Sardino: Ultra-Fast Dynamic Ensemble for Secure Visual Sensing at Mobile Edge

Qun Song[1], Zhenyu Yan[2*], Wenjie Luo[1], and Rui Tan[1]

[1]Nanyang Technological University, Singapore

[2]Chinese University of Hong Kong, HKSAR, China

Abstract

Adversarial example attack endangers the mobile edge systems such as vehicles and drones that adopt deep neural networks for visual sensing. This paper presents *Sardino*, an active and dynamic defense approach that renews the inference ensemble at run time to develop security against the adaptive adversary who tries to exfiltrate the ensemble and construct the corresponding effective adversarial examples. By applying consistency check and data fusion on the ensemble's predictions, Sardino can detect and thwart adversarial inputs. Compared with the training-based ensemble renewal, we use HyperNet to achieve *one million times* acceleration and per-frame ensemble renewal that presents the highest level of difficulty to the prerequisite exfiltration attacks. We design a run-time planner that maximizes the ensemble size in favor of security while maintaining the processing frame rate. Beyond adversarial examples, Sardino can also address the issue of out-of-distribution inputs effectively. This paper presents extensive evaluation of Sardino's performance in counteracting adversarial examples and applies it to build a real-time car-borne traffic sign recognition system. Live on-road tests show the built system's effectiveness in maintaining frame rate and detecting out-of-distribution inputs due to the false positives of a preceding YOLO-based traffic sign detector.

Categories and Subject Descriptors

I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Experimentation, Measurement, Performance

* Part of this work was completed when Zhenyu Yan was with Nanyang Technological University.

Keywords

Adversarial examples, moving target defense, neural networks, edge computing

1 Introduction

Deep neural network (DNN)-based visual sensing is an important perception approach for mobile edge systems such as vehicles and drones. In Apollo [3], which is an autonomous vehicle driving agent, the tasks of recognizing road signs, traffic lights, and lane markers are implemented with DNN-based visual sensing. However, the *adversarial example* attack introduces much concern. Recent research shows that an external adversary can systematically craft minute perturbations added to the inference samples and mislead a DNN to yield absurd results [12]. Readily deployable adversarial examples like paper stickers pasted on the road [4] and traffic sign plate [11] are shown effective against lane detection and traffic sign recognition systems. Thus, the designs of DNN-based visual sensing for safety-critical mobile edge systems should incorporate effective defense against adversarial examples.

Various countermeasures have been proposed, e.g., adversarial training [22, 33], input transformation [30], gradient masking [7], and provable defenses [20, 36]. These approaches build their security upon the assumption that the adversary is ignorant of the defense mechanisms. Such *static* defenses can be breached if the adaptive adversary obtains the details of the defense mechanisms and designs the next-generation attacks [7, 30]. Using an ensemble of multiple distinct DNNs has also been considered as a defense [13]. Specifically, the ensemble uses some rule (e.g., majority vote) to combine multiple DNNs' inference results to generate a final result. Intuitively, it becomes harder for an adversarial example to mislead multiple DNNs than a single DNN. However, the adaptive adversary who has exfiltrated the ensemble can subvert the ensemble-based defense with substantial probabilities (e.g., 52% as shown in §4 of this paper). The static ensemble can be exfiltrated from the mobile edge device's memory or by social engineering against the system designer's employees.

To strengthen mobile edge's visual sensing security against adaptive adversary, we propose using *dynamic ensemble* for active defense under the strategy of *moving target defense* (MTD) [18]. MTD improves system security and increases the difficulties for effective attacks by dynamically

changing the system configurations at run time. In this paper, the ensemble is renewed frequently at run time and unpredictable by the adversary. This approach's effectiveness stems from an observation that the adversarial examples have limited transferability to the DNNs different from those used for attack construction. Its security strength is greatly affected by two aspects. First, larger ensemble sizes and higher renewal rates enhance security strength. Specifically, it is harder for the adversary to construct adversarial examples that can mislead all DNNs of a larger ensemble. Meanwhile, if the ensemble is renewed more frequently, the adaptive adversary has shorter time for exfiltrating the ensemble. Second, higher diversity of an ensemble's DNNs fosters attack detection, because these DNNs tend to produce more diverse classification results for an adversarial example input.

Our earlier work [32] uses dynamic ensembles to counteract adaptive adversarial example attacks. It is based on a simple approach of retraining DNNs using data stored on the mobile edge device, which impedes achieving high-rate ensemble renewal. As reported in [32], it takes 45 minutes on NVIDIA Jetson AGX Xavier to retrain an ensemble of 20 DNNs for traffic sign classification. As the retraining process is very compute-intensive, the ensemble renewal in [32] is performed when the mobile system is idle (e.g., when the car is parked). In the cases of fuel cars, it requires using the car battery to power the lengthy retraining. If multiple task ensembles are renewed, the retraining risks battery overdischarge. In addition, the retraining requires a large training dataset stored on the mobile edge, which is cumbersome.

In contrast to the off-time, infrequent ensemble renewal achieved in [32], this work aims to achieve run-time and high-rate ensemble renewal, which gives two advantages. First, higher renewal rates mean better MTD security. Second, in the context of cars and drones, run-time renewal avoids lengthy battery discharge during parking. However, the run-time renewal and the execution of large ensembles in favor of security should be carefully managed to avoid jeopardizing the visual sensing's real-time performance. In this paper, we design *Sardino*[1] to achieve the goal. Specifically, the design of Sardino consists of the following two aspects.

First, we follow the *HyperNet* concept [28] to design the DNN generator for fast ensemble renewal. The generator is a set of multilayer perceptrons (MLPs) that take random numbers as input and generate the weights of DNNs. A key advantage of Sardino is that the ensemble renewal becomes forwarding the MLPs, which is much faster than DNN training and does not require storing training data on the mobile. We show that generating a DNN for the aforementioned traffic sign classification task on Jetson AGX Xavier only takes 0.1 milliseconds, which is 0.66 and 1.35 million times faster than the two DNN retraining approaches in [25,32]. Owing to the accelerated DNN generation, Sardino achieves per-frame ensemble renewal that renders the highest MTD security.

Second, we design a run-time ensemble size planner, such that the total delay of renewing and executing the ensemble on a mobile edge device shared by other continuing inference tasks meets a soft deadline determined by the sensing frame rate. To this end, the ability to predict the delay is needed but developing this ability is non-trivial. With extensive profiling experiments, we identify that the latest GPU utilization and power usage are two factors affecting the delay. With a decision tree regressor that predicts the delay based on the affecting factors, we maximize in real time the ensemble size in favor of security, subject to the deadline. In other words, Sardino uses the available compute time to increase security.

Adversarial examples can be viewed as a crafted type of out-of-distribution (OOD) inputs that fall out of the training data distribution. In practice, naturally occurring OOD inputs are common. Since Sardino can address adversarial examples under a highly adversarial setting, it can also address the naturally occurring OOD inputs. To demonstrate this, we implement a real-time car-borne traffic sign recognition system based on Sardino. Extensive evaluation including live on-road tests shows the effectiveness of Sardino in meeting soft deadlines and detecting OOD inputs due to the preceding YOLO's [5] false positives in detecting traffic signs.

This paper's main contributions are as follows:

- We propose Sardino for high-rate ensemble renewal to defeat the external adversary's DNN exfiltration as a prerequisite for adversarial example construction. We design a HyperNet to implement the high-rate renewal.
- We conduct extensive evaluation to show Sardino's superior performance in counteracting both adversarial examples and naturally occurring OOD inputs, compared with the existing retraining approaches [25, 32] and the HyperGAN approach [28].
- We design an ensemble size planner to meet a specified soft deadline for ensemble renewal and execution, which is imperative to real-time visual sensing. The design is applicable to the execution on either graphics processing unit (GPU) or central processing unit (CPU).

Paper organization: §2 presents background. §3 overviews Sardino. §4 studies effectiveness of dynamic ensemble. §5 presents the ensemble size planner. §6 presents the car-borne traffic sign recognition system. §7 discusses several related issues. §8 concludes this paper.

2 Background

2.1 Adversarial Examples and OOD Data

Consider a classifier $f(\cdot;\theta)$ with weights θ that classifies an input $\mathbf{x}$ as y, i.e., $f(\mathbf{x};\theta) = y$. An adversarial example $\mathbf{x}' = \mathbf{x} + \delta$, where δ is a perturbation, results in $f(\mathbf{x}';\theta) \neq y$. The magnitude of δ is often minimized to reduce perceptual change. Fig. 1a and Fig. 1b illustrate the impacts of adversarial examples on a convolutional neural network (CNN) trained for traffic sign recognition. In Fig. 1a, the δ is computed by the Carlini and Wagner (C&W) method [10] and added to a clean speed limit sign, leading to a wrong classification of "no heavy vehicle." When the adversary cannot tamper with each pixel, they may construct *adversarial patches* [9]. In Fig. 1b, an adversarial patch [9] is added to a speed limit sign, leading to a wrong classification of "priority road." Such adversarial patches pasted on road can mislead

[1] Sardino is the Esperanto word of sardine. When threatened, sardines form a school that undertakes complicated maneuvers and startling shape changes. The many moving targets of the school create a sensory overload of the predator's visual and electrosensory channels [23].

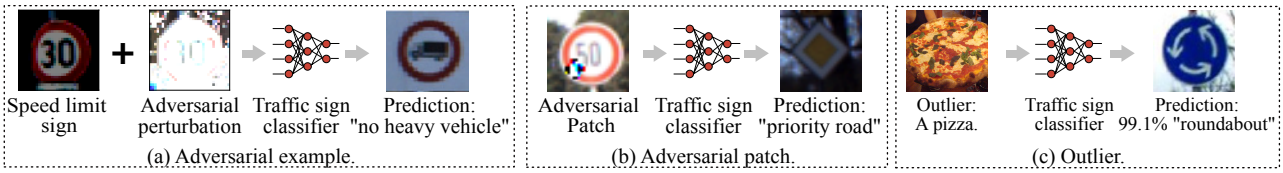


Figure 1. Illustration of the adversarial example and OOD input challenges for DNN-based visual sensing.

Tesla Autopilot to direct a car to the opposite lane [4].

Besides adversarial examples, OOD data or outliers can naturally occur because a training dataset cannot include all future unseen data [14]. A DNN may have high confidence about its wrong classification for an OOD input. As illustrated in Fig. 1c, when the input is a pizza picture, the aforementioned CNN yields a "roundabout" classification result with a high confidence score of 99.1%. Tesla Autopilot has made similar mistakes, e.g., recognize a Burger King sign as a stop sign [27] and moon as yellow traffic light [17]. Although the CNN in Fig. 1c can be retrained to recognize pizzas by adding pizza pictures to the training dataset, this approach cannot cover every possible non-traffic sign object.

This paper aims to improve the resilience of the mobile edge's visual sensing against the issues illustrated in Fig. 1.

2.2 Related Work

The existing countermeasures against adversarial examples are categorized as follows [30]. *Adversarial training* [22,33] includes adversarial examples in the training dataset. The enhanced DNN is secure against the adversarial examples considered during adversarial training. *Transformations* on input such as random resizing/padding, image compression, and noisification are shown effective against the attack. However, they can be defeated by attackers who know the adopted transformation [30]. *Gradient masking* [7] manipulates the victim DNN's gradients to render gradient-based attacks ineffective. However, an adversary aware of the defense can recover the gradients by querying the victim DNN or use other loss functions to construct attack [7]. *Provable defense* [20,36] develops certifiable methods that give lower-bounded defense effectiveness against a certain class of attacks. The above defenses cannot address adaptive adversary. Sardino addresses such adaptive adversary by updating the ensemble at a speed faster than the adversary's exfiltration for the ensemble.

A method to detect outliers is to train DNNs to make highly uncertain predictions for outliers [15]. The work in [31] trains a one-class neural network to detect outliers. The study in [29] trains a generative model and evaluates the likelihood of OOD inputs under that model at run time. Static ensembles generated by HyperNet [28] or other methods [13] have been used to counteract outliers and adversarial examples. Their focuses are on the trade-off between the ensemble size and OOD/attack detection performance, under the non-adaptive adversary setting. As shown in §4, the adversary can construct effective adversarial examples once they obtain the ensemble. The work [28] does not exploit the key advantage of HyperNet, i.e., its ability to renew the ensemble quickly for implementing MTD.

Recent studies aim to improve GPU utilization and processing throughput under the multi-tasking setting. The work [38] schedules multiple DNN tasks in the granularity of GPU kernels for improved GPU utilization. The work [26] achieves acceleration by performing operator fusion and I/O sharing across multiple DNNs. In above studies, the GPU is accessed by the kernels in a round-robin fashion. Alternatively, multiple kernels can run simultaneously on their own GPU cores to enable *spatial sharing*. The work [37] uses this to improve GPU-accelerated network function virtualization. The above studies [26,37,38] focus on task scheduling to maximize processing throughput and do not enforce deadlines. Differently, Sardino plans the ensemble size at run time, aiming at meeting a soft deadline for using the ensemble to process each frame.

3 Approach Overview

3.1 System Model and Objective

We consider a mobile edge computer equipped with a GPU. It runs a general-purpose operating system that orchestrates various sensing tasks. The GPU is shared by the tasks for executing their DNNs. The tasks run simultaneously, take inputs from sensors, and yield the inference results. Among all the tasks, we focus on a *resilient vision task* that needs to have resilience against adversarial examples and OOD inputs, and meet a soft deadline. We view the composite of the remaining tasks as the *background computation*.

We apply dynamic ensemble for the resilient vision task. The ensemble is dynamic in both the number of DNNs of the ensemble (i.e., ensemble size) and the weights of each DNN.

• **Dynamic ensemble size:** We aim to maximize in real time the ensemble size for every frame in favor of resilience subject to the soft deadline. But it is challenging to model the ensemble execution time to enable ensemble size planning.

• **Dynamic DNN weights:** We also aim to renew the weights of each DNN every frame for achieving the highest level of MTD security. The short time of down to milliseconds for completing the renewal presents a main challenge.

We use traffic sign recognition on an autonomous vehicle driving agent to illustrate the system model. The agent receives image frames captured by camera and stores them in a buffer. Each frame fetched from the buffer is processed by an always-running traffic sign detector. When the detector identifies k ($k \geq 1$) traffic sign objects in the current frame, a bounding box containing each of the detected traffic signs is cropped from the frame and passed to the traffic sign classifier. The classifier is executed on each detected sign sequentially. The detected traffic sign may contain adversarial perturbations [11]. Moreover, the traffic sign detector may generate false positives and present outliers to the traffic sign classifier. We view the classifier as resilient vision task and all other tasks collectively as background computa-

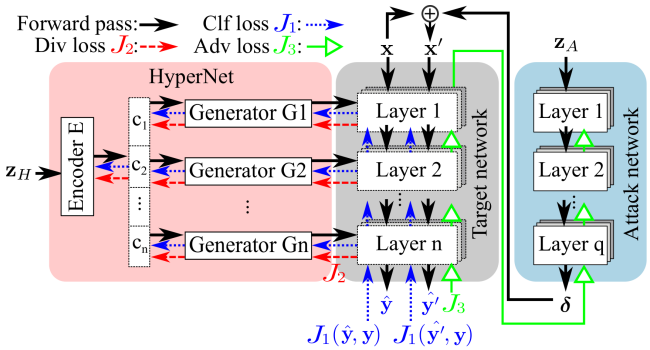


Figure 2. Adversarial learning framework for Sardino.

tion. Suppose the system designer aims to maintain a processing throughput of x frames per second (fps). Thus, the soft deadline for processing each frame is $\frac{1}{x}$ seconds. If the image preprocessing and detection take t_d seconds for the current frame containing k signs, the soft deadline for classifying each sign is $\frac{1/x - t_d}{k}$ seconds. Although t_d and k vary across the frames, they are measurable. Thus, the soft deadline for the classifier, i.e., $\frac{1/x - t_d}{k}$, is variable and known for each frame. The setting for x depends on the vision task's design requirement; it can be also updated at run time according to the vehicle's speed.

In §6, we will apply Sardino to implement the resilient traffic sign recognition. In addition, §7 will discuss how to apply Sardino to protect both the traffic sign detection and recognition simultaneously as two resilient vision tasks.

3.2 HyperNet Design & Adversarial Learning

3.2.1 *Hypernet preliminaries*

HyperNet [28] is a neural network (denoted by $h(\cdot;\phi)$ where ϕ denotes the weights) that generates the weights θ of the *target neural network* denoted by $f(\cdot;\theta)$. Fig. 2 shows the designs of the HyperNet and target network, which is a n-layer CNN, used in this paper. The input to the HyperNet, denoted by $\mathbf{z}_H$, is a random vector sampled from a normal distribution. The $\mathbf{z}_H$ is mapped by an *encoder* E with weights ϕ_E to n *latent codes* $\{c_i | i = 1, 2, \ldots, n\}$. Then, the HyperNet uses n *weight generators* with weights ϕ_G to convert the latent codes to the weights of the target CNN's n layers. The HyperNet's weights are $\phi = \{\phi_E, \phi_G\}$.

3.2.2 *Adversarial learning framework*

In this paper, the goal for training the HyperNet is to generate target networks that are: (1) accurate on clean input samples; (2) diverse in parameter values; and (3) secure against adversarial examples. This subsection presents the designs of the loss functions and the training procedure to meet the three objectives.

For objective (1), we define the *classification loss* (denoted by J_1) by the average cross-entropy loss on the inputs:

$$J_1 = L(f(\mathbf{x}; G(E(\mathbf{z}_H;\phi_E);\phi_G)), y),$$

where $E(\mathbf{z}_H;\phi_E)$ represents latent code, $G(E(\mathbf{z}_H;\phi_E);\phi_G)$ denotes target network's weights generated by the HyperNet, $f(\mathbf{x}; G(E(\mathbf{z}_H;\phi_E);\phi_G))$ is the target network's classification result for input $\mathbf{x}$, and $L(\cdot,\cdot)$ denotes cross-entropy. For objective (2), we design the *diversity loss* denoted by J_2 as:

$$J_2 = \exp(-\mathrm{Var}(G(E(\mathbf{z}_H;\phi_E);\phi_G))),$$

where $\mathrm{Var}(\cdot)$ is the average variance of the generated target network's weights given a batch of $\mathbf{z}_H$. As $\mathrm{Var}(\cdot)$ is not bounded, we apply the exponential function to avoid divergence. For objective (3), inspired by [8], we employ the *adversarial learning* technique [16] to train the HyperNet. Adversarial learning addresses a game between a *defender* that trains a task model to thwart the attacker's objective and an *attacker* that trains an attack model to mislead the defender's task model. In [8], an *attack network* is designed to be the attacker that tries to breach the privacy protection mechanism provided by the HyperNet-based defender. In this work, we design an attack network, as shown in Fig. 2, to be the attacker that tries to generate adversarial examples to mislead the HyperNet-generated target network. The attack network $f_A(\cdot;\theta_A)$ takes random numbers $\mathbf{z}_A$ sampled from a normal distribution and outputs adversarial perturbation δ. The perturbation δ is added to the clean input $\mathbf{x}$, forming the adversarial example $\mathbf{x}'$. The goal of training the attack network is to generate minimized adversarial perturbations that mislead the target network $f(\cdot;\theta)$. Thus, the *adversarial loss* for training the attack network, denoted by J_3, is designed as:

$$J_3 = F(\mathbf{x}')_y - \max_{y_i \neq y} F(\mathbf{x}')_{y_i} + \|\delta\|_2,$$

where $F(\cdot)_{y_i}$ denotes the target network's logit value corresponding to class y_i. Logit value is the output of neural network's last layer before applying the softmax function. $\|\cdot\|_2$ is the Euclidean norm. The attack network and HyperNet are jointly trained, where the attack network is trained to minimize the loss J_3 and the HyperNet is trained to minimize the following composite loss: $\mathbb{E}_{\mathbf{z}_H,\mathbf{z}_A,(\mathbf{x},y)}[J_1] + \mathbb{E}_{\mathbf{z}_H}[J_2]$. Fig. 2 illustrates the training procedure of the adversarial learning.

During the adversarial learning, we do not employ specific methods for crafting adversarial examples (e.g., FGSM, C&W), because doing so usually leads to security improvement specific to the employed attack construction methods only [30]. However, in reality, the attacker's construction method is unpredictable. Our design uses the attack network to generate nondeterministic adversarial examples, which improves Sardino's security against a variety of adversarial examples. This will be demonstrated in §4.2.2.

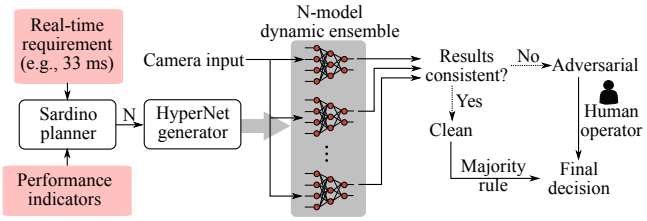


Figure 3. Run-time workflow of Sardino.

3.3 System Design of Sardino

Fig. 3 illustrates the run-time workflow of Sardino. Given a new image frame, Sardino uses the HyperNet to generate a new ensemble of N DNNs to process the input. Before

the generation, Sardino uses an ensemble size planner to determine the largest possible N based on the mobile edge device's performance indicators (i.e., GPU utilization and power usage) and the soft deadline described in §3.1. After the execution of the N DNNs on the image frame, Sardino computes the *output consistency*, which is the percentage of the majority of the DNNs' outputs. If the consistency is larger than a pre-defined threshold T_s, the input is considered clean and the majority of the DNNs' outputs is yielded as the final result. If the output consistency is smaller than T_s, the input is considered adversarial or OOD, and will be classified by a human operator for final decision. In summary, based on the mobile edge device's run-time performance indicators, Sardino adapts the ensemble size N to meet the soft real-time requirement, then generates and executes the dynamic ensemble to process the incoming image frame.

3.4 Threat Model

The key objective of dynamic ensemble is to prevent the external adversary from obtaining the ensemble in use. Our test shows that, if the adversary obtains the ensemble in use, the adversarial example constructed by the approach in [21] can mislead the ensemble-based attack detection described in §3.3 with probabilities of 52% and 37% when the false positive rates are 1.1% and 5.2%, respectively. Thus, high-rate ensemble renewal is key to MTD security. In this paper, we consider the external adversary who obtains some critical static information about the dynamic ensemble. We consider two kinds of static information: (1) training dataset and (2) the HyperNet itself.

• **Adversary with training dataset:** The training dataset can be acquired by feeding massive unlabeled input samples to the black-box target DNN (e.g., its binary executable) and obtaining the corresponding labels. Then, the adversary can train a *surrogate DNN* with the obtained training dataset and construct adversarial examples against it.

• **Adversary with HyperNet:** Since the HyperNet is static information, it can be obtained by the adversary under the scenario of advanced persistent threat (APT), e.g., conducting social engineering against employees of the car factory. Then, the adversary can generate *surrogate ensemble* using the obtained HyperNet and follow the approach in [21] to craft adversarial examples against it.

In this paper, we show that HyperNet can achieve per-frame ensemble renewal capability. However, for a specific application, the system designer can decide the ensemble renewal rate that affects the level of MTD security. For example, if an external adversary with limited access to the ensemble is considered, the system may adopt per-night ensemble renewal to save computing resources. However, if the system considers an insider adversary that can frequently access the generated ensemble, per-frame ensemble renewal may be adopted to achieve the highest level of MTD security. Note that the internal adversary who has broken into the system and can obtain each renewed ensemble regardless of renewal rate is out of the scope of this paper, since the internal adversary should directly subvert the whole system rather than resort to adversarial examples. Besides adversarial examples, we also consider the naturally occurring OOD data. In this paper, the ensemble renewal rates are the same for adversarial and OOD inputs, i.e., per-frame renewal. In §7, we will discuss how to achieve more efficient ensemble renewal by differentiating adversarial and OOD inputs.

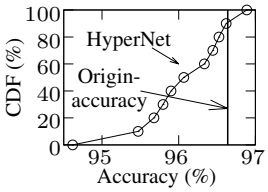


Figure 4. CDF of 100,000 HyperNet-generated DNNs' accuracy on GTSRB dataset.

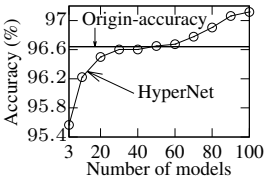


Figure 5. Test accuracy vs. size of HyperNet-generated ensemble (Dataset: GTSRB).

4 Effectiveness of Dynamic Ensemble

4.1 Profiling Experiment Setup

We conduct experiments on NVIDIA Jetson AGX Xavier with an octa-core 2.26GHz ARM CPU, a 512-core Volta GPU, and 16GB RAM. It runs Linux4Tegra. We write code in Python using PyTorch 1.4.0. Most experiments are based on German Traffic Sign Recognition Benchmark (GTSRB) dataset [34] with over 50,000 image samples in 43 classes. To evaluate outlier detection, we use the MNIST [1] and notMNIST [2] datasets. MNIST is a 10-class set of grayscale images of handwritten digits from 0 to 9; notMNIST is a 10-class set of grayscale images of letters A to J, which is often used for studying outlier detection [28]. The target CNN has two convolutional layers with 32 5x5 filters with rectified linear unit activation, max pooling, and a dense layer with width equal to the class number. HyperNet's encoder has two 64-neuron dense layers and a dense layer with 64x3 neurons. The encoder's input is a 256x1 Gaussian random vector. The encoder is followed by three weight generators, each of which has two 64-neuron dense layers and one dense layer with identical width as the output layer.

4.2 Profiling Experiments and Results

4.2.1 HyperNet ensemble's classification accuracy

The curve in Fig. 4 is the cumulative distribution function (CDF) of the test accuracies of 100,000 HyperNet-generated DNNs. The vertical line labeled *origin-accuracy* is the test accuracy of the original DNN trained from the GTSRB dataset following the design in [34], which is 96.6%. The accuracies of HyperNet-generated DNNs are within 94.6% to 96.9%, showing that HyperNet can generate quality DNNs. We also investigate the accuracy improvements of fusing the outputs of multiple HyperNet-generated DNNs using the majority rule (called HyperNet ensemble). As shown in Fig. 5, the HyperNet ensemble's accuracy increases with ensemble size N. In particular, compared with the accuracy when no fusion is applied, the accuracy improvement is up to 1% when N is 3. When N is 100, the improvement is up to 2.5% and the ensemble's accuracy is 0.5% higher than the origin-accuracy.

4.2.2 Performance in thwarting adversarial examples

We evaluate the attack thwarting performance against the two types of adversary described in §3.4. We consider the

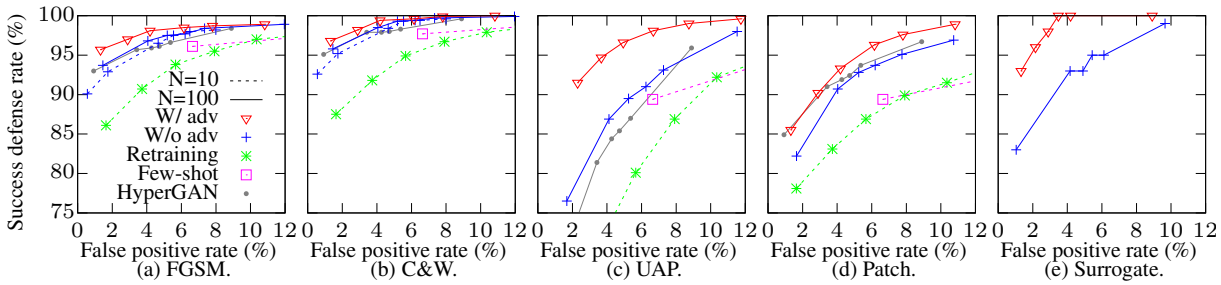


Figure 6. SDR vs. FPR in thwarting various types of adversarial examples, i.e., FGSM [12] in (a), C&W [10] in (b), UAP [24] in (c), Patch [9] in (d), and that against surrogate ensemble [21] in (e). The lines labeled with "W/ adv" and "W/o adv" are for *HyperNet-ensemble* with and without adversarial learning. Legends for (b)-(e) are same as (a).

evaluation for five variants of the ensemble-based detector, namely, *retraining-ensemble* [32], *few-shot retraining-ensemble* [25], *HyperGAN-ensemble* [28], and *HyperNet-ensemble* proposed in this paper with and without adversarial learning. In *retraining-ensemble*, each DNN is trained from scratch with random initialization. Each DNN of the *few-shot retraining-ensemble* is obtained by the few-shot domain adaptation approach in [25] that adapts a base model trained with a big source-domain data subset to the target domain using a small data subset containing 7 samples for each class. *HyperGAN-ensemble* is generated by HyperGAN [28] that trains a generator to transform random numbers into target network's weights together with the help of a discriminator to promote the diversity of the generated weights.

We follow the workflow in Fig. 3 with N fixed to implement the defense. We assume that the detected adversarial examples are classified by the human operator without errors, since adversarial perturbations are crafted to be visually imperceptible. We measure the *successful defense rate* (SDR), which is the percentage of the adversarial examples failing to mislead the system. We use the false positive rate (FPR) to characterize the unnecessary overhead incurred to the human operator.

■ **Adversary with training dataset:** There are two types of adversarial examples [30]: *input-specific* perturbation is crafted against a specific clean sample, while ideally, *universal* perturbation is effective against any clean sample. We consider two input-specific attacks, which are FGSM [12] and C&W [10], and two universal attacks, which are universal adversarial perturbation (UAP) [24] and adversarial patch (Patch) [9]. From our measurements, the FGSM, C&W, UAP and Patch attacks can mislead the surrogate DNN on 97.4%, 100%, 45% and 33.1% of clean test samples.

Fig. 6 shows SDR versus FPR in thwarting adversarial examples. From Figs. 6a-d, *HyperNet-ensemble* with adversarial learning produces the highest curves, i.e., the best trade-off between the security and the overhead incurred to human. An intuitive explanation for the better attack thwarting performance of the *HyperNet-ensemble* over the *HyperGAN-ensemble* is that HyperGAN only increases the diversity of the generated weights and does not consider adversarial examples during training. When FPR is around 2%, SDRs of the *HyperNet-ensemble* with adversarial learning are 96.3%, 97.5%, 91.5%, and 88.2% against the four attacks, respectively. For a certain attack, when N increases, the curve becomes higher. This indicates that larger N settings are beneficial to the effectiveness of defense.

We also compare our approach with an adversarial training approach [22] in terms of defense performance. Adversarial training is considered the state-of-the-art defense [30]. The top-ranked defenses in the latest adversarial defense leaderboards [6] are based on adversarial training. The adversarial training approach includes adversarial examples constructed using the project gradient descent method [19] into the training dataset. It achieves SDRs of 35.7%, 25%, 85%, and 68% against the four attacks. Its poor defense is due to that adversarial training's effectiveness is specific to the considered type of adversarial examples [30]. Differently, the HyperNet hardened by adversarial learning with nondeterministic adversarial examples shows better generalizable security against various types of adversarial examples.

■ **Adversary with HyperNet:** We evaluate the SDRs of the dynamic ensembles generated by the same HyperNet. Fig. 6e shows that the SDRs are much higher than those without MTD in which the adversary obtains the ensemble in use as mentioned in §3.4 (i.e., 100% - 52% = 48% and 100% - 37% = 63% when the FPR is 1.1% and 5.2%, respectively). The SDR for the HyperNet with adversarial learning is higher than that without adversarial learning.

The above results suggest that the MTD of preventing the adversary from obtaining the ensemble in use is effective in counteracting adversarial example attacks. HyperNet-based MTD security is further enhanced with adversarial learning when the adversary constructs adversarial examples against the surrogates based on static information of the defense.

4.2.3 Outlier detection performance

We evaluate the outlier detection performance of the ensemble-based detectors and a baseline outlier detector described in [14], which uses a single DNN and declares an outlier if the maximum of the softmax probabilities of all classes is below a threshold. We perform training using MNIST. During testing, we use notMNIST to assess the true positive rate of outlier detection and use MNIST to assess the false positive rate. Fig. 7 shows the receiver operating characteristic (ROC) of various outlier detectors. By default, $N = 20$. To generate ROCs, we vary the consistency threshold T_s from 50% to 100% for ensemble-based detectors and vary the softmax probability threshold from 90% to

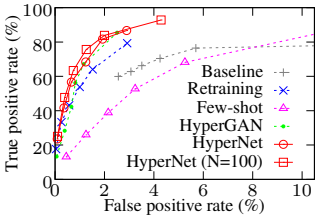


Figure 7. Outlier detection (Dataset: MNIST & notMNIST; $N = 20$).

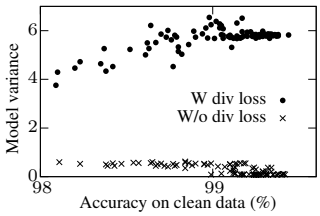
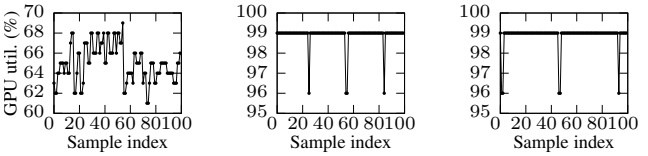
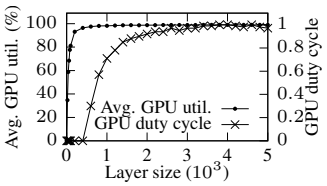


Figure 8. Weights' variance of the ensemble DNNs (Dataset: MNIST).

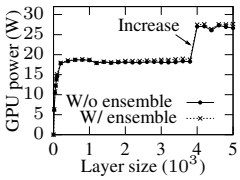


(a) Layer size = 100. (b) Layer size = 4000. (c) Layer size = 5000.

Figure 9. GPU utilization of background computation.



(a) Without ensemble



(b) Without/with ensemble

Figure 10. Background GPU utilization and power usage vs. the layer size of the background DNN. "W/o ensemble" means only the background computation is running. "W/ ensemble" means the background computation and the ensemble generation and execution are running simultaneously.

100% for the baseline detector. The HyperNet-ensemble's ROC curves for $N = 20$ and $N = 100$ are the highest in the plot, suggesting that HyperNet-ensemble outperforms other detectors. The ROC curve for $N = 100$ is higher than that for $N = 20$, suggesting that larger ensemble size is beneficial to outlier detection.

4.2.4 *Diversity of HyperNet-generated DNNs*

Fig. 8 compares the HyperNets trained with or without the diversity loss J_2. Each point corresponds to a HyperNet-ensemble with $N = 32$. The x-axis is the ensemble's accuracy on the MNIST clean samples. We calculate the variance for each weight parameter across all DNNs of an ensemble. The y-axis is the average of all weights' variances. We can see that the diversity loss J_2 diversifies the generated DNNs. Besides, the superior outlier detection performance of HyperNet in Fig. 7 demonstrates that HyperNet-generated ensemble is more diverse than the ensembles generated by other baselines. The intuition is that, given an outlier as input, a more diverse ensemble generates a more diverse set of predictions. Thus, evaluating ensemble diversity via outlier detection performance is common in the literature [28].

4.3 Summary of Profiling Results

From §4.2, we can draw the following observations. First, HyperNet generates diverse DNNs that achieve high accuracy on clean examples. Second, HyperNet-ensemble outperforms adversarial training [22], retraining-ensembles [25, 32], and HyperGAN-ensemble [28] in counteracting adaptive adversarial example attacks based on certain static information of the defense. Third, HyperNet-ensemble outperforms the OOD detection approaches based on softmax probability [14], retraining-ensembles [25, 32], and HyperGAN-ensemble [28]. Lastly, HyperNet-ensemble's accuracy on clean examples and security/resilience against adversarial examples/outliers increase with N.

5 Run-Time Planning of Ensemble Size

From §4, it is desirable to maximize N subject to the soft deadline of the resilient vision task. The key is the ability to predict the ensemble generation and execution time for any N in the presence of time-varying background computation. The prediction should have low compute overhead. With this ability, we can find the maximum N meeting the deadline.

The general problem of scheduling GPU computing tasks to meet deadlines is challenging due to the non-preemptive nature of GPU kernels. Recent studies [37, 38] enable concurrent executions of multiple kernels and schedule the kernels to maximize processing throughput. Although solutions to the general problem are still lacking, our reduced problem of predicting the ensemble generation and execution time in the presence of uncoordinated background computation may have an effective solution if we can identify the major factors correlated with the ensemble generation/execution time and then apply supervised learning to characterize the correlations. Following this method, we conduct measurements to identify the correlated factors in §5.1; we design and evaluate the ensemble latency predictor in §5.2 and §5.3, respectively.

5.1 Identifying Latency-Correlated Factors

We set up a continuous DNN inference process as the *background computation*. As convolution is compute-intensive, we adjust the number of neurons of the background DNN's every convolutional layer (referred to as *layer size*) to affect the intensity of the background computation. Fig. 9 shows the instantaneous GPU utilization traces on AGX Xavier when the layer size is 100, 4,000, and 5,000. Note that we use the `tegrastats` utility to measure the GPU utilization and power usage. When the layer size is 100, the GPU utilization fluctuates at 65%. When the layer size is 4,000 and 5,000, the GPU utilization mostly remains at 99%. To understand the impact of the layer size on GPU utilization and power usage, we use the average value and duty cycle to characterize a GPU utilization trace obtained under a certain layer size. The duty cycle is the percentage of time at which the GPU utilization is higher than 99%. Fig. 10 shows the GPU utilization and power when the layer size varies from 10 to 5,000. From Fig. 10(a), GPU utilization's average and duty cycle increase smoothly with the layer size. The curve labeled "Without ensemble" in Fig. 10(b) shows a step increase of the GPU power when the layer size increases to 4,000. It can be caused by the increase of active stream processors to compute more neurons. The results in Fig. 10 imply that GPU utilization and power depict different aspects of remaining GPU computing capability.

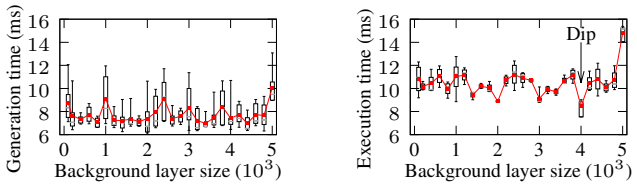


Figure 11. 100-DNN ensemble generation/execution time vs. background computation layer size. (Grey line represents median; dot represents mean; box represents 20%/80% percentiles; whiskers represent max/min. Same style is applied for all error bars in this paper.)

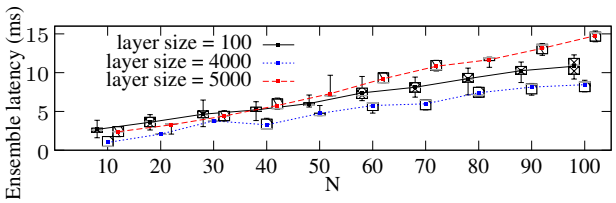


Figure 12. Ensemble latency vs. ensemble size.

Fig. 11 shows how the background computation affects the delays for ensemble generation and execution. Both delays are relatively stable when the background layer size is up to 4,800. Both increase saliently when the background layer size increases from 4,800 to 5,000, which can be caused by the contention between background computation and ensemble generation/execution. In Fig. 11, the ensemble execution time has a dip when the background layer size is 4,000. A potential reason is that, at this point, the GPU increases active stream processors as indicated by the sudden increase of GPU power in Fig. 10(b). From Fig. 11, it takes about 10 ms to generate 100 DNNs using HyperNet. In contrast, the retraining approach in [32] and the few-shot retraining approach in [25] require 45 and 22 minutes to generate 20 DNNs on AGX Xavier. Thus, HyperNet achieves 0.66 to 1.35 million times acceleration in per-DNN generation.

Fig. 12 shows the impact of N on the ensemble execution time in the presence of background computation. Under a certain background layer size, the ensemble latency increases linearly with N in general. When the layer size varies, the line of ensemble latency versus N changes.

The above results show that the background GPU utilization and power, and N are three factors correlated with the ensemble latency. From the near-linear relationships shown in Fig. 12, simple models may effectively characterize the impact of these three factors on the ensemble latency.

5.2 Design of Ensemble Latency Predictor

We choose the background GPU utilization and power usage traces sampled by `tegrastats` at 100 Hz in the past 100 ms before the start of the ensemble generation and N as the three inputs to the machine learning model. The output is the predicted ensemble latency. We consider three candidate machine learning models and the evaluation in §5.3 and §6 will recommend a good choice. (1) **Decision tree (DT)** predicts the ensemble latency using a set of if-then-else decision rules. The tree structure and the decision rules associated with the tree nodes are learned from the training data. Fig. 13 shows a DT and its top three layers learned from data collected from AGX Xavier. The root is the entry. Each node compares an element of the input with a threshold to decide which branch to proceed with. The tree leaves are the output nodes associated with predicted ensemble latency. (2) **Linear regression (LR)** predicts ensemble latency by a weighted sum of all inputs, where the weights are learned from the training data. (3) **DNN** uses two 200-neuron hidden layers with ReLU to predict the ensemble latency.

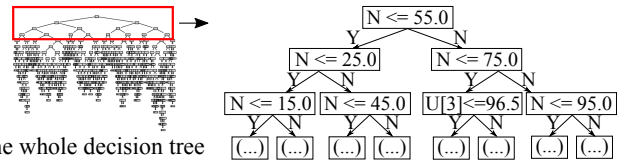


Figure 13. Visualization of the decision tree for ensemble latency prediction on Jetson AGX Xavier. "U[3]" means the background GPU utilization collected at 30 ms before the start of the ensemble generation.

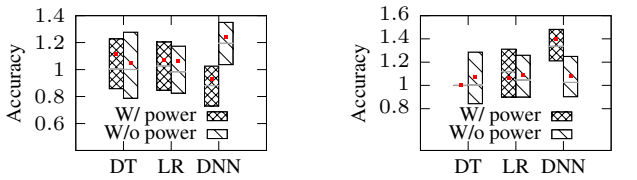
A trained predictor is specific to the mobile edge device's hardware and software configurations. The training needs a profiling process to collect training data. This overhead is acceptable since the profiling can be automated. Moreover, the profiling and training are only performed by expert designers during system design and software updates.

Multi-core CPU shares some similarities with the many-core GPU in terms of parallelism. Our design is also applicable to the CPU-only devices. Although CPU-only device is not suitable for real-time visual sensing and thus not our focus, we will briefly evaluate our design on CPU in §6.

Lastly, we present how a trained predictor is used at run time. For each image input, Sardino queries the latest GPU utilization and power usage traces, predicts the ensemble latency for every candidate N setting, and chooses the maximum setting meeting the deadline. DT and LR predictors can be executed by CPU due to their low compute overheads.

5.3 Evaluation of Latency Predictor

We evaluate the prediction models described in §5.2. We collect 500 data points on AGX Xavier by varying the layer size of the background process from 100 to 5,000 with a step size of 100 and the ensemble size N from 10 to 100 with a step size of 10. We set the background process always running and start the ensemble generation and execution at random time instants. We shuffle and split the collected data into training and testing data with a ratio of 4:1. The performance of the machine learning models is evaluated on the test data using two assessment metrics: (1) accuracy, defined as the ratio between the predicted and true values of ensemble latency, i.e., $\frac{T_{pred}}{T_{true}}$, and (2) root mean squared error (RMSE). The accuracies are shown in Fig. 14(a). The box plots labeled with "W/ power" are the results of the machine learning models with N, GPU utilization trace, and GPU power usage trace as input; those labeled with "W/o power" are the results of the models designed with N and GPU utilization trace as input. Table 1 shows the three models' RMSEs. The



(a) Jetson AGX Xavier (b) Jetson Nano

Figure 14. Ensemble latency prediction accuracy of the three models designed with or without GPU power usage trace as part of input. Accuracy is defined as the ratio between predicted value and true value.

Table 1. RMSE (ms) of ensemble latency prediction.

	Jetson AGX Xavier		Jetson Nano	
	W/ power	W/o power	W/ power	W/o power
DT	1.59	1.61	2.60e-6	16.70
LR	1.09	1.18	14.60	23.31
DNN	1.41	1.36	15.58	16.19

inclusion of GPU power improves the prediction accuracy for DT and LR.

Then, we evaluate whether the machine learning approach works for NVIDIA Jetson Nano, which is a less powerful platform with a quad-core Cortex-A57 CPU, a 128-core Maxwell GPU, and 4GB RAM. The results are shown in Fig. 14(b) and Table 1. The inclusion of GPU power into input also improves prediction accuracy. DT outperforms the other two models.

From Fig. 14 and Table 1, the three prediction models achieve similar accuracy on AGX Xavier; the DT outperforms significantly on Nano. In terms of compute overhead, DT's time complexity is linear to its depth, which is sublinear to the number of decision variables. Compared with LR and DNN that have linear and super-linear time complexities, DT is more efficient and preferred. On both Jetson boards, DT achieves sub-2 ms RMSEs. This accuracy is acceptable for meeting the soft deadlines of tens of milliseconds. The prediction errors will cause jitters, which will be evaluated in §6.

DT's superior performance is because the hierarchical structure of DT better captures the priority hierarchy of the affecting factors in determining the ensemble latency. For instance, in Fig. 13, the top layers of the tree make decisions based on N. The conditions for the latest GPU utilization appear at lower layers. These match with the observations from Fig. 12 that (1) N determines the range of the ensemble latency value and (2) the background computation intensity determines which line to follow and the exact value. In contrast, LR is short of capturing such non-linear priority hierarchy. Although DNN can capture sophisticated patterns, it needs a rich training dataset. A possible reason for DNN's degraded accuracy on Nano than AGX Xavier is that the more GPU contention on the less powerful Nano increases the pattern complexity and thus requires more training data.

Note that other factors such as environment temperature and processor cache hit rate may also generate impact on the ensemble latency. Including these factors to the machine learning model's input may further improve prediction accuracy. We leave this further improvement to future study.

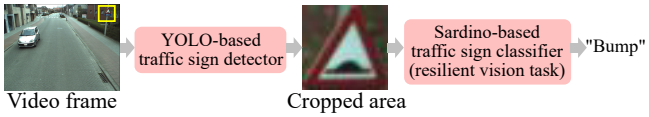


Figure 15. The pipeline of the traffic sign recognition.

6 Real-Time On-Car Traffic Sign Recognition

6.1 System Implementation

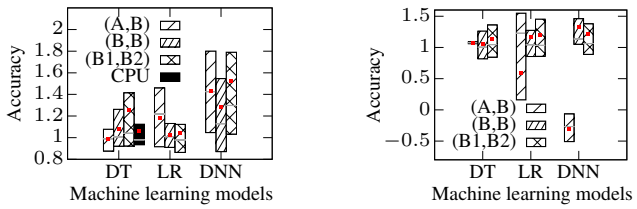
We apply Sardino to build a real-time car-borne traffic sign recognition system. A traffic sign recognition system usually consists of the sign detection and classification phases [39]. The detector identifies and locates traffic signs in an incoming frame captured by a car-mounted camera. The detected traffic sign is then interpreted by the classifier. We implement a Belgian traffic sign recognition system based on the publicly available KUL Belgium Traffic Signs Dataset [35], which has two datasets for traffic sign detection (BelgiumTSD) and classification (BelgiumTSC) and four recorded videos.

Fig. 15 illustrates the processing pipeline of our implementation. We extend YOLO and YOLO-tiny [5] as the traffic sign detector. YOLO is a DNN-based object detection system that achieves good accuracy and latency performance among various systems and YOLO-tiny is a simplified version with fewer layers and same class number. We use BelgiumTSD to augment the original training set for YOLO and YOLO-tiny and retrain them to achieve mAP@0.5 of 58.9 and 33.7, respectively. Note that mAP@0.5 is the mean Average Precision when a prediction is considered positive if intersection over union (IoU) is no smaller than 0.5. The mAP@0.5 scores for the original YOLO and YOLO-tiny are 57.9 and 33.1 [5]. The extended YOLO and YOLO-tiny can detect the Belgian traffic signs as the class "traffic sign" and use a bounding box to contain each detected sign. The bounding box area is cropped from the original frame, resized, and passed to the Sardino-based traffic sign classifier. Other objects detected are not processed. We train Sardino's HyperNet using the 62-class BelgiumTSC training set. The HyperNet-generated DNNs achieve an average accuracy of 96.1% on the BelgiumTSC testing set, which is comparable to the accuracy of 97.0% reported in [35]. We deploy the extended YOLO and YOLO-tiny on Jetson AGX Xavier and Jetson Nano, respectively. On AGX Xavier, YOLO's processing throughputs are 82, 55, and 42 fps when the input frame sizes are 320x320, 416x416, and 512x512, respectively. On Nano, YOLO-tiny's processing throughputs are 37, 26, and 22 fps for the three input frame sizes.

6.2 Performance Evaluation

6.2.1 Ensemble latency prediction

The YOLO on the AGX Xavier and YOLO-tiny on Nano are viewed as the background computation. We set the frame sizes to be 320x320, 416x416, and 512x512 to obtain different background computation intensities. For each frame size, we vary N from 10 to 100 with a step size of 10. We repeat each setting for 20 times and collect 600 data points for



(a) AGX Xavier, YOLO (b) Nano, YOLO-tiny

Figure 16. Ensemble latency prediction on car-borne traffic sign recognition.

Table 2. RMSE (ms) of ensemble latency prediction.

Board	Jetson AGX Xavier			Jetson Nano		
(train,test)	(A,B)	(B,B)	(B1,B2)	(A,B)	(B,B)	(B1,B2)
DT	1.73	1.18	1.25	1.49	11.01	11.64
LR	1.77	0.69	0.88	14.08	8.09	12.75
DNN	2.91	1.96	2.82	4.52	8.47	20.23

*The minimum RMSE among the three models is underlined.

evaluation on each platform. Fig. 16 shows the ensemble latency prediction accuracy and Table 2 shows RMSEs. In particular, we evaluate the transferability of the trained prediction models across different background computations. Label (A,B) means the model is trained with the customized background computation described in §5.1 and tested using the YOLO background computation with all frame size settings; label (B,B) means the model is trained and tested using the YOLO background computation; label (B1,B2) means the model is trained using the YOLO background computation with 320x320 and 512x512 frame sizes and tested with 416x416 frame size. From Table 2, DT and LR achieve similar RMSEs. However, on Nano with setting (A,B), LR performs poorly. The results also show that DT exhibits good transferability across different background computations. In the rest of this paper, we use DT.

To evaluate whether DT is applicable to CPU-only devices, we run YOLO background computation and ensemble generation/execution on the CPU of AGX Xavier. The inputs to DT are N, background CPU utilization, and CPU power usage. The DT is trained using data traces when the system operates on 320x320 and 512x512 frame sizes and tested on 416x416 frame size. The prediction accuracy, as shown in Fig. 16(a) by the box labeled "CPU", is comparable to those on GPU. However, YOLO (without Sardino) only achieves a throughput of 0.05 fps on CPU. Thus, CPU-only devices are ill-suited for real-time visual sensing although the DT is still applicable.

6.2.2 *Real-time performance of Sardino*

The total processing time for each frame consists of: (1) YOLO detection time, which depends on the frame size; (2) ensemble generation time; and (3) ensemble execution time. YOLO's detection time is measured at run time. Then, we follow the method presented in §3.1 to determine the soft deadline and pass it to the ensemble size planner. Fig. 17 shows the planned N and per-frame processing time traces, where the target per-frame processing time is 40 ms. We repeatedly play a video to introduce disturbances during the experiments. We can see that Sardino frequently adjusts N.

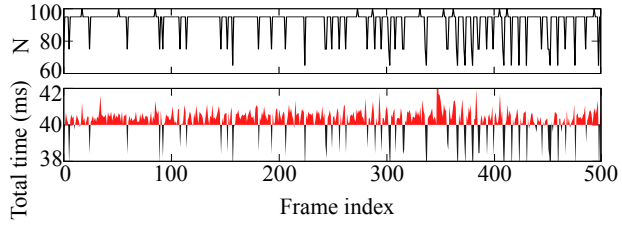
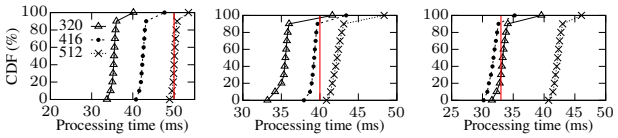


Figure 17. Traces of planned N and per-frame total processing time. Frame rate: 25 fps; deadline: 40 ms.



(a) Deadline: 50ms (b) Deadline: 40ms (c) Deadline: 33ms

Figure 18. CDF of per-frame processing time under three deadline settings (represented by vertical lines) and three frame size settings of 320×320, 416×416, and 512×512 (different CDFs for different frame sizes).

The per-frame processing time fluctuates at 40 ms. The average per-frame processing time is 40.43 ms; the maximum deviation is about 2 ms, consistent with the error level of the DT predictor.

Fig. 18 shows the CDFs of the per-frame processing time on AGX Xavier under various settings of deadline and YOLO frame size. When the specified deadline is 50 ms for processing a 20 fps frame stream, from Fig. 18(a), the deadline can be always met for frame sizes of 320x320 and 416x416, and mostly met for frame size of 512x512. For the former two cases, the allowed time of 50 ms is not fully utilized, because we set an upper bound of 100 for N. Larger settings for N often lead to memory exhaustion. When the specified deadline is 40 ms, the system with 320x320 and 416x416 frame sizes can still largely meet the deadline. When the frame size is 512x512, YOLO's detection time is very close to the 40 ms deadline. The deadline may be exceeded even if the minimum setting $N = 3$ is chosen. When the specified deadline is 33 ms, the system with 512x512 frame size completely misses the deadline because the uncontrollable YOLO detection time already exceeds the deadline. The system with the other two frame sizes can still largely meet the deadline. Thus, by properly choosing settings that will not overwhelm the system, Sardino can maximize N while meeting required frame rate.

6.2.3 *Resilience against OOD data*

The false positives of the YOLO-based traffic sign detector are naturally occurring OOD data for traffic sign classifier. Fig. 19 shows six examples of such false positives. We measure Sardino's true positive rate in detecting OOD data using 500 YOLO's false positives in processing the four test videos of the KUL dataset. Fig. 20 shows the ROCs of the Sardino and the baseline detector described in §4.2.3. The baseline detector is ineffective. This is because YOLO's internal detector only yields objects (including false positives) detected with high confidence. Fig. 20 shows that Sardino

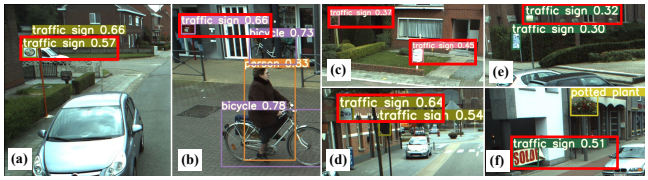


Figure 19. Some OOD samples (highlighted by red frames) caused by YOLO's false positives in detecting traffic sign. (a) pattern on a car; (b) mailbox; (c) plant & mailbox; (d) shop signboard; (e) street light; (f) banner.

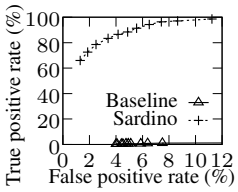


Figure 20. Outlier detection performance.

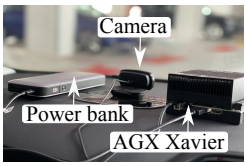


Figure 21. Setup installed under a car's front windshield.

advances the resilience of traffic sign classification against YOLO's false positives.

We do not measure Sardino's performance in thwarting adversarial examples in this application. This is because, the numeric experiment results in §4.2.2 based on real traffic sign data and ideal attack settings (e.g., pixel-level perturbation capability) characterize the lower bound of Sardino's attack thwarting performance. The results in §4.2.2 have already shown the superior performance of Sardino. Differently, the numeric experiments in §4.2.3 on OOD detection are based on the simplistic handwritten digit recognition task for illustration only. Thus, in this section, we focus on evaluating Sardino's OOD detection performance for this real-world application of traffic sign recognition.

6.2.4 Stress tests on live roads

The test videos in the KUL dataset have a limited number of objects in the camera's field of view. To evaluate the performance of Sardino under more challenging settings, we conduct live tests on the busy roads of Singapore. To stress-test Sardino's real-time performance, we let Sardino process each object detected by YOLO, not limited to traffic sign objects. As shown in Fig. 21, we install our system under a car's front windshield. The AGX Xavier is connected with a Logitech C525 camera via USB and powered by a portable battery. We drove the car for multiple runs, each lasting for about 30 minutes. Each run covers various road types, including campus, locals, and expressways. We test three video settings in terms of frame size and rate: ❶ 320x320 at 30 fps; ❷ 416x416 at 24 fps; ❸ 512x512 at 20 fps.

Fig. 22 shows the number of detected objects k, the planned N, and the per-frame total processing time in about 17 seconds during a run. The number of detected objects in a frame can be up to 9. Sardino frequently adjusts N to maintain the per-frame processing time at 33 ms. The average per-frame processing times under the three video settings are 33.0 ms, 40.4 ms, and 50.5 ms, respectively. Although

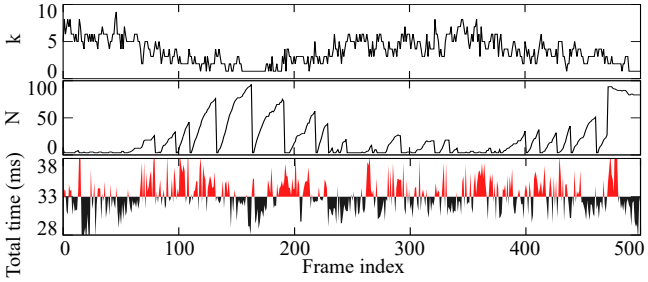


Figure 22. Number of detected objects (k), planned N, and per-frame total processing time under setting ❶.

the system has jitters, the average processing times are very close to the setpoints of 33 ms, 41.7 ms, and 50 ms.

7 Discussions

Currently, Sardino uses HyperNet to generate target networks with homogeneous architecture. It is interesting for future research to study how to further improve Sardino's performance by generating neural networks with heterogeneous model architectures. One possible approach is to employ multiple HyperNets that generate target networks with heterogeneous architectures. In addition, the target networks generated by Sardino have relatively small sizes. To support the sensing tasks addressed by larger-scale neural networks, Sardino may first apply model compression techniques, e.g., knowledge distillation, to represent the model more efficiently and alleviate the costs of execution on resource-constrained edge devices. Beyond the HyperNet technique employed in this paper, Sardino may also apply techniques such as the Monte Carlo Dropout and Bayesian neural networks that sample different neural networks directly.

A possible concern is that the ensemble's mutability may impede post-incident faulty analysis in the context of autonomous driving, because storing each renewed ensemble incurs high storage overhead. In fact, we only need to store the random seeds fed to HyperNet, which introduces only 5.5MB/hour storage overhead. With the seeds, we can trace back to the ensembles in fault analysis. Another related concern is that the inference accuracy of the renewed ensembles is not validated. To mitigate this concern, a validation dataset can be used to test the inference accuracy of each renewed ensemble at run time. Only the ensembles passing the test will be commissioned. On Jetson AGX Xavier, it takes about 7.2 seconds to complete the validation of a 100-DNN ensemble using 4,410 samples. Therefore, with this validation process, Sardino cannot achieve the per-frame ensemble renewal. However, compared with the off-time retraining-based approaches [25, 32], the run-time ensemble renewal at a rate of every 7.2 seconds provides much stronger MTD security and avoids battery over-discharge as discussed in §1.

In the traffic sign recognition application, the traffic sign detector can be also vulnerable to adversarial example attacks. Sardino can be easily extended to support multiple pipelined resilient vision tasks (e.g., traffic sign detector and classifier). Specifically, for each image frame, we can allocate the remaining processing time of $\frac{1/x - t_d}{k}$ calculated using

the approach described in §3.1 to the multiple resilient vision tasks by following a pre-defined policy (e.g., equal split) and use the respective ensemble size planner for each resilient vision task. This paper aims to fully utilize the remaining computing resources to improve the security of a resilient task. How to schedule the computing resources to balance the security of the resilient task and the performance of other tasks is an interesting problem for future study.

In this paper, we do not discriminate the adversarial examples and OOD data for the ensemble renewal of Sardino. This is because the detection of whether an input belongs to adversarial example or OOD data is still an open question. Future research can develop detection mechanism to differentiate between adversarial examples and OOD data as the first step, such that the frequencies of the subsequent ensemble renewal can be adjusted adaptively. For example, the ensemble may be updated more frequently if the input is detected as an adversarial example and less frequently if the input is detected as OOD data.

8 Conclusion

This paper presented Sardino, a HyperNet-based ultra-fast MTD approach for visual sensing at edge. Sardino generates quality ensembles that provide good classification accuracy on clean data and improved resilience against adversarial examples and naturally occurring OOD inputs. With the ultra-fast ensemble renewal and ensemble generation/execution time prediction, Sardino continuously updates the ensemble size such that each video frame can be processed with a new ensemble within a soft deadline, rendering the highest level of MTD security against adaptive adversarial example attacks. We use Sardino to build a real-time carborne traffic sign recognition system and extensively evaluate its performance.

Acknowledgments

The authors wish to thank the anonymous reviewers and shepherd Dr. Olga Saukh for providing valuable feedback on this work. This research is supported by the National Research Foundation, Singapore and National University of Singapore through its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) office under the Trustworthy Computing for Secure Smart Nation Grant (TCSSNG) award no. NSOE-TSS2020-01.

9 References

[1] http://yann.lecun.com/exdb/mnist.

[2] http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html.

[3] Apollo. https://github.com/ApolloAuto/apollo.

[4] Security research of tesla autopilot. https://youtu.be/6QSsKy0I9LE.

[5] Yolo: Real-time object detection. https://pjreddie.com/darknet/yolo/.

[6] Adversarial defense, 2022. https://paperswithcode.com/task/adversarial-defense/latest.

[7] A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, 2018.

[8] Author(s) omitted. PriMask: Cascadable and collusion-resilient data masking for mobile cloud inference. Unpublished.

[9] T. B. Brown, D. Mane, A. Roy, M. Abadi, and J. Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[10] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *S&P (Oakland)*, 2017.

[11] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, 2018.

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[13] W. He, J. Wei, X. Chen, N. Carlini, and D. Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *WOOT*, 2017.

[14] D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*, 2017.

[15] D. Hendrycks, M. Mazeika, and T. Dietterich. Deep anomaly detection with outlier exposure. In *ICLR*, 2018.

[16] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *AISec*, 2011.

[17] S. Jain. Watch: Tesla autopilot feature mistakes moon for yellow traffic light, 2021. https://www.ndtv.com/offbeat/watch-tesla-autopilot-feature-mistakes-moon-for-yellow-traffic-light-2495804.

[18] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang. *Moving target defense: creating asymmetric uncertainty for cyber threats*, volume 54. Springer Science and Business Media, 2011.

[19] A. Kurakin, I. Goodfellow, S. Bengio, et al. Adversarial examples in the physical world. In *ICLR Workshops*, 2017.

[20] S. Lee, W. Lee, J. Park, and J. Lee. Towards better understanding of training certifiably robust models against adversarial examples. *NeurIPS*, 2021.

[21] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. In *ICLR*, 2017.

[22] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[23] M. Milinski and R. Heller. Influence of a predator on the optimal foraging behaviour of sticklebacks (gasterosteus aculeatus l.). *Nature*, 275(5681):642–644, 1978.

[24] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *CVPR*, 2017.

[25] S. Motiian, Q. Jones, S. M. Iranmanesh, and G. Doretto. Few-shot adversarial domain adaptation. In *NeurIPS*, 2017.

[26] D. Narayanan, K. Santhanam, A. Phanishayee, and M. Zaharia. Accelerating deep learning workloads through efficient multi-model execution. In *NeurIPS Workshops*, 2018.

[27] G. Rapier. Tesla's autopilot confused a burger king sign for a stop sign. the fast-food chain turned it into an ad., 2020. https://www.businessinsider.com/tesla-autopilot-mistakes-burger-king-stop-sign-new-ad-2020-6.

[28] N. Ratzlaff and L. Fuxin. Hypergan: A generative model for diverse, performant neural networks. In *ICML*, 2019.

[29] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. Depristo, et al. Likelihood ratios for out-of-distribution detection. In *NeurIPS*, 2019.

[30] K. Ren, T. Zheng, Z. Qin, and X. Liu. Adversarial attacks and defenses in deep learning. *Engineering*, 6(3):346–360, 2020.

[31] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, et al. Deep one-class classification. In *ICML*, 2018.

[32] Q. Song, Z. Yan, and R. Tan. Moving target defense for embedded deep visual sensing against adversarial examples. In *SenSys*, 2019.

[33] G. Sriramanan, S. Addepalli, A. Baburaj, et al. Towards efficient and effective adversarial training. *NeurIPS*, 2021.

[34] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The german traffic sign recognition benchmark. In *IJCNN*, 2011.

[35] R. Timofte, K. Zimmermann, and L. Van Gool. Multi-view traffic sign detection, recognition, and 3d localisation. *Machine Vision and Applications*, 25(3):633–647, 2014.

[36] E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018.

[37] K. Zhang, B. He, J. Hu, Z. Wang, B. Hua, J. Meng, and L. Yang. G-net effective gpu sharing in nfv systems. In *NSDI*, 2018.

[38] H. Zhou, S. Bateni, and C. Liu. S3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads. In *RTAS*, 2018.

[39] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu. Traffic-sign detection and classification in the wild. In *CVPR*, 2016.