RAD: Rapid Automobile Data Analytics Framework for Structured Data

Nikhil Muralidhar[1,2]*, Brian Mayer[1,2]*, Nathan Self[1,2]*, Panduranga Kondoju[3], Basavaraj Tonshal[3], John Schmotzer[3], Naren Ramakrishnan[1,2]
[1]Discovery Analytics Center, Virginia Tech, Arlington, Virginia
[2]Dept. of Computer Science, Virginia Tech, Blacksburg, Virginia
[3]Ford Motor Company, Dearborn, Michigan

ABSTRACT

Machine learning models are useful for vehicle performance optimization and characterization. They can be used to forecast future events or conditions, classify events as cautious or concerning, and act as a prognostic tool. Currently, supporting these machine learning models requires analytical computations and analysis of raw data collected from vehicles in the cloud. This incurs a large cost associated with transferring large amounts of data to (and from) the cloud to train and run models. Alternatively, models could be executed on board the vehicle. The compromise is that there are limited resources available on an automotive electronic control unit (ECU) and the architecture is currently decentralized to perform these computations. Therefore, any deployed models would need a model execution environment that uses limited computing resources. In this scenario it is important to also consider the trade-off between resources and performance. In this paper, we develop a unified framework enabling rapid deployment of flexible machine learning models to handle a variety of use-cases in constrained environments called the Rapid Automobile Data Analytics (RAD) framework. This paper focuses primarily on creating models and architectures for sequential and structured data. Multiple architectures and models are investigated and evaluated, and an automated pipeline for deployment of the models is developed.

1 INTRODUCTION

Automobiles in contemporary times are increasingly being equipped with various sensors and computation to automate and fine-tune safety, and security features like airbag deployment, anti-lock brakes, traction control, and many others. Many of these features like blind-spot detection serve the explicit purpose of alerting the driver to various potential hazards around the vehicle. Such sensors are geared towards making the driver aware of the current state of the vehicle during a drive, thereby increasing driver safety. With the increase in the magnitude and the heterogeneity of the data being generated in the vehicle, it is important to leverage the insights offered by this data about vehicle state, useful for many purposes like optimizing vehicle processes like fuel consumption or providing explicit driver alerts to ensure safety.

The above trends have led to machine learning making significant inroads in automobile analytics. In [12], for instance, the authors use machine learning to study the effect of displaying a particular expected state of a traffic light on driver behavior and use the insights to offer safer recommendations of expected traffic light state. In [21], the authors employ machine learning to recognize the driver based on their driving style, thereby enabling the vehicle to adapt the driver assistance system to the driving style of the specific driver to improve safety and driver comfort.

One particular area that machine learning can play a role in is in addressing driver distraction. The National Highway Traffic Safety Administration (NHTSA) reported that one in ten fatal crashes and two in ten injury crashes were caused by driver distractions in the United States during 2014. The NHTSA also reported that 2841 lives were lost to distracted driving in 2018 alone.

In this work, we leverage machine learning techniques to develop a comprehensive framework, called the Rapid Automobile Data Analytics (RAD) framework, for distracted driver detection using only the Controller Area Network (CAN) data signals produced in the vehicle. A salient feature of our framework is the ability of the models therein to be amenable for deployment in a cloud-centric and a distributed (on-device) context allowing for flexibility and adaptability to various settings. We evaluate our learning models using a set of real-world driver data and report results. We also test model performance on distracted driver detection of learning models deployed on an edge device and report results. In addition to modeling performance characterization, we also report the on-device memory, disk space and the prediction latency results for models in our framework. Finally, we evaluate the memory and disk footprint requirements, limitations of the current pipeline and other lessons from deploying and running a distributed machine learning framework for distracted driver detection using automobile CAN data.

2 RELATED WORK

In recent times machine learning has become ubiquitous and is in extensive use in automobiles for various control and monitoring purposes and to augment the driving experience. In [20], Manasseh et al. propose a machine learning algorithm based on decision trees with pruning for driving destination prediction. Such destination prediction applications have various uses in increasing traffic safety and mobility. Simmons et al. [24] adopt a probabilistic approach and apply a Hidden Markov Model to the problem of route and destination prediction achieving state-of-the-art results.

Driving style or driver profile identification is another problem imperative to optimal vehicular control system adaptation, and has also been addressed using machine learning. Hallac et al. [14] utilize driving data collected by Audi AG and Audi Electronics Venture vehicles, on real roads, to classify the different drivers corresponding to each drive. They treat the problem as a time series classification task and found that turning styles especially help in discriminating between drivers. Wang et al. [26] use a random

*Equal Contribution

forest model to identify the driver using vehicle telemetrics data and characterize the importance of different features in their helpfulness in the driver identification task. A detailed survey of driver behavior detection has been conducted by Chhabra et al. [9]. There have also been efforts to create general purpose embeddings of vehicular state for use in downstream learning tasks like the work of Hallac et al. [13]. Jachimczyk et al. [17] propose a framework for driving style estimation and estimate three facets of driving style: safety, economy and comfort.

