DistilLock: Safeguarding LLMs from Unauthorized Knowledge Distillation on the Edge

Anonymous ACL submission

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 and model-stealing attacks while maintaining high computational efficiency, offering a secure and practical solution for edge-based LLM personalization.

1 Introduction

Proprietary Large Language Models (LLMs) such as GPT (OpenAI et al., 2024), Gemini (Team et al., 2025), and Claude (Anthropic, 2024) 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 valuable intellectual property and are typically not released publicly. Downstream users who wish to customize these models must usually 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, especially for sensitive or proprietary datasets, as illustrated in Figure 1(a).

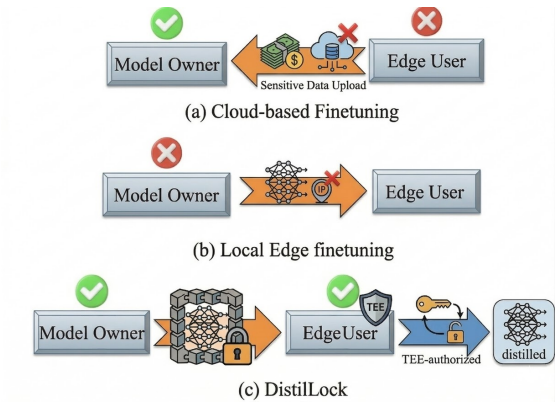

Figure 1: Motivation for DistilLock. (a) Cloud-based fine-tuning exposes sensitive user data and incurs extra API costs. (b) Fully local fine-tuning protects data privacy but exposes proprietary model weights to an untrusted edge device. (c) DistilLock enables secure on-device distillation through TEE-based authorization, protecting both user data and model IP.

The growing availability of edge-centric ML accelerators (Qualcomm, 2025; NVIDIA, 2025) and resource-efficient training algorithms (Zhu et al., 2023; Gao et al., 2025) has made it increasingly feasible to fine-tune LLMs directly on edge devices. This shift offers a strong privacy advantage because user data can remain entirely local. However, it also introduces a new challenge: the model owner must send proprietary foundation models to an untrusted edge environment, thereby exposing sensitive model parameters and creating IP risks, as shown in Figure 1(b). In particular, an adversarial user may attempt to extract the model or use it for unauthorized knowledge distillation.

Recent works have explored partial solutions to

this dilemma. Offsite-Tuning (Xiao et al., 2023) 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 (Shen et al., 2023) divides the model into two 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 methods reduce data exposure, they still rely on remote model-owner participation and may sacrifice utility due to compression or privacy mechanisms. More importantly, they do not directly address the problem of protecting a proprietary teacher model during fully local distillation.

To overcome these limitations, we introduce DistilLock, a secure on-device knowledge distillation framework that protects both user data privacy and model confidentiality. DistilLock uses a Trusted Execution Environment (TEE) to authorize execution of a proprietary foundation model as a black-box teacher on the user's device. To avoid the high cost of running the full model inside the TEE, DistilLock employs model obfuscation so that most compute-intensive operations can be offloaded to untrusted accelerators while still preventing direct recovery of useful model internals. In this way, authorized users can perform local distillation with correct teacher outputs, while unauthorized users cannot obtain meaningful supervision signals. Figure 1(c) shows this design. Our main contributions are summarized as follows:

- We identify secure edge-based knowledge distillation as a joint privacy and intellectual-property problem, where user data should remain local while proprietary teacher models must remain protected.
- We propose DistilLock, a TEE-assisted framework that combines lightweight authorization with model obfuscation to enable efficient and secure on-device distillation.
- We show experimentally that DistilLock blocks unauthorized distillation, reduces model-stealing attack success, and introduces only small trusted-computing overhead.

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 (McKeen et al., 2013), Intel TDX (Cheng et al., 2023), and ARM TrustZone (Alves, 2004). 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 (NVIDIA, 2022), 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 (Gu et al., 2024; Ko et al., 2024; Agarwal et al., 2024), 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 the model itself.

