NeuOS: A Latency-Predictable Multi-Dimensional Optimization Framework for DNN-driven Autonomous Systems

Soroush Bateni and Cong Liu, *University of Texas at Dallas*

https://www.usenix.org/conference/atc20/presentation/bateni

This paper is included in the Proceedings of the
2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Open access to the Proceedings of the
2020 USENIX Annual Technical Conference
is sponsored by USENIX.

NeuOS: A Latency-Predictable Multi-Dimensional Optimization Framework for DNN-driven Autonomous Systems

Soroush Bateni and Cong Liu
The University of Texas at Dallas

Abstract

Deep neural networks (DNNs) used in computer vision have become widespread techniques commonly used in autonomous embedded systems for applications such as image/object recognition and tracking. The stringent space, weight, and power constraints seen in such systems impose a major impediment for practical and safe implementation of DNNs, because they have to be latency predictable while ensuring minimum energy consumption and maximum accuracy. Unfortunately, exploring this optimization space is very challenging because (1) smart coordination has to be performed among system- and application-level solutions, (2) layer characteristics should be taken into account, and more importantly, (3) when multiple DNNs exist, a consensus on system configurations should be calculated, which is a problem that is an order of magnitude harder than any previously considered scenario. In this paper, we present NeuOS, a comprehensive latency predictable system solution for running multi-DNN workloads in autonomous systems. NeuOS can guarantee latency predictability, while managing energy optimization and dynamic accuracy adjustment based on specific system constraints via smart coordinated system- and application-level decision-making among multiple DNN instances. We implement and extensively evaluate NeuOS on two state-of-the-art autonomous system platforms for a set of popular DNN models. Experiments show that NeuOS rarely misses deadlines, and can improve energy and accuracy considerably compared to state of the art.

1 Introduction

The recent explosion of computer vision research has led to interesting applications of learning-driven techniques in autonomous embedded systems (AES) domain such as object detection in self-driving vehicles and image recognition in robotics. In particular, deep neural networks (DNNs) with generally the same building blocks have been dominantly applied as effective and accurate implementation of image recognition, object detection, tracking, and localization towards enabling full autonomy in the future [60, 50]. For example, using such DNNs alone, Tesla has recently demonstrated that a great deal of autonomy in self-driving cars can be achieved [33]. Another catalyzer for the feasibility of DNN-driven autonomous systems in practice has been the advancement of fast, energy-efficient embedded platforms, particularly accelerator-enabled multicore systems such as the NVIDIA Drive AGX and the Tesla AI platforms [44, 22].

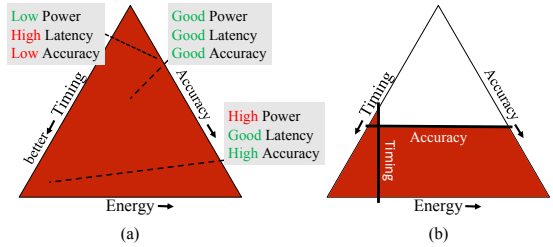


Figure 1: Ternary depiction of the 3D optimization space.

Autonomous systems based on embedded hardware platforms are bounded by stringent Space, Weight, and Power (SWaP) constraints. The SWaP constraints require system designers to carefully take into account energy efficiency. However, DNN-driven autonomous embedded systems are considered mission-critical real-time applications and thus, require predictable latency[1] and sufficient accuracy[2] (of the DNN output) in order to pass rigorous certifications and be safe for end users [46]. This causes a challenging conflict with energy efficiency since accurate DNNs require a tremendous amount of resources to be feasible and to be timing-predictable, and are by far the biggest source of resource consumption in such systems [5]. This usually results in less complicated (and less resource-demanding) DNN models to be designed and used in these systems, reducing accuracy considerably.

Fig. 1(a) shows a hypothetical three-dimensional space between latency, power, and accuracy mapped to a ternary plot [55] (where (Energy + Timing + Accuracy) has been normalized to 3). Each dot in Fig. 1(a) represents a configuration with a unique set of latency, power, and accuracy characteristics. The power consumption is usually

[1] Latency from each system component (including the DNNs) in AES will add up to the reaction latency between when a sensor observes an event and when the system externally reacts to that event, such as by applying the breaks in a self-driving vehicle. The faster a system reacts, the more likely it is for the system to avoid a disaster, such as an accident. However, policymakers might adopt a reasonable reaction time, such as 33ms or even 300ms [48, 27, 12, 14, 9, 4] as "safe enough".

[2] We should mention here that there is currently no established standard to connect DNN accuracy to the safety of a particular system, such as DNNs in self-driving vehicles. In this paper, we assume the more accurate the DNN, the safer the system is.

adjusted at system-level via dynamic voltage/frequency scaling (DVFS) [5, 21]. The accuracy adjustment is done at application-level via DNN approximation configuration switching (see Sec. 4 for details). Note that both DVFS and DNN configuration adjustments would impact runtime latency. This figure highlights three configurations with various levels of latency, power consumption, and accuracy tradeoff that might or might not be acceptable given the current performance constraints. Choosing the best three-dimensional trade-off optimization point is a significant challenge given the vast and complex DVFS and accuracy configuration space.

Although all autonomous systems are required to be latency predictable in nature, the constraints on power and accuracy may vary based on the type of autonomous system (e.g., highly constrained power for drones and maximum accuracy requirement for autonomous driving). To illustrate one such variation, note Fig. 1(b), which shows a constraint on latency, and a constraint on accuracy imposed in the configuration space limiting the possible configurations considerably.

Challenges specific to DNN-driven AES. In addition to the aforementioned optimization problem, DNNs are constructed from layers, where each layer responds differently to DVFS changes and has unique approximation characteristics (as we shall showcase in Sec. 3.1). In order to meet a latency target with optimized energy consumption and accuracy, each layer requires a unique DVFS and approximation configuration, whereas existing approaches such as Poet [23] and JouleGuard [21] deal with DNNs as a black-box. Moreover, system-level DVFS adjustments and application-level accuracy adjustments happen at two separate stages. Without smart coordination, the system might fall in a negative feedback loop, as we shall demonstrate in Sec. 3.2. This coordination needs to happen at layer boundaries, making the problem at least an order of magnitude harder than previous work.

Furthermore, existing techniques mostly focus on single-tasking scenarios [5, 3, 20] whereas AES generally require multiple instances of different DNNs. As we shall motivate in Sec. 3.3 using a real-world example, these DNNs need to communicate and build a cohort on a layer-by-layer basis to avoid greedy and inefficient decision-making. Moreover, system-level and application-level coordination in this multi-DNN scenario is much harder than isolated processes considered in previous work.

Finally, existing approaches [13, 5] optimize latency performance on a best-effort basis (e.g., by using control theory) that can overshoot a latency target (as demonstrated in Sec. 3.2). A better solution should include proven real-time runtime strategies such as LAG analysis [51].

Contribution. In this paper, we present NeuOS[3], a comprehensive timing-predictable system solution for multi-DNN workloads in autonomous embedded systems. NeuOS can manage energy optimization and dynamic accuracy adjustment for DNNs based on specific system constraints via smart coordinated system- and application-level decision-making.

NeuOS is designed fundamentally based on the idea of multi-DNN execution by introducing the concept of cohort, a collective set of DNN instances that can communicate through a shared channel. To track this cohort, we address how latency, energy, and accuracy can be measured and propagated efficiently in the multi-DNN cohort.

Besides the fundamental goal of providing latency predictability (i.e., meeting deadlines for processing each DNN instance), NeuOS addresses the challenge of balancing energy at system level and accuracy at application level for DNNs, which has never been addressed in literature to the best of our knowledge. Balancing three constraints at various execution levels in the multi-DNN scenario requires smart coordination 1) between system level and application level decision making, and 2) among multiple DNN instances.

Towards these coordination goals, we introduce two algorithms in Sec. 4.2 that are executed at the layer completion boundary of each DNN instance: one algorithm that can predict the best system-level DVFS configuration for each DNN member of the cohort to meet deadline and minimize power for that specific member in the upcoming layer, and one algorithm that decides what application level approximation configuration is required for others if any one of these system-level DVFS decisions were chosen. These two algorithms effectively propagate all courses of action for the next layer in order to meet the deadline. Based on these two algorithms, we propose an optimization problem in Sec. 4.3 that can decide the best course of action depending on the system constraint, and minimize system overhead. This method is effective because 1) it introduces an identical decision-making among all DNN instances in the cohort and solves the coordination problem between system-level and application-level decision making, and 2) provides adaptability to three typical scenarios imposing different constraints on energy and accuracy.

Implementation and Evaluation. We implement a system prototype of NeuOS and extensively evaluate NeuOS using popular image detection DNNs as a representative of convolutional deep neural networks used in AES. The evaluation is done under the following conditions:

- **Extensible in terms of architecture.** We fully implement NeuOS using a set of popular DNN models on two different platforms: an NVIDIA Jetson TX2 SoC (with architecture designed for low overhead embedded systems), and an NVIDIA AGX Xavier SoC (with architecture designed for complex autonomous systems such as self-driving cars).
- **Multi-DNN scenarios.** We ensure that our system can trade-off and balance multiple DNNs in all conditions by testing NeuOS under three cohort sizes: a small 1-process, a medium 2-4 process, and a large 6-8 process.

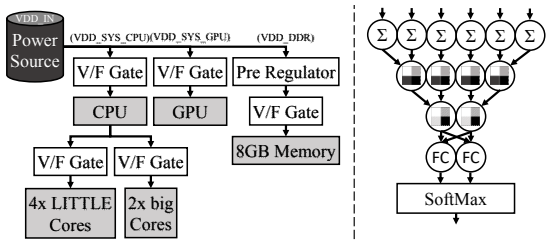
[3]The latest version of NeuOS can be found at https://github.com/Soroosh129/NeuOS.

- **Latency predictability.** We extensively compare NeuOS to six state-of-the-art solutions in literature, and find that NeuOS rarely misses deadlines under all evaluated scenarios, and can improve runtime latency on average by 68% (between 8% and 96% depending on DNN complexity) on TX2, by 40% on average (between 12% and 89%) on AGX, and by 54% overall.
- **Versatility.** NeuOS can be easily adapted to the following three constraint scenarios:
  - **Balanced energy and accuracy.** Without any system constraints given, NeuOS is proved to be energy efficient while sacrificing an affordable degree of accuracy, improving energy consumption on average by 68% on TX2, by 40% on average on AGX, while incurring an accuracy loss of 21% on average (between 19% and 42%).
  - ***Min energy.*** When energy is constrained to be minimal, NeuOS is able to sacrifice accuracy a small amount (at most 23%) but further improve energy consumption by 11% over the general unrestricted case, while meeting the latency requirement.
  - ***Max accuracy.*** When accuracy is given as a constraint, NeuOS is able to improve accuracy by 10% on average compared to balanced case, but also sacrifices energy by only a small amount, increasing by 23% on average.

2 Background

DVFS space in autonomous systems. The trade-off between latency and power consumption is usually achieved via adjustments to frequency and/or voltages of hardware components. A software and hardware technique typical of modern systems is DVFS. Through DVFS, system software such as the operating system or hardware solutions can dynamically adjust voltage and frequency. To understand this technique better, consider Fig. 2(a), showing the components of a Jetson TX2, which contains a Parker SoC with a big.LITTLE architecture with 2 NVIDIA Denver big cores and 4 ARM Cortex A53 LITTLE cores. The Parker SoC also contains a 256-core Pascal-architecture GPU. The TX2 module also contains 8 GB of shared memory (the Jetson AGX Xavier also used in Sec. 5 has a more advanced Xavier SoC with 8 NVIDIA "Carmel" cores, a 512 Volta-architecture GPU, and 16GB of shared memory). Each component includes a voltage/frequency (V/F) gate that can be adjusted via software. The value for frequency and voltage for each component forms a unique topple, called a DVFS configuration throughout this paper.

DNN and its approximation techniques. Fig. 2(b) depicts a simplified version of a Deep Neural Network (DNN). Neurons are the basic building blocks of DNNs. Depending on the layer neurons belong to, they perform various different operations. A DNN may contain multiple layers of different types, such



(a) Jetson TX2 V/F Structure (b) An example DNN

Figure 2: DVFS configuration space and DNN structure.

as the convolutional and the normalization layers, which are connected via their inputs and outputs.

DNNs by nature are approximation functions [36]. DNNs are trained on a specific training set. After training, accuracy is measured by using a test data set, set aside from the training set and measuring the accuracy (e.g., top-5 error rate– comparing the top 5 guesses against the ground truth). The accuracy of the overall DNN can be adjusted by manipulating the layer parameters.

A rich set of DNN approximation techniques have been proposed in the literature and adopted in the industry [17, 58, 24, 29, 43, 56, 7, 18]. Such techniques aim at reducing the computation and storage overhead for executing DNN workloads. An example technique to provide approximation for convolutional layers is Lowrank [45], which performs a lowrank decomposition of the convolution filters. In our implementation, dynamic accuracy adjustment or "hot swapping" layers will refer to applying the lowrank decomposition to the upcoming layers before their execution. Note that applying such approximation adjustments on the fly is possible because the generated pair of layers have the exact combined input and output dimensions. Moreover, this adjustment is only possible for future layers at each layer boundary.

Measuring Accuracy. The approximation on the fly will affect the final accuracy. Due to the dynamic nature of this adjustment, the exact value of accuracy measurement using traditional methodology is impractical. Most related work thus incorporate an alternative scoring method [3], where the system will deduce the accuracy score accordingly if certain approximation techniques are to be applied to the next layer. In our method, we assume a perfect score for the original DNN, and switching to the lowrank approximation of any layer will reduce the score by a set amount. For example, running AlexNet in its entirety will result in a score of 100. If we swap a convolutional layer with a lowrank version of that layer, the overall accuracy will be affected by some amount (e.g., 1 in our method), thus yielding a lower score (e.g., 99 under the scoring method). Therefore, the score is always relative to the original DNN configuration and not related to the absolute value of accuracy on a particular dataset. This method of keeping relative accuracy is still invaluable to maximizing accuracy in a dynamic runtime environment but cannot be used to calculate the exact accuracy loss.

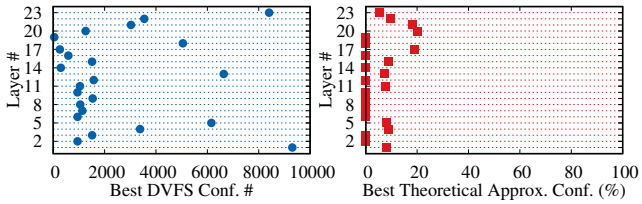


Figure 3: Calculated best system level DVFS configuration and best application level theoretical approximation configuration for AlexNet on Jetson TX2 in order to meet a 12ms deadline (0 means no approximation).

3 Motivation

In this section, we lay out several motivational case studies to understand the challenges that exist for DNNs, and gain insights on why existing approaches (or naively extended ones) may fail under our problem context.

3.1 Balancing in two-dimensional Space

The trade-off to meet a specified latency target while maximizing accuracy is done in a 2-dimensional space by choosing an approximation configuration for the application. Similarly, the 2-dimensional trade-off between energy and latency is done by changing an optimal DVFS configuration. Traditional control-theory based solutions treat the entire application as a black-box, and decide on what DVFS or approximation configuration should be chosen every few iterations of that specific application [21, 20]. However, treating DNNs as a blackbox does not yield the most efficient results. Fig. 3 left hand shows the best DVFS configuration for each layer of AlexNet among all possible DVFS configurations for a Jetson TX2 in terms of energy consumption. The y-axis is the layer number for AlexNet, and the x-axis is the DVFS configuration index, partially sorted based on frequency and activated core counts. The dots show the configuration that has the absolute minimum energy consumption. As is evident, each layer has a different optimal DVFS configuration. More interestingly, we observe a non-linearity where sometimes faster DVFS configurations have lower energy consumption. This is due to the massive parallelism of GPUs, where increasing the frequency by 2x for example can yield a 10-fold improvement in performance, which outweighs the momentary increase in energy consumption. Fig. 3 right hand shows the best theoretical approximation configurations required for each layer of AlexNet in order to meet a 12ms deadline[4]. As is evident in the figure, each layer requires a different approximation configuration for optimal results.

Thus, the DNN must somehow become transparent to the system, conveying layer-by-layer information in order to make the correct decisions. This can make the decision space in the 2-dimensional space at least an order of magnitude harder (e.g., AlexNet has 23 layers) since every layer must be considered for each execution of the DNN application.

[4]Please see Sec. 4.2 for more details on how this is calculated.

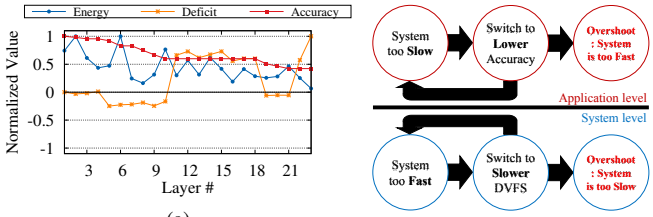


Figure 4: Negative feedback loop between an application-level solution and a system-level solution.

Observation 1: Layer-level trade-off makes the problem an order of magnitude harder than ordinary blackbox techniques.

3.2 Balancing in three-dimensional Space

Balancing energy/latency and accuracy/latency in isolation can be naive, and lead to unnecessary consumption of energy or reduced accuracy. Fig. 4a shows a similar experiment to Sec. 3.1, but both the system and application (Alexnet) are employed at the same time without any coordination. The goal of both solutions is to reach a 20ms deadline (by using latency deficit, LAG, as a guide (Sec. 4.2)). In the case of AlexNet, the system-level DVFS adjustment can be enough to meet the desired deadline. In an ideal scenario, only energy is adjusted slightly until AlexNet is not behind schedule. However, as is evident in the figure, normalized energy consumption and accuracy for each layer are both decreased continuously and dramatically. This is due to an unwanted negative loop, where a negative deficit (indicating that the system is behind schedule) has resulted in the application-level solution switching to a lower approximation configuration. Because these configurations are discrete, as we shall discuss in Sec. 4.2, the deficit will overshoot (at around layer 10) and becomes positive (meaning the system is ahead of schedule). The system-level solution would see this deficit as a headroom to reduce energy consumption, and in the case of Fig. 4a, has turned the positive deficit into a small negative at around layer 18. This cycle (as depicted in Fig. 4b) is repeated until the minimum approximation configuration is reached. This result is extremely undesirable in accuracy-sensitive applications such as autonomous driving (but can be okay for energy sensitive applications such as remote sensing). Thus, a feasible solution would be for the system and application to communicate, and make decisions based on given constraints for an application based on given constraints. This communication should be done at the granularity of layers, which makes the problem extra hard.

Observation 2: Trade-off in a 3-dimensional latency, energy, and accuracy optimization space is a significant challenge due to both system constraints as well as lacking harmony between application-level and system-level solutions.

3.3 Balancing for Multi-DNN Scenarios

To the best of our knowledge, no existing approach deals with multiple DNN instances in a coordinated manner.

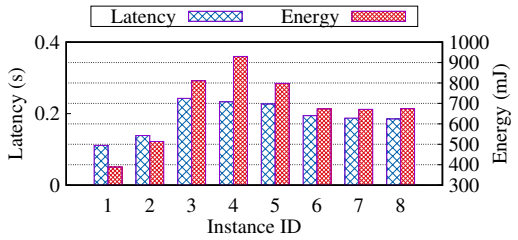


Figure 5: Energy consumption and execution time of running 8 instances of Resnet-50 on a Jetson TX2 under PredJoule.

Straightforwardly extending single-tasking latency/energy trade-off approaches, such as PredJoule [5], to multi-tasking scenarios would only result in decision-making that is local and greedy, based on locally measured variables. To showcase why coordination in this additional dimension is a key issue, examine Fig. 5, which shows the latency and energy consumption for running 8 DNN instances together averaged over 20 iterations under PredJoule on a Jetson TX2. We chose PredJoule because in our experiments, it outperformed all other existing solutions on exploring the 2D tradeoff between latency and energy for DNNs. The left (right) y-axis in Fig. 5 depicts the latency (energy consumption) in seconds (miliJoules) for each instance. As is evident in the figure, the DVFS management is greedy, resulting in instances 1 and 2 having relatively good latency and energy consumption. This greediness has pushed the rest of the DNN instances into unacceptable latency range (which is above 150ms for ResNet-50) because the chosen DVFS configuration at each layer boundary has been mostly beneficial only to the current layers of DNN instance 1 and 2. Moreover, the distribution of timing and energy consumption is not even across all instances because of the same reason. This disparity is the result of an uncoordinated system solution that chooses DVFS configurations greedily based on local variables.

Observation 3: In addition to the 2D and 3D complexities of solving the latency/accuracy/energy trade-off, a complete system solution must also accommodate for Multi-DNN scenarios, which are inherently more complicated to model and predict than single-DNN scenarios. The case studies also imply that naive extensions on existing single-DNN 2D solutions may fail in multi-DNN cases because they make greedy decisions based on local variables without coordination towards being globally optimal.

4 System Design

4.1 NeuOS Overview

To optimize the three-dimensional tradeoff space at the layer granularity, two basic research questions need to be answered first: 1) *how* to define and track the values of the three performance constraints in the system, and 2) *what* target should be imposed for optimizing each constraint.

For the first research question, we define a value of LAG (defined in Sec. 4.2, as a measurement of how far behind the DNN is compared to an ideal schedule that meets the relative deadline D), which tracks the progress of DNN execution at layer boundaries, P for energy consumption (in mJ) for each layer, and a variable X to reflect accuracy. We choose to track LAG at runtime instead of using an end-to-end optimization because it is more practical due to two reasons: 1) in a multi-DNN scenario, predicting the overlap between different DNN instances (and thus coordinating an optimal solution) cannot be done offline without making unrealistic assumptions, such as synchronized release times, and, 2) LAG is especially useful in a real system since it can account for outside interference, such as interference by other processes in the system, whereas an end-to-end optimization framework could miss the latency target. Moreover, as we shall discuss in Sec. 4.2, the value of P can be inferred by LAG in our design as these two variables fundamentally depend on the runtime DVFS configuration. Thus, the essential variables to track the status of a DNN execution can be simplified to $\{LAG, X\}$. Since we are dealing with a multi-DNN scenario, each DNN instance will have its own set of these variables. To know the collective status of the system, each DNN instance will put its variables in a shared queue.

In order to answer the second question regarding what optimization targets should be imposed on the system, we focus on the following three typical scenarios (expanded on in Sec. 4.3) that entail different performance constraints:

- **Min Energy** (M_P) is when NeuOS is deployed on an embedded system with a critically small energy envelope. Thus, the system should minimize energy without sacrificing too much accuracy. This scenario is motivated by applications seen in extremely power-limited systems such as drones, robotics, and a massive set of internet-of-thing devices.
- **Max Accuracy** (M_A) is when NeuOS is deployed on a system that has limited energy but accuracy is of utmost importance. Thus, the system should try to maximize accuracy without losing too much energy. This scenario is motivated by CPS-related applications such as autonomous driving.
- **Balanced Energy and Accuracy** (S) describes a more general, flexible scenario when the system is limited by both energy consumption and accuracy requirements, but no priority is given to either. Thus, the system should try to balance energy consumption and accuracy.

With the given scenarios and the values of $\{LAG, X\}$ at hand, we can answer the two key research questions presented in our motivation: 1) how to coordinate in a multi-DNN scenario such that the overall system is balanced and can meet the performance constraints, and, 2) how to efficiently tradeoff between latency, energy, and accuracy given the complexity of the problem space and how to prevent the negative feedback loop discussed in Sec. 3.2?

