AegisDNN: Dependable and Timely Execution of DNN Tasks with SGX

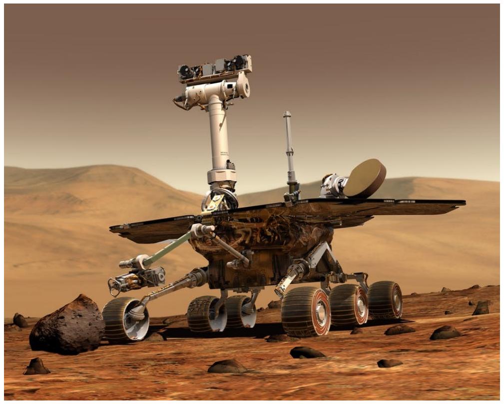
Yecheng Xiang, Yidi Wang, Hyunjong Choi, Mohsen Karimi and Hyoseung Kim

RTSS 2021

Introduction

- Rising usage of emerging DNN applications in safety-critical systems.

Autonomous-driving Vehicles

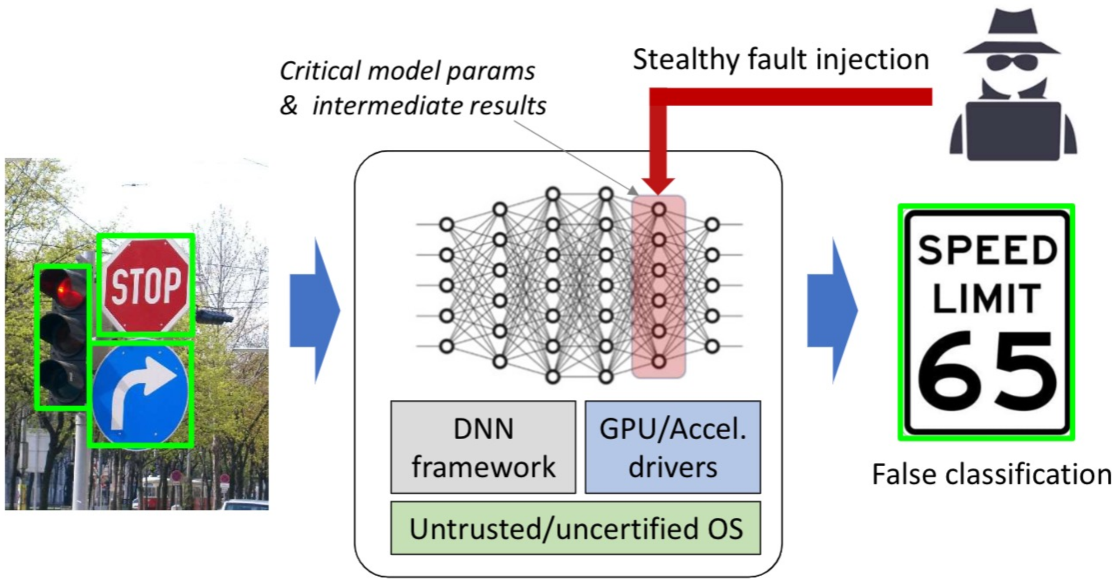


Robotics

Defense

Introduction

- Erroneous outputs in such systems can have catastrophic consequences.



Introduction

- <u>Late</u> outputs in such systems are also not acceptable.

Introduction

- To ensure the system function and safety, we need DNN execution:
  - "**Dependable**" against fault-injection attacks
  - "**Timely**" against task deadlines
- We propose AegisDNN to address dependability and timeliness simultaneously.

Related Work

- Modern DNN frameworks, e.g., PyTorch, TensorFlow, and Caffe
  - do not provide any run-time protection against fault-injection attacks, and
  - do not provide real-time performance guarantee

- Prior work provides
  - either real-time performance guarantee, e.g., DART[1],
  - or privacy protection using **Intel SGX** against malicious attackers on cloud systems, e.g., Serdab[2], Privado[3], Occlumency[4].

[1] Xiang et al. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. (RTSS, 2019)
[2] Elgamal et al. Serdab: An IoT framework for partitioning neural networks computation across multiple enclaves.
[3] Grover el al. Privado: Practical and secure DNN inference with enclaves.
[4] Lee et al. Occlumency: Privacy-preserving remote deep-learning inference using sgx. (MobiCom, 2019)

Intel SGX

Intel SGX is a hardware-assisted security extension.