Many efforts have also focused on anomaly and event detection in the context of automobiles. Cheng et al. [8] developed a learning framework using recurrent neural networks for anomalous trip detection from taxicab trip trajectory data. Makke et al. [19] develop a hybrid prognostic approach based on physics enabled data aggregation and cloud-based data driven prognostics. They apply the framework to the task of estimating brake pad wear and cabin air filter prognostics. Coellingh et al. [10] propose a pedestrian detection and collision warning system. Schlechtriemen et al. [22] propose a state-of-the-art generative model for lane change intent detection. Taylor et al. [25] propose an LSTM architecture for detecting CAN bus attacks by modeling attack detection as part of a CAN bus instruction forecasting problem.

There have also been research efforts in the Internet of Things (IoT) and connected vehicle space, Han et al. [15] propose a one-way ANOVA test based statistical model to detect whether a connected vehicle is in an abnormal state. TinyML is yet another related area of machine learning concerned with bringing ML inference to ultra-low power devices. Banbury et al. [2] highlight some of the major lines of research in this yet nascent field related to IoT. One of the main considerations in TinyML and other IoT settings is the development models with light memory and disk footprints, Kumar et al. [18] develop a novel tree based lightweight learning algorithm called Bonsai for efficient prediction on resource constrained IoT devices. Another way to ensure lightweight models is to focus on deploying shallow (student) neural network models while ensuring that they have expressive power close to a deeper (teacher) network jointly trained on a particular learning task such that the student network learns to "mimic" the decisions made by the teacher. This technique called mimic learning or knowledge distillation has been used extensively by Ba et al. [1] and Hinton et al. [16] to develop shallower models with a faster prediction time and lower footprint but with very similar (or in some cases the same) model accuracy as corresponding deep models jointly trained for a learning task. In [27], Yadawadkar et al. develop a learning methodology for characterizing the importance of various naturalistic driving features in detecting distracted, drowsy, and attentive driving behaviors.

In line with the recent efforts to develop automated machine learning pipelines in the context of automobiles, we propose a framework for rapid deployment of machine learning models for on-board analytics and demonstrate its effectiveness on the application of distracted driver detection in automobiles.

3 RAPID AUTOMOBILE DATA ANALYTICS

3.1 Background

We define three different levels of vehicle network architecture as follows:

- *Device Computing*: Any processing that occurs at the data generating process, defined in an automotive context as an Electronic Control Unit (ECU) device.
- *Edge Computing*: Traditionally, the device is physically and logically separated from the *edge*. The edge refers to computing resources located in the LAN of the data generating process. In our case it is a modem connected to and receiving CAN data through an OBD-II connection.
- *Cloud Computing*: A suite of compute infrastructure physically separate from the vehicle context.

3.2 Architecture

We initially defined three different architecture styles similar to those defined by Simmhan [23].

(1) *Cloud Centric Architecture*: Data must be sent to the cloud in real-time or near real-time to train, run, and retrain the model. Any delays in communication will result in delays in analysis.
(2) *Edge Centric Architecture*: Data is sent to the cloud in batches to train and retrain the model as necessary. The model is deployed to the edge to analyze streaming data onboard the vehicle. See Figure 1.
(3) *Distributed Architecture*: Data is sent to the cloud once to train an initial model. The model is deployed to the edge to analyze streaming data onboard the vehicle. The model can be retrained onboard the vehicle. See Figure 2.

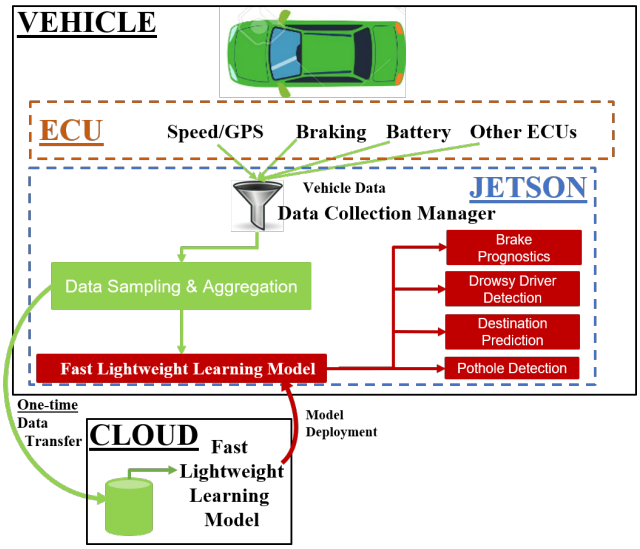

Figure 1: Edge Centric Architecture: Data is collected onboard the vehicle and sent to the cloud as needed where a fast-lightweight learning model is trained. The model is then deployed to the vehicle to address the use cases it was designed for. Additional data is sent to the cloud periodically, for model retraining. The retrained model is then deployed to the vehicle.

