

Phantom: Privacy-Preserving Deep Neural Network Model Obfuscation in Heterogeneous TEE and GPU System

Juyang Bai¹, Md Hafizul Islam Chowdhury³, Jingtao Li⁴,
Fan Yao³, Chaitali Chakrabarti², Deliang Fan²

¹Johns Hopkins University, ²Arizona State University,
³University of Central Florida, ⁴Sony AI

Corresponding author: Deliang Fan

Email: dfan@asu.edu

Website: <https://faculty.engineering.asu.edu/dfan/>

Open source code: https://github.com/ASU-ESIC-FAN-Lab/PHANTOM_USenix



Sony AI

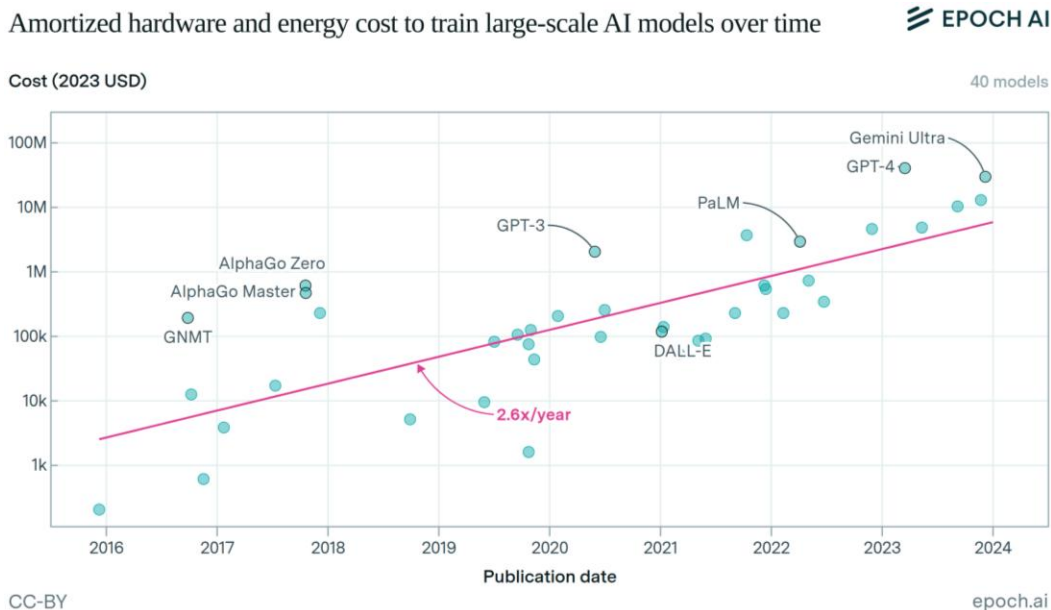
SONY

Content

- **Motivation**
- Threat Model
- Phantom
- Evaluation
- Conclusion

Motivation

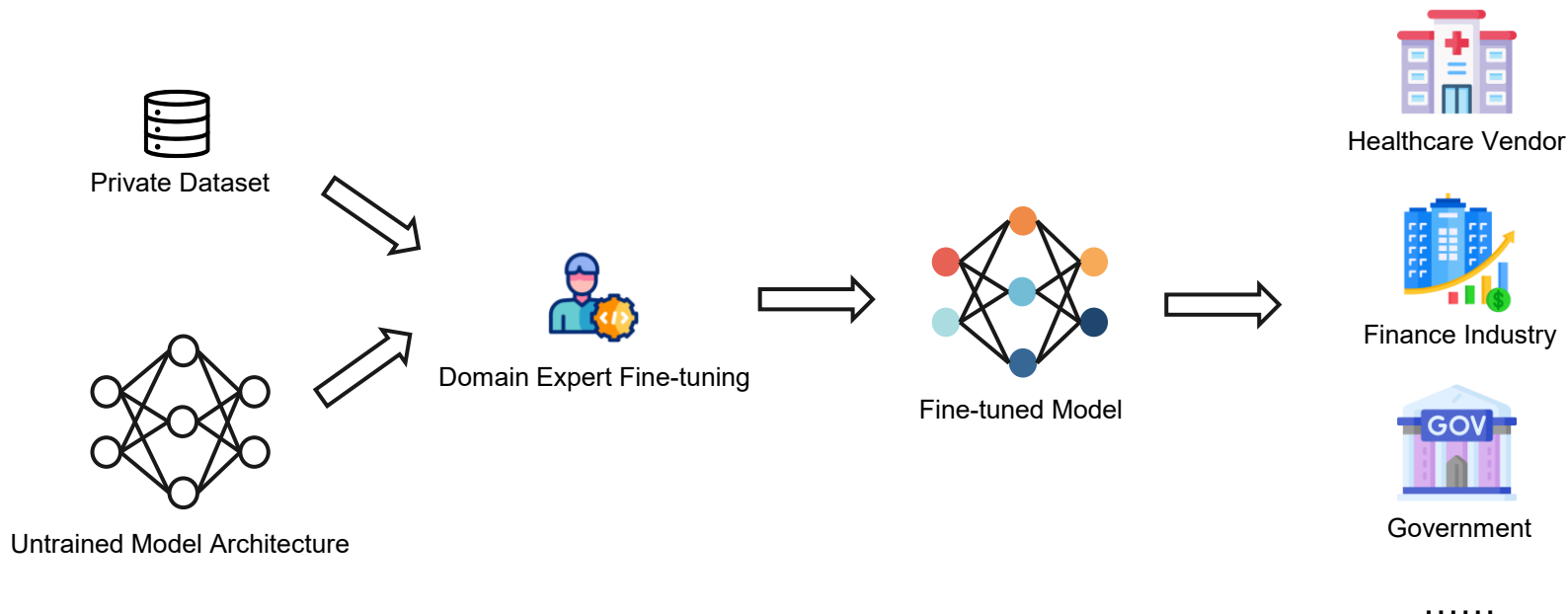
- The cost of training large-scale machine learning models is doubling every nine months.



