Dynamic Layer Routing Defense for Real-Time Embedded Vision

ZIMO MA, Nanyang Technological University, Singapore
XIANGZHONG LUO[*], Nanyang Technological University, Singapore
QUN SONG, City University of Hong Kong, China
RUI TAN, Nanyang Technological University, Singapore

Deep neural networks have advanced the perception and decision-making functions of smart embedded systems, such as car-borne driver assistance. Deploying these embedded neural networks often faces two challenges: (i) *security vulnerabilities* to adversarial examples that can be deployed in the perceived physical environment; (ii) *limited computational resources* coupled with *dynamic conditions* that necessitate real-time adaptation of model execution. However, these two challenges are often addressed separately in existing research. This paper presents LeapNet, which aims to address both challenges simultaneously. It comprises two versions: LeapNet-1 and LeapNet-2. LeapNet-1 employs dynamic layer routing to counteract adaptive adversarial-example attacks and reduce computational redundancy. Building upon LeapNet-1, LeapNet-2 further adapts its layer routing configurations in real time to meet the frame processing rate requirements under dynamic conditions while maintaining defense performance. Extensive experiments on various representative datasets, neural network models, and adaptive attacks demonstrate the superiority of LeapNet over existing defense methods. On-road tests with a real-time car-borne traffic sign recognition system validate its effectiveness in maintaining frame processing rate under dynamic conditions.

CCS Concepts: • **Computer systems organization → Embedded software**; • **Security and privacy → Domain-specific security and privacy architectures**.

Additional Key Words and Phrases: Deep neural network, adversarial attack, dynamic inference routes, embedded computing

1 Introduction

Deep neural networks (DNNs) have shown good performance in various computer vision tasks, such as image classification, object detection, and tracking [1–3]. Their uses in embedded and mobile applications, such as unmanned terrestrial and aerial vehicles, have attracted increasing attention [4–6]. However, despite these interests, two challenges remain in deploying them on embedded systems. First, the adoption of DNNs raises security and reliability concerns due to their

[*]Currently with Southeast University, Nanjing, Jiangsu, China.

This research/project is supported by the National Research Foundation, Singapore, under its AI Singapore Programme (AISG Award No: AISG4-GC-2023-006-1B).
This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2025.
Authors' Contact Information: Zimo Ma, zimo001@e.ntu.edu.sg, College of Computing and Data Science, Nanyang Technological University, Singapore; Xiangzhong Luo, College of Computing and Data Science, Nanyang Technological University, Singapore, xiangzho001@e.ntu.edu.sg; Qun Song, qunsong@cityu.edu.hk, Department of Electrical Engineering, City University of Hong Kong, Hong Kong, China; Rui Tan, tanrui@ntu.edu.sg, College of Computing and Data Science, Nanyang Technological University, Singapore.

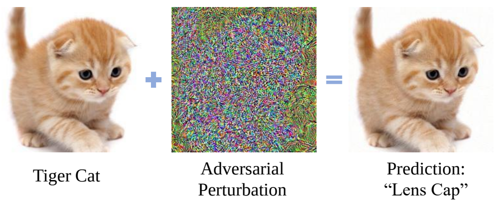
ACM 1558-3465/2025/7-ARTXXX
https://doi.org/10.1145/xxxxxxxxx

known vulnerability to adversarial-example attacks [7–9]. Second, the deployment of the DNNs on embedded systems is challenged by both the constrained computational resources and the complex dynamic conditions that require real-time adaptation of model execution. These two challenges are explained in detail as follows.

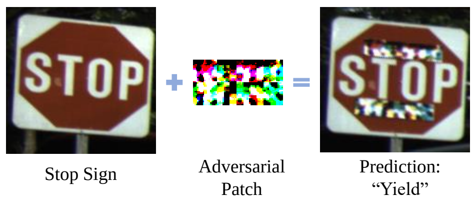
Security challenge: DNN-based embedded perception systems are vulnerable to adaptive adversarial attacks. Embedded perception is a key component of various systems such as driving assistance agents [10, 11]. However, recent accidents related to these agents point to safety concerns stemming from perception errors, including misses in detecting articulated buses or pedestrians [12]. According to the 2023 annual disengagement reports from the California Department of Motor Vehicles, 40% of disengagements were attributed to errors in the perception module, the highest proportion among all contributing factors [13]. Such potential unsafety undermines confidence in the core machine learning technologies that underpin modern embedded perception. Furthermore, recent studies have demonstrated the fragility of DNN-based computer vision systems under hostile settings. For instance, with adversarial perturbations resembling natural noises, computer vision models misclassify stop signs as speed limit signs [14]. The robustness of the safety-critical embedded perception against the crafted adversarial perturbations needs to be enhanced.

A plethora of defense mechanisms have been developed to harden DNNs against threats, such as adversarial training, input transformations, and gradient obfuscation [15–17]. However, these defenses are *deterministic*, in that the same input leads to the same processing path with no variability or stochasticity. Such determinism makes them vulnerable to the more advanced *adaptive attacks*, where the attackers iteratively probe the system, analyze its responses, and refine attack strategies to bypass defenses and maximize attack impacts [2]. To address adaptive attacks, *dynamic defense* approaches have recently gained research interest [18, 19]. A dynamic defense changes its mechanism for each inference so as to challenge the adversaries to craft effective attacks. Following this approach, existing studies apply randomized transformations to the inputs [18] and generate time-varying model ensembles [2] to enhance robustness. However, these studies do not consider embedded vision systems' main constraint of limited computation resources and the dynamic environments that these systems often operate in. For instance, the dynamic ensemble approach in [2] leads to multiplied computational overhead due to the simultaneous inferences with multiple models, making it unsuitable for resource-constrained embedded systems.

Efficiency and adaptability challenge: Resource constraints and dynamic conditions impede the deployment of DNN-based functions in embedded systems. DNNs' complex architectures and massive parameters pose challenges to the deployment on embedded systems with limited computation and memory resources. As DNNs often have inherent redundancy, prior studies have proposed various model tailoring techniques to facilitate DNNs' efficient deployment. To decrease the model size for less computational and memory cost, the pruning technique [20] directly reduces parameters by sparsifying the network, while quantization [21] casts floating-point values into lower-precision representations. Other approaches, such as early exit [22], layer routing [23], and width scaling [24], focus on reducing model structures. However, these approaches fall short of addressing dynamic conditions, such as changeable frame processing rate requirements and varying environments (e.g., power modes, the number of objects) [25, 26]. For instance, the speed of a vehicle dictates the upper bound of the inference latency. At higher speeds, vehicles require significantly longer distances to stop safely. For instance, the typical braking distance is approximately 9 meters at 40 km/h, but increases to around 67 meters at 110 km/h [27]. Therefore, it is necessary for autonomous vehicles to adjust their perception's inference latency according to the speed to ensure sufficient safe stopping distances [25]. In such dynamic scenarios, one-time model tailoring approaches often sacrifice accuracy to address the worst case, such as maintaining safe braking distances corresponding to the maximum possible speed. This results in a fixed trade-off that restricts their effectiveness across



(a) PGD attack.



(b) Patch attack.

Fig. 1. Illustration of two representative adversarial attacks: PGD attack [15] and Patch attack [14].

lower speed scenarios. As a relevant solution, neural architecture search (NAS) can find the optimal architectures to meet specific requirements [26, 28]. However, recent NAS-based approaches such as AdaptiveNet [26] incur long update latencies of up to minutes, making them unsuitable for real-time adaptation.

In this work, we aim to jointly address the above two challenges. To this end, we follow the *progressive system development* methodology to design two versions of a system called *LeapNet*, for resource-constrained embedded vision. The first version, LeapNet-1, focuses on both the security challenge and computational inefficiency. The second version, LeapNet-2, further incorporates real-time adaptation to dynamic conditions.

LeapNet-1 is a dynamic defense design based on *layer routing*. First, it learns the optimal layer routing distribution. Then, at runtime, it dynamically samples a route from the input-specific distribution to drop certain layers. Specifically, we introduce a lightweight decision network for routing, which is trained with reinforcement learning. LeapNet-1 reduces the number of layers for less computational redundancy and introduces stochasticity to limit the attackers' ability in constructing effective perturbations because achieving so requires the inference route. Moreover, LeapNet-1 promotes efficiency in terms of both inference latency and memory usage.

LeapNet-2 further integrates a latency-aware component to adapt the layer routing for maintaining the required frame processing rate under dynamic conditions. Specifically, we build a latency predictor for the target embedded vision system, which can predict the inference latency of different layer routing configurations. This facilitates the training of the decision network to develop latency awareness when learning the layer routing distributions. With LeapNet-2, the average latency of all sample routes from these routing distributions can adhere to the required frame processing rate in the continual processing of frames, and still maintain stochasticity to counteract adaptive attacks.

To summarize, this paper makes the following technical contributions:

- We design LeapNet to simultaneously address the two challenges faced by embedded vision: ensuring security against adaptive adversarial attacks while managing the embedded deployments given limited computational resources and achieving real-time adaptability under dynamic conditions.
- We design two versions of LeapNet. LeapNet-1 leverages a decision network to generate dynamic layer routes for counteracting adaptive attacks. It improves computational efficiency, and maintains an attainable accuracy simultaneously. LeapNet-2 can further maintain the required frame processing rate under dynamic conditions at the expense of some of the randomness used to improve adversarial robustness.
- We evaluate LeapNet on collected datasets and in real-world outdoor experiments. The experimental results demonstrate its superior defense performance against various attacks

Table 1. Adversarial accuracy comparisons under diverse attacks and defense settings, in which the deterministic defense corresponds to BlockDrop [23] and the dynamic defense corresponds to the proposed LeapNet-1.

Dataset	Defense Method	Static Attack	Adaptive Attack
CIFAR-10	w/o Defense (Baseline)	29.1%	29.1%
	w/ Deterministic Defense	61.7%	35.8%
	w/ Dynamic Defense	61.0%	**57.7%**
GTSRB	w/o Defense (Baseline)	37.2%	37.2%
	w/ Deterministic Defense	85.6%	22.3%
	w/ Dynamic Defense	79.2%	**71.6%**

compared with recent defense methods. Additionally, LeapNet-2 achieves real-time adaptation to varying conditions across different devices and target DNN models.

The remainder of the paper is organized as follows. Section 2 introduces the preliminaries and discusses the key observations and motivations. Section 3 and Section 4 describe LeapNet-1 and LeapNet-2, respectively. Section 5 presents the experimental settings and results. Section 6 presents our system implementation. Section 7 reviews related work. Section 8 discusses several related issues. Finally, Section 9 concludes the paper.