Non-cloud-based architectures have the distinct advantage of being able to make decisions in the edge, thus saving on latency and

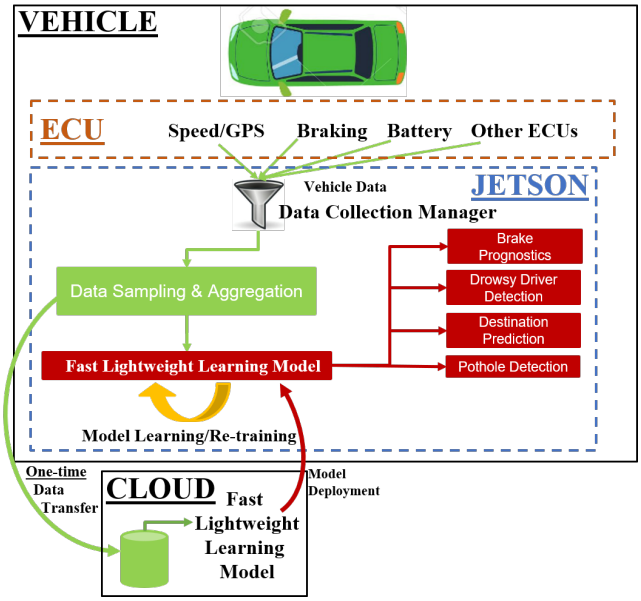

Figure 2: Distributed Architecture: Similar to the edge centric model, data is collected onboard the vehicle but is only sent to the cloud once where the fast-lightweight learning model is trained. The model is then deployed to the vehicle to address the use cases it was designed for. In the distributed architecture the model can be retrained on board the vehicle using data available on the vehicle.

other data transfer costs that may be incurred while transferring data onto the cloud. Further, in the case of distributed architectures, as more data is accumulated on the edge, the model running on the edge, can be re-trained on the edge device itself, hence effectively decoupling it from the cloud if the need arises.

The model re-train interval can be specified by the user for all architectures. In each of the three architectures, the system assumes an independent validation phase where the trained model is evaluated for performance on the learning task using a hold-out test set. In the case of the distributed architecture, an updated hold-out test set is expected to be present on the edge device for effective validation of a newly re-trained model.

The focus of this project was on developing edge centric and distributed architectures as they are more efficient and require significantly less data transfer costs.

3.3 Data

3.3.1 Data Source. Data was collected exclusively from the Strategic Highway Research Program 2 (SHRP 2) Naturalistic Driving Study (NDS) [4] database from the Virginia Tech Transportation Institute (VTTI). As the largest naturalistic driving dataset available worldwide, the SHRP2 NDS database offers detailed and accurate pre-crash information not available from other crash databases. This pre-crash information serves as strong and powerful evidence identifying the progression of critical driving behaviors, in addition to, traffic and vehicle dynamics. These were either captured by an installed on-board Data Acquisition System (DAS) or manually processed post-hoc by viewing video. The DAS includes forward radar; four video cameras, including one forward-facing, color, wide-angle view; accelerometers; vehicle network information; Geographic Positioning System; on-board computer vision lane tracking, plus other computer vision algorithms; and data storage capability [4].

Data was initially collected on the vehicle and then downloaded periodically by research staff to a central database (Figure 3). Multiple researchers work constantly on data quality and control. Unique "triggers," i.e., anomalies in the time-series data, were used to identify and extract all the crash and near-crash (C/NC) events from the database. Additionally, a separate set of baseline events were randomly selected for comparison. These events were then reviewed, coded, and evaluated by data reductionists. The coded information enables researchers to easily identify specific events of interest (EOIs).

We initially started with the three event classes listed below along with their definition using the coded event data in the SHRP2 NDS database.

(1) A **drowsy event** is an event from the C/NC or baseline dataset where the driver exhibits obvious signs of being asleep or tired, or is actually asleep while driving, degrading performance of the driving task.
(2) A **distracted event** is an event from the C/NC dataset where the driver is not maintaining acceptable attention on the driving task due to engagement in one or more secondary tasks. This is a subjective judgment call by the reductionist indicating whether any secondary tasks the driver might be involved in contributed to the C/NC.
(3) An **attentive event** is an event from the baseline dataset where the driver is not engaged in any secondary task. A secondary task is defined as an observable driver engagement not critical to the driving task such as non-driving related glances away from the direction of vehicle movement.

EOIs under each class were pulled from the SHRP2 NDS database for the purposes of this research effort. Unfortunately the data set we received was an unbalanced set of 3,669 total events (570 drowsy events, 915 distracted events, and 2,184 attentive events).

Once the EOIs were identified, time-series data of the corresponding complete trips were retrieved from the SHRP2 NDS database (Figure 3). Epochs were created by extracting 60 seconds of driving data starting from 65 seconds before the event time to 5 seconds before the event time. We assumed that the driver behavior did not change throughout this 1-minute epoch. The event data consisted of 44 variables. Among these 44 variables, 28 of them were raw data directly collected by the DAS in the SHRP2 NDS vehicle, mainly vehicle dynamics (Table 1). The rest of the variables were calculated based off the raw variables (Table 2).

