Protect Data Confidentiality for On-device Machine Learning through Split Inference

Yongzhi Wang
Department of Computer Science
Texas A&M University-Corpus Christi
Corpus Christi, TX, USA
yongzhiwang@icloud.com

Ahsan Habib
Department of Computer Science
Texas A&M University-Corpus Christi
Corpus Christi, TX, USA
ahabib2@islander.tamucc.edu

***Abstract*—On-Device Machine Learning (ODML) is a rapidly emerging paradigm, particularly suited for edge computing applications such as mobile computing, robotics, cyber-physical systems, etc. While ODML enables intelligent local processing, it introduces significant data confidentiality risks. When an application sends its data to an ODML model, there is potential for the model to eavesdrop on or leak this data to unauthorized third parties, thereby compromising sensitive information. To address this security concern, we propose a novel secure ODML architecture called Split Inference, designed to safeguard data confidentiality. Split Inference prevents compromised models from accessing or leaking application data by utilizing multi-enclave Trusted Execution Environment (TEE) technology. The architecture splits the model into three groups of layers: input, middle, and output. Each of the input and output group is isolated within its own respective TEE enclave, ensuring that the model input and output are protected against direct access. We also showed that it is challenging for the attacker to restore the model input and output based on the data in the unprotected middle group. As a result, application's data is protected at all stages of inference. We have implemented a prototype system for the MobileNet V1 model on a RISC-V-based multi-enclave TEE framework, Keystone. Experimental results demonstrated that Split Inference incurs an inference overhead ranging from 71% to 430%, outperforming the state-of-the-art solutions. We also discussed the application of the Split Inference on a wider spectrum of on-device models and hardware platforms.**

***Index Terms*—Trusted Execution Environment, RISC-V, MobileNet, Data Confidentiality, On-Device Machine Learning, Model Inference, Deep Learning, AI Security, Split Learning**

I. Introduction

On-Device Machine Learning (ODML) is an emerging paradigm that is widely used in various edge computing settings, such as mobile devices, autonomous systems, robotics, cyber-physical systems, etc. In ODML, pre-trained machine learning (ML) models are deployed to an edge device to provide *model inference services* (*MIS*es) for the *applications* (*APP*s). To provide intelligent processing, an APP on the device sends data to MISes as *inference requests* and receives *inference response*. Compared with the traditional on-cloud machine learning paradigm that requests the APP to receive inference response from the cloud, ODML avoids response latency, protects data secrecy, and supports offline intelligent processing.

While ODML is increasingly adopted, security concerns raise simultaneously. A malicious or vulnerable model could pose various threats that jeopardize the security of intelligent information processing [28]. In particular, a curious model could eavesdrop its input and output data exchanged with the APP and therefore compromise the data confidentiality of the APP.

In this paper, we aim to protect the input and output confidentiality of the ODML model using multi-enclave Trusted Execution Environment (TEE). The challenge is to use limited TEE resources to achieve high confidentiality and maintain low model inference delay. We propose *Split Inference*, which splits a model into three groups of layers, namely *input group*, *middle group*, and *output group*, and has each of the input and output group protected by a respective isolated enclave, namely *Input Enclave* and *Output Enclave*, respectively. By leveraging Remote Attestation of TEE, we ensure the input and output groups are free from confidentiality breach, therefore the model input and output are safely processed. To reduce performance overhead and TEE resource consumption, the middle group is not protected by an enclave and processed in plaintext. However, our analysis showed that Split Inference can effectively raised the bar for the attacker to restore the model input and output from the data observed in the middle group and can efficiently prevent the attacks by combining existing defense methods against data reconstruction and label inference attacks. As a result, the confidentiality of the model input and output will be protected. Different from most existing works utilizing single enclave [19], our work utilizes multi-enclave TEE to ensure separate isolation of the input and the output group. Even though all groups co-locate on one device, multi-enclave TEE ensures each group can run securely if other groups are compromised, therefore having an enhanced security guarantee. Additionally, multi-enclave TEE enables multiple ML models colocate in one device, supporting a broader spectrum of applications.

We implemented a prototype of the Split Inference framework for the MobileNet V1 [11] model on Keystone [14], a RISC-V based Multi-Enclave TEE framework. Our experiments showed that Split Inference incurs 71% to 430% of inference overhead. The state-of-the-art solutions in this direction [19], [26], [32], which protect minority but critical layers of the model using the TEE enclave and protect majority remaining layers through linear transformations, incur at best 142% to 577% of overhead (See Table 6 of [32]). By comparison, Split Inference outperforms existing work in performance.

Our contributions are as follows:

- By leveraging multi-enclave TEE, we proposed a secure on-device machine learning architecture, Split Inference, which protects the confidentiality of the model input and output against curious model. To the best of our knowledge, this paper is the first work applying split learning into on-device machine learning setting using TEE technology.
- To support Split Inference, we designed a communication protocol to ensure secure data communication among the APP and the enclaves.
- We implemented a Split Inference prototype for the MobileNet V1 model on a RISC-V multi-enclave TEE framework, Keystone.
- We run a series of experiments and analyzed the performance overhead. The result showed that Split Inference incurred smaller inference overhead comparing with latest TEE based solutions [32].

