DistilLock: Safeguarding LLMs from Unauthorized Knowledge Distillation on the Edge

Asmita Mohanty[1], Gezheng Kang[2], Lei Gao[1], Murali Annavaram[1]
[1]University of Southern California, [2]University of California, Davis
[1]{asmitamo, leig, annavara}@usc.edu, [2]gzkang@ucdavis.edu

Abstract

Large Language Models (LLMs) have demonstrated strong performance across diverse tasks, but fine-tuning them typically relies on cloud-based, centralized infrastructures. This requires data owners to upload potentially sensitive data to external servers, raising serious privacy concerns. An alternative approach is to fine-tune LLMs directly on edge devices using local data; however, this introduces a new challenge: the model owner must transfer proprietary models to the edge, which risks intellectual property (IP) leakage. To address this dilemma, we propose DistilLock, a TEE-assisted fine-tuning framework that enables privacy-preserving knowledge distillation on the edge. In DistilLock, a proprietary foundation model is executed within a trusted execution environment (TEE) enclave on the data owner's device, acting as a secure black-box teacher. This setup preserves both data privacy and model IP by preventing direct access to model internals. Furthermore, DistilLock employs a model obfuscation mechanism to offload obfuscated weights to untrusted accelerators for efficient knowledge distillation without compromising security. We demonstrate that DistilLock prevents unauthorized knowledge distillation processes and model-stealing attacks while maintaining high computational efficiency, but offering a secure and practical solution for edge-based LLM personalization.

1 Introduction

Proprietary Large Language Models (LLMs) such as GPT [1], Gemini [2], and Claude [3] have demonstrated superior performance compared to open-source models, largely due to the availability of extensive computing infrastructure, high-quality datasets, and advanced optimization pipelines. As a result, their trained weights are considered valuable intellectual property and are typically not released to the public. Downstream users who wish to customize these models must rely on cloud-based APIs and submit labeled data to remote servers for fine-tuning. This centralized setup not only incurs cloud costs but also raises serious privacy concerns, particularly for sensitive or proprietary datasets.

The growing availability of edge-centric ML accelerators [4, 5] and resource-efficient training algorithms [6, 7] has made it feasible to fine-tune LLMs with billions of parameters directly on edge devices. This shift offers a compelling privacy advantage, as user data remains local and never leaves the device. However, this alternative comes with a new risk: the model owner must send their proprietary foundation model to an untrusted edge environment, thereby exposing sensitive model internals and violating intellectual property boundaries.

Recent works have explored methods to mitigate this dilemma by enabling partial decentralization of the fine-tuning process. For instance, Offsite-Tuning [8] compresses the foundation model and deploys it on the user's device as a lightweight backbone to train local adapters, which are then sent back to the model owner for integration. Split-and-Privatize [9] divides the model into two

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Lock-LLM.

parts, training the first half locally and transmitting privatized intermediate embeddings to the server, where the second half completes the forward and backward pass. While these solutions reduce data exposure, they sacrifice training performance due to model compression or differential privacy, and still depend on the remote model owner to perform inference after training.

To overcome these limitations, we introduce DistilLock, a secure on-device knowledge distillation framework that protects both user data privacy and foundation model confidentiality. DistilLock uses Trusted Execution Environments (TEEs) to run the proprietary foundation model as a black-box teacher inside a hardware-protected enclave on the user's device. This allows knowledge distillation to happen entirely on-device without exposing the model weights or user data. Crucially, to avoid the computational overhead of a full model forward pass within the TEE, DistilLock employs a model obfuscation strategy that enables most compute-heavy operations to be offloaded to untrusted accelerators (e.g., GPUs). The TEE is used only for lightweight authorization, and the offloaded weights are obfuscated so they remain functional but cannot be reverse-engineered to reveal sensitive model information. Once distillation completes, the obfuscated teacher is removed, leaving only the distilled student model tailored to the user's task.

We evaluate DistilLock across multiple foundation models and diverse downstream tasks. Our results show that DistilLock prevents unauthorized knowledge distillation and resists model-stealing attacks, even when adversaries attempt surrogate training on the obfuscated model. Furthermore, we analyze the efficiency of our framework and demonstrate that DistilLock introduces only minimal computation overhead through lightweight TEE authorization and obfuscated model offloading.