3.3.2 Data Pre-processing. All the sensors during data collection were synchronized with respect to the trip clock, which is the first variable listed in Table 1, and down-sampled to 1 Hz. The timing of the data across variables was asynchronous leading to missing variables at each collection time point. These missing values for each variable were replaced with the last known corresponding value. Since the rate of these sensor values was on the order of

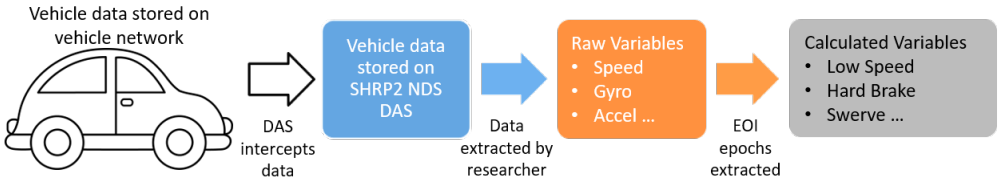

Figure 3: Data collection, extraction, and pre-processing

Table 1: List of raw variables

Variable Name	Unit	Hz	Note
Timestamp	ms		Time since the beginning of the trip
Speed	km/h	1	Vehicle speed
Gyro z	deg/s	10	Lateral angular velocity
Accel y	g	10	Lateral acceleration
Distance to left lane marker	cm	30	Positive when on the left side of the marker and negative when on the right side
Distance to right lane marker	cm	30	Positive when on the left side of the marker and negative when on the right side
Probability of left marker exist		30	Probability a painted marker exists on the left side of the vehicle's lane
Probability of right marker exist		30	Probability a painted marker exists on the right side of the vehicle's lane
Time of Day	ms	1	UTC time of day
Day		1	From 1 to 31
Month		1	From 1 to 12
Year		1	Last two digits of year
Longitudinal Distance Target 1–8	m	15	Longitudinal distance to radar target 1–8
Lateral Distance Target 1–8	m	15	Lateral distance to radar target 1–8

10-30hz, the above approximation was considered accurate and reasonable.

Initial assessments of data for the three events indicated that it might be difficult to distinguish to a high degree of accuracy between the drowsy and attentive events (See Figure 4). We notice from the figure that the average *drowsy* drive is very similar to the average *attentive* drive in a majority of the features considered. Hence, in this study we limit ourselves to distinguishing between *distracted* and *attentive* drives and address potential augmentations in the future to incorporate *drowsy* driving detectors in section 5.2. Additionally, of the various raw and calculated variables, we used only a subset (8 in all) of variables which were the least sparse in nature and encoded rich discriminative power between the two classes. The variables used have been depicted in Figure 4.

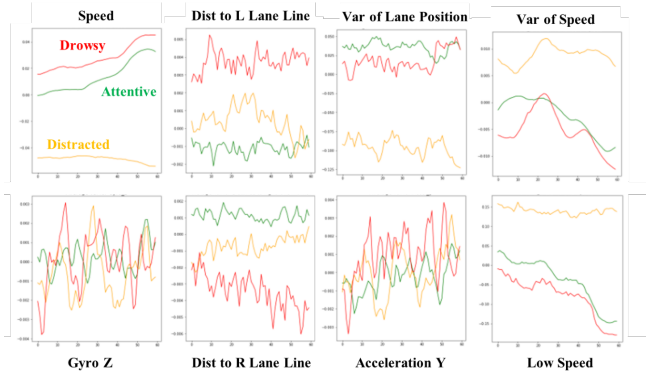

Figure 4: The average values throughout a drive for many CAN feature variables. For several variables the values for drowsy EOIs are fairly similar to attentive EOIs making the classification problem fairly non-trivial. This figure also depicts all CAN variables used as features for classification.

3.4 Learning Models, Results & Discussion

In our framework, we provide support for four different types of classification models:

(1) *Multilayer Perceptron* (MLP): A standard feed-forward neural network trained by gradient descent. The model architecture is dynamically set by the user. We used 4 hidden layers in our model variant.

(2) *Random Forest* (RF): An ensemble of decision trees introduced by Breiman et al. [3] which is one of the state-of-the-art

Table 2: List of calculated variables

Variable Name	Unit	Hz	Note
Timestamp	ms		Time since the beginning of the trip
Variance of Speed	km/h	10	Variance of speed of last 30 seconds
Variance of Lane Position		30	Variance of left lane distance of last 30 seconds
Variance of Throttle Position		80	Variance of throttle position of last 30 seconds
Low Speed		1	Indicates the speed is below 30kph: binary
Hard Brake		10	1- 30 seconds after heavy deceleration (0.4g): binary
Day of Week			0 is Sunday, 1 is Monday, etc.
Swerve		10	Indicates if within a 30-second window after a swerve: binary
Passing		15	Passing a vehicle in adjacent lanes: binary
Being Passed		15	Being passed in adjacent lanes: binary
Traffic Flow		15	Indicates if a vehicle is passed more than it is passing: binary
Traffic Level		15	Number of vehicles on radar
Tire Out of Lane		30	Indicates if the vehicle's tire is outside the lane: binary
Lane Change		30	Indicates if within a 30-second window after a lane change: binary
Lane Bust		30	Indicates if within a 30-second window after a lane bust: binary
Time to Line Crossing	sec	30	The time to cross a lane line under current status using lane distance and lateral speed: positive: approaching left line and negative: approaching right line
Active Steering		10	The entire course of steering where peak value exceeds a threshold

models for classification. We used 100 decision trees in our random forest classifier.