2 Background and Motivation

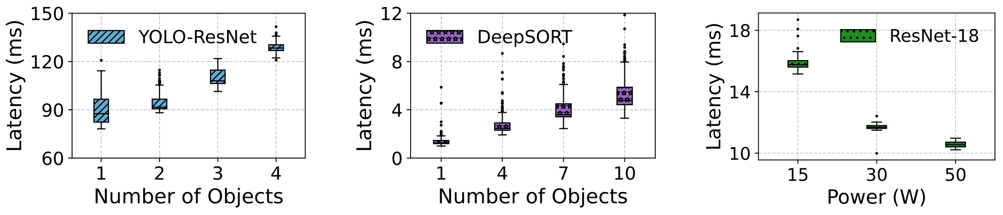
This section introduces the preliminaries on attacks and defenses, followed by our key observations and motivations behind the proposed LeapNet.

2.1 Preliminaries

■ **Adversarial Examples.** Adversarial examples refer to crafted inputs modified with small perturbations to mislead DNNs with incorrect predictions [15]. Let $f(x;\theta)$ denote the classifier with weights θ and input x associated with label y. An adversarial example $x' = x + \delta$ is tailored to mislead the classifier, such that $f(x';\theta) \neq y$, where δ is the perturbation. Fig. 1 illustrates the impact of adversarial examples on pre-trained DNN models. In Fig. 1 (a), a digital perturbation generated using PGD algorithm [15] causes a "tiger cat" picture to be misclassified as a "lens cap". In the physical world, where pixel-level adjustments are impractical, adversaries can use patch attacks [14] to mislead classifiers into recognizing a "stop" sign as "yield", as shown in Fig. 1 (b).

■ **Static Attacks vs. Adaptive Attacks.** A static attack refers to an attack that is designed and executed once, remaining unchanged throughout its deployment. Specifically, once launched, a static attack does not leverage any further interaction or feedback from the target model or its defense mechanisms. In contrast, an adaptive attack represents a more advanced and iterative threat model. Adaptive attackers can repeatedly interact with the target model, observe the model's defense mechanism, and iteratively adjust their attack strategy. Such adaptability enables attackers to continuously refine their strategies so as to circumvent defenses by dynamically evolving attacks.

■ **Deterministic Defense vs. Dynamic Defense.** A deterministic defense against adversarial examples refers to employing a fixed strategy to mitigate adversarial attacks. Deterministic defense always responds in the same way given a particular input, which introduces no randomness or variability. Deterministic defenses can be vulnerable to adaptive attacks due to their predictable behavior. In contrast, since a dynamic defense introduces randomness or variability to each inference, it becomes difficult for attackers to anticipate or exploit the defense behavior for each inference process. Thus, dynamic defenses are more resilient to adaptive attacks, which rely on detailed knowledge of the defense strategy.



(a) Object detection. (b) Object tracking. (c) Classification on Jetson Orin.

Fig. 2. Impact of dynamic conditions for embedded vision scenarios on the inference latencies of three tasks.

2.2 Observations and Performance Preview of LeapNet

■ **Observation 1.** *Existing deterministic defense mechanisms exhibit inferior defense performance compared with their dynamic counterparts, especially when confronting adaptive attacks.* Beyond static attacks, adaptive attacks are increasingly adopted for evaluating defense performance [2]. These attacks degrade model accuracy by circumventing deterministic defenses. To demonstrate this, we conduct an experiment that uses ResNet-18 as the target model and employs FGSM attack algorithm [7] to generate static attack and adaptive attack, on the CIFAR-10 [29] and GTSRB [30] datasets. The two datasets cover 10-class general image classification and 43-class traffic sign classification, respectively. The difference in attack effectiveness arises from the attacker's knowledge of the target model, despite using the same attack algorithm. Table 1 presents the adversarial accuracy (i.e., the accuracy under attack), before security enhancement and with enhancement by deterministic or dynamic defense. Deterministic defense refers to BlockDrop [23], a representative layer routing network that can generate a fixed inference route per input across different inputs. Although it performs reasonably under static attacks, its effectiveness drops remarkably under adaptive attacks. The dynamic defense in Table 1 refers to LeapNet-1 proposed in this paper, which introduces stochasticity to the layer routing network. It achieves better defense performance against both static and adaptive attacks, compared with the deterministic defense.

■ **Observation 2.** *Embedded devices often operate under various dynamic conditions, where the available resources and requirements may change over time.* There are existing research efforts toward efficient DNN deployment on embedded systems. However, these studies may not be sufficient to address the dynamics from the embedded environments [26]. We identify two types of dynamics:

(1) **Varying Embedded Environments.** Embedded systems may operate under changing environments, such as fluctuating scene complexity and computational resources. These dynamic environmental conditions can affect task processing speed, which challenges the performance of DNN models, especially for time-series tasks.

Scene complexity refers to variations in the number and complexity of objects in input images or videos, which directly influence the inference speed of DNN models. Fig. 2 (a) and Fig. 2 (b) illustrate how an increase in the number of objects changes the runtimes of the object detector and tracker on a server with an RTX 2080Ti GPU. DeepSORT [31] is used as the tracker, while YOLO-ResNet represents a system combining YOLO for detection with ResNet-18 for further classification [2]. These results demonstrate that runtime latency for each frame processing grows with the number of detected objects. This poses a challenge to the real-time performance of time-series processing.

Fluctuations in available computing resources, such as power supply mode, also affect the operation of embedded tasks. For example, an insufficient power supply may force an embedded device to switch to a low-power mode, such as the automatic low-power battery

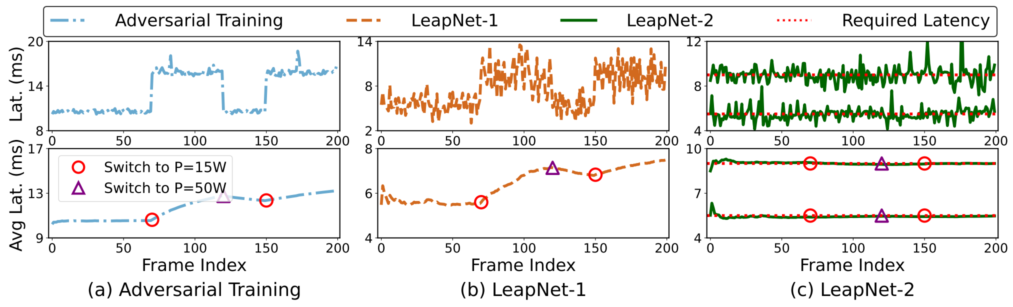


Fig. 3. Visualization of the runtime latency measured on NVIDIA Jetson Orin under different power modes.

switch of a drone for returning home safely [32]. When the system switches to a low-power mode, inference latency can increase. As a result, the requirements of embedded vision tasks may be potentially violated. Fig. 2 (c) shows the inference latency distributions of ResNet-18 [33] under different power modes. We can see an approximately 50% inference latency increase when the power mode is switched from 50 W (watt) to 15 W. This highlights that limited power supply in embedded systems can impair the timely execution of vision tasks, potentially leading to performance violations.

(2) **Varying Performance Requirements.** Embedded systems often have diverse and evolving performance requirements, such as varying frame processing rates [26]. These systems need to dynamically adjust computational resources or models to maintain real-time responsiveness under different requirements. For instance, in autonomous vehicles (AVs), inference latency largely decides certain safety measures, such as safe stopping distance. Considering an AV traveling at 7 m/s, if the object detector EDet2 with a relatively shorter frame processing latency is used, a safe stopping distance of 7.66 meters is required [25]. Differently, if EDet6 with a relatively longer frame processing latency is used, a safe stopping distance of 11.14 meters is required [25]. This variation underscores the necessity for AVs to scale the complexity of vision models in response to real-time speed and braking constraints. Achieving such adaptive control is essential for balancing accuracy and latency requirements in embedded applications.

■ **Observation 3.** *Existing deterministic and dynamic defense mechanisms cannot adapt themselves to dynamic conditions.* Effective deployment on embedded systems needs to consider limited computational resources and complex dynamic conditions. However, existing defense approaches, including both deterministic and dynamic defenses, often overlook these specific constraints of the practical deployment on embedded systems. Many defense methods, such as ensemble learning, enhance the security of DNNs at the expense of increased computational load. The adversarial training defense is not subjected to additional inference-time costs. However, it falls short of accommodating the dynamic operational conditions encountered during embedded deployment. One direct strategy for adapting to dynamic conditions is deploying multiple security-enhanced model variants for coarse adjustment. However, this results in frequent model switching and causes overhead due to the paging-in/paging-out of entire models [26]. Thus, there is a need for a unified DNN-based framework capable of simultaneously addressing security concerns and efficiently managing computational limitations and dynamic conditions in embedded environments.

To demonstrate the above, we conduct a preliminary latency measurement experiment that evaluates the performance of various defense methods on an embedded device, Jetson Orin. Fig. 3 shows the runtime latency traces of these defense methods applied to ResNet-18 under dynamic power conditions. Compared with adversarial training, our proposed dynamic layer routing approach, LeapNet-1, shows reduced inference latency. However, both adversarial training and LeapNet-1 exhibit clear latency increases when the system switches to a lower power mode, as shown by the blue and orange lines. In contrast, LeapNet-2 (depicted by the green line in Fig. 3) maintains latency closely aligned with the required value (depicted by the red line in Fig. 3) under different settings, which demonstrates LeapNet-2's capability to stabilize latency performance.

2.3 Motivations

On the one hand, as discussed in Observation 1, the weakness of deterministic defenses against adaptive attacks motivates the design of effective dynamic defense methods. To this end, we propose LeapNet-1 that features dynamic layer routing. It leverages a decision network to learn the optimal layer routing distribution for the given input (see Fig. 4 (*top*)). At runtime, LeapNet-1 dynamically samples layer routes from the layer routing distribution produced by the decision network, based on the inference input. LeapNet-1 maintains superior accuracy and exhibits stochasticity desired for counteracting adaptive attacks. To the best of our knowledge, LeapNet-1 is the first defense approach to enable dynamic layer routes.

On the other hand, as discussed in Observation 2, beyond resource constraints, dynamic conditions such as varying runtime environments and changing performance requirements challenge the embedded deployment of security-critical scenarios. Existing security-enhanced DNNs cannot adapt to real-world dynamic embedded conditions, especially those time-series embedded vision tasks (e.g., video processing). To this end, we draw insights from LeapNet-1 and further introduce LeapNet-2 (see Fig. 4 (*bottom*)). Specifically, LeapNet-2 features a latency predictor that enables the decision network to further learn the latency of different layer routes. Then, this latency-aware decision network decides the optimal layer routing distribution that satisfies the specified, adjustable latency constraint. Furthermore, dynamically sampled layer routes operate near the latency constraint. It allows the average latency to adapt to the latency constraint in real time.

Taking the above together, the proposed LeapNet can effectively counteract adaptive attacks while reducing computation redundancy. Furthermore, LeapNet can adapt to dynamic conditions in real time for embedded vision systems with a slight sacrifice of defense capability.