The paper is organized as follows. We introduced Preliminaries in Section II. We delineated the system and security assumptions in Section III. We introduced the system design in Section IV. We discussed the security in Section V. We described the implementation in Section VI. We presented the experiments in Section VII. We discussed application of Split Inference on other ML models and hardware platforms in Section VIII. We discussed related works in Section IX. We concluded the paper and the future work in Section X.

II. Preliminaries

A. *Trusted Execution Environment*

Trusted Execution Environment (TEE) [13] provides a secure area on the device to ensure the code and data loaded inside are protected with respect to confidentiality and integrity. By creating a hardware-isolated environment, named *enclave*, TEE ensures that information in an enclave remains inaccessible to unauthorized entities outside the enclave. *Remote Attestation* (RA) is a security mechanism provided by TEE that enables an application to prove its integrity to an authorized remote party. This process involves the generation of cryptographic evidence by the system's TEE to demonstrate that the data and software in an enclave has not been tampered with and will be executed as expected. During remote attestation, TEE produces a report or certificate, known as *measurement*, that includes a TEE signed cryptographic hash of the attested application in the enclave. The remote party can verify the signature and compare the hash against known good values to ensure the application's integrity.

1) Multi-enclave TEE: Extension of TEE from single-enclave to multi-enclave has been explored in recent years. Keystone [14] is a RISC-V based multi-enclave TEE framework. It allows each application, known as *host app*, to create more than one isolated enclaves using RISC-V's Physical Memory Protection (PMP) feature and deploy functions written in C/C++ into each enclave, running as an *enclave app*. Enclaves are running in both the User Mode (U-mode) and Supervisor Mode (S-mode) of the RISC-V system and are managed by the Security Monitor (SM), running in the Machine Mode (M-mode). Being designed as a general TEE framework, keystone users can customize the framework to satisfy their needs. They can select their own set of security primitives, including memory encryption, dynamic memory management, cache partitioning, etc.

Keystone provides a basic remote attestation mechanism: the verifier can verify the measurement (cryptographic hash) of the Security Monitor (SM) and the enclave (the enclave at initialization and the data block from the enclave of up-to 1KB) matches the expected value, therefore verifying that "the program and data in the enclave are as expected". This basic remote attestation mechanism does not address the issue of key distribution to and secure communication with enclaves, especially when multiple enclaves are mutually distrusting each other. In this paper, we introduced secure key distribution and communication protocols on top of the basic remote attestation mechanism.

B. *MobileNet*

MobileNet is a lightweight image classification neural network, which is suitable for resource constraint environment, such as edge devices. The MobileNet V1 [11] model introduces the depthwise separable convolution and includes the traditional DL layers such as fully connected layer and softmax. Fig. 1 showcases the basic architecture of MobileNet V1.

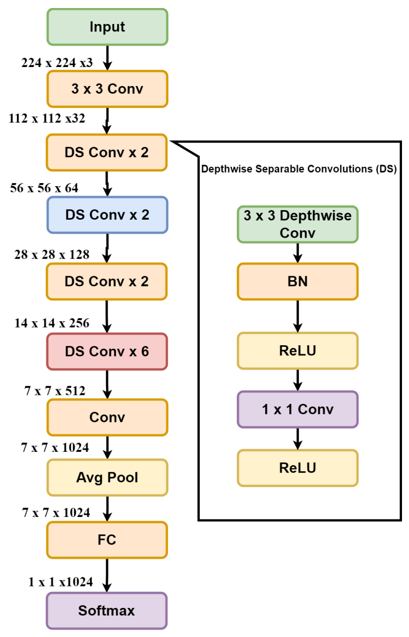

Fig. 1. Architecture of MobileNet V1

MobileNet V1 introduces width multiplier α and resolution multiplier ρ to balance the accuracy and performance. The width multiplier ranges from 0.25 to 1.0, with improved accuracy while increasing performance overhead. The resolution multiplier ranges from 128 to 224, indicating the input image size, which increases from 128x128 to 224x224 pixels. MobileNet has evolved by improving its performance, leading to MobileNet V2 [25] and V3 [10].

III. System and Security Assumptions

We assume the edge device supports multi-enclave TEE, which can provide multiple isolated enclaves that prevent direct access or modification to the information in them, including program and data. Data exchange with the enclave can be done only through an *edge call* defined by the TEE. The multi-enclave TEE supports the basic *remote attestation* protocol, which allows the enclave to attest the program and data loaded in the enclave. We assume the design and implementation of TEE is secure so that the attacker cannot directly eavesdrop the data computed, stored and communicated inside the enclave. We assume the remote attestation described above is securely designed and implemented so that the measurement of the content inside the enclave can be faithfully attested.

The focus of this paper is on protecting the confidentiality of the APP data sent to and received from the MIS. We assume that the MIS may eavesdrop on the data processed by the model in an attempt to reconstruct the model's input and output. To achieve that, we assume the attacker can compromise any component of the device except for the enclaves and the APP. As a result, the attack is able to eavesdrop on the entire model inference procedure outside of the TEE enclave.