(3) *Gradient Boosting* (GB): Gradient boosting models are also ensemble models which combine several "weak learners" to form a single strong learner, as studied in Friedman et al. [11]. We used 100 estimators in our gradient boosting classifier presented in the performance evaluation.

(4) *XGBoost* (XGB): A more sophisticated variant of gradient boosting architectures introduced in Chen et al. [7] with improvements like clever penalization of trees and proportional shrinking of leaf nodes.

These models are fed the 60-second EOIs and are trained to classify the time series into a certain category (*attentive* or *distracted*). As previously mentioned the data set was unbalanced by more than an order of magnitude. Owing to the small amount of data, we felt it was important to preserve as many EOIs as possible and attempt to utilize the unbalanced set (2,184 attentive and 915 distracted). The results for each of the models trained on the unbalanced data set are shown in table 3. The XGBoost model performs the best (77% micro average accuracy) and is also highly precise. We believe incremental training with more quality labeled data will increase the recall of the XGBoost model (and other models in our framework).

The imbalanced nature of the data resulted in all the models' poor ability to identify all distracted events (i.e., low recall). If the model was unsure, it was more effective for the model to classify it as attentive because it had a higher chance of being right. We tested an artificial balancing method, synthetic minority oversampling technique (SMOTE) [6], on the data. This improved the recall of distracted events slightly (28% to 36%) but accuracy decreased slightly as well (77% to 74%). We also tested each model with a balanced training dataset (i.e., 915 attentive and 915 distracted drives) with the hold-out evaluation set consisting of 53 attentive and 25 distracted drives (this holdout set is used for all experiments throughout the paper). Table 4 details results for the balanced data case.

We notice a degradation in model performance across all the models in terms of overall accuracy in the balanced data experiments relative to the original imbalanced data experiments showcased in

Table 3: Comparative performance of the tested models on the imbalanced dataset (2,184 attentive and 915 distracted). We notice that the performance of the Random Forest (RF) and the XGBoost (XGB) models are superior to the other models, with the XGB model yielding a slightly higher overall accuracy than the RF ensemble model.

Model	Class	Precision	Recall	F1	Overall Accuracy
GB	Attentive	0.73	0.98	0.84	0.74
	Distracted	0.86	0.24	0.38	
MLP	Attentive	0.7	0.81	0.75	0.64
	Distracted	0.41	0.28	0.33	
RF	Attentive	0.76	0.94	0.84	0.76
	Distracted	0.75	0.36	0.49	
XGB	Attentive	0.75	1.0	0.85	**0.77**
	Distracted	1.0	0.28	0.44	

Table 4: Comparative Performance of the tested models on a balanced dataset (training set consists of 915 Attentive, 915 Distracted drives). The XGBoost (XGB) model outperforms other models in terms of overall accuracy.

Model	Class	Precision	Recall	F1	Overall Accuracy
GB	Attentive	0.74	0.66	0.7	0.62
	Distracted	0.42	0.52	0.46	
MLP	Attentive	0.76	0.66	0.71	0.63
	Distracted	0.44	0.56	0.49	
RF	Attentive	0.73	0.66	0.69	0.6
	Distracted	0.4	0.48	0.44	
XGB	Attentive	0.8	0.74	0.76	**0.69**
	Distracted	0.52	0.6	0.56	

Table 3. However, we notice that the recall of the *Distracted* class improves significantly when models are trained with the balanced dataset. We believe that training the models with a larger *Distracted* dataset will significantly improve the recall of the already highly precise XGB and RF models.

Additionally, a distributed architecture was created for the classification model. This enabled retraining of the model on board the vehicle. The distributed model retraining procedure is flexible as the user is able to control the dynamics of re-training and model update (i.e., the frequency of model re-training and model update can be controlled by the user). Both the retraining and model testing are run concurrently and asynchronously on the edge device. The evaluation results of the retrained (distributed) XGBoost model are depicted in table 5.

Table 5: Model performance comparison of XGBoost (XGB) classifier vs. the distributed version of the XGBoost (XGB-Dist.) classifier. We notice a performance degradation when the XGB model is re-trained on new data (online) on the device. This degradation may be alleviated as more data is incorporated on the device to re-train the distributed model.

Model	Class	Precision	Recall	F1	Overall Accuracy
XGB	Attentive	0.75	1.0	0.85	**0.77**
	Distracted	1.0	0.28	0.44	
XGB-Dist.	Attentive	0.7	0.97	0.81	0.7
	Distracted	0.75	0.18	0.29	

We notice a slight degradation in performance accuracy of the model but this should be alleviated as the model is re-trained on a greater volume of data on the edge device.