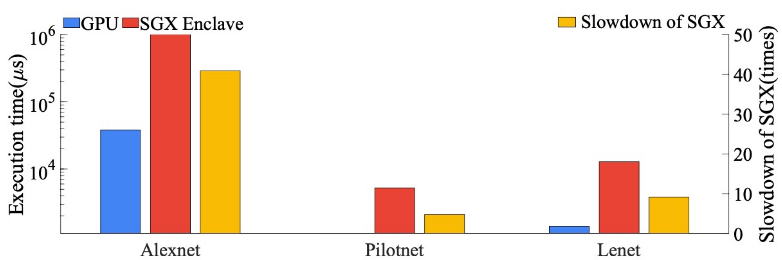
- It provides a software abstraction, called enclave.

- Code and data contents in the enclave are protected.
  - Encrypted and stored in the Processor Reserved Memory (PRM) (max 128MB)

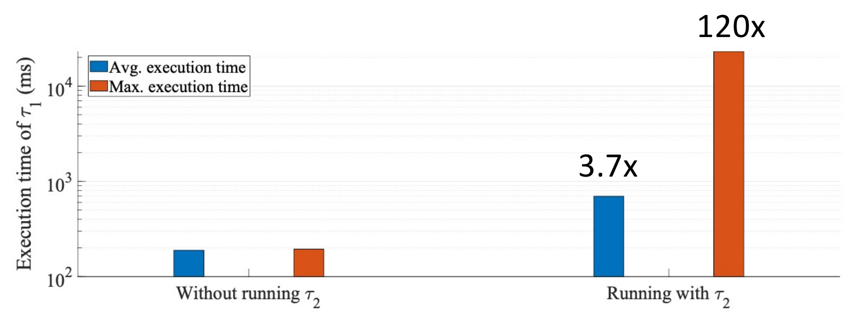
- Execution model: Similar to GPU execution model(H2D, Kernel, D2H)

Challenges

- **Significant Performance Overhead**
  - ~5x to ~40x slowdown
  - due to extra memory copy, data encryption, and CPU-only execution
- **Memory Thrashing Issue**
  - Caused by small SGX memory



Slowdown



Memory thrashing: Unpredictable WCET

Contributions

- **AegisDNN**: **Dependable** and **Timely** Execution of DNN Tasks with SGX

- Key Contributions**:**
  - **The first work aiming at dependable and timely DNN inference execution simultaneously**
  - **Leverage SGX for protecting only the critical parts of real-time DNN tasks against fault injection attacks**
  - **Designed amenable to formal real-time schedulability analysis**

System Model

- System is equipped with a **GPU** and a Intel **SGX Enclave**.
- Explicit data transmission is required between enclave and main memory.
- Both enclave and GPU are treated as **mutual exclusive resources**, we use **lock-base synchronization** to solve the unpredictability of memory thrashing challenge**.**
- SGX page swapping is enabled to support large DNN models.

Task Model

- Sporadic task model
- Each task uses one DNN model

General Task Model

$$\tau_i := (C_i, T_i, D_i, N_i, M_i)$$

WCET, min inter-arrival time, deadline, # of layers, DNN model used

Layer Execution Model

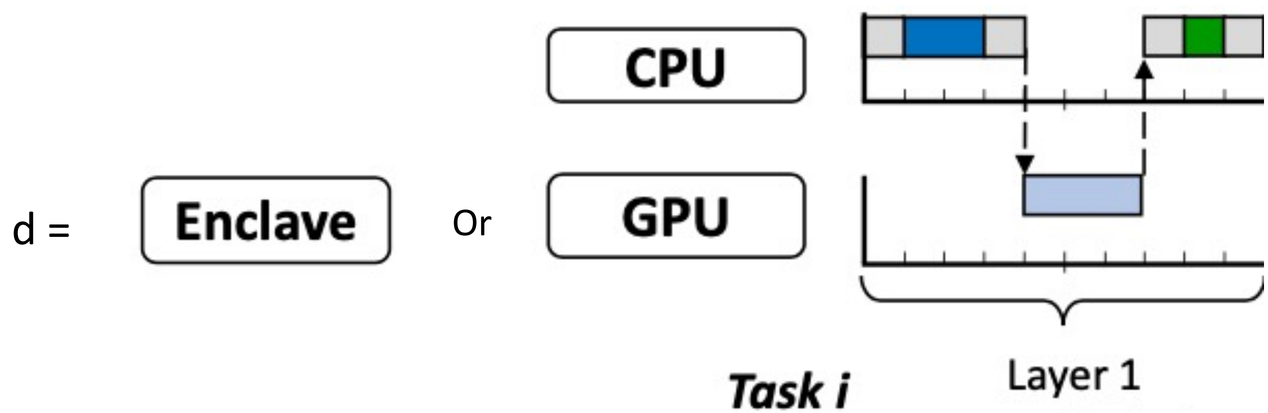
$$C_{i,j}(d) := (C^{hd}_{i,j}(d), C^{e}_{i,j}(d), C^{dh}_{i,j}(d), C^{m}_{i,j}(d))$$