We assume the attacker is "benign but curious", meaning it interests in breaching data confidentiality only, but not the data integrity. Protecting model inference integrity is an orthogonal problem, which will not be addressed in this paper.

IV. System Design

A. Split Inference Architecture

The architecture of Split Inference is shown in Fig. 2. Here, we use MobileNet V1 as a concrete example. In Section VIII-B, we show how to extend the design to other models. The Split Inference splits the MobileNet V1 model into three groups of layers, namely *input group*, *middle group*, and *output group*, as shown in Fig. 2. The input group is deployed to the Input Enclave, containing a convolution layer, followed by an activation operator. The output group is deployed to the Output Enclave, containing a fully connected layer followed with a `softmax` operator. The middle group contains all other layers between and are not protected by any enclave.

For each inference request, the APP encrypts and sends it to the Input Enclave. The Input Enclave decrypts the received input, processes the data through the layers in the input group, and sends the result to the middle group, which is outside any enclave. The output of the Input Enclave does not need to be encrypted so that it can be quickly processed by the middle layer. The data then is processed through layers in the middle group and sent to the Output Enclave. The Output Enclave receives the unencrypted output from the middle

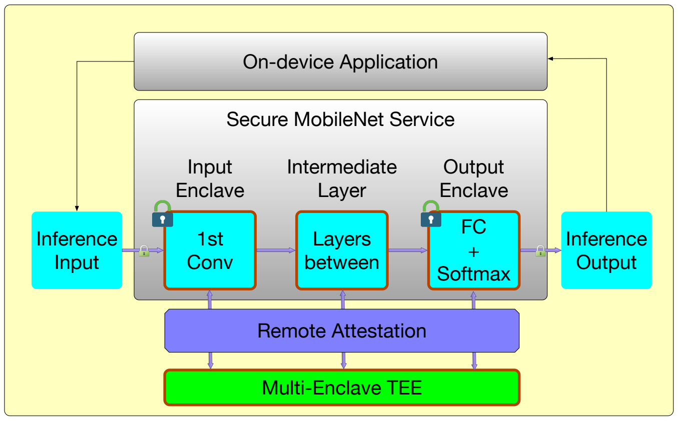

Fig. 2. Architecture of Split Inference in the case of MobileNetV1

group, processes the data through the output group, and sends the encrypted output back to the APP.

In Section V, we discussed how the confidentiality of the inference input and output is preserved. The intuition is as follows: Without knowing the parameters of the layers in the input group, the attacker in the middle group can hardly restore the inference input based on the output of the first layer. Similarly, the attacker in the middle layer can hardly restore the inference output: the inference output will be generated through a fully connected layer in the output enclave. Since the attacker cannot obtain the parameters in the fully connected layer, it lacks the information to derive the inference output.

B. Key Distribution Infrastructure and Protocol

To preserve the confidentiality of model input and output, we need to set up two secure communication channels, i.e., a channel between the APP and the Input Enclave, as well as a channel between the APP and the Output Enclave. A security channel between APP and the Input Enclave protects the confidentiality of the model input. A secure channel between the Output Enclave and the APP protects the confidentiality of the model output.

By leveraging the Remote Attestation provided by the TEE, we designed a key distribution protocol so that the APP can negotiate a unique encryption key with the Input Enclave and the Output Enclave, respectively. The key distribution infrastructure is shown in Fig. 3.

To support secure key distribution, we introduced a *Quote Enclave*, which stores the measurements of the Input Enclave and the Output Enclave and maintains a *Key Repository*, storing the public key of the APP. Any APP a who wants to establish a secure channel with an enclave needs to initialize a pair of public key, pk_a and sk_a. the public key pk_a will be safely deployed to the Key Repository in the Quote Enclave.

The Quote Enclave has a public key pair (pk_QE, sk_QE). It publishes pk_QE so that any party can send requests to the Quote Enclave secretly by encrypting the request with pk_QE.

As shown in Fig. 3, if an APP a wants to establish a security channel with an enclave p, it first generates an asymmetric key pair (spk_a, ssk_a) (In the notation, the first "s" stands

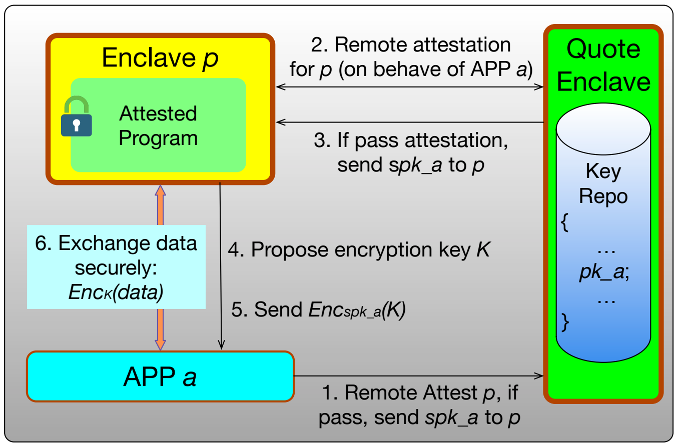
Enclave p
Attested Program
2. Remote attestation for p (on behave of APP a)
3. If pass attestation, send spk_a to p
Quote Enclave
Key Repo
{ … pk_a; … }
4. Propose encryption key K
5. Send $Enc_{spk\_a}(K)$
6. Exchange data securely: $Enc_K(data)$
APP a
1. Remote Attest p, if pass, send spk_a to p