3 LeapNet-1

In this section, we present LeapNet-1, which employs a decision network to learn the optimal layer routing distribution for the given input. Furthermore, LeapNet-1 samples the layer routes from the resulting distribution for the processing of that input. This process introduces randomness that obscures the exact execution path and thereby impedes adaptive attacks that require precise structural knowledge of the network.

3.1 Preliminaries on Target Model

As shown in Fig. 4, LeapNet leverages a decision network to generate dynamic layer routes for the target model. Following recent conventions [23, 26], we take ResNet [33] as the default target model. It is a widely adopted basis model for vision tasks due to its effectiveness and simplicity. ResNet utilizes shortcut connections to add the input to the output of the residual block. Although each residual block is composed of multiple layers, for simplicity, we treat each residual block as a single layer and thus use the term *layer* instead of *block* when describing the routing process. Similar layer routing mechanisms can be extended to other DNN architectures through gating mechanisms with minor revisions [34].

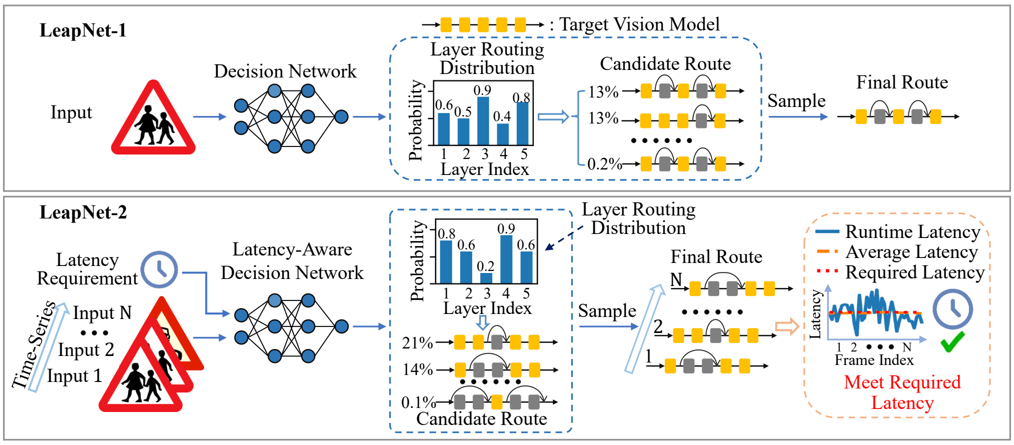


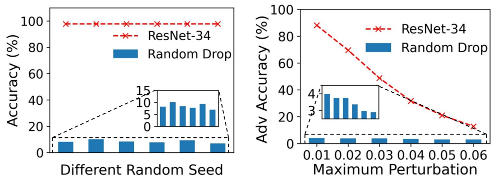
Fig. 4. Overview of LeapNet, where (i) LeapNet-1 focuses on learning the optimal layer routing distribution toward robust defense performance against adaptive attacks and computational efficiency, and (ii) LeapNet-2 focuses on learning the optimal layer routing distribution that satisfies the specified latency constraint under dynamic conditions while maintaining defense performance.

We use x and y to denote the input image and its ground-truth label, and consider a target DNN model $f_C(\cdot)$ with N layers. Finally, we can formulate the forward propagation of $f_C(\cdot)$ as $y' = f_C(x;\phi_C)$, where ϕ_C denotes the weights of $f_C(\cdot)$ and y' denotes the predicted result. Furthermore, the training process optimizes ϕ_C to minimize the cross-entropy loss on the training dataset: $\text{minimize}_{\phi_C}\, J_C = L_C(f_C(x;\phi_C), y)$, where $L(\cdot,\cdot)$ denotes the cross-entropy loss.

3.2 Decision Network

As discussed in [23], DNNs allow some layers to be dropped with minimal accuracy loss due to their redundancy. However, randomly dropping layers may degrade accuracy. To demonstrate this, we conduct a preliminary experiment using ResNet-34 on the KUL dataset [35]. As shown in Fig. 5, applying a random layer dropping strategy yields only 7.0%∼10.1% accuracy on clean inputs and 3.0%∼3.9% accuracy on PGD-attacked inputs. In contrast, ResNet-34 without layer dropping can have a 97.7% clean accuracy and 88.1% to 12.7% accuracy under PGD attacks as the maximum perturbation bound increases from 0.01 to 0.06 (relative to the maximum input pixel value).

Despite the acknowledged redundancy, determining which layers are redundant remains challenging. Given a target DNN with N layers, the number of possible layer routes grows exponentially (i.e., 2^N), making exhaustive exploration impractical. To address this issue, we propose a reinforcement learning-based decision network capable of simultaneously deriving layer routings across all layers. Instead of deterministically deciding which layers to drop, the decision network can learn the optimal layer routing distribution for the specific input, and then randomly sample the actual dropping decisions from this distribution. Since adaptive attacks typically rely on precise knowledge of the model's structure and deterministic behavior to craft effective perturbations, this stochastic design obscures the exact execution path. This makes it difficult for adversaries to anticipate or exploit specific routes. As a result, adversaries are unable to reliably compute accurate gradients for attack generation. Consequently, crafting effective adversarial examples becomes more difficult. Even if an attacker manages to breach the system and observe the real-time layer routes, the immediate execution of inference upon route sampling leaves insufficient time to construct and deploy an effective attack.



(a) Clean accuracy. (b) Adversarial accuracy.

Fig. 5. Accuracy of ResNet-34 with and without random layer dropping on KUL dataset. Maximum Perturbation indicates the perturbation upper bound.

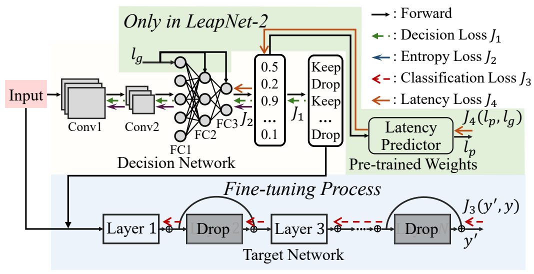


Fig. 6. The proposed training algorithms of LeapNet.

The decision network features a lightweight DNN model that comprises two convolutional and three fully-connected layers. Specifically, the decision network outputs a layer routing distribution, $p = [p_1, ..., p_n, ..., p_N]$, where $p_n \in [0, 1]$ is the probability of executing the n-th layer. Then, the inference routes are sampled based on this vector. In this case, the decision for the target network with N layers that is dropped or executed would follow the Bernoulli distribution:

$$\pi_{\phi_D}(a|x) = \prod_{n=1}^{N} p_n^{a_n}(1 - p_n)^{1-a_n}, \text{ s.t., } p = f_D(x; \phi_D), \tag{1}$$

where $f_D(\cdot)$ denotes the decision network and ϕ_D denotes its weights. The $a = [a_1, ..., a_n, ..., a_N]$ represents the action obtained through Bernoulli sampling according to probability p. The $a_n \in \{0, 1\}$ represents the action of either dropping ($a_n = 0$) or executing ($a_n = 1$) the n-th layer.

The optimization objectives of LeapNet-1 are three-fold, including (i) maximizing the accuracy on clean inputs, (ii) maximizing the number of dropped layers for less computation, and (iii) maximizing the stochasticity against adaptive attacks. To this end, we design the reward function $R(\cdot)$ and introduce a decision loss to optimize the objectives (i) and (ii). Note that an entropy loss is further added to optimize the objective (iii) as shown in Eq. (4). Specifically, the reward function $R(\cdot)$ is formulated as follows:

$$R(a, \phi_C, x, y) = \begin{cases} 1 - \left(\frac{\|a\|_1}{N}\right)^2 = 1 - \left(\frac{\sum_{n=1}^{N}|a_n|}{N}\right)^2, & \text{if } f_C(x; \phi_C(a)) = y, \\ -\gamma, & \text{otherwise,} \end{cases} \tag{2}$$

where $\frac{\|a\|_1}{N}$ quantifies the number of activated layers. $\|\cdot\|_1$ is the Manhattan l_1 norm. The $f_C(x; \phi_C(a))$ is the classification model with layer route action a. The γ is used to penalize the decision that produces incorrect predictions, which can trade off between accuracy and the number of activated layers. Finally, we formulate the decision loss as the negative objective of the reward function $R(\cdot)$ and minimize the decision loss to maximize the reward as follows:

$$\begin{aligned} J_1 &= -R(a, \phi_C, x, y) \log \pi_{\phi_D}(a|x) \\ &= -R(a, \phi_C, x, y) \sum_{n=1}^{N} (a_n \log p_n + (1 - a_n) \log(1 - p_n)) . \end{aligned} \tag{3}$$

Moreover, the entropy loss to maximize the decision stochasticity to counteract adaptive attacks is:

$$J_2 = -\sum_{n=1}^{N}(p_n \log p_n + (1 - p_n) \log(1 - p_n)). \tag{4}$$

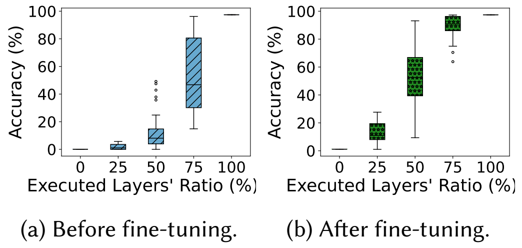


Fig. 7. Accuracy vs. executed layers' ratio of ResNet-18 on KUL dataset before and after fine-tuning.

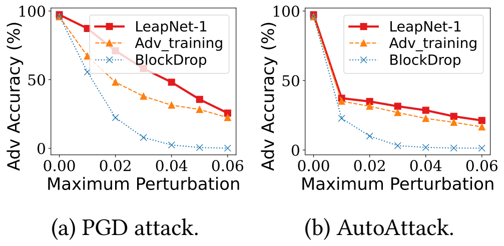


Fig. 8. Adversarial accuracy comparisons under different attacks on GTSRB dataset.

The above two losses in Eq. (3) and Eq. (4) will be used to steer the training and fine-tuning processes, as presented shortly.

3.3 Training & Fine-tuning

LeapNet-1 involves two separate stages, i.e., training and fine-tuning. Specifically, the training stage only optimizes the decision network, while the fine-tuning stage optimizes both the decision network and the target model.

Stage 1: Training. During training, we aim to increase the decision stochasticity, the number of dropped layers, and the classification accuracy simultaneously. Specifically, the decision network is trained with the following loss function:

$$J_{D_1} = \mathbb{E}_{\phi_D,x,y}[J_1(\phi_C, \phi_D, x, y)] + \alpha \cdot \mathbb{E}_{\phi_D,x}[J_2(\phi_D, x)], \tag{5}$$

where α is a constant to control the level of stochasticity. Note that this stage only optimizes the decision network's weights ϕ_D, while the pre-trained target model's weights ϕ_C are frozen.