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 (Chen et al., 2022a,b) 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 Design of DistilLock

3.1 Method Overview

A straightforward approach to protect a proprietary teacher model against an adversary would be to execute the full model inside a TEE. However, TEEs typically lack support for powerful accelerators such as GPUs and are therefore much slower for LLM execution, with prior work reporting overheads on the order of tens of times compared to normal execution (Li et al., 2024). Existing partial TEE-shielded approaches therefore place only part of the model inside the TEE and offload the rest to an untrusted environment (Elgamal and Nahrstedt, 2020; Mo et al., 2020; Sun et al., 2023). While this improves efficiency, it often leaves enough model structure and parameters exposed to enable effective model extraction.

DistilLock addresses this challenge by separating authorization from heavy computation. Instead of using the TEE to execute the full teacher model, DistilLock uses the TEE only to authorize valid executions and apply the secret transformations that make the obfuscated teacher functional. The bulk of the matrix multiplications still runs on an untrusted accelerator. This design has two goals: (1) authorized users can obtain the correct teacher logits needed for local knowledge distillation, and (2) unauthorized users who only access the obfuscated model outside the TEE cannot recover meaningful teacher behavior.

DistilLock operates in two phases: *model obfuscation* before deployment and *model authorization* at runtime. In the model obfuscation phase, the model owner transforms the proprietary teacher using secret permutation matrices and deploys only the obfuscated model to the edge device. In the model authorization phase, the TEE applies the corresponding secret transformations to the input representation so that the obfuscated teacher produces correct logits only for authorized executions. Because the authorized hidden representation preserves the same permutation structure across layers, most subsequent computation can be carried out on the GPU without repeated TEE intervention.

3.2 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 (Grattafiori et al., 2024) and Qwen (Bai et al., 2023) as follows:

$$\begin{aligned} &Q = xW_q, \quad K = xW_k, \quad V = xW_v, \\ &u = \mathrm{Softmax}(QK^T/\sqrt{k})VW_o, \\ &v = \mathrm{LayerNorm}(u + x; \gamma_1, \beta_1), \\ &z = \big(\mathrm{SiLU}(vW_1) \odot (vW_3)\big)W_2, \\ &y = \mathrm{LayerNorm}(z + v; \gamma_2, \beta_2), \end{aligned} \tag{1}$$

where $W_q, W_k, W_v, W_o \in \mathbb{R}^{d\times d}$, $W_1, W_3 \in \mathbb{R}^{d\times f}$, $W_2 \in \mathbb{R}^{f\times d}$, and $\gamma_1, \beta_1, \gamma_2, \beta_2 \in \mathbb{R}^d$.

The main idea of model obfuscation is to preserve model functionality under a secret permutation while hiding the original parameter semantics from the untrusted environment. Let $\pi_{emb} \in \{0,1\}^{vocab\times vocab}$ and $\pi \in \{0,1\}^{d\times d}$ be permutation matrices. Before the model is transferred to the user side, the model owner transforms the parameters as

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

3.3 Model Authorization

The obfuscated model and the encrypted permutation matrix π are delivered to the user, who then instantiates the TEE on the local device. As shown in Figure 2, the user input is represented as one-hot