Fig. 3. Key Distribution Infrastructure

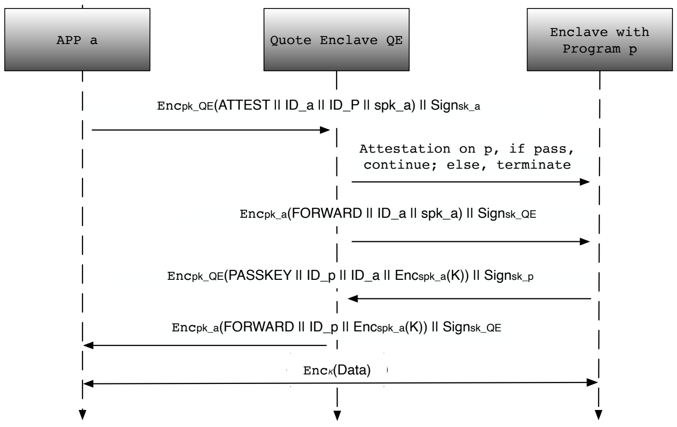
APP a
Quote Enclave QE
Enclave with Program p
Enc_{pk_QE}(ATTEST II ID_a II ID_P II spk_a) II $Sign_{sk\_a}$
Attestation on p, if pass, continue; else, terminate
Enc_{pk_a}(FORWARD II ID_a II spk_a) II $Sign_{sk\_QE}$
Enc_{pk_QE}(PASSKEY II ID_p II ID_a II Enc_{spk_a}(K)) II $Sign_{sk\_p}$
Enc_{pk_a}(FORWARD II ID_p II Enc_{spk_a}(K)) II $Sign_{sk\_QE}$
Enc_K(Data)

Fig. 4. Key Distribution Protocol

for "Session"). Then, it issues a request to the Quote Enclave (as shown in step 1 in Fig. 3), indicating: "perform a remote attestation on p on behalf of APP a; if pass, ask p to contact APP a using public key spk_a". Note that this request is encrypted with pk_QE and signed by the sk_a to preserve the request's confidentiality and integrity. The Quote Enclave, once receiving the request, performs remote attestation on p (as shown in step 2). If the attestation is passed, it forwards spk_a to the receiver enclave (step 3). The enclave p then proposes a symmetric encryption key K (step 4) and sends K to APP a through QE using the encryption key spk_a (step 5). Upon the APP decrypting and restoring K, the APP a and p start to exchange data using encryption key K (step 6). Fig. 4 shows the detailed protocol based on the steps described above. Please note that Fig. 3 is a high-level description of the protocol in Fig. 4. As a result, it skipped the participation of QE displayed in Fig. 4. In the protocol, we introduce three instructions in the messages, including ATTEST, FORWARD and PASSKEY. To protect message confidentiality, integrity, and authenticity, each message is encrypted with the receiver's public key and signed with the sender's secret key.

V. Security Analysis

In this section, we show that Split Inference has raised the bar for the attacker to compromise the input and output confidentiality for the MobileNet V1 model. The model input and output confidentiality also applies to other deep learning models for the same reason.

Firstly, the data transmitted between the APP and the enclave cannot be leaked due to the encryption using symmetric key K generated through key distribution protocol introduced in Section IV-B.

Secondly, the confidentiality of model input and output is preserved in the input and output enclave. Our design makes the input and output enclave contain simple and standardized operations. The input enclave contains a convolution layer and an activation operator. The output enclave contains a fully connected layer and a `softmax` operator. By carefully evaluating the standardized implementations, data leakage logic can be eliminated. The remote attestation on the two enclaves ensures that those layers will not be tamperred when deployed and executed in the enclave. Therefore, data leakage in the input and output enclaves is prevented.

Finally, it is difficulty for the attacker to restore input and output based on the data in the intermediate layers. Mathematically, the operators in the input enclave can be represented as $O = \phi(I \circ k + b)$, where O and I are the layer output and input, respectively, k is the convolution kernel, b is the bias of the model, and ϕ is the activation function. The attacker's goal is to restore I based on the observed O. However, without knowing ϕ, k, and b, it is infeasible to accurately restore I. Similarly, the operator in the output enclave can be represented as $O = softmax(\phi(w * I + b))$, where w and b are the weights and bias of the fully connected layer, I is the input of the Output Enclave. ϕ is the activation function of the fully connected layer and the $softmax$ function returns the probabilistic scores for each label, i.e., the Inference Output. The attacker's goal is to recover O based on I. which is mathematical challenging without knowing w and b.