Stage 2: Fine-tuning. The above layer-routing scheme inevitably suffers from inferior accuracy compared with the default target model [23], primarily due to the unavoidable omission of certain contributing layers. To mitigate this performance degradation, we further adapt the target model to the layer routing behaviors learned from the decision network. Specifically, as shown in Fig. 6, we fine-tune both the target model and the decision network to optimize the following loss function:

$$J_{D_2} = \mathbb{E}_{\phi_D,x,y}[J_1(\phi_C, \phi_D, x, y)] + \alpha \cdot \mathbb{E}_{\phi_D,x}[J_2(\phi_D, x)] + \beta \cdot \mathbb{E}_{\phi_C,\phi_D,x,y}[J_3(\phi_C, \phi_D, x, y)], \tag{6}$$

where α and β are the coefficients to control the trade-off magnitude between J_1, J_2, and J_3. Here, J_3 corresponds to the classification loss for the target model based on the routes generated from the decision network, which can be described as follows:

$$J_3 = L_C(f_C(x; \phi_C(a)), y). \tag{7}$$

Results. *(i) Accuracy:* We conduct an experiment on the KUL dataset [35] to demonstrate the effectiveness of fine-tuning. The results in Fig. 7 depict the relationship between accuracy and ratio of target model ResNet-18's executed layers before fine-tuning to that after fine-tuning. The results show that the joint fine-tuning stage can improve the accuracy across all executed routes of the target model. Besides, during fine-tuning, the decision network is also optimized to learn the layer routes with better accuracy. This joint optimization of both the target model and decision model enables LeapNet-1 to achieve good accuracy. *(ii) Adversarial Performance:* We compare LeapNet-1, BlockDrop [23], and adversarial training with FGSM [7] (denoted as Adv_training) in terms of adversarial robustness against adaptive attacks, based on PGD [15] and AutoAttack [36]. The results on the GTSRB dataset are shown in Fig. 8. LeapNet-1 consistently outperforms both Adv_training

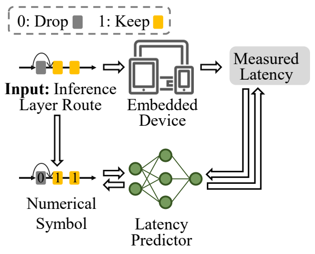


Fig. 9. The training process of the latency predictor with layer routes as input.

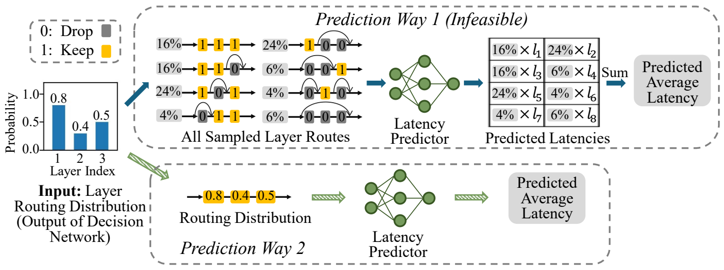


Fig. 10. Two different prediction ways during LeapNet-2's training: (i) Prediction by inputting all sampled routes from layer routing distributions. (ii) Prediction by inputting layer routing distributions.

and BlockDrop across varying levels of adversarial perturbation, which demonstrates superior robustness under adaptive attack settings.

4 LeapNet-2

In this section, we present LeapNet-2, an enhanced version that incorporates latency awareness into the decision-making process. Specifically, LeapNet-2 first builds a latency predictor to estimate the inference latency of the target model across different layer routes. With this latency predictor, LeapNet-2 trains a latency-aware decision network to produce the optimal layer routing distribution that satisfies the specified latency constraint. Consequently, the layer routes sampled from the resulting distribution consistently operate around the given latency setpoint. As a result, the average latency of sampled routes rapidly converges toward the setpoint. Note that the latency setpoint can be adjusted at run time.

4.1 Latency Predictor

First, we construct a latency predictor tailored for input layer routes of the target model. We train the predictor with collected data by measuring the inference latency of these different layer routes on the target hardware, as explained by Fig. 9. For a small target model, we measure the latency for all possible layer routes multiple times to construct the dataset. However, for a large target model (such as ResNet-101 with 34 blocks), the routing space is large (e.g., 2^{34} decisions). To deal with this, we use stratified sampling and random sampling techniques to select representative routes from this large routing space. Then, we build a latency predictor to model the relationship between inference latency and layer routes, described as $l_p = f_P(a; \phi_L)$, where l_p denotes the predicted latency, $f_P(\cdot)$ denotes the latency predictor, and ϕ_L denotes its weights.

We consider five representative prediction methods from the *scikit-learn* library [37], including MLP regression, linear regression, SVM regression, SGD regression, and kernel regression. Taking RTX 2080Ti as an example target device, we visualize the latency prediction results of ResNet-18 in Fig. 11. Among the above five prediction methods, we observe that the MLP regression algorithm can achieve the lowest root mean squared error (RMSE). This demonstrates its superior latency prediction performance across various layer routes of ResNet-18.

To develop latency awareness, LeapNet-2's decision network needs to learn the latency information of the target model across different layer routes from the latency predictor. Since the decision network learns the layer routing distribution, a straightforward approach is to input the sampled routes from the learned layer routing distribution to the pre-trained predictor to estimate the latencies of these sampled routes and then compute the average latency during LeapNet-2's

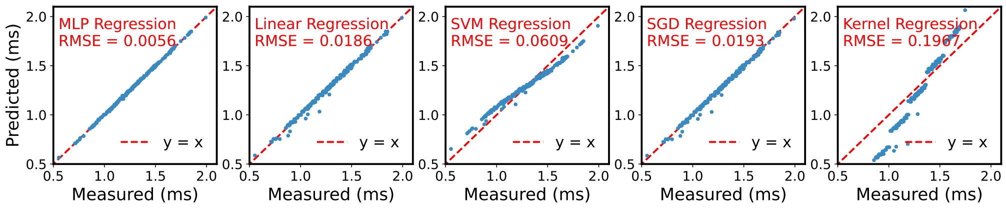


Fig. 11. Latency prediction performance of different regression algorithms under different layer routes.

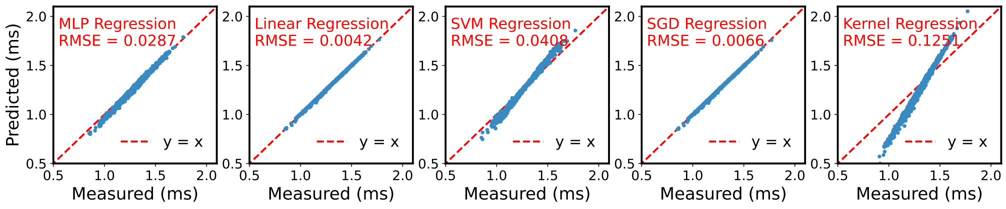


Fig. 12. Latency prediction performance of different regression algorithms (trained with input layer routes) under layer routing distributions. “Predicted” and “Measure” refer to the average latency of all sampled routes from the distribution.

training stage, as shown by *prediction way 1* in Fig. 10 (*top*). However, the sampling process is non-differentiable, which disrupts gradient flow and prevents the decision network from converging. As a result, *prediction way 1* (shown in Fig. 10 (*top*)) becomes infeasible.

Therefore, the only feasible approach for LeapNet-2's training is to use a latency predictor that directly estimates the average inference latency of all sampled routes by taking the layer routing distribution as input, as shown by *prediction way 2* in Fig. 10 (*bottom*). However, directly training such a latency predictor is infeasible due to the need to consider all possible sampled routes when computing the latency of this distribution during the training phase. Performing this computation for each layer routing distribution is impractical since it incurs substantial computational overhead.

We anticipate that the pre-trained latency predictor designed for input layer routes can be extended to handle layer routing distributions (*prediction way 2* in Fig. 10 (*bottom*)). It avoids the computationally intensive process of directly learning from these distributions. The observation that linear regression performs well in latency prediction proves this extension's feasibility, since there exists an inherent near-linear relationship between routing decisions and inference latencies, from the results in Fig. 11. According to Jensen's inequality, in a linear relationship, the predicted average inference latency $f_P(p; \phi_L)$ under a layer routing distribution p is equal to the weighted sum of the predicted latencies of all sampled routes $f_P(a; \phi_L)$ from this routing distribution.

We conducted an experiment to evaluate whether pre-trained latency predictors designed for input layer routes remain effective when applied to layer routing distributions. Specifically, we input the routing distributions p into the predictor to derive the predicted average inference latency $f_P(p; \phi_L)$. Next, we input all possible sampled routes a from the distribution into the predictor to derive the latency for each route $f_P(a; \phi_L)$. Then, we compare the predicted average inference latency with the weighted average of all possible routes. The detailed results are shown in Fig. 12, where the linear regression-based latency predictor achieves the best performance. When inputting layer routes into the pre-trained predictors, the MLP regression method achieves a smaller RMSE. However, when inputting layer routing distributions, the MLP regression introduces an additional error due to its non-linear characteristics, as suggested by Jensen's inequality. In contrast, the linear regression method avoids this additional error, as it lacks non-linear complexity. Thus, we choose the linear regression algorithm as the latency predictor under stochastic routing decisions.

Note that the predictor includes only 3000 parameters, which uses 0.01 MB memory. Thus, the predictor introduces low overhead, in comparison with the target models that have millions of parameters or even more. For instance, a small ResNet, ResNet-18, has 11.2 million parameters. The training cost and inference cost of the predictor are only 1.98 s and 0.15 ms on a server with an RTX 2080Ti GPU, respectively.

4.2 Latency-Aware Decision Network

To enable latency awareness in the decision network of LeapNet-2, latency information of layer-routing must be explicitly incorporated into the decision network. To this end, we first integrate a latency branch within the decision network to receive the latency requirements as input. Directly incorporating latency into the input challenges the decision network to learn latency requirements, as the input image is far more complex than a single latency value. To address this, we integrate latency information before multiple fully connected layers to enlarge its significance. Furthermore, we leverage a pre-trained latency predictor to facilitate the training of the decision network: the output layer routing distributions from the decision network are sent to the latency predictor to estimate the predicted average latency during the forward stage of training. This allows the decision network to adjust its parameters accordingly through backpropagation based on the required latency and estimated latency, as shown in Fig. 6. Note that the latency predictor is only required in the training process.

