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.
Introduction

• **Erroneous** outputs in such systems can have catastrophic consequences.
Introduction

• Late 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].

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
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

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

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

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

\[ d = \begin{cases} \text{Enclave} & \text{or} \end{cases} \begin{cases} \text{GPU} \end{cases} \]
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

AegisDNN Framework

DNN Layer Profiler (§IV-A)
- WCET Profiling (Measurement-based)
- SDC Probability Estimation
  - Fault-injection-based Method
  - ML-based SDC Prediction (§IV-C)

Layer Protection Configuration (§IV-D)
- Dynamic Programming (DP) based Search Algorithm
- Guarantee min dependability threshold while maintaining schedulability
- Maximize dependability with available computing resources

Safety-critical DNN applications

Task

Protected

Unprotected

Critical layer

Non-critical layer

Fault injection attack
DNN Layer-wise Profiler

• WCET Profile
• SDC Profile – $SDC_{ln}$ & $SDC_{weight}$

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

Similar Slowdown, much higher SDC
-> Better to protect layer 19
What Layers to Protect?

• **SDC probability** 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)

• Can we guarantee the **schedulability** if protecting a combination layers?
  • Real-time schedulability analysis
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

<table>
<thead>
<tr>
<th>DNN model</th>
<th>Cross-validation MAE%</th>
<th>Ground-truth MAE%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pilotnet</td>
<td>2.14</td>
<td>1.03</td>
</tr>
<tr>
<td>Lenet</td>
<td>4.55</td>
<td>4.32</td>
</tr>
<tr>
<td>Alexnet</td>
<td>1.21</td>
<td>-</td>
</tr>
<tr>
<td>Resnet-18</td>
<td>4.80</td>
<td>-</td>
</tr>
</tbody>
</table>

Cross-validation and Ground-truth Validation

<table>
<thead>
<tr>
<th>DNN model</th>
<th>Training</th>
<th>Pred. All Config.</th>
<th>Est. FI All Config.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pilotnet</td>
<td>3.75h</td>
<td>1.27s</td>
<td>59.84h</td>
</tr>
<tr>
<td>Lenet</td>
<td>0.56</td>
<td>0.2s</td>
<td>2.25h</td>
</tr>
<tr>
<td>Alexnet</td>
<td>72hr</td>
<td>0.5h</td>
<td>33yr\textsuperscript{11}</td>
</tr>
<tr>
<td>Resnet-18</td>
<td>28hr</td>
<td>0.4h</td>
<td>17yr\textsuperscript{11}</td>
</tr>
</tbody>
</table>

Significant Time Saving

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 ($2^{\text{number of layers}}$)
  • ML Solution: Linear Regression

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

- Soft real-time systems: LST \( \rightarrow \sum_{\tau_i \in I} U_P^{D}[1, N_i, k_{max}] \leq 1 \)

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

\[
R_i = C_i + B_i + \sum_{\pi_h > \pi_i, \pi_h = \pi_i} \left[ \sum_{d \in \{g, e\}} \max_{\pi_l < \pi_i, \pi_l = \pi_i} C_{i,j}^*(d) \right] + \sum_{\pi_l < \pi_i, \pi_l = \pi_i} \max_{1 \leq j \leq K_i} C_{i,j}^*(d)
\]

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

\[
B_{i,j}(d) = \max_{1 \leq w \leq K_i, \pi_l < \pi_i} C_{i,w}^*(d) + \left[ \sum_{\pi_l < \pi_i} \sum_{\pi_h > \pi_i, 1 \leq x \leq K_h} \left( \sum_{d = \text{type}(\tau_{h,x})} \left[ \frac{B_{i,j}(d)}{T_h} + 1 \right] C_{i,x}^*(d) \right) \right]
\]