Source: Cottier & Rahman (2024), Epoch AI [1]

Motivation

- High-stakes domains, such as healthcare, finance and government, require fine-tuning ML models using private data for domain-specific tasks.



Motivation

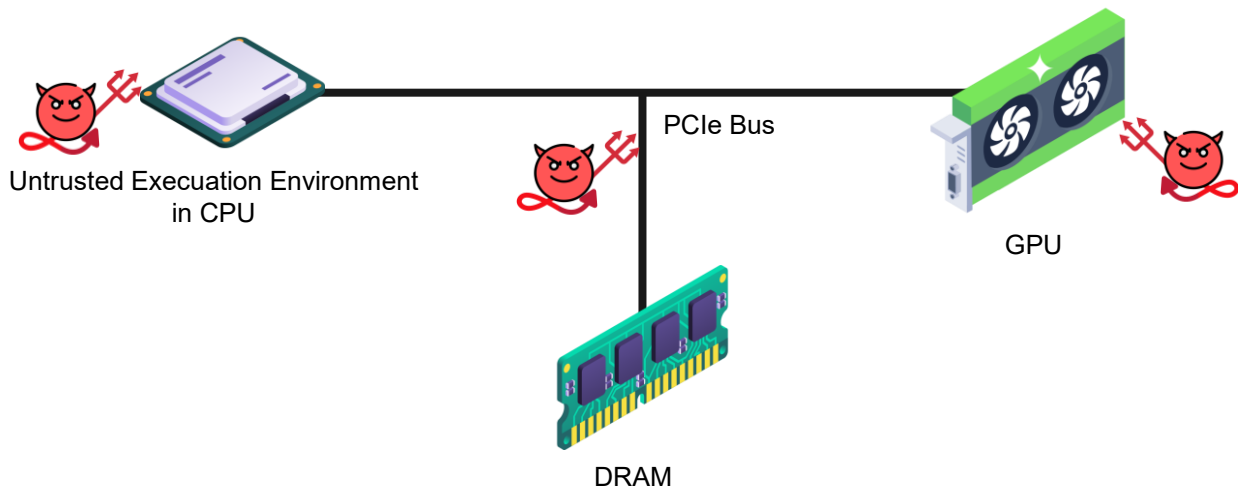
- Security Threats: recent research demonstrates that adversaries can extract model information by exploiting vulnerabilities in untrusted system components:
 - DeepSteal [2]: Extracts DNN model parameters by exploiting memory access patterns through hardware performance counters and cache side-channels.
 - Hermes Attack [3]: Steals neural network architectures and weights by analyzing electromagnetic emanations from GPU memory accesses during inference.
 - Cache Telepathy [4]: Infers DNN model structures and parameters by monitoring cache access patterns across different processes on shared hardware.
 - DeepSniffer [5]: Reconstructs CNN architectures by analyzing memory access traces and timing information during model execution.

Content

- Motivation
- **Threat Model**
- Phantom
- Evaluation
- Conclusion

Threat Model

- Powerful adversaries can access the untrusted execution environment, including the operating system (OS) and GPU.
- Deployed DNN models are assumed to return only class labels, not confidence scores, to both authorized users and adversaries.



Attack Methods

- Model-stealing attack
 - The adversary queries the victim model with a **limited inputs** to collect outputs and builds a transfer dataset. This dataset is then used to train a surrogate model that mimics the victim's behavior.
- Fine-tuning attack
 - The adversary gains access to **at most 10%** of the original training data. Using this limited dataset, they fine-tune a partially known model (e.g., with stolen weights or architecture) to recover its functionality or improve performance.

