29.68%

TABLE I: Simulation results from highway scenario using different values of successful transmission time and packet drop rate. Values indicate the percentage of vehicles at the end of the simulation with gradients. Each simulation was run for 1,000 seconds.

Downtown Scenario		**STT**	
		1 s	2 s
Packet Drop Rate	0%	93.40%	46.53%
	10%	92.19%	6.94%

TABLE II: Simulation results from downtown scenario using different values of successful transmission time and packet drop rate. Values indicate the percentage of vehicles at the end of the simulation with gradients. Each simulation was run for 1,000 seconds.

they came within range of the vehicle that started with gradients.

When we increased the STT to 2 seconds, the percentage of vehicles with gradients dropped to 34.41%. This is not too surprising as it doubled the amount of time vehicles had to be within range of each other. When vehicles are driving opposite directions on the highway, it is not unreasonable to expect 1 second of interaction between them, but 2 seconds is less common. This solidifies our hypothesis that gradients need to be sparsified as much as possible to keep the STT as small as possible. However, we were still pleased with the result of 34.41% because without our protocol, vehicles would not share any of their gradients.

Increasing the packet drop rate to 10% did not have as much of a negative impact as we expected. The percentage of vehicles with gradients at the end of the simulation only dropped slightly as a result of this parameter change. Despite the 10% packet drop rate, more than 90% of the vehicles had gradients at the end of the simulation for the 1 second STT simulation and almost 30% of the vehicles had gradients for the 2 second STT simulation.

2) Downtown Scenario: The results from our downtown scenario simulations are shown in Table II. For plots of the percent of vehicles with gradients over the entire duration of the simulations, see Fig. 11 in Appendix B. As mentioned earlier, the downtown scenario differs from the highway scenario in that vehicles drive past more vehicles, but are not next to them for as long. Based on our results, the fact that the vehicles were passing other vehicles more often helped the gradients spread more. This makes sense because the initial vehicle was able to pass the gradients to more vehicles at the beginning of the simulation, thus allowing the gradients to reach vehicles that may have otherwise left the simulation too early.

It is worth noting that the one experiment where the downtown scenario performed (significantly) worse than the highway scenario was for the 2 second STT simulation with 10% packet drop. This is likely because of the fact that most vehicles are not near one another for long enough to transmit gradients in time if too many packets are dropped. This limits the amount of successful exchanges between vehicles and thus results in the very low percentage of 6.94%. Again, this proves our argument that gradients must be sparsified as much as possible.

B. Learning

Plots of the holdout loss for each epoch are in Appendix C (Fig. 12). The results for the final training epoch can be seen in Table III. To verify that our results were significant, we ran a Wilcoxon non-parametric hypothesis test [44], as seen in Table IV.

C. Security & Privacy

Overall, attacks performed in the black-box setting of DJGRAD are less effective than in the white-box

Car	**Hours**	**None**	**Fed**	**DJG(1.0)**	**DJG(0.9)**
1	[0,5]	1.25	1.70	**1.05**	1.53
	[6,11]	117.04	**3.48**	101.37	109.88
	[12,17]	176.32	**5.34**	155.14	164.28
	[18,23]	132.65	**3.82**	113.76	126.27
2	[0,5]	12.80	**1.70**	7.96	11.67
	[6,11]	18.88	**3.48**	11.39	8.93
	[12,17]	78.83	**5.34**	54.42	41.66
	[18,23]	69.10	**3.82**	48.29	42.19
3	[0,5]	36.99	**1.70**	34.97	23.00
	[6,11]	12.05	**3.48**	18.22	12.42
	[12,17]	7.14	**5.34**	22.77	12.65
	[18,23]	13.53	**3.82**	20.09	12.34
4	[0,5]	37.15	**1.70**	42.49	50.35
	[6,11]	121.52	**3.48**	134.95	180.98
	[12,17]	54.04	**5.34**	55.33	88.36
	[18,23]	14.62	**3.82**	11.23	14.45

TABLE III: Simulation results from training models using no sharing ("None"), FL ("Fed"), and our proposed method with $p = k$ ("DJG(k)"). Each method was run for 4,000 epochs. First and second lowest loss are bolded and underlined, respectively.