Timely?
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

• $U^D[i,j,k] \rightarrow$ 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

```plaintext
Algorithm 1 Finding layer protection configuration of all tasks
Requires: \( \Gamma = \{\tau_1, \tau_2, ..., \tau_n\} \): taskset
Requires: \( D \): minimum dependability threshold
Requires: \( \mathbf{D}_k \): a set of search dependability values including \( D \)
Requires: \( \mathbf{K}_k \): a set of candidate \( k \) values used in Eqs. (4.2) and (4.3)
Ensures: \( \mathbf{S}^{\text{sol}} = \{\mathbf{S}^{\text{sol}}[1], \mathbf{S}^{\text{sol}}[2], ..., \mathbf{S}^{\text{sol}}[n]\} \): Solution layer protection configuration for each task; \( \mathbf{S}^{\text{sol}} = \emptyset \), if failed.

1: function FIND_SOLUTION(\( \Gamma \), \( \mathbf{D}_k \), \( \mathbf{K}_k \))
2: \( \mathbf{S}^{\text{sol}} = \emptyset \) /* initialization */
3: \( \mathbf{k}_{\text{max}} = \max[\mathbf{K}_k] \)
4: for all \( \tau_i \in \Gamma \) do
5:   for all \( k \in \mathbf{K}_k \) do
6:     Compute \( \mathbf{U}^{[1, N_i, k]} \) by Eqs. (4.2) and (4.3)
7:     Store \( \mathbf{S}[1, N_i, k] \) accordingly
8:     \( \mathbf{S}^{\text{sol}}[i] = \{\mathbf{S}^{[1, N_i, k]}[1], \mathbf{S}^{[1, N_i, k]}[2], ..., \mathbf{S}^{[1, N_i, k]}[n]\} \)
9:   if Taskset \( \Gamma \) is feasible under \( \mathbf{S}^{\text{sol}} \) then
10:   for all \( d \in \mathbf{D}_k \), in descending order do
11:     for all \( \tau_i \in \Gamma \) do
12:       Replace the i-th term in \( \mathbf{S}^{\text{sol}} \) with \( \mathbf{S}[1, N_i, k_{\text{max}}] \)
13:     if Taskset \( \Gamma \) is feasible under \( \mathbf{S}^{\text{sol}} \) then
14:       /* The best solution is found for \( \tau_i^{*} \) */
15:         end if
16:     else
17:       for all \( \tau_i \in \Gamma \) do
18:         Restore the old i-th config in \( \mathbf{S}^{\text{sol}} \)
19:     end for
20:     return \( \mathbf{S}^{\text{sol}} = \emptyset \) /* no solution* */
21:   end if
22: end function
```

- **STEP1:**
  - Compute all the \( \mathbf{U} \) 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)

---

Integrated System Evaluation

<table>
<thead>
<tr>
<th>Taskset 1</th>
<th>Task</th>
<th>DNN model</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>LeNet</td>
<td>30 ms</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>LeNet</td>
<td>50 ms</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>PilotNet</td>
<td>50 ms</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>PilotNet</td>
<td>80 ms</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>AlexNet</td>
<td>200 ms</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>AlexNet</td>
<td>250 ms</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>AlexNet</td>
<td>300 ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Taskset 2 (INT8-Quantized)</th>
<th>Task</th>
<th>DNN model</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>ResNet-18</td>
<td>100 ms</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>ResNet-18</td>
<td>200 ms</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>ResNet-18</td>
<td>200 ms</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>ResNet-18</td>
<td>400 ms</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>AlexNet</td>
<td>500 ms</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>AlexNet</td>
<td>500 ms</td>
</tr>
</tbody>
</table>

RANFI & TFI

BFA
Integrated System Evaluation – Soft Real-time

**QoS**: Percentage of jobs finished both **timely** and **dependably**

AegisDNN **meets** Dependability requirement and **dominates** other approaches
Integrated System Evaluation – Soft Real-time

**QoS**: Percentage of jobs finished both **timely** and **dependably**

*AegisDNN meets** Dependability requirement and **dominates** other approaches
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)
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.
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!