In practice, the attacker might try to restore model input and output through data reconstruction attacks [7] [30] [8] or label inference attacks [16]. On the other hand, existing countermeasures [21] [23] [29] [33] against these attacks can be applied to the Split Inference framework to defeat those attacks.

The design of Keystone thwarts several major TEE-based side-channel attacks, including cache side-channel, control side-channel, etc [14]. Although the timing side-channel is not addressed by Keystone, in which the attacker infers the input and output group execution time on the eapp and hopefully infers some data information in the enclaves, we can eliminate this side-channel by uniforming the inference time of the input and output group.

VI. Implementation

We implemented a prototype system on Keystone, an open-source RISC-V based multi-enclave TEE framework. The entire MobileNet V1 model was implemented in C++. In our prototype, we created two enclaves, running as the Input Enclave, Output Enclave. We implemented the secure communication protocol by using basic remote attestation service supported by the Keystone. Due to the limited library support in RISC-V, we try to make the implementation self-contained, meaning not relying on external library. To follow that principle, we integrate lightweight encryption library, such as tiny-AES-c for encryption and hardcode model parameters instead of having the enclave reading the model from a stored file. Doing so can also reduce the performance overhead by avoiding file I/O. It also improved the performance by avoiding interacting with external libraries or files outside the enclave. A drawback is that such an approach makes the code included in the enclave larger. Fortunately, the generated executable can fit in the enclaves. For the MobileNet V1 implementation, with the resolution multiplier ρ as 224x224 and the width multiplier α as 0.25, the size of generated keystone .ke file is 13.1 MB; with α as 0.5, the size of the generated .ke file is 20.5 MB; with α as 0.7, the size of the generated .ke file is 29.5 MB. For information encryption, we use AES-128CBC for symmetric encryption (secure data transmission) and RSA for asymmetric encryption (key distribution and secure communication channel set up).

VII. Experiments

We measured the performance overhead introduced by the Split Inference through experiments. The experiments were run on a Google Cloud Platform e2-standard-4 instance, which has 4 vCPUs and 16 GB of memory. The VM is running Ubuntu-22.04.5_LTS OS, on which we launched the prototype on a QEMU emulator that emulates a RISC-V hardware environment. We set the emulated RISC-V machine's memory to 2 GB and the vCPU number to 1 to simulate the resource-restricted setting. In the prototype, the memory size for the Input Enclave and the Output Enclave are set to 64MB and 4MB, respectively.

We run the performance experiments to measure the execution time for different width multiplier α, ranging from 0.7 to 0.25. We set the resolution multiplier ρ to 224×224. We compared the performance of Split Inference on the Keystone with a baseline model, the original MobileNet V1 model deployed to the Keystone but without using any enclaves. Table I shows the time it takes to complete one inference request under different α. It includes the time of enclave and model layer initialization, image reading from APP, key distribution to the enclaves and communication channel set up. The result showed that the execution overhead for completing the entire procedure ranges from 101% to 516%, when width multiplier changes from 0.7 to 0.25. Notice that the initializations of enclaves and model layers, as well as the execution of key distribution and secure communication protocol happened only once, even if the same model is repeatedly used for inference tasks. That means the overhead of model inference only should be lower.

To understand the overhead details, we further investigated the composition of the time on the Split Inference and on

TABLE I
Execution Time for one Inference on MobileNet V1

Width Multiplier α	Split Inference	Baseline	Execution Overhead
0.7	71.1034	35.2980	1.0144
0.5	50.6748	18.5858	1.7265
0.25	35.0644	5.6916	5.1607

baseline and present the time breakdown in table II, III, and IV. For each setting, we broke down the tasks for Input Enclave, middle group (executed on the host), and Output Enclave. Each enclave has tasks including Enclave Init, Remote Attestation (key distribution & secure communication channel set up), Model Layer Init, Layer Inference, Data Transmission (send the group output out of the enclave. For the Output Enclave, it also includes layer output encryption), and Enclave Destroy. For the host, it includes tasks of Host Init (read in the image), Input Transmission (including encrypting the image and sending it to the Input Enclave), Model Layer Init, Layer Inference (for middle group), and Data Transmission (sending middle group output to the Output Enclave). The Layer Inference time includes the time takes to receive output from the previous group/APP, decrypt the data if needed, and perform the inference. For baseline, we broke down the time to Model Init and Model Inference, as shown in the last two rows of each table. The second right-most column recorded the total execution time in different settings. The same inference task is executed five times and the average time for each task is collected and presented in the table.

In a practical scenario, MIS will provide inference service for multiple requests. In this setting, Input Transmission, Layer Inference, and Data Transmission will take place for each inference request in the Split Inference, while other tasks will be executed only once when the APP and MIS is initialized. In comparison, for Baseline, only model inference will be executed every time when an inference request occurred. We highlighted the tasks executed in each inference request with yellow box in Table II, III, and IV. For each inference, each task depends on the result of its predecessor task. Therefore, all the highlighted tasks are executed sequentially, forming the actual execution time for each inference. Based on the above reason, we define the *inference overhead* as

$$(T_{InputTransmission} + \sum T_{LayerInference} + \sum T_{DataTransmission})/T_{ModelInference} - 1$$

Notice the $T_{ModelInference}$ is the Model Inference time for Baseline; $\sum T_{LayerInference}$ and $\sum T_{DataTransmission}$ are the summation of the corresponding time on three groups. According to the data in Table II, III, and IV, the inference overheads of Split Inference for 0.7 MobileNet-224, 0.5 MobileNet-224, and 0.25 MobileNet-224 are 71%, 133%, and 430%, respectively.

