ACM CCS 2025

TensorShield

Safeguarding On-Device Inference by Shielding Critical DNN Tensors with TEE

Tong Sun[1], Bowen Jiang[1], Hailong Lin[1], Borui Li[2], Yixiao Teng[1], Yi Gao[1], and Wei Dong[1]

[1]The State Key Laboratory of Blockchain and Data Security,
College of Computer Science & School of Software Technology, Zhejiang University, China
[2]School of Computer Science and Engineering, Southeast University, China

浙江大學 区块链与数据安全全国重点实验室
STATE KEY LABORATORY OF BLOCKCHAIN AND DATA SECURITY
ZHEJIANG UNIVERSITY

東南大學
SOUTHEAST UNIVERSITY

Background: Why On-Device AI

- **On-device inference is increasingly popular for privacy- and latency-sensitive mobile/IoT tasks (e.g., face recognition, video analytics)**
  - **Advantages:**

✓ Data stays local (protects user privacy)

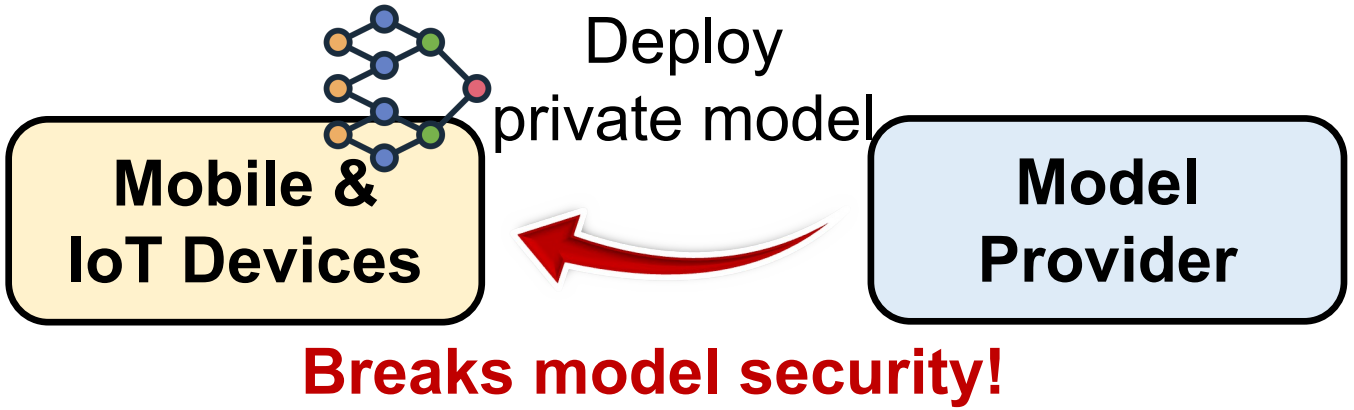
✓ Low latency, offline capability

Breaks data privacy!

Remote inference

Background: Two Typical Threats

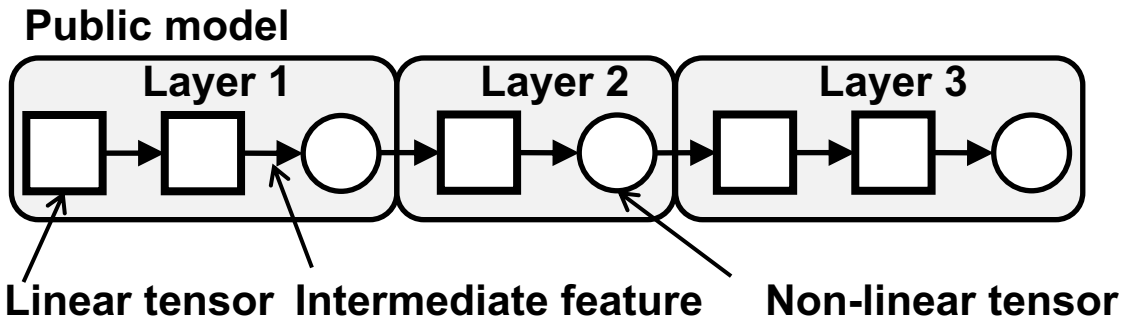
- **Deploying third-party private models on user devices exposes the model itself to attacks:**
  - **Model Stealing (MS)** – attacker extracts model parameters & trains a surrogate
  - **Membership Inference Attack (MIA)** – attacker infers whether a sample was in the training data.
- **These attacks threaten both model IP and training data privacy**

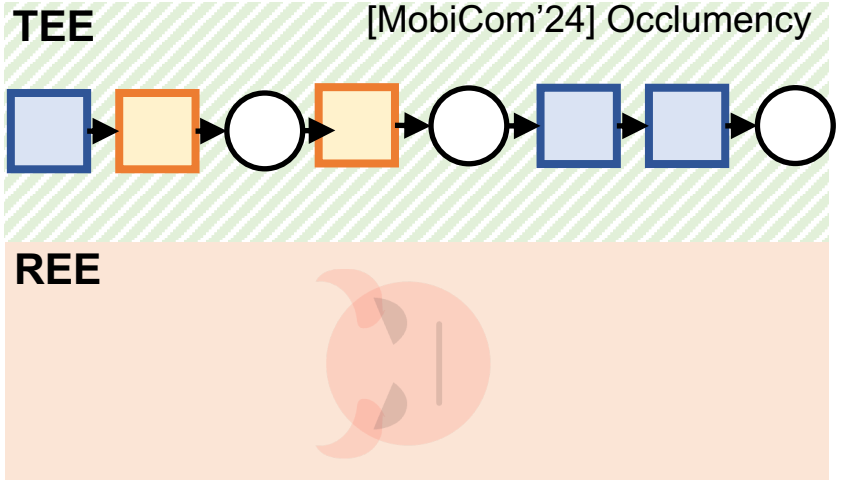


Directly on-device inference

Existing Defense & Limitations

- **Trusted Execution Environment (TEE)**
  - secure isolated hardware environment (e.g., ARM TrustZone).
- **Shield entire model in TEE**
  - **limited secure memory** (e.g., RPI3B: 16MB secure memory) ⇒ using on-demand loading of weights and partitioned convolution calculation ⇒ **huge latency overhead**