Baseline	Alternative	P-Value			
	Fed	0.125	**0.0625**	**0.0625**	**0.0625**
None	DJG(1.0)	**0.0625**	**0.0625**	0.9375	0.875
	DJG(0.9)	0.125	**0.0625**	0.4375	0.9375

TABLE IV: P-values for each car (1-4) given a baseline method and an alternative. H_1 is that the alternative method achieves a lower loss than the baseline. H_0 is that they are the same.

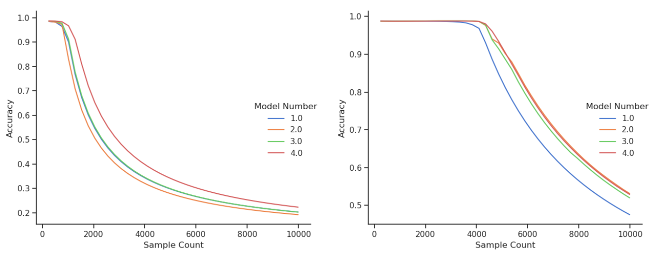
setting. We find that in a gradient sharing protocol, carefully crafted gradients can still harm performance, inject backdoors, and steal data, albeit with less efficacy. In each scenario, "Model 1" is the ego or adversarial vehicle.

1) Harm Performance: The formulated attack is effective in both the white-box and black-box settings, although a larger number of malicious gradients is required to reduce performance in the black-box setting (Fig. 6). Note that the number of samples required to degrade performance below 50% accuracy is on the order of thousands in both settings.

2) Inject Backdoor: Using the gradient poisoning technique to inject backdoors into models is shown to be quite effective in both settings, achieving greater than 30% backdoor trigger efficacy with fewer than 15 samples (Fig. 7). However, the attack in the black-box setting limits this threat to a maximum of around a 30% backdoor efficacy.

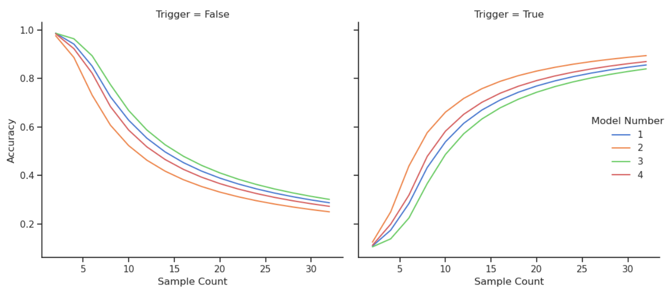
3) Steal Data: The major limitation of the gradient reconstruction attack lies in its ability to steal data in the black-box setting. While the attack is able to reconstruct nearly 100% of the original data in the white-box setting, it is only able to reconstruct data in the black-box setting if the "victim and surrogate models" have similar weights. To quantify the similarity, weights are randomly instantiated $(-0.5, 0.5)$ and copied. The weights of the surrogate model are then randomly altered by a small $\epsilon < 0.015$ value. We find that weights differing by $\epsilon \leq 0.0115$ can recover the majority of data, whereas weights differing by $\epsilon > 0.0115$ can only recover few samples (Fig. 8).

4) Defenses: While the attacks on DJGRAD were shown to be somewhat effective with limitations, the robustness of DJGRAD could be further improved by adding simple defense mechanisms against these attacks. Anomaly detection systems could be applied to detect harmful or poisoned gradients. Alternatively, vehicles could create copies of their models before applying gradients. If the application of any received gradients proves

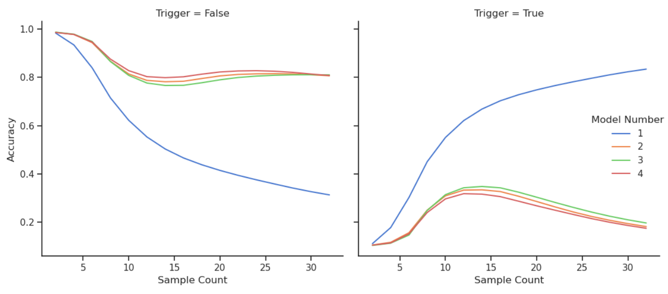


(a) White-box (b) Black-box

Fig. 6: Attacks in the distributed learning setting which harm performance of other vehicles' models.