Design overview. Fig. 6 shows the overall design of NeuOS

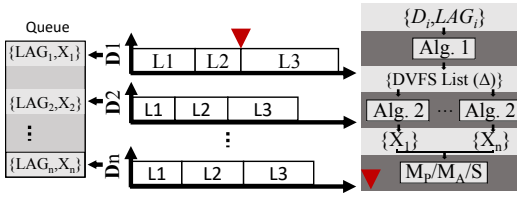


Figure 6: Design Overview

around $\{LAG, X\}$. The left side depicts the shared queue among multiple DNN instances. In the middle, a simple example of n concurrently running DNN instances each with three layers is shown. NeuOS makes runtime decisions on DVFS and DNN approximation configuration adjustments at layer boundaries, i.e., whenever a layer of a DNN instance completes. This is beneficial not only because applying approximation on-the-fly is possible only at layer boundaries, but in terms of overhead as well (as proved by our evaluation).

As illustrated in the figure, at the boundary between layers L2 and L3 of the first DNN instance, NeuOS is going through the process of decision-making which contains several steps. The first step is Alg.1, which senses the last known value of *LAG* for each DNN instance. Alg. 1 decides what DVFS configuration (at system level) is best for each instance in order to meet their deadlines D, outputting a list of potential DVFS configurations (Δ), where each member of the list corresponds to a DNN instance. In the next step, the list of potential DVFS configurations are fed into Alg.2, which predicts what approximation $\{X_i\}$ (at application level) would be required for other DNN instances to meet the deadline if the DVFS configuration for any one of the DNN instances is applied. Thus, Alg. 1 and Alg. 2 in tandem discover all possible courses of action the system can take to meet the deadline. However, at this point, no decision has been made on what DVFS configuration or accuracy configuration should be chosen for the system, because that depends on the given system constraint. This problem is inherently an optimization problem of finding the best possible choice in the propagated configuration space. We present this optimization problem formally in Sec. 4.3, where depending on broad scenarios, a particular setting is chosen for the next period of execution. In the last step of NeuOS, the system chooses one of these possibilities based on the scenario involved.

4.2 Coordinated System- and Application-level Adjustments

In this section, we expand on how runtime LAG is measured, how it relates to energy consumption, how accuracy X is calculated, and how the two developed algorithms take advantage of these two measurements to discover all possible choices the system can make efficiently in order to reduce the LAG to zero and meet the deadline.

LAG. We quantify the relationship between the partial execution time at time t of DNN instance i (e_i) and its relative deadline D_i as a form of LAG [51], denoted by LAG_i. LAG_i is a local variable (that can be updated at layer boundaries) for each DNN instance that keeps track of how far ahead or how far behind the DNN instance is compared to the deadline at time t. LAG_i is calculated as:

$$LAG_i(t, L_i(t)) = \sum_{l \in L_i(t)} (d_l - e_l), \tag{1}$$

in which $L_i(t)$ is the list of the layers of instance i that have completed by time t. For layer $l \in L_i(t)$, d_l and e_l depict the sub-deadline for layer l and the recorded execution time for layer l, respectively. NeuOS keeps track of e_l by measuring the elapsed time between each layer. Moreover, we use the proportional deadline method [38] to devise sub-deadlines for each layer based on D_i, the relative (end-to-end) deadline of DNN instance i, in which the subdeadline d_l for layer l is calculated as:

$$d_l = (e_l / \sum_{x \in L_i} (e_x)) \cdot D_i, \tag{2}$$

where $\sum_{x \in L_i}(e_x)$ denotes the execution time of DNN i. The proportional nature of sub-deadlines means that they only need to be calculated once for the lifetime of a given DNN instance on a platform.

Each DNN instance i would broadcast LAG_i among all instances via the shared queue. Thus, LAG_i would reflect the last known status of DNN instance i up to the last executed layer. We call the collection of LAG from all instances the LAG cohort, and we denote it by Φ. At completion of a DNN instance, a special message is sent to the cohort so that every DNN instance in the system is aware of their exit.

Based on the LAG cohort, the DNN instances can make decisions on accuracy and DVFS. A cohort will be perfect if every LAG within it is 0, or $\forall LAG_i \in \Phi, LAG_i = 0$. This means that all layers have exactly finished by their sub-deadline so far. Thus, the system has reasons to believe that the DNN instances will exactly finish by the deadline and do not require a faster DVFS or an approximation configuration, saving energy and accuracy in the process.

Since *LAG* indicates how far behind ($LAG < 0$) or ahead ($LAG > 0$) each DNN is, the DVFS and the approximation configuration need to be adjusted to run faster or slower accordingly. However, energy consumption and accuracy constraints must also be considered. We discuss each next.

System-level DVFS adjustment. At system-level, the question is which DVFS configuration is the best given the state of Φ to minimize energy consumption while reducing *LAG* to zero? The answer would vary between different DNN instances in the cohort, as they exhibit different LAGs. Moreover, different layers react differently to DVFS adjustments.

Alg. 1 is responsible for finding the best DVFS configuration for each DNN instance in the cohort. Alg. 1 takes as input the LAG cohort Φ and a SpeedUp/PowerUp table for the current layer of each DNN instance i. The structure of

Algorithm 1 Δ Calculator.

Input: Φ ▷ Progress Cohort
Input: SpeedUp/PowerUp[] ▷ The SpeedUp/PowerUp table of DNNs.
Output: Δ

1: **function** RETURN$\Delta(\Phi)$
2: **for** LAG_i in Φ **do**
3: $S_{P_i} \leftarrow \frac{D_i+LAG_i}{D_i}$.
4: $\delta_i \leftarrow$ **LookUp**(SpeedUp/PowerUp[S_{P_i}])

Table 1: SpeedUp/PowerUp and SpeedUp/Accuracy tables.

(a) SpeedUp/PowerUp for a layer of DNN instance i.

DVFS Configuration(δ)	SpeedUp	PowerUp
1	1x	1x
2	2.1x	2x
3	2.8x	1.5x

(b) SpeedUp/Accuracy.

X	SpeedUp
81%	1x
71%	1.8x
59%	2.5x

the SpeedUp/PowerUp table is depicted in Table 1a. The first column of Table 1a is the index for all the possible DVFS configurations in the system. The second column indicates how fast each DVFS configuration is in the worst case scenario compared to the baseline DVFS configuration (baseline is usually chosen to be the slowest configuration). The third column indicates how much power that DVFS configuration will consume relative to baseline.

Storing relative speedup and powerup values (instead of absolute measurements) is useful for looking up the table. In Alg. 1, given a LAG_i (line 2) and a relative deadline D_i for DNN instance i, the required speedup (denoted as S_i) could be directly calculated as (line 3):

$$S_P = \frac{D_i + LAG_i}{D_i}, \tag{3}$$

in which S_P is the speedup (or slowdown) value calculated as the relationship between the current projected execution time ($D_i + LAG_i$) and the ideal execution time (D_i). Since LAG can be negative or positive, the value of S_P can indicate a slowdown or speedup, where the slowdown is a way to conserve energy, which is the goal of NeuOS. The LookUp procedure (line 4) would then find the closest DVFS configuration that matches the speedup (or slowdown) in relation to the current configuration.

For our Alg. 1 to operate, we prepare a structure such as Table 1a for all DNN instances in a hashed format[5]. The LookUp procedure would then directly find a bucket by using the SpeedUp as an index. The output of Alg. 1 is a set $\Delta = \{\delta_1, \delta_2, ..., \delta_n\}$, in which δ_i is the ideal DVFS configuration for DNN instance i in order to meet the deadline. Imagine we ultimately decide that $\delta_c \in \Delta$ is the best DVFS configuration for the next scheduling period. A very interesting question would be that, what is the effect of applying δ_c on other DNN instances $i \neq c$? The speedup of δ_c for other DNN instances can be calculated by using δ_c as the lookup key in their corresponding SpeedUp/PowerUp table. But what if this speedup does not reduce LAG_i to zero? To solve this problem, we next present the algorithm that calculates the application-level approximation required to reduce LAG_i to zero given a DVFS configuration $\delta_c \in \Delta$.

[5]Our hashing is custom, and hashes the relationship between SpeedUp and PowerUp. This method relies on partially sorting the DVFS configuration space. You can find the latest hashing code at https://git.io/Jfogq

Algorithm 2 X_i Calculator.

Input: Δ ▷ Potential DVFS list.
Input: SpeedUp/Accuracy[] ▷ The SpeedUp/Accuracy table of DNNs.
Input: SpeedUp/PowerUp[] ▷ The SpeedUp/PowerUp table of DNNs.
Output: $X[][]$ ▷ The accuracy list for each DNN instance for each δ

1: **function** RETURN$X_i(\Delta)$
2: **for** δ_c in Δ **do**
3: **for** $i = 0$ to $i < |\Delta|$ **do**
4: $S_{A_i} \leftarrow \frac{S_{P_i}(\delta_c)\cdot(D_i+LAG_i)}{D_i}$
5: $X[c][i] \leftarrow$ **LookUp**(SpeedUp/Accuracy[S_{A_i}])

Application-level accuracy adjustment. Alg. 2 portrays the procedures to calculate the required approximation for the upcoming layers of all DNN instances based on a DVFS configuration. If the instance i is behind the ideal schedule by LAG_i, with a relative deadline of D_i, and if the chosen DVFS configuration is δ_c, the remaining required speedup can be calculated as follows (line 4):

$$S_{A_i}(\delta_c) = \frac{S_{P_i}(\delta_c) \cdot (D_i + LAG_i)}{D_i}, \tag{4}$$

in which $S_{A_i}(\delta_c)$ is the required speedup (or slowdown) via approximation for DNN instance i when DVFS configuration δ_c is chosen, and $S_{P_i}(\delta_c) \cdot (D_i + LAG_i)$ is the new projected execution time of DNN instance i. The value of S_{A_c}, the speedup from accuracy for the chosen DVFS configuration, should always be zero or less than zero since by definition, δ_c is the ideal DVFS configuration for c and requires no additional speedup from approximation.

The value of S_{A_i} is then used as a lookup key to a new table, called the SpeedUp/Accuracy table, depicted in Table 1b. Table 1b stores the relative worst case execution times for each layer's approximation configuration. We index each row by X, which is the value of the total accuracy of that configuration[6]. Note that the exact value of X has no effect in the algorithm and what matters is the relative order in Table 1b (i.e., the lower we go down the table, the lower the relative accuracy). The output of Alg.2 is the row index in the SpeedUp/Accuracy table sufficient to meet the deadline for all DNN instances except c. We denote this index for layer k of DVFS configuration i as X_i^k. This value is then broadcasted in the accuracy cohort and indicates the application-level

[6]Each row could be indexed by any measure. However, indexing with X has benefits in overhead reduction for the LookUp procedure in Alg. 2 because it can be more easily hashed.

configuration chosen for the next immediate layer of the corresponding DNN instance.

The remaining question is that which δ_c should be chosen. We answer this question next.

4.3 Constraints and Coordination

The combination of Alg. 1 and Alg. 2 produces a list of potential DVFS configurations Δ, and for each DVFS configuration in Δ, a corresponding list of required approximations for all DNN instances in the cohort if that DVFS configuration were to be applied. Such a scenario can be visualized as a decision tree. The remaining question of our design would be which path to go down to in order to have a perfect *LAG* cohort. As discussed in Sec. 3.2, the requirements on energy and accuracy can vary depending on specific scenarios. We present the following three approaches based on the three scenarios defined in Sec. 4.1, i.e., minimum energy (M_P), maximum accuracy (M_A), and balanced energy and accuracy (S).

Min Energy. This approach aims at minimizing power usage at the cost of accuracy. To choose the best DVFS configuration in the DVFS candidate set Δ, we should look at the corresponding $S_{P_i}(\delta_c), \delta_c \in \Delta$ values in the SpeedUp/PowerUp table and choose the δ_c that has the smallest PowerUp value for that corresponding DNN instance, namely:

$$\delta_c = \{\delta_i \in \Delta \mid PowerUp_i(\delta_i) \leq PowerUp_i(\delta_x), \forall \delta_x \in \Delta\}, \tag{5}$$

in which $PowerUp_i(\delta_i)$ is extracted from the SpeedUp/PowerUp table of DNN instance i. Note that in our experience, the values of PowerUp can be non-linear in relation to SpeedUp, and hence, a comprehensive search as noted above is required. Then, using Alg. 2, the accuracy cohort can be calculated and broadcasted based on the projected new execution times. Even though this approach has the best power consumption, it will not have the best accuracy since many processes will most likely not meet the deadline without significant loss of accuracy, since the speedup from DVFS alone will likely not make up for the vast majority of the progress values in the cohort.

Max Accuracy. In this method, our system chooses the DVFS configuration δ_c in such a way that:

$$\delta_c = \{\delta_i \in \Delta \mid \sum(S_{Aj}(\delta_i)) \leq \forall \sum(S_{Aj}(\delta_x \in \Delta)), \\ \forall \text{ DNN instance j in cohort}\}, \tag{6}$$

in which $\sum S_{Aj}(\delta_i)$ is the sum of all the required speedups from approximation (S_{Ai}) for configuration δ_i, and $\leq \forall \sum(S_{Aj}(\delta_x \in \Delta))$ is indicating that the sum of approximation-induced speedup for the chosen δ_c should be less than or equal any other sum of approximation values for other $\delta_x \in \Delta$ (this indirectly ensures minimized accuracy loss).

Statistical Approach for Balanced Energy and Accuracy. To achieve balanced energy and accuracy, we propose a statistical approach that checks the state of Δ and the projected accuracy cohort in statistical terms to make a decision. The calculation of S_{P_i} and S_{A_i} (which depends on S_{P_i}) resemble the form of Bivariate Regression Analysis (BRA) [57], in which:

$$S_{A_i} = S_{P_i} \cdot \frac{D_i + LAG_i}{D_i} + 0, \tag{7}$$

in which, $\frac{D_i+LAG_i}{D_i}$ is called the influence of S_{P_i} on the required approximation. To measure this influence, we first calculate

$$I = \sum_{i=0}^{i=n-1} \frac{D_i + LAG_i}{D_i}, \tag{8}$$

in which I is the collective influence of LAG on approximation. If the value of I is high, it means that the accuracy can be more adversely affected by a low value of DVFS-induced speedup(S_{P_i}). Similarly, a low value of I means that the accuracy can remain minimal even with a low value for DVFS-induced speedup. We simplify our decision making by dividing the *LAG* cohort Φ into three groups based on how big or small the value of *LAG* is. The boundary for the intervals is calculated using:

$$Boundary = \frac{\max\{\Phi\} - \min\{\Phi\}}{3}. \tag{9}$$

The three groups $G1[0...Boundary]$, $G2[Boundary...2 \cdot Boundary]$, and $G3[2 \cdot Boundary...3 \cdot Boundary]$ are then formed, and the ultimate DVFS configuration is chosen as:

$$\delta_c = \begin{cases} median(G1) & if(I < t) \\ median(G2) & if(t < I < 1+t) \\ median(G3) & if(I > 1+t) \end{cases}, \tag{10}$$

in which, t is a threshold for I, set to the standard deviation σ of the set I. However, t can be chosen by the system designer to indicate a requirement on power consumption and accuracy. A small value for t will push the system towards faster DVFS configurations and vice versa.

Discussion on choosing modes and safety. We would like to conclude our design by a discussion on which modes to choose and the safety concern it might entail. Our stand from a system perspective is to design a flexible system architecture that can adapt to various external needs. Where absolute mission-critical applications are concerned, we offer Max Accuracy. Nonetheless, depending on the safety requirement, our Balanced approach might be good enough with the proper threshold t even for applications such as self-driving vehicles. However, choosing a mode dynamically at runtime or statically for a particular system offline has more to do with the certification standards (which are in their preliminary stages for self-driving vehicles) as well as the requirement on maximum reaction time and accuracy. Thus, we believe the decision should be relegated to an external policy controller [54, 31, 6, 34, 52]

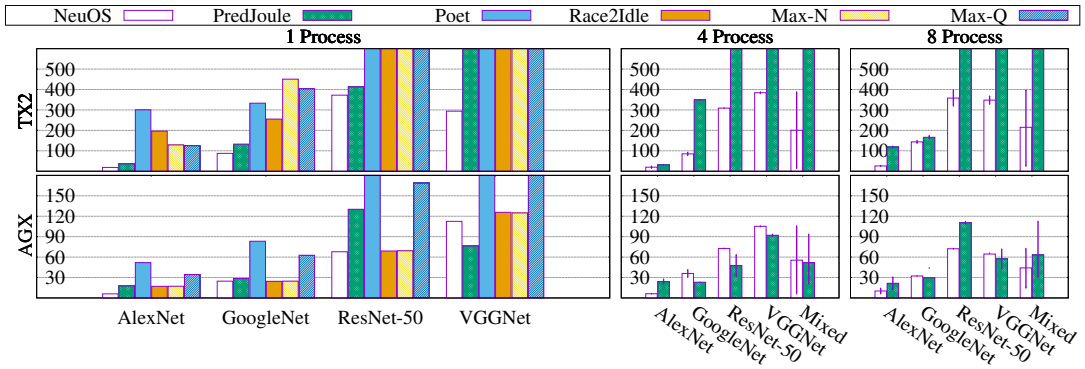


Figure 7: Energy under various methods (in mJ) for 1, 4, and 8 instances of 4 DNN models.

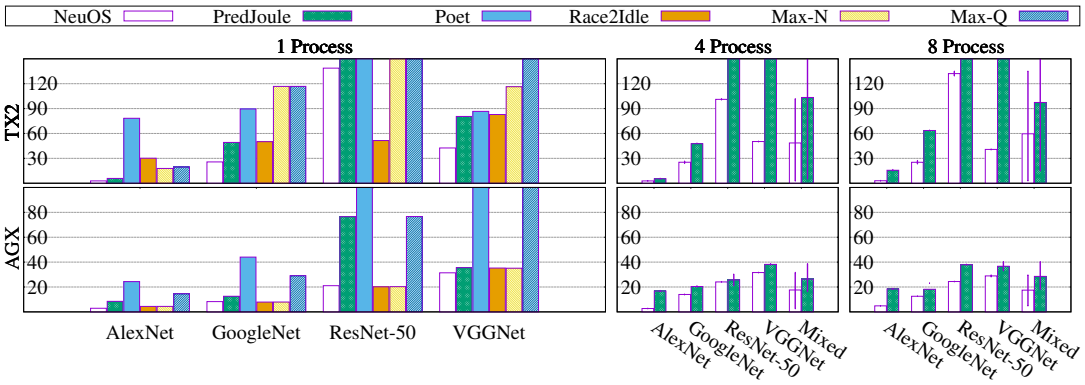


Figure 8: Latency under various methods (in ms) for 1, 4, and 8 instances of 4 DNN models.

5 Evaluation

In this section, we test our full implementation on top of Caffe [25] with an extensive set of evaluations.

5.1 Experimental Setup

In this section we lay out our experimental setup, which includes two embedded platforms and four popular DNN models. We compare NeuOS to 6 existing approaches.

Testbeds. We have chosen two different NVIDIA platforms imposing different architectural features (since deployed autonomous systems solutions, particularly for autonomous driving and robotics, seem to gravitate towards NVIDIA hardware as of writing this paper [26, 30]) to showcase the cross-platform nature of our design when it comes to hardware. We use NVIDIA Jetson TX2, with 6 big.LITTLE ARM-based cores and a 256-core Pascal based GPU with 11759 unique DVFS configurations, and the NVIDIA Jetson AGX Xavier, the latest powerful platform for robotics and autonomous vehicles with an 8-core NVIDIA Carmel CPU and a 512-core Volta-based GPU with 51967 unique DVFS configurations.

DNN models. Having a diversified portfolio of DNN models can showcase that NeuOS is future proof in the fast-moving field of neural networks. To that end, we use AlexNet [32], ResNet [19], GoogleNet [2], and VGGNet [49] in our experiments. Our method dynamically applies a lowrank version of a convolutional layer whenever approximation is necessary by keeping both version of the layer in memory for fast switching. The deadline for each DNN instance is based on their worst-case execution time (WCET) on each platform, and is set to 10ms, 30ms, 150ms, and 40ms respectively for Jetson TX2 and 5ms, 10ms, 25ms, 30ms respectively for AGX Xavier. Note that ResNet is much slower on Jetson TX2 due to the older JetPack software.

Small Cohort, Medium Cohort, Large Cohort sizes. We test NeuOS under three different cohort size classes to test for adaptability and balance: 1 process for small, 2 to 4 processes for medium, and 6 to 8 processes for large. Each of these cohort sizes have their own unique challenges. We measure average timing, energy consumption, and accuracy for these scenarios and provide a measure of balancing where applicable. For medium and large cohorts, we include a mixed scenario, where different DNN models are executed, which represents systems that use different DNNs (for example for voice and image recognition). For the medium cohort, one instance of each DNN model and for the large cohort, two instances of each model are initiated.

Adaptability to different system scenarios. As discussed in Sec. 4.1, we consider three different scenarios with different limits on latency, energy and accuracy: minimum energy, maximum accuracy, and balanced energy and accuracy.

Compared Solutions. We implement and compare six state-

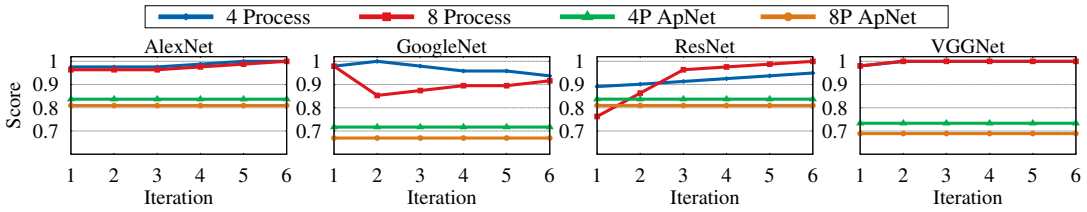


Figure 9: Average accuracy (y-axis as a fraction of 1) of the cohort over iteration of execution for 4 different DNN models with 4 and 8 instances compared to ApNet (x-axis is the iteration number).

of-the art solutions from the literature, including DNN-specific and DNN-agnostic ones, software-based DVFS and hardware-based DVFS, and application-level and system-level solutions. We present a short detail for each as follows.

PredJoule [5] is a system-level solution tailored towards DNN by employing a layer-based DVFS adjustment solution for optimization latency and energy. *Poet* [23] is a system-level control-theory based software solution that balances energy and timing in a best-effort manner via adjusting DVFS. We choose to compare against Poet instead of its extended approaches including JouleGuard [21] and CoAdapt [20], as they employ essentially the same set of control theory-based techniques as Poet. *ApNet* [3] is an application-level solution based on DNNs that can theoretically provide a per-layer approximation requirement offline to meet deadlines. *Race2Idle* [28] is the classic "run it as fast as you can" philosophy, which is always interesting to compare to. *Max-N* [10, 11] is a reactive hardware DVFS that maximizes frequency and sacrifices energy in the name of speed, in NVIDIA embedded hardware. *Max-Q* [10] is a hardware DVFS on Jetson TX2 that dynamically adjusts DVFS on the fly to conserve energy. However, this feature has been removed from the Xavier platform [11], and is replaced by low level power caps, such as 10W, 15W, and 30W. We use the 15w cap instead of Max-Q on Xavier.

5.2 Overall Effectiveness

In this section, we measure the efficacy of NeuOS on the two evaluated platforms under the balanced scenario. Since our design is concerned with timing predictability, energy consumption, and DNN accuracy, we measure all three constraints and compare against state-of-the-art literature under each platform and each scenario.

5.2.1 Small Cohort

Energy. The left column of Fig. 7 depicts our measurements in terms of average energy consumption compared to a GPU-enabled Poet, Max-Q, Max-N, PredJoule, and Race2Idle using AlexNet, GoogleNet, ResNet-50 and VGGNet as the base DNN model and using lowrank as the approximation method. As is evident in the figure, NeuOS is able to save energy considerably compared to all other methods on Jetson TX2 on all DNN models, with improvements of 68% on average for Jetson TX2 and 46% on average for AGX Xavier. This saving is due to the fact that in some cases accuracy is minimally traded off for the benefit of energy and timing.