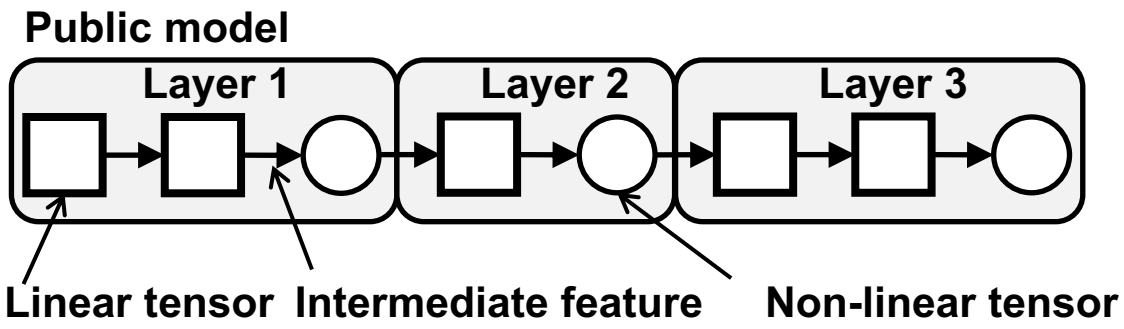


Shield all layers
Privacy: ✓ Latency: ✗

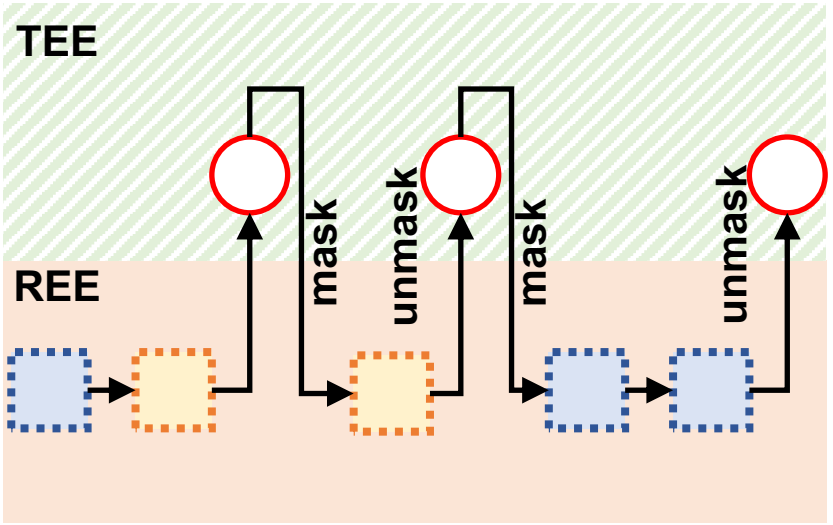
Existing Defense & Limitations (con.)

- **Shield non-linear layers in TEE**
  - linear layers can be protected in REE with **obfusacting** by matrix transformation

  ⇒ **high execution overhead because of deobfusaction and masking**



[S&P'23] ShadowNet
[ICML'24] GroupCover

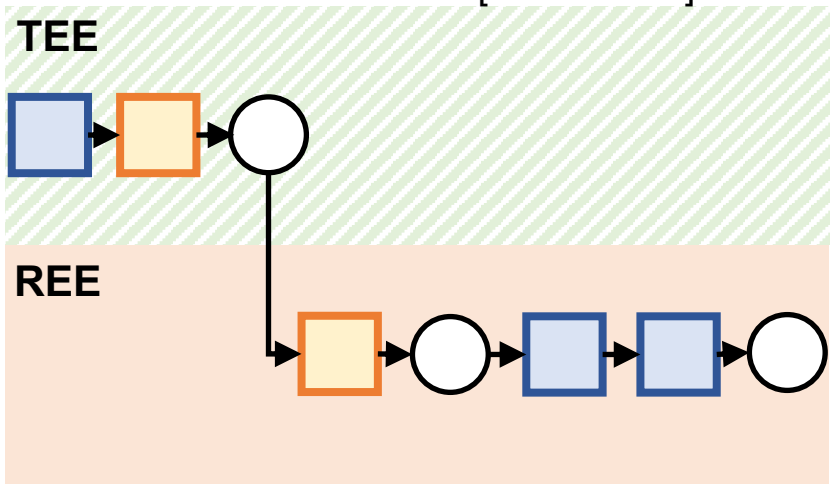


Shield non-linear tensors
Privacy: ✗ Latency: ✓

Existing Defense & Limitations (con.)

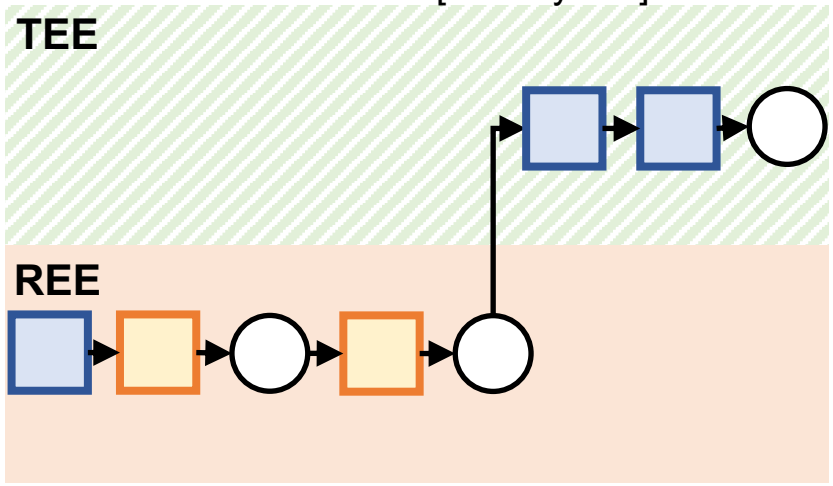
- **Shield partial layers in TEE**
  - Others only protect *partial layers* (e.g., shallow, deep) ⇒ **insufficient protection against MS & MIA**