(a) White-box



(b) Black-box

Fig. 7: Attacks in the distributed learning setting which inject backdoor triggers in other vehicles' models. Left: Natural accuracy, Right: Backdoor trigger efficacy.

harmful, the vehicle can easily revert to the backup model state. These defenses would mitigate the threat of an adversary wishing to do harm, though it would not provide guarantees against an adversary reconstructing or stealing data from gradients.

VI. Discussion

A. Limitations

In our experiments involving ML models, we did not use realistic autonomous vehicle datasets or ML architectures. Our simulation of sparse gradient learning did not have many vehicles, and was too slow to simulate more than a few hyperparameter values. Additionally, we

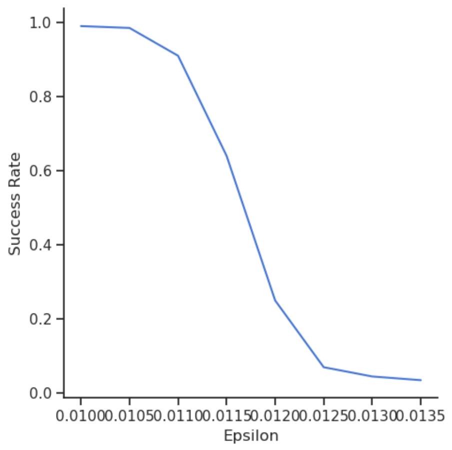


Fig. 8: Attacks in the distributed learning setting which reconstruct data using other vehicles' gradients. The efficacy decreases with larger ϵ.

would have liked to evaluate the ML model performance and security using a more fine-grained analysis from our vehicle network simulations.

B. Future Work

1) Networking: To further analyze the variables that impact the efficient spread of gradients throughout the vehicular network, other parameters in the simulations could be modified. Such parameters include vehicle speed, traffic density, and WSA periodicity.

2) Learning Simulations: In future works, we will utilize realistic autonomous vehicle datasets/simulators such as [45] and larger DNN architectures for Deep Q-Learning such as [46]. We will extend our ML simulations to include tens of cars, rather than four, and evaluate with more dropping parameters. However, before working on any additional implementations, we intend to develop a robust theoretical framework to analyze our problem statement and provide mathematically-verifiable guarantees in place of our current heuristics.

3) Security & Privacy: More work can be done exploring attacks and defenses in the setting of our protocol. For example, our backdoor injection could leverage more recent advances in data poisoning attacks such as the attacks from Turner et al. [41].

VII. Conclusion

We propose and implement a distributed, asynchronous, consensus-free, real-time protocol for training ML models on consumer CAVs. Through simplified simulations, we show the following:

- Gradients can efficiently spread throughout vehicular networks if they are sparsified enough to only require an STT of 1 second, even with a packet drop rate of 10%.
- DJGrad outperforms individually-trained NNs, as verified by a statistical hypothesis test, and is robust to dropped/lost gradients.
- DJGrad is robust against attacks in the black-box setting, and defenses can be created against the 3 formulated attacks.

References

[1] Z. Yang, Y. Chai, D. Anguelov, Y. Zhou, P. Sun, D. Erhan, S. Rafferty, and H. Kretzschmar, *Surfelgan: Synthesizing realistic sensor data for autonomous driving*, 2020. arXiv: 2005.03844 [cs.CV].

[2] J. Plungis, "Who Owns the Data Your Car Collects?" *Consum. Rep.*, May 2018. [Online]. Available: https://www.consumerreports.org/automotive-technology/who-owns-the-data-your-car-collects.

[3] F. Pourpanah, M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, and X.-z. Wang, "A review of generalized zero-shot learning methods," *ArXiv*, vol. abs/2011.08641, 2020.

[4] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, *Liquid time-constant networks*, 2020. arXiv: 2006.04439 [cs.LG].

[5] *BMW Group launches BMW CarData: new and innovative services for customers, safely and transparently*, [Online; accessed 9. Dec. 2021], Dec. 2021. [Online]. Available: https://www.press.bmwgroup.com/global/article/detail/T0271366EN/bmw-group-launches-bmw-cardata:-new-and-innovative-services-for-customers-safely-and-transparently?language=en.

[6] *Federated learning: Privacy and incentive.*

[7] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Advances in Neural Information Processing Systems*, 2019.