LeapNet-2 should be capable of generating layer-routing distributions conditioned on the latency requirements. In particular, it needs to ensure that the expected inference latency—computed as the weighted average of predicted latencies over sampled routing paths—matches the given constraint. To achieve this, compared with LeapNet-1, LeapNet-2 introduces an additional latency objective, i.e., to minimize the difference between the estimated average latency of the sampled routes from resulting layer routing distributions and the target latency requirements. Moreover, a latency loss function designed based on the mean squared error (MSE) is employed for capturing this objective:

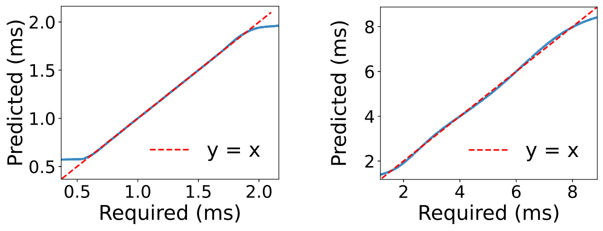
$$J_4 = L_{MSE}(f_P(p; \phi_L); l_g), \tag{8}$$

where l_g denotes the given latency requirements. Based on this loss function, the latency-aware decision network can learn the optimal layer routing distributions that closely align with the target latency requirements. Given the predictor's effectiveness in matching predicted latency with measured latency, this decision network can enable real-time adjustments of layer routes of the target model to meet the dynamic conditions of embedded deployment.

Note that multiple layer routing paths can satisfy the same latency constraints. The reinforcement learning-based decision network is trained to select routes that yield high accuracy. During inference, the routing path is stochastically sampled from the probability distribution generated by the decision network. This enables diverse execution paths even across the same inputs. As explained earlier, this stochasticity is a key factor for counteracting adaptive attacks.

4.3 Latency-Aware Training & Fine-tuning

As shown in Fig. 6, the training phase of LeapNet-2, similar to LeapNet-1, consists of two stages: first, training the decision network with the target model's weights fixed, and second, jointly fine-tuning both the decision network and the target model. Note that the latency predictor is pre-trained, and its weights remain fixed during both training and fine-tuning stages.



(a) ResNet-18. (b) ResNet-101.

Fig. 13. Illustration of the relationship between the required latency and the predicted inference latency for LeapNet-2 across different target models on RTX 2080Ti, including ResNet-18 and ResNet-101.

The first training stage consists of two update steps. First, similar to LeapNet-1, we train the main branch of the decision model to minimize the loss $\mathbb{E}_{\phi_D,x,y}[J_1(\phi_C, \phi_D, x, y)] + \alpha \cdot \mathbb{E}_{\phi_D,x}[J_2(\phi_D, x)]$, for defense objectives without compromising its accuracy. In the second step, we train only the latency branch to minimize the latency loss J_4. During this process, the weights before the latency input of the decision network are frozen. This second step can be repeated multiple times to balance performance between the latency objective and other objectives.

Finally, we jointly fine-tune the target model with the decision network to improve the accuracy performance while maintaining other objectives by adapting the target model to the learned latency-aware layer routing behavior. Note that the latency requirements are randomly generated based on the measured range of the target model with different possible routes so that LeapNet-2 can adapt to variable latency requirements without any manual adjustment.

In summary, building upon LeapNet-1, LeapNet-2 explicitly incorporates the dynamic conditions in practical embedded scenarios. It first employs a latency predictor to estimate the execution latency of dynamically sampled routing paths. Then, a latency-aware decision network adjusts the layer routing distributions flexibly to adapt to varying latency requirements. By sampling execution paths from this adaptive distribution, LeapNet-2 effectively meets the latency requirements of time-series processing under dynamic conditions, while simultaneously increasing the difficulty of constructing effective adversarial attacks.

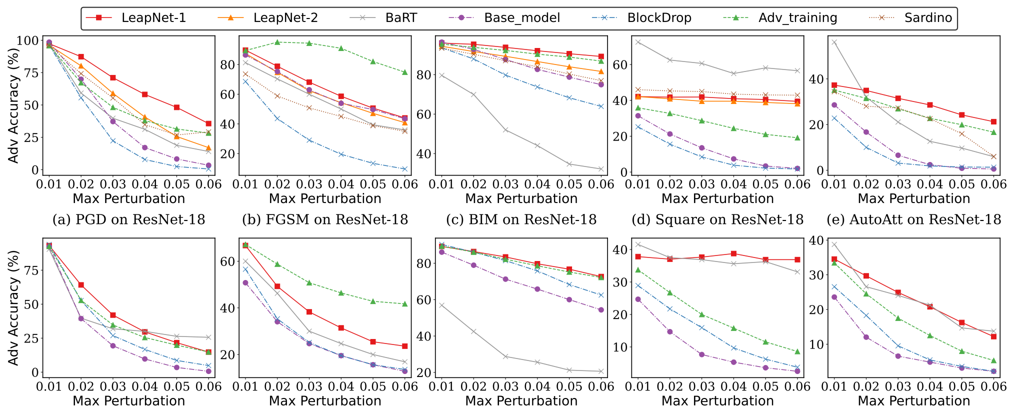
Results. Fig. 13 shows the predicted latency of the layer routes generated by the proposed latency-aware decision network, compared with given latency requirements for target models ResNet-18 and ResNet-101 on an edge device with RTX 2080Ti GPU. We can see that the predicted latency can match the given latency requirements closely for all target models. Furthermore, given the effectiveness of the latency predictor in aligning predicted latency with measured latency (shown in Fig. 11), the final average inference latency closely adheres to the latency requirements.

5 Experiments

In this section, we evaluate LeapNet to demonstrate its adversarial robustness and adaptability to dynamic conditions.

5.1 Experiment Setup

Environments. We implement our method using Python with PyTorch and conduct experiments on an edge server with NVIDIA RTX 2080Ti GPU with 11 GB memory, and two embedded devices, a Jetson Orin with 64 GB memory and a Jetson Xavier with 16 GB memory, which are commonly used in various embedded systems [3, 38] due to their balanced trade-off between computational capability and power efficiency.



(a) PGD on ResNet-18 (b) FGSM on ResNet-18 (c) BIM on ResNet-18 (d) Square on ResNet-18 (e) AutoAtt on ResNet-18

(f) PGD on MobileNetV1 (g) FGSM on MobileNetV1 (h) BIM on MobileNetV1 (i) Square on MobileNetV1 (j) AutoAtt on MobileNetV1

Fig. 14. Comparisons of adversarial accuracy under different adaptive attacks with different target models on GTSRB dataset.

Datasets and Models. We evaluate LeapNet on three datasets: German Traffic Sign Recognition Benchmark (GTSRB) dataset [30], KUL Belgium Traffic Signs (KUL) dataset [35], and Tiny-ImageNet dataset [39]. We experiment with four ResNets with different sizes as the target model: ResNet-18, ResNet-34, ResNet-101, and ResNet-152. To show the generalizability of LeapNet across different network architectures, we conduct experiments using MobileNetV1 [40] as the target model.

Attacks. We consider five typical attack algorithms to construct adaptive attacks in evaluating defense mechanisms: FGSM [7], PGD [15], BIM [41], Square attack [42], AutoAttack [36]. We use adversarial accuracy as the metric, defined as the classification accuracy on adversarial examples.

Baselines. We evaluate the performance of LeapNet with multiple baselines, including no defense, two deterministic defense methods, and two dynamic defense methods. `Base_model` is the target model of LeapNet with no defense by default. `BlockDrop` [23] is a representative layer routing mechanism that can switch different inference paths for different inputs, which can be seen as a deterministic defense. `Adv_training` [15], a common and effective static defense method, adopts both real data and data generated by a simulated adversary to improve its robustness. Specifically, it employs the FGSM-based adversarial examples in the training stage. `BaRT` [18] is a dynamic defense method that randomly leverages a combination of input transformation methods as a robust defense. Since many transformations are non-differentiable, we choose to directly tamper with the transformed images to form adversarial examples, rather than performing gradient-based attacks through these transformations. `Sardino` [2], a recent dynamic defense method, adopts a HyperNet [43] to generate an ensemble of classification models with high-rate ensemble renewal.

5.2 Performance on Adversarial Defense

On the GTSRB dataset, we evaluate the defense performance of LeapNet with multiple baselines under various adaptive attacks. We use ResNet-18 and MobileNetV1 as the target models. The results are shown in Fig. 14. LeapNet-1 and LeapNet-2 achieve higher accuracy than unhardened `Base_model` across different perturbation settings. This shows their enhanced defense ability across different attacks. Compared with `BlockDrop`, a representative layer routing method, LeapNet-1 and LeapNet-2 demonstrate superior adversarial accuracy under all attacks with a maximum 37.7% accuracy improvement. This demonstrates that the introduction of randomness in layer routing decisions can complicate attack construction. Although `Adv_training` surpasses LeapNet models

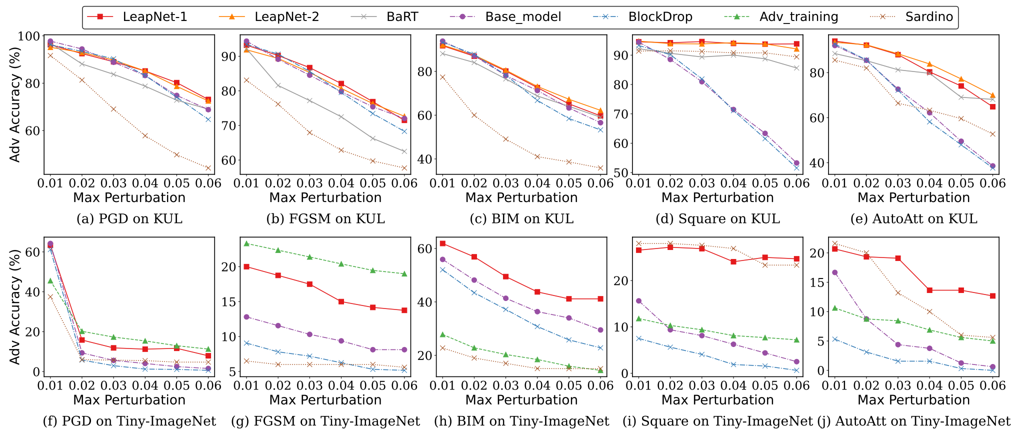
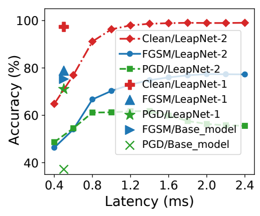


Fig. 15. Comparisons of adversarial accuracy under different adaptive attacks on KUL dataset using ResNet-34 as the target model and Tiny-ImageNet dataset using ResNet-152 as the target model.

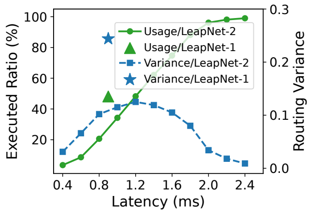
under the FGSM attack due to using the same attack during training, it lacks generalizability to unknown attack types common in real-world applications. In contrast, LeapNet offers general defense performance across different attacks, and thus outperforms `Adv_training` in counteracting the attacks that the Adv_training is not adversarially trained upon.

LeapNet-1 outperforms `Sardino` in counteracting most attacks. This advantage primarily stems from `Sardino`'s design limitations, which restrict its applicability to small DNNs. Specifically, `Sardino` dynamically regenerates network weights to enhance robustness. However, its effectiveness on deeper architectures like ResNet-18 is limited due to the exponentially increasing overhead with network complexity. For a fair evaluation, we adopt `Sardino`'s original setup using a smaller three-layer baseline network. However, as smaller networks inherently provide weaker adversarial defenses, it weakens `Sardino`'s effectiveness against strong adaptive attacks. In addition, LeapNet outperforms `BaRT` under most attack scenarios and configurations, except that they achieve similar accuracy under AutoAttack. This advantage arises because `BaRT`'s randomness is limited to a fixed number of input transformation methods, whereas LeapNet introduces greater diversity through its design. Moreover, `BaRT` and LeapNet optimize different and independent components of the DNN architecture, making them compatible and potentially complementary for enhancing robustness.

Besides, LeapNet-2 shows similar or lower accuracy compared with LeapNet-1 (0.4%∼10.9%), because of sacrificing some stochasticity in exchange for adaptability to dynamic conditions. We further investigate this performance degradation using ResNet-18 as the target model, as illustrated in Fig. 16. Specifically, Fig. 16 (a) shows the classification accuracy of LeapNet under clean conditions as well as under FGSM and PGD attacks, across varying latency constraints. Fig. 16 (b) shows the corresponding layer routing behaviors of LeapNet, including the average usage ratio of executed layers and the variance of layer dropping. As observed, a significant drop in accuracy occurs when the latency constraint is under 1.0 ms, for both clean and adversarial settings, which notably reduces the average robustness of LeapNet-2. This accuracy degradation can be attributed to aggressive layer dropping under tight latency budgets, as evidenced by the substantial decrease in the executed layer ratio shown in Fig.16 (b). Moreover, LeapNet-2 exhibits reduced defense effectiveness compared with LeapNet-1, partially due to its limited routing randomness. As shown in Fig.16 (b), the variance in layer selection is noticeably lower for LeapNet-2. This stems from the latency constraint imposed on LeapNet-2, which restricts feasible routing paths and thereby limits stochastic variability.



(a) Accuracy vs latency.



(b) Ratio/variance vs latency.

Fig. 16. Adversarial accuracy, executed layers' ratio and variance of LeapNet-2 under different latency settings, compared with LeapNet-1.

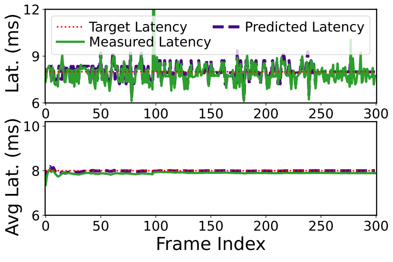


Fig. 17. Runtime and average latency of continuous frames for LeapNet-2 under different power settings on Jetson Orin.

We further analyze the performance differences of LeapNet under FGSM and PGD attacks. As shown in Fig.16 (a) and Fig.16 (b), the classification accuracy under PGD attack follows the trend of routing variance, highlighting the role of the stochasticity in enhancing robustness. The improved accuracy of LeapNet compared with the `Base_model` further supports the effectiveness of randomized routing against PGD attacks. In contrast, under the FGSM attack, the accuracy appears to correlate more with the number of executed layers. However, this does not imply that randomness plays no role in defending against FGSM attacks. For instance, when the latency constraint is set to 1.2 ms, LeapNet-1 achieves higher adversarial accuracy than LeapNet-2, despite having a similar layer execution ratio but higher routing variance. This observation confirms that stochastic routing contributes to robustness against the FGSM attack. Nevertheless, this improvement is less pronounced for FGSM rather than PGD. Specifically, LeapNet-1 only shows a 3.5% improvement over `Base_model` under FGSM attack, but achieves a 33.9% improvement under PGD attack. These findings suggest that different types of adversarial attacks exploit model vulnerabilities in distinct ways, and that the efficacy of randomness-based defense mechanisms varies depending on the attack strategy.

We extend the evaluation of LeapNet's defense performance across other datasets and use different target models, as shown in Fig. 15. In Fig. 14, `Adv_training` shows improved robustness against the FGSM attack, which matches the method used during training. To demonstrate the compatibility of LeapNet with current defense methods for better robustness, we evaluate LeapNet's performance when combined with adversarial training on the KUL dataset using ResNet-34 as the target model, as shown in Fig. 15 (a) to Fig. 15 (e). Specifically, we apply FGSM-based adversarial training to the `Base_model` and all baseline models to ensure a fair comparison. Compared with `Base_model`, LeapNet-1 achieves superior adversarial accuracy under various attacks. For example, LeapNet-1 can provide up to 40.6% accuracy improvement over `Base_model` under the Square attack. The superior performance of LeapNet-1 over other baselines shows its better compatibility with existing defense methods. To demonstrate the generalizability of LeapNet to larger models and more complex datasets, we evaluate the defense performance of LeapNet on the Tiny-ImageNet dataset using ResNet-152 as the target model, as shown in Fig. 15 (f) to Fig. 15 (j). The results show that LeapNet outperforms existing defense methods across various attacks under most settings.

5.3 Performance on Clean Accuracy

We test the clean accuracy delivered by our methods and the baselines, as shown in Fig. 14. When the maximum adversarial perturbation is set to 0, it indicates that no adversarial perturbation is applied. This allows us to assess the models' performance on clean data. The original ResNet-18

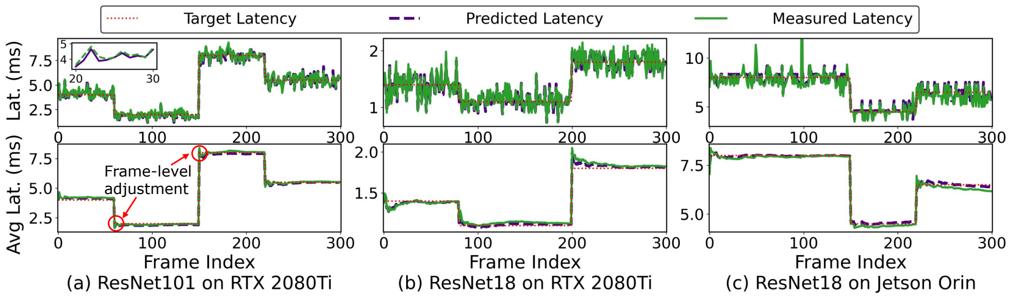


Fig. 18. Runtime and average latency of continuous frames for LeapNet-2 under different settings.

on the GTSRB dataset achieves a test accuracy of 98.5%. In comparison, LeapNet-1 and LeapNet-2 attain 97.3% and 96.2% accuracy, respectively, which reflects only a marginal decrease relative to the original ResNet-18. This slight drop can be attributed to the model's emphasis on learning more generalized features, rather than solely optimizing for performance on clean data. A similar trend is observed in `BlockDrop` and `Adv_training` with 96.4% and 95.8% accuracy. It also supports the notion of the minor trade-off in clean accuracy for robustness-enhancing techniques.

We also evaluate the accuracy of LeapNet-2 with ResNet-18 as the target model on the GTSRB dataset under various latency requirements, as shown in Fig. 16. When the latency constraint is stringent, such as 0.4 ms, almost all selectable layers are dropped. In this case, the accuracy drops to 60%, representing severe performance degradation. However, some relaxation of the latency constraint results in accuracy improvements. Specifically, executing only 20% of the layers boosts the accuracy to nearly 90%. To maintain an acceptable level of accuracy in embedded systems, a minimum latency requirement can be enforced. This requirement can be adjusted based on the system's needs to ensure that a sufficient number of layers remain active for reliable performance.

5.4 Performance on Adaptability

Varying Embedded Environments. We evaluate LeapNet-2's adaptability to dynamic embedded environments with varying computing resources by altering the power mode on Jetson Orin. The power supply transitions through three stages: (i) initially operating at 50 W, (ii) then reducing to 30 W after processing 100 frames, and (iii) further dropping to 15 W after processing 250 frames. Fig. 17 depicts the measured, predicted, and target latency per frame (shown as in Fig. 17 (*top*)) and their corresponding average latencies (shown as in Fig. 17 (*bottom*)), computed after each latency change. The alignment between measured latency with the predicted latency shows the effectiveness of the latency predictor. Although individual frame latencies fluctuate around the target, LeapNet-2 consistently maintains an average latency close to the target, even under significant power reduction. This shows LeapNet-2's stable performance in the presence of resource variations. The frame-wise fluctuation is inevitable due to the inherent stochasticity in layer routing to deal with adversarial attacks.

Varying Performance Requirements. We also test the inference latency of LeapNet-2 when processing continuous frames under varying latency requirements, as shown in Fig. 18. To test the generalizability, we consider three different scenarios: ResNet-101 as the target model on RTX 2080Ti, ResNet-18 as the target model on RTX 2080Ti, and ResNet-18 as the target model on Jetson Orin. The measured latency closely matches the predicted latency across various model sizes and embedded devices under different latency constraints. This demonstrates the effectiveness of the

Table 2. Comparisons of inference cost on Jetson Orin.

Model	Runtime (ms)	Static Memory (MB)	Runtime Max Memory (MB)
ResNet-18	10.6	42.7	186.4
Decision Model	1.3	0.7	47.6
LeapNet	6.8	43.4	185.6

Table 3. Comparisons of training cost on one single RTX 2080Ti GPU.

Method	Phase	Runtime Max Memory (MB)	Runtime (s)/Epoch	Epoch Number	Total Runtime (s)
Adversarial Training		951	57.7	20	1154
LeapNet	ResNet-18 Training	677	22.9	16	817.9
	Decision Training	492	12.6	10	
	Fine-tuning	732	21.7	15	

latency predictor. When the given latency requirement changes, the average measured latency quickly adapts to the new target. This result shows the real-time adaptability of LeapNet-2.

We further evaluate the adjustable latency range of LeapNet-2 across different target models and embedded edge devices, as illustrated in Fig. 19. Specifically, we deploy LeapNet-2 with ResNet-18, ResNet-34, and ResNet-101 as target models on an edge server with an RTX 2080Ti, as well as on Jetson Orin and Jetson Xavier, which exhibit progressively lower computational capabilities. The results show that the inference latency spans from as low as 0.5 ms to nearly 80 ms. This highlights LeapNet-2's adaptability in dynamically adjusting computational complexity to accommodate diverse hardware constraints.

5.5 Inference Latency and Memory Usage

The cost of LeapNet comprises two components: inference cost and training cost.