[CCGRID'20] Serdab



Shield shallow layers
Privacy: ✗ Latency: ✓

[MobiSys'20] DarkneTZ



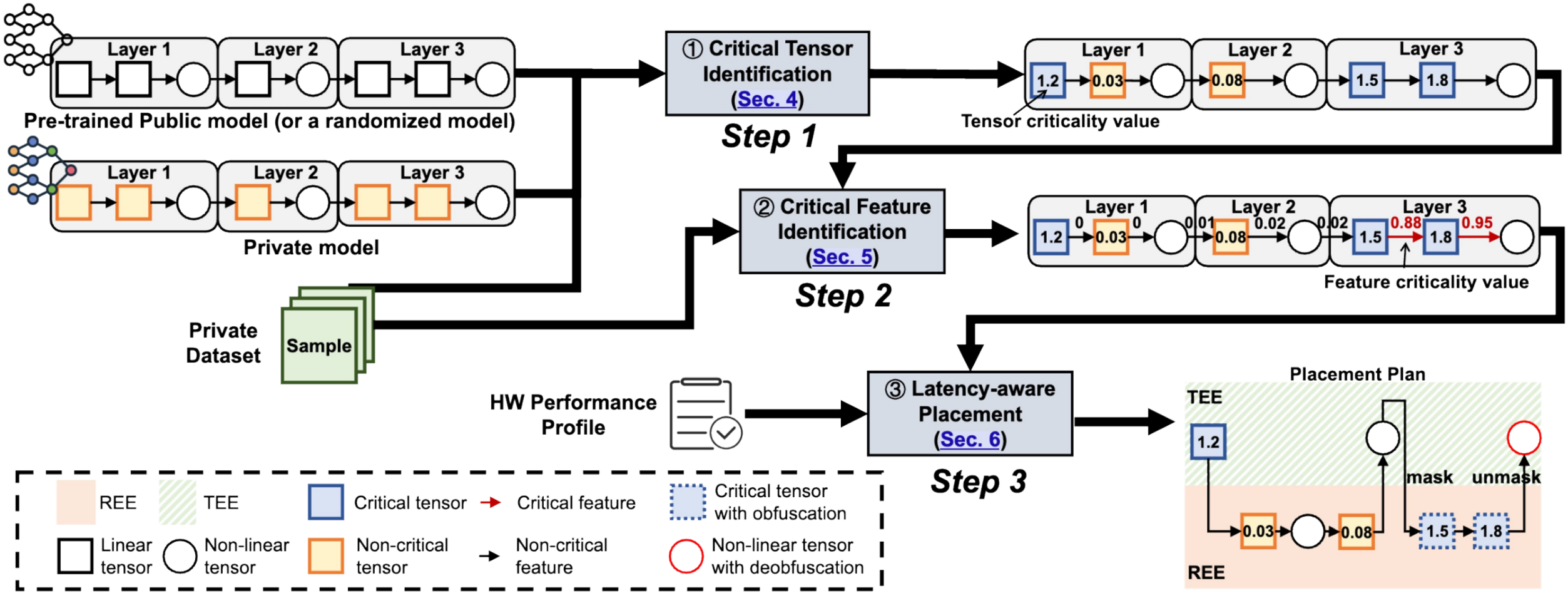
Shield deep layers
Privacy: ✗ Latency: ✓

Key idea

- **Insight**
  - The essential of MS is to steal the decision capabilities of the model.
  - The decision capabilities of a model do not necessarily depend on the shallow layers or deep layers.
- **Idea**
  - **Identify the most critical part of a model at the granularity of tensors, which offers a finer granularity than layers.**

By shielding these critical tensors (in TEE or in REE via obfuscation), we can achieve a high level of security without a large execution overhead.

TensorShield



Challenge

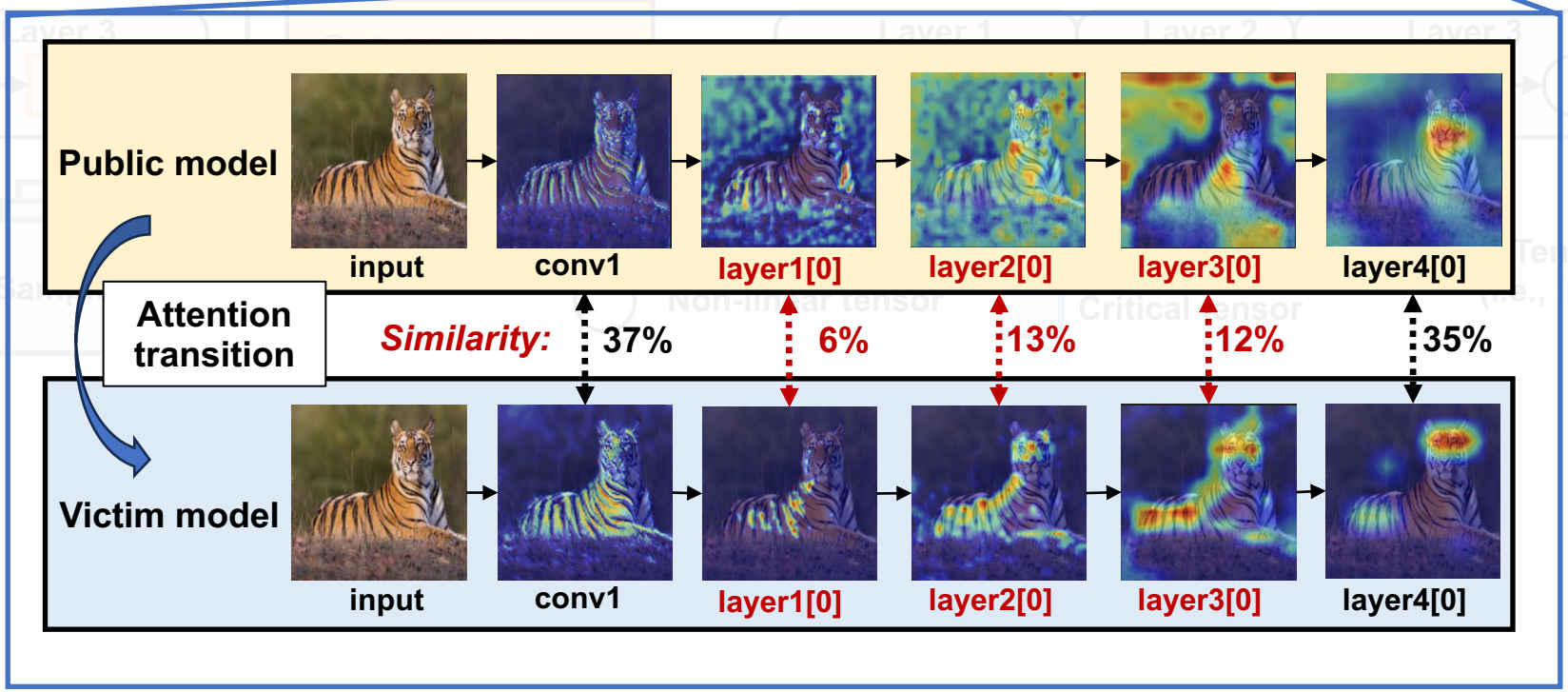
How to identify the critical tensors?