Task Model

H2D memcpy, Kernel execution, D2H memcpy, misc. CPU operations

$$C_{i,j}(d) := (C_{i,j}^{hd}(d), C_{i,j}^{e}(d), C_{i,j}^{dh}(d), C_{i,j}^{m}(d))$$

$C_{i,j}^{hd}(d)$ $C_{i,j}^{e}(d)$ $C_{i,j}^{dh}(d)$ $C_{i,j}^{m}(d)$



Threat and Fault Model

- **Dependability**: the capability to ensure the integrity of output generated by real-time DNN tasks in the presence of malicious fault injection attacks

- Trusted: CPU chip package, SGX, enclaves.

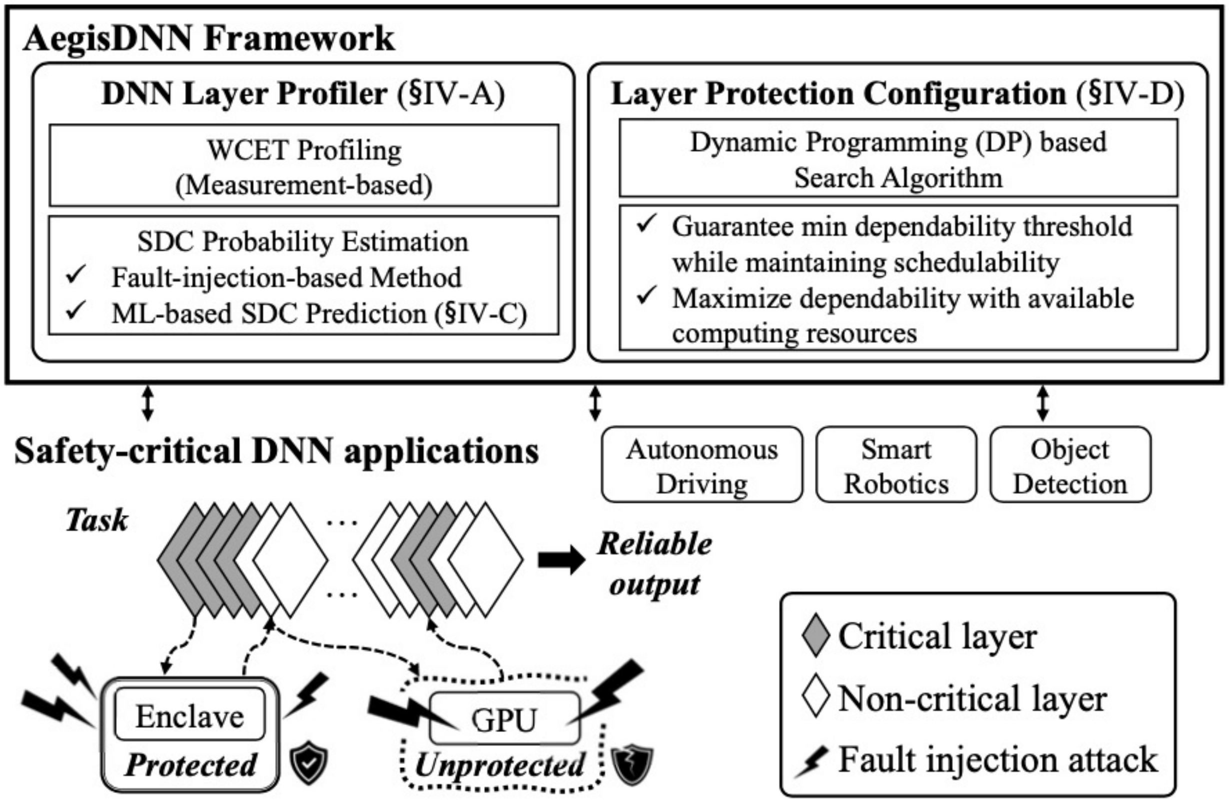
- Untrusted:
  - Off-chip hardware, e.g., GPUs, DRAM, memory bus
  - Software components running out of enclave are all untrusted, including OS, device drivers, middleware, libraries and etc.