By analyzing the times, we observed that the model inference time in baseline is almost the same as the layer inference time on the intermediate layer. The layer inference times on the input and output enclaves are trivial. We also noticed

TABLE II
EXECUTION TIME BREAKDOWN ON AN IMAGE INFERENCE USING 0.7 MOBILENET-224

Environment	Task	Input Enclave	Host (Intermediate Layer)	Output Enclave	Total	Inference Overhead
Split Inference	Host Init	-	10.8178	-	71.1034	71.2805%
	Input Transmission	-	21.5018	-		
	Enclave Init	9.6748	-	20.3394		
	Remote Attestation	0.0948	-	0.0062		
	Model Layer Init	0.8560	18.1108	0.0602		
	Layer Inference	0.0038	37.5590	0.0092		
	Data Transmission	0.2660	0.0064	0.0174		
	Enclave Destroy	0.0508	-	0.0026		
Baseline	Model Init	0.5877			35.2980	
	Model Inference	34.6587				

TABLE III
EXECUTION TIME BREAKDOWN ON AN IMAGE INFERENCE USING 0.5 MOBILENET-224

Environment	Task	Input Enclave	Host (Intermediate Layer)	Output Enclave	Total	Inference Overhead
Split Inference	Host Init	-	7.6098	-	50.6748	133.5292%
	Input Transmission	-	23.2750	-		
	Enclave Init	6.4866	-	11.0028		
	Remote Attestation	0.0908	-	0.0070		
	Model Layer Init	0.8378	9.2450	0.0404		
	Layer Inference	0.0050	18.7478	0.0082		
	Data Transmission	0.1926	0.0066	0.0322		
	Enclave Destroy	0.0042	-	0.0030		
Baseline	Model Init	0.5354			18.5858	
	Model Inference	18.0994				

TABLE IV
EXECUTION TIME BREAKDOWN ON AN IMAGE INFERENCE USING 0.25 MOBILENET-224

Environment	Task	Input Enclave	Host (Intermediate Layer)	Output Enclave	Total	Inference Overhead
Split Inference	Host Init	-	6.5782	-	35.0644	430.1635%
	Input Transmission	-	22.3354	-		
	Enclave Init	6.0074	-	9.1850		
	Remote Attestation	0.0468	-	0.0070		
	Model Layer Init	0.4232	5.9534	0.0258		
	Layer Inference	0.0156	5.1402	0.0072		
	Data Transmission	0.1040	0.0074	0.0430		
	Enclave Destroy	0.0026	-	0.0022		
Baseline	Model Init	0.4536			5.6916	
	Model Inference	5.2159				

that in each setting, Input Transmission takes a noticeable amount of time, while the Data Transmission times are trivial when compared with the Input Transmission time. Based on the above observation, we have the following conclusion: *To process one inference request, the model inference time on Split Inference is almost the same as that time for Baseline, and the major extra overhead occurred in Input Transmission.*

Based on the above conclusion, we can explain the reason why a smaller model has a higher overhead. It is because the Input Transmission time is decided by the model input size, which is unchanged. On the other hand, the model inference time reduces with a smaller width multiplier. As a result, the inference overhead increases when the width multiplier decreases.

VIII. DISCUSSION

A. *Supporting Hardware*

The Split Inference framework can be implemented on most hardware platforms that support multi-enclave TEE, including Intel Skylake or newer architecture supporting SGX [5] [4], ARMv9 architecture supporting CCA [15] [1], and RISC-V architecture supporting Keystone, such as SiFive HiFive Unleashed [2] and Unmatched [3], etc.

B. *Supporting Models*

Split Inference can be applied to other deep learning models that have multiple layers. Table V listed the first and last group of layers of several on-device deep learning models. The general trend is that the first group uses standard convolutions with strides and pooling to rapidly reduce input dimensions while learning basic features, while the last group focus on

TABLE V
THE FIRST AND LAST GROUP OF LAYERS FOR DIFFERENT ON-DEVICE DEEP LEARNING MODELS.