Critical Tensor Identification

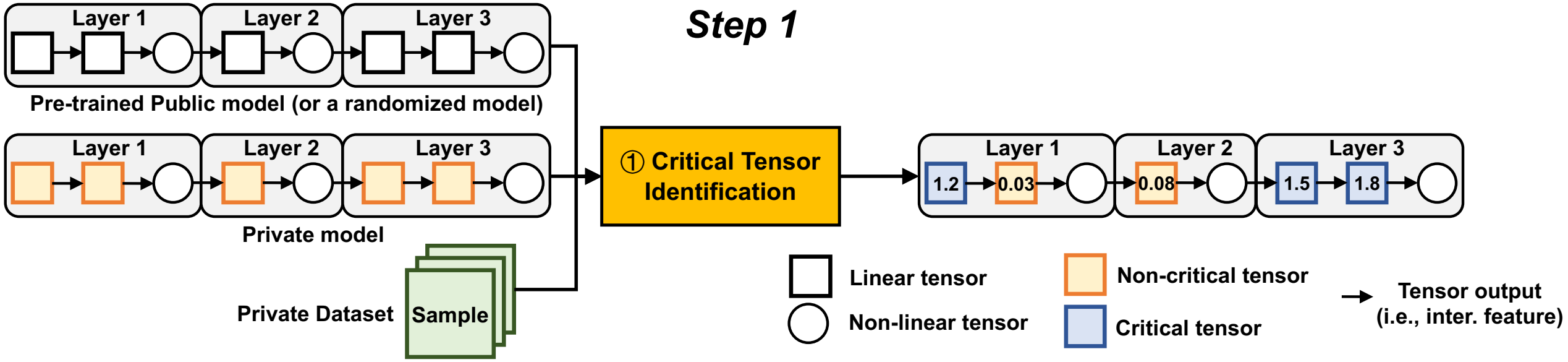
Tensor criticality value

$$I_k = \underbrace{\sum_i \frac{\partial L}{n \partial w_i^k} \Delta w_i^k}_{\text{Intrinsic tensor importance}} \times \underbrace{\left(1 - \cos(f(M_{Vic}^k), f(M_{Pub}^k))\right)}_{\text{Attention transition}},$$

eXplainable AI (XAI)



Critical Tensor Identification



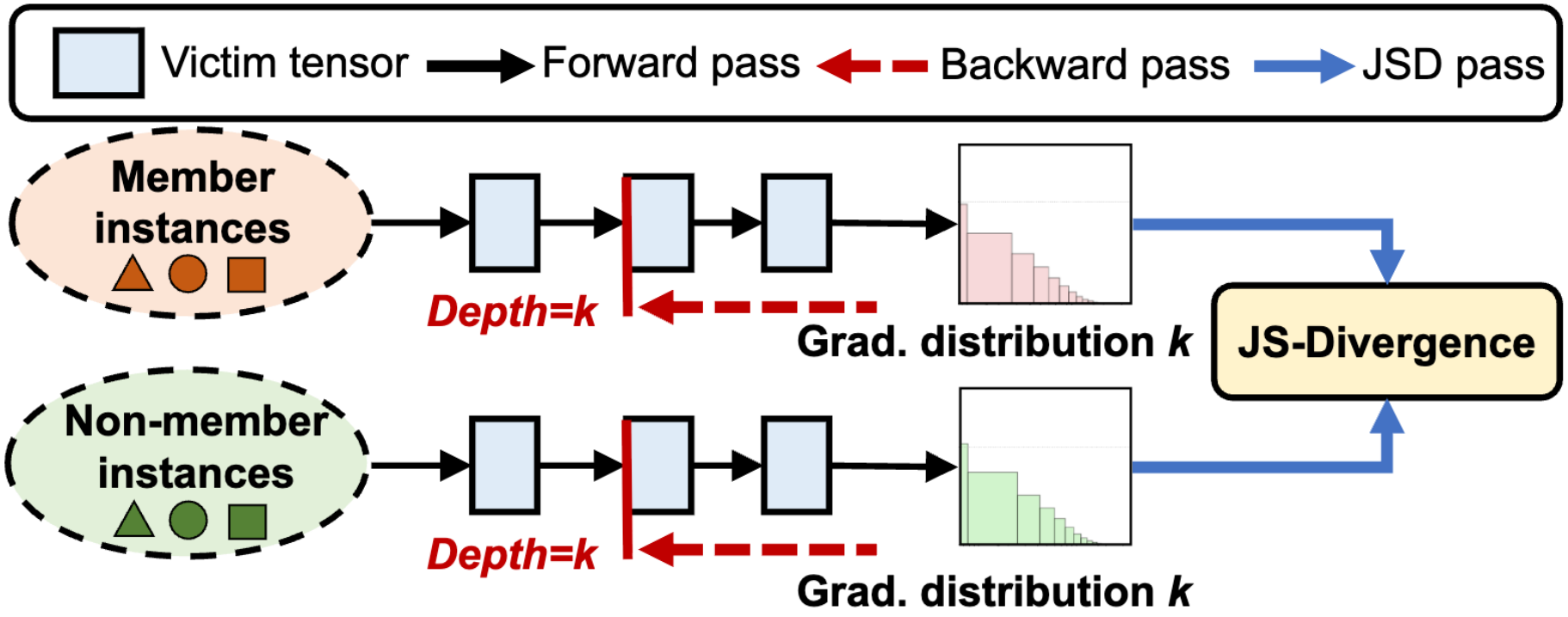
Critical Feature Identification

- **The key to preventing privacy leakage of intermediate features is to perform masking so that these features cannot be observed when they are transmitted from TEE to REE**

- **Challenge: masking all intermediate features imposes significant execution overheads**
  - e.g., [S&P'23] ShadowNet incurs **150%** and **160%** execution overheads for MobileNet and ResNet, respectively