3.5 Vehicle Environment

3.5.1 Hardware. We used the NVIDIA Jetson TX2, an embedded computing board based on Tegra, a system on a chip, which integrates, among other things, an NVIDIA Pascal GPU and an ARM architecture CPU with 8GB of memory and 59.7GB/s of memory bandwidth. The Jetson TX2 Developer Kit was used to develop and test a hardware/software combination that simulates a potential vehicle environment. It supports NVIDIA JetPack, a software development kit that includes libraries for deep learning, computer vision, GPU computing, multimedia processing, among others.

Our current implementation of the simulated vehicle environment is designed to be as flexible as possible with regard to the vendors of components. For instance, any of the competitors to the Jetson boards with sufficient specifications could be used instead of the Jetson.

3.5.2 Architecture Implementation Software. We chose to use AWS Greengrass for deploying models to our Jetson devices. We explored two different options, Greengrass and Microsoft's Azure IoT Edge. At the time, Greengrass was slightly more mature but its main advantage is that it uses AWS's Lamdba service to encapsulate code to be run on devices. Lambda's serverless architecture allows us write model code without having to explicitly handle containers or deployment. Azure IoT Edge, on the other hand, requires inputs for containerization, provisioning, and deployment on a lower level. Greengrass handles the containerization and deployment to device automatically. These containers can be customized and configured but by default it is automated away from the end user. Greengrass's features also include "connectors" to collect data from interfaces on the device, "shadows" to track states of devices, and built-in security and cloud logging solutions. Azure IoT Edge has several analogous services but ultimately the ease of use of Greengrass and better documentation led us to use Greengrass.

Our developed learning pipeline is flexible and can be used with either of these services or their other competitors. AWS Greengrass was chosen as a test for deploying models and their resources to several devices, i.e., NVIDIA Jetsons, via "Greengrass groups."

3.5.3 Setup. To create a simulated test vehicle environment we provisioned two Jetson devices, one at Ford and one at Virginia Tech. Then we configured AWS Greengrass to deploy the XGBoost model defined above to each Jetson. On the devices, the model ingested data from files that were generated by real-life driving to simulate real inputs from a vehicle.

In addition to the operating system and software that comes pre-installed on the developer kit for Jetson devices, we installed *Greengrass Core*, which manages deploying models from the cloud and communicating with them, using an AWS setup script. This script installs Greengrass Core and creates an empty Greengrass group in the cloud that is associated with the device by installing the newly created group's security resources onto the device and configuring the device's Greengrass Core. After this, we installed Archiconda, a distribution of Python environment management software called Anaconda for ARM architectures. All python packages required by our machine learning models were installed using Archiconda. These include machine learning related packages like numpy, pandas, scipy, statsmodels and xgboost. At this point, we start the Greengrass daemon which waits for a deployment of the Greengrass group to be initiated in the cloud.

On the cloud, setting up the Greengrass group involves several steps.

(1) Create a zip file of the code for each model and upload them to AWS Lambda.

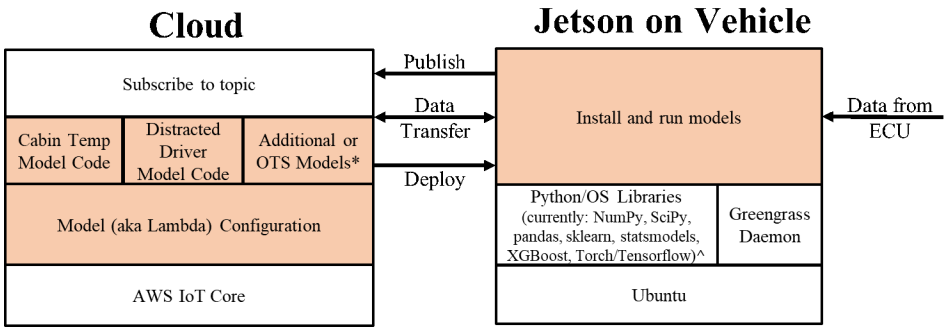

Figure 5: The Jetson/Greengrass ecosystem. Adding new models(*), off-the-shelf or custom, requires authoring a new configuration for the cloud and (∧) installing any new dependencies on the Jetson as necessary. The cloud and the Jetson can exchange data for training, retraining, and updates.

(2) Configure the Greengrass group to use each of those lambdas.
(3) Create a zip file of any configuration files and/or data that the models need to run. Upload these resources to S3 and configure them as machine learning resources in the Greengrass group associated with the appropriate lambda.
(4) Set up subscriptions that specify which topic name each lambda will use to communicate from the device to the group. This way the lambdas can send results back to the cloud.

At this point, the group is configured and can be deployed. A group can be deployed through the Greengrass console or through the AWS API bulk deployments. When a group is deployed it will install each model onto its associated Jetson device and run them all. Each model can publish topics and the group will listen for them based on the subscriptions set up earlier. In this way we set up a Greengrass deployment that is analogous to a deployment onto a real vehicle. We used this setup to test the effectiveness of the proposed RAD framework for our task. Figure 5 depicts the Jetson/Greengrass ecosystem.