Model	**First Group**	**Last Group**
MobileNet V2 [25] & V3 [10]	Standard 3×3 convolution with stride 2, followed with batch normalization and ReLU activation.	1×1 convolution that aggregates features, followed with global average pooling and a classification head (dense layer)
EfficientNet / Lite [27]	Small, efficient 3×3 convolution with stride 2 to capture low-level features	1×1 convolution to expand the channel dimension, followed with global average pooling, then a dense layer for final classification
SqueezeNet [12]	Larger convolution (commonly 7×7) to capture broad features, followed by a max pooling layer	1×1 convolution that compresses the feature maps to the number of classes, followed with global average pooling to yield class scores
ShuffleNet [31]	3×3 convolution with batch normalization and activation, followed with a max pooling operation for early downsampling	1×1 pointwise convolution to mix channel information, followed with global average pooling followed by a fully connected layer
Tiny YOLO [24] (Detection)	Efficient convolutional layer (commonly 3×3) to extract initial features with appropriate downsampling	Convolutional layer that outputs a grid of predictions (bounding box coordinates, objectness scores, and class probabilities)

compressing the spatial information through global pooling (or similar techniques) so that the final classification or detection head can operate on a compact feature representation. Since the first and last group are typically simple and standardized, Split Inference can be applied so that they will be deployed to sperate enclaves, which protects the model input and output confidentiality. The same principle can be applied to other more complex and advanced on-device models, such as on-device transformer MobileViT [18] and on-device LLMs(or known as SLM) [17] to protect data confidentiality of the APPs that use those models.

C. Application

The Split Inference can be applied on various edge computing settings, such as smart devices, automomous systems, robotics, etc., to prevent data leakage from malicious or vulnerable on-device deep learning models. For each model, we can deploy the first group of layers to the Input Enclave and the last group of layers to the Output Enclave, and keep the remaining layers outside of the enclave. The APP that uses the model only needs to establish a secure channel with the Input and Output Enclave, respectively, using the protocol in Section IV-B. After that, the APP can send inference requests and receive inference responses through the secure channels.

IX. RELATED WORKS

A. Multi-enclave TEE

The exploration of multi-enclave TEE is emerging as single-enclave TEE failed to satisfy the need of multi-module machine learning system [28]. Existing implementations, such as Keystone [14] and Multizone [22], do not offer complete security solutions. Multizone, a proprietary multi-enclave solution, can only support up to four isolated enclaves, and does not support remote attestation. Keystone does not have enclave number limitation and provides basic remote attestation. However, it does not directly support key management and secure communication issues.

B. Split Learning

Split learning [9] is a distributed machine learning framework that partitions a neural network into the *client* model, which runs on the device and the *server* model, which runs on the server. In this way, client and server collaborate to complete training and inference. The client locally computes the first few layers, and the server computes rest of the layers. Pasquini et al. [20] first studied security issues of split learning, introduced the feature-space hijacking attack, in which the attacker server can direct the client models towards its own malicious goal, independent of the actual classification task. Following this trend, various data reconstruction [8] and label inference attacks [16] were proposed targeting on split learning.

Different from split learning, our Split Inference model only focuses on model inference. The model is pretrained before deployed to the device for model inference services. Therefore, the model is immutable during the inference. Malicious server will not have a chance to hijack the training and mislead the model.

There exists some attacks do not rely on the training phase: Erdoğan et al. [7] presented UnSplit to inverse a model input based on the intermediate data. It leverages deep learning methods to restore simple input such as images from the MNist, F-MNist, and Cipher-10 dataset. However, UnSplit is ineffective in restoring complex model input. [30] proposed GAN-based data reconstruction attack method, showing an improved attack performance.

On the other hand, defense against the above attacks were presented. Erdoğan et al. [6] proposed SplitGuard, which detects and mitigates training-hijack attacks listed in [20]. Various different methods were proposed to defeat the data reconstruction and label inference attacks through differential privacy [21], HashVFL [23], Patch Shuffle [29], Potential Energy Loss [33], etc. These methods are all compatible with our Split Inference framework to enhance the defense.

C. Model Data Protection

To protect model data confidentiality, many works [19], [26], [32] protect minority but critical layers of the model using an TEE enclave, but protect majority remaining layers through linear transformations. These works focus on one enclave only and did not explore multi-enclave architecture. The state-of-the-art solution in this direction claimed that its inference overhead ranges between 142% (VGG16 BN) and 577% (AlexNet) at the best case (See Table 6 of [32]), which is higher than our solution (71% to 430% of inference overhead).

X. Conclusion and Future Work

In this paper, we introduced the Split Inference architecture that can protect data confidentiality in untrusted models on the on-device machine learning system. We achieved the goal through multi-enclave TEE and implemented a prototype for MobileNet model on Keystone, a RISC-V based multi-enclave TEE. We tested and measured the performance against the baseline that does not use enclaves. Experiments showed that Split Inference incurred 71% to 430% of inference overhead, which outperforms existing state-of-the-art work that uses TEE protect on-device model data. Our future work includes to expand the Split Inference framework to other on-device models and implement the framework on hardware devices.

XI. Acknowledgement

This work is partially funded by CAHSI-Google Institutional Research Program.

References

[1] Arm Confidential Compute Architecture — arm.com. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture. [Accessed 16-04-2025].

[2] HiFive Unleashed (Discontinued) - SiFive Boards — sifive.com. https://www.sifive.com/boards/hifive-unleashed. [Accessed 16-04-2025].

[3] HiFive Unmatched Rev B - SiFive Boards — sifive.com. https://www.sifive.com/boards/hifive-unmatched-revb. [Accessed 16-04-2025].

[4] Intel® Software Guard Extensions (Intel® SGX) — intel.com. https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html. [Accessed 16-04-2025].

[5] Costan, V., and Devadas, S. Intel sgx explained. *Cryptology ePrint Archive* (2016).

[6] Erdogan, E., Küpçü, A., and Cicek, A. E. Splitguard: Detecting and mitigating training-hijacking attacks in split learning. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society* (2022), pp. 125–137.

[7] Erdoğan, E., Küpçü, A., and Çiçek, A. E. Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society* (2022), pp. 115–124.

[8] Gao, X., and Zhang, L. PCAT: Functionality and data stealing from split learning by Pseudo-Client attack. In *32nd USENIX Security Symposium (USENIX Security 23)* (Anaheim, CA, Aug. 2023), USENIX Association, pp. 5271–5288.

[9] Gupta, O., and Raskar, R. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications 116* (2018), 1–8.

[10] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3, 2019.

[11] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR abs/1704.04861* (2017).

[12] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[13] Jauernig, P., Sadeghi, A.-R., and Stapf, E. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy 18*, 2 (2020), 56–60.

[14] Lee, D., Kohlbrenner, D., Shinde, S., Asanović, K., and Song, D. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems* (New York, NY, USA, 2020), EuroSys '20, Association for Computing Machinery.

[15] Li, X., Li, X., Dall, C., Gu, R., Nieh, J., Sait, Y., Stockwell, G., Knight, M., and Garcia-Tobin, C. Enabling realms with the arm confidential compute architecture.

[16] Liu, J., Lyu, X., Cui, Q., and Tao, X. Similarity-based label inference attack against training and inference of split learning. *IEEE Transactions on Information Forensics and Security 19* (2024), 2881–2895.

[17] Lu, Z., Li, X., Cai, D., Yi, R., Liu, F., Zhang, X., Lane, N. D., and Xu, M. Small language models: Survey, measurements, and insights, 2025.

[18] Mehta, S., and Rastegari, M. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations* (2022).

[19] Mo, F., Shamsabadi, A. S., Katevas, K., Demetriou, S., Leontiadis, I., Cavallaro, A., and Haddadi, H. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2020), MobiSys '20, Association for Computing Machinery, p. 161–174.

[20] Pasquini, D., Ateniese, G., and Bernaschi, M. Unleashing the tiger: Inference attacks on split learning. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2021), CCS '21, Association for Computing Machinery, p. 2113–2129.

[21] Pham, N. D., Phan, K. T., and Chilamkurti, N. Enhancing accuracy-privacy trade-off in differentially private split learning. *IEEE Transactions on Emerging Topics in Computational Intelligence 9*, 1 (2025), 988–1000.

[22] Pinto, S., and Martins, J. The industry-first secure iot stack for risc-v: a research project. In *RISC-V Workshop,(Zurich)* (2019).

[23] Qiu, P., Zhang, X., Ji, S., Fu, C., Yang, X., and Wang, T. Hashvfl: Defending against data reconstruction attacks in vertical federated learning. *IEEE Transactions on Information Forensics and Security 19* (2024), 3435–3450.

[24] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[25] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 4510–4520.

[26] Shen, T., Qi, J., Jiang, J., Wang, X., Wen, S., Chen, X., Zhao, S., Wang, S., Chen, L., Luo, X., Zhang, F., and Cui, H. SOTER: Guarding black-box inference for general neural networks at the edge. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)* (Carlsbad, CA, July 2022), USENIX Association, pp. 723–738.

[27] Tan, M., and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (2019), PMLR, pp. 6105–6114.

[28] Wang, Y., and Boggram, V. Towards protecting on-device machine learning with risc-v based multi-enclave tee. In *Proceedings of the 14th International Workshop on Security, Privacy, and Trust for IoT* (Big Island, HI, USA, 2024), IoTSPT '24, IEEE.

[29] Yao, D., Xiang, L., Xu, H., Ye, H., and Chen, Y. Privacy-preserving split learning via patch shuffling over transformers. In *2022 IEEE International Conference on Data Mining (ICDM)* (2022), pp. 638–647.

[30] Zeng, B., Luo, S., Yu, F., Yang, G., Zhao, K., and Wang, L. Gan-based data reconstruction attacks in split learning. *Neural Networks 185* (2025), 107150.

[31] Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018), pp. 6848–6856.

[32] Zhang, Z., Gong, C., Cai, Y., Yuan, Y., Liu, B., Li, D., Guo, Y., and Chen, X. No Privacy Left Outside: On the (In-)Security of TEE-Shielded DNN Partition for On-Device ML . In *2024 IEEE Symposium on Security and Privacy (SP)* (Los Alamitos, CA, USA, May 2024), IEEE Computer Society, pp. 3327–3345.

[33] Zheng, F., Chen, C., Lyu, L., Fu, X., Fu, X., Wang, W., Zheng, X., and Yin, J. Protecting split learning by potential energy loss. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24* (8 2024), K. Larson, Ed., International Joint Conferences on Artificial Intelligence Organization, pp. 5590–5598. Main Track.