On the Jetson AGX Xavier, NeuOS has better energy consumption compared to all other approaches on every DNN model except compared to PredJoule for VGGNet. As we shall see for timing, PredJoule misses the deadline of 30 for VGGNet, and NeuOS has decided to sacrifice energy to meet timing.

Latency. Fig. 8 shows the average execution time for NeuOS compared to the 5 methods and using 4 DNN models. NeuOS outperforms all other approaches, improving on average execution time by 68% on Jetson TX2 and by 40% on AGX Xavier. It is also interesting to note that AGX Xavier is much faster than Jetson TX2, by 70% on average.

Tail Latency. Through response time measurements, we find that NeuOS rarely misses the deadline (3.25% of the time). Moreover, the variance is low with the 99th percentile execution time for AlexNet, GoogleNet, ResNet-50, and VGGNet as 9.2 ms, 48 ms, 130.3 ms and 39.1 ms for TX2 and 5.0 ms, 12.0 ms, 26.1 ms and 36.2 ms for AGX respectively.

Accuracy. We also measure the accuracy loss of NeuOS compared to ground truth and compared to ApNet (Fig. 9 omits small cohort for clarity). ApNet is the only DNN-specific application level solution we are aware of. NeuOS has an approximation score of 0.94% on average (out of 1), which is better than ApNet by 21%.

5.2.2 Medium and Large Cohorts

In order to save space, we only compare PredJoule for the 4-process medium and 8-process large cohort sizes. In our testings, PredJoule already vastly outperforms other methodologies, and thus is a good comparison to NeuOS.

Energy. As is evident in the second and third columns of Fig. 7, NeuOS can almost always outperform PredJoule in terms of energy consumption on Jetson TX2 improving 70% on average. However, rather interestingly, NeuOS performs worse in terms of energy compared to PredJoule for GoogleNet, ResNet, and VGGNet on AGX Xavier. This is due to the fact that PredJoule again misses the deadlines on AGX Xavier, and NeuOS has sacrificed a negligible amount of energy (1.5% on average) in order to meet the deadline.

Latency. NeuOS always outperforms state-of the art, improving by 53% on average for Jetson TX2, and by 32% on average for AGX. This is due to the fact that NeuOS is able to leverage a small amount of accuracy and energy loss (in the case of AGX Xavier) for better timing and energy characteristics.

Tail Latency. We find that deadline miss ratio is about the

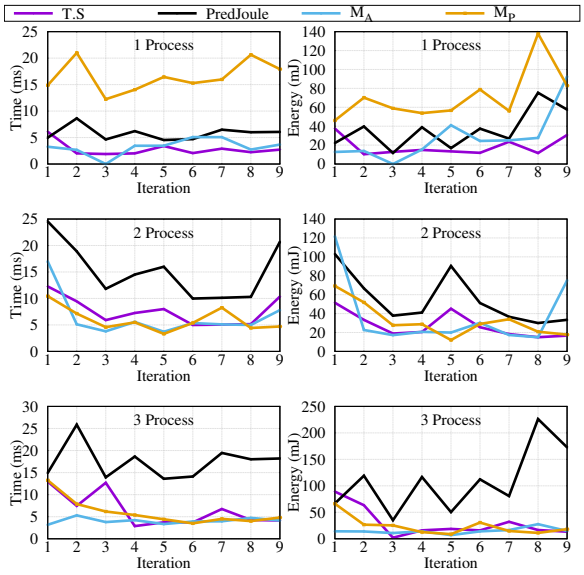


Figure 10: Performance of NeuOS under three scenarios compared to PredJoule on Jetson TX2.

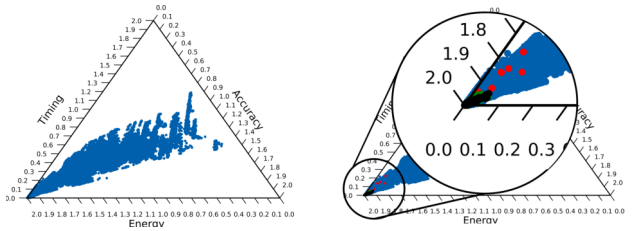
same as the small cohort. Moreover, the variance is similarly low with the 99th percentile execution time for AlexNet, GoogleNet, ResNet-50, and VGGNet as 10.4 ms, 39.2 ms, 101.7 ms and 69 ms for TX2 and 11 ms, 12.5, 26.3 and 35.9 ms for AGX respectively for the medium cohort and 13.6 ms, 40.8 ms, 190 ms and 72 ms for TX2 and 10.7 ms , 54 ms , 62 ms and 36.1 ms for AGX respectively for the large cohort.

Accuracy. Fig. 9 shows the average accuracy of the cohort over 6 iterations on AGX Xavier. As is evident in the figure, NeuOS generally improves upon accuracy as the system progresses because the optimization in Sec. 4.3 is able to find better DVFS configurations. When compared to the efficient approximation-aware solution APnet, NeuOS is able to achieve noticeably better accuracy in all scenarios.

Balance. A very important measure discussed in Sec. 3.3 is how balanced the system solution is when faced with multiple processes. To measure how balanced NeuOS is compared to PredJoule in the 4-process and 8-process scenarios, we include min-max bars in Fig. 7 and Fig. 8 to showcase the discrepancy between minimum and maximum timing/energy. As is evident in the figure, the discrepancy is negligible compared to the total energy consumption and execution time (up to 79 mJ and 4 ms). Thus, NeuOS maintains balance in the cohort. This is due to the coordinated cohorts and a uniform non-greedy decision making approach introduced in our design.

5.3 Detailed Examination on Tradeoff

In this section, we focus on the fact that system designers might require certain constraints that limits the ability of NeuOS in a certain dimension. To this end, we test our platform under three different scenarios: Maximum Accuracy (M_A), Minimum Power (M_P), and Balanced ($T.S$).



(a) The entire configuration space for all DVFS and accuracy combinations for Jetson TX2.

(b) Chosen configurations in the triangle space.

5.3.1 Energy and Latency.

We compare against PredJoule and measure average timing for the cohort in ms and average energy in mJ in Fig. 10 for 1-process (small), 2-process (medium), and 3-process (large) scenarios on AlexNet over 9 iterations on the Jetson TX2. PredJoule is shown as a black line. The deadline in this scenario is set to 25 to show the interesting characteristics of each method.

Balanced. As is evident in the figure, our statistical balanced approach outperforms PredJoule over all iterations. Notably in the case of medium and large cohorts, PredJoule has a particularly bad start in terms of timing and has higher fluctuation due to the greedy nature of DVFS selection.

Min Energy. Interestingly, M_P performs very bad (still meets the deadline) for the small cohort both in terms of timing and energy consumption. This is due to the algorithms discussed in Sec. 4.3. The system has switched to a very slow DVFS configuration to save energy. However, because of the non-linearity inherent in very slow DVFS configurations for GPUs [5], this has resulted in a very bad energy consumption as well. However, the coordination starts to pay off for medium and large cohorts. This is because a coordinated multi-process cohort needs faster DVFS configurations and thus, the circumstances push M_P out of the slow and power inefficient DVFS configuration subset. The greediness of PredJoule is inherent for medium and large cohorts in the form of very large fluctuations throughout the iterations.

Max Accuracy. M_A should improve accuracy while sacrificing energy and timing. We shall discuss the accuracy decisions shortly. However, the timing for M_A is worse than balanced energy by a negligible amount. The same is true for energy consumption (23% on average). This highlights a big design decision of the balanced scenario overall. Even for the balanced general approach, sacrificing accuracy is done very conservatively as was discussed earlier. Thus, the slight push toward perfect accuracy does not introduce very large overheads. However, as was discussed earlier guaranteeing a tight deadline (such as 10ms for AlexNet) requires some approximation if energy consumption is a consideration.

5.3.2 Energy-Accuracy Tradeoff.

For accuracy and to show where the variations of NeuOS jump in terms of system and application configurations, we bring back the triangle of Fig. 1, but with real DVFS and

Table 2: Average execution time overhead of NeuOS compared to other approaches on AlexNet (ms).

	1 Process	4 Process	8 Process
NeuOS	0.145	0.571	0.738
PredJoule	0.772	0.929	1.597
ApNet	0	3.27	5.85
Poet	151.03	604.12	1208.27

accuracy configurations with the selected configurations of M_P, M_A, and $T.S$ highlighted in Fig. 11b. As is evident in the triangle, the deadline limits the possible configurations to the bottom left corner. However, within that limitation, M_P (in red) has chosen configurations that are lower on energy consumption toward the upper right. On the other hand M_A (in green) has chosen configurations that are not as good in terms of energy consumption, but are better in terms of accuracy toward bottom left. Finally, $T.S$, colored black, is similar to M_A because of the high emphasis on accuracy in our design.

5.4 Overhead

Execution time: Table 2 shows the overhead of NeuOS compared to Poet, PredJoule, and ApNet using AlexNet as the baseline model on Jetson TX2 (times are in milliseconds). As is evident in Table 2, the overhead for NeuOS is negligible, especially compared to Poet and the overhead is also negligible compared to the overall execution time of AlexNet itself. The reason Poet is so slow is because it has to go through all DVFS configurations in a quadratic way ($O(n^2)$, n is the number of DVFS configurations). Even $O(n)$ would be unacceptable on embedded systems with more than 10000 unique DVFS configurations. This proves that applying our complexity reduction techniques (via hashing) is a must for a practical system solution. Moreover, NeuOS is more efficient than PredJoule and ApNet, especially in 4 Process and 8 Process scenarios.

Memory: As discussed in Sec. 5.1, our implementation keeps both the original and the lowrank approximation of the model in GPU memory for fast switching at layer boundaries. Moreover, NeuOS also holds the per-layer hash tables containing approximation and DVFS configuration information as described in Sec. 4. Table 3 depicts the added overhead in terms of both raw and percentage of the total NeuOS memory usage. As expected, the lowrank approximated version of each model has slightly less overall size compared to the original model. Nonetheless, this technique sacrifices memory overhead to improve latency and energy consumption. A viable alternative left as future work is dynamic approximation on the fly, which trades off latency for lower memory consumption. Moreover, the overhead of the hash tables is negligible compared to the total memory usage. Finally, the last two columns depict the cumulative maximum percentage of total available memory occupied by one instance of NeuOS for each platform (this memory usage also includes the temporary intermediate layer data [39]).

Table 3: Raw and percentage based memory overhead of applying NeuOS for each model.

	(a) Overhead in addition to Caffe			(b) Ratio to total memory	
	Lowrank	Hash Table	Ratio	Jetson TX2	AGX Xavier
AlexNet	226 MB	331 B	49%	10%	4%
GoogleNet	23 MB	2.1 KB	30%	2%	1%
ResNet-50	82 MB	3.2 KB	45%	7%	3%
VGGNet	509 MB	634 B	48%	25%	12.7%

6 Related Work

Trading off latency and power efficiency has been a hot topic in the related fields including real-time embedded systems and mobile computing [40, 8, 41, 53, 59, 16, 37, 42, 47, 1]. Due to the explosion of approximation techniques in different application domains, there has been several recent works [15, 35, 13] seeking to address this problem in a three dimensional space covering accuracy as well. Unfortunately, these works cannot resolve the problem as their applicability is limited in scope in various ways. JouleGuard [21], MeanTime [13], CoAdapt [20] and other similar approaches provide general system or hardware solution for non-DNN applications that claim to explore the three dimensional optimization space.

A very recent set of works including PredJoule [5], and ApNet [3] are able to provide latency predictability (meeting deadlines) for DNN-based workloads, yet they only consider two out of the three dimensions and focus on single-DNN scenarios. As we have discussed in Sec. 3.2, running ApNet and PredJoule at the same time even at different frequencies will result in a negative feedback loop. Thus, a better, more coordinated approach is required.

Such limitations would dramatically reduce the complexity of the optimization space, as the system-level and application-level tradeoffs focusing on a single task can be considered in an independent manner (e.g., pure system-level and application-level optimization under Poet [23] and CoAdapt, respectively). This results in solutions that are inapplicable to any practical autonomous real-time system featuring a multi-DNN environment.

7 Acknowledgment

We would like to thank our shepherd, Amitabha Roy, and the anonymous referees for their invaluable insight and advice into making this paper substantially better. This work is supported by NSF grant CNS CAREER 1750263.

8 Conclusion

This paper presents NeuOS, a comprehensive latency predictable system solution for running multi-DNN workloads in autonomous embedded systems. NeuOS can guarantee latency predictability, while managing energy optimization and dynamic accuracy adjustment based on specific system constraints via smart coordinated system- and application-level decision-making among multiple DNN instances. Extensive evaluation results prove the efficacy and practicality of NeuOS.

References

[1] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 1–13. IEEE Press, 2016.

[2] Mohammadhossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab Mirrokni. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems*, pages 2591–2599, 2014.

[3] Soroush Bateni and Cong Liu. Apnet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 67–79, Dec 2018.

[4] Soroush Bateni and Cong Liu. Predictable data-driven resource management: an implementation using autoware on autonomous platforms. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 339–352. IEEE, 2019.

[5] Soroush Bateni, Husheng Zhou, Yuankun Zhu, and Cong Liu. Predjoule: A timing-predictable energy optimization framework for deep neural networks. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 107–118, Dec 2018.

[6] Raunak P Bhattacharyya, Derek J Phillips, Changliu Liu, Jayesh K Gupta, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Simulating emergent properties of human driving behavior using multi-agent reward augmented imitation learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 789–795. IEEE, 2019.

[7] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[8] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 27–39. IEEE Press, 2016.

[9] European Commission. Rolling plan for ict standardisation, 2018.

[10] NVIDIA Corp. Power management for jetson agx xavier devices. https://docs.nvidia.com/jetson/l4t/index.html#page/TegraLinuxDriverPackageDevelopmentGuide/power_management_jetson_xavier.html.

[11] NVIDIA Corp. Power management for jetson tx2 series devices. https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/index.html#page/TegraLinuxDriverPackageDevelopmentGuide/power_management_tx2_32.html.

[12] ETSI. Intelligent transport systems (its); vehicular communications; basic set of applications; part 3: Specifications of decentralized environmental notification basic service, Aug 2018.

[13] Anne Farrell and Henry Hoffmann. Meantime: Achieving both minimal energy and timeliness with approximate computing. In *USENIX Annual Technical Conference*, pages 421–435, 2016.

[14] ISO International Organization for Standardization. 26262: 2018. *Road vehicles—Functional safety*.

[15] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 123–136, New York, NY, USA, 2016. ACM.

[16] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[17] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[18] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Henry Hoffmann. Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, pages 223–232, 2014.

[21] Henry Hoffmann. Jouleguard: energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 198–214. ACM, 2015.

[22] Sean Hollister. Tesla's new self-driving chip is here, and this is your best look yet., Apr 2019.

[23] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 75–86, April 2015.

[24] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[25] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[26] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296. IEEE, 2018.

[27] Jaanus Kaugerand, Johannes Ehala, Leo Mõtus, and Jürgo-Sören Preden. Time-selective data fusion for in-network processing in ad hoc wireless sensor networks. *International Journal of Distributed Sensor Networks*, 14(11), 2018.

[28] D. H. K. Kim, C. Imes, and H. Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*, pages 78–85, Aug 2015.

[29] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[30] B. Kisacanin. Deep learning for autonomous vehicles. In *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 142–142, May 2017.

[31] Olga Kouchnarenko and Jean-François Weber. Adapting component-based systems at runtime via policies with temporal patterns. In *International Workshop on Formal Aspects of Component Software*, pages 234–253. Springer, 2013.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[33] TIMOTHY B. LEE. Tesla's autonomy event: Impressive progress with an unrealistic timeline, Apr 2019.

[34] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168. IEEE, 2011.

[35] Yongbo Li, Yurong Chen, Tian Lan, and Guru Venkataramani. Mobiqor: Pushing the envelope of mobile edge computing via quality-of-result optimization. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1261–1270. IEEE, 2017.

[36] Shiyu Liang and R Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

[37] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 255–266. IEEE Press, 2016.

[38] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[39] Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta. Modeling the resource requirements of convolutional neural networks on mobile devices. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1663–1671, 2017.

[40] M. Maggio, E. Bini, G. Chasparis, and K. Årzén. A game-theoretic resource manager for rt applications. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 57–66, July 2013.

[41] Nikita Mishra, Huazhe Zhang, John D Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 267–281. ACM, 2015.

[42] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 267–278. IEEE Press, 2016.

[43] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[44] Francisca Rosique, Pedro J Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3):648, 2019.

[45] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

[46] Rick Salay, Rodrigo Queiroz, and Krzysztof Czarnecki. An analysis of iso 26262: Using machine learning safely in automotive software. *arXiv preprint arXiv:1709.02435*, 2017.

[47] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[48] Akshay Kumar Shastry. *Functional Safety Assessment in Autonomous Vehicles*. PhD thesis, Virginia Tech, 2018.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[50] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4241–4247. IEEE, 2017.

[51] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *17th IEEE Real-Time Systems Symposium*, pages 288–299, Dec 1996.

[52] Chen Tang, Zhuo Xu, and Masayoshi Tomizuka. Disturbance-observer-based tracking controller for neural network driving policy transfer. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[53] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.

[54] Jean-François Weber. Tool support for fuzz testing of component-based system adaptation policies. In *International Workshop on Formal Aspects of Component Software*, pages 231–237. Springer, 2016.

[55] Wikipedia contributors. Ternary plot - wikipedia,the free encyclopedia. https://en.wikipedia.org/wiki/Ternary_plot, 2019. [Online; accessed 31-May-2019].

[56] Jian Xue, Jinyu Li, Dong Yu, Mike Seltzer, and Yifan Gong. Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6359–6363. IEEE, 2014.

[57] Taro Yamane. Statistics: An introductory analysis. 1973.

[58] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint*, 2017.

[59] Huazhe Zhang and Henry Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *SIGPLAN Not.*, 51(4):545–559, March 2016.

[60] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic autonomous driving system testing. *arXiv preprint arXiv:1802.02295*, 2018.

PhysGAN: Generating Physical-World-Resilient Adversarial Examples for Autonomous Driving

Zelun Kong[†], Junfeng Guo[†], Ang Li[‡,*] and Cong Liu[†]
[†]The University of Texas at Dallas
[‡]The University of Maryland, College Park

Abstract

Although Deep neural networks (DNNs) are being pervasively used in vision-based autonomous driving systems, they are found vulnerable to adversarial attacks where small-magnitude perturbations into the inputs during test time cause dramatic changes to the outputs. While most of the recent attack methods target at digital-world adversarial scenarios, it is unclear how they perform in the physical world, and more importantly, the generated perturbations under such methods would cover a whole driving scene including those fixed background imagery such as the sky, making them inapplicable to physical world implementation. We present PhysGAN, which generates physical-world-resilient adversarial examples for misleading autonomous driving systems in a continuous manner. We show the effectiveness and robustness of PhysGAN via extensive digital- and real-world evaluations. We compare PhysGAN with a set of state-of-the-art baseline methods, which further demonstrate the robustness and efficacy of our approach. We also show that PhysGAN outperforms state-of-the-art baseline methods. To the best of our knowledge, PhysGAN is probably the first technique of generating realistic and physical-world-resilient adversarial examples for attacking common autonomous driving scenarios.

1. Introduction

While deep neural networks (DNNs) have established the fundamentals of vision-based autonomous driving systems, they are still vulnerable to adversarial attacks and exhibit erroneous fatal behaviors. Recent works on adversarial machine learning research have shown that DNNs are rather vulnerable to intentional adversarial inputs with perturbations focusing on classification problems [4, 12, 19, 22, 25]. To address the safety issues in autonomous driving systems, techniques were proposed to automatically generate adversarial examples, which add small-magnitude perturbations to inputs to evaluate the robustness of DNN-based autonomous driving systems [12, 8, 27].

Figure 1: Illustration of an adversarial roadside advertising sign (top-right) visually indistinguishable from the original sign (top-left) and its deployment in the physical world (bottom).

However, these techniques mostly focus on generating *digital* adversarial examples, *e.g.*, changing image pixels, which can never happen in real world [12]. They may not be applicable to realistic driving scenarios, as the perturbations generated under such techniques would cover the whole scene including fixed background imagery such as the sky. Very recently, a few works took the first step in studying physical-world attacking/testing of static physical objects [2, 17], human objects [24, 7], traffic signs [23, 18, 8]. Although they are shown effective under the targeted scenarios and certain assumptions, they focus on studying a static physical-world scene (*e.g.*, a single snapshot of a stop sign [8, 23]), which prevent themselves to be applied in practice as real-world driving is a continuous process where dynamically changes are encountered (*e.g.*, viewing angles and distances). Moreover, their generated adversarial examples are visually unrealistic (*e.g.*, driver-noticeable black and white stickers pasted onto a stop sign which is easily noticeable for attack purposes [8]). Most of these methods also have a focus on classification models different from our studied steering model which is a regression model. Also note that directly extending the existing digital per-

[*]Now at DeepMind

turbation generation techniques (e.g., FGSM) to physical world settings, i.e., inputting just the targeted roadside sign into such techniques would output a corresponding adversarial example, may be ineffective. The resulting attack efficacy may dramatically decrease (proved in our evaluation as well) as the process of generating perturbations has not considered any potential background imagery in the physical world (e.g., the sky) which would be captured by any camera during driving.

We aim at generating a realistic single adversarial example which can be physically printed out to replace the corresponding original roadside object, as illustrated in Fig. 1. Since the target vehicle observes this adversarial printout continuously, a major challenge herein is how to generate a single adversarial example which can continuously mislead the steering model at every frame during the driving process. Additionally, for a practical physical-world deployment, any generated adversarial example shall be visually indistinguishable from its original sign (the one already deployed in the physical world).

To address these challenges, we propose a novel GAN-based framework called PhysGAN [1] which generates a single adversarial example by observing multiple frames captured during driving while preserving resilience to certain physical-world conditions. Our architecture contains an encoder (*i.e.*, the CNN part of the target autonomous driving model) that extracts features from frames during driving and transforms them into a vector serving as the input to the generator. By considering all factors extracted from the frames, this design ensures that the generator could generate adeversarial examples that have attack effect. Without this encoder, the efficacy would dramatically decrease.To generate an adversarial example that can continuously mislead the steering model, PhysGAN takes a 3D tensor as input. This enhances the resilience of the generated example against certain physical world dynamics, as using video slices makes it more likely to capture such dynamics.

We demonstrate the effectiveness and robustness of PhysGAN through conducting extensive digital- and real-world experiments using a set of state-of-the-art steering models and datasets. Digital experimental results show that PhysGAN is effective for various steering models and scenarios, being able to mislead the average steering angle by up to 21.85 degrees. Physical case studies further demonstrate that PhysGAN is sufficiently resilient in generating physical-world adversarial examples, which is able to mislead the average steering angle by up to 19.17 degrees. Such efficacy is also demonstrated through comparisons against a comprehensive set of baseline methods.

To the best of our knowledge, PhysGAN is the first technique of generating realistic and physical-world-resilient adversarial examples for attacking common autonomous steering systems. Our contributions can be summarized in three folds as follows.

- We propose a novel GAN-based framework PhysGAN which can generate physical-world-resilient adversarial examples corresponding to any roadside traffic/advertising sign and mislead autonomous driving steering model with the generated visually indistinguishable adversarial examples.
- We propose a GAN architecture using 3D tensor as input in optimizing the generator, which resolves a key technical challenge in physical-world deployment of using a single adversarial example to continuously mislead steering during the entire driving process.
- We conduct extensive digital and physical-world evaluations with several metrics, which shows the superior attack performance of PhysGAN over state-of-the-art methods. We believe PhysGAN could contribute to future safety research in autonomous driving.

2. Related Works

Adversarial Attacks. Many works have recently been proposed to generate adversarial examples for attacking in the white-box setting [21], where the adversary knows the network's parameters. The fast gradient sign method (FGSM) [11] represents the pioneer among such methods, which performs a one-step gradient update along the direction of the sign of gradient at each pixel. FGSM is further extended in [13] to a targeted attack strategy through maximizing the probability of the target class, which is referred as the OTCM attack. Optimization-based approaches [26, 14, 4, 5, 29] have also been proposed. GAN was recently introduced in [10], implemented by a system of two neural networks competing with each other in a zero-sum game framework. GAN achieves visually appealing results in both face generation [16] and manipulation [30]. [29] presents AdvGAN, which leverages GAN to produce adversarial examples with high attack success rate on classification problems. These methods focus on applying perturbations to the entire input and consider only digital-world attacking scenarios. It is hard to apply them to the real world because it is impossible to use some of the generated perturbations to replace the real-world background (*e.g.*, the sky).

Generating Physical Adversarial Examples. To the best of our knowledge, only a very recent set of works [15, 8] started working on generating physical attacks. [15] focuses on the understanding of static physical adversarial examples. [8] explicitly designs perturbations to be effective in the presence of diverse real-world conditions. Their method mainly focuses on the classification of physical road sign under dynamic distance and angle of the viewing camera. Unfortunately, these works focus on static attacking scenarios (e.g., maximizing the adversarial effectiveness

[1] https://github.com/kongzelun/physgan.git

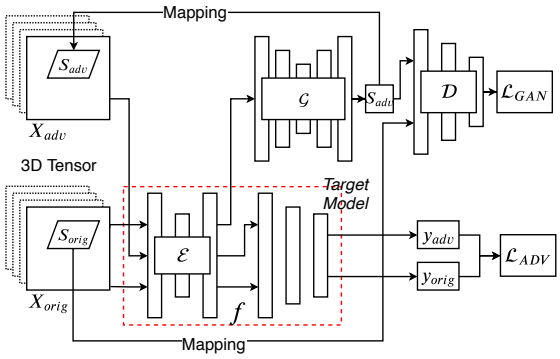


Figure 2: Overview of the PhysGAN framework.

w.r.t. to a single snapshot of the physical example) and thus do not require to resolve the one-to-many challenge.

Different from these works, PhysGAN is able to generate physical-world-resilient adversarial examples only corresponding to the roadside traffic/advertising sign; no perturbations will be generated on areas other than the street sign. PhysGAN addresses the one-to-many challenge which continuously attack the steering model, and generates realistic adversarial examples that are resilient to various physical world conditions and visually indistinguishable from the original roadside sign.

3. Our Approach: PhysGAN

The goal of PhysGAN is to generate an adversarial example that is visually indistinguishable from any common roadside object (*e.g.*, roadside traffic or advertising signs) to continuously mislead the steering angle model (target model) of a drive-by autonomous driving vehicle by physically replacing the roadside board with the adversarial example. When an autonomous driving vehicle drives by the roadside sign, the steering angle model would be fooled and make incorrect decision.

3.1. Problem Definition

We define our problem and notations in this section. Let $\mathcal{X} = \{X_i\}$ be the video slice set such that $\mathcal{X} \subseteq \mathbb{R}^{n \times w \times h}$, where n is the number of frames in the video slice, w and h is the *width* and *height* of a frame, respectively. Let $\mathcal{Y} = \{Y_i\}$ be the ground truth steering angle set, $\mathcal{Y} \subseteq \mathbb{R}^n$. Suppose (X_i, Y_i) is the i^{th} sample in the dataset, which is composed of video slice $X_i \in \mathcal{X}$ and $Y_i \in \mathcal{Y}$, each element of which denotes the ground truth steering angle corresponding to its frame. The pre-trained target steering model (e.g., Nvidia Dave-2, Udacity Cg23 and Rambo) learns a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ from the video slice set $\mathcal{X}$ to the ground truth steering angle set $\mathcal{Y}$ during training phase.

Given an instance (X_i, Y_i), the goal of PhysGAN is to produce an adversarial roadside sign S_{adv}, which aims to mislead the target autonomous driving model f as $f(X_i) \neq Y_i$ and maximize $\sum_i |f(X_i) - Y_i|$. To achieve the goal, PhysGAN needs to generate an adversarial roadside sign S_{adv} to replace original roadside sign S_{orig} in digital- or physical-world. The adversarial roadside sign S_{adv} is supposed to be close to the original roadside sign S_{orig} in terms of ℓ^2-norm distance metrics, which implies that adversarial roadside sign S_{adv} and original roadside sign S_{orig} are almost visually indistinguishable.

3.2. Physical World Challenges

Physical attacks on an object must be able to survive changing conditions and remain effective at fooling the steering angle model. We structure our decision of these conditions around the common drive-by scenarios (*i.e.*, the vehicle drives towards the roadside sign).

The "One-to-Many" challenge. A key technical challenge is to resolve the "one-to-many" challenge, *i.e.*, generating a single adversarial sample to continuously mislead the steering angle decision of a vehicle throughout the entire driving process. Considering multiple frames in generating an adversarial sample is challenging because the vehicle-to-board distance, view angle, and even subtle pixels on each frame could be different. An effective adversarial sample must be able to exhibit maximum overall attack effect among all the frames. To achieve this goal, the adversarial sample needs to be resilient to the changing conditions exhibited on each frame. To resolve this problem, PhysGAN applies a novel GAN-based framework and consider the entire drive-by video slice, rather than a single frame, as the input in the generation process (see Sec. 3.5).

Limited manipulated area. Unlike most digital-world adversarial methods which add perturbations to the entire input image, techniques focused on physical-world scenarios are constrained to add perturbations only to a fragment of an image, *i.e.*, the fragmented area corresponding to the original physical object. Moreover, the underlying assumption of a static image background does not hold in physical attacks since the background can consistently change over the driving process.

3.3. PhysGAN Overview

Fig. 2 illustrates the overall architecture of PhysGAN, which mainly consists of four components: an encoder $\mathcal{E}$, a generator $\mathcal{G}$, a discriminator $\mathcal{D}$ and the target autonomous driving model f. The encoder $\mathcal{E}$ represents the convolutional layers of target autonomous driving model f, which takes 3D tensors as inputs and is used to extract features of a video (of both original and perturbed ones). To resolve the challenge of generating only a single example which continuously impacts the driving process, we introduce a novel idea of considering 3D tensors as inputs in the GAN-based framework. 2D tensors often represent images while 3D

tensors are used to represent a small slice of video, which typically contains hundreds of frames.

As seen in Fig. 2, the extracted features of original video slice X_{orig} are used as the input fed to the generator to generate an adversarial roadside sign S_{adv}. Doing so allows us to take into account the fact that different original video slice X_{orig} may have different influence on the generated adversarial roadside sign S_{adv}, thus ensuring the generator $\mathcal{G}$ to generate the best adversarial roadside sign S_{adv} corresponding to a certain original video slice X_{orig}. The adversarial roadside sign S_{adv} and original roadside sign S_{orig} are sent to the discriminator $\mathcal{D}$, which is used to distinguish the adversarial roadside sign S_{adv} and the original roadside sign S_{orig}. The discriminator $\mathcal{D}$ represents a loss function, which measures the visual distinction between adversarial roadside sign S_{adv} and original roadside sign S_{orig}, and also encourages the generator to generate an example visually indistinguishable to the original sign.

3.4. Training GAN Along With the Target Model

In order to ensure the adversarial roadside sign S_{adv} to have adversarial effect on the target autonomous driving model f, we introduce the following loss function:

$$\mathcal{L}_{ADV}^{f} = \beta \exp\left(-\frac{1}{\beta} \cdot l_f(f(X_{orig}), f(X_{adv}))\right) \quad (1)$$

where β is a sharpness parameter, l_f denotes the loss function used to train the target autonomous driving model f, such as MSE-loss or ℓ^1-loss, X_{orig} denotes the original video slice X_{orig}, and X_{adv} represents the adversarial video slice X_{adv}, which is generated by mapping the adversarial roadside sign S_{adv} into every frame of the original video slice X_{orig}. By minimizing $\mathcal{L}_{ADV}^{f}$, the distance between the prediction and the ground truth will be maximized, which ensures the adversarial effectiveness.

To compute $\mathcal{L}_{ADV}^{f}$, we obtain the adversarial video slice X_{adv} by substituting the original roadside sign S_{orig} with the generated adversarial roadside sign S_{adv}. Note that the generated adversarial roadside sign S_{adv} is a rectangular image and the original roadside sign S_{orig} in the video slice may exhibit an arbitrary quadrilateral shape which could vary among different frames. We leverage a classical perspective mapping method [1] to resolve this mismatch. We first get the four coordinates of the original roadside sign S_{orig} within each frame, and then map the generated adversarial roadside sign S_{adv} onto the corresponding quadrilateral area inside each frame (details can be found in supplementary material).

The final objective of PhysGAN is expressed as:

$$\mathcal{L} = \mathcal{L}_{GAN} + \lambda \mathcal{L}_{ADV}^{f}, \quad (2)$$

where λ denotes a co-efficient to balance the tradeoff between the two terms and $\mathcal{L}_{GAN}$ is the classic GAN loss,

Algorithm 1 Optimization for PhysGAN

Require: I - Iteration numbers;
Require: f - Target model with fixed parameters;
1: **while** $i < I$: **do**
2: $S_{adv} = \mathcal{G}(\mathcal{E}(X_{orig}))$;
3: $\mathcal{L}_{GAN} = \log \mathcal{D}(S_{orig}) + \log(1 - \mathcal{D}(S_{adv}))$;
4: // fix the parameters of $\mathcal{G}$
5: do back-propagation to optimize $\arg\max_{\mathcal{D}} \mathcal{L}_{GAN}$;
6: $S_{adv} = \mathcal{G}(\mathcal{E}(X_{orig}))$
7: $\mathcal{L}_{GAN} = \log \mathcal{D}(S_{orig}) + \log(1 - \mathcal{D}(S_{adv}))$;
8: // fix the parameters of $\mathcal{D}$
9: for each frame in the input video slice, perform perspective mapping to substitute the original roadside sign S_{orig} using the adversarial roadside sign S_{adv}.
10: do back-propagation to optimize $\arg\min_{\mathcal{G}} \mathcal{L}_{GAN}$;
11: $\mathcal{L}_{ADV} = \beta \exp(-\frac{1}{\beta} \cdot l_f(f(X_{orig})))$;
12: do back-propagation to optimize $\arg\min_{\mathcal{G}} \mathcal{L}_{ADV}$;
13: **end while**

which can be represented as

$$\begin{aligned}\mathcal{L}_{GAN} = &\mathbb{E}_{S_{orig} \sim p_{S_{orig}}} [\log \mathcal{D}(S_{orig})] \\ &+ \mathbb{E}_{S_{adv} \sim p_{S_{adv}}} [\log(1 - \mathcal{D}(S_{adv}))] \ .\end{aligned} \quad (3)$$

To interpret this objective function, $\mathcal{L}_{GAN}$ encourages the adversarial roadside sign S_{adv} to be visually similar to the original roadside sign S_{orig}, while $\mathcal{L}_{ADV}^{f}$ is leveraged to generate adversarial video slice X_{adv} which maximizes attack effectiveness. We obtain the encoder $\mathcal{E}$, the generator $\mathcal{G}$, and the discriminator $\mathcal{D}$ by solving:

$$\arg\min_{\mathcal{G}} \max_{\mathcal{D}} \mathcal{L}. \quad (4)$$

3.5. Attacks with PhysGAN

We assume that the target autonomous driving model f was pre-trained and the parameters of target autonomous driving model f are fixed, and the generator $\mathcal{G}$ of PhysGAN can only access the parameters of the target autonomous driving model f during training. Our algorithm to train PhysGAN is illustrated in Algorithm 1, which consists of two phases. As seen in Algorithm 1, the first phase is to train the discriminator $\mathcal{D}$, which is used later to form a part of the $\mathcal{L}_{GAN}$ (Line 2 – 5); the second phase is to train generator $\mathcal{G}$ with two loss functions, $\mathcal{L}_{ADV}^{f}$ and $\mathcal{L}_{GAN}$, which encourages the generator $\mathcal{G}$ to generate a visually indistinguishable adversarial sample and make the generated sample be adversarial for the target autonomous driving model f, respectively (Line 6 – 11). The encoder $\mathcal{E}$, which is the CNN part of the target autonomous driving model f, aims at extracting features from all the observed frames during driving and transforms them into a vector input to the generator. This design ensures that the generator could gener-

ate visually indistinguishable examples with an attack effect through considering all useful features extracted from the video slice. For physical world deployment, the attacker shall print the adversarial example of the same size as the target roadside sign to ensure visual indistinguishability.

4. Experiments

We evaluate PhysGAN via both digital and physical-world evaluations using widely-studied CNN-based steering models and popular datasets.

4.1. Experiment Setup

Steering models. We evaluate PhysGAN on several popular and widely-studied [6, 28, 3] CNN-based steering models, like NVIDIA Dave-2 [2], Udacity Cg23 [3] and Udacity Rambo [4]. Notably, since the original models applies 2D CNNs which is trained with individual images, we adapt the 2D CNN into a 3D CNN, and train the 3D-CNNs with a set of 20-frame video slices.

Datasets. The datasets used in our digital experiments include (1) Udacity automatic driving car challenge dataset [5], which contains 101396 training images captured by a dashboard mounted camera of a driving car and the simultaneous steering wheel angle applied by a human driver for each image; (2) DAVE-2 testing dataset [20] [6], which contains 45,568 images to test the NVIDIA DAVE-2 model; (3) Kitti [9] dataset which contains 14,999 images from six different scenes captured by a VW Passat station wagon equipped with four video cameras; and (4) custom datasets for physical-world evaluation, which contain more than 20000 frames used to train PhysGAN in physical cases.

For physical-world experiments, we first perform color augmentation to improve the image contrast, making the adversarial example be more robust against varying light illumination conditions. Then, we print out the generated example under each evaluated approach, and paste it on the selected roadside object. We drive a vehicle by this object and perform offline analysis using the captured driving videos. To understand how PhysGAN would perform on actual autonomous vehicles, we have also done online driving testing which mimics a realistic driving process when facing with such an adversarial roadside object.

Video slice selection criteria. Our driving scene selection criteria is that the roadside traffic or advertising signs should appear entirely in the first frame of a driving video with more than 400 pixels and partially disappear in the last frame. We select 7 scenes from the aforementioned datasets, and evaluate on all selected scenes. The selected scenes in each dataset cover both straight and curved lane scenarios. Since all these datasets do not contain coordinates of roadside signs, we have to label the four corners of the signs in every frame of the selected scenes. We use the motion tracker functionality of Adobe After Effects [7] to automatically track the movement of the signs four corners among consecutive frames. Table 1 show the attributes of the scenes we selected.

Scenes	Images	Size	min	max
Dave-straight1	20	455×256	21×22	41×49
Dave-curve1	20	455×256	29×32	51×49
Udacity-straight1	20	640×480	48×29	66×35
Udacity-curve1	20	640×480	51×51	155×156
Kitti-straight1	20	455×1392	56×74	121×162
Kitti-straight2	20	455×1392	80×46	247×100
Kitti-curve1	20	455×1392	64×74	173×223

Table 1: Scenes evaluated in the experiment.

Baseline methods. We compare PhysGAN with several baseline approaches:

- *Original sign.* The first baseline is to simply test the original roadside sign. This comparison is important as it verifies whether steering angle errors are due to PhysGAN but not the original sign. We include this baseline in both digital and physical evaluations.
- *FGSM.* FGSM [11] is remarkably powerful and it is designed to attack neural networks by leveraging the gradients. In our problem context, we directly apply FGSM to generate perturbations given a captured input frame. We only include FGSM in our digital evaluation, as it is impossible to apply the generated perturbations which covers the entire image frame (*e.g.*, the sky) in physical world.
- *PhysFGSM.* In order to apply FGSM in a physical-world setting, we develop a new method called PhysFGSM as an additional baseline, which is based on FGSM and only generate perturbations for the targeted roadside sign in an input image. Doing so allows us to print the perturbed image and paste it onto the corresponding sign. We include PhysFGSM in both digital and physical evaluations. Since the video slice contains multiple frames, PhysFGSM generate perturbations based upon the middle frame.
- *RP2.* We also compare PhysGAN to a physical-world baseline, RP2 [8], which is an optimization approach that generated perturbations for a single input scene. The original RP2 method focuses on classification

[2] https://devblogs.nvidia.com/deep-learning-self-driving-cars/

[3] https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/cg23

[4] https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/rambo

[5] https://medium.com/udacity/challenge-2-using-deep-learning-to-predict-steering-angles-f42004a36ff3

[6] https://github.com/SullyChen/driving-datasets

[7] https://www.adobe.com/products/aftereffects.html

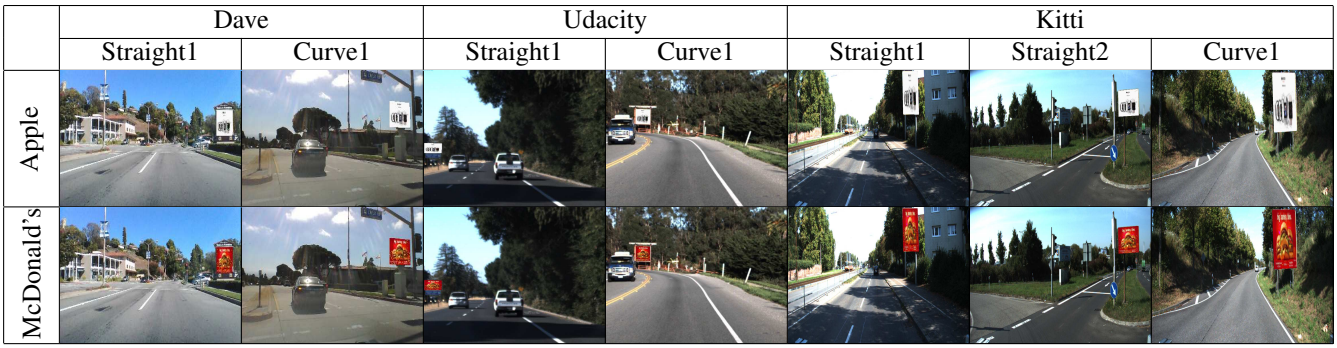
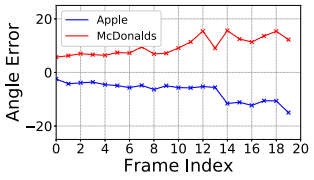
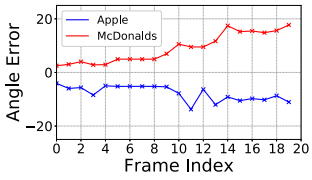


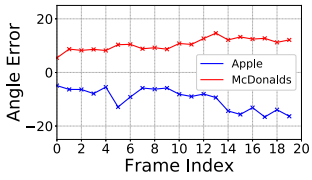
Table 2: The original and the generated adversarial fragments and the corresponding image frames under various scenes.



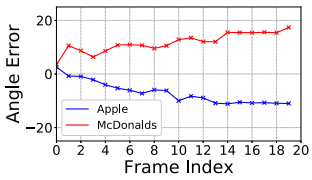
(a) Dave Straight1



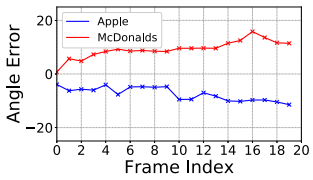
(b) Dave Curve1



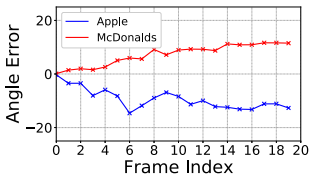
(c) Udacity Straight1



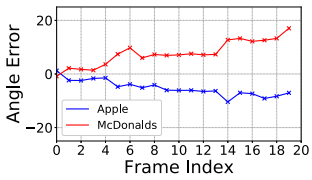
(d) Udacity Curve1



(e) Kitti Straight1



(f) Kitti Straight2



(g) Kitti Curve1

Figure 3: The illustration of Steering angle error variations along the timeline on steering model Nvidia Dave-2.

problems, so we extend it to be applicable to the steering module by substituting the classification loss with the regression loss.

- *Random Noise.* We also print an image perturbed with random noise and paste it on top of the roadside sign.

Evaluation metrics. In our experiments, we use two metrics to evaluate the efficacy of PhysGAN: steering angle mean square error (denoted *steering angle MSE*), and *max steering angle error* (MSAE). *Steering angle MSE* measures the average of the squares of the error between the predicted steering angle and the ground truth, and *MSAE* denotes the maximum steering angle error observed among all frames belonging to a video slice. A large *steering angle MSE* and *MSAE* implies better attack efficacy.

In addition, we perform online driving testing case studies where we manually control the steering angle in each frame (approximately) according to the real-time calculation of the resulting steering angle error under each evaluated approach. We use the metric *time-to-curb* herein to measure the attack efficacy, which measures the amount of time an actual autonomous driving vehicle would take to drive onto the roadside curb. Please be advised that all the results are relative to the ground truth steering angle.

4.2. Results

We first report the overall efficacy under PhysGAN in both digital and physical-world scenarios. A complete set of results is given in the supplementary document.

Results on digital scenarios. Table 2 shows a representative frame of each scene where the signs are replaced with adversarial examples generated from PhysGAN (using the targeted steering model NVIDIA Dave-2). Each column of Table 2 represents a specific scene. It is observed that PhysGAN can generate rather realistic adversarial samples, visually indistinguishable from the original objects. The targeted roadside signs in the original video slices are replaced by our selected McDonald and Apple Watch signs, and the modified video slices are used in all experiments. This is because the roadside signs in the original video slices have a low resolution, which makes it hard to verify whether our generated roadside signs are visually distinguishable.

Fig. 3 shows the results on steering angle error along the timeline in each frame scene, where the size of the adversarial image increases nearly monotonically over time. Each sub-figure in Fig. 3 indicates a specific scene, where the

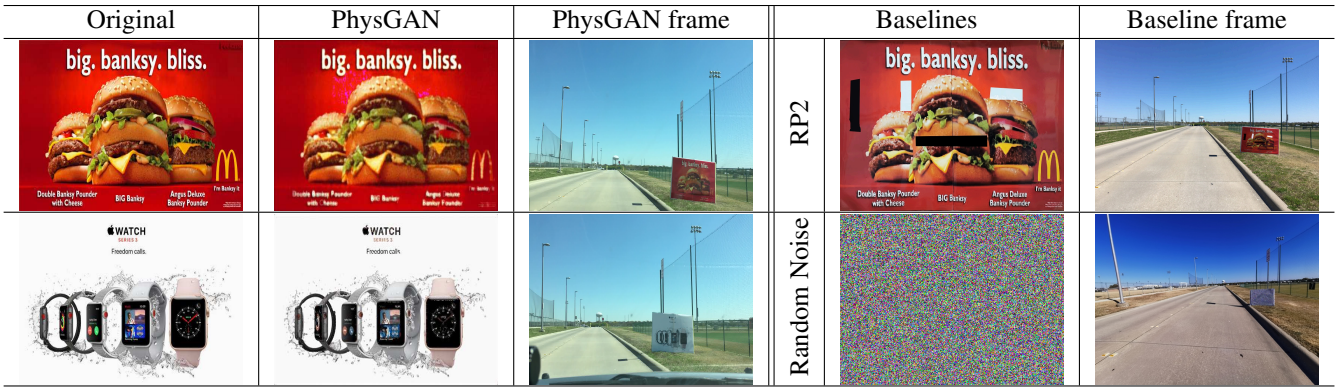


Table 3: Illustration of physical-world adversarial scenarios under different approaches.

Frame #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Original Apple Sign	0.36	-0.51	0.82	0.45	0.10	-0.16	0.84	-1.38	-2.16	-0.86	0.60	-1.11	0.21	-0.49	-0.55	-0.56	0.10	-0.51	0.49	-1.00
PhysGAN (Apple)	0.17	0.89	1.68	7.94	1.93	4.79	2.87	6.34	2.08	3.54	9.06	8.37	5.93	12.51	13.43	11.37	12.75	11.74	13.63	13.44
Original McDonald's Sign	-0.17	-0.42	-1.49	-1.34	-0.51	-0.08	0.60	-0.35	0.70	-0.75	-0.43	-0.35	0.59	-0.89	1.49	0.61	0.94	-0.99	1.13	-0.00
PhysGAN (McDonald's)	-1.24	-1.37	-0.02	-0.30	-2.48	-0.17	-1.06	-0.80	-0.01	-5.37	-1.60	-2.62	-2.45	-4.68	-11.71	-10.85	-9.83	-8.74	-11.35	-19.17

Table 4: Per-frame steering angle error under physical-world experiments. Rows 2 and 4 (rows 3 and 5) show the steering angle error when the original signs (corresponding adversarial signs generated by PhysGAN) are deployed.

x-axis represents the frame index along the timeline, and the y-axis represents the steering angle error. We clearly observe that PhysGAN leads to noticeable angle error for almost all frames, even for earlier frames in which the adversarial sample is relatively small compared to the background.

Results on physical-world scenarios. We perform physical-world experiments as follows. We first record training videos of driving a vehicle towards the original roadside sign and use these videos to train the Dave-2 model. We then train PhysGAN following the same configuration as the digital-world evaluation to generate adversarial samples. The generated adversarial samples was then printed and pasted on the original roadside sign. We then recorded testing videos of the same drive-by process but with the adversarial sample. The steering angle error are then obtained by analyzing these testing videos. Specifically, for both training and testing video slices, we start recording at 70 ft away and stop recording when the vehicle physically passes the advertising sign. For training videos, the driving speed is set to be $10mph$ to capture sufficient images. The speed for the testing video is set to be $20mph$ to reflect ordinary on-campus driving speed limit. The physical case studies are performed on a straight lane due to safety reasons. The size of the roadside advertising board used in our experiment is $48' \times 72'$.

Table 3 shows the original sign and the corresponding adversarial examples generated under different methods as well as a camera-captured scenes for each example. To clearly interpret the results, we list the per-frame steering angle error due to PhysGAN and using the original sign in Table 4 (additional comparison results are detailed in Sec. 4.3). As seen in Table 4, PhysGAN is able to generate a single printable physical-world-resilient adversarial example which could mislead the driving model for continuous frames during the entire driving process. An interesting observation herein is that the steering angle error tends to increase along with the increased frame index. This is because, with a larger frame index, the size of the adversarial sample occupies a relatively large space in the entire frame, so being able to more negatively impact the steering model. Also, we observe that with the original roadside sign, the steering angle error is almost negligible under all frames.

4.3. Comparison against Baseline Approaches

Digital Baselines. For each steering model, we compare our approach with four other baselines including FGSM, PhysFGSM, random noise, and original sign. Table 5 shows the results on seven different scenes. These results suggest the following observations: (1) although FGSM achieves the highest attacking effect, it needs to apply perturbations to the entire scene, which is not applicable to the physical world; (2) the attacking effectiveness of our approach is much better than PhysFGSM, implying that once considering physical world implementation constraints, PhysGAN would outperform direct extensions of existing approaches. (3) each steering model is reasonably robust as the angle errors under random noise and original sign are trivial.

Physical Baselines. For physical-world scenarios, we compare PhysGAN against PhysFGSM, random noise, and original sign. The results are shown in Table 6. We observe that both random noise and original sign have negligible im-

Steering Model	Approach	Dave		Udacity		Kitti		
		Straight1	Curve1	Straight1	Curve1	Straight1	Straight2	Curve1
Nvidia Dave-2	PhysGAN	106.69 / 15.61	66.79 / 11.63	81.76 / 17.04	114.13 / 14.64	108.76 / 17.72	150.00 / 17.34	95.87 / 15.83
	FGSM	115.91 / 17.41	199.27 / 19.47	141.23 / 16.17	192.19 / 21.23	156.16 / 17.84	217.52 / 19.50	103.38 / 14.54
	PhysFGSM	15.88 / 6.42	4.73 / 4.87	13.91 / 5.74	3.08 / 2.89	15.17 / 8.04	8.67 / 4.54	13.12 / 7.24
	Random Noise	3.00 / 2.01	2.25 / 2.37	2.36 / 2.60	1.77 / 3.10	3.15 / 3.16	1.60 / 0.96	5.92 / 4.41
	Original Sign	4.17 / 3.15	4.35 / 2.40	3.84 / 1.79	1.09 / 0.72	4.20 / 2.98	3.06 / 1.23	2.86 / 1.30
Udacity Cg23	PhysGAN	91.85 / 13.80	113.41 / 14.78	50.61 / 10.43	78.56 / 15.46	46.53 / 11.72	62.64 / 11.64	71.09 / 18.14
	FGSM	203.34 / 19.70	157.98 / 14.67	171.92 / 19.89	96.74 / 17.75	136.08 / 14.00	162.35 / 18.53	89.75 / 16.71
	PhysFGSM	58.53 / 11.86	36.44 / 10.68	30.72 / 9.41	46.74 / 8.88	28.89 / 11.37	22.63 / 7.61	61.23 / 10.95
	Random Noise	5.32 / 3.67	3.75 / 2.72	4.05 / 2.52	4.20 / 2.26	5.31 / 4.49	6.54 / 1.98	6.10 / 3.68
	Original Sign	4.17 / 3.15	4.35 / 2.40	3.84 / 1.79	4.09 / 2.72	4.20 / 2.98	3.06 / 1.23	2.30 / 1.86
Udacity Rambo	PhysGAN	61.87 / 11.28	113.78 / 15.29	87.68 / 13.90	42.71 / 12.55	56.41 / 12.42	58.67 / 10.42	145.66 / 21.85
	FGSM	209.81 / 21.78	147.28 / 16.43	151.14 / 15.28	166.50 / 16.27	169.17 / 18.57	126.14 / 14.19	175.28 / 19.36
	PhysFGSM	16.43 / 8.95	14.24 / 8.34	5.32 / 3.73	14.82 / 6.11	16.58 / 7.78	13.89 / 7.93	29.58 / 19.18
	Random Noise	1.90 / 2.55	3.49 / 5.79	6.06 / 5.00	1.92 / 3.98	3.82 / 5.42	2.09 / 3.05	1.52 / 1.91
	Original Sign	3.93 / 2.01	6.30 / 4.46	1.80 / 1.28	6.54 / 2.52	5.06 / 3.52	5.75 / 4.03	4.95 / 2.07

Table 5: *Steering angle MSE* (left) and *MSAE* (right) under all evaluated approaches. Although FGSM produces the maximal attacks, it modifies the whole image observation and is not applicable to the real world. Among all physical-world attack approaches, our approach PhysGAN produces the best performance.

	PhysGAN	RP2	Random Noise	Original Sign
Nvidia Dave-2	73.94 / 13.63	23.48 / 6.52	2.48 / 1.02	2.12 / 1.56
Udacity Cg23	99.23 / 14.56	25.15 / 7.86	2.56 / 2.11	2.15 / 1.73
Udacity Rambo	87.56 / 17.60	32.54 / 7.51	1.51 / 1.15	3.12 / 2.48

Table 6: *Steering angle MSE* (left) and *MSAE* (right) under PhysGAN, RP2, random noise, and original sign.

pact on the steering models, which indicate the pre-trained steering models (without being attacked) are sufficiently robust in physical world settings. As seen in Table 6, PhysGAN significantly outperforms RP2 and can achieve very high attack efficacy under all steering models, which may lead to dangerous driving actions in the real world.

	PhysGAN	RP2	Random Noise	Original
Time-to-curb	$10s$	-	-	-
Distance-to-center	1.5m	1.09m	0.29m	0.47m

Table 7: Online driving testing results. The second row shows the time-to-curb result and the third row shows the maximum distance that the vehicle deviates from the correct path (*i.e.*, driving straight).

4.4. Online Driving Case Studies

The above evaluations are off-policy where the driving trajectory was not affected by the adversarial signs. In this section, we further conduct on-policy evaluation, *i.e.*, online driving case studies mimicking the actual driving scenario to learn how would PhysGAN impact the steering decision by an actual autonomous vehicle. In these case studies, we manually control steering in a real-time fashion within each frame according to the calculated steering angle error under each approach with the steering model Nvidia Dave-2. We ask a human driver to drive the vehicle at $5mph$ for 1 second to reflect one frame and a corresponding manual steering action. We note that this online evaluation setup is a proxy of real autonomous vehicle and provides proper evaluation of an attack system. We do not use virtual simulators for evaluation because they normally causes sim-to-real transfer issues. So the evaluation results on a simulator would not reflect the models capability in the physical world. As seen in Table 7, PhysGAN outperforms the other baselines under the two metrics. Also, only the adversarial sign generated under PhysGAN leads the vehicle to drive onto the roadside curb, which takes 10s (given the very low driving speed due to safety concerns). This online driving case study further demonstrates the dangerous steering action an autonomous vehicle would take due to PhysGAN, indicating its effectiveness when applied to actual autonomous vehicles.

5. Conclusion

We present PhysGAN, which generates physical-world-resilient adversarial examples for misleading autonomous steering systems. We proposed a novel GAN-based framework for generating a single adversarial example that continuously misleads the driving model during the entire trajectory. The generated adversarial example is visually indistinguishable from the original roadside object. Extensive digital and physical-world experiments show the efficacy and robustness of PhysGAN. We hope our work could inspire future research on safe and robust machine learning for autonomous driving.

References

[1] Perspectives mapping, https://www.geometrictools.com/Documentation/PerspectiveMappings.pdf. 4

[2] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017. 1

[3] Alberto Broggi, Michele Buzzoni, Stefano Debattisti, Paolo Grisleri, Maria Chiara Laghi, Paolo Medici, and Pietro Versari. Extensive tests of autonomous driving technologies. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1403–1415, 2013. 5

[4] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017. 1, 2

[5] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017. 2

[6] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015. 5

[7] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. In *Advances in Neural Information Processing Systems*, pages 3914–3924, 2018. 1

[8] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pages 1625–1634, 2018. 1, 2, 5

[9] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 5

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680. 2014. 2

[11] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015. 2, 5

[12] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016. 1

[13] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *ICLR*, 2017. 2

[14] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *ICLR*, 2017. 2

[15] Jiajun Lu, Hussein Sibai, Evan Fabry, and David Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. *arXiv preprint arXiv:1707.03501*, 2017. 2

[16] Yongyi Lu, Yu-Wing Tai, and Chi-Keung Tang. Conditional cyclegan for attribute guided face image generation. *arXiv preprint arXiv:1705.09966*, 2017. 2

[17] Jan Hendrik Metzen, Mummadi Chaithanya Kumar, Thomas Brox, and Volker Fischer. Universal adversarial perturbations against semantic image segmentation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2774–2783. IEEE, 2017. 1

[18] Andreas Mogelmose, Mohan Manubhai Trivedi, and Thomas B Moeslund. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1484–1497, 2012. 1

[19] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, pages 2574–2582, 2016. 1

[20] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017. 5

[21] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017. 2

[22] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016. 1

[23] Pierre Sermanet and Yann LeCun. Traffic sign recognition with multi-scale convolutional networks. In *IJCNN*, pages 2809–2813, 2011. 1

[24] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. ACM, 2016. 1

[25] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 1

[26] Thomas Tanay and Lewis Griffin. A boundary tilting persepective on the phenomenon of adversarial examples. *arXiv preprint arXiv:1608.07690*, 2016. 2

[27] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *ICSE*, pages 303–314. ACM, 2018. 1

[28] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314. ACM, 2018. 5

[29] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. *IJCAI*, 2018. 2

[30] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, pages 597–613. Springer, 2016. 2

Diagnosing Vehicles with Automotive Batteries

Liang He
University of Colorado Denver

Linghe Kong
Shanghai Jiaotong University

Ziyang Liu
Shanghai Jiaotong University

Yuanchao Shu
Microsoft Research

Cong Liu
University of Texas at Dallas

ABSTRACT

The automotive industry is increasingly employing software-based solutions to provide value-added features on vehicles, especially with the coming era of electric vehicles and autonomous driving. The ever-increasing cyber components of vehicles (i.e., computation, communication, and control), however, incur new risks of anomalies, as demonstrated by the millions of vehicles recalled by different manufactures. To mitigate these risks, we design `B-Diag`, a battery-based diagnostics system that guards vehicles against anomalies with a cyber-physical approach, and implement `B-Diag` as an add-on module of commodity vehicles attached to automotive batteries, thus providing vehicles an additional layer of protection. `B-Diag` is inspired by the fact that the automotive battery operates in strong dependency with many physical components of the vehicle, which is observable as correlations between battery voltage and the vehicle's corresponding operational parameters, e.g., a faster revolutions-per-minute (RPM) of the engine, in general, leads to a higher battery voltage. `B-Diag` exploits such physically-induced correlations to diagnose vehicles by cross-validating the vehicle information with battery voltage, based on a set of data-driven norm models constructed online. Such a design of `B-Diag` is steered by a dataset collected with a prototype system when driving a 2018 Subaru Crosstrek in real-life over 3 months, covering a total mileage of about 1, 400 miles. Besides the Crosstrek, we have also evaluated `B-Diag` with driving traces of a 2008 Honda Fit, a 2018 Volvo XC60, and a 2017 Volkswagen Passat, showing `B-Diag` detects vehicle anomalies with >86% (up to 99%) averaged detection rate.

CCS CONCEPTS

• **Computer systems organization → Sensors and actuators**.

MobiCom '19, October 21–25, 2019, Los Cabos, Mexico

ACM ISBN 978-1-4503-6169-9/19/10…$15.00
https://doi.org/10.1145/3300061.3300126

KEYWORDS

vehicle diagnostics; batteries-as-sensors; CPSes

ACM Reference Format:
Liang He, Linghe Kong, Ziyang Liu, Yuanchao Shu, and Cong Liu. 2019. Diagnosing Vehicles with Automotive Batteries. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19), October 21–25, 2019, Los Cabos, Mexico.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3300061.3300126

1 INTRODUCTION

• **Background.** The automotive industry is increasingly employing software-based solutions to provide value-added features on vehicles, such as automatic crash response and remote diagnostics, especially with the coming era of (hybrid) electric vehicles and autonomous driving. As a result, modern vehicles are commonly installed with software systems consisting of hundreds of millions of lines of codes distributed across over 80 Electronic Control Units (ECUs) [1, 2], rendering vehicles prototypical cyber-physical systems (CPSes). The ever-increasing cyber components of vehicles, however, prove to be a double-edged sword and incur new risks to vehicles' reliability/safety [3–5].

First, software system becomes error-prone with the ever growing data volume in the in-vehicle network [6–8]. Taking the automatic gear shifting in Fig. 1 as an example:

(a) the vehicle's engine control module first gathers readings of the crankshaft position sensor to calculate the RPM[1],
(b) it then actuates based on the thus-calculated RPM to control the activation frequency of spark plug,
(c) the engine control module also broadcasts the RPM to other ECUs via the in-vehicle network, e.g., in form of the controller area network (CAN) [9],
(d) the transmission control module receives and then processes the broadcasted RPM, and changes gears accordingly.

As can be seen, any software defects in the computation/communication/control of the above process could compromise the gear shifting. Software flaws, unfortunately, have been frequently identified in vehicles: (i) a bug causing unintended

[1]Revolutions per minute (RPM) quantifies the engine speed.

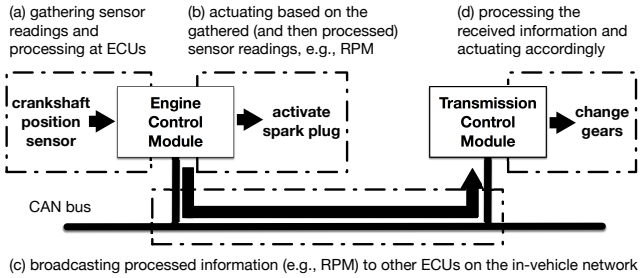


Fig. 1: Vehicle's cyber operations: gathering, processing, transmitting information and actuating accordingly.

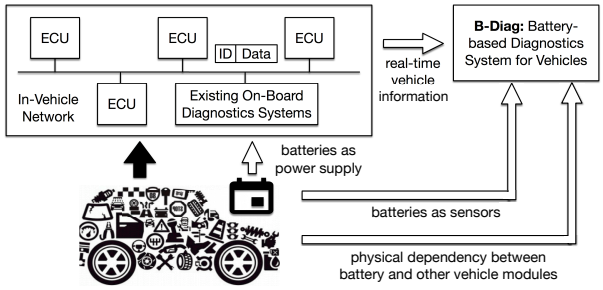


Fig. 2: `B-Diag` diagnoses vehicles with a cyber-physical approach by exploiting automotive batteries as sensors.

acceleration forced Toyota to recall 7.5 million vehicles between 2009–2011 [10], (ii) a glitch unlocking the door without notifying drivers caused Jaguar to recall 65,000 Range Rover in 2015 [11], (iii) defects in the cruise control software caused Chrysler to recall 4.8 million vehicles in 2018 [12], to name a few.

Second, the proliferation of in-vehicle sensing and communication modules eases the inter-connection between cars and third-party devices, thus exposing new vulnerabilities to cyber attacks [13–19]. For example, many work on the injection and modification of data packets in the in-vehicle network through WiFi, Bluetooth, or other cyber interfaces have been reported [20–24]. People have even successfully stopped a Jeep Cherokee on a highway by masquerading its in-vehicle data packets [25], triggering a recall of 1.4 million vehicles by Jeep in 2015 [26].

These risks, albeit of different causes, lead to the same consequence of *unintended information in the in-vehicle network*, referred to as *cyber anomalies*, disrupting the automotive industry and degrading vehicles' reliability/safety.

• **State-of-the-Art.** Vehicle anomalies are traditionally diagnosed with the On-Board Diagnostics System (OBD-II) [27], which however, is ineffective in detecting cyber-induced anomalies, as demonstrated by the fact that many of the above cyber flaws/attacks do not trigger any diagnostic trouble code of OBD-II. To fill this need of anomaly diagnostics, researchers have designed various solutions such as message authentication [28–32] and intrusion detection [33–35]. These solutions, however, still suffer from the following three deficiencies. First, these solutions are defective in systematically exploiting a vehicle's CPS nature [36] — i.e., a system of sub-systems interacted constantly in the cyber and physical spaces — missing a reliable opportunity in vehicle diagnostics, as we will see in this work. Second, these solutions are commonly implemented at vehicles' ECUs as part of the in-vehicle network, and thus also suffer from the risks of anomalies thereof, i.e., the diagnostics systems themselves could be abnormal [31, 37, 38]. Last but not the least, many existing solutions are grounded on an offline knowledge of known vehicle anomalies, thus being defective in adapting to unexpected but inevitable vehicle dynamics [39–43].

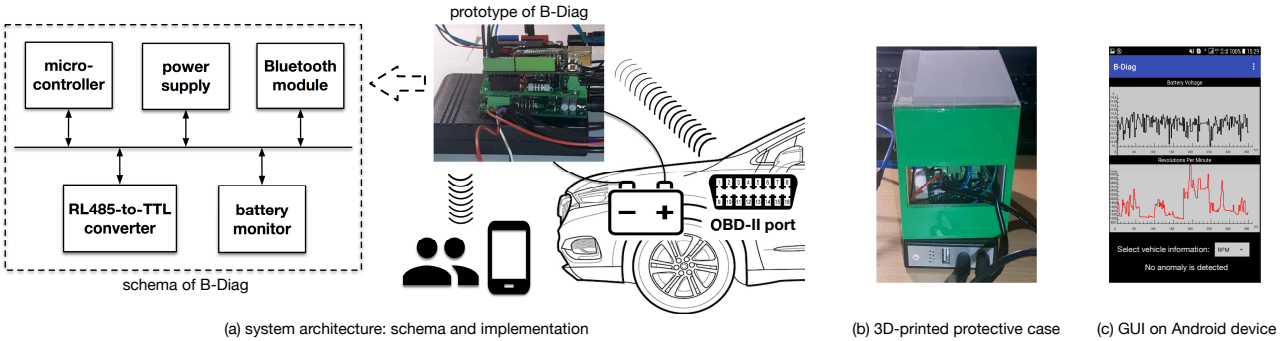
Table 1: A (nonexclusive) list of vehicle information that are corroborated to be diagnosable by `B-Diag`.

Vehicle Information	
1) Absolute Throttle Position B	12) Intake Manifold Pressure
2) Accelerator PedalPosition D	13) Mass Air Flow Rate
3) Accelerator PedalPosition E	14) O2 Sensor1 Equivalence Ratio
4) Air Fuel Ratio (Commanded)	15) O2 Sensor1 Equivalence Ratio (alternate)
5) Air Fuel Ratio (Measured)	16) O2 Sensor1 Wide-Range Voltage
6) Commanded Equivalence Ratio	17) Throttle Position (Manifold)
7) Engine Coolant Temperature	18) Transmission Temperature (Method 1)
8) Engine Load	19) Transmission Temperature (Method 3)
9) Engine Load (Absolute)	20) Voltage (Control Module)
10) Engine RPM	21) Voltage (OBD Adapter)
11) Fuel Level (From Engine ECU)	22) Volumetric Efficiency (Calculated)

• **Battery-based Diagnostics of Vehicles.** To mitigate these deficiencies, we design a battery-based diagnostic system for vehicles, called `B-Diag`, and implement `B-Diag` as an add-on module of commodity vehicles attached to automotive batteries, thus providing vehicles an additional protection on top of the traditional OBD-II (see Fig. 2). `B-Diag` has the following salient properties.

(1) Diagnosing with a Cyber-Physical Approach. The foundation of `B-Diag` is the fact that many physically inter-connected modules of the vehicle operate in close dependency — e.g., a faster engine RPM increases the alternator's output current and then the automotive battery's voltage — which is observable as correlations among the vehicle's operational parameters in the cyber space. `B-Diag` exploits such correlations to diagnose vehicles with a cyber-physical approach by: (i) capturing the physically-induced correlations in the cyber space with data-driven norm models constructed online, and (ii) detecting (and then verify) vehicle anomalies by cross-validating, in real-time, the vehicle information. These online constructed norm models also make `B-Diag` adaptive to the inevitable changes in vehicles.

(2) Exploiting Batteries as Sensors. `B-Diag`'s cross-validation of vehicle information requires a trustworthy ground, to which no information from the in-vehicle network satisfies due to the risks of cyber-induced anomalies. As a mitigation, `B-Diag` novelly grounds its cross-validation on the voltage of automotive batteries by exploiting batteries as sensors: (i) battery voltage can be reliably (and easily)



(a) system architecture: schema and implementation (b) 3D-printed protective case (c) GUI on Android device

Fig. 3: We have prototyped B-Diag as an add-on module of commodity vehicles attached to automotive batteries.

(a) 2018 Subaru Crosstrek (b) attach the prototype to battery (c) OBD-II adapters (d) OBD-II port in vehicle (e) summary of traces

Fig. 4: Using B-Diag's prototype to collect the real-life driving data of a 2018 Subaru Crosstrek.

collected from the physical batteries without going through the in-vehicle network, thus serving as the *hardware-based root of trust* [44] and making the cross-validation *reliable*; (ii) battery operates in strong dependency with many vehicle modules, and thus battery voltage correlates with many vehicle parameters, making the cross-validation *effective*. This way, B-Diag's anomaly detection will be robust and effective even when the in-vehicle network is compromised. Such a battery-based diagnostics of B-Diag also magnifies its practicality because no re-designing of existing in-vehicle network is needed, which is crucial for the cost-conscious automotive industry with only 4–9% profit margin [45]. Table 1 summarizes the vehicle information that has been corroborated to be diagnosable by B-Diag.[2]

The design of B-Diag is steered by a dataset collected with a prototype system when driving a 2018 Subaru Crosstrek in real-life over 3 months, covering a total mileage of about 1,400 miles. Besides the Crosstrek, we have also evaluated B-Diag using driving traces collected from a 2008 Honda Fit, a 2018 Volvo XC60, and a 2017 Volkswagen Passat, showing B-Diag detects anomalies in vehicle information with >86% (up to 99%) detection rate on average. In this paper, we use B-Diag's diagnosis of engine RPM as a complete walk-through example, and then validate B-Diag's ability of individually diagnosing the vehicle information listed in Table 1. An integrated approach to diagnose all vehicle information in real-time, however, is still missing in this work.

2 SYSTEM PROTOTYPING

We have prototyped B-Diag as an add-on module of commodity vehicles attached to their automotive batteries, as shown in Fig. 3(a), including: (i) an Arduino-based micro-controller attached to the automotive battery in the vehicle's engine cabin, (ii) a battery monitor collecting the voltage of the automotive battery in real-time, (iii) a RS485-to-TTL converter transforming the voltage signal and sending it to the micro-controller, (iv) a Bluetooth module collecting the vehicle information from the in-vehicle network — e.g., via the OBD-II port with off-the-shelf OBD-II adapters — and reporting the results to the smartphone of the vehicle's driver/owner, and (v) a power supply supporting the above components. This prototype is installed in a 3D-printed protective case, as shown in Fig. 3(b). Fig. 3(c) shows an example of the prototype's GUI on an Android phone. Note the Bluetooth-based collection of vehicle information from the OBD-II port is only for the ease of implementation. Wired OBD-II adapters are readily available in the literature and could be adopted to further improve the reliability. The total hardware cost of this prototype is below US$50, which could be further reduced, e.g., by using the automotive battery to power the prototype and thus removing the power supply.

We have used this prototype of B-Diag to collect the real-life driving traces of a 2018 Subaru Crosstrek, as shown in Figs. 4(a) and (b). We used four commodity Bluetooth OBD-II adapters (Fig. 4(c)) to collect the vehicle information via the OBD-II port at 10Hz (Fig. 4(d)) and upload the information to B-Diag's prototype. These data are collected over 3 months, on both highway and urban roads, and also when driving during rush hour traffic jams and in snowing/raining weather, as summarized in Fig. 4(e). These data cover most of activities during driving such as turning, breaking, cruise control, operating the vehicle's e-systems such as air con and radio, etc. No abnormal behavior of the vehicle is observed during the collection of these traces, which is also confirmed

[2]Please see [46] for the details of these vehicle information.

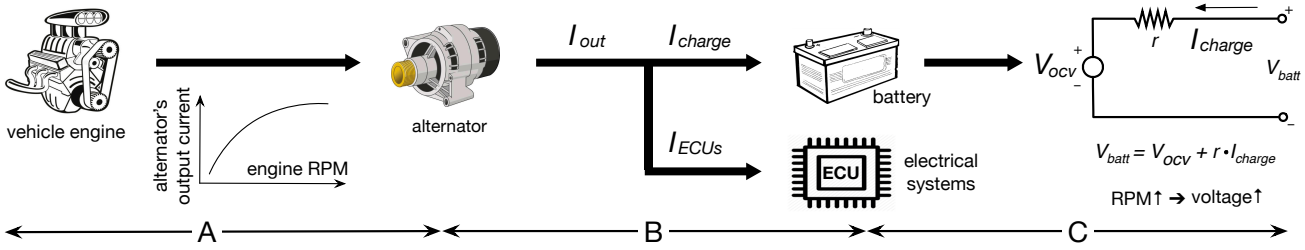


Fig. 5: Other things being equal, a faster engine RPM leads to a higher battery voltage.

when performing its regular maintenance at the auto dealer. This way, we treat these collected traces as normal. We have also identified another Bluetooth OBD-II adapter (not among the four adapters in Fig. 4(c)) that is not reliable in collecting the driving traces. We will use the "abnormal" vehicle information collected with this faulty adapter to evaluate `B-Diag`, as we explain in Sec. 5.1.

Note that such an installment of `B-Diag` is general for all vehicles because (i) all automotive batteries have positive/negative terminals exposed to the environment, via which `B-Diag` can be connected and thus the battery voltage collected; (ii) OBD-II port — under the dash in virtually all modern vehicles — has been mandatory for all vehicles sold in the US since 1996 and Europe since 2001, via which the well-defined vehicle information can be collected in real-time without knowing the vehicle architecture or the format of the in-vehicle messages.

Knowing the hardware components of `B-Diag`, we explain `B-Diag`'s diagnostic algorithms in the next two sections, steered by the above empirically collected driving traces.

3 CASE-STUDY: DIAGNOSING ENGINE RPM WITH BATTERY

We first use `B-Diag`'s detection of anomalies in engine RPM as an example to walk through its diagnostics of vehicles. The related algorithms are also applicable to the detection of anomalies in other vehicle information listed in Table 1, as we elaborate in Sec. 4.

3.1 Automotive Battery and Engine

We first explain the physically-induced correlations between the automotive battery and vehicle's engine.

• **Automotive Battery.** Automotive battery — normally a rechargeable lead-acid battery with 12/24V nominal voltage depending on vehicle type — supplies the necessary current to the starter motor and the ignition system while cranking to start the engine. The battery will be charged by the vehicle's alternator once the engine is running. It is crucial to note that even (hybrid) electric vehicles such as Chevrolet Volt and Bolt — which use high-voltage (e.g., up to 400V) battery packs to supply the driving power — have such low-voltage batteries, ensuring their compatibility to standard 12/24V automotive accessories.

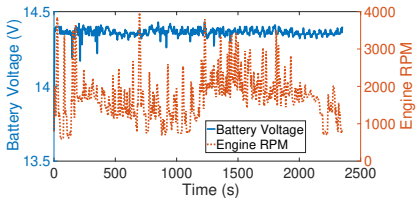
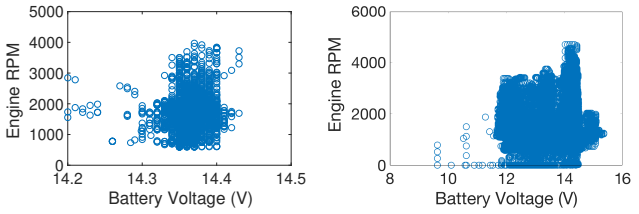


Fig. 6: Exemplary traces of battery voltage and engine RPM.

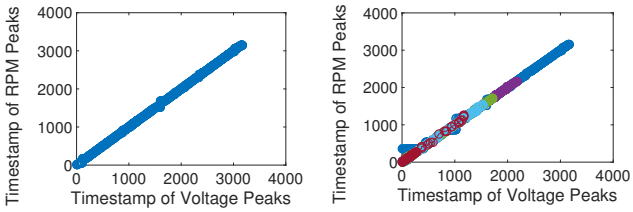
• **Engine RPM.** Revolutions per minute (RPM) is the metric quantifying the engine speed, defined as the number of rotations per minute made by engine's crankshaft and monitored by the crankshaft position sensor in real-time. RPM is crucial to engine's timing functions for ignition, fuel injection, spark events, and valve timing. For example, RPM is needed to determine the activation frequency of spark plugs, normally every 10–17ms [36], to control the fuel injection to each cylinder in real-time. As a result, inaccurate RPMs cause a variety of problems such as misfiring, motor vibration, backfires, hesitant acceleration, abnormal shaking, or, the car may simply do not start [47, 48]. Koscher et al. has experimentally demonstrated the feasibility of fabricating engine RPM via cyber attacks [22], rendering the abnormal RPM a real-life risk. For example, fabricating a large RPM to a low level could falsely convince the Power Steering Control Module (PSCM) that the vehicle is driving slowly, thus tricking PSCM to start a diagnostic session even when driving on a highway, causing critical safety risk [24].

• **Physical Dependency betw. Battery and Engine.** The automotive battery and engine operate in close dependency, as summarized in Fig. 5. First, the engine's rotation triggers that of the alternator at a speed about 1–3 times of engine RPM [49]. The alternator's rotation, in turn, generates an electric power that is monotonic to its rotation speed (up to a certain safe level). This way, a faster RPM leads to a larger output current I_{out} of the alternator (see part-A of Fig. 5) [50]. Second, part of the alternator's I_{out} is used to power the vehicle's electrical systems, and the remaining current charges the battery. Other things being equal, a larger I_{out} increases the battery's charging current (see part-B of Fig. 5). Third, a larger charging current increases the battery voltage. This can be explained by the battery's circuit model shown in Part-C of Fig. 5: the battery will have a voltage of $V_{\text{batt}} = V_{\text{ocv}} + r \cdot I_{\text{charge}}$ when charging with current I_{charge} [51], where r is the internal resistance of the battery.



(a) with the trace in Fig. 6 (b) with all traces in Fig. 4(e)

Fig. 7: Weakly correlated battery voltage and RPM.



(a) with the trace in Fig. 6 (b) with all traces in Fig. 4(e)

Fig. 8: Strongly correlated peaks of battery voltage and RPM.

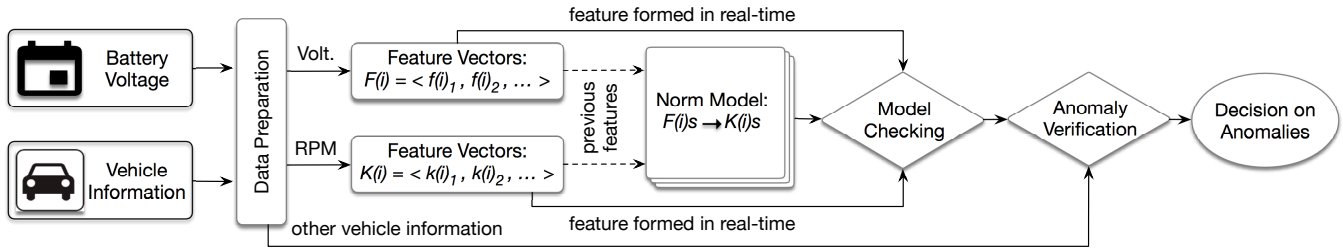


Fig. 9: B-Diag detects (and verifies) the anomalies in engine RPM based on battery voltage with an online data-driven model.

Combining these three facts uncovers a dependency between the automotive battery and engine induced by their physical design/connection — *the battery will has a higher voltage with a faster engine speed.*

• **Correlation betw. Battery Voltage and RPM.** We next examine if such a dependency between battery and engine could be observed as a correlation between the vehicle's corresponding operational parameters.

Observation-I: Weakly Correlated Raw Readings. Fig. 6 plots the traces of battery voltage and engine RPM during a 39-minute driving trip, and Fig. 7(a) shows the corresponding scatter plot, confirming their dependency that a larger RPM, in general, leads to a higher voltage. Significant variance, however, is observed: the battery voltage varies in a wide range of [14.2,14.4]V when the RPM is about 2,000, rendering such a correlation weak. Also, the two traces have only a small Pearson correlation coefficient of 0.06. Fig. 7(b) plots all the voltages and RPMs of the 64 traces summarized in Fig. 4(e), confirming again such *weakly correlated raw readings of battery voltage and engine RPM.* A potential explanation for such a weak correlation is that the battery voltage is affected by, besides the engine RPM, a variety of other factors, such as the power requirements of the vehicle's electrical systems (i.e., I_{ECUs} in Fig. 5), and thus rendering the battery voltage highly dynamic [52].

Observation-II: Strongly Correlated Peaks. We further identify the local maximums of the RPM/voltage readings in Fig. 6, referred to as *peaks*, and then use *dynamic time warping* [53] to align these peaks' time-stamps, as shown in Fig. 8(a). The close-to-diagonal warp path indicates we can find a voltage peak at a similar time whenever an RPM peak is observed, i.e., the peaks of battery voltage and engine RPM are synchronized (and hence correlated). Fig. 8(b) plots the warp paths obtained by aligning the RPM/voltage peaks of all the 64 Crosstrek traces in Fig. 4(e), validating again such *strongly correlated RPM/voltage peaks.* Note that here the correlation between RPM/voltage peaks is in a general sense and not necessarily in terms of the Pearson correlation.

B-Diag exploits the above two correlations between the battery voltage and engine RPM to detect the potential anomalies in RPM readings, as we explain next.

3.2 Detecting RPM Anomalies with Battery

Fig. 9 shows an overview of B-Diag's detection of potential anomalies in engine RPM: taking as input (i) the real-time battery voltage collected from the battery directly and (ii) the engine RPM from the in-vehicle network, B-Diag outputs an online decision value indicating if anomalies are detected in RPM readings. B-Diag conducts such an anomaly detection with three steps: data preparation, norm model construction, and anomaly detection. In what follows, we elaborate on the design of B-Diag using the trace shown in Fig. 6.

• **Data Preparation.** B-Diag applies a set of operations to prepare the collected battery voltage and engine RPM before constructing the norm model.

Data Alignment. B-Diag collects battery information from the battery and vehicle information from the in-vehicle network. Such different approaches of data collection make the collected battery voltage and engine RPM (likely) not aligned in the time domain. B-Diag aligns the data by exploiting the engine's cranking time as the anchor, which can be reliably identified based on the fact that both the battery voltage and engine RPM (i) keep stable before cranking and then (ii) change abruptly and significantly while cranking,

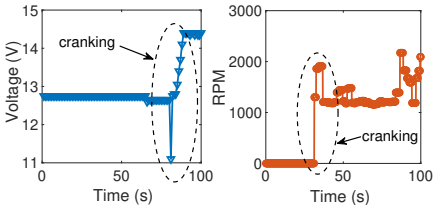


Fig. 10: `B-Diag` **aligns the battery voltage and the engine RPM based on engine's cranking time.**

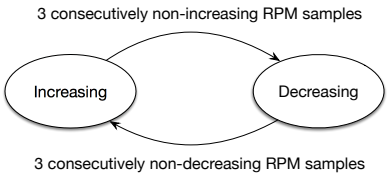


Fig. 11: `B-Diag` **confirms a transition in the trend of RPMs only when observing such a transition with three consecutive RPM samples.**

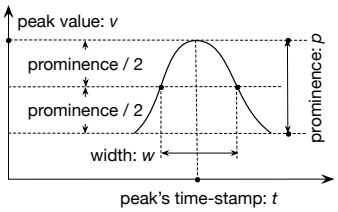


Fig. 12: `B-Diag` **describes a peak with its peak** v**, time-stamp** t**, width** w**, and prominence** p**.**

as shown in Fig. 10. `B-Diag` detects the cranking time by identifying the local min/maximums of voltage/RPM with significant magnitudes and are proceeded by stable readings.

Real-Time Peak Detection. `B-Diag`, besides collecting and recording the battery voltage and engine RPM, also checks the RPM to determine, in real-time, if a new RPM peak is observed, and triggers its diagnostics of potential anomalies if yes. Specifically, `B-Diag` identifies and maintains the current *trend* of RPMs as *increasing* or *decreasing* — a peak is detected if the *trend* changes from *increasing* to *decreasing*. `B-Diag` confirms such a change in *trend* only when it has been observed with three consecutive RPM samples, as illustrated in Fig. 11, reducing the variance in the peak detection caused due to signal dynamics.

Time Window Construction. `B-Diag` maintains the time at which the previous RPM peak is observed, denoted as t_{pre}. Once detecting a new RPM peak, `B-Diag` constructs a time window of $[t_{\text{pre}}-T_w, t_{\text{pre}}]$, where T_w is the window size. The window terminates at t_{pre}, instead of the current time t_{now}, because not all properties of the newly detected RPM peak can be determined at t_{now}, as we explain later.

Peak Identification in Time Window. `B-Diag` fetches the two time-series of battery voltage and engine RPM within the above-constructed time window, and then identifies the peaks therein. `B-Diag` describes each peak by its peak value v, width w, prominence p, and time-stamp t, as illustrated in Fig. 12, i.e., peak $= \{v, w, p, t\}$. `B-Diag` stores the peaks in the current time window to facilitate identification of peaks in the next window, exploiting their (likely) overlapping. Also, `B-Diag` discards the 10% of peaks (in both voltage and RPM) with the least prominence in the window, further improving its tolerance to the inherent dynamics of voltage and RPM readings. Note that neither the width or the prominence of a peak can be determined upon its detection. This is why the time window ends at the time of previous peak.

• **Norm Model Construction.** `B-Diag` constructs an online norm model capturing the relationship between the battery voltage and engine RPM, based on the two correlations observed in Sec. 3.1. Instead of manually constructing rules that map the battery voltage to RPM, we opted to use a machine learning-based classifier to increase the accuracy of `B-Diag`'s mapping. Specifically, `B-Diag` abstracts each of the two sub-traces of battery voltage and RPM in the time window to a 10-parameter feature vector, i.e.,

$$F = \{f_1, f_2, \cdots f_{10}\} \text{ for battery voltage,}$$
$$G = \{g_i, g_2, \cdots g_{10}\} \text{ for engine RPM,}$$

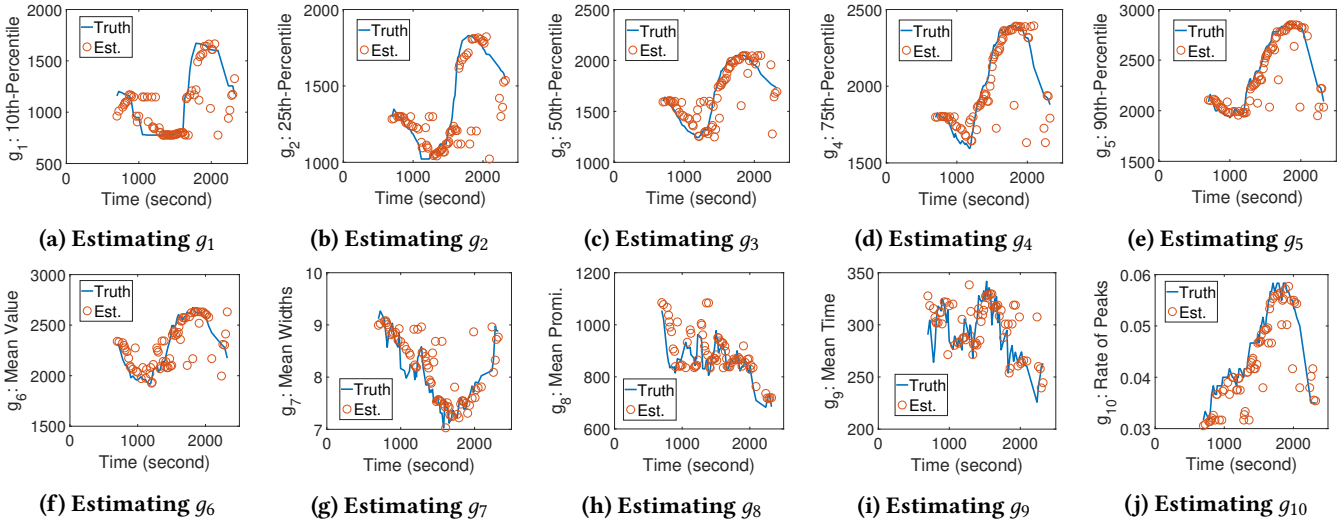
and constructs a norm model that estimates Gs based on Fs.

Feature Extraction. `B-Diag` forms its feature vectors of Fs and Gs based on the two correlations observed in Sec. 3.1: (i) defining f_1–f_5 and g_1–g_5 based on the weakly correlated raw readings of voltage and RPM, thus facilitating the detection of RPMs' unusual values; (ii) defining f_6–f_{10} and g_6–g_{10} based on the strongly correlated peaks of voltage and RPM, thus facilitating the detection of RPMs' unusual dynamics. The weakly correlated raw readings of battery voltage and RPM indicates the feasibility to infer RPMs based on battery voltage, but the accuracy of such an estimation may be limited. As a mitigation, `B-Diag` uses the statistics, instead of the raw values, of voltage/RPM readings to form the first part of its feature vectors. Specifically, for each time window, `B-Diag` uses the $[10, 25, 50, 75, 90]$% percentiles of the voltage and RPM readings as f_1–f_5 for F and g_1–g_5 for G, respectively. `B-Diag` forms the second part of its feature vectors based on the strongly correlated peaks of voltage and RPM. Specifically, for each time window, `B-Diag` uses the mean of the voltage/RPM peaks' value as f_6/g_6, width as f_7/g_7, prominence as f_8/g_8, relative time-stamp as f_9/g_9, and the ratio of their counts over the window size as f_{10}/g_{10}. The relative time-stamp of a peak is defined as its relative position in the current time window, i.e., $t_{\text{r}} = t - (t_{\text{pre}} - T_w)$ where t is the peak's time-stamp. Also, `B-Diag` defines f_{10} and g_{10} as the normalized peak counts to the window size — i.e., the rate at which peaks are observed — to reduce its dependency to the particular setting of window size.

This way, by constructing two feature vectors for each time window, `B-Diag` transforms the two time-series of battery voltage and engine RPM into another two time-series of their corresponding feature vectors, i.e.,

$$\mathbb{F} = \{F^1, F^2, \cdots\} \text{ for battery voltage,}$$
$$\mathbb{G} = \{G^1, G^2, \cdots\} \text{ for engine RPM.}$$

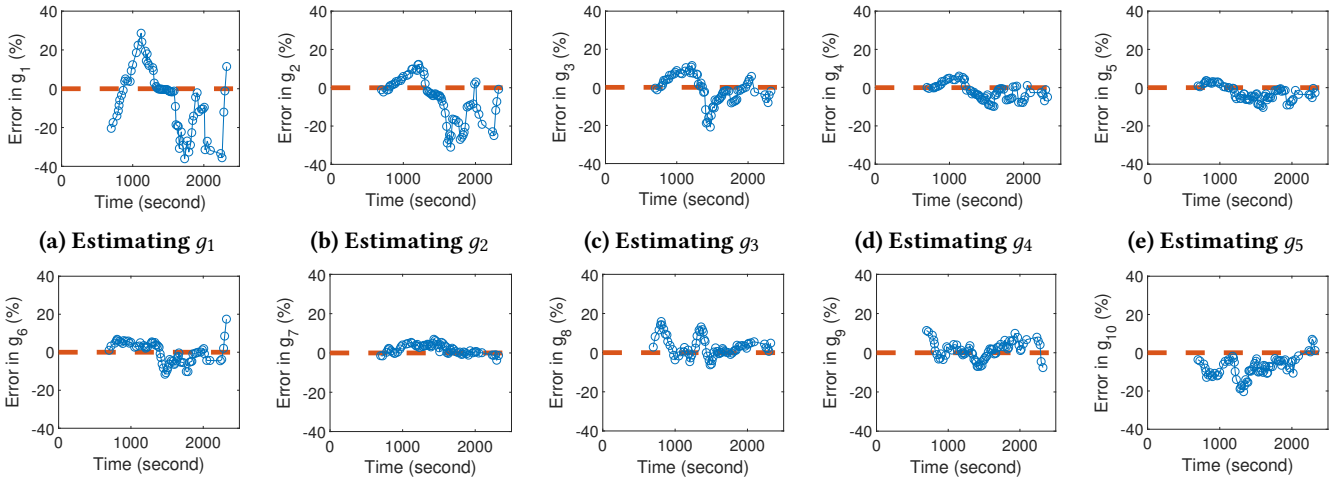
Classifier. `B-Diag` uses machine learning-based classifiers to construct a norm model that maps from $\mathbb{F}$ to $\mathbb{G}$. We opted



(a) Estimating g_1 (b) Estimating g_2 (c) Estimating g_3 (d) Estimating g_4 (e) Estimating g_5

(f) Estimating g_6 (g) Estimating g_7 (h) Estimating g_8 (i) Estimating g_9 (j) Estimating g_{10}

Fig. 13: Estimating the features of engine RPM based on those of battery voltage, with the traces in Fig. 6 as an example.



(a) Estimating g_1 (b) Estimating g_2 (c) Estimating g_3 (d) Estimating g_4 (e) Estimating g_5

(f) Estimating g_6 (g) Estimating g_7 (h) Estimating g_8 (i) Estimating g_9 (j) Estimating g_{10}

Fig. 14: An averaged error of 0.8–11% is achieved when estimating the features of RPM based on battery voltage, but with significant variance.

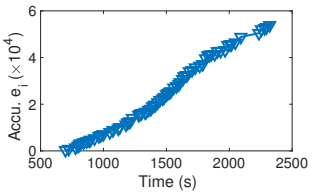
to use *decision tree* as the classifier because of its simplicity and high interpretability.[3] Again, taking the traces in Fig. 6 as an example and with a 10-minute time window, Fig. 13 plots the results when estimating $\mathbb{G}$, or more specifically the g_is in each of the feature vector $G \in \mathbb{G}$, based on $\mathbb{F}$. Fig. 14 summarizes the estimation errors normalized to the corresponding true values (i.e., the error ratios): (i) the errors are clustered around 0 and thus accurate, e.g., the mean errors when estimating g_1–g_{10} are within $[0.8, 11]$%; (ii) variance, however, is observed in both the estimation errors of individual g_is and across different g_is, thus requiring further mitigation when grounding `B-Diag`'s anomaly detection on these errors.

[3]We have tried other classifiers such as KNN and SVM, and observed no clear advantages over the decision tree.

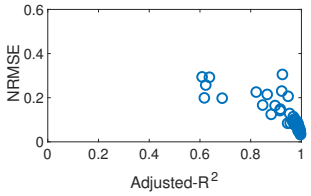
• **Anomaly Detection.** For the i-th time window (and the corresponding feature vectors F^i and G^i), `B-Diag` trains a tree-based model based on the previous feature vectors (i.e., F^1 to F^{i-1} and G^1 to G^{i-1}), and then uses the trained model to estimate G^i based on F^i — an anomaly in RPM is detected if the empirically collected $G^i = \{g^i_j\}$ deviates significantly from the model estimated $\hat{G}^i = \{\hat{g}^i_j\}$ ($j = 1, 2, \cdots, 10$). Specifically, `B-Diag` defines the error of estimating G^i as

$$e_i = ||G^i - \hat{G}^i|| = \sum_{j=1}^{10} \sqrt{(\hat{g}^i_j - g^i_j)^2}/g^i_j \times 100\%. \quad (1)$$

Such a summation of individual estimation errors of g_js suppresses their relatively large variance observed in Fig. 14. To



(a) Accumulated e_is obtained with the traces in Fig. 6, showing clear linearity



(b) Fitting accumulated e_is linearly with all traces in Fig. 4(e)

Fig. 15: The accumulated e_is (see Eq. (1)) increase linearly.

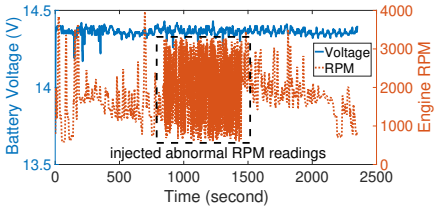


Fig. 17: Adding anomalies to genuine RPM readings in Fig. 6.

collaborate this, Fig. 15(a) plots the accumulated e_is obtained with the traces shown in Fig. 6, which increases steadily and linearly. Fig. 15(b) plots the goodness-of-fit when fitting the accumulated e_is linearly for each of the 64 traces summarized in Fig. 4(e). The fact that all the fitting results are clustered at the right-bottom corner of the figure — i.e., with close-to-1 Adjusted-R^2 and close-to-0 NRMSE — validates the high fitting goodness and thus the linearity of accumulated e_is.

This linearity of accumulated e_is allows B-Diag to describe it as a linear regression model. A linear parameter identification problem is thus formulated as

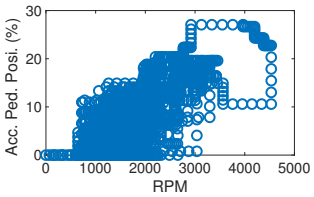
$$E_{\text{acc}}[i] = S[i] \cdot t[i] + \delta[i], \tag{2}$$

where for the i-th time window ending at time $t[i]$, $E_{\text{acc}}[i]$ is the accumulated e_is, $S[i]$ is the regression parameter, and $\delta[i]$ is the identification error. The regression parameter S represents the slope of the linear model and thus the averaged e_i over time. The identification error δ represents the residual which is not explained by the model. B-Diag uses the Recursive Least Squares (RLS) algorithm to solve such a linear regression. Also, the reliable linear model of accumulated e_is indicates the corresponding identification error δs should be small and stable in the normal cases, motivating B-Diag to make its decision on anomaly detection based on δs. Specifically, B-Diag defines

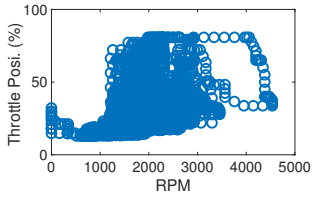
$$\tau_i = |\delta_i - \mu(\delta_1 : \delta_{i-1})| / \sigma(\delta_1 : \delta_{i-1}), \tag{3}$$

i.e., τ_i is the deviation of δ_i from the mean of $\{\delta_1, \cdots, \delta_{i-1}\}$ in terms of their standard deviation, and concludes an anomaly in RPM is detected if $\tau_i > \theta$. We set θ=5 by default [35, 54].

• **Anomaly Verification.** B-Diag further verifies the detected anomalies by exploiting the fact that engine RPM, besides correlates strongly with the battery voltage, also correlates with other vehicle parameters. For example, we have



(a) RPM v.s. acc. pedal



(b) RPM v.s. throttle position

Fig. 16: B-Diag verifies the detected RPM anomalies based on the correlations between RPM and other vehicle information, e.g., accelerator pedal position and throttle position.

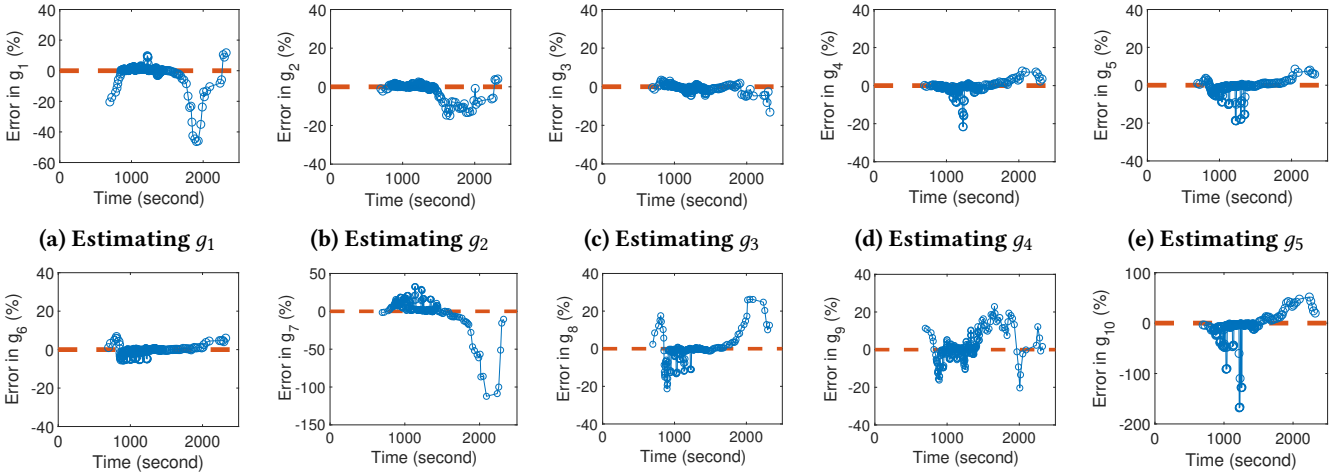
identified the physically-induced correlations between RPM and the accelerator pedal position and throttle position, as shown in Fig. 16. B-Diag further exploits these non-battery correlations with RPM to verify the above-detected RPM anomalies, based on the hypothesis that RPM anomalies, besides causing abnormal behavior with regard to the battery voltage, will also cause its abnormal behaviors with regard to other correlated vehicle information. B-Diag conducts such an anomaly verification, again, via norm model construction and then checking, with similar approaches explained above. B-Diag will confirm the detected RPM anomalies if abnormal behaviors between RPM and any of these correlated vehicle information is detected.

• **Walk-Through Example.** Next we use a walk-through example to show how B-Diag detects and then verifies anomalies in engine RPM based on the battery voltage. Specifically, we emulate RPM anomalies by injecting randomly fabricated RPM readings into the traces in Fig. 6, and test if B-Diag can detect such anomalies. Fig. 17 plots the altered RPM trace after injecting anomalies during the time period of [849, 1449]s. Applying B-Diag to the thus-altered trace with a window size of 600s, Fig. 18 plots the errors when estimating RPM's feature parameters based on those of the battery voltage, showing much degraded accuracy at $\{g_5, g_7, g_8, g_9, g_{10}\}$ when compared to Fig. 14.[4] Fig. 19 plots the accumulated estimation errors — i.e., e_is as defined in Eq. (1) — showing the injected anomalies change the slope of the accumulated e_is, and thus being detectable. Note that no anomaly is detected when applying B-Diag to the raw traces in Fig. 6, and thus no false detection is caused. We further verify the detected RPM anomalies by cross-validating with the accelerator pedal position and throttle position, which are confirmed with the changed slopes of accumulated e_is (see Fig. 20).

4 DIAGNOSING VEHICLE BEYOND RPM

We have used the detection of RPM anomalies with battery voltage to walk through B-Diag's cyber-physical approach of vehicle diagnostics. Besides the engine, physical dependencies with the automotive battery also exists at other vehicle

[4]The specific feature parameters with degraded estimation accuracy will depend on the particular anomalies.



(a) Estimating g_1 (b) Estimating g_2 (c) Estimating g_3 (d) Estimating g_4 (e) Estimating g_5

(f) Estimating g_6 (g) Estimating g_7 (h) Estimating g_8 (i) Estimating g_9 (j) Estimating g_{10}

Fig. 18: The abnormal RPMs in Fig. 17 magnify the errors when estimating $\{g_5, g_7, g_8, g_9, g_{10}\}$, when compared to Fig. 14.

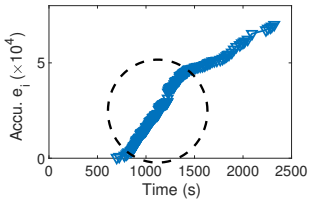


Fig. 19: Abnormal RPMs change the slope of the accumulated e_is with regard to battery voltage, and thus being detectable.

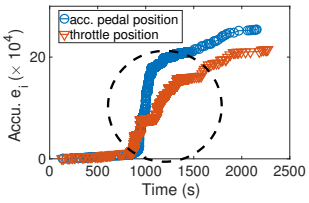


Fig. 20: Abnormal RPMs change the slopes of the accumulated e_is regarding to acc. pedal position and throttle position.

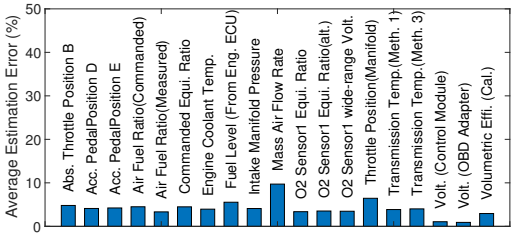


Fig. 22: Errors in estimating feature vectors of vehicle information listed in Table 1 (besides those relate to engine RPM).

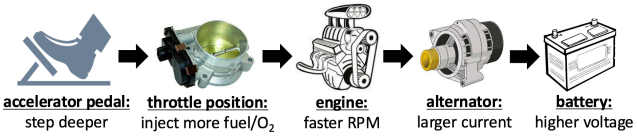


Fig. 21: Physically-induced correlations among a vehicle's accelerator pedal, throttle, engine, alternator, and battery.

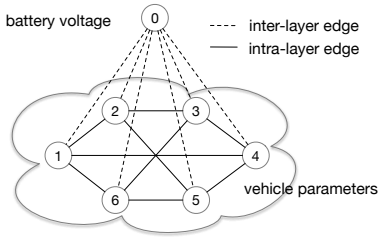


Fig. 23: B-Diag uses a correlation graph to abstract the vehicle and the correlations thereof.

modules, thus offering opportunities to generalize B-Diag's anomaly detection to other vehicle information. Specifically, examining the physical inter-connection among vehicle's sub-systems, Fig. 21 shows a dependency diagram among vehicle's accelerator pedal, throttle, engine, alternator, and battery. Steered by this dependency diagram, we further checked the related vehicle information collected when driving the Crosstrek, and confirmed the correlations with the battery voltage at the information listed in Table 1. As an example, Fig. 22 plots the averaged error when estimating the vehicle information listed in Table 1 (besides those relate to engine RPM), or more specifically their feature vectors, based on the traces collected during the same trip as with Fig. 6 , showing averaged errors within [0.9, 9.7]%. Note that Fig. 21 may not be thorough in capturing the physical dependency among vehicle's sub-systems, and thus Table 1 may not be exclusive. These multi-modal correlations between battery voltage and other vehicle information enable B-Diag to act as a comprehensive diagnostic system for vehicles.

To facilitate the systematic exploitation of these physically-induced correlations for vehicle diagnostics, B-Diag abstracts the vehicle with a 2-layer correlation graph $\mathbb{G}_{\text{corr}} = \{V, E\}$, in which: (i) the vertex set V represents the operational parameters of the vehicle with the battery voltage at the upper layer and other vehicle parameters at the lower layer; (ii) the edge set E captures the correlations among vehicle parameters, i.e., an edge $e_{i,j}$ connecting vertex v_i and v_j exists in E if v_i and v_j are correlated. Fig. 23 shows an exemplary correlation graph.

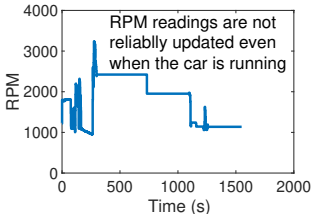


Fig. 24: RPM collected with an unreliable adapter.

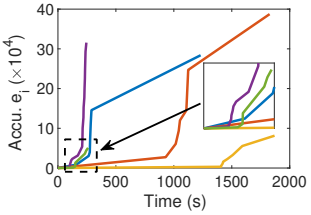
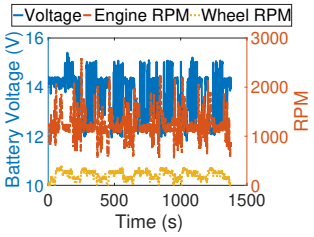
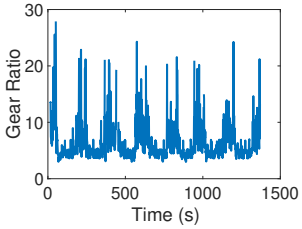


Fig. 25: Accu. e_is of engine RPM with abnormal traces.



(a) Raw traces



(b) Gear ratio

Fig. 27: Emulating abnormal engine RPM with wheel's RPM.

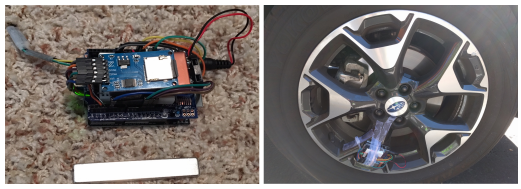


Fig. 26: Attach a Hall-based RPM sensor to the wheel.

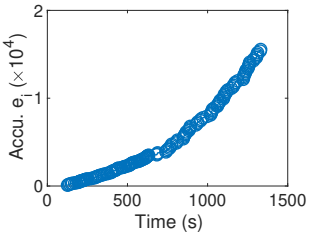


Fig. 28: Accumulated e_is of emulated anomalies.

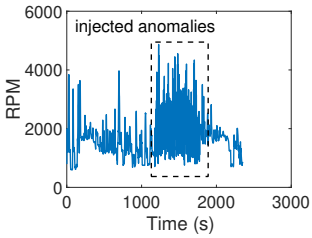


Fig. 29: Example on injected abnormal RPM with AM-2.

This 2-layer correlation graph facilitates exploiting the uniqueness of automotive batteries to diagnose vehicles: (i) battery voltage can be directly (and easily) collected from the battery without going through the in-vehicle network, thus being tolerable to the risks of cyber-induced anomalies thereof and serving as a trustworthy ground for B-Diag's diagnostics of vehicles; (ii) battery draws power from the alternator and supplies power to the vehicle's electrical systems, implying strong correlations between battery voltage and many vehicle parameters. As a result, the correlation graph consists of two types of edges: the *inter-layer edges* representing the correlations between battery voltage and other vehicle parameters, and the *intra-layer edges* capturing the pairwise correlations between vehicle parameters besides battery voltage. Note the correlation graph also defines B-Diag's ability/limit of diagnosing vehicles, i.e., which vehicle modules/parameters B-Diag can guard.

With such an abstracted correlation graph $\mathbb{G}_{\text{corr}}$, B-Diag's detection/verification of anomalies at each vehicle information is transformed to the construction (and then checking) of the data-driven norm model(s) defined by the corresponding inter/intra-layer edges. This way, B-Diag can take a round-robin approach to check the individual inter-layer edges of $\mathbb{G}_{\text{corr}}$ for anomaly detection: substituting the engine RPM in Sec. 3 with the target vehicle information and detecting/verifying the anomalies thereof with similar approaches.[5]

5 EVALUATIONS

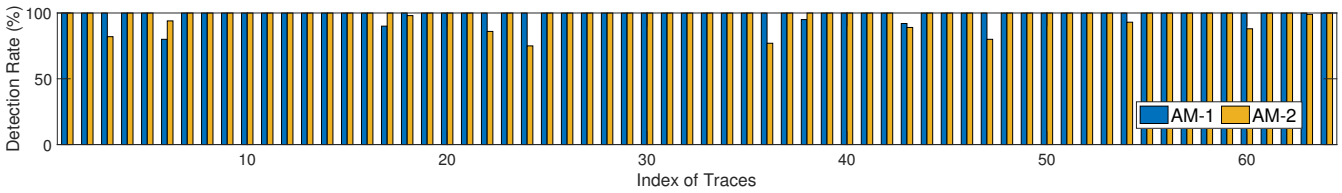
We have evaluated B-Diag with four vehicles: a 2018 Subaru Crosstrek, a 2008 Honda Fit, a 2018 Volvo XC60, and a 2017 Volkswagen Passat. The major challenge in B-Diag's evaluation is the short of real-life cases on vehicle anomalies, gathering of which incurs safety risks. We mitigate this by evaluating B-Diag with (i) anomalies caused by an unreliable OBD-II adapter, (ii) emulated anomalies based on wheel's RPM, and (iii) simulated anomalies by injecting fabricated values to normal vehicle traces.

5.1 B-Diag against "True" Anomalies

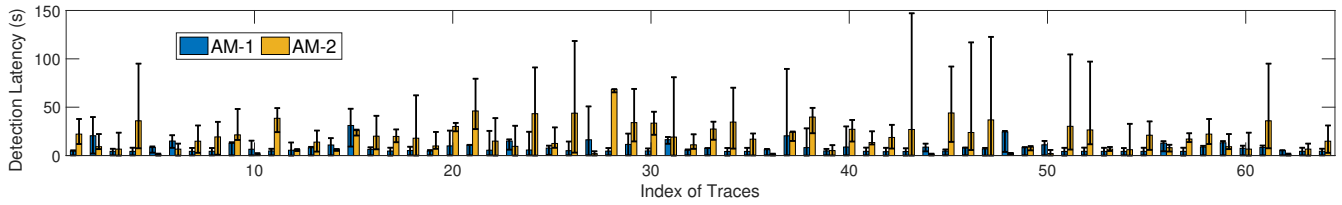
• **Methodology.** We have identified one faulty Bluetooth OBD-II adapter that is unreliable in collecting the vehicle information, as plotted in Fig. 24 with the engine RPM as an example: the RPM keeps constant for up to over 10 minutes when driving the Crosstrek in urban road with frequent acceleration and braking. We have further verified the unreliability of this adapter with different vehicles. The abnormal vehicle information collected with this faulty adapter serves as a promising candidate to evaluate B-Diag's ability in detecting the anomalies thereof, even though these anomalies are caused due to the faults of the OBD-II adapter and not the vehicle. For example, the deficient updates of engine RPM in Fig. 24 could map to faulty (or hacked [22]) tachometer of the vehicle.

• **Evaluation Results.** We have collected 5 abnormal vehicle traces with the Subaru Crosstrek using this "faulty" adapter, each lasting about {25, 41, 33, 6, 6} minutes. We then apply B-Diag with a moving window of 60s to these traces, to detect the anomalies in the vehicle information listed in Table 1. B-Diag successfully detects the anomalies in *all* these vehicle information of *all* the 5 traces. As an example, Fig. 25 plots the accumulated e_is of the engine RPM of these abnormal traces: the abrupt changes in slopes validate the detectability of anomalies thereof by B-Diag.

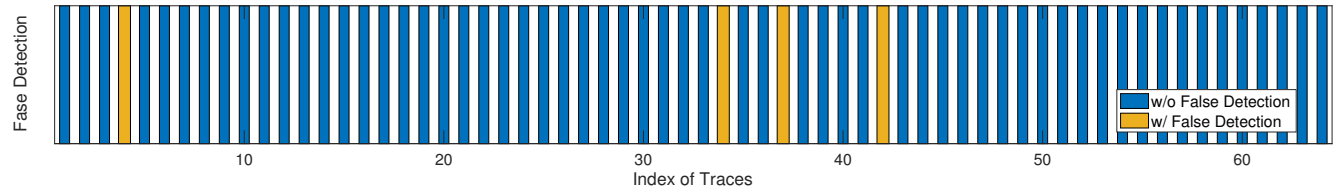
[5]It is possible to design advanced scheduling methods to check the edges based on factors such as the criticality of vehicle information [55].



(a) `B-Diag` **achieves >75% detection rate against RPM anomalies for each of these traces, with an averaged detection rate of** 99% **(with AM-1) and** 97% **(with AM-2) across all traces**



(b) `B-Diag` **achieves <68s average latency in detecting the RPM anomalies for each of these traces, with an averaged detection latency of** 8s **(with AM-1) and** 19s **(with AM-2) across all traces**



(c) `B-Diag` **falsely detects RPM anomalies in 4 out of the 64 traces**

Fig. 30: `B-Diag`**'s detection rate, latency, and false detection against RPM anomalies, with the 64 Crosstrek traces in Fig. 4(e).**

• **Adapter Faults or Vehicle Faults?** The above evaluation of `B-Diag` leads to an interesting and important question: can `B-Diag` differentiate the anomalies caused due to the faults of its own data collection and those by the actual vehicle failures? The answer is confirmative because the faults of data collection will cause anomalies in all the collected vehicle information, while the actual vehicle failures will, unless in a rare case where most of the vehicle modules fail, cause anomalies only in the vehicle information related to the faulty vehicle modules. Also note the risk of these adapter-caused anomalies can be reduced by employing reliable OBD-II adapters, e.g., using the wired OBD-II adapters to collect the vehicle information instead of those based on Bluetooth.

5.2 B-Diag against Emulated Anomalies

Next we evaluate `B-Diag` by emulating anomalies of engine RPM based on the wheel's RPM empirically collected during the same driving trip. Wheel RPM quantifies the rotation speed of the vehicle's wheels. Mechanically,

$$RPM_{\text{wheel}} = \alpha_i(t) \cdot RPM_{\text{engine}}, \tag{4}$$

where $\alpha(t)$ is the gear ratio at time t, determined by the real-time driving behavior. The empirically collected wheel's RPMs are a promising candidates to emulate the anomalies of engine RPMs during the same driving trip, because: (i) wheel's RPMs fall in the legal range of engine RPM (i.e., with $\alpha_i(t)$=1), making the thus-emulated anomalies possible to occur in practice and not diagnosable by existing range-based diagnostics systems [2, 36], (ii) the wheel RPM is strongly correlated to, but different from, the engine RPM, and such a correlation is dynamic over time.

Inspired by this, we build an Arduino-based RPM sensor with a hall sensor and a magnetic, and attach it to the front-right wheel of the Subaru Crosstrek (see Fig. 26). Fig. 27(a) plots the collected wheel RPM, engine RPM, and battery voltage during a 23-minute drive of the vehicle. The corresponding gear ratio during this driving is plotted in Fig. 27(b). We then emulate the abnormal engine RPMs by concatenating the first 10-minute trace of engine RPM and the last 13-minute trace of wheel RPM, and examine if `B-Diag` is able to detect such emulated anomalies. Fig. 28 plots the accumulated e_is obtained with such an emulation, whose change in slope at about the 10.5th minute — i.e., about 0.5 minute after the emulated anomalies begin — validates `B-Diag`'s ability of detecting such anomalies.

5.3 B-Diag against Simulated Anomalies

We also evaluated `B-Diag` against simulated anomalies in the vehicle information.

• **Anomaly Model.** We emulate vehicle anomalies by injecting fabricated vehicle information to the collected normal

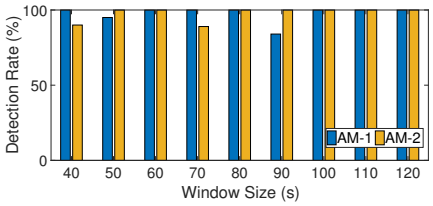


Fig. 31: Detection rate v.s. window size.

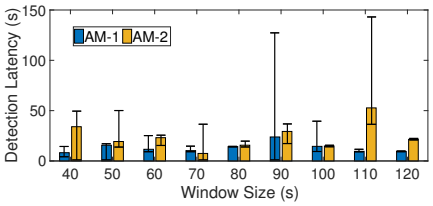


Fig. 32: Detection delay v.s. window size.

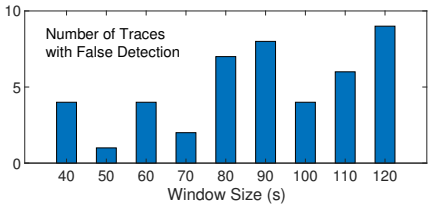


Fig. 33: False detection v.s. window size.

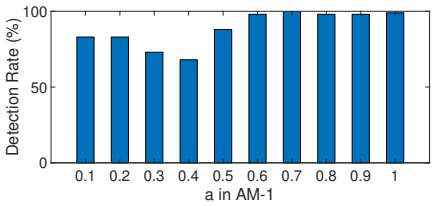


Fig. 34: Detection rate v.s. AM-1's model parameter a in Eq. (5).

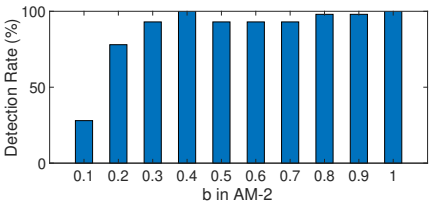


Fig. 35: Detection rate v.s. AM-2's model parameter b in Eq. (6).

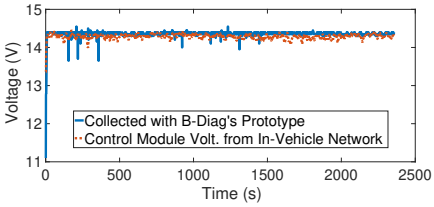


Fig. 36: `Control Module Voltage` is a close approximation to battery voltage.

traces. Specially, denoting the genuine time series of the targeting vehicle information (e.g., engine RPM) as $v(t)$, we inject anomalies to $v(t)$ with the following two anomaly models, emulating the fabrication and masquerade attacks in practice [35].

AM-1: Injecting anomalies after a randomly selected time t_{anom} by fabricating $v(t)$ within a legal range:

$$v'(t) = (1 - rand(a)) \cdot v_{\min} + rand(a) \cdot v_{\max} \ (t \geq t_{\text{anom}}), \quad (5)$$

where $v_{\min}$ and $v_{\max}$ are the min/maximum of $v(t)$, $rand(a)$ returns a random value in $[0, a]$, and $a \in [0, 1]$ controls the levels of the fabricated readings (a=0.8 unless specified otherwise). The thus-fabricated vehicle information will still be within the min/maximum of its genuine levels, thus voiding existing range-based diagnostics systems [2, 36]. This is also how we generate the RPM anomalies in Fig. 17.

AM-2: Injecting anomalies after a randomly selected time t_{anom} by shifting $v(t)$s from their true values randomly:

$$v'(t) = (1 - b + 2b \cdot rand(1)) \cdot v(t) \ (t \geq t_{\text{anom}}), \quad (6)$$

where b controls the maximum shift of the fabricated $v(t)$s from their true values (b=0.5 unless specified otherwise). Fig. 29 shows an example of such generated RPM anomalies based on the genuine trace in Fig. 6.

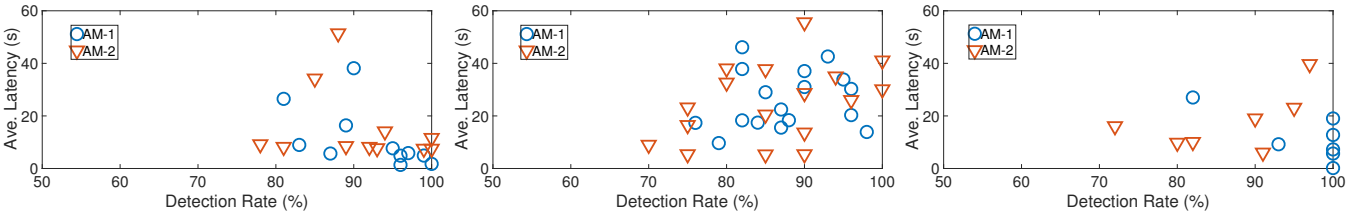
• **Evaluation with Subaru Crosstrek.** We first evaluate `B-Diag` with the 64 driving traces of the Subaru Crosstrek summarized in Fig. 4(e), taking again, the anomalies in engine RPM as an example.

Overall Performance. Fig. 30 summarizes `B-Diag`'s performance in anomaly detection, in terms of the detection rate (Fig. 30(a)), detection latency (Fig. 30(b)), and false detection (Fig. 30(c)), with a 60s moving window. The results in Figs. 30(a) and 30(b) are based on randomly injected RPM anomalies according to AM-1 and AM-2, each with 100 tests. As shown in Fig. 30(a), `B-Diag` achieves an overall averaged detection rate of 99% and 97% against the anomalies injected according to AM-1 and AM-2, respectively, and archives 100% detection rate for many of these 64 traces. The minimum detection rate of these traces with AM-1/2 is 80/75%. Fig. 30(b) shows `B-Diag` detects the anomalies with an averaged latency of less than 31s and 68s, for the two anomaly models respectively. The overall average detection latency across all these 64 traces is 8s for AM-1 and 19s for AM-2. We have also evaluated `B-Diag`'s false detection of anomalies. Specifically, we apply `B-Diag` to the genuine RPM traces without injecting anomalies, and check if any false detection of RPM anomalies is triggered. Fig. 30(c) shows `B-Diag` falsely detects RPM anomalies in only 4 of the 64 traces. Moreover, `B-Diag` only falsely detects the anomalies in $\{1, 1, 2, 2\}$ of the 60s moving windows in the 4 traces with false detection, which last about $\{30, 26, 25, 25\}$ minutes, respectively.

Impact of Window Size. We next evaluate the impact of the size of `B-Diag`'s moving window on its performance in anomaly detection, based on the RPM traces shown in Fig. 6. Figs. 31 and 32 summarize `B-Diag`'s detection rate and latency of injected anomalies with the window size varying from 40–120s, showing an over 84/89% detection rate for all the explored cases and an average latency of 13s and 24s, with AM-1 and AM-2, respectively. No clear dependency between `B-Diag`'s detection rate against anomalies and the window size is observed from Fig. 31. On the other hand, Fig. 32 shows a larger moving window tends to increase the latency of `B-Diag`'s anomaly detection, which is expected as a larger window requires more samples of abnormal RPM readings to conclude the detection of anomalies, thus requiring a longer time. Fig. 33 summarizes the false detection of anomalies when applying `B-Diag` to each of these genuine 64 traces (and thus without anomalies) with varying window size. The

(a) 2008 Honda Fit (b) 2018 Volvo XC60 (c) 2017 Volkswagen Passat

Fig. 37: Further evaluating B-Diag with the driving traces collected with a 2008 Honda Fit, a 2018 Volvo XC60, and a 2017 Volkswagen Passat.



(a) 2008 Honda Fit (b) 2018 Volvo XC60 (c) 2017 Volkswagen Passat

Fig. 38: B-Diag detects the anomalies of Honda Fit, Volvo XC60, and Volkswagen Passat with averaged rate/latency of $\{92\%, 88\%, 96\%\}/\{91\%, 86\%, 87\%\}$ **and** $\{11, 26, 12\}/\{15, 28, 18\}$**s, with the two anomaly models of AM-1 and AM-2, respectively.**

Table 2: Summary of the traces collected with a 2008 Honda Fit, a 2018 Volvo XC60, and a 2017 Volkswagen Passat.

Vehicles	# of Traces	Total Duration	Total Distance
Honda Fit	11	4.7 hour	≈160 miles
Volvo XC60	17	4.1 hour	≈210 miles
Volkswagen Passat	7	29 hour	≈1,840 miles

number of traces in which B-Diag falsely observes anomalies increases as the window becomes larger, varying from 1–9 with window size between 40–120s. Also note that even for traces with falsely observed anomalies, only a limited number of time windows therein detect such anomalies, e.g., with a maximum of 4 windows in all the cases in Fig. 33, and thus accounting for only a limited period when compared to the total driving duration of about 31 hours (as summarized in Fig. 4(e)).

Impacts of Anomaly Models' Parameters. Figs. 34 and 35 summarize B-Diag's detection rates of injected anomalies based on the RPM traces in Fig. 6, with varying a in Eq. (5) and b in Eq. (6) respectively. The results are averaged over 100 tests for each setting. B-Diag detects the anomalies fabricated with AM-1 with over 68% detection rates with $a \in [0.1, 1]$ (see Fig. 34). The relatively low detection rate of 68% with a=0.4 is because in this case, the abnormal RPMs deviate little from their genuine levels. Specifically, with the RPM trace shown in Fig. 6, $a = 0.4$ leads to abnormal RPMs within [573, 2069] according to Eq. (5), which are close to their true values (as observed in Fig. 6). Also note that other things being equal, a smaller deviation of RMP readings from the genuine levels will cause less safety/reliability risks, when compared to those change the RPMs dramatically. Fig. 35 shows B-Diag accurately detects the anomalies when the model parameter b in Eq. (6) is not too small, e.g., with over 93% detection rates when $b \geq 0.3$. The low detection rate with b=0.1 is because, again, a small b in AM-2 causes little deviation of abnormal RPM readings from their true levels.

• **Evaluation with Other Vehicles.** To validate B-Diag's generality with different vehicles, we have further evaluated B-Diag based on the driving traces collected with a 2008 Honda Fit, a 2018 Volvo XC60, and a 2017 Volkswagen Passat (see Fig. 37), each with its respective owner/driver. Table 2 summarizes the details of these traces. Different from the Crosstrek traces where the battery voltage is collected with our prototype and in physical separation of the in-vehicle network, we use the `control module voltage` collected from the in-vehicle network via the OBD-II port as a close approximation of the battery voltage for these three vehicles, for the ease of data collection. The `control module voltage` represents the real-time voltage supplied to the vehicle's ECUs, i.e., the battery voltage minus any voltage drop in the wiring between the battery and ECUs, normally less than a few tenths of a volt. Fig. 36 compares the `control module voltage` with the corresponding battery voltage collected directly from the battery, corroborating their closeness. Fig. 38 plots B-Diag's detection rate and latency against the added anomalies, with a 60s moving window and averaged over 100 runs. For the two anomaly models AM-1 and AM-2, B-Diag detects the anomalies with (i) an averaged detection rate of 92/91% and a latency of 11/15s for Honda Fit, (ii) an averaged detection rate of 88/86% and a latency of 26/28s for Volvo XC60, and (iii) an averaged detection rate of 96/87% and a latency of 12/18s for Volkswagen Passat.

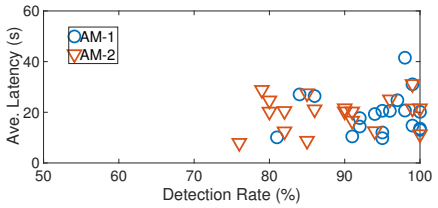


Fig. 39: Using B-Diag to diagnose the information in Fig. 22.

• **Diagnosing beyond Engine RPM.** We also corroborated B-Diag's feasibility as a comprehensive diagnostics system for vehicles. Specifically, we use B-Diag to detect the anomalies at each of these vehicle information in Fig. 22, injected according to the two anomaly models. Fig. 39 summarizes the detection results: all of these anomalies are detected with over 81/76% detection rate and a latency less than 42/31s, with an average of 94/89% and 19/20s, respectively. Note that no false detection is observed when applying B-Diag to the raw traces without injecting anomalies.

Last but not the least, we further verify B-Diag's generality in diagnosing the vehicle information listed in Table 1 — which are originally identified with the Subaru Crosstrek — with the Honda Fit, Volvo XC60, and Volkswagen Passat. The strong correlations of these vehicle information with the battery voltage are observed in all these three vehicles, thus validating B-Diag's generality. This also demonstrates the advantage of B-Diag's cyber-physical approach in vehicle diagnostics: the physical dependencies among a vehicle's individual modules are (likely) general for different vehicles, making the correlations among corresponding vehicle information universal.

6 LIMITATIONS

Below we discuss a few limitations of the current design/evaluation of B-Diag, and their potential mitigations.

• **Norm Model Construction.** B-Diag constructs the norm models with a decision tree defined by 10 features. We will further improve such a model construction by (i) reducing the number of features and (ii) exploiting the (likely) overlapped time windows, thus reducing the complexity and improving the online diagnostics of vehicles. Also, B-Diag assumes the availability of normal traces to construct the norm model. We will further explore B-Diag's ability of vehicle diagnostics when the normal traces are not available.

• **Anomaly Detection and Verification.** B-Diag detects the anomalies based on the τ_is (defined in Eq. (3)) for each time window. An alternative is to conclude the detection of anomalies only when multiple τ_is satisfying Eq. (3) have been observed in several consecutive time windows, trading off between the anomaly detection's false positive and false negative. We will also need to consider the latency of anomaly detection, which is desirable to be as small as possible. We envision the window size should be a promising control knob to reduce the latency. B-Diag verifies the detected anomalies by checking the norm models defining the interplays among vehicle information, and confirms the detected anomalies if any of these checkings fail. We will investigate other methods for anomaly confirmation, e.g., by weighting the checking results of individual norm models.

• **Fault Identification.** After detecting/verifying the anomalies, B-Diag will need to identify the corresponding causes, i.e., which modules (or ECUs) of the vehicle fail? Such a fault identification is required to provide a swift repair/forensic, otherwise the vehicle remains unreliable no matter how accurate the anomalies are detected. We will steer B-Diag's fault identification based on the correlation graph defined in Fig. 23, by examining the connectivity among vertexes (i.e., vehicle information) with detected anomalies.

• **Diagnosing beyond Engine RPM.** We have validated B-Diag's ability in individually diagnosing other vehicle information beyond engine RPM in Sec. 4. An integrated solution that guards all vehicle information in real-time, however, is still needed to make B-Diag a comprehensive solution for vehicle diagnosis, especially in view of the possibility of cascaded anomalies in vehicles, i.e., anomalies in one vehicle information may cause anomalies in other information.

• **Evaluation against Real-Life Vehicle Anomalies.** Although we have validated B-Diag with different approaches in Sec. 5, B-Diag's evaluation against real-life vehicle anomalies is still missing. Such an evaluation of B-Diag may cause vehicle malfunction and thus incur safety risks. We will mitigate these challenges with two steps: (i) testing when using jack stands to raise the vehicle from the ground, thus ensuring safety, and then (ii) testing when driving on testing field, such as the Mcity Test Facility at University of Michigan.

7 CONCLUSION

In this paper, we have designed B-Diag, a battery-based diagnostics system that guards vehicles against anomalies in real-time, and implemented B-Diag as an add-on module of commodity vehicles attached to automotive batteries. B-Diag is inspired by the physically-induced correlations between the battery voltage and other operational parameters of the vehicle such as engine RPM. B-Diag exploits these correlations to diagnose vehicles by exploiting automotive batteries as anomaly sensors: cross-validating vehicle information with online constructed norm models with regard to the battery voltage, steered by a dataset collected when driving a 2018 Subaru Crosstrek in real-life for over 3 months. We have evaluated B-Diag based on the driving traces collected with, besides the Crosstrek, a 2008 Honda Fit, a 2018 Volvo XC60, and a 2017 Volkswagen Passat, showing B-Diag detects anomalies in vehicle information with over 86% detection rate on average.

Acknowledgments. We would like to thank the anonymous reviewers and the shepherd, Dr. Marco Gruteser, for constructive suggestions. The work reported in this paper was supported by NSF under Grant CNS-1739577.

REFERENCES

[1] Car Software: 100M Lines of Code and Counting. https://www.linkedin.com/pulse/20140626152045-3625632-car-software-100m-lines-of-code-and-counting.

[2] M. Muter, A. Groll, and F. C. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *IAS'10*, 2010.

[3] Kyong-Tak Cho and Kang G. Shin. Error handling of in-vehicle networks makes them vulnerable. In *CCS'16*, 2016.

[4] Kyong-Tak Cho, Kang G. Shin, and Taejoon Park. CPS approach to checking norm operation of a brake-by-wire system. In *ICCPS'15*, 2015.

[5] J. P. Hubaux, S. Capkun, and Jun Luo. The security and privacy of smart vehicles. *IEEE Security Privacy*, 2(3):49–55, 2004.

[6] BMW cars found to contain more than a dozen flaws. http://www.bbc.com/news/technology-44224794.

[7] Software Glitches in the Auto Industry and What that Means for You. https://www.proservicescorp.com/auto-industry-software-glitches/.

[8] Study: Tesla, Jaguar highest in auto software defects. https://www.usatoday.com/story/money/cars/2016/05/24/jd-power-software-defects-tesla-motors-jaguar-land-rover/84841174/.

[9] CAN Bus. https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en.

[10] Toyota vehicle recalls. https://en.wikipedia.org/wiki/2009%E2%80%9311_Toyota_vehicle_recalls.

[11] Jaguar recall shows how software glitches are the new speed bump. http://fortune.com/2015/07/13/jaguar-recall-software-glitch/.

[12] Fiat Chrysler recalls 4.8 million vehicles that could get stuck in cruise control. http://money.cnn.com/2018/05/25/autos/fca-recall-cruise-control/index.html.

[13] R. R. Brooks, S. Sander, J. Deng, and J. Taiber. Automobile security concerns. *IEEE Vehicular Technology Magazine*, 4(2):52–64, 2009.

[14] D. Nilsson and U. Larson. A roadmap for securing vehicles against cyber attacks. In *NITRD National Workshop on High-Confidence Automotive Cyber-Physical Systems*, 2008.

[15] D. Nilsson and U. Larson. Simulated attacks on CAN buses: vehicle virus. In *AsiaCSN'08*, 2008.

[16] Ishtiaq Rouf, Rob Miller, Hossen Mustafa, Travis Taylor, Sangho Oh, Wenyuan Xu, Marco Gruteser, Wade Trappe, and Ivan Seskar. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *USENIX Security'10*, 2010.

[17] Tesla responds to Chinese hack with a major security upgrade. https://www.wired.com/2016/09/tesla-responds-chinese-hack-major-security-upgrade/.

[18] Yasser Shoukry, Paul Martin, Paulo Tabuada, and Mani Srivastava. Non-invasive spoofing attacks for anti-lock braking systems. In *CHES'13*, 2013.

[19] A. Palanca. *A stealth, selective, Link-layer Denial-of-Service attack against automotive networks*. PhD thesis, Politecnico Milano, 2016.

[20] Kyong-Tak Cho and Kang G. Shin. Viden: Attacker identification on in-vehicle networks. In *CCS'17*, 2017.

[21] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security'11*, 2011.

[22] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *S&P'10*, 2010.

[23] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. In *DEFCON'11*, 2011.

[24] Charlie Miller and Chris Valasek. CAN Message Injection. http://illmatics.com/can%20message%20injection.pdf.

[25] Hackers remotely kill a Jeep on the Highway — with me in it. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[26] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. In *Black Hat USA'15*, 2015.

[27] On-Board Diagnostics. http://www.car-engineer.com/introduction-to-on-board-diagnostic-obd/.

[28] D. K. Nilsson, U. E. Larson, and E. Jonsson. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *VTC'08*, 2008.

[29] Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede. CANAuth – A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus. In *LC'11*, 2011.

[30] Chris Szilagyi and Philip Koopman. Low cost multicast authentication via validity voting in time-triggered embedded control networks. In *WESS'10*, 2010.

[31] Kyong Tak Cho. *From Attack to Defense: Toward Secure In-vehicle Networks*. PhD thesis, University of Michigan, 2018.

[32] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee. Identifying ECUs through inimitable characteristics of signals in controller area networks. *IEEE Transactions on Vehicular Technology*, pages 1–1, 2018.

[33] M. Muter and N. Asaj. Entropy-based anomaly detection for in-vehicle networks. In *IV'11*, 2011.

[34] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks — practical examples and selected short-term countermeasures. In *SAFECOMP'08*, 2008.

[35] Kyong-Tak Cho and Kang G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *USENIX Security'16*, 2016.

[36] Armin Wasicek, Mert D. Pese, Andre Weimerskirch, Yelizaveta Burakova, and Karan Singh. Context-aware intrusion detection in automotive control systems. In *ESCAR'17*, 2017.

[37] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. In *Black Hat USA*, 2015.

[38] Patrick E. Lanigan, Soila Kavulya, Priya Narasimhan, Thomas E. Fuhrman, and Mutasim A. Salman. Diagnosis in automotive systems: A survey, 2011.

[39] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee. Identifying ECUs Using Inimitable Characteristics of Signals in Controller Area Networks. *ArXiv e-prints*, 2016.

[40] P. S. Murvay and B. Groza. Source identification using signal characteristics in controller area networks. *IEEE Signal Processing Letters*, 21(4):395–399, 2014.

[41] David S. Breed. System and method for vehicle diagnostics, 2006.

[42] David S. Breed. Telematics system for vehicle diagnostics, 2004.

[43] Christopher R. Baker, David Ferguson, and John M. Dolan. Robust mission execution for autonomous urban driving. In *IAS'08*, pages 155–163, July 2008.

[44] Galen Hunt, George Letey, and Ed Nightingale. The seven properties of highly secure devices. Technical report, 2017.

[45] Selected worldwide automotive manufacturers' profit margin between January 2016 and June 2016. https://www.statista.com/statistics/697263/automotive-manufacturers-profit-margin-worldwide/.

[46] OBD-II PIDs. https://en.wikipedia.org/wiki/OBD-II_PIDs.

[47] 6 Most Common Crankshaft Position Sensor Symptoms. https://carfromjapan.com/article/car-maintenance/common-crankshaft-position-sensor-symptoms/.

[48] How to Diagnose a Bad or Failing Transmission Speed Sensor? https://www.yourmechanic.com/article/symptoms-of-a-bad-or-failing-transmission-speed-sensor.

[49] Understanding Your Alternator. http://www.hotrod.com/articles/0206sr-understanding-your-alternator/.

[50] Robert Bosch. *Bosch Automotive Electrics and Automotive Electronics*. Springer, 2014.

[51] L. He, G. Meng, Y. Gu, C. Liu, J. Sun, T. Zhu, Y. Liu, and K. G. Shin. Battery-aware mobile data service. *IEEE Transactions on Mobile Computing*, 6(16):1544–1558, 2017.

[52] J. Lepkowski, B. Wolfe, and W. Lepkowski. EMI/ESD solutions for the CAN network. In *NSC'05*, 2005.

[53] Dynamic Time Warping. http://web.science.mq.edu.au/~cassidy/comp449/html/ch11s02.html.

[54] D. Montgomery. *Introduction to statistical quality control*. Wiley, 2000.

[55] Stergios Mavromatis and Alexandra Laiou. Safety assessment of control design parameters through vehicle dynamics model. In *RSS'17*, 2017.