[8] Z. Xu, X. Li, X. Zhao, M. H. Zhang, and Z. Wang, "Dsrc versus 4g-lte for connected vehicle applications: A study on field experiments of vehicular communication performance," *Journal of advanced transportation*, vol. 2017, 2017.

[9] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *CoRR*, vol. abs/1912.04977, 2019. arXiv: 1912.04977. [Online]. Available: http://arxiv.org/abs/1912.04977.

[10] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Access*, vol. 8, pp. 140 699–140 725, 2020.

[11] S. K. Lo, Q. Lu, C. Wang, H.-Y. Paik, and L. Zhu, "A systematic literature review on federated machine learning: From a software engineering perspective," *ACM Comput. Surv.*, vol. 54, no. 5, May 2021, ISSN: 0360-0300. DOI: 10.1145/3450288. [Online]. Available: https://doi-org.proxy.lib.umich.edu/10.1145/3450288.

[12] F. Wu, "Plfg: A privacy attack method based on gradients for federated learning," in *Security and Privacy in Digital Economy*, S. Yu, P. Mueller, and J. Qian, Eds., Singapore: Springer Singapore, 2020, pp. 191–204, ISBN: 978-981-15-9129-7.

[13] H. A. Lianto, Y. Zhao, and J. Zhao, *Attacks to federated learning: Responsive web user interface to recover training data from user gradients*, 2020. arXiv: 2006.04695 [cs.CR].

[14] A. M. Elbir, B. Soner, and S. Coleri, "Federated learning in vehicular networks," *arXiv preprint arXiv:2006.01412*, 2020.

[15] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, "Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent," *arXiv preprint arXiv:1803.05880*, 2018.

[16] C. Xu, Y. Qu, Y. Xiang, and L. Gao, *Asynchronous federated learning on heterogeneous devices: A survey*, 2021. arXiv: 2109.04269 [cs.DC].

[17] X. Xian, X. Wang, J. Ding, and R. Ghanadan, "Assisted learning: A framework for multi-organization learning," *arXiv preprint arXiv:2004.00566*, 2020.

[18] *A survey of deep learning techniques for autonomous driving*. DOI: 10.1002/rob.21918.

[19] J. Sun, T. Chen, G. Giannakis, and Z. Yang, "Communication-efficient distributed learning via lazily aggregated quantized gradients," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/4e87337f366f72daa424dae11df0538c-Paper.pdf.

[20] S.-P. Wang, P. Liu, and J.-J. Wu, "Communication usage optimization of gradient sparsification with aggregation in deep learning," in *Proceedings of the 2018 VII International Conference on Network, Communication and Computing*, ser. ICNCC 2018, Taipei City, Taiwan: Association for Computing Machinery, 2018, pp. 22–26, ISBN: 9781450365536. DOI: 10.1145/3301326.3301347. [Online]. Available: https://doi-org.proxy.lib.umich.edu/10.1145/3301326.3301347.

[21] A. A. Awan, J. Bedorf, C.-H. Chu, H. Subramoni, and D. K. Panda, "Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation," *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2019. DOI: 10.1109/ccgrid.2019.00064. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2019.00064.

[22] F. Bai and H. Krishnan, "Reliability analysis of dsrc wireless communication for vehicle safety applications," in *2006 IEEE intelligent transportation systems conference*, IEEE, 2006, pp. 355–362.

[23] Y. L. Morgan, "Notes on dsrc & wave standards suite: Its architecture, design, and characteristics," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 4, pp. 504–518, 2010.

[24] C. Campolo and A. Molinaro, "Multichannel communications in vehicular ad hoc networks: A survey," *IEEE Communications Magazine*, vol. 51, no. 5, pp. 158–169, 2013.

[25] R. A. Uzcátegui, A. J. De Sucre, and G. Acosta-Marum, "Wave: A tutorial," *IEEE Communications magazine*, vol. 47, no. 5, pp. 126–133, 2009.

[26] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–25, Jul. 2021, ISSN: 1557-7341. DOI: 10.1145/3459991. [Online]. Available: http://dx.doi.org/10.1145/3459991.

[27] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, no. 1, pp. 3–15, Jan. 2011. DOI: 10.1109/TMC.2010.133.