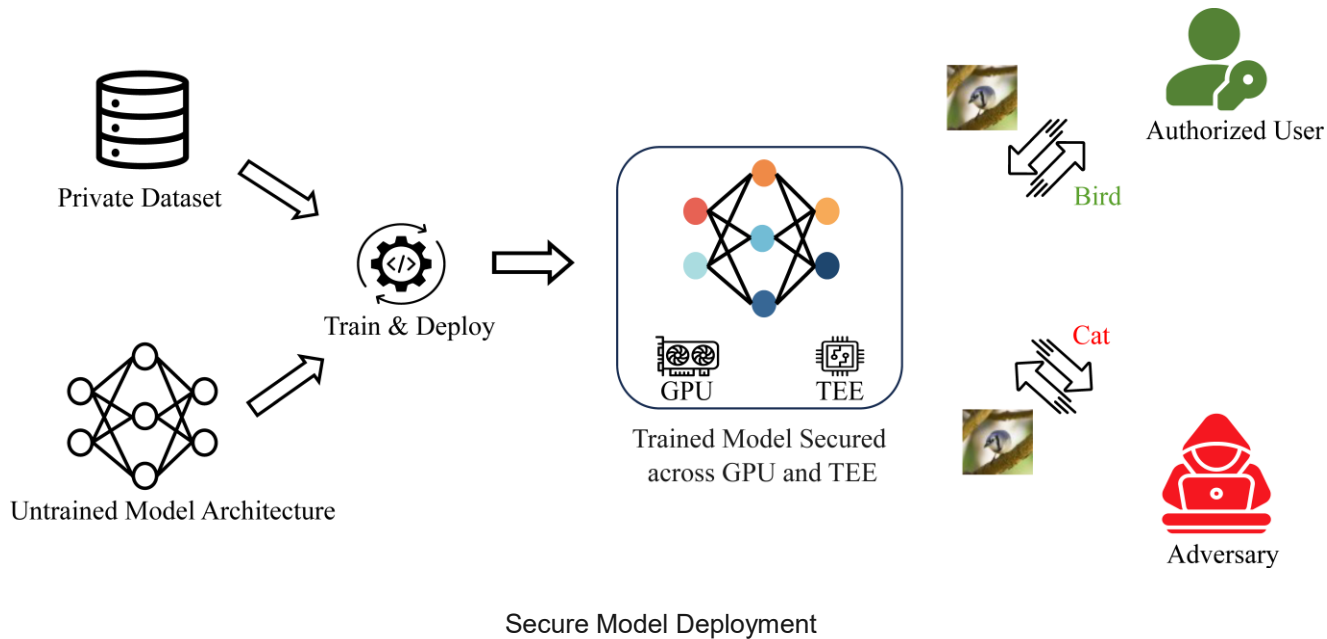
Content

- Motivation
- Threat Model
- **Phantom**
- Evaluation
- Conclusion

Overview

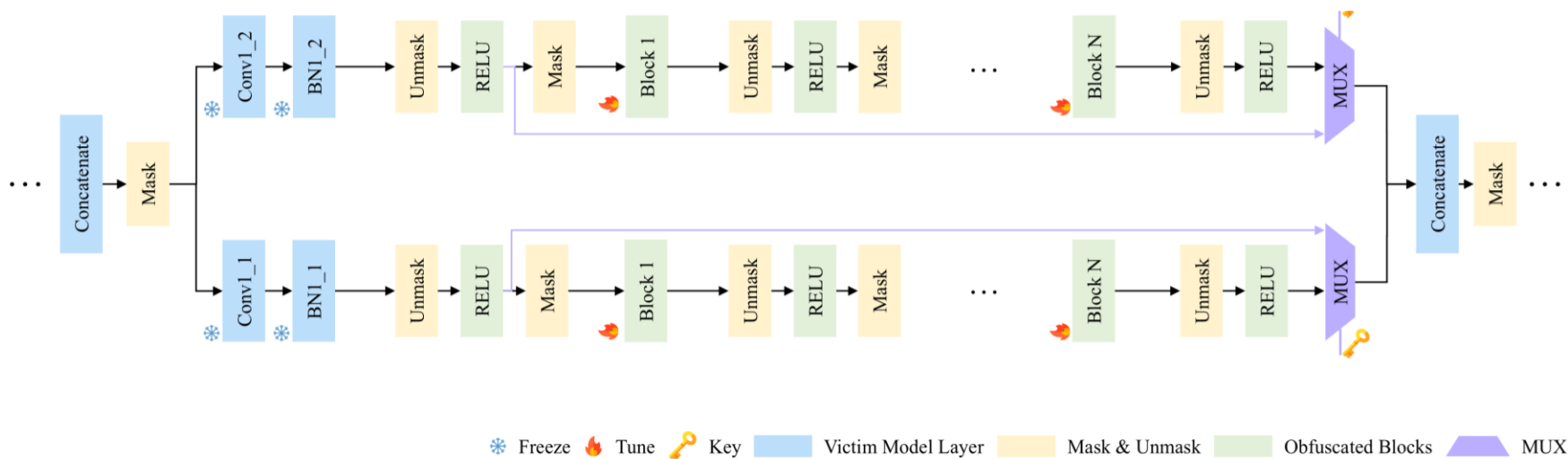
How Can We Protect ML Model Privacy Against Model Stealing Attacks?