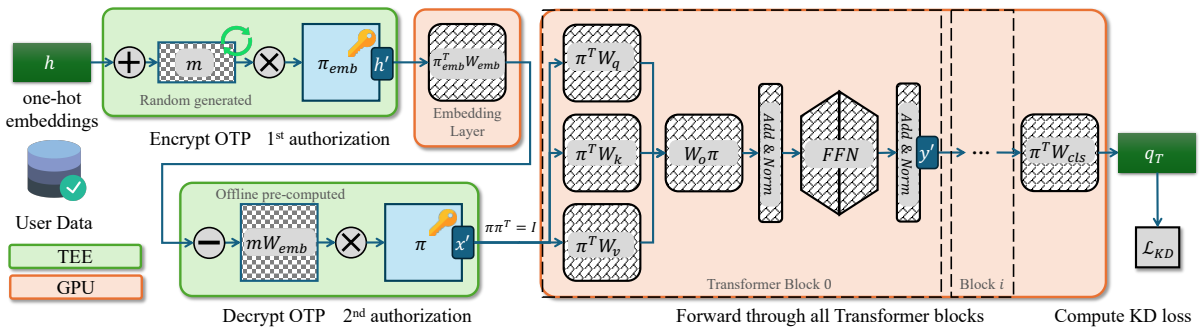

Figure 2: Overview of DistilLock. The proprietary teacher model is obfuscated before deployment and executed with TEE authorization on the user's device. The TEE applies the correct permutation and one-time pad to the input, while compute-intensive operations run on untrusted accelerators using obfuscated weights. This design preserves model functionality for authorized knowledge distillation while preventing unauthorized access or model extraction.

vectors $h \in \mathbb{R}^{n \times vocab}$ and first sent to the TEE. The TEE does not execute the full teacher model. Instead, it performs a lightweight authorization step that converts the input into the feature order expected by the obfuscated model.

Inside the TEE, we first sample a one-time pad (OTP) $m \in \mathbb{R}^{n \times vocab}$ and construct the protected input $h' = (h + m)\pi_{emb}$. Here, the OTP is a random mask added to the input for each execution, so the raw input is hidden from the untrusted environment. Then, the permutation matrix π_{emb} shuffles the columns of h.

The protected input is then processed by the obfuscated embedding matrix:

$$\begin{aligned} h'W'_{emb} &= (h+m)\pi_{emb} \cdot \pi_{emb}^T W_{emb} \\ &= (h+m)W_{emb}, \end{aligned} \tag{3}$$

where we use $\pi_{emb}\pi_{emb}^T = I$.

The resulting feature is sent back to the TEE, where the OTP contribution is removed and the hidden-dimension permutation is applied:

$$\begin{aligned} x' &= \big((h+m)W_{emb} - mW_{emb}\big)\pi \\ &= hW_{emb}\pi = x\pi. \end{aligned} \tag{4}$$

Therefore, the TEE converts the original embedding output x into the authorized representation $x\pi$, which is exactly the input format expected by the obfuscated Transformer layers. Once this authorized hidden state is constructed, the remaining Transformer computation can proceed on the GPU using the obfuscated model. Specifically, the functionality of each subsequent Transformer block is preserved with the obfuscated parameters:

$$\begin{aligned} Q' &= x'W'_q = x\pi\pi^T W_q = Q, \\ K' &= x'W'_k = x\pi\pi^T W_k = K, \\ V' &= x'W'_v = x\pi\pi^T W_v = V, \\ u' &= \mathrm{Softmax}(QK^T/\sqrt{k})V'W'_o \\ &= \mathrm{Softmax}(QK^T/\sqrt{k})VW_o\pi = u\pi, \\ v' &= \mathrm{LayerNorm}(u' + x\pi; \gamma_1\pi, \beta_1\pi) \\ &= \mathrm{LayerNorm}(u\pi + x\pi; \gamma_1\pi, \beta_1\pi) = v\pi, \\ z' &= \big(\mathrm{SiLU}(v'W'_1) \odot (v'W'_3)\big)W'_2 \\ &= \big(\mathrm{SiLU}(v\pi\pi^T W_1) \odot (v\pi\pi^T W_3)\big)W_2\pi = z\pi, \\ y' &= \mathrm{LayerNorm}(z' + v'; \gamma_2\pi, \beta_2\pi) \\ &= \mathrm{LayerNorm}(z\pi + v\pi; \gamma_2\pi, \beta_2\pi) = y\pi. \end{aligned} \tag{5}$$

Therefore, once the first authorized hidden state is constructed, the same permutation π is preserved across all subsequent decoder layers. This is important for efficiency: unlike prior designs that may require repeated secure intervention across layers, DistilLock only needs the TEE to bootstrap the correct hidden representation at the beginning. The detailed proof of the LayerNorm operation is provided in Appendix A.