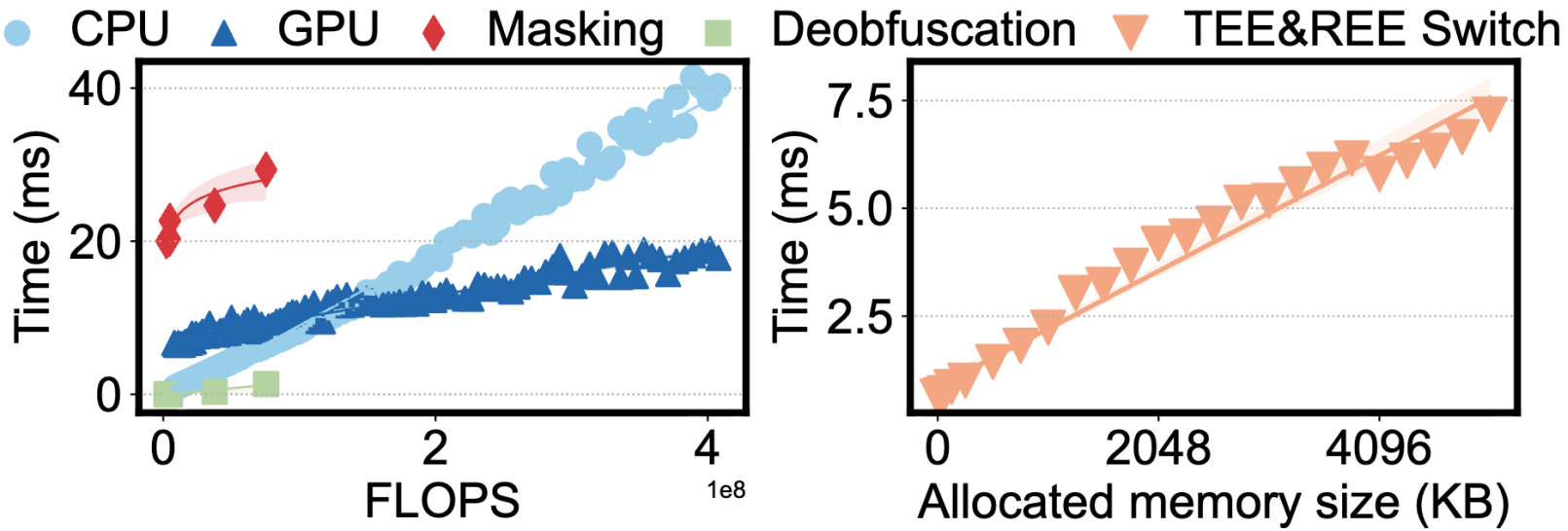
Critical Feature Identification

- Measure membership infromation sensitivity



Latency-aware Placement

- **Observation**
  - Different FLOPS computations are better suited for different accelerators.
  - As the communication memory between the REE and TEE increases, the overhead associated with switching increases. This drives our effort to comprehensively establish a latency-aware placement method to minimize the inference latency while maintaining model security.



Latency-aware Placement

- **Observation**
  - Different FLOPS computations are better suited for different accelerators.
  - As the communication memory between the REE and TEE increases, the overhead associated with switching
  - increases.

Motivates us to comprehensively establish a latency-aware placement method to minimize the inference latency while maintaining model security.

Latency-aware Placement

- To address the placement issue, we **establish a secure inference latency model** and formulate it as a numerical optimization problem

$$x_{i,j} = \begin{cases} 1 & \text{tensor } i \text{ is executed with the } j \text{ option} \\ 0 & \text{tensor } i \text{ is not executed with the } j \text{ option} \end{cases}$$

$$t_{i,j} = \begin{cases} t_i^{(REE,CPU)} & j = 0 \\ t_i^{(REE,GPU)} & j = 1 \\ t_i^{(REE,CPU)} + t_i^{deobf} + t_i^{mask} & j = 2 \\ t_i^{(REE,GPU)} + t_i^{deobf} + t_i^{mask} & j = 3 \\ t_i^{(TEE,CPU)} & j = 4 \end{cases}$$

$$\arg\min_x \underbrace{\sum_{i\in N, j\in 0,1,2} x_{i,j} \cdot t_{i,j}}_{\text{computation time}} + \underbrace{\left(\sum_{i\in N-1} x_{(i-1),0} \oplus \left(\sum_{j=1}^{4} x_{i,j} \right) \right) \cdot t^{switch}}_{\text{switch time}}$$

$$s.t. \quad \text{(Correctness)} \quad \sum_{j\in 0,1,2} x_{i,j} = 1, \forall_{i\in N}$$

$$\text{(Memory)} \quad x_{i,0} \cdot m_i \le M, \forall_{i\in N}$$

Implementation

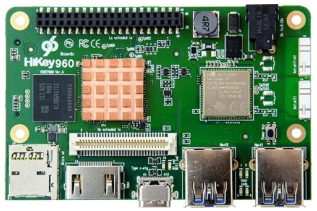
- We have implemented TensorShield in C/C++. Our code is open-sourced.

https://github.com/suntong30/TensorShield

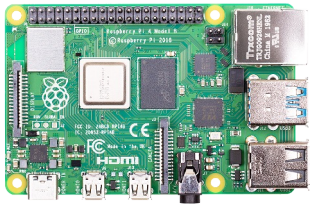
Evaluation: Setup

- **Two real-world devices**

Hikey960 (Mobile)



RPI3B+ (IoT)



Device Name	CPUs	GPU	RAM	TEE RAM	REE OS	TEE OS
Hikey960 [2]	4xCortex-A73 (@2.36 GHz) 4xCortex-A53 (@1.84 GHz)	ARM Mali G71 MP8	4 GB	64 MB	Android 7	OP-TEE v3.4.0
Raspberry Pi 3B+ (RPI3B+) [3]	4xCortex-A53 (@1.40 GHz)	VideoCore 4	1 GB	32 MB	Raspbian OS	OP-TEE v3.4.0

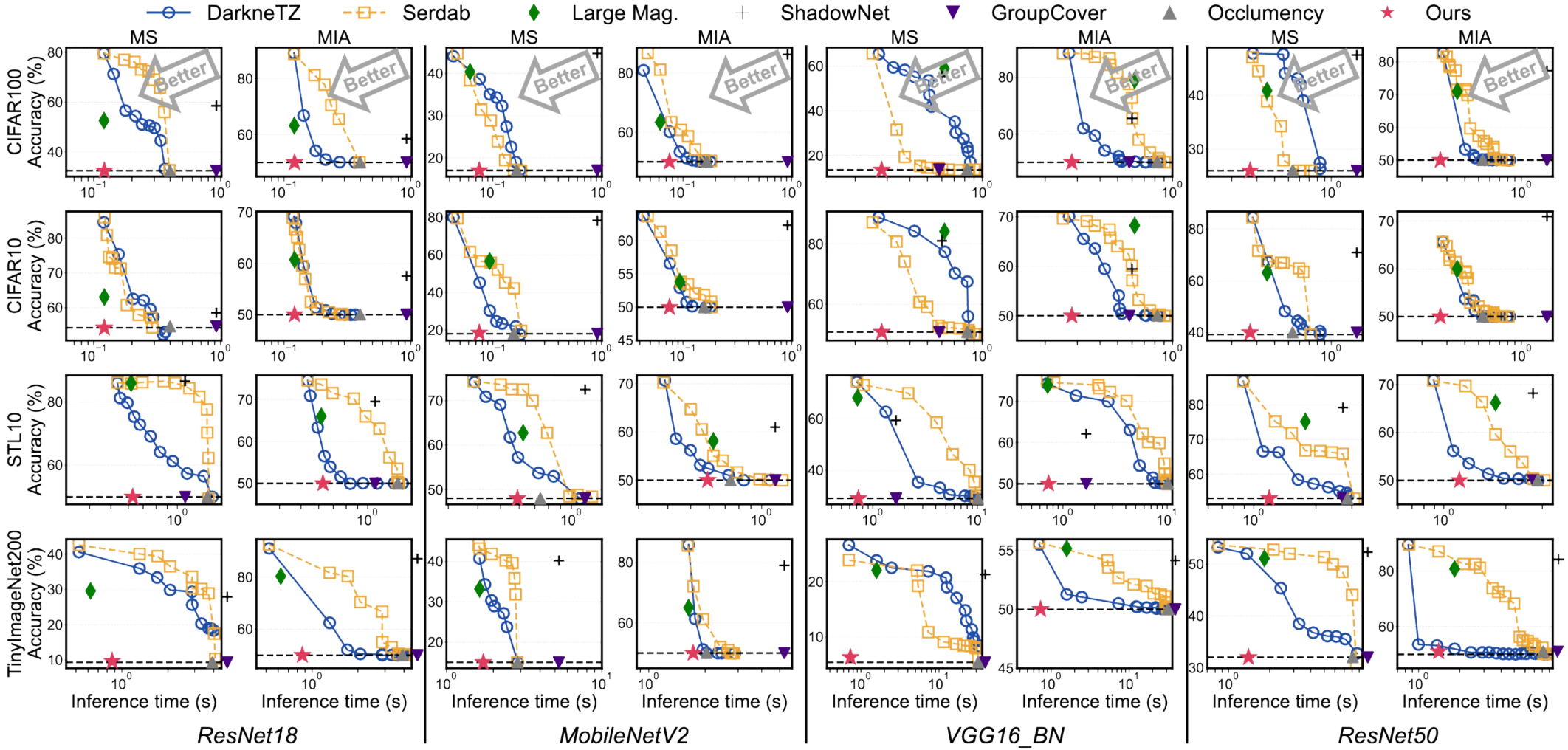
- **DNN models and Datasets**
  - MobileNetV2, ResNet-18, ResNet-50, and VGG16
  - CIFAR-10, CIFAR-100, STL-10, Tiny-ImageNet