Overview

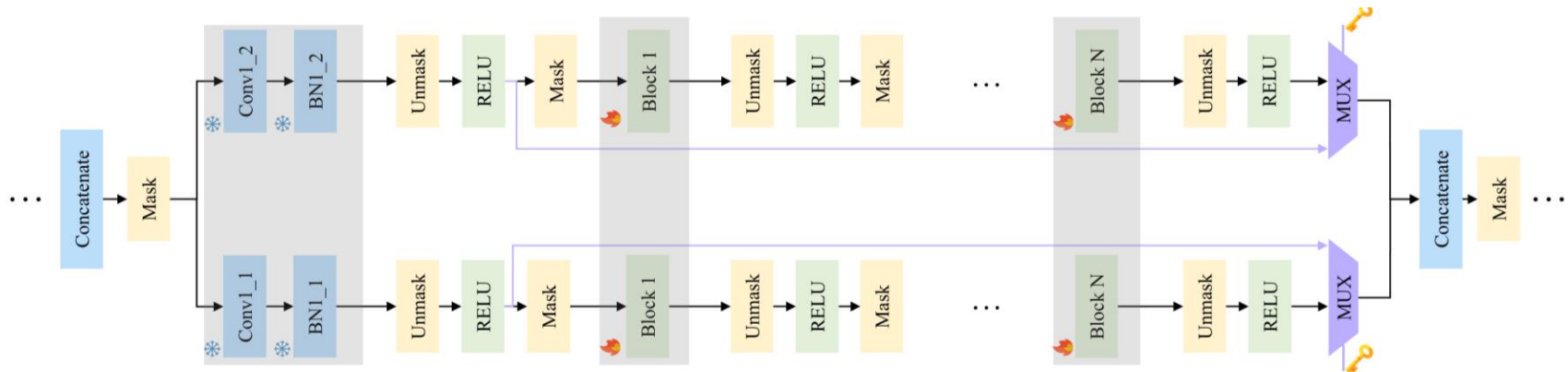


Overview

- We introduce a reinforcement learning-based neural architecture search method that injects small, lightweight obfuscation layers with corresponding "keys" that determine the correct execution path.