2 Background and Threat Model

Trusted Execution Environment. Trusted Execution Environments (TEEs) are hardware-enforced secure regions that provide confidentiality and integrity guarantees for code and data during execution. Widely used implementations include Intel SGX [10], Intel TDX [11], and ARM TrustZone [12]. Despite architectural differences, they share the common goal of shielding sensitive computation from untrusted system software such as the OS or hypervisor. In this work, we prototype our system using Intel SGX, which offers fine-grained enclave abstraction for user-space applications and is readily available on edge CPUs. Although recent GPUs like the NVIDIA H100 support TEEs [13], such capabilities are not yet widely deployed in the edge environments we target.

Knowledge Distillation. Knowledge distillation (KD) transfers knowledge from a large, high-capacity teacher model to a smaller student model by training the student to mimic the teacher's behavior. The student learns from both the ground-truth labels and the teacher's soft output distribution, enabling it to achieve comparable performance with significantly fewer resources. Let y be the ground-truth label, q_S and q_T be the logits from the student and teacher, respectively, and let $q^{(\tau)} = \text{softmax}(q/\tau)$ denote the softened output with temperature $\tau > 1$. The distillation loss is defined as: $\mathcal{L}_{\text{KD}} = \alpha \cdot \mathcal{L}_{\text{CE}}(y, q_S) + \beta \cdot \mathcal{L}_{\text{KL}}(q_T^{(\tau)}, q_S^{(\tau)})$, where $\mathcal{L}_{\text{CE}}$ is the standard cross-entropy loss and $\mathcal{L}_{\text{KL}}$ is the Kullback–Leibler divergence between the teacher and student distributions. While many advanced variants of the KD algorithm have been proposed [14, 15, 16], we focus on the vanilla formulation in this work, as our primary goal is to study the security and system-level aspects of on-device distillation.

Threat Model. We define two primary parties: the defender and the attacker. The defender owns the model deployed on an edge device, while the attacker's goal is to steal it.

Defender's Goal: The defender's primary objective is to ensure their deployed teacher model, $\mathcal{M}_T$, generates the correct logits, q_T, only when proper authorization is given by a TEE. To maintain efficiency, the defender offloads the majority of the computation to an untrusted GPU, which can be accessed in a white-box manner by the user. The defender's ultimate goal is to degrade white-box attacks to a black-box setting, preventing the attacker from accessing the model's weights and intellectual property.

Adversary's Goal and Capability: The attacker's goal is to develop an independent surrogate model, $\mathcal{M}'_T$, that can replicate the performance of the authorized teacher model, $\mathcal{M}_T$. Our threat model assumes that the attacker has white-box access to the model details, including architecture and weights, for any part of the model that resides outside the TEE (e.g., on the GPU). The attacker can use existing techniques [17, 18] to infer the full architecture and obtain the weights of these

exposed components. We also assume the attacker possesses a well-labeled dataset for the targeted task, which they can use to facilitate the attack. The TEE itself, however, is considered a secure and uncompromisable environment.

3 DistilLock's Defense Mechanism

TEEs often lack support for powerful accelerators like GPUs, making them approximately 50 times slower for running LLMs [19]. Therefore, different methods [20, 21, 22] have been proposed to shield and execute partial models inside TEEs and offload the remaining parts to an untrusted environment. Inspired by recent works [19, 23], we leverage model obfuscation to ensure model and data confidentiality while preserving utility and efficiency. DistilLock consists of two steps: model obfuscation and model authorization.

Model Obfuscation. Let $x \in \mathbb{R}^{n \times d}$ denote the input where n is the sequence length (e.g., the number of tokens) and d is the model dimension. We define the forward pass in a Transformer block in typical LLMs like Llama [24] and Qwen [25] as follows:

$$\begin{aligned}
&Q = xW_q, \quad K = xW_k, \quad V = xW_v, && W_q, W_k, W_v \in \mathbb{R}^{d \times d}, \\
&u = \text{Softmax}\left(\frac{QK^T}{\sqrt{k}}\right) VW_o, && W_o \in \mathbb{R}^{d \times d}, \\
&v = \text{LayerNorm}(u + x; \gamma_1, \beta_1), && \gamma_1, \beta_1 \in \mathbb{R}^d, \\
&z = (\text{SiLU}(vW_1)vW_3)W_2, && W_1, W_3 \in \mathbb{R}^{d \times m}, \quad W_2 \in \mathbb{R}^{m \times d}, \\
&y = \text{LayerNorm}(z + v; \gamma_2, \beta_2), && \gamma_2, \beta_2 \in \mathbb{R}^d,
\end{aligned} \tag{1}$$