Inference Cost. The proposed decision network contains 0.17 million parameters, much lower than the smallest target model, ResNet-18, which has 11.2 million parameters. This corresponds to a static memory footprint of merely 0.67 MB for the decision network, compared with 42.7 MB for the target model ResNet-18. LeapNet-2 further includes a latency predictor module to guide the routing decision during training. However, this module is used only during the training phase and is discarded at inference time. It comprises approximately 3,000 parameters, corresponding to just 0.01 MB memory, and introduces negligible computational overhead. We adopt the runtime cost of LeapNet-1 to represent both variants (named as LeapNet in Table 2) and Table 3 during inference-time evaluation.

We benchmark the inference latency and memory usage of ResNet-18 and LeapNet on a Jetson Orin device in processing 100 continuous image frames, as shown in Table 2. The decision model incurs an additional latency of approximately 12.3% relative to ResNet-18. This is attributed to the kernel launch overhead on Jetson Orin, which becomes more significant for small models with low GPU utilization. Despite this, LeapNet achieves a 35.8% reduction in average inference latency. Although the decision model consumes additional memory during inference, the overall runtime memory of LeapNet remains comparable to that of ResNet-18. This is primarily because the dominant contributor to inference-time memory usage in convolutional neural networks is the storage of intermediate feature maps. By selectively dropping certain layers, LeapNet effectively reduces both the computation and memory footprint associated with these feature maps. For reference, when all selectable layers in ResNet-18 are bypassed, the peak runtime memory usage can be reduced to 47.1 MB.

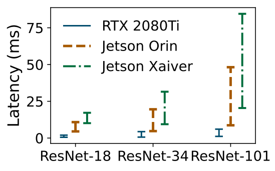


Fig. 19. Adjustable latency range for various models on different devices.

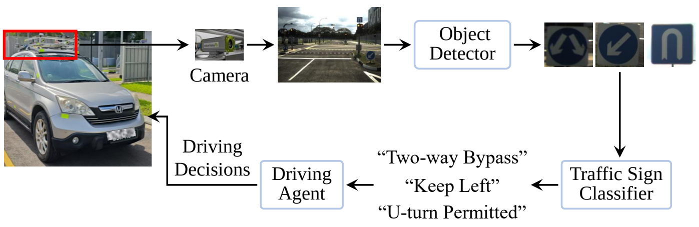


Fig. 20. Pipeline of traffic sign recognition-assisted driving systems.

Training Cost: We also compare the training time and peak memory usage of LeapNet and adversarial training (with FGSM attack) on a server equipped with an RTX 2080Ti GPU, as shown in Table 3. The training process of LeapNet consists of three phases: (i) training the target model ResNet-18, (ii) training the decision model, and (iii) joint fine-tuning. Despite the three-stage training process, LeapNet demonstrates lower training time and memory usage compared with the adversarial training baseline. This is primarily because adversarial training requires the generation of perturbed inputs based on model parameters in each iteration, which introduces additional computational overhead. These results highlight the training efficiency of LeapNet.

In summary, LeapNet-1 can provide robust and universal defense performance under various adaptive attacks while maintaining superior accuracy. LeapNet-2 further introduces the capability to adapt to varying embedded conditions across different edge devices and target DNN models. By effectively considering computational efficiency and model robustness, LeapNet-2 is suitable for security-critical embedded applications that operate under dynamic conditions.

6 Real-Time On-Vehicle Traffic Sign Recognition

In this section, we apply LeapNet to a traffic sign recognition system and evaluate its adaptability to dynamic embedded conditions.

6.1 System Implementation

We apply LeapNet to a traffic sign recognition (TSR) system [44], as illustrated in Fig. 20. While deep neural network (DNN)-based object detection models are capable of directly classifying traffic sign types, their scalability is often limited when dealing with a large number of sign categories. To address this challenge, a two-stage TSR approach—comprising a sign detection phase followed by a fine-grained classification stage—has been shown to support a broader range of traffic sign classes [2, 45]. In a two-stage TSR, images captured by the onboard camera are buffered and passed to the detector. Then, detected traffic signs are cropped, resized, and sent to a classifier to determine their type. The results are provided to the driving agent for real-time decision-making. Given a latency constraint t_c, if detection takes t_d seconds for a frame containing n signs, the soft deadline for classifying each sign is $\frac{t_c - t_d}{n}$ seconds. We implemented and tested this system on a vehicle equipped with an Alvium G1-510c camera and a Jetson Orin. The latter is commonly adopted in modern AI-enabled vehicles [38].

In this system, we utilize YOLO and YOLO-tiny [46] as traffic sign detectors and LeapNet-2 as the classifier. YOLO is a representative DNN-based object detection network that achieves good accuracy and latency performance on embedded systems. YOLO-tiny is a lightweight version of YOLO with less computational cost. The detector model is trained based on the publicly available COCO dataset [47] and KUL Belgium Traffic Signs Detection dataset [35], with the "stop sign" class

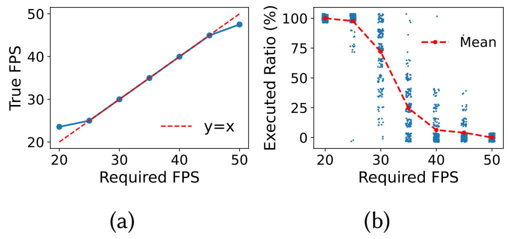


Fig. 21. True FPS and selected layers' ratio for LeapNet-2 under different FPS settings.

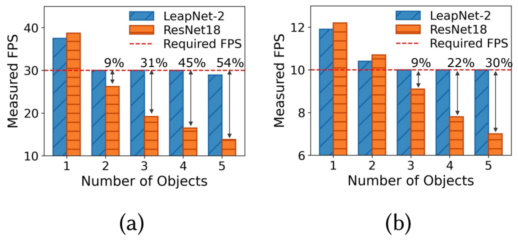


Fig. 22. FPS under different traffic signs' number settings. 30 FPS for YOLO-tiny (a) and 10 FPS for YOLO (b).

relabeled as "traffic sign" to enhance the ability to detect traffic signs. Besides, we train LeapNet as the classifier model with the KUL Belgium Traffic Signs Classification dataset.

Compared with the original YOLO and YOLO-tiny with mAP@0.5 scores of 57.9 and 33.1 [46], the retrained YOLO and YOLO-tiny achieve improved mAP@0.5 scores of 58.9 and 33.7. Note that the mAP@0.5 refers to the mean average precision computed using a fixed intersection over union (IoU) threshold of 0.5, where a predicted bounding box is considered correct if its IoU with a ground-truth box is at least 0.5. The LeapNet-based classifier achieves a clean accuracy of 96.3%, comparable to the accuracy of 97.0% reported in [35]. *Latency performance:* On Jetson Orin, YOLO achieves throughputs of 23 frames per second (FPS) and 15 FPS at input frame sizes of 544×608 and 704×832, respectively. LeapNet, using ResNet-18 as the target model, achieves inference latency ranging from 4.5 ms to 10.8 ms.

6.2 Performance Evaluation

In this pipeline system, the dynamics arise from two key factors: (i) varying FPS requirements and (ii) the fluctuating number of traffic signs in each frame. The first factor demands that the system adapts to flexible processing speed requirements, while the second necessitates maintaining a stable processing speed under changing conditions. Note that FPS is a variant of latency requirements. Therefore, we conduct outdoor experiments to evaluate LeapNet-2's ability to adapt to the dynamic conditions to satisfy the system requirements.

First, we test LeapNet-2 under different FPS settings with a real-world video consisting of 200 continuous frames. We utilize YOLO-tiny as the detector model and ResNet-18 as LeapNet-2's target model. The image frames are resized to 608×544 before feeding into the detector model. Fig. 21 illustrates the measured FPS and the number of selected layers for LeapNet-2 under different FPS settings. In Fig. 21 (a), the measured FPS closely aligns with the required FPS. Specifically, it reaches the system's minimum (i.e., 24 FPS) and maximum (i.e., 47 FPS) achievable performance when the required FPS is set to 20 and 50, respectively. Fig. 21 (b) further supports this observation. When the required FPS is set to 20, LeapNet-2 retains all the layers of ResNet-18 to maximize the inference latency of the classifier. Conversely, when the required FPS is set to 50, LeapNet-2 drops all the selectable layers of ResNet-18 to minimize inference latency, retaining only a convolutional layer and a fully connected layer to ensure baseline performance.

Furthermore, we evaluate the impact of the number of objects per frame on the actual FPS. To assess system limits, we send a fixed number of objects per frame to the classifier. This serves as a stress test for LeapNet to process objects beyond just traffic signs. Fig. 22 shows the measured FPS and the layer retention ratio for LeapNet-2 across different average object counts. Fig. 22 (a) employs settings identical to Fig. 21, while Fig. 22 (b) uses YOLO with an input size of 832×704. The target FPS is set at 30 in Fig. 22 (a) and 10 in Fig. 22 (b). Note that these settings are adjustable based on

Table 4. Comparisons with existing defense methods.

	BlockDrop [23]	BaRT [18]	AdaptiveNet [26]	Sardino [2]	LeapNet-1	LeapNet-2
Dynamic Defense	✗	✓	✗	✓	✓	✓
Computation-efficient	✓	✗	✓	✗	✓	✓
Latency-aware	✗	✗	✓	✓	✗	✓
Real-time Adaption	✗	✗	✗	✓	✗	✓

the specified embedded system requirements. The results show that the FPS decreases significantly as the average number of objects increases. For instance, when there are five objects on average in each frame, the FPS drops to 13.8, less than half of the required FPS. In contrast, LeapNet-2 maintains stable FPS across varying object counts as required. Besides, LeapNet-2 achieves slightly lower FPS performance than ResNet-18 when the object count is low, due to a minor overhead (<3%) introduced by its decision network.

7 Related Work

In this section, we first review related work about adversarial defense, including deterministic and dynamic defense. Next, we discuss the adaptation of DNNs for embedded devices.

■ **Deterministic Defense.** Several strategies have been proposed to address security concerns regarding adversarial examples that can mislead DNN models. Adversarial training [15] includes adversarial examples with correct labels in the training dataset. By iteratively training the DNN on these examples, the model enhances its robustness against adversarial attacks. Several image processing methods also mitigate the influence of adversarial attacks, like lossy compression [48] and feature-squeezing [16], by disrupting or removing subtle perturbations introduced by adversaries. For example, lossy compression introduces quantization that eliminates small, imperceptible changes, while feature-squeezing reduces the dimensionality of the input image and applies spatial smoothing to constrain the adversary's search space. Besides, static ensemble [49] uses an ensemble of models to make joint decisions with improved robustness. Gradient masking [17] alters the gradients of the target DNN to obstruct the effectiveness of gradient-based attacks. However, as the above mechanisms are deterministic, they are vulnerable to adaptive attacks where adversaries can learn the defense strategies and develop new attack iterations.

■ **Dynamic Defense.** Dynamic defense strategies challenge the adaptive attacker in constructing effective attacks. Stochastic pre-processing defenses [18] apply randomized input transformations to the input. A "random resizing and padding layer" [50] is incorporated at the beginning of the DNN architecture to proactively defend against attacks. BaRT [18] builds a robust defense by stochastically combining numerous individual transformations into a randomized barrage. Besides, SAP [19] introduces randomness into the activation function. However, as these methods rely on weak stochasticity, they are vulnerable to attacks targeting the entire set of random functions [51, 52]. Dynamic ensembles capture broader variability by leveraging predictions from multiple models [2]. However, combining multiple models incurs extra computational overheads, which are impractical for resource-constrained embedded devices. Our proposed LeapNet counteracts adaptive attacks by dynamically adapting the layer routes while reducing computational redundancy.

■ **DNNs Adaption for Embedded Systems.** Recent studies focus on efficient model deployment for embedded scenarios to satisfy resource constraints, as most DNN models are computationally heavy. Model compression methods, including pruning [20] and quantization [21], shrink model sizes for lower storage and computation costs. Other model scaling methods, like early exit [22], layer routing [23], and width scaling [24], simplify model structures without compromising their accuracy. However, these methods are, in general, not designed to address dynamic resource and latency budgets. An automated model adaptation has also received research attention. NAS [26, 28], as a relevant model generation solution, attempts to discover architectures that meet specific requirements. However, even recent methods like AdaptiveNet [26] require minutes to update the model, making real-time adaptation infeasible. Besides, Sadino [2] introduces a real-time dynamic adaptation method by adjusting the ensemble size. However, on resource-constrained devices, Sadino is only suitable for small DNN models because of the use of multiple models for inference.

8 Discussion

While LeapNet is designed for CNN-based architectures, particularly in the vision domain, the core idea of stochastic layer routing holds potential for broader applications, like Transformer-based models. Vision transformers (ViTs) have gained popularity due to their strong representational capabilities. Several works, such as AdaViT [53] have shown that dynamically selecting transformer layers or tokens based on input characteristics is feasible for reducing computation. Inspired by this, our method could be extended to apply stochasticity in selecting encoder layers or attention blocks to enhance robustness. However, such extensions introduce challenges due to the global and interdependent nature of transformer computations. Randomly dropping layers may disrupt this aggregation process and result in unstable representations. Additionally, training stability and convergence issues may become more pronounced due to the deeper and more interdependent structure of transformers. Despite these challenges, applying stochastic routing to transformers is interesting for future work.

LeapNet produces a layer routing distribution that results in an average inference time meeting a relatively stable latency requirement. However, this design may fall short under highly dynamic latency requirements, where strict per-frame latency bounds are necessary. To address this, future work could explore adaptive mechanisms such as Proportional-Integral (PI) control, which dynamically adjusts execution paths based on recent latency deviations. For instance, if one frame exceeds the target budget, subsequent frames could drop less critical layers to compensate. Enabling such real-time adaptation may require reinforcement learning–based routing decision networks to learn the relationship between latency and each layer while preserving randomness for adversarial robustness.

9 Conclusion

In this paper, we propose LeapNet, comprising two variants (i.e., LeapNet-1 and LeapNet-2), for resource-constrained embedded vision systems to maintain robustness against adversarial adaptive attacks and real-time adaptation for embedded deployment. Specifically, LeapNet-1 learns the optimal layer routing distribution and dynamically samples layer routes for inference to enhance robust accuracy while preserving clean accuracy and reducing computational redundancy. LeapNet-2 further optimizes the learned optimal layer routing to adapt to the dynamic embedded environment under time-series settings. We conduct extensive experiments on various popular datasets and real-world outdoor scenarios to evaluate LeapNet. The effectiveness of LeapNet-1 and LeapNet-2 is demonstrated in comparison with recent state-of-the-art static and dynamic defense methods. Moreover, LeapNet-2 can flexibly adapt the layer routes to meet the evolving demands of the time-series embedded vision system, achieving real-time adjustments.

References

[1] Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. Bringing ai to edge: From deep learning's perspective. *Neurocomputing*, 485:297–320, 2022.

[2] Qun Song, Zhenyu Yan, Wenjie Luo, and Rui Tan. Sardino: Ultra-fast dynamic ensemble for secure visual sensing at mobile edge. In *Proceedings of the 2022 International Conference on Embedded Wireless Systems and Networks (EWSN)*,

pages 24–35, 2022.

[3] Donghwa Kang, Seunghoon Lee, Cheol-Ho Hong, Jinkyu Lee, and Hyeongboo Baek. Batch-mot: Batch-enabled real-time scheduling for multi-object tracking tasks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024. Presented at The ACM SIGBED International Conference on Embedded Software (EMSOFT).

[4] Jiaming Qiu, Ruiqi Wang, Ayan Chakrabarti, Roch Guérin, and Chenyang Lu. Adaptive edge offloading for image classification under rate limit. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):3886–3897, 2022. Presented at The ACM SIGBED International Conference on Embedded Software (EMSOFT).

[5] Xisheng Li, Ye Ma, Yuting Chen, Jinghao Sun, Wanli Chang, Nan Guan, Liming Chen, and Qingxu Deng. Priority optimization for autonomous driving systems to meet end-to-end latency constraints. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, pages 402–414. IEEE, 2024.

[6] Bashima Islam and Shahriar Nirjon. Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(3):1–29, 2020.

[7] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[8] Meng Shen, Zelin Liao, Liehuang Zhu, Ke Xu, and Xiaojiang Du. Vla: A practical visible light-based attack on face recognition systems in physical world. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(3):1–19, 2019.

[9] Brian Testa, Yi Xiao, Harshit Sharma, Avery Gump, and Asif Salekin. Privacy against real-time speech emotion detection via acoustic adversarial evasion of machine learning. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 7(3):1–30, 2023.

[10] Ahmet Soyyigit, Shuochao Yao, and Heechul Yun. Valo: A versatile anytime framework for lidar-based object detection deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):4045–4056, 2024. Presented at The ACM SIGBED International Conference on Embedded Software (EMSOFT).

[11] Yuhang Xu, Zixuan Liu, Xinzhe Fu, Shengzhong Liu, Fan Wu, and Guihai Chen. Flex: Adaptive task batch scheduling with elastic fusion in multi-modal multi-view machine perception. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, pages 294–307. IEEE, 2024.

[12] The evolving safety and policy challenges of self-driving cars, 2024. https://www.brookings.edu/articles/the-evolving-safety-and-policy-challenges-of-self-driving-cars/.

[13] Tyler Ward. Areas of improvement for autonomous vehicles: A machine learning analysis of disengagement reports. In *2024 4th Interdisciplinary Conference on Electrics and Computer (INTCEC)*, pages 1–6. IEEE, 2024.

[14] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018.

[15] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[16] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society, 2018.

[17] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[18] Edward Raff, Jared Sylvester, Steven Forsyth, and Mark McLean. Barrage of random transforms for adversarially robust defense. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6528–6537, 2019.

[19] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy D Bernstein, Jean Kossaifi, Aran Khanna, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018.

[20] Aojun Zhou, Yang Li, Zipeng Qin, Jianbo Liu, Junting Pan, Renrui Zhang, Rui Zhao, Peng Gao, and Hongsheng Li. Sparsemae: Sparse training meets masked autoencoders. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16176–16186, 2023.

[21] Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1972–1981, 2023.

[22] Nafiul Rashid, Berken Utku Demirel, Mohanad Odema, and Mohammad Abdullah Al Faruque. Template matching based early exit cnn for energy-efficient myocardial infarction detection on low-power wearable devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(2):1–22, 2022.

[23] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8817–8826, 2018.

[24] Haichen Shen, Jared Roesch, Zhi Chen, Wei Chen, Yong Wu, Mu Li, Vin Sharma, Zachary Tatlock, and Yida Wang. Nimble: Efficiently compiling dynamic neural networks for model inference. *Proceedings of Machine Learning and Systems*, 3:208–222, 2021.

[25] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Joseph E Gonzalez, and Ion Stoica. D3: a dynamic deadline-driven approach for building autonomous vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 453–471, 2022.

[26] Hao Wen, Yuanchun Li, Zunshuai Zhang, Shiqi Jiang, Xiaozhou Ye, Ye Ouyang, Yaqin Zhang, and Yunxin Liu. Adaptivenet: Post-deployment neural architecture adaptation for diverse edge environments. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2023.

[27] Stopping distances: Vehicle speed, crash risk, thinking and braking time, 2016. https://www.qld.gov.au/transport/safety/road-safety/driving-safely/stopping-distances.

[28] Qinsi Wang and Sihai Zhang. Dgl: device generic latency model for neural architecture search on mobile devices. *IEEE Transactions on Mobile Computing*, 2023.

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[30] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

[31] Nicolai Wojke and Alex Bewley. Deep cosine metric learning for person re-identification. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 748–756. IEEE, 2018.

[32] Dji mavic air 2 user manual, 2020.05. https://dl.djicdn.com/downloads/Mavic_Air_2/20200511/Mavic_Air_2_User_Manual_v1.0_en.pdf.

[33] 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.

[34] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 409–424, 2018.

[35] Radu Timofte, Karel Zimmermann, and Luc Van Gool. Multi-view traffic sign detection, recognition, and 3d localisation. *Machine vision and applications*, 25:633–647, 2014.

[36] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020.

[37] scikit-learn, 2024. https://scikit-learn.org/stable/.

[38] Li auto inc. announces the adoption of nvidia's next generation autonomous driving smart chip orin., 2020. https://ir.lixiang.com/news-releases/news-release-details/li-auto-inc-announces-adoption-nvidias-next-generation/.

[39] Yann Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[40] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[41] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[42] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *European Conference on Computer Vision*, pages 484–501, 2020.

[43] Neale Ratzlaff and Li Fuxin. Hypergan: A generative model for diverse, performant neural networks. In *International Conference on Machine Learning*, pages 5361–5369. PMLR, 2019.

[44] The intelligence driving technogy of lixiang 9., 2025. https://driveevgh.com/cars/li-xiang-l9/.

[45] Takami Sato, Sri Hrushikesh Varma Bhupathiraju, Michael Clifford, Takeshi Sugawara, Qi Alfred Chen, and Sara Rampazzi. Invisible reflections: Leveraging infrared laser reflections to target traffic sign perception.

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

[47] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[48] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M Roy. A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853*, 2016.

[49] Thilo Strauss, Markus Hanselmann, Andrej Junginger, and Holger Ulmer. Ensemble methods as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arXiv:1709.03423*, 2017.

[50] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018.

[51] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *Advances in neural information processing systems*, 33:1633–1645, 2020.

[52] Chawin Sitawarin, Zachary J Golan-Strieb, and David Wagner. Demystifying the adversarial robustness of random transformation defenses. In *International Conference on Machine Learning*, pages 20232–20252. PMLR, 2022.

[53] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12309–12318, 2022.