- The degree of faults is quantified by Bit Error Rate (**BER**)
  - # of fault bits / # of total bits

Threat and Fault Model

- Only consider **stealthy** attacks.
- The faults can be induced by either physical attacks or software attacks.
- Silent Data Corruption (**SDC**) probability as a metric to evaluate the dependability of the system.
  - SDC + Dependability = 1
- SDC probability: the probability of compromised DNN output
  - TOP-1
  - E.g., 1% SDC probability means 1 out of 100 outputs is compromised and generate different TOP-1 result from its fault-free execution

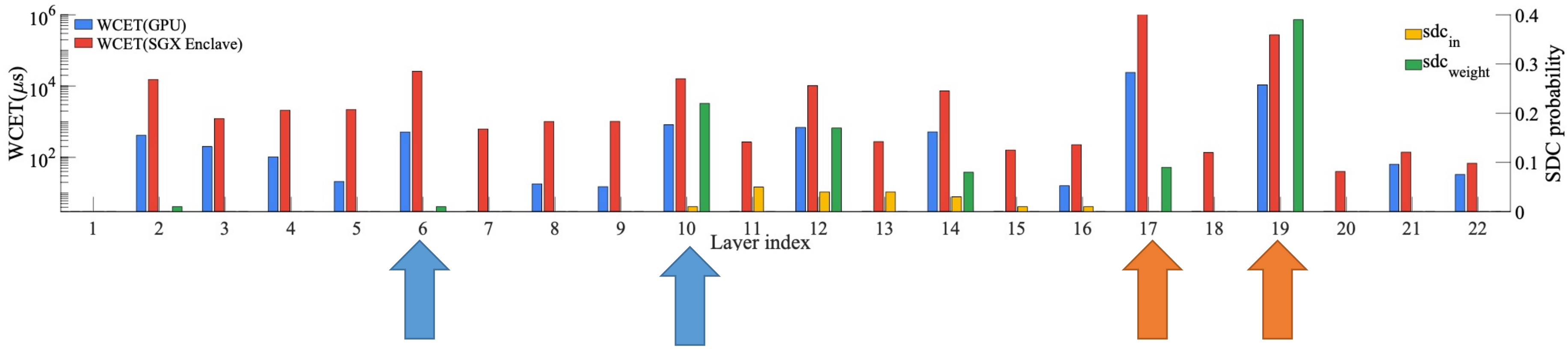
AegisDNN – Overview



DNN Layer-wise Profiler

- WCET Profile
- SDC Profile – SDC_{In} & SDC_{weight}

AlexNet layer-wise profile



Similar Slowdown, much higher SDC
-> Better to protect layer 10

Similar Slowdown, much higher SDC
-> Better to protect layer 19

What Layers to Protect?

- **<u>SDC probability</u> of a model if protecting a combination of layers?**
  - Can achieve dependability requirement?
  - Naïve solution: Run fault-injection and estimate the SDC probability for all the possible protection methods
    - Complexity: Exponential (2^number of layers)

<u>Dependable</u>

- **Can we guarantee the <u>schedulability</u> if protecting a combination layers?**
  - Real-time schedulability analysis

<u>Timely</u>

Predicting SDC Probability

- ML Approach: Linear Regression
- Key Idea:
  - Each layer has a linear contribution to the overall SDC probability when protecting a combination of layers

$$\hat{y}_i = c_i + \sum_{j=1}^{N_i} \alpha_{i,j} x_{i,j}^{in} + \sum_{j=1}^{N_i} \beta_{i,j} x_{i,j}^{weights}$$

- Steps:
  - Step 1: Uniformly-distributed training sample
  - Step 2: Train the Linear Regression Model
  - Step 3: Generate Comprehensive SDC profile

Predicting SDC Probability

ML prediction accuracy

DNN model	Cross-validation MAE%	Ground-truth MAE%
Pilotnet	2.14	1.03
Lenet	4.55	4.32
Alexnet	1.21	-
Resnet-18	4.80	-

Cross-validation and Ground-truth Validation

Time required for generating the SDC profile

DNN model	Training	Pred. All Config.	Est. FI All Config.
Pilotnet	3.75h	1.27s	59.84h
Lenet	0.56	0.2s	2.25h
Alexnet	72hr	0.5h	33yr[11]
Resnet-18	28hr	0.4h	17yr[11]

Significant Time Saving

[11]This is an estimate based on the speed of progress on our tested platform.