Let $\pi_{emb} \in \{0,1\}^{vocab \times vocab}$, $\pi \in \{0,1\}^{d \times d}$ denote two permutation matrices. Before the model is transferred to the user side, the model owner first transforms the parameters as follows:

$$\begin{aligned}
W'_{emb} &= \pi_{emb}^T W_{emb}, \\
W'_q = \pi^T W_q, \quad W'_k = \pi^T W_k, \quad W'_v &= \pi^T W_v, \quad W'_o = W_o \pi, \\
W'_1 = \pi^T W_1, \quad W'_3 = \pi^T W_3, \quad W'_2 &= W_2 \pi, \\
\gamma'_1 = \gamma_1 \pi, \quad \beta'_1 = \beta_1 \pi, \quad \gamma'_2 &= \gamma_2 \pi, \quad \beta'_2 = \beta_2 \pi, \\
W'_{cls} &= \pi^T W_{cls},
\end{aligned} \tag{2}$$

where $W_{emb} \in \mathbb{R}^{vocab \times d}$ is the word embedding layer and $W_{cls} \in \mathbb{R}^{d \times vocab}$ is the final linear classifier.

Model Authorization. The obfuscated model and the encrypted permutation matrix π are delivered to the user, who then instantiates the TEE. As shown in Figure 1, the user's input $h \in \mathbb{R}^{n \times vocab}$ (represented as one-hot vectors) is first sent to the TEE. Inside the TEE, h is encrypted with a one-time pad (OTP) $m \in \mathbb{R}^{n \times vocab}$ and multiplied by π_{emb} on the GPU: $h' = (h + m)\pi_{emb}$. This operation hides both m and π_{emb} from the user. The encrypted and permuted input h' then exits the TEE and is multiplied by the obfuscated embedding layer: $h'W'_{emb} = (h + m)\pi_{emb} \cdot \pi_{emb}^T W_{emb} = (h+m)W_{emb}$, since $\pi_{emb}\pi_{emb}^T = I$. The result $(h+m)W_{emb}$ is returned to the TEE, where the OTP is removed and a new permutation is applied: $x' = \big((h+m)W_{emb} - mW_{emb}\big)\pi = hW_{emb}\pi = x\pi$, with mW_{emb} pre-computed offline.

The functionality of each subsequent Transformer block is preserved with the obfuscated parameters.

$$\begin{aligned}
Q' = x'W'_q = x\pi\pi^T W_q = Q, \quad K' = x'W'_k = x\pi\pi^T W_k = K, \quad V' = x'W'_v = x\pi\pi^T W_v = V, \\
u' = \text{Softmax}\left(\frac{QK^T}{\sqrt{k}}\right) V'W'_o = \text{Softmax}\left(\frac{QK^T}{\sqrt{k}}\right) VW_o\pi = u\pi, \\
v' = \text{LayerNorm}(u' + x\pi; \gamma_1\pi, \beta_1\pi) = \text{LayerNorm}(u\pi + x\pi; \gamma_1\pi, \beta_1\pi) = v\pi, \\
z' = (\text{SiLU}(v'W'_1)v'W'_3)W2' = (\text{SiLU}(v\pi\pi^T W_1)v\pi\pi^T W_3)W_2\pi = z\pi, \\
y' = \text{LayerNorm}(z' + v'; \gamma_2\pi, \beta_2\pi) = \text{LayerNorm}(z\pi + v\pi; \gamma_2\pi, \beta_2\pi) = y\pi,
\end{aligned} \tag{3}$$

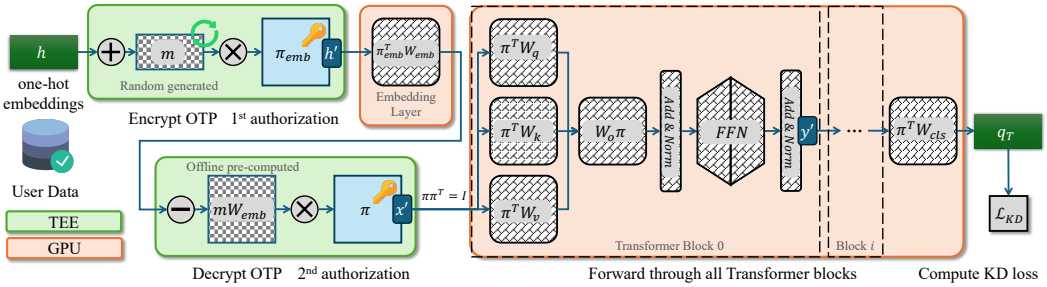

Figure 1: Overview of DistilLock. The proprietary teacher model is obfuscated before deployment and executed with TEE authorization on the user's device. The TEE applies the correct permutation and one-time pad to the input, while compute-intensive operations run on untrusted accelerators using obfuscated weights. This design preserves model functionality for authorized knowledge distillation while preventing unauthorized access or model extraction.