The output y' of one Transformer block becomes the input of the next block. At the last Transformer block, the output y'_{final} is multiplied with the final linear classifier W'_{cls} to produce the correct logits:

$$q_T = y'_{final}W'_{cls} = y_{final}\pi\pi^T W_{cls} = y_{final}W_{cls}. \tag{6}$$

This shows that DistilLock preserves the exact teacher prediction under authorized execution. In contrast, without TEE authorization, the attacker cannot construct the correct $x\pi$ representation. The

Algorithm 1 DistilLock Authorization and Execution on the Edge

Require: input h, OTP m, permutation matrices π_{emb}, π, obfuscated model $\mathcal{M}'$
Ensure: teacher logits q_T

1: **// TEE authorization**
2: Construct protected input: $h' \leftarrow (h + m)\pi_{emb}$
3: Compute protected embedding on GPU: $\tilde{x} \leftarrow h' W'_{emb}$
4: Remove OTP and authorize hidden state in TEE: $x' \leftarrow (\tilde{x} - mW_{emb})\pi$
5: **// Obfuscated teacher execution on GPU**
6: Run obfuscated Transformer blocks with input x' to obtain y'_{final}
7: Recover teacher logits: $q_T \leftarrow y'_{final} W'_{cls}$
8: **return** q_T

obfuscated teacher may still be executable at the operator level, but it will not produce the correct logits required for effective knowledge distillation. In this sense, DistilLock transforms the proprietary teacher into a *locked* model: its utility depends on enclave-controlled authorization.

Since the permutation matrix π is maintained throughout the hidden states of all decoder layers, TEE computation is only required for the initial word embedding stage and authorization of the first valid hidden representation. All remaining operations, including teacher forward computation, loss evaluation, and the student model's forward and backward passes, proceed as in standard knowledge distillation on the GPU. This design is what allows DistilLock to simultaneously preserve efficiency and strengthen model-IP protection.

3.4 Security Analysis

Our security analysis focuses on a model-stealing threat model, where an adversary attempts to extract the teacher model's functionality by exploiting query responses or by training a surrogate model. One possible attack is to recover the original parameters by guessing the secret permutation matrix π. However, the probability of recovering the correct permutation is approximately $\frac{1}{d!}$, where d is the hidden dimension of the model. For foundation LLMs, d is typically in the thousands, making brute-force recovery computationally infeasible.

An attacker may also try to bypass DistilLock by approximating the authorization process itself. In our design, however, authorization is tied to both the secret permutation and the OTP applied inside the TEE. Because the OTP is freshly sampled for each execution, the same input does not lead to a fixed observable authorization pattern outside the TEE. This prevents the attacker from directly learning a deterministic mapping that can replace the enclave-side authorization step.

In addition, since the regular word embedding layer is normally a lightweight look-up operation, DistilLock introduces extra cost when the protected input is processed as $(h + m)W_{emb}$. To reduce this overhead, we restrict the OTP noise m to be a random sparse k-hot vector with the number of hot entries satisfying $k \geq \frac{h}{\log_2(h)}$. This reduces the embedding computation from a dense matrix multiplication to an efficient vector summation while still maintaining entropy comparable to a uniform random vector. As a result, DistilLock preserves the security benefit of OTP-based protection while keeping the added authorization overhead small.

4 Experiments

We evaluate DistilLock to answer three research questions: (1) Can DistilLock prevent unauthorized knowledge distillation when the adversary has access only to the obfuscated teacher model but not the TEE authorization? (2) Can DistilLock defend against model-stealing attacks in which the adversary trains a surrogate model under the threat model described in Section 2? (3) Does DistilLock introduce significant TEE-side overhead?