4 FRAMEWORK RESULTS SUMMARY & DISCUSSION

In addition to the accuracy of each of the models we trained (detailed in tables 3, 5), we also collected metrics for model deployment in the simulated vehicle environment. These results are shown in the Table 6.

The disk footprint measures the space on disk occupied only by the model parameters. However memory footprint is the space occupied on RAM by the model parameters, the data being processed by the model and the other dependencies required for the model to run. Hence, it should be the case that the RAM is higher than disk footprint as recorded above. Also, it should be noted that this is peak memory footprint. Note the additional model footprint necessary for converting the classification model from an edge centric architecture to a distributed architecture. This enables the model to retrain on the device.

Table 6: Memory, disk footprint and execution time characterization of distracted driving model for both the edge-centric and distributed model variants.

	Distracted Driver Model	
Architecture	Edge-Centric	Distributed
Model Accuracy	77%	70%
Time to Predict	0.071s	0.071s
Time to Read Model	N/A	0.007s
Memory Footprint (RAM)	208Mb	209Mb
Disk Footprint	76kb	76kb
Disk Space for Software Dependencies	568Mb	568Mb

4.1 Recommendations

For the distracted driving use-case, the amount of RAM required for running a single model is approximately 210 MB. The pipeline for distracted driving has been developed to employ models that are relatively frugal in their usage of memory during execution. In light of this property, it would be possible to run up to 8 - 10 models for distracted driving, or related tasks on the employed edge device simultaneously without running out of device memory or facing other performance issues. It is still possible to address multiple tasks with a single model (considering the same input) by leveraging multi-task learning approaches as proposed by Caruana et al. in [5].

5 CONCLUSIONS & FUTURE WORK

In this work, we have detailed our experiences in developing the RAD framework for leveraging CAN data to perform vehicle analytics. We have showcased the performance of the machine learning models supported in RAD, on the specific task of detecting distracted drivers. We have also detailed the entire RAD pipeline related to on-device deployment and cloud based training and showcased model memory, disk footprint and prediction times on-device. Finally, we now discuss the difficulties faced and how we plan to augment RAD moving forward.

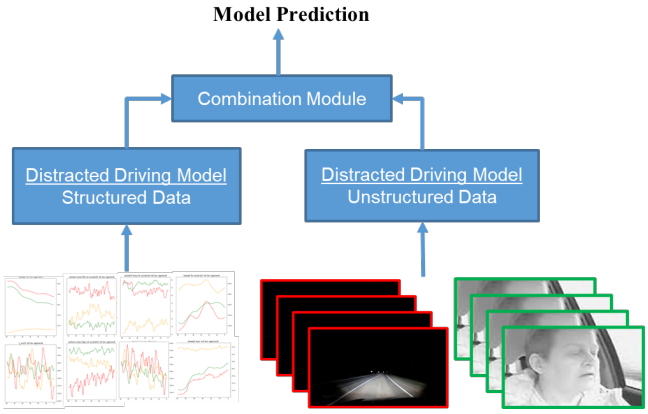

Figure 6: A multimodal system design that uses structured (from CAN) and unstructured (from camera) data.

5.1 Difficulties Faced

Limited data: One of the primary difficulties faced was in the case of the distracted driver detection task. The complexity of this task was due to the imbalanced nature of the data i.e., we had relatively little representative data for effectively training models to detect distracted drives and avoid bias towards attentive drives which was the majority class.

Limited number of distinguishing features: Another challenge we faced was to identify features which would enable us to effectively discriminate between the different driver states (i.e., attentive and distracted). The characterization of average drives for different features helped us narrow down the set of useful features which provided discriminatory capabilities.

Capabilities of OTS support tools: Another challenge was to develop a lightweight machine learning framework, capable of running multiple machine learning models in the same learning framework, all while remaining nimble enough to be deployed onto an edge device with memory and disk size constraints. The implementation of the framework hit some difficulties because the underlying technology is so new. Ultimately, as things mature, implementation of new features will become easier.

5.2 Future Work

Data Quantity and Accuracy Tradeoff: During this project, we faced the problem of seemingly not having enough data to train more accurate and effective models. Ford representatives identified that Ford developers are likely to run into similar problems. In addition, they will be faced with the problem of identifying how much data to collect to train a model. Collecting too much data will incur an additional cost. Not collecting enough will result in poorly trained models. Understanding this trade-off is critical to enabling an effective model development environment.

Unstructured Data: CAN data can be combined with images and other exogenous datasets to yield a multifaceted and holistic modeling of distracted and drowsy driving alleviating the limitations of our current feature set. This project has been extended to expand the RAD architecture to incorporate unstructured data. Fig. 6 showcases an architecture we plan to develop to incorporate image data into the drowsy and distracted driving detection process. The expanded project scope will also include analysis and "smart" collection of unstructured vehicle data. Similarly this work will provide insights into the architecture and framework necessary to deploy unstructured models in limited resource environments.