The output y' of this Transformer block will be the input of the next Transformer block. At the last Transformer block, the output y'_{last} is multipled with the final linear classifier W'_{cls} to produce the correct logits $q_T = y'_{last} W'_{cls} = y_{last} \pi \pi^T W_{cls}$. Since the permutation matrix π is maintained throughout the output of each decoder layer, encrypting the input for the subsequent layers, TEE computation is only required for the initial word embedding layer. Without the authorization in TEE by applying the correct π_{emb} and π to input, the teacher model will not produce the correct logits q_T, thereby locking the KD process. All other operations, including loss evaluation and the student model's forward and backward passes, proceed as in standard KD on GPU.

Security Analysis. Our security analysis focuses on a model-stealing threat model, where an adversary attempts to extract model functionality by exploiting query responses and performing surrogate training. An attacker might attempt to recover the original parameters by guessing the permutation matrix π. However, the probability of guessing the correct matrix is approximately $\frac{1}{d!}$, where d is the dimension of the model. This is computationally infeasible for foundational LLMs.

In addition, since the regular word embedding layer is a look-up operation with negligible compute cost, while in DistilLock, it becomes a matrix multiplication operation after adding one-time pad noise (i.e., $(h+m)W_{emb}$), which could add additional overhead. We limit the one-time pad noise m to be a random spare k-hot vector with a number of hot keys $k \geq \frac{h}{log_2(h)}$, which reduces the expensive matrix multiplication operation to a vector summation operation while still having comparable entropy to a uniform random vector to preserve the security guarantees.

4 Experiments

4.1 Distillation Lockdown

We evaluate DistilLock under two scenarios: (i) the user is authorized to perform knowledge distillation on their local device after receiving the obfuscated teacher model from the model owner, and (ii) the user is unauthorized, where the input embeddings lack the correct transformation π_{emb} and π before being passed through the teacher model. We use LLaMA3.1-8B and Qwen2-1.5B as teacher models, with LLaMA3.2-1B and Qwen2-0.5B serving as the corresponding student models [24, 25]. The student models are fine-tuned on the Alpaca dataset [26] under teacher guidance and evaluated on the HellaSwag [27] and CommonsenseQA [28] benchmarks. As shown in Figure 2, authorized knowledge distillation yields accuracy gains over pre-distillation baselines, whereas unauthorized distillation under DistilLock collapses student performance because the obfuscated teacher produces random logits that provide no useful learning signal.

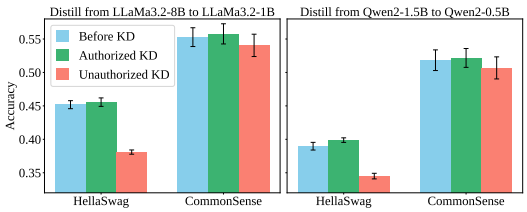

Figure 2: Knowledge distillation results under DistilLock. Authorized distillation improves student accuracy over pre-distillation baselines, while unauthorized distillation collapses performance due to random logits from the obfuscated teacher.

Table 1: Evaluation of surrogate model attack accuracy under different defense schemes. The last row reports average accuracy relative to the black-box baseline. The lowest attack accuracy for each task is shown in green, and the second-lowest in red. DistilLock consistently achieves the lowest attack accuracy across tasks, demonstrating effectiveness in mitigating surrogate model extraction.