4.1 Distillation Lockdown

We first evaluate whether DistilLock can lock the knowledge distillation process without model authorization. We consider two settings: (i) an *authorized* setting, where the user performs local knowledge distillation with the correct TEE authorization, and (ii) an *unauthorized* setting, where the user has access only to the obfuscated teacher model and does not apply the correct transformations π_{emb} and π before execution. In the latter case, the teacher remains executable at the operator level but should not provide valid supervision signals.

We study two teacher–student pairs: LLaMA-3.1-8B → LLaMA-3.2-1B and Qwen2.5-1.5B → Qwen2.5-0.5B (Grattafiori et al., 2024; Bai et al., 2023). For distillation training, we use the Alpaca dataset (Taori et al., 2023) and implement the pipeline with Torchtune APIs (torchtune, 2024). The student models are trained under teacher guidance and evaluated on HellaSwag and Common-

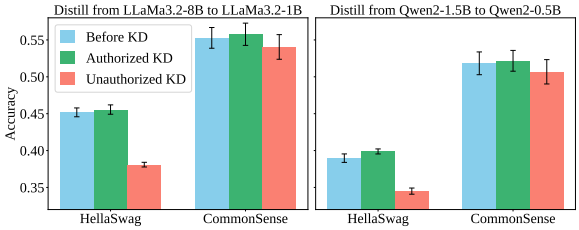

Figure 3: Knowledge distillation results under DistilLock. Authorized distillation improves student accuracy over pre-distillation baselines, while unauthorized distillation collapses performance due to random logits from the obfuscated teacher.

senseQA (Zellers et al., 2019; Talmor et al., 2019). Initial obfuscation and authorization are executed in an Intel SGX enclave, while training and evaluation run on an NVIDIA A100 (40GB) GPU. For the LLaMA case, we use Adam with learning rate 1×10^{-4}, KD ratio 1.0, batch size 4, and gradient accumulation of 8. For the Qwen case, we use Adam with learning rate 3×10^{-4}, KD ratio 0.5, batch size 8, and gradient accumulation of 8. Each experiment is repeated over three random seeds.

Figure 3 shows that DistilLock preserves utility under authorized execution while effectively locking the teacher under unauthorized use. With proper authorization, knowledge distillation improves student accuracy over the pre-distillation baseline. In contrast, without authorization, student performance collapses because the obfuscated teacher produces invalid logits and therefore provides no useful learning signal. These results confirm that DistilLock can selectively enable local distillation only for authorized executions.

4.2 Adversarial Training

We next evaluate DistilLock against model-stealing attacks. Although unauthorized users cannot directly perform valid knowledge distillation, they may still attempt to recover the teacher's functionality by training a surrogate model on downstream tasks. To test this threat, we simulate adversarial training under varying levels of model exposure.

We consider a diverse set of NLP tasks: AI2 Reasoning Challenge (ARC-e, ARC-c) for science reasoning (Clark et al., 2018), HellaSwag for commonsense reasoning (Zellers et al., 2019), BoolQ for yes/no reading comprehension (Clark et al., 2019), and several GLUE tasks (Wang et al., 2018), including SST-2 for sentiment classification, RTE for textual entailment, QNLI for question answering, and QQP for question-pair similarity (Poliak, 2020; Demszky et al., 2018; Sharma et al., 2019). For all adversarial simulations, we follow prior work and assume the attacker has access to only a small portion of the original training distribution. Concretely, we sample 1000 training and 1000 testing examples for each task (Li et al., 2024).