Overview

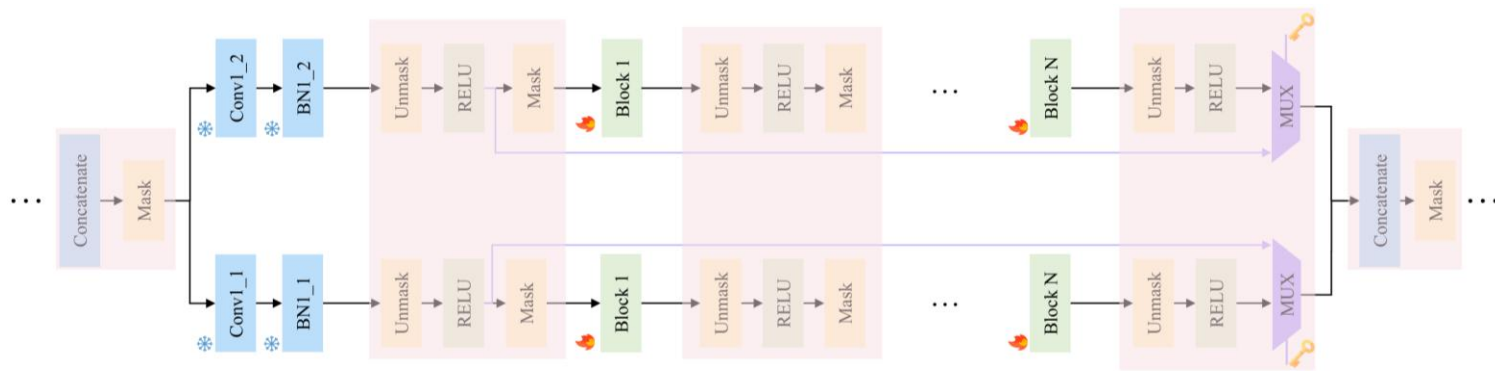


Vulnerable model components remain susceptible to extraction by adversaries

❄ Freeze 🔥 Tune 🔑 Key Victim Model Layer Mask & Unmask Obfuscated Blocks MUX

Overview

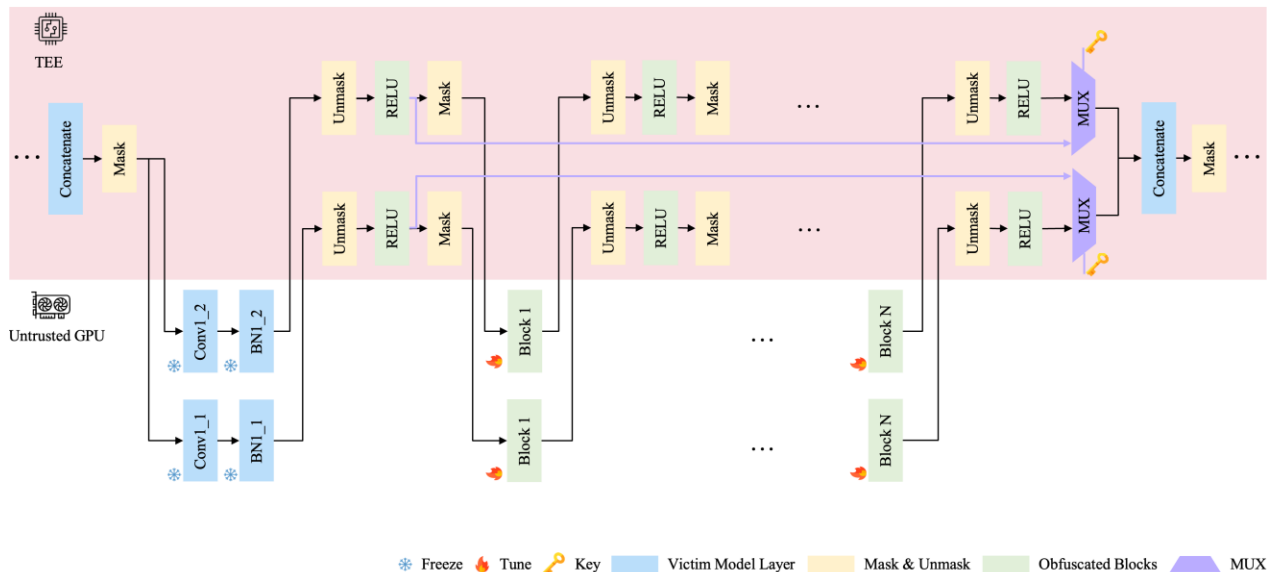
- Mask and Unmask operation using one-time pad (OTP) encryption following Slalom [6]:
 - Mask non-linear layers' sensitive data with random values in TEE before GPU computation, then unmask results to recover original outputs in TEE after GPU computation while maintaining confidentiality throughout the process.



Obfuscated model components contain sensitive privacy information requiring protection

Overview

- With Authorized Key (Authorized Users): MUX operations in TEE use keys to recover real computation paths, ensuring **correct** model predictions for authorized users.
- Without Key (Adversaries): Obfuscated layers in untrusted GPU generate **false** predictions, deliberately misleading adversaries with degraded accuracy.



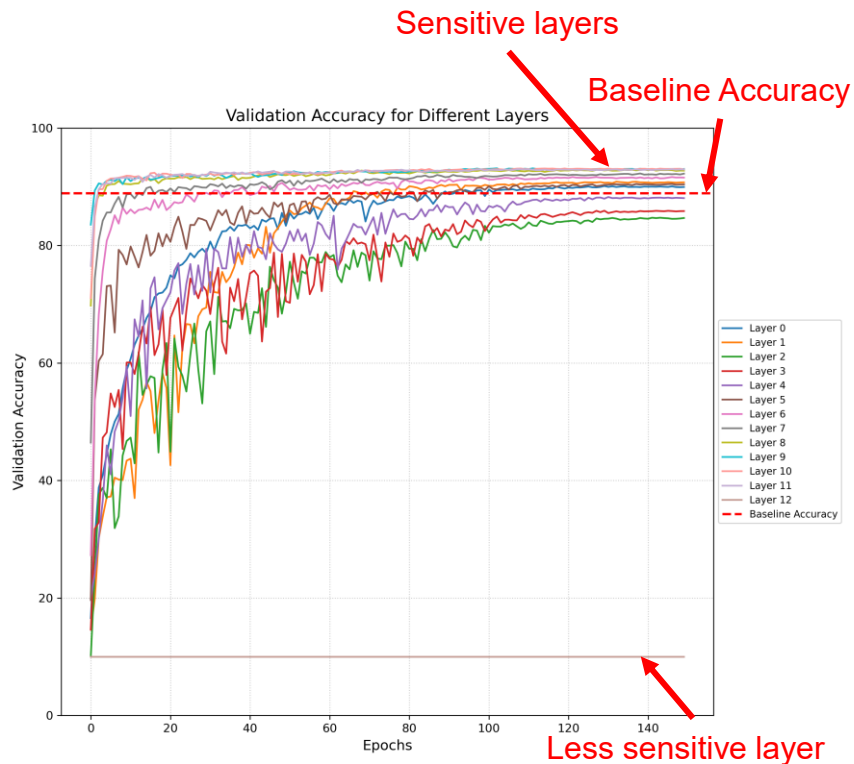
Deploy Obfuscated Model in Heterogeneous TEE and GPU System