What Layers to Protect?

- **SDC probability of a model if we protect a combination of layers?**
  - Can achieve dependability requirement?
  - ~~Naïve solution: Run fault-injection and estimate the SDC probability for all the possible protection methods~~
    - ~~Complexity: Exponential (2^number of layers)~~
  - ML Solution: Linear Regression

Dependable

- **Can we guarantee the schedulability if protecting a combination of layers?**
  - Real-time Schedulability Analysis

Timely

Schedulability Conditions

Timely?

- Soft real-time systems: LST -> $\sum_{\tau_i \in \Gamma} \mathbf{U}_i^{\mathbf{D}}[1, N_i, k_{max}] \leq 1$

- Hard real-time systems: fixed-priority scheduling:
  - Mutual exclusive device
  - MPCP

$$R_i = C_i + B_i + \sum_{\substack{\pi_h > \pi_i \\ \mathbb{P}_h = \mathbb{P}_i}} \lceil * \rceil \frac{R_i}{T_h} (C_h + B_h) + \sum_{d \in \{g,e\}} \max_{\substack{\pi_l < \pi_i \\ \mathbb{P}_l = \mathbb{P}_i \\ 1 \leq j \leq K_l}} C_{l,j}^*(d)$$

$$B_i = \sum_{1 \leq j \leq K_i} B_{i,j}(type(\tau_{i,j}))$$

$$B_{i,j}(d) = \max_{\substack{1 \leq w \leq K_l \\ \pi_l < \pi_i}} C_{l,w}^*(d) + \sum_{\substack{d = type(\tau_{h,x}) \\ 1 \leq x \leq K_h \\ \pi_h > \pi_i}} \left(\lceil * \rceil \frac{B_{i,j}(d)}{T_h} + 1 \right) C_{h,x}^*(d)$$

Finding Layer Protection Configurations

- **Known:** for each combination of protected layers (i.e., layer protection config)
  - Comprehensive SDC profile -> **whether** **dependable?** ✓
  - Comprehensive sched analysis based on WCET profile -> **whether** **timely?** ✓
- **Decide**: Which combination of layers to protect?
- **Goal**: Max **dependability** while **satisfying schedulability requirement**
- Exhaustive Search
  - Go through each combination for each task
  - Exponential Complexity! $2^{\sum_{\tau_i \in \Gamma} N_i}$

Finding Layer Protection Configurations

- We propose a **Dynamic-Programming (DP)** based algorithm
  - Polynomial Complexity

- How it works?
  - **Minimize utilization need** for each task (DP)
  - **Maximize dependability** using available system resource

Finding Layer Protection Configurations

- How it works?
  - **Minimize utilization need** for each task (DP)
  - **Maximize dependability** using available system resource

- $U^D[i,j,k]$ -> Min utilization while protecting up to k **continuous subsequence** from layer i to layer j and meeting the dependability requirement D.

- We use DP to calculate the min required utilization for each task in the taskset.

Finding Layer Protection Configurations

- How it works?
  - **Minimize utilization need** for each task (DP)
  - **Maximize dependability** using available system resource

Algorithm 1 Finding layer protection configuration of all tasks

Require: $\Gamma = \{\tau_1, \tau_2, \tau_3, ..., \tau_n\}$: taskset
Require: $\mathbf{D}$: minimum dependability threshold
Require: $\mathbf{D_s}$: a set of search dependability values including $\mathbf{D}$
Require: $\mathbf{K_s}$: a set of candidate k values used in Eqs. (4.2) and (4.3)
Ensure: $\mathbf{S}^{\text{sol}} = \{S_1^{\text{sol}}, S_2^{\text{sol}}, ..., S_n^{\text{sol}}\}$: Solution layer protection configuration for each task; $\mathbf{S}^{\text{sol}} = \emptyset$, if failed.