Evaluation: Seven Baselines

- **One no-shield**
  - **Native** executes the entire victim model in REE
- **Four partial-shield**
  - **DarkneTZ**[1] shields victim model's *deep* layers
  - **Serdab**[2] shields victim model's *shallow* layers
  - **Magnitude**[3] shields victim model's *large magnitude* weights
  - **ShadowNet**[4] shields non-linear layers and *obfuscates linear layers in a fixed strategy*
- **Two all-shield**
  - **GroupCover**[5] shields non-linear layers and *obfuscates linear layers in a random strategy*
  - **Occlumency**[6] shields *the entire victim model* in TEE

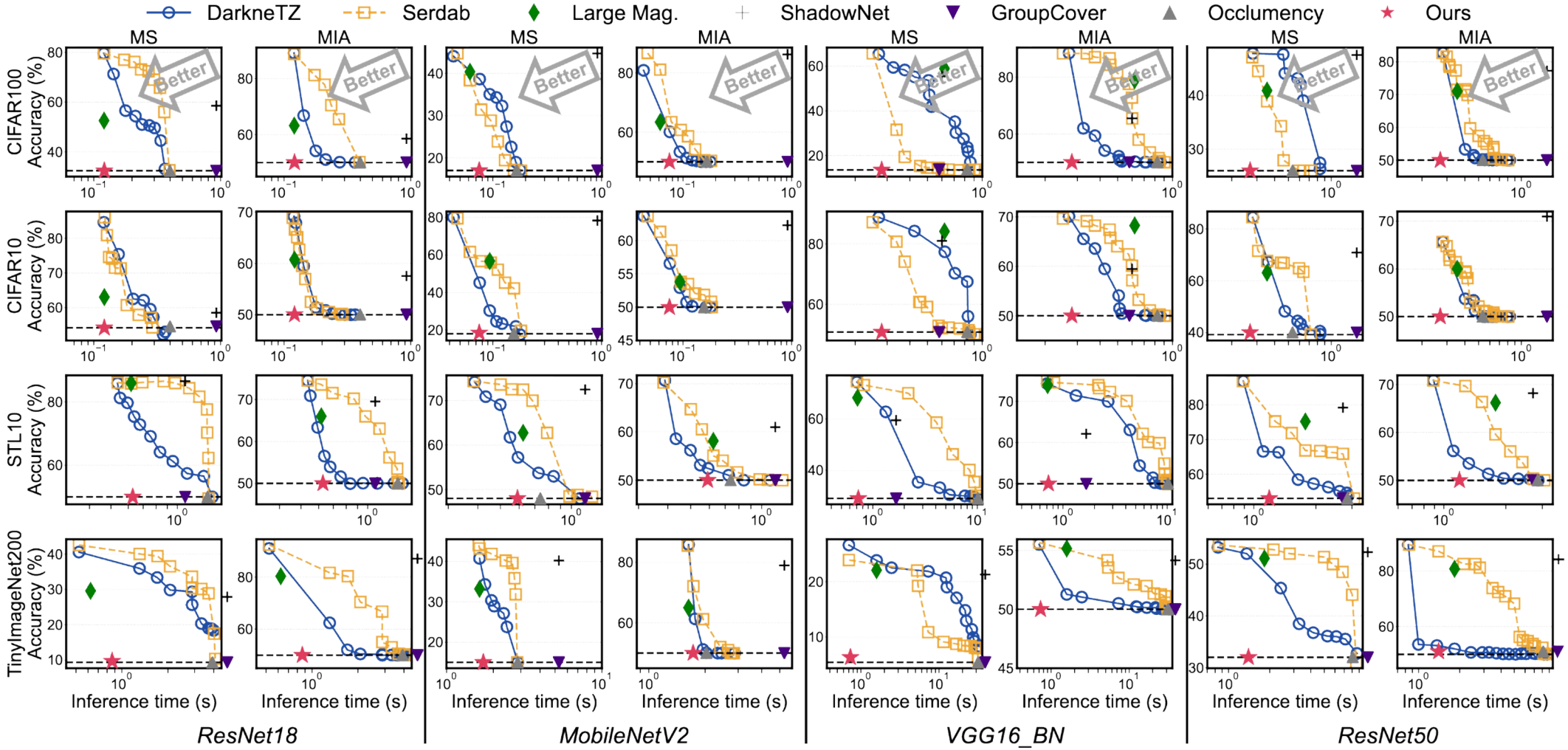
[1] DarkneTZ: Towards Model Privacy at the Edge using Trusted Execution Environments. In Proc. of ACM MobiSys 2020.
[2] Serdab: An IoT Framework for Partitioning Neural Networks Computation across Multiple Enclaves. In Proc. of IEEE/ACM CCGRID 2020.
[3] Model Protection: Real-Time Privacy-Preserving Inference Service for Model Privacy at the Edge.. IEEE TDSC 2021.
[4] ShadowNet: A Secure and Efficient On-Device Model Inference System for Convolutional Neural Networks. In Proc. of IEEE S&P 2023.
[5] GroupCover: A Secure, Efficient and Scalable Inference Framework for On-device Model Protection based on TEEs. In Proc. of ICML 2024.
[6] Occlumency: Privacy-preserving Remote Deep-learning Inference Using SGX. In Proc. of ACM MobiCom 2019.

Evaluation: Overall Performance



TensorShield achieves an inference speedup of up to **25.35× (avg. 5.85×)** and up to **16.89× (avg. 4.32×)** compared to **GroupCover** and **Occlumency**, respectively

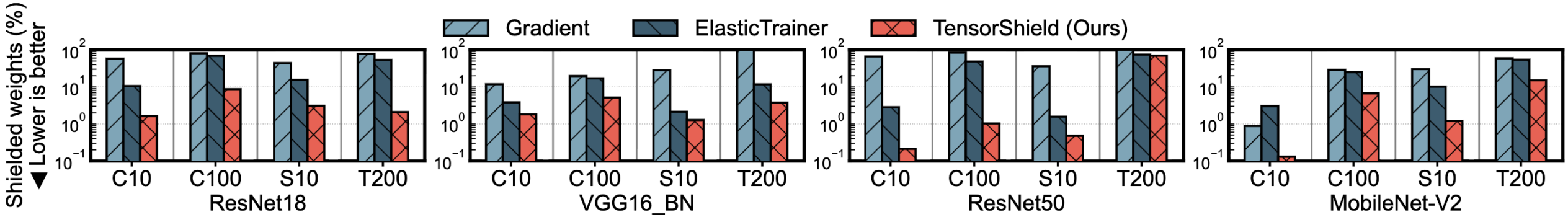
Evaluation: Overall Performance



Meanwhile, **TensorShield** achieves almost the **same security protection** as shielding the entire model inside TEE, i.e., **1.03× for MS** and **1.00× for MIA**

Evaluation: Efficacy of Critical Tensor Identification

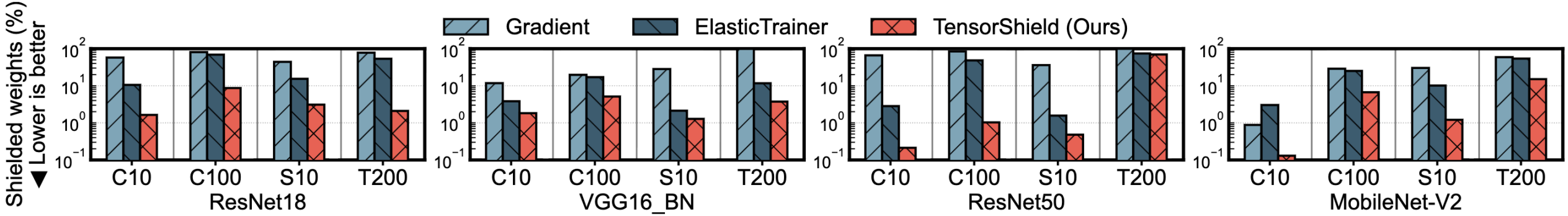
- **Two XAI baselines**
  - **Gradient** utilizes integrated gradients
  - **ElasticTrainer** utilizes gradients and weight updates (SOTA)



TensorShield selects average 87.82% (absolutely 50.60%) fewer parameters compared to the *Gradient method*, and relatively 70.32% (absolutely17.62%) fewer than *ElasticTrainer*.

Evaluation: Efficacy of Critical Tensor Identification

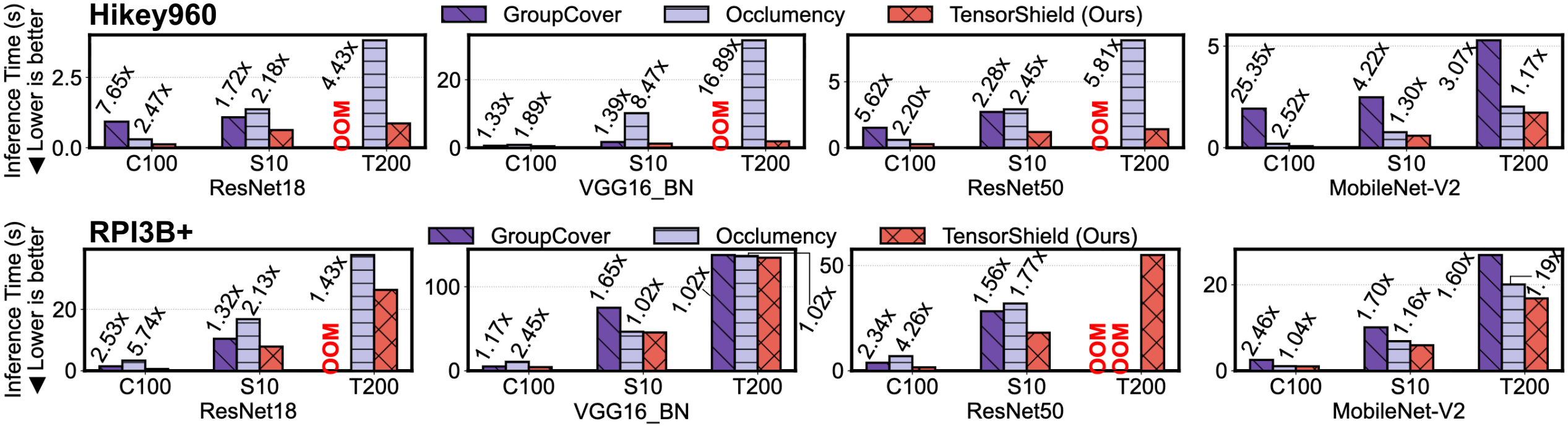
- **Two XAI baselines**
  - **Gradient** utilizes integrated gradients
  - **ElasticTrainer** utilizes gradients and weight updates (SOTA)



This improvement can be attributed to the public pre-trained models carrying more decision-making information from simpler datasets. TensorShield effectively reduces the importance of these tensors, thus excluding them from selection.