We compare DistilLock against multiple baselines that represent different attacker capabilities and different defense mechanisms. The *white-box* attack assumes full access to both architecture and pre-trained weights, and therefore serves as the strongest attack baseline. The *black-box* attack assumes the attacker knows only the architecture and must train the surrogate from random initialization, providing the weakest attack baseline. We also compare against three partial-defense baselines from prior work (Li et al., 2024): *Serdab*, which protects the first Transformer block inside the TEE and exposes the remaining layers; *DarkneTZ*, which protects the final Transformer block and classifier while exposing earlier layers; and *ShadowNet*, which obfuscates linear layers via matrix transformations while leaving the rest of the model outside the TEE. For ShadowNet, we follow prior work (Zhang et al., 2023) and simulate the attacker by decoding the obfuscated linear layers to obtain surrogate initialization. For DistilLock, the attacker initializes the surrogate directly from the obfuscated weights and fine-tunes on the labeled attack dataset.

Table 1 shows that DistilLock consistently achieves the lowest attack success across tasks and models. On average, DistilLock reduces surrogate accuracy to $0.98\times$ the black-box baseline, effectively removing the advantage of model exposure. In contrast, prior defenses such as ShadowNet ($1.58\times$) and DarkneTZ ($1.41\times$) still leave substantial room for the attacker to recover useful model functionality. These results suggest that DistilLock provides stronger resistance to surrogate extraction than prior partial-shielding approaches.

4.3 Efficiency Cost

Finally, we evaluate the efficiency overhead introduced by DistilLock, with a focus on TEE-side computation. Our implementation performs the initial authorization inside an Intel SGX enclave and offloads the remaining model computation to the GPU. We measure the fraction of total model FLOPs that must be executed inside the TEE and compare DistilLock against prior TEE-assisted defenses.

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×

Table 1: Evaluation of surrogate model attack accuracy under different defense schemes. The last row reports average accuracy relative to the black-box baseline. Among the defense schemes, 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.

As reported in Table 2, DistilLock incurs the lowest TEE overhead among all compared methods. The main reason is that DistilLock requires enclave computation only during the initial authorization step, after which the same hidden-state permutation is preserved across all subsequent layers. This avoids repeated secure intervention throughout the network. In addition, the OTP-related embedding processing is implemented efficiently, so the added trusted-computing cost remains small relative to the full teacher forward pass. These results show that DistilLock improves security without sacrificing practicality for on-device knowledge distillation.

4.4 Ablation on OTP Secure Guarantee

DistilLock uses a sparse k-hot OTP to protect the input during TEE authorization, with $k \geq \frac{V}{\log_2(V)}$ (where V is the vocabulary size) to balance security and efficiency. A denser OTP increases the entropy of the masked input but also raises the cost of the embedding computation, since $(h+m)W_{emb}$ requires summing more rows of the embedding matrix. We ablate this tradeoff by varying k from minimal (k=1) to our default threshold to fully dense. Using LLaMA3.2-3B as the teacher, we apply the same surrogate attack pipeline under each sparsity level and report both attack accuracy and relative authorization cost on a subset of tasks. As shown in Table 3, extremely sparse OTP (k=1) leads to noticeably higher surrogate attack accuracy, as the low-entropy mask makes it easier for the attacker to approximate the authorization mapping. Once k reaches the default threshold $V/\log_2(V)$, security saturates and matches the fully dense setting, while authorization cost remains orders of magnitude lower. This confirms that the chosen threshold

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%

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

OTP hot entries	$k{=}1$	$k{=}\frac{V}{\log_2 V}$	$k{=}V$
SST-2	88.91%	63.17%	63.09%
ARC-e	40.45%	25.32%	25.27%
HellaSwag	37.80%	24.90%	25.00%
BoolQ	77.40%	65.00%	64.90%
Relative auth. cost	1×	∼7.5k×	∼128k×

Table 3: OTP sparsity ablation on LLaMA3.2-3B. Sparse OTP weakens security, whereas DistilLock's default threshold matches fully dense protection at far lower cost.

provides a practical sweet spot between security and efficiency.