Models	Tasks	BlackBox	WhiteBox	DarkneTZ	ShadowNet	Serdab	**DistilLock**
LLaMA3.2-3B	SST-2	67.43%	92.43%	92.54%	92.55%	67.51%	63.17%
	ARC-e	29.14%	74.31%	30.84%	73.95%	29.68%	25.32%
	ARC-c	29.93%	56.80%	30.41%	48.98%	30.53%	28.53%
	BoolQ	64.20%	78.30%	78.40%	80.40%	65.94%	65.00%
	HellaSwag	26.30%	57.30%	54.90%	37.10%	26.30%	24.90%
	RTE	53.79%	79.78%	80.14%	74.01%	62.11%	54.51%
	QQP	66.60%	80.80%	81.60%	78.80%	68.50%	66.70%
	QNLI	54.90%	82.80%	79.10%	82.40%	67.10%	56.10%
Qwen2.5-3B	SST-2	66.63%	93.69%	91.28%	93.23%	69.71%	65.55%
	ARC-e	27.69%	88.52%	28.41%	83.61%	28.05%	25.87%
	ARC-c	26.53%	77.89%	26.53%	71.43%	29.53%	24.49%
	BoolQ	52.50%	82.40%	78.80%	80.80%	62.50%	51.30%
	HellaSwag	26.30%	70.40%	68.90%	68.90%	26.90%	25.10%
	RTE	45.85%	83.03%	84.84%	83.03%	51.26%	48.74%
	QQP	71.20%	84.30%	80.90%	83.80%	69.81%	68.70%
	QNLI	53.90%	85.80%	90.10%	51.00%	61.40%	56.60%
Qwen2.5-1.5B	SST-2	66.62%	93.58%	84.73%	93.35%	69.74%	65.68%
	ARC-e	30.96%	87.25%	27.76%	79.31%	28.23%	27.51%
	ARC-c	28.57%	79.93%	29.36%	64.29%	29.93%	27.64%
	BoolQ	52.70%	79.40%	76.70%	79.50%	69.60%	52.50%
	HellaSwag	25.70%	60.60%	63.23%	60.70%	27.30%	25.30%
	RTE	46.57%	77.98%	79.78%	68.95%	54.87%	46.43%
	QQP	71.80%	84.10%	83.10%	84.40%	74.60%	70.40%
	QNLI	52.80%	85.30%	86.00%	82.10%	62.30%	52.40%
Qwen2-1.5B	SST-2	66.63%	93.23%	84.12%	91.40%	70.98%	67.09%
	ARC-e	30.96%	78.69%	28.31%	77.76%	28.68%	25.13%
	ARC-c	28.91%	79.10%	26.12%	54.42%	29.77%	27.93%
	BoolQ	52.70%	81.50%	76.70%	76.50%	61.30%	51.70%
	HellaSwag	26.30%	59.70%	63.74%	54.90%	26.53%	24.10%
	RTE	46.57%	83.03%	78.34%	76.53%	50.18%	48.01%
	QQP	70.40%	84.30%	81.70%	84.70%	74.80%	70.40%
	QNLI	52.80%	85.80%	84.50%	74.10%	66.90%	52.45%
Relative Average Attack Accuracy		1.00×	1.69×	1.41×	1.58×	1.09×	0.98×

4.2 Adversarial Training

While adversarial users can be prevented from performing knowledge distillation, they may still attempt to extract the teacher model by training a surrogate that replicates its performance, as described in Section 2. To evaluate the robustness of defense mechanisms against such model extraction attacks, we simulate surrogate training across diverse NLP tasks: AI2 Reasoning Challenge (ARC-e, ARC-c) [29] for science reasoning, GLUE [30] tasks including SST-2 for sentiment classification, RTE [31] for textual entailment, QNLI [32] for question-answering, and QQP [33] for question-pair similarity, as well as HellaSwag [27] for commonsense reasoning and BoolQ [34] for yes/no reading comprehension.

We consider six attack scenarios with different adversary capabilities to comprehensively measure defense effectiveness. The white-box attack assumes full access to model weights, yielding the best achievable attack accuracy baseline, while the black-box attack assumes knowledge of the architecture only with random weight initialization, representing the worst-case baseline. Intermediate defenses include Serdab [20], which protects only the first transformer layer inside the TEE; DarkneTZ [21], which shields the final transformer layer and subsequent components; and ShadowNet [22], which obfuscates linear layers via matrix transformations while executing all other layers outside the TEE. Finally, we evaluate DistilLock, in which the attacker initializes the surrogate model directly with the obfuscated weights. A more detailed experimental setup is provided in Appendix C.

Results in Table 1 demonstrate that DistilLock consistently achieves the lowest attack success across all evaluated tasks and models. On average, DistilLock reduces surrogate model accuracy to 0.98× relative to the black-box baseline, effectively neutralizing the attacker's advantage. In contrast, prior defenses such as ShadowNet (1.58×) and DarkneTZ (1.41×) leave significant headroom for the adversary, highlighting the stronger robustness of our approach.

4.3 Efficiency Cost

We evaluate efficiency by measuring the fraction of total FLOPs that must be executed inside the TEE under different defense schemes. As shown in Table 2, DistilLock incurs the lowest overhead, requiring less than 1.2% of total FLOPs, compared to 3–16% for DarkneTZ, ShadowNet, and Serdab. The efficiency gain comes from reducing permutation matrix multiplications to simple column shuffling and pre-computing the OTP decryption factor offline. These results show that DistilLock delivers stronger protection while imposing significantly lower efficiency costs.

Table 2: TEE computation overhead across defense schemes and models. DistilLock requires the lowest FLOPs inside the TEE.

Models	Total FLOPs	DarkneTZ-TEE	ShadowNet-TEE	Serdab-TEE	**DistilLock-TEE**
LLaMA3.2-3B	11,913,901,394,688	427,016,202,048	1,469,885,448,192	425,440,192,320	52,539,949,056
	% of total	3.58%	12.34%	3.57%	0.44%
Qwen2.5-1.5B	5,613,429,499,648	199,157,061,696	920,602,437,942	260,459,698,240	62,236,131,328
	% of total	3.55%	16.40%	4.64%	1.11%

5 Related Work

TEE-assisted Model Execution. Serdab [20] protects shallow transformer layers, which are considered more critical. ShadowNet [22] obfuscates all linear layers via matrix transformations and offloads them, along with the remaining layers, to untrusted hardware, but remains vulnerable to model-stealing attacks [35]. DarkneTZ [21] secures the final transformer layers and the output classifier inside the TEE. TransLinkGuard [19] applies a permutation-based obfuscation strategy to protect model weights, but requires TEE authorization at each transformer block, introducing substantial communication overhead. In contrast, our method requires only a single TEE authorization at the beginning of the forward pass. More importantly, these methods focus on protecting models during inference, whereas our work targets protection during the fine-tuning process.

Private Knowledge Distillation. RONA [36] assumes the provider has trained a foundation model on both sensitive and public data. Distillation is performed only on public data to train a compact student model whose representations approximate those of the foundation model. The student is deployed to users, while the provider retains both the foundation model and the sensitive data. To ensure formal guarantees, RONA perturbs the distilled knowledge to satisfy differential privacy. Swing Distillation [37] assumes disjoint teacher and student datasets and introduces dynamic temperature and soft target protection to reduce leakage of private information. However, these approaches primarily protect the provider's sensitive training data in the cloud, whereas our solution focuses on protecting user data by performing fine-tuning directly on local devices.

Efficient On-device Fine-tuning. Offsite-Tuning [8] enables lightweight adapter fine-tuning on downstream tasks, assisted by a lossy compressed emulator sent from the provider to the user. The fine-tuned adapter is then returned and integrated into the full model on the provider's side. This workflow, however, exposes the user's fine-tuned adapters to the provider and lacks formal privacy guarantees. PockEngine [6] and MobiZO [7] focus on memory-efficient on-device training. PockEngine employs a first-order optimizer but restricts updates to a subset of layers to reduce activation storage. MobiZO instead uses a zeroth-order optimizer, eliminating backward propagation and relying solely on inference engines to estimate gradients. However, neither approach addresses model ownership concerns, which our method explicitly considers.

6 Conclusion

We introduced DistilLock, a TEE-assisted fine-tuning framework that safeguards both user data privacy and model intellectual property during on-device knowledge distillation. By combining lightweight TEE authorization with model obfuscation, DistilLock prevents unauthorized distillation and model-stealing attacks while keeping computational overhead minimal. Our experiments demonstrate that DistilLock achieves secure and efficient edge-based LLM personalization, outperforming prior defenses in both robustness and efficiency.

Acknowledgment

We sincerely thank all the reviewers for their time and constructive comments. This material is based upon work supported by NSF award number 2224319, REAL@USC-Meta center, and VMware gift. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the U.S. Government.

References

[1] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, and et. al. Gpt-4 technical report, 2024.

[2] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul R. Barham, Tom Hennigan, and et. al. Gemini: A family of highly capable multimodal models, 2025.

[3] Anthropic. The Claude 3 Model Family: Opus, Sonnet, Haiku, 2024.

[4] Qualcomm. Qualcomm Hexagon NPU. `https://www.qualcomm.com/products/technology/processors/hexagon`. Accessed: August 31, 2025.

[5] NVIDIA. Embedded Systems Developer Kits & Modules from NVIDIA Jetson. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/`. Accessed: August 31, 2025.

[6] Ligeng Zhu, Lanxiang Hu, Ji Lin, Wei-Chen Wang, Wei-Ming Chen, and Song Han. Pockengine: Sparse and efficient fine-tuning in a pocket. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

[7] Lei Gao, Amir Ziashahabi, Yue Niu, Salman Avestimehr, and Murali Annavaram. MobiZO: Enabling efficient llm fine-tuning at the edge via inference engines. In *The 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

[8] Guangxuan Xiao, Ji Lin, and Song Han. Offsite-tuning: Transfer learning without full model, 2023.