Federated Learning: Further research and development into federated learning and incorporating it into the RAD model development system would be valuable. This would further enhance the distributed nature of models and it would also enhance privacy limiting the amount of identifiable data that would be transferred to the cloud.

6 ACKNOWLEDGEMENTS

This work has been partially supported by the Ford Motor Company and by the National Science Foundation via grants DGE-1545362 and IIS-1633363.

REFERENCES

[1] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep?. In *Advances in neural information processing systems*. 2654–2662.
[2] Colby R Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, et al. 2020. Benchmarking TinyML Systems: Challenges and Direction. *arXiv preprint arXiv:2003.04821* (2020).
[3] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
[4] Kenneth L Campbell. 2012. The SHRP 2 naturalistic driving study: Addressing driver performance and behavior in traffic safety. *Tr News* 282 (2012).
[5] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.
[6] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.
[7] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, and Yuan Tang. 2015. Xgboost: extreme gradient boosting. *R package version 0.4-2* (2015), 1–4.
[8] Yunyao Cheng, Bin Wu, Li Song, and Chuan Shi. 2019. Spatial-Temporal Recurrent Neural Network for Anomalous Trajectories Detection. In *International Conference on Advanced Data Mining and Applications*. Springer, 565–578.
[9] Rishu Chhabra, Seema Verma, and C Rama Krishna. 2017. A survey on driver behavior detection techniques for intelligent transportation systems. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*. IEEE, 36–41.
[10] Erik Coelingh, Andreas Eidehall, and Mattias Bengtsson. 2010. Collision warning with full auto brake and pedestrian detection-a practical example of automatic emergency braking. In *13th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 155–160.
[11] Jerome H Friedman. 2002. Stochastic gradient boosting. *Computational statistics & data analysis* 38, 4 (2002), 367–378.
[12] Matthew L Ginsberg, Erin O Keenan, Louie V McCrady, and Paul AC Chang. 2011. Driver Safety System Using Machine Learning. US Patent App. 12/886,100.
[13] David Hallac, Suvrat Bhooshan, Michael Chen, Kacem Abida, Jure Leskovec, et al. 2018. Drive2vec: Multiscale state-space embedding of vehicular sensor data. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 3233–3238.
[14] David Hallac, Abhijit Sharang, Rainer Stahlmann, Andreas Lamprecht, Markus Huber, Martin Roehder, Jure Leskovec, et al. 2016. Driver identification using automobile sensor data from a single turn. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 953–958.
[15] Mee Lan Han, Jin Lee, Ah Reum Kang, Sungwook Kang, Jung Kyu Park, and Huy Kang Kim. 2015. A statistical-based anomaly detection method for connected cars in internet of things environment. In *International Conference on Internet of Vehicles*. Springer, 89–97.
[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
[17] Bartosz Jachimczyk, Damian Dziak, Jacek Czapla, Pawel Damps, and Wlodek J Kulesza. 2018. IoT on-board system for driving style assessment. *Sensors* 18, 4 (2018), 1233.
[18] Ashish Kumar, Saurabh Goyal, and Manik Varma. 2017. Resource-efficient machine learning in 2 KB RAM for the internet of things. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1935–1944.

[19] Omar Makke and Oleg Gusikhin. 2018. Connected Vehicle Prognostics Framework for Dynamic Systems. In *International Conference on Intelligent Information Technologies for Industry*. Springer, 3–15.

[20] Christian Manasseh and Raja Sengupta. 2013. Predicting driver destination using machine learning techniques. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 142–147.

[21] María Victoria Martínez, Ines Del Campo, Javier Echanobe, and Koldo Basterretxea. 2015. Driving behavior signals and machine learning: A personalized driver assistance system. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. IEEE, 2933–2940.

[22] Julian Schlechtriemen, Andreas Wedel, Joerg Hillenbrand, Gabi Breuel, and Klaus-Dieter Kuhnert. 2014. A lane change detection approach using feature ranking with maximized predictive power. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 108–114.

[23] Yogesh Simmhan. 2017. IoT analytics across edge and cloud platforms. *IEEE IoT Newsletter* (2017).

[24] Reid Simmons, Brett Browning, Yilu Zhang, and Varsha Sadekar. 2006. Learning to predict driver route and destination intent. In *2006 IEEE Intelligent Transportation Systems Conference*. IEEE, 127–132.

[25] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. 2016. Anomaly detection in automobile control network data with long short-term memory networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 130–139.

[26] Bo Wang, Smruti Panigrahi, Mayur Narsude, and Amit Mohanty. 2017. *Driver identification using vehicle telematics data.* Technical Report. SAE Technical Paper.

[27] Sujay Yadawadkar, Brian Mayer, Sanket Lokegaonkar, Mohammed Raihanul Islam, Naren Ramakrishnan, Miao Song, and Michael Mollenhauer. 2018. Identifying Distracted and Drowsy Drivers Using Naturalistic Driving Data. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019–2026.