5 Related Work

TEE-assisted Model Execution. Serdab (Elgamal and Nahrstedt, 2020) protects shallow transformer layers, which are considered more critical. ShadowNet (Sun et al., 2023) 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 (Zhang et al., 2023). DarkneTZ (Mo et al., 2020) secures the final transformer layers and the output classifier inside the TEE. TransLinkGuard (Li et al., 2024) 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 (Wang et al., 2019) 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 (Li et al., 2022) 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 LLM Fine-tuning. Offsite-Tuning (Xiao et al., 2023) 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 (Zhu et al., 2023) and MobiZO (Gao et al., 2025) 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 maintaining a minimal computational overhead. Our experiments demonstrate that DistilLock achieves secure and efficient edge-based LLM personalization, outperforming prior defenses in robustness and efficiency.

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 through lightweight 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.

References

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

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

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

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *Preprint*, arXiv:2309.16609.

Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. 2022a. 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.

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

Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2023. Intel tdx demystified: A top-down approach. *Preprint*, arXiv:2303.15540.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *Preprint*, arXiv:1905.10044.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Dorottya Demszky, Kelvin Guu, and Percy Liang. 2018. Transforming question answering datasets into natural language inference datasets. *Preprint*, arXiv:1809.02922.

Tarek Elgamal and Klara Nahrstedt. 2020. 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.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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

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

Junzhuo Li, Xinwei Wu, Weilong Dong, Shuangzhi Wu, Chao Bian, and Deyi Xiong. 2022. Swing distillation: A privacy-preserving knowledge distillation framework. *Preprint*, arXiv:2212.08349.

Qinfeng Li, Zhiqiang Shen, Zhenghan Qin, Yangfan Xie, Xuhong Zhang, Tianyu Du, Sheng Cheng, Xun Wang, and Jianwei Yin. 2024. 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.

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

Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. 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.

NVIDIA. 2022. Nvidia h100 tensor core gpu architecture overview. Accessed: August 31, 2025.

NVIDIA. 2025. Embedded Systems Developer Kits & Modules from NVIDIA Jetson. Accessed: August 31, 2025.

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, and 2 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Adam Poliak. 2020. A survey on recognizing textual entailment as an NLP evaluation. *Preprint*, arXiv:2010.03061.

Qualcomm. 2025. Qualcomm Hexagon NPU. Accessed: August 31, 2025.

Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. 2019. Natural language understanding with the quora question pairs dataset. *Preprint*, arXiv:1907.01041.

Xicong Shen, Yang Liu, Huiqi Liu, Jue Hong, Bing Duan, Zirui Huang, Yunlong Mao, Ye Wu, and Di Wu. 2023. A split-and-privatize framework for large language model fine-tuning. *Preprint*, arXiv:2312.15603.

Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. 2023. 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.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. 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*.

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

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, and 6 others. 2025. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

torchtune. 2024. torchtune: Pytorch's finetuning library.

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

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

Guangxuan Xiao, Ji Lin, and Song Han. 2023. Offsite-tuning: Transfer learning without full model. *Preprint*, arXiv:2302.04870.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Ziqi Zhang, Chen Gong, Yifeng Cai, Yuanyuan Yuan, Bingyan Liu, Ding Li, Yao Guo, and Xiangqun Chen. 2023. 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.

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

A 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

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

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

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

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,

$$\mathrm{LayerNorm}(x\pi; \gamma\pi, \beta\pi) = \gamma\pi \circ \frac{x\pi - \mu_x}{\sigma_x} + \beta\pi$$
$$= \left(\gamma \circ \frac{x - \mu_x}{\sigma_x} + \beta\right)\pi = \mathrm{LayerNorm}(x; \gamma, \beta)\pi.$$

Proof of RMSNorm. We aim to prove that

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

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

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

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,

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

Thus, both LayerNorm and RMSNorm are permutation-equivariant.