[9] Xicong Shen, Yang Liu, Huiqi Liu, Jue Hong, Bing Duan, Zirui Huang, Yunlong Mao, Ye Wu, and Di Wu. A split-and-privatize framework for large language model fine-tuning, 2023.

[10] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[11] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel tdx demystified: A top-down approach, 2023.

[12] Tiago Alves. TrustZone: Integrated hardware and software security. *Information Quarterly*, 3:18–24, 2004.

[13] NVIDIA. NVIDIA H100 GPU Whitepaper. `https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c`. Accessed: August 31, 2025.

[14] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[15] Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. Distillm: towards streamlined distillation for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[16] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024.

[17] Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. Copy, right? a testing framework for copyright protection of deep learning models. In *2022 IEEE symposium on security and privacy (SP)*, pages 824–841. IEEE, 2022.

[18] Yufei Chen, Chao Shen, Cong Wang, and Yang Zhang. Teacher model fingerprinting attacks against transfer learning. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3593–3610, 2022.

[19] Qinfeng Li, Zhiqiang Shen, Zhenghan Qin, Yangfan Xie, Xuhong Zhang, Tianyu Du, Sheng Cheng, Xun Wang, and Jianwei Yin. Translinkguard: Safeguarding transformer models against model stealing in edge deployment. In *Proceedings of the 32nd ACM International Conference on Multimedia*, MM '24, page 3479–3488. ACM, October 2024.

[20] Tarek Elgamal and Klara Nahrstedt. Serdab: An IoT framework for partitioning neural networks computation across multiple enclaves. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 519–528. IEEE, 2020.

[21] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. DarkneTZ: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.

[22] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. ShadowNet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1596–1612. IEEE, 2023.

[23] Mu Yuan, Lan Zhang, and Xiang-Yang Li. Secure transformer inference protocol, 2024.

[24] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models, 2024.

[25] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report, 2023.

[26] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[27] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence?, 2019.

[28] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[29] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018.

[30] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[31] Adam Poliak. A survey on recognizing textual entailment as an NLP evaluation, 2020.

[32] Dorottya Demszky, Kelvin Guu, and Percy Liang. Transforming question answering datasets into natural language inference datasets, 2018.

[33] Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. Natural language understanding with the quora question pairs dataset, 2019.

[34] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions, 2019.

[35] Ziqi Zhang, Chen Gong, Yifeng Cai, Yuanyuan Yuan, Bingyan Liu, Ding Li, Yao Guo, and Xiangqun Chen. 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)*, pages 52–52. IEEE Computer Society, 2023.

[36] Ji Wang, Weidong Bao, Lichao Sun, Xiaomin Zhu, Bokai Cao, and Philip S. Yu. Private model compression via knowledge distillation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[37] Junzhuo Li, Xinwei Wu, Weilong Dong, Shuangzhi Wu, Chao Bian, and Deyi Xiong. Swing distillation: A privacy-preserving knowledge distillation framework, 2022.

A Limitations

While DistilLock demonstrates strong protection against unauthorized knowledge distillation and model-stealing attacks, several limitations remain.

First, our current implementation is built on Intel SGX and has not been extended to other trusted execution environments such as ARM TrustZone or emerging GPU TEEs. Second, although obfuscation reduces FLOPs by simplifying permutation operations, it still introduces extra steps such as OTP application and permutation management, and future adaptive attacks (e.g., side-channel leakage or advanced surrogate training) may weaken its guarantees. Finally, our evaluation is limited to standard KD settings and datasets, leaving the effectiveness of DistilLock under advanced distillation variants and more complex scenarios as an open question.

B LayerNorm and RMSNorm Proof

LayerNorm. Layer Normalization normalizes activations across the feature dimension of each sample to stabilize training and reduce covariate shift. It has been widely adopted in early transformer architectures such as BERT and GPT-2.

RMSNorm. Recent LLMs such as LLaMA and Qwen adopt Root Mean Square Normalization (RMSNorm), a simplified alternative to LayerNorm. RMSNorm removes mean-centering since the residual connection dominates, and instead normalizes only by the root mean square of the activations. This reduces computation overhead while maintaining stability.

In both LayerNorm and RMSNorm, the statistics (mean, variance, or RMS) are permutation-invariant, but the feature-wise scaling and shifting parameters must be permuted consistently with the activations. Otherwise, functional equivalence is broken. We show that both normalization layers are *permutation-equivariant* when the inputs and their parameters (γ, β) are permuted by the same permutation matrix π.