Method

- Layer Sensitivity Analysis
- Obfuscated Architecture Transformation
- Obfuscation Layer Training

Layer Sensitivity Analysis

- Identify which layers are most vulnerable to attacks
 - Test individual layers: Add one obfuscation layer after each original layer to create N separate test models
 - Measure defense effectiveness: Train each test model and evaluate how well it resists fine-tuning attacks
 - Select Top- K most sensitive layers: Rank layers by attack resistance ($1/\text{accuracy}$) and choose only the Top- K most sensitive layers



VGG-16 Layer Sensitivity Analysis Example

Obfuscated Architecture Transformation

- Reinforcement learning-based search to select optimal obfuscation layer positions and types from a predefined candidate pool
- Achieves optimal trade-off between **maximizing** model privacy protection and **minimizing** computational and system overhead
- Reward Function

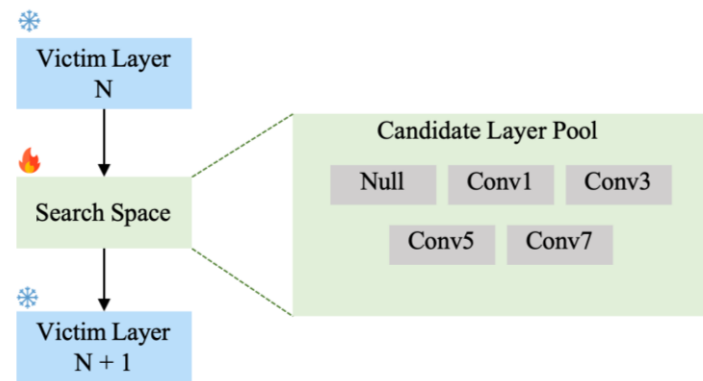
$$R = -(-\alpha \cdot \Delta S + \beta \cdot \Delta L)$$



where:

ΔS : Change in obfuscation effectiveness score

ΔL : Change in system latency or overhead

α, β : Trade-off weights for balancing security vs. performance



 Freeze  Tune

Search space of obfuscation block

Obfuscation Layer Training

- Freeze original model weights: Keep all victim model parameters unchanged to preserve core functionality for authorized users
- Train obfuscation layers to maximize loss: Optimize only the newly added obfuscation layers using cross-entropy loss maximization to deliberately degrade model performance

$$\max \mathcal{L}(\theta) = - \sum_{i=1}^N y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

- Generate misleading outputs for adversaries: Create a model that provides incorrect predictions to unauthorized users while maintaining original performance for authorized user

Content

- Motivation
- Threat Model
- Phantom
- **Evaluation**
- Conclusion

Experiments Setup

- Model: AlexNet, ResNet-18 and VGG-16
- Dataset: CIFAR-10, CIFAR-100 and STL-10
- Hardware
 - Searching and Training
 - An NVIDIA A6000 GPU
 - Inference & System Evaluation
 - SGX1 Platform: Intel Core i7-9700K + NVIDIA GTX 1080Ti
 - SGX2 Platform: Intel Xeon Gold 6342 + NVIDIA A40

Obfuscation Effectiveness

Table 1: Comparison of model performance with and without authorized user keys. In each cell, the left accuracy is with a key, and the right accuracy is without a key. **Bold the lowest defensive accuracy** against attackers.

	AlexNet			ResNet-18			VGG-16		
	CIFAR-10	CIFAR-100	STL10	CIFAR-10	CIFAR-100	STL10	CIFAR-10	CIFAR-100	STL10
No Privacy Left Outside ¹	86.37% / 83.71%	61.96% / 56.46%	80.17% / 76.54%	93.65% / 95.47%	76.79% / 79.94%	86.22% / 87.51%	93.06% / 91.62%	73.11% / 73.03%	89.67% / 89.42%
NNSplitter	84.13% / 59.30%	56.09% / 22.94%	82.37% / 22.99%	93.01% / 9.99%	72.75% / 0.98%	94.79% / 22.28%	93.02% / 10.00%	72.76% / 1.00%	94.80% / 25.11%
Ours (Whole Layer)	84.13% / 9.90%	56.09% / 1.00%	82.37% / 9.97%	93.01% / 10.00%	72.75% / 0.94%	94.79% / 10.53%	93.02% / 10.00%	72.76% / 0.94%	94.80% / 10.53%
Ours (Top-3 Layer)	84.13% / 10.03%	56.09% / 0.98%	82.37% / 10.86%	93.01% / 9.90%	72.75% / 1.19%	94.79% / 10.04%	93.02% / 9.90%	72.76% / 1.19%	94.80% / 10.04%

¹ For 'No Privacy Left Outside' work, the performance with key means the performance of their method's pruned and trained model, while the performance without key is from the pre-trained backbone model.