function FIND_SOLUTION(Γ, $\mathbf{D}$,$\mathbf{D_s}$, $\mathbf{K_s}$)
    $\mathbf{S}^{\text{sol}} = \emptyset$ /* initialization */
    $k_{max} = \max(\mathbf{K_s})$
    for all $\tau_i \in \Gamma$ **do**
        for all $d \in \mathbf{D_s}$ **do**
            for all $k \in \mathbf{K_s}$ **do**
                Compute $\mathbf{U}_i^d[1, N_i, k]$ by Eqs. (4.2) and (4.3)
                Store $\mathbf{S}_i^d[1, N_i, k]$ accordingly
    $\mathbf{S}^{\text{sol}} = \{\mathbf{S}_1^{\mathbf{D}}[1, N_1, k_{max}], ..., \mathbf{S}_n^{\mathbf{D}}[1, N_n, k_{max}]\}$
    if Taskset Γ is feasible under $\mathbf{S}^{\text{sol}}$ **then**
        for all $d \in \mathbf{D_s}$ in descending order **do**
            for all $\tau_i \in \Gamma$ **do**
                Replace the i-th term in $\mathbf{S}^{\text{sol}}$ with $\mathbf{S}_i^d[1, N_i, k_{max}]$
            if Taskset Γ is feasible under $\mathbf{S}^{\text{sol}}$ **then**
                /* The best solution is found for τ_i*/
            else
                for all $\tau_i \in \Gamma$ **do**
                    Restore the old i-th config in $\mathbf{S}^{\text{sol}}$
    else
        return $\mathbf{S}^{\text{sol}} = \emptyset$ /* no solution*/
end function

- STEP1:
  - Compute all the **U** for all tasks in the taskset
  - Given dependability requirement D, we check whether taskset is feasible

- STEP2:
  - If not feasible -> no solution available
  - If feasible -> find the maximum system dependability while taskset is still feasible

Evaluation

- **Hardware Specs:**
  - Intel 7700K Quad-core, with SGX enabled
  - 16GB RAM
  - Maximum 128 MB of encrypted SGX memory
  - RTX 2080 Super

- **DNN Models**: ResNet-18, AlexNet, PilotNet, LeNet

- **Attacks Considered:**
  - Random-fault-injection (RANFI) from TensorFI[1] and Ares[2] (FP models)
  - Target-fault-injection (TFI) from BinFI[3] (FP models)
  - Bit-flip attack (BFA) with progressive bit search[4] (on quantized INT8 models)

[1] Z. Chen et al. TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications. (ISSRE, 2020)
[2] B Reagen et al. Ares : A framework for quantifying the resilience of deep neural networks. (DAC, 2018)
[3] Z. Chen et al. BinFI an efficient fault injector for safety-critical machine learning systems. (SC, 2019)
[4] A. Rakin. Bit-Flip Attack: Crushing Neural Network With Progressive Bit Search . (ICCV, 2019)

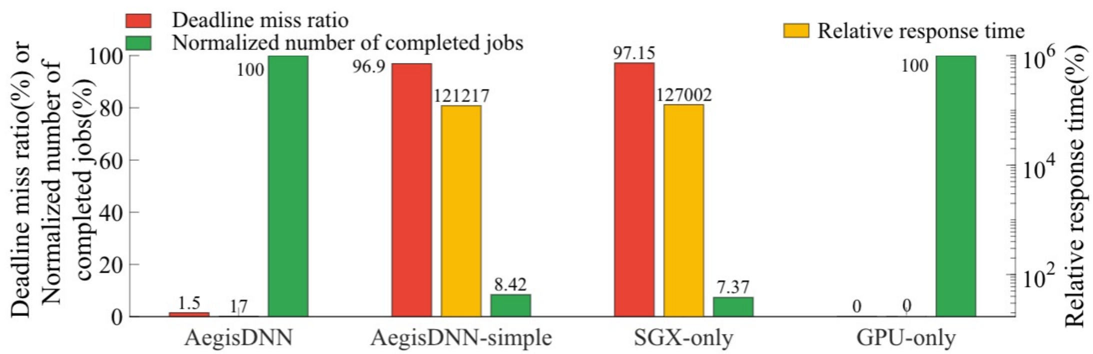
Integrated System Evaluation

Taskset 1			Taskset 2 (INT8-Quantized)		
Task	**DNN model**	**Deadline**	**Task**	**DNN model**	**Deadline**
1	LeNet	30 ms	1	ResNet-18	100 ms
2	LeNet	50 ms	2	ResNet-18	200 ms
3	PilotNet	50 ms	3	ResNet-18	200 ms
4	PilotNet	80 ms	4	ResNet-18	400 ms
5	AlexNet	200 ms	5	AlexNet	500 ms
6	AlexNet	250 ms	6	AlexNet	500 ms
7	AlexNet	300 ms			