Evaluation: Runtime Performance

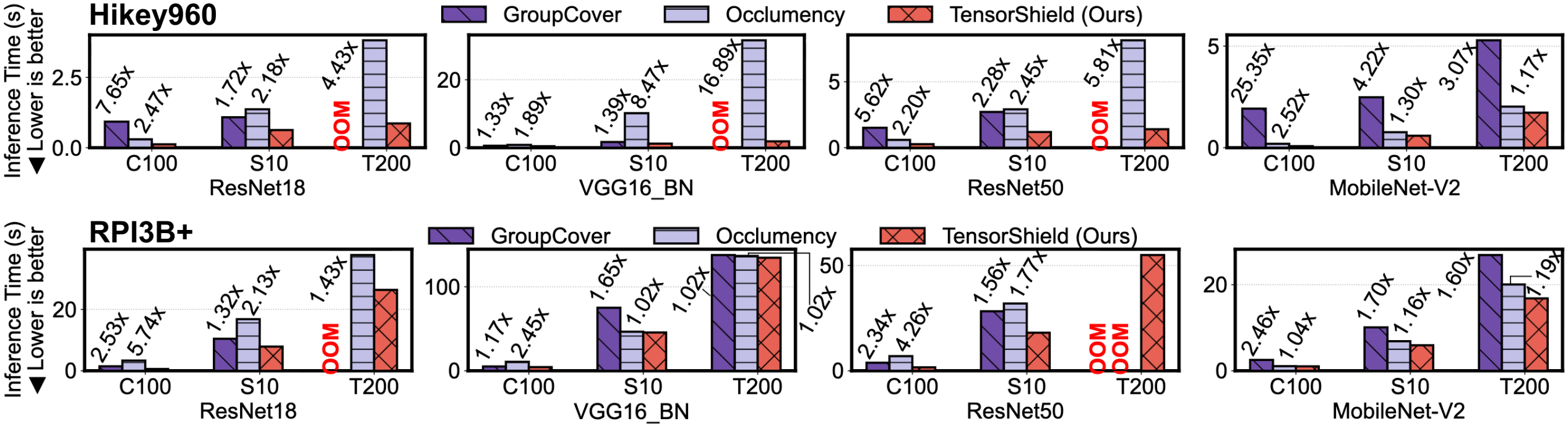
- **Inference Time**



On mobile devices (i.e., Hikey960), TensorShield's inference time is up to **25.35× (avg. 5.85×)** and **16.89×(avg. 4.32×)** faster than GroupCover and Occlumency, respectively.

Evaluation: Runtime Performance

- **Inference Time**



Even on IoT devices lacking GPU acceleration (i.e., Raspberry Pi 3B+), TensorShield achieves inference times that are up to **2.53× (avg. 1.74×)** and **5.74× (avg. 2.11×)** faster than GroupCover and Occlumency, respectively.

Evaluation: Runtime Performance

- Inference Time Breakdown

Dataset	Work	Execution time					
		Cal.	Dec.	Comm.	Mask.	Deobf.	Total
C100	Occlumency [34]	0.219s	0.080s	/	/	/	0.299s
	GroupCover [76]	0.497s	/	0.023s	0.403s	0.049s	0.926s
	Ours	**0.122s**	/	**0.002s**	/	/	**0.124s**
S10	Occlumency [34]	1.288s	0.080s	/	/	/	1.368s
	GroupCover [76]	0.510s	/	0.065s	0.481s	0.027s	1.083s
	Ours	**0.347s**	/	**0.037s**	**0.228s**	**0.016s**	**0.628s**
T200	Occlumency [34]	3.741s	0.080s	/	/	/	3.821s
	GroupCover [76]	✗	✗	✗	✗	✗	✗
	Ours	**0.381s**	/	**0.056s**	**0.364s**	**0.062s**	**0.863s**

Depending on the size of the input, TensorShield adaptively chooses whether to execute tensors within the TEE or after obfuscating them to the REE.

Evaluation: Runtime Performance

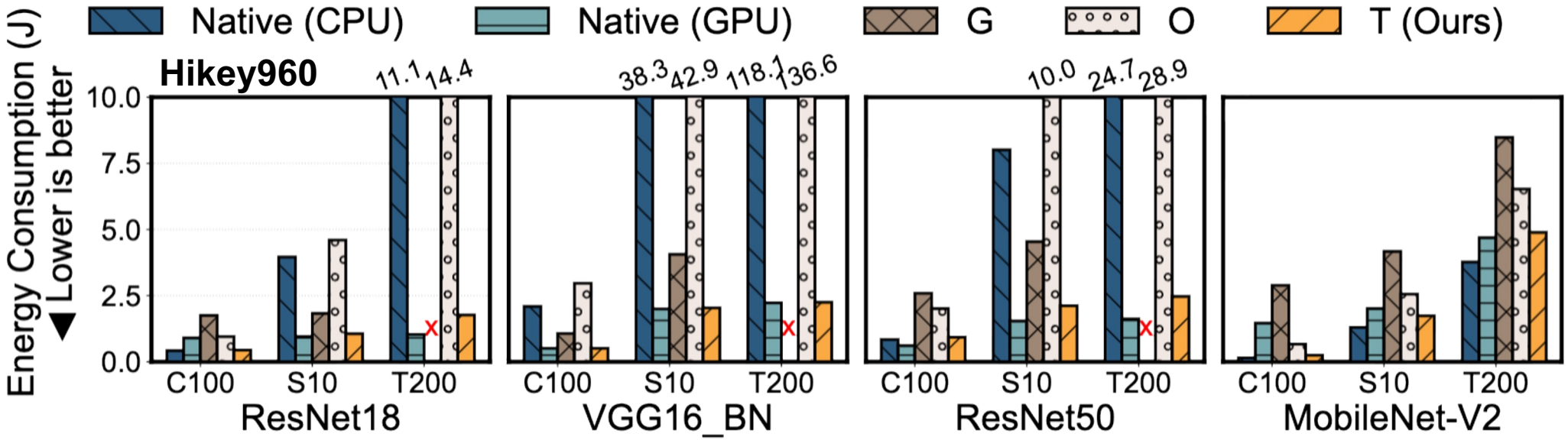
- Inference Time Breakdown

Dataset	Work	Execution time					
		Cal.	Dec.	Comm.	Mask.	Deobf.	Total
C100	Occlumency [34]	0.219s	0.080s	/	/	/	0.299s
	GroupCover [76]	0.497s	/	0.023s	0.403s	0.049s	0.926s
	Ours	**0.122s**	/	**0.002s**	/	/	**0.124s**
S10	Occlumency [34]	1.288s	0.080s	/	/	/	1.368s
	GroupCover [76]	0.510s	/	0.065s	0.481s	0.027s	1.083s
	Ours	**0.347s**	/	**0.037s**	**0.228s**	**0.016s**	**0.628s**
T200	Occlumency [34]	3.741s	0.080s	/	/	/	3.821s
	GroupCover [76]	✗	✗	✗	✗	✗	✗
	Ours	**0.381s**	/	**0.056s**	**0.364s**	**0.062s**	**0.863s**

TensorShield computes critical tensors in the TEE without needing to mask intermediate features, whereas GroupCover obfuscates all parameters to the REE GPU and masks all intermediate features.

Evaluation: Runtime Performance

- **Energy Consumption**



TensorShield achieves an average reduction in energy consumption of **58.66% (up to 91.35%)** and **69.86% (up to 98.35%)** compared to GroupCover and Occlumency, respectively

TensorShield

Safeguarding On-Device Inference by Shielding Critical DNN Tensors with TEE

- *Critical tensor identification*
- *Critical feature identification*
- *Latency-aware Placement*

Thank you for your attention!

Tong Sun, Bowen Jiang, Hailong Lin, Borui Li, Yixiao Teng, Yi Gao, and Wei Dong

If you have any questions, please contact **tongsun@zju.edu.cn**

浙江大學 区块链与数据安全全国重点实验室
STATE KEY LABORATORY OF BLOCKCHAIN AND DATA SECURITY
ZHEJIANG UNIVERSITY

東南大學
SOUTHEAST UNIVERSITY