Accuracy **with** a key Accuracy **without** a key

- Phantom reduces unauthorized model accuracy to near-random performance levels (e.g., ~10% on CIFAR-10/STL-10, ~1% on CIFAR-100) while maintaining full accuracy for authorized users.

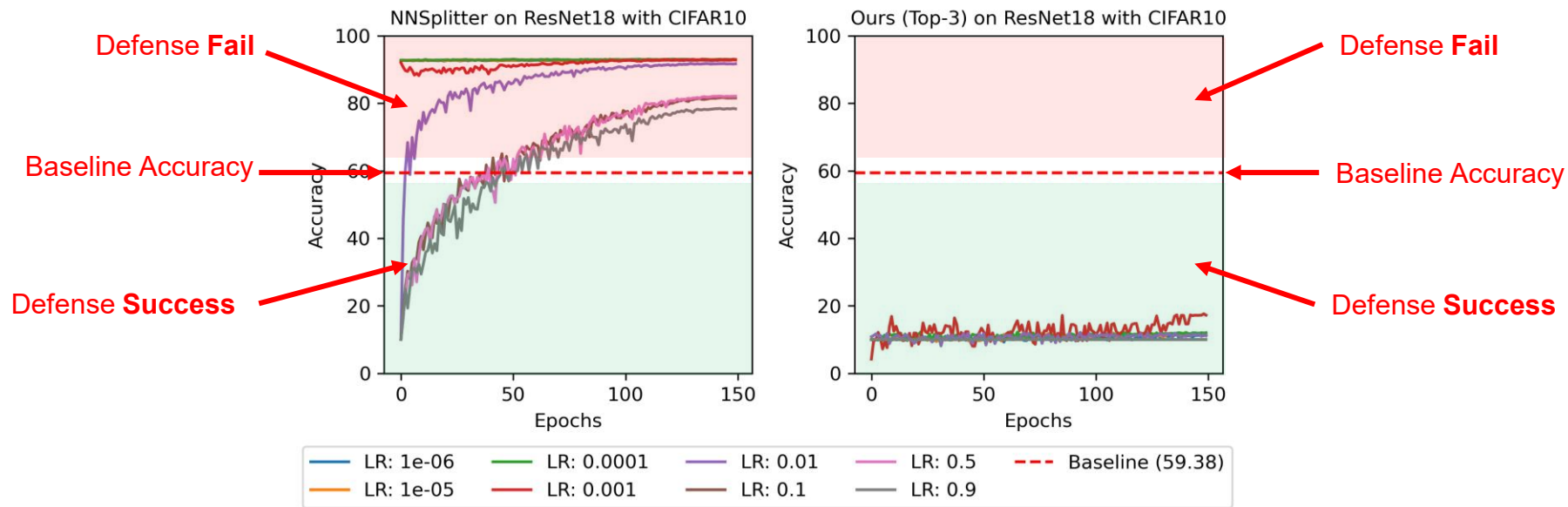
Fine-Tuning Attack

Table 2: Comparison of defensive accuracy for fine-tuning attack. green indicates success and red indicates failure.

	AlexNet			ResNet-18			VGG-16			Average
	CIFAR-10	CIFAR-100	STL-10	CIFAR-10	CIFAR-100	STL-10	CIFAR-10	CIFAR-100	STL-10	
Baseline (Random)	77.26%	41.87%	52.01%	59.38%	37.33%	50.47%	31.26%	47.91%	64.35%	51.32%
No Privacy Left Outside (LR: 0.01)	83.98%	59.21%	81.13%	85.09%	59.27%	90.71%	90.79%	68.97%	93.97%	79.25%
No Privacy Left Outside (LR: 0.001)	80.35%	54.82%	79.87%	78.36%	50.02%	86.52%	85.91%	60.37%	93.39%	74.40%
NNSplitter (LR: 0.01)	9.99%	1.00%	10.00%	91.79%	70.39%	75.89%	93.19%	67.91%	77.99%	55.35%
NNSplitter (LR: 0.001)	84.31%	58.09%	79.39%	93.00%	71.98%	75.77%	93.81%	72.23%	78.18%	80.26%
Ours (Whole Layer) (LR: 0.01)	10.00%	1.00%	10.03%	12.13%	32.83%	10.03%	10.01%	12.40%	10.03%	12.05%
Ours (Whole Layer) (LR: 0.001)	10.00%	1.00%	10.03%	17.64%	11.23%	10.03%	11.27%	6.42%	10.03%	9.74%
Ours (Top-3) (LR: 0.01)	10.00%	1.00%	10.03%	10.00%	1.43%	10.03%	10.00%	13.92%	10.03%	8.49%
Ours (Top-3) (LR: 0.001)	10.00%	1.00%	11.18%	10.00%	1.00%	17.22%	10.00%	12.67%	10.03%	9.23%

- Phantom consistently reduces fine-tuning attack success to near-random levels (8.49%-12.05% average) while competing methods fail with success rates above 51.32% baseline, demonstrating superior defense effectiveness.