RANFI & TFI **BFA**

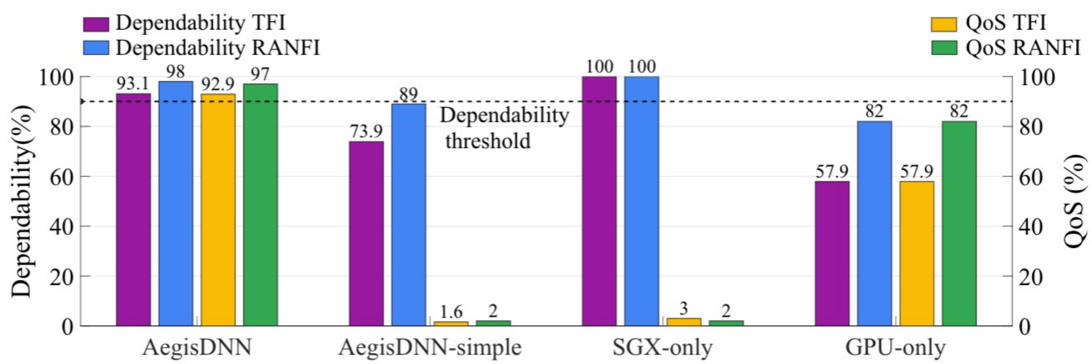
Integrated System Evaluation – Soft Real-time

QoS: Percentage of jobs finished both timely and dependably



(a) Taskset 1 with BER=1e-6

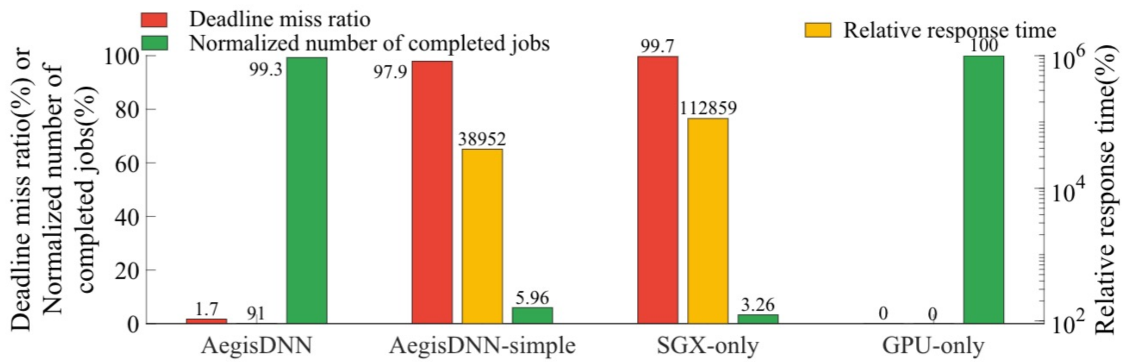
AegisDNN **meets** Dependability requirement and **dominates** other approaches



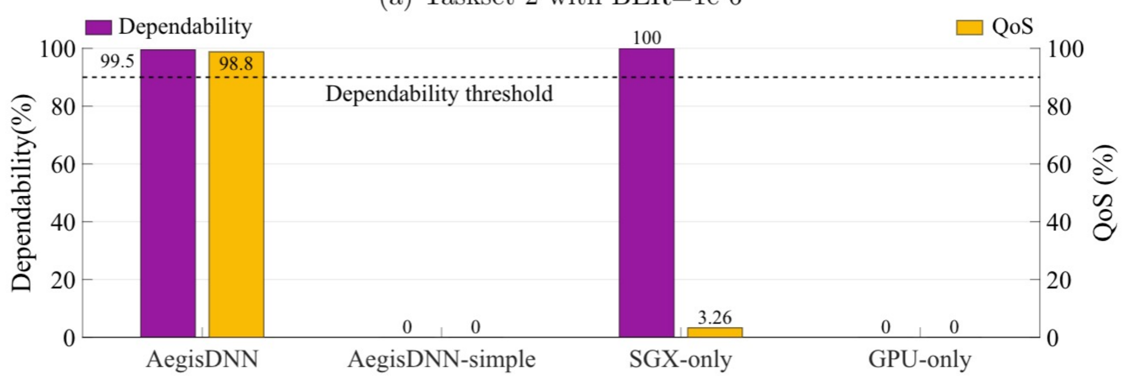
(b) Taskset 1 with BER=1e-6 (Dependability and QoS)

Integrated System Evaluation – Soft Real-time

QoS: Percentage of jobs finished both timely and dependably



(a) Taskset 2 with BER=1e-6



(b) Taskset 2 with BER=1e-6 (Dependability and QoS)

AegisDNN **meets** Dependability requirement and **dominates** other approaches

Integrated System Evaluation – Hard Real-time

Taskset 1		
Task	DNN model	Deadline
1	LeNet	30 ms
2	LeNet	50 ms
3	PilotNet	50 ms
4	PilotNet	80 ms
5	AlexNet	200 ms
6	AlexNet	250 ms
7	AlexNet	300 ms

We found the taskset 1 could not be used with hard real-time constraints even if we lower the dependability requirements
(probably due to the analytical pessimism)

Modified

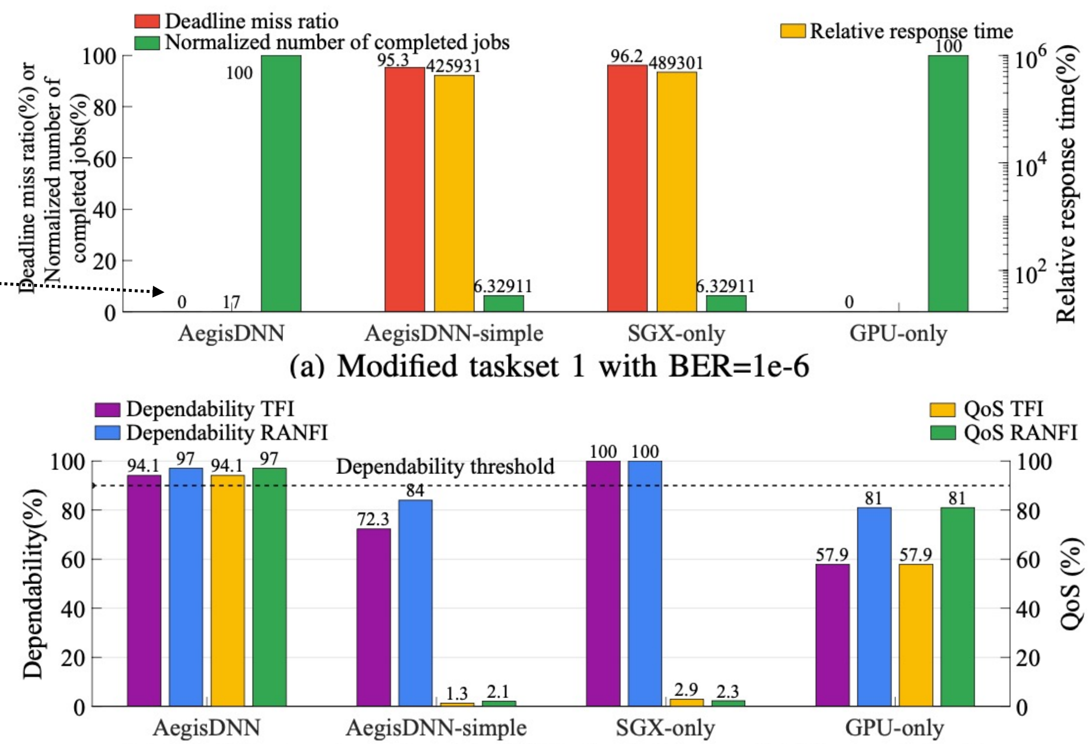
Task	DNN model	Deadline
1	LeNet	**100** ms
2	LeNet	50 ms
3	PilotNet	**100** ms
4	PilotNet	80 ms
5	AlexNet	**250** ms
6	AlexNet	250 ms
7	AlexNet	300 ms

Integrated System Evaluation – Hard Real-time

AegisDNN was able to guarantee the hard real-time constraints

Our hard real-time schedulability analysis can reject unsafe tasksets

AegisDNN **meets** Dependability requirement and **dominates** other approaches



(a) Modified taskset 1 with BER=1e-6

(b) Modified taskset 1 with BER=1e-6 (Dependability and QoS)

Conclusion

- We presented AegisDNN, a DNN inference framework for **timely** and **dependable** execution with SGX.

- We discussed the related work and challenges of using SGX.

- We solve the challenges by proposing AegisDNN:
  - layer-wise WCET and SDC profiling mechanisms
  - ML-based SDC prediction method
  - DP-based configuration-finding algorithm
  - Schedulability analysis

- We have implemented and evaluated against several state-of-the-art DNN fault-injection attacks.

- Experimental results indicate AegisDNN dominates the other approaches in many aspects, including response time, throughput, dependability, and QoS under both soft and hard real-time scenarios.

AegisDNN: Dependable and Timely Execution of DNN Tasks with SGX

Yecheng Xiang, Yidi Wang, Hyunjong Choi, Mohsen Karimi and Hyoseung Kim

Thank you!