[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow. org (2015)," *URL https://www. tensorflow. org*, 2015.

[29] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. 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)*, IEEE, 2018, pp. 2575–2582.

[30] A. Varga, "Omnet++," in *Modeling and tools for network simulation*, Springer, 2010, pp. 35–59.

[31] OpenStreetMap contributors, *Planet dump retrieved from https://planet.osm.org*, https://www.openstreetmap.org, 2017.

[32] C.-s. Oh and J.-m. Yoon, "Hardware acceleration technology for deep-learning in autonomous vehicles," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2019, pp. 1–3. DOI: 10.1109/BIGCOMP.2019.8679433.

[33] H. Fanaee-T and J. Gama, "Event labeling combining ensemble detectors and background knowledge," *Progress in Artificial Intelligence*, pp. 1–15, 2013, ISSN: 2192-6352. DOI: 10.1007/s13748-013-0040-3. [Online]. Available: [Web%20Link].

[34] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[35] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[36] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[38] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[39] H. Xiao, H. Xiao, and C. Eckert, "Adversarial label flips attack on support vector machines," in *ECAI 2012*, IOS Press, 2012, pp. 870–875.

[40] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[41] A. Turner, D. Tsipras, and A. Madry, "Clean-label backdoor attacks," 2018.

[42] G. Sun, Y. Cong, J. Dong, Q. Wang, L. Lyu, and J. Liu, "Data poisoning attacks on federated machine learning," *IEEE Internet of Things Journal*, 2021.

[43] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*, Springer, 2020, pp. 480–501.

[44] F. Wilcoxon, *Individual comparisons by ranking methods. biom. bull., 1, 80–83*, 1945.

[45] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[46] M. Vitelli and A. Nayebi, "Carma : A deep reinforcement learning approach to autonomous driving," 2016.

APPENDIX A
FLOW CHART

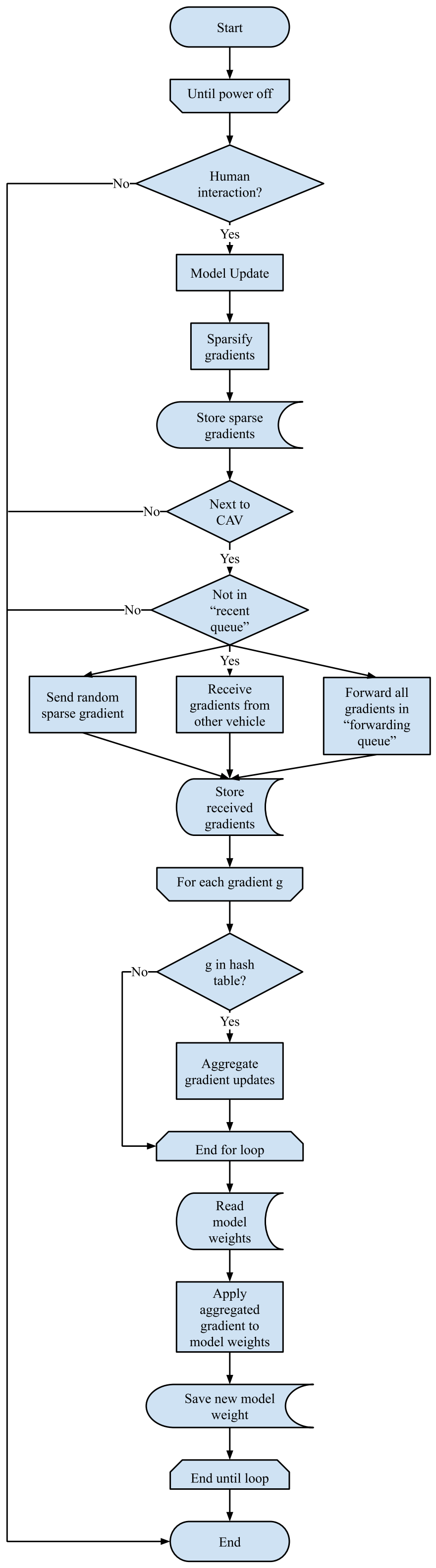


Fig. 9: Flowchart of our proposed DJGRAD protocol.

Appendix B
Network Simulation Plots

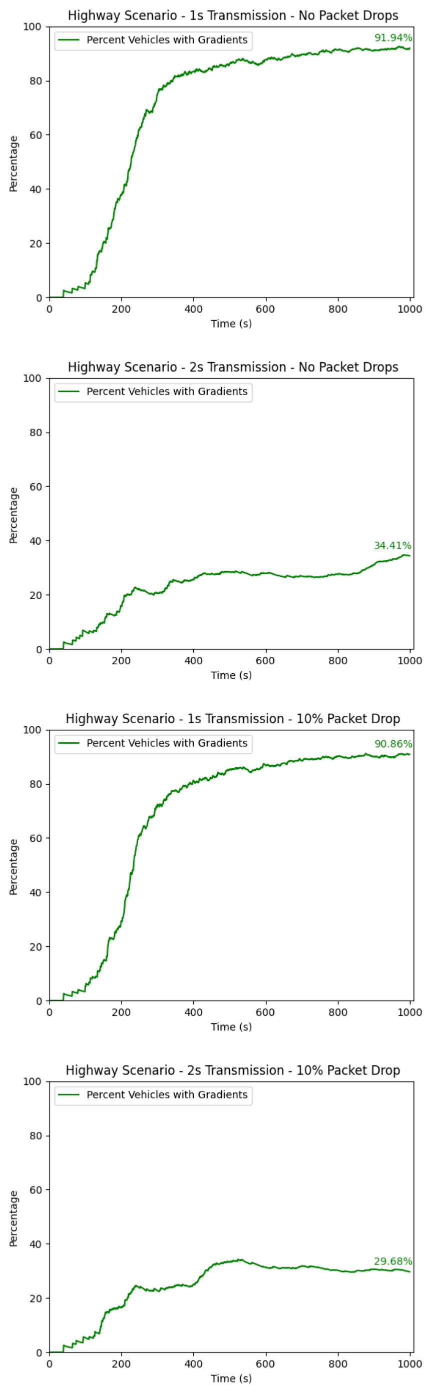


Fig. 10: Plots of our four highway experiments over the duration of 1,000 seconds. Graphs show percentage of vehicles in simulation with gradients out of the number of vehicles that have arrived in the scene.

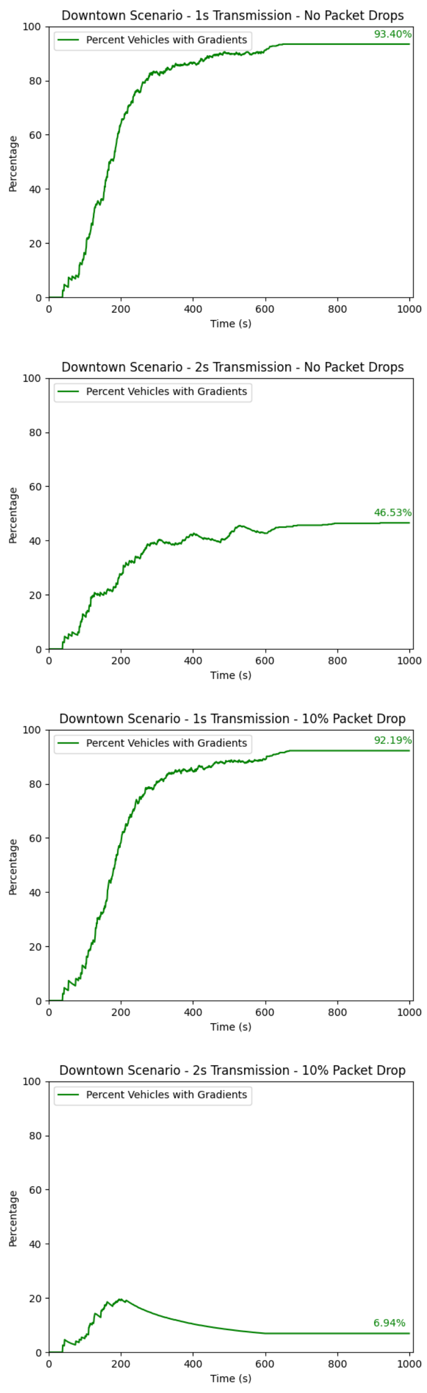


Fig. 11: Plots of our four downtown experiments over the duration of 1,000 seconds. Graphs show percentage of vehicles in simulation with gradients out of the number of vehicles that have arrived in the scene.