Fine-Tuning Attack



- The fine-tuning attack evaluation tested defense effectiveness across eight different learning rates using 150 training epochs with SGD optimizer and CosineAnnealingLR scheduler.
- Phantom demonstrates consistent defense success across all tested learning rates.

Model Stealing Attack

Table 3: Comparison of defensive accuracy for model stealing attack. We **bold** the lowest defensive accuracy. The lower the defensive accuracy for model stealing attacks, the better the performance in defending against model stealing attacks.

	AlexNet			ResNet-18			VGG-16			Average
	CIFAR-10	CIFAR-100	STL10	CIFAR-10	CIFAR-100	STL10	CIFAR-10	CIFAR-100	STL10	
No Privacy Left Outside ¹	19.04%	8.27%	24.15%	31.40%	10.90%	29.19%	30.87%	9.78%	32.92%	21.84%
NNSplitter	10.00%	1.00%	15.90%	12.50%	1.10%	11.00%	35.60%	14.30%	15.40%	12.89%
Ours (Whole Layer)	10.00%	1.00%	10.00%	10.00%	1.00%	10.00%	10.00%	1.00%	10.00%	7.00%
Ours (Top-3 Layer)	10.00%	1.00%	10.00%	10.00%	1.00%	10.00%	10.00%	1.00%	10.00%	6.99%

¹ The accuracy of the model stealing attack 'No Privacy Left Outside' is sourced from their paper.

- Phantom significantly reduces model stealing success rates, with surrogate models achieving only random baseline performance ($\approx 10\%$ for CIFAR-10/STL-10, $\approx 1\%$ for CIFAR-100).
- Top-3 layer obfuscation achieves nearly equivalent defense effectiveness as full-layer obfuscation while reducing computational overhead.

System Overhead

Table 4: Inference Time Overhead Evaluation. We **bold** the lowest total inference time with its breakdown time.

		Total Execution Latency (<i>ms</i>)	GPU Latency (<i>ms</i>)	TEE Latency (<i>ms</i>)	Data Transfer Latency (<i>ms</i>)
ResNet18 ¹	GPU-Only	2.51	2.51 (100%)	-	-
	TEE-Only	34.27	-	34.27 (100%)	
	GPU+TEE	11.09	1.66 (15%)	5.96 (36%)	5.42 (49%)
No Privacy Left Outside	GPU+TEE	17.42	1.72 (10%)	5.96 (34%)	9.74 (56%)
Ours (Whole Layers)	GPU+TEE	37.11	4.92 (13%)	12.02 (32%)	20.17 (54%)
Ours (Top-3 Layers)	GPU+TEE	11.33	1.65 (14%)	3.31 (29%)	6.41 (57%)

¹ The ResNet18 evaluated here is a layer-branched version in which all convolutional layers are split into two parallel branches.

- Phantom's Top-3 obfuscation strategy achieves the best runtime performance among all evaluated defense mechanisms.
- SGX 2.0 shifts the performance bottleneck from TEE computation (as in SGX 1.0) to data transfer overhead between TEE and GPU, accounting for 50-60% of total execution time.

Content

- Motivation
- Threat Model
- Phantom
- Evaluation
- **Conclusion**

Conclusion

- We introduces a reinforcement learning-based neural architecture search method that injects small, lightweight obfuscation layers with corresponding "keys" that determine the correct execution path, achieving an optimal balance between obfuscation effectiveness and inference overhead on both SGX1 and SGX2 platforms.
- SGX2 shifts the performance bottleneck from TEE computation (as observed in SGX1) to data transfer overhead between the TEE and GPU.



Thanks & Questions?

This work is supported in part by the National Science Foundation under Grant No.2452657, No.2503906, No. 2019536 and No.2505209.



Sony AI

SONY

Reference

1. Cottier, B., & Rahman, R. (2024). Training compute costs are doubling every nine months for the largest AI models. Epoch AI. <https://epoch.ai/data-insights/cost-trend-large-scale>
2. Rakin, Adnan Siraj, et al. "Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories." 2022 IEEE symposium on security and privacy (SP). IEEE, 2022.
3. Zhu, Yuankun, et al. "Hermes attack: Steal {DNN} models with lossless inference accuracy." 30th USENIX Security Symposium (USENIX Security 21). 2021.
4. Yan, Mengjia, Christopher W. Fletcher, and Josep Torrellas. "Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures." 29th USENIX Security Symposium (USENIX Security 20). 2020.
5. Hu, Xing, et al. "Deepsniffer: A dnn model extraction framework based on learning architectural hints." Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020.
6. Florian Tramer and Dan Boneh. "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware". In: arXiv preprint arXiv:1806.03287 (2018).