Proof of LayerNorm. We aim to prove that

$$\text{LayerNorm}(x\pi; \gamma\pi, \beta\pi) = \text{LayerNorm}(x; \gamma, \beta)\pi.$$

The LayerNorm function is defined for $x \in \mathbb{R}^{n\times d}$ by

$$\text{LayerNorm}(x; \gamma, \beta) = \gamma \circ \frac{x - \mu_x}{\sigma_x} + \beta,$$

where $\mu_x, \sigma_x \in \mathbb{R}^n$ are the row-wise mean and standard deviation, and $\circ$ denotes the Hadamard (element-wise) product.

Since μ_x and σ_x are computed row-wise, they are invariant under column permutations, i.e., $\mu_{x\pi} = \mu_x$ and $\sigma_{x\pi} = \sigma_x$. Thus,

$$\text{LayerNorm}(x\pi; \gamma\pi, \beta\pi) = \gamma\pi \circ \frac{x\pi - \mu_x}{\sigma_x} + \beta\pi = \big(\gamma \circ \tfrac{x-\mu_x}{\sigma_x} + \beta\big)\pi = \text{LayerNorm}(x; \gamma, \beta)\pi.$$

Proof of RMSNorm. We aim to prove that

$$\text{RMSNorm}(x\pi; \gamma\pi) = \text{RMSNorm}(x; \gamma)\pi.$$

The RMSNorm function is defined for $x \in \mathbb{R}^{n\times d}$ by

$$\text{RMSNorm}(x; \gamma) = \gamma \circ \frac{x}{\sqrt{\frac{1}{d}\sum_{j=1}^{d} x_j^2}},$$

where the denominator is the root mean square computed over the feature dimension.

Since $\sum_{j=1}^{d}(x\pi)_j^2 = \sum_{j=1}^{d} x_j^2$, the RMS value is permutation-invariant. Therefore,

$$\text{RMSNorm}(x\pi; \gamma\pi) = \gamma\pi \circ \frac{x\pi}{\sqrt{\frac{1}{d}\sum_{j=1}^{d}(x\pi)_j^2}} = \big(\gamma \circ \tfrac{x}{\sqrt{\frac{1}{d}\sum_{j=1}^{d} x_j^2}}\big)\pi = \text{RMSNorm}(x; \gamma)\pi.$$

Thus, both LayerNorm and RMSNorm are permutation-equivariant.

C Experimental Setup Details

For knowledge distillation, we used the Alpaca dataset and implemented training with PyTorch's Torchtune API. We conducted two separate distillation cases: (i) LLaMA-3.1-8B distilled into LLaMA-3.2-1B, and (ii) Qwen2.5-1.5B distilled into Qwen2.5-0.5B. Initial obfuscation and authorization of the teacher models were performed on an Intel SGX enclave, after which training and evaluation of DistilLock were carried out on an NVIDIA A100 (40GB) GPU. For the LLaMA case, we used the Adam optimizer with a learning rate of 1×10^{-4}, a KD ratio of 1.0, a batch size of 4, and gradient accumulation steps of 8. For the Qwen2 case, we used the Adam optimizer with a learning rate of 3×10^{-4}, a KD ratio of 0.5, a batch size of 8, and gradient accumulation steps of 8. Each experiment was repeated under three different random seeds. The total GPU cost for training and evaluation across both cases was approximately 70 GPU hours.

For adversarial training simulations, we modeled varying attacker capabilities. The white-box attack served as the best-case scenario for the adversary, assuming full access to both architecture and pre-trained weights. In this case, the attacker simply duplicated the original model and fine-tuned it on a separate labeled dataset, yielding the highest possible attack accuracy. The black-box attack represented the worst-case scenario, where the attacker was assumed to know only the architecture but not the weights. Hence, the surrogate model was initialized with random weights and trained from scratch on the attacker's dataset.

For partial-defense baselines, we followed prior work to configure different levels of model parameter exposure [19]. In Serdab, the first Transformer block was protected inside the TEE and thus randomly initialized, while all remaining layers were accessible to the attacker. DarkneTZ instead protected the final Transformer block and classifier, which were randomly initialized, leaving earlier layers exposed. ShadowNet obfuscated all linear layers via matrix transformations; following prior work [35], we simulated the attacker by decoding these obfuscated layers to obtain surrogate initialization. For DistilLock, the surrogate model was initialized directly with obfuscated weights and then trained on the attacker's labeled dataset. In all simulations, we sampled 1000 training and 1000 testing examples per task, following the assumption in prior work [19] that the attacker has access to only a small portion of the foundation model's training data. The total GPU time consumed for these simulations was approximately 80 hours.