Appendix C
Loss Plots

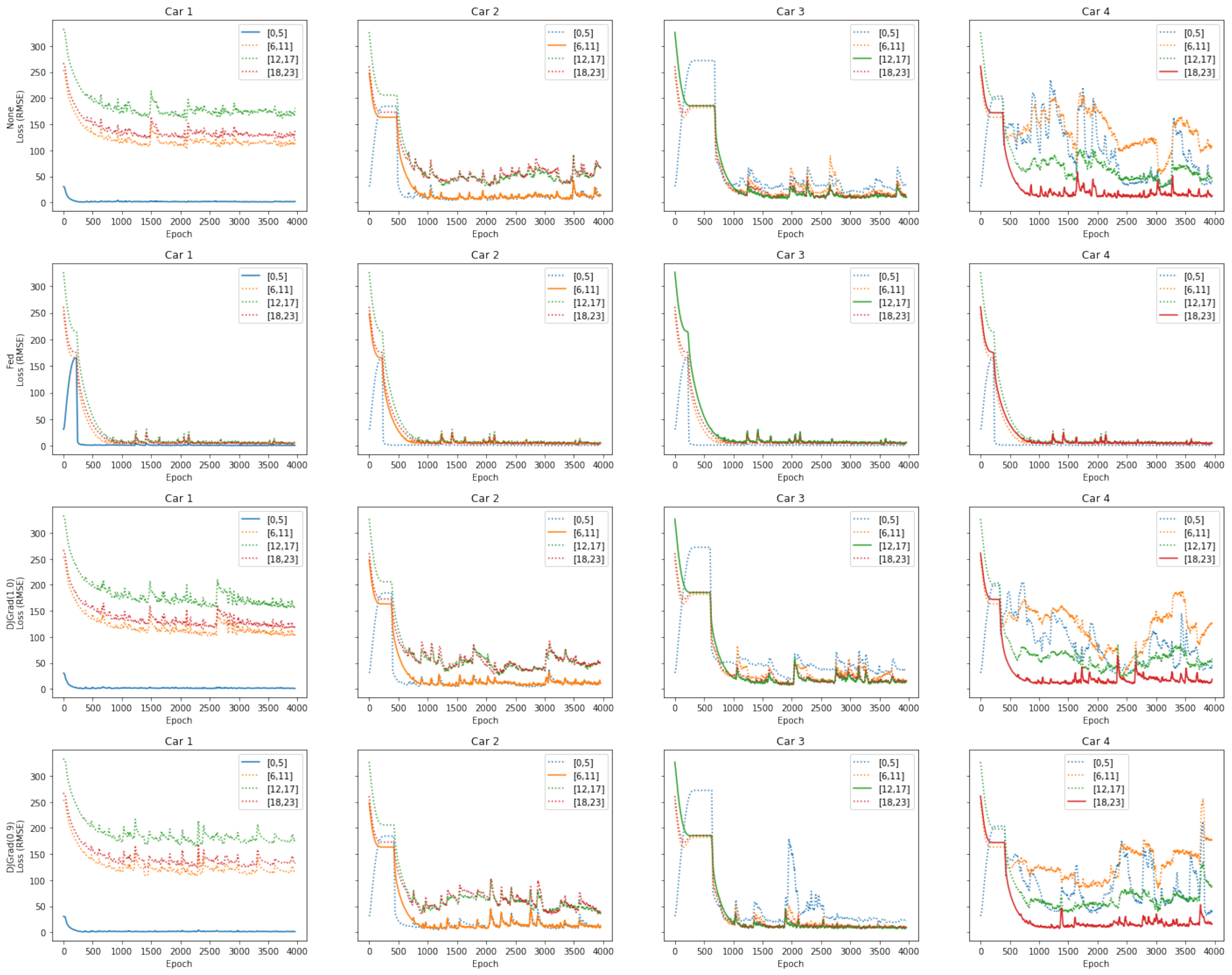


Fig. 12: Loss plots (RMSE) of the model for each car over 4,000 epochs. The training class for each car is represented as a solid line, and the other classes as dotted lines.