LLA: Enhancing Security and Privacy for Generative Models with L̲ogic-L̲ocked A̲ccelerators

You Li[*], **Guannan Zhao**[*], **Yuhao Ju, Yunqi He, Jie Gu, Hai Zhou**

Northwestern University, Evanston, IL, USA

{you.li, gnzhao, yuhaoju2017, yunqi.he}@u.northwestern.edu, {jgu, haizhou}@northwestern.edu

Abstract

We introduce LLA, an effective intellectual property (IP) protection scheme for generative AI models. LLA leverages the synergy between hardware and software to defend against various supply chain threats, including model theft, model corruption, and information leakage. On the software side, it embeds key bits into neurons that can trigger outliers to degrade performance and applies invariance transformations to obscure the key values. On the hardware side, it integrates a lightweight locking module into the AI accelerator while maintaining compatibility with various dataflow patterns and toolchains. An accelerator with a pre-stored secret key acts as a license to access the model services provided by the IP owner. The evaluation results show that LLA can withstand a broad range of oracle-guided key optimization attacks, while incurring a minimal computational overhead of less than 0.1% for 7,168 key bits.

1 Introduction

Generative AI (GenAI) is a revolutionary technology that automatically creates a variety of content, including text, images, videos, and audio, in response to users' input. Training generative models requires massive data, substantial training resources, and specialized expertise. As a result, new business models have emerged to allow clients with limited resources to access GenAI capabilities. For instance, Inference-as-a-Service (*IaaS*) refers to a deployment approach in which generative models are hosted on cloud platforms and accessed through APIs. In contrast, *self-hosting* enables organizations to deploy models internally, offering greater control over confidential information. In such situations, model parameters are directly exposed to various participants in the *supply chain*, including cloud service providers, network operators, and end-users. Unauthorized use or distribution of proprietary models can cause significant financial losses to intellectual property (IP) owners. Moreover, unrestricted access to model parameters poses serious security risks. For example, adversarial prompts can cause the generation of harmful or misleading content (Zhuang, Zhang, and Liu 2023; Zou et al. 2023), while backdoor attacks can manipulate generated outputs through

malicious triggers embedded in compromised models (Zhao et al. 2023; Chou, Chen, and Ho 2023).

Model locking leverages the principle of *hardware root-of-trust* to defend against model theft and various supply chain threats. It inserts key-controlled protection units into the neural network, with the secret key stored on hardware in a tamper-proof module (Nicholas et al. 2021; Chakraborty and Bhunia 2009; Kamali et al. 2018). Without knowing the correct key, an adversary cannot restore the original functionality of the model. The main advantages of model locking are three-fold. Firstly, it is a proactive approach that offers guaranteed security. In contrast, reactive approaches such as watermarking (Kirchenbauer et al. 2023) and fingerprinting (Xu et al. 2024) can provide evidence of model ownership but cannot prevent unauthorized use of the model. Secondly, model locking introduces minimal computational and hardware overhead. Other proactive approaches, such as TEE-based execution (Mo et al. 2020), parameter encryption (Lin et al. 2020), and memory encryption (Zuo et al. 2021), provide high levels of security but come with significant computational or hardware costs. Lastly, model locking assumes a general and realistic threat model, allowing the architecture and all model parameters to be publicly released. This allows the IP owner to host the model on a cloud platform or send it to an end-user without compromising security or ownership.

This paper presents LLA, a comprehensive model locking framework for generative models (Fig. 1). LLA achieves effectiveness, robustness, and efficiency simultaneously through an integration of software and hardware components. In the software domain, LLA identifies feature outliers within an FFN module and inserts key bits to manipulate these outliers. As such, it can cause substantial degradation in model performance with a small number of key bits. Moreover, it applies invariant transformations to obfuscate key values, thus thwarting a wide range of oracle-guided attacks at minimal cost. In the hardware domain, LLA embeds a lightweight locking module within systolic array AI accelerators. This module is designed to be fully compatible with existing dataflow patterns and model formats. A pre-activated AI accelerator can serve as a license to access all current and future services offered by the IP owner.

Our main contributions are as follows:

- We propose LLA, the first complete method to apply

---

model locking on large generative models.

• We devise a systematic approach to find and construct critical components within a generative model. Locking these components can significantly impair model functionality while preserving the confidentiality of key values.

• We develop a lightweight solution to enable the execution of LLA-protected models on general AI hardware.

• We conduct comprehensive experiments to evaluate the effectiveness of LLA across a diverse set of generative models, assess its efficiency on AI hardware, and examine its robustness against various attacks.

2 Background and Related Work

IP Protection for AI Models. Researchers have proposed various methods to safeguard the IP of AI models. *Watermarking* embeds hidden information into model parameters (Uchida et al. 2017) or model outputs (Adi et al. 2018) to prevent unauthorized use. Unfortunately, it can only passively verify the ownership of a model after it has been stolen or infringed. *Model encryption* (Zhou et al. 2023; Mu et al. 2024) prevents unauthorized access to model parameters with encryption and obfuscation techniques. However, an encrypted model must be restored to the original state before execution, which incurs high overhead and makes it vulnerable to side channel attacks (Li, Huang, and Zhang 2025). *Model splitting* isolates a subset of sensitive computations and executes them within a Trusted Execution Environment (TEE) of a processor (Khan et al. 2021; Wang et al. 2023). Its drawbacks include hardware cost, memory constraints, and performance overhead. *Homomorphic encryption* can guarantee the security and privacy of AI models (Gilad-Bachrach et al. 2016; Sun et al. 2018). However, due to the extremely high computational cost, it can barely be applied in practice.

Model locking originates from *logic locking* (Kamali et al. 2022), which uses binary key bits to protect the IP of logic circuits. Hardware-protected neural network (HPNN) (Chakraborty, Mondai, and Srivastava 2020) performs model locking in the following steps: *i)* selecting neurons at fixed locations of hidden layers as the *protected neurons*; *ii)* associating a key bit with each protected neuron to control whether to flip the pre-activation value; *iii)* training the model as a function of a predetermined secret key. NN-Lock (Alam et al. 2022; Goldstein et al. 2021) utilizes cryptographic primitives to obfuscate the model parameters. While these methods are highly efficient and robust, they have several limitations in common: *i)* it is not clear how an encrypted model can be executed on general AI hardware; *ii)* they are designed for discriminative models rather than large generative models. GenAI presents unique challenges to model locking, and standard model locking techniques are no longer effective as the model scales. For example, removing 25% of layers from Llama2-13B results in only a slight performance drop (Men et al. 2024), and over 95% of neurons in FFN modules of OPT-175B are inactive during inference (Li et al. 2023).

Transformer Architecture. A decoder-only transformer is a stack of N transformer *blocks*, each consisting of two main modules: a multi-head self-attention module and a feed-forward network (FFN) module (Fig. 2(a)). Let $\mathbf{X} \in \mathbb{R}^{T \times D_\mathrm{m}}$ denote the input matrix of both modules, where T is the number of tokens and D_m is the number of features of each token. Following the self-attention module, the same FFN module is applied identically to each token. The FFN module comprises two linear layers (Fig. 2(b)). The first layer, up, expands a token's feature vector from the model dimension D_m to the intermediate dimension D_ff. Certain architectures incorporate a gated matrix $\mathbf{W}^\mathtt{gate}$ in parallel with the original up-projection matrix $\mathbf{W}^\mathtt{up}$, and the output vectors of the two matrices are subsequently combined through element-wise multiplication. A non-linear activation layer, act, is applied between the two linear layers. Finally, the second layer, down, projects the expanded vector from the intermediate space back to the *hidden state space*.

Outliers in GenAI Models. Outliers refer to abnormally large values within GenAI models (Dettmers et al. 2022; Kovaleva et al. 2021). *Weight outliers* appear in specific columns of the down-projection matrices of the feed-forward network (FFN), $\mathbf{W}^\mathtt{down}$. *Feature outliers* concentrate in the block output vectors and the input vectors of the down-projection matrices, $\mathbf{x}_\mathtt{down}$ (Sun et al. 2024). Both types of outliers are strongly correlated and persistent in that they are almost always located in the same feature dimension across different layers (An et al. 2025). Outliers typically emerge in the second block and gradually vanish in the final blocks of a model. Outliers are disproportionately important to model performance (Yin et al. 2024). According to a study on Llama-7B, suppressing only six top outliers has a greater impact than pruning hundreds of thousands of normal weights (Yu et al. 2024).

Threat Model. Our adversary model is similar to those in previous work on hardware-based model protection. The adversary could be a collusion of malicious end-users, cloud service providers, and network service providers. Its objective is to use the model without the permission of the IP owner or launch a white-box attack against the model. The adversary can *i)* directly access the model architecture and all the model parameters; *ii)* query the *oracle model* through a cloud API or an activated AI hardware; *iii)* obtain a small portion of the training dataset. However, the adversary cannot read or probe the secret key stored in the AI hardware, nor can they run an unauthorized model on the AI hardware. In practice, the secret key can be stored in or derived from a tamper-proof memory (Tuyls et al. 2006), a trusted platform module (TPM) (Nicholas et al. 2021), a physical unclonable function (PUF) (Chakraborty and Bhunia 2009), an FPGA bitstream (Kamali et al. 2018), or a camouflaged circuit layout (Li et al. 2017). The adversary aims to either infer the correct key or restore the functionality of the generative model without knowing the correct key.

3 Locking Methodology

3.1 Overview

LLA is a post-training model locking approach. Unlike existing approaches such as HPNN (Chakraborty, Mondai, and
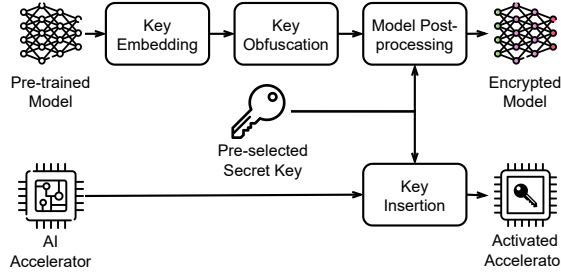
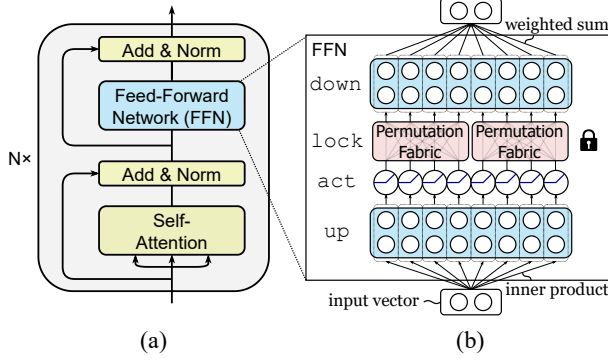Figure 1: Workflow of the LLA model locking framework.

Figure 2: (a) Architecture of a transformer block. (b) Simplified illustration of the proposed locking mechanism. LLA embeds key bits into permutation modules inserted before the down-projection layer, which shuffle intermediate values to alter the model's functionality.

Srivastava 2020), it does not require additional training resources and can be applied to existing models. LLA aims to achieve the following objectives simultaneously:

• *Effectiveness:* A wrong key will substantially degrade the performance of the GenAI model.

• *Efficiency:* The locking scheme introduces a small number of key bits and incurs negligible performance, power and area overhead.

• *Robustness:* An adversary cannot decrypt or bypass the secret key to restore the performance of the model.

• *Hardware friendliness:* The locking scheme can be easily adapted to general AI hardware and requires minimal modifications to both the hardware architecture and the compiler.

The general workflow of the proposed LLA framework is shown in Fig. 1. LLA exploits the synergy of software and hardware to achieve these objectives. At the software level, LLA embeds key bits in the intermediate layer of the FFN module (§3.2). To maximize corruption to model performance, it selects protected neurons by tracking feature outliers of the model. Afterwards, it employs obfuscation techniques dedicated to GenAI models (§3.3) to protect them from oracle-guided attacks and fine-tuning attacks. Finally, LLA adjusts the model parameters according to the key values. The resulting encrypted model can be sent to the user through a public channel and is functional only with a cor-

rect key. At the hardware level (§3.4), LLA attaches locking units to the output of the systolic array of an AI accelerator. With a lightweight control unit, LLA dispatches the protected neurons on the fly and triggers the locking units when they reach the output of the systolic array. As such, it does not depend on specific AI compilers and can be easily integrated with most AI accelerators. An accelerator with the inserted secret key serves as a license to access the service provided by the IP owner.

3.2 Key Embedding

Locking Mechanism. LLA designates a transformer block as the *protected block* and selects a subset of neurons within its FFN module as *protected neurons*. The selected neurons are divided into groups with size m, and the post-activation values within each group are shuffled using a key-controlled *permutation fabric*. Figure 2 illustrates the locking mechanism of LLA, which offers three key advantages. *i)* Localized permutation is agnostic to the input data type and can be efficiently implemented in hardware. Programmable Array Logic (PAL) (Takhar et al. 2022) and embedded FPGA (eFPGA) (Tang et al. 2019) can be configured to implement interconnections, while butterfly networks (Beneš 1964) can realize arbitrary permutations based on control bits. *ii)* Shuffling an FFN's intermediate values can lead to a more substantial degradation in model performance. Notably, FFN modules act as crucial memory elements in GenAI models (Geva et al. 2021), and the same FFN module is applied repeatedly to all tokens in an input sequence. *iii)* Placing all key bits within a single block can enhance resilience against a broad spectrum of attacks. Specifically, the discrete nature and high redundancy of the permutation fabric enhance its resilience to gradient-based key decryption attacks. §4 provides a detailed security analysis of LLA.

Identify Protected Neurons. GenAI models contain billions of parameters, making them inherently robust to random perturbations. As a result, altering a small subset of randomly selected parameters has a minimal effect on the model outputs. Recent studies reveal that a small number of outliers are disproportionally important to model performance. In the following, we present a lightweight approach for selecting protected neurons within the FFN module of a designated transformer block. Intuitively, a neuron can have a significant impact if it can trigger weight outliers within $\mathbf{W}^{\text{down}}$, which then propagate to the hidden state space and produce feature outliers. We trace outliers in reverse to find such critical neurons. In the first step, we randomly generate a batch of input samples and perform forward passes through the model to identify feature outliers within the output vector of the FFN module:

$$O_f = \{i \mid \bar{y}_i > \tau \cdot \mu_y\}, \tag{1}$$

where $\bar{y}_i$ is the average magnitude of the i-th feature produced by the designated FFN module, $\mu_y \triangleq \frac{1}{D_{\text{m}}} \sum_{i=1}^{D_{\text{m}}} \bar{y}_i$ is the mean of these average magnitudes, and τ is the threshold that controls the number of selected features. In the second

step, we rank the impact of individual neurons based on the following scoring function:

$$s_j = \sum_{i \in O_f} |\mathbf{W}_{j,i}^{\text{down}}| \cdot \bar{u}_j, \qquad (2)$$

where $\bar{u}_j$ is the average magnitude of the post-activation value of the j-th neuron. s_j estimates the contribution of the j-th neuron to the features selected in the first step. Neurons with the highest scores are selected as candidate protected neurons.

As discussed in §2, feature outliers tend to persist at the same location across consecutive transformer blocks. To prevent information leakage from earlier blocks, we choose the first block exhibiting emergent feature outliers as the protected block.

3.3 Key Obfuscation

Outlier features are disproportionally important for model performance. Therefore, an adversary can launch an approximate oracle-guided attack to recover most of the model's functionality. For example, it can identify a small subset of *critical neurons* by iteratively flipping or muting each protected neuron and measuring the impact on the model's output. Subsequently, it enumerates all permutations of the subset and selects the one that minimizes the discrepancy with the oracle model (Li et al. 2024). In this way, the adversary can quickly find an approximately correct permutation, as the remaining neurons have only a negligible effect on the overall model performance.

We propose an obfuscation method to address this issue. Our technique aims to *i)* smooth the features in the intermediate layer of the protected block, so that each key bit has a similar impact on model performance; *ii)* enhance the correlations among the key bits, so that the quality of model outputs depends on as many key bits as possible. LLA utilizes a sequence of orthogonal transformations (Ashkboos et al. 2024b; Lin et al. 2024; Ashkboos et al. 2024a) to realize these goals. An orthogonal matrix $\mathbf{M}$ is a square matrix such that $\mathbf{M}\mathbf{M}^\top = \mathbf{I}$. Hence, an orthogonal matrix and its transpose can be inserted before and after a sequence of linear layers without changing the functionality of the model. We present the details of our obfuscation method in the remainder of this subsection.

Permutation of Outliers. In the first step, we build an orthogonal *permutation matrix* $\mathbf{P}$ to reorder neurons within the designated FFN module. Concretely,

$$\mathbf{P} = \mathbf{P}_1 \cdots \mathbf{P}_n, \qquad (3)$$

where $\mathbf{P}_j$ swaps the j-th protected neuron (§3.2) with the j-th neuron in the original FFN module. After this step, the identified outlier features are repositioned to the front of the feature dimension.

Rotation of Features. In the second step, we construct an orthogonal *rotation matrix* $\mathbf{R}$ as follows:

$$\mathbf{R} = \begin{bmatrix} \mathbf{H} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \qquad (4)$$

where $\mathbf{H}$ is a randomized Hadamard matrix (Tseng et al. 2024; Ashkboos et al. 2024b) and $\mathbf{I}$ is an identity matrix. $\mathbf{H}$
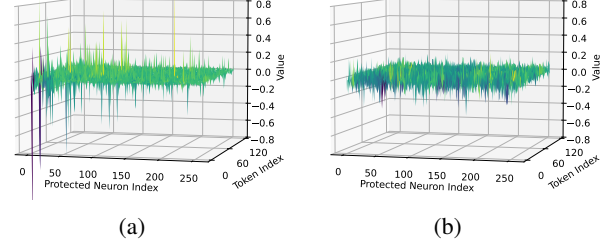

(a)   (b)

Figure 3: (a) Feature outliers are prominent before the application of $\mathbf{R}$; (b) Feature outliers are eliminated after the application of $\mathbf{R}$.

is constructed by scaling an n-dimensional Hadamard matrix with $\frac{1}{\sqrt{n}}$, then multiplying it with a *random diagonal matrix* with entries independently sampled from $\{-1, 1\}$. A Hadamard matrix is a square matrix whose entries are either -1 or 1 and whose rows are mutually orthogonal. An n-dimensional Hadamard matrix is guaranteed to exist if n is a power of 2, and it is known to exist for almost all n that is a multiple of 4 and less than 1000 (Wallis 1976). $\mathbf{R}$ is inserted after the activation layer for a standard FFN module, or after the element-wise multiplication operator for a gated FFN module. Fig. 3 visualizes the smoothing effect of the rotation matrix.

The rotation transformation has the following benefits: *i)* it evenly distributes the effects of outlier features across the first n dimensions by smoothing their magnitudes; *ii)* it enhances the interdependence among key bits, thereby obscuring the statistical relationships between key bits and model outputs (Shannon 1949); *iii)* it discretizes the effects of outlier features in the output space of $\mathbf{R}$, thus mitigating gradient-based oracle-guided attacks.

Insertion of Keys. In the third step, we create an orthogonal *key matrix* $\mathbf{K}$ given a permutation $\boldsymbol{\pi}$ that rearrange n elements:

$$\mathbf{K} = \begin{bmatrix} \mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \qquad (5)$$

where $\mathbf{G}$ is also a permutation matrix. Each element $\mathbf{G}_{i,j}$ is equal to 1 if $\boldsymbol{\pi}(i) = j$, and 0 otherwise. To reduce hardware complexity, we further require that $\boldsymbol{\pi}$ can be partitioned into disjoint groups, each of which is a permutation of m elements. As a result, the computation of $\mathbf{G}$ can be realized using key-controlled permutation fabrics of size m. We elaborate on the hardware support in §3.4.

Obfuscation with Orthogonal Transformations. A standard FFN module can be expressed as $\mathbf{Y} = \sigma(\mathbf{X}\mathbf{W}^{\text{up}})\mathbf{W}^{\text{down}}$, where σ denotes the element-wise activation function, $\mathbf{X}$ represents the input matrix of the FFN module, and $\mathbf{Y}$ represents the corresponding output matrix. The overall obfuscation method can be summarized as the following:

$$\mathbf{Y} = \sigma(\mathbf{X}\mathbf{W}^{\text{up}}\mathbf{P})\mathbf{R}\mathbf{K}(\mathbf{K}^\top\mathbf{R}^\top\mathbf{P}^\top\mathbf{W}^{\text{down}}). \qquad (6)$$

After commuting $\mathbf{P}$ with $\sigma(\cdot)$, each orthogonal matrix is canceled by its transpose, so the final output after transfor-
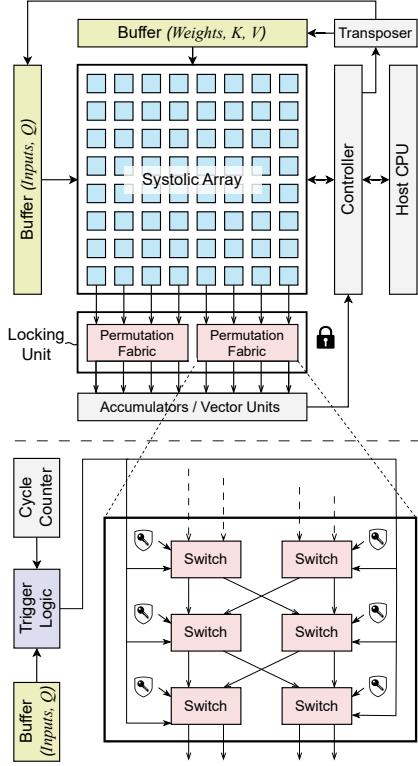
Figure 4: Schematic of a systolic AI accelerator that supports model locking.

mations remains unchanged. To reduce computational cost, we merge all orthogonal matrices within the same parentheses into $\mathbf{W}^{\mathrm{up}}$ or $\mathbf{W}^{\mathrm{down}}$, resulting in the following transformations:

$$\mathbf{Y} = \sigma(\mathbf{X}\tilde{\mathbf{W}}^{\mathrm{up}})\mathbf{R}\mathbf{K}(\tilde{\mathbf{W}}^{\mathrm{down}}). \qquad (7)$$

As shown in Formula 4, the main component of $\mathbf{R}$ is a randomized Hadamard matrix $\mathbf{H}$, whose dimensionality corresponds to the key size n. In contrast, the dimensionality of $\mathbf{R}$ matches the intermediate dimension D_{ff} of the FFN module, which is typically on the order of 10,000 or higher in modern GenAI models (Dubey et al. 2024). Therefore, when the key size is on the order of hundreds, the computational overhead caused by $\mathbf{R}$ is negligible. Notice that the random diagonal matrix within $\mathbf{R}^{\top}$ introduces additional confusion to $\tilde{\mathbf{W}}^{\mathrm{down}}$, helping to offset the potential information leakage caused by $\mathbf{K}^{\top}$.

The above method can also be applied to a gated FFN module by distributing $\mathbf{P}$ to both $\mathbf{W}^{\mathrm{up}}$ and $\mathbf{W}^{\mathrm{gate}}$.

3.4 Hardware Support

Systolic Array Architecture. The *systolic array* is the core computational module in modern AI accelerators (Chen et al. 2020; Ju and Gu 2022). It comprises a mesh of interconnected process elements (PEs), each performing a scalar multiplication and accumulation (MAC) operation in a clock cycle. *Weight-stationary* and *output-stationary* are two representative dataflow schemes of systolic arrays. In

the weight-stationary scheme, a tile of the weight matrix is retained with the PEs, whereas the input matrix and the partial sums are streamed through the PEs during computation. In the output-stationary scheme, the partial sums are retained within the PEs, while the input and weight matrices are streamed through the PEs.

Hardware Design. LLA is designed to be agnostic to specific AI compilers or accelerators. Fig. 4*(left)* illustrates the architecture of a systolic array accelerator that supports model locking. The *locking module* consists of several permutation fabrics, each shuffling a fixed number of consecutive output lanes of the systolic array. This additional module is responsible for performing the multiplication with the key matrix $\mathbf{K}$. Fig. 4*(right)* shows an implementation of a 4×4 permutation fabric with a key-controlled Benes network (Beneš 1964). The network comprises multiple stages of 2×2 switches, each controlled by a key bit that determines whether to pass through or swap the two input signals. The *trigger logic* is inactive when the systolic array computes any matrix other than the key matrix $\mathbf{K}$. In that case, it outputs the default key pattern that preserves the original order of all lanes. When the systolic array computes the $\mathbf{K}$ matrix, the trigger unit introduces specific delays to each output lane based on the dataflow schemes, ensuring that the signals reach the permutation fabric within the same clock cycle. Meanwhile, it configures the routing of the Benes network with input key bits to realize the desired permutation. Weight-stationary systolic arrays may compute an output matrix in multiple rounds by accumulating several intermediate matrices. LLA applies the same key to each intermediate matrix to achieve the intended result.

4 Experiment and Analysis

Setup. We apply the proposed LLA model locking approach to four pre-trained LLMs from the Minitron family (Muralidharan et al. 2024). These models represent different types of FFNs with varying sizes. Their statistics are summarized in Table 1. As model size increases, LLMs tend to exhibit more feature outliers (Dettmers et al. 2022), which in turn raises the cost of successful attacks (Li et al. 2024). Consequently, the positive results observed on these relatively small models indicate that LLA can be more effective when applied to larger models. All experiments are conducted on a Linux workstation with a 2.4 GHz CPU and an NVIDIA RTX A6000 GPU.

We use MMLU and perplexity to evaluate the capabilities of LLMs. A higher MMLU score or a lower perplexity indicates better performance. Fidelity is defined as the proportion of protected neurons whose original indices are restored by an attacking algorithm. We choose the Jensen-Shannon divergence (JSD) to measure the similarity between the output distributions of the encrypted model and the oracle model.

The primary threat to model locking is the oracle-guided (OG) attack. In this setting, an adversary can query the oracle model with any input sequence and observe the corresponding output logits. If the oracle model is unavailable, the adversary can instead launch a pretraining-style oracle-
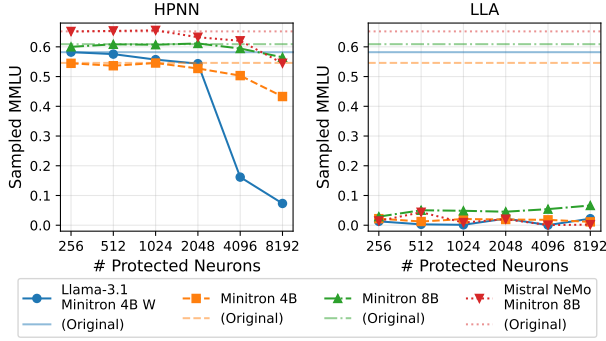
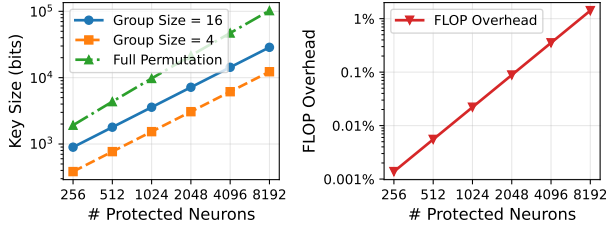Figure 5: Pre-attack locking effectiveness: LLA *vs.* HPNN.

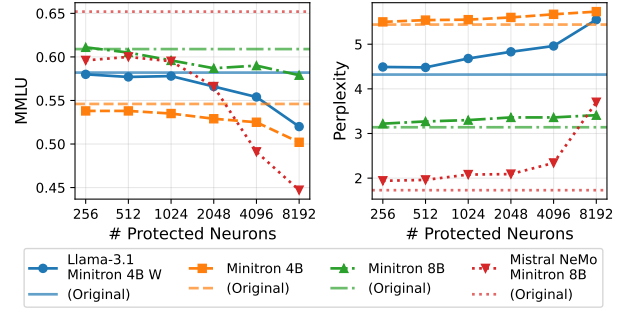Figure 6: Locking efficiency: key size and FLOP overhead.

Figure 7: Locking robustness: performance of LLA-protected models after the OG gradient-based attack.
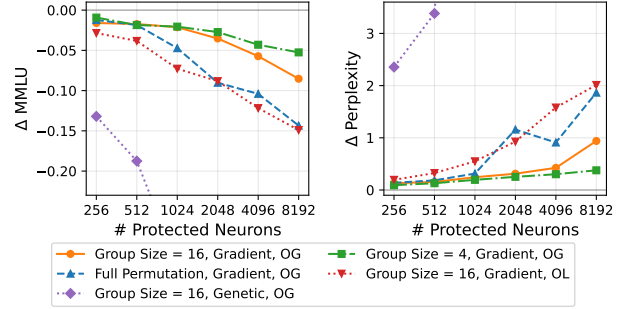

Figure 8: Locking robustness: performance of LLA-protected models under various attack and defense settings (averaged across 4 models).

less (OL) attack using a curated dataset. We consider two prevalent optimization techniques for key decryption. The *genetic-based attack* (Alam et al. 2022) iteratively evolves the candidate key pattern in the binary key space. Our implementation employs the tournament selection mechanism, uses JSD as the fitness function, applies within-group pairwise swaps as the mutation operator, and performs crossover across permutation groups. For the *gradient-based attack* (Li et al. 2024), existing methods cannot be applied directly to the permutation fabric. According to our experiments, the most effective way is to directly restore the permutation matrix **G**. Specifically, we use a *softmax* function to approximate each column of the matrix. Upon completion, the element with the highest magnitude is set to 1, and the remaining elements are set to 0. We use Adam optimization with a learning rate of 0.03. We choose JSD as the loss function for the OG attack and cross-entropy for the OL attack. We observe that the gradient-based attack is consistently more effective than the genetic-based attack. Therefore, our evaluation focuses primarily on the former attack. We set a time limit of 7,200 seconds for every execution.

We assess LLA with various numbers of protected neurons, ranging from 256 to 8192. By default, the permutation group size is set to 16. We configure the outlier threshold τ to 5, which can be increased for larger generative models. Based on the emergence of outliers, we select the first block of `Mistral NeMo Minitron 8B` and the second block of the other three models as protected blocks.

Results. We systematically evaluate LLA in terms of its effectiveness, efficiency, and robustness. First, we assess the locking effectiveness of LLA by comparing it with HPNN (Chakraborty, Mondai, and Srivastava 2020), the

state-of-the-art model locking technique. For each locking configuration, we report the average model performance across 10 randomly generated key patterns. As shown in Figure 5, HPNN causes only a slight performance degradation in most configurations. In contrast, LLA consistently renders the models unusable in all configurations.

Second, we measure the locking efficiency of LLA in terms of key length and computational overhead (Figure 6). For a fixed permutation group size, the key length grows linearly with the number of protected neurons. LLA requires 3.5 key bits per protected neuron for a group size of 16 and 1.5 bits per neuron for a group size of 4. On the other hand, the computational overhead in FLOP count is negligible, especially for smaller key sizes. The effectiveness and efficiency of LLA can be attributed to its ability to manipulate outliers and absorb orthogonal matrices.

Third, we evaluate the locking robustness of LLA by testing it against the aforementioned genetic-based and gradient-based attacks. For each configuration, we report the *best* attack result observed across three attempts. As depicted in Figure 7, post-attack performance varies by model type, but robustness consistently improves as the key length increases. The fidelity (Figure 10*(right)*) never exceeds 86% for a small key length (256 protected neurons) and 38% for a large key length (8192 protected neurons), suggesting that an adversary cannot recover the original functionality of the model. We further analyze the robustness of LLA under var-

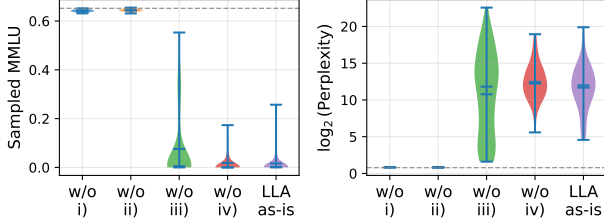| Model | FFN Type $D_{\mathrm{m}} \times D_{\mathrm{ff}}$ | Original Perf. | |
		MMLU	Perplexity
Minitron 4B	standard 3072 × 9216	0.546	5.44
Minitron 8B	standard 4096 × 16384	0.609	3.14
Llama-3.1 Minitron 4B W	gated 3072 × 9216	0.582	4.32
Mistral Nemo Minitron 8B	gated 4096 × 11520	0.652	1.73

Table 1: LLMs used for evaluation.

Figure 9: Ablation study: pre-attack performance of `Mistral NeMo Minitron 8B` under various LLA configurations. The dashed lines indicate the performance of the original model.

ious attack and defense settings (Figure 8). We observe that *i)* robustness can be enhanced with a greater permutation group size, though it comes at the expense of increased key length; *ii)* the gradient-based attack outperforms the genetic-based attack; *iii)* OG attacks are more effective than OL attacks when other configurations are the same.

Ablation Study. We conduct an ablation study to identify the key factors that contribute to the effectiveness and robustness of LLA. Specifically, we examine the following factors: *i)* selection of the protected block, *ii)* selection of the protected neurons, *iii)* application of the rotation matrix, and *iv)* use of permutation-based locking instead of negation-based locking. We select HPNN as the baseline method.

We compare the locking effectiveness across various LLA configurations in a non-adversarial setting. For each configuration, we randomly generate 100 key patterns and plot the corresponding model performance in Fig. 9. Removing either *i)* or *ii)* leads to minimal performance degradation, highlighting the critical role of outlier features. Removing *iii)* still allows certain key patterns to maintain relatively high performance. Further analysis reveals that these patterns retain the original positions of most critical neurons, exposing the vulnerability of model locking under insufficient obfuscation. The last two configurations consistently degrade model performance across all key patterns, demonstrating their effectiveness in a non-adversarial setting.

Finally, we evaluate the locking robustness of the last two configurations in Fig. 9. The negation-based locking scheme was first proposed by HPNN, and the permutation-based locking scheme was introduced by LLA. For the negation-based locking scheme, we replace every key bit with a *tanh* function during the attack. Upon convergence, the key bits
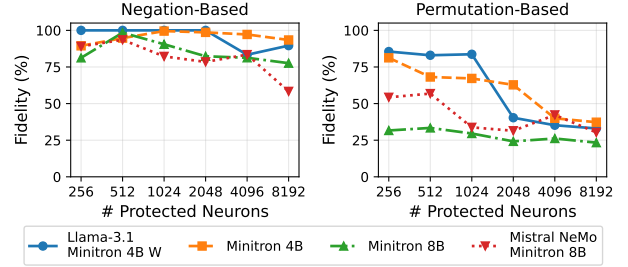

Figure 10: Ablation study: comparing the locking robustness of negation-based and permutation-based locking schemes.

with negative values are set to `true`, while the remaining bits are set to `false` (Li et al. 2024). Figure 10 compares the post-attack fidelity of the two schemes. The negation-based scheme is more vulnerable to gradient-based attacks. Notably, the adversary can even recover 100% of key bits for the `Llama-3.1 Minitron 4B W` model under various settings. Therefore, the permutation-based scheme is necessary to ensure the robustness of model locking.

Security Analysis. As discussed previously, LLA-protected models are resistant to *oracle-guided key optimization attacks*. Alternatively, an adversary may fix a key value and launch *oracle-guided fine-tuning attacks* to improve model performance. However, training with an incorrectly fixed key can be more costly than training from scratch. LLA is also resistant to *probing attacks* and various *side-channel attacks* because: *i)* there are only delay registers between the systolic array and the locking module, and *ii)* the execution of a protected model follows a fixed pattern that is independent of the key value. The adversary may exploit the geometric and algebraic properties of neural networks to launch *oracle-guided geometric attacks* (Li et al. 2024). LLA can thwart these attacks because: *i)* all key bits are placed within the same intermediate layer, which has a larger dimension D_{ff} than the input dimension D_{m}; geometric attacks are ineffective on expansive layers. *ii)* The key bits are tightly correlated due to orthogonal transformations, whereas geometric attacks rely on a divide-and-conquer strategy to reduce computational complexity.

5 Conclusion and Future Work

This paper presents LLA, the first model locking method tailored for large generative models. It brings together a set of techniques, including outlier selection, key obfuscation, and systolic array hardware support, to safeguard the supply chain of generative models. Experiments demonstrate that LLA can mitigate a wide range of attacks, particularly oracle-guided key optimization attacks, at a minimal computational and hardware cost. Limitations of this work include: *i)* it is not as effective on tiny generative models where prominent outliers are absent; *ii)* an adversary with sufficient computational resources may leverage model distillation to replace the entire protected block. We plan to address these challenges in future work.

Acknowledgments

References

Adi, Y.; Baum, C.; Cisse, M.; Pinkas, B.; and Keshet, J. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *USENIX Security 2018*.

Alam, M.; Saha, S.; Mukhopadhyay, D.; and Kundu, S. 2022. NN-Lock: A lightweight authorization to prevent IP threats of deep learning models. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(3): 1–19.

An, Y.; Zhao, X.; Yu, T.; Tang, M.; and Wang, J. 2025. Systematic Outliers in Large Language Models.

Ashkboos, S.; Croci, M. L.; do Nascimento, M. G.; Hoefler, T.; and Hensman, J. 2024a. SliceGPT: Compress Large Language Models by Deleting Rows and Columns.

Ashkboos, S.; Mohtashami, A.; Croci, M.; Li, B.; Cameron, P.; Jaggi, M.; Alistarh, D.; Hoefler, T.; and Hensman, J. 2024b. Quarot: Outlier-free 4-bit inference in rotated llms. *NIPS 2024*.

Beneš, V. E. 1964. Permutation groups, complexes, and rearrangeable connecting networks. *Bell System Technical Journal*, 43(4): 1619–1640.

Chakraborty, A.; Mondai, A.; and Srivastava, A. 2020. Hardware-assisted intellectual property protection of deep learning models. In *DAC 2020*.

Chakraborty, R. S.; and Bhunia, S. 2009. HARPOON: An obfuscation-based SoC design methodology for hardware protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10): 1493–1502.

Chen, Y.; Xie, Y.; Song, L.; Chen, F.; and Tang, T. 2020. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3): 264–274.

Chou, S.-Y.; Chen, P.-Y.; and Ho, T.-Y. 2023. How to backdoor diffusion models? In *CVPR 2023*.

Dettmers, T.; Lewis, M.; Belkada, Y.; and Zettlemoyer, L. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *NIPS 2022*.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv:2407.21783*.

Geva, M.; Schuster, R.; Berant, J.; and Levy, O. 2021. Transformer Feed-Forward Layers Are Key-Value Memories. In *EMNLP 2021*.

Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; and Wernsing, J. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML 2016*.

Goldstein, B. F.; Patil, V. C.; Ferreira, V. C.; Nery, A. S.; França, F. M.; and Kundu, S. 2021. Preventing DNN model IP theft via hardware obfuscation. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(2): 267–277.

Ju, Y.; and Gu, J. 2022. A 65nm systolic neural CPU processor for combined deep learning and general-purpose computing with 95% PE utilization, high data locality and enhanced end-to-end performance. In *ISSCC 2022*.

Kamali, H. M.; Azar, K. Z.; Farahmandi, F.; and Tehranipoor, M. 2022. Advances in logic locking: Past, present, and prospects. *Future Microelectronics Security Research Series*.

Kamali, H. M.; Azar, K. Z.; Gaj, K.; Homayoun, H.; and Sasan, A. 2018. Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection. In *ISVLSI 2018*.

Khan, N.; Nitzsche, S.; López, A. G.; and Becker, J. 2021. Utilizing and extending trusted execution environment in heterogeneous SoCs for a pay-per-device IP licensing scheme. *IEEE Transactions on Information Forensics and Security*, 16: 2548–2563.

Kirchenbauer, J.; Geiping, J.; Wen, Y.; Katz, J.; Miers, I.; and Goldstein, T. 2023. A watermark for large language models. In *ICML 2023*.

Kovaleva, O.; Kulshreshtha, S.; Rogers, A.; and Rumshisky, A. 2021. Bert busters: Outlier dimensions that disrupt transformers. *arXiv:2105.06990*.

Li, M.; Shamsi, K.; Meade, T.; Zhao, Z.; Yu, B.; Jin, Y.; and Pan, D. Z. 2017. Provably secure camouflaging strategy for IC protection. *IEEE transactions on computer-aided design of integrated circuits and systems*, 38(8): 1399–1412.

Li, P.; Huang, J.; and Zhang, S. 2025. LicenseNet: Proactively safeguarding intellectual property of AI models through model license. *Journal of Systems Architecture*, 159: 103330.

Li, Y.; Zhao, G.; He, Y.; and Zhou, H. 2024. Evaluating the Security of Logic Locking on Deep Neural Networks. In *DAC 2024*.

Li, Z.; You, C.; Bhojanapalli, S.; Li, D.; Rawat, A. S.; Reddi, S. J.; Ye, K.; Chern, F.; Yu, F.; Guo, R.; et al. 2023. The Lazy Neuron Phenomenon: On Emergence of Activation Sparsity in Transformers. In *ICLR 2023*.

Lin, H.; Xu, H.; Wu, Y.; Cui, J.; Zhang, Y.; Mou, L.; Song, L.; Sun, Z.; and Wei, Y. 2024. Duquant: Distributing outliers via dual transformation makes stronger quantized llms. *NIPS 2024*.

Lin, N.; Chen, X.; Lu, H.; and Li, X. 2020. Chaotic weights: A novel approach to protect intellectual property of deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7): 1327–1339.

Men, X.; Xu, M.; Zhang, Q.; Wang, B.; Lin, H.; Lu, Y.; Han, X.; and Chen, W. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv:2403.03853*.

Mo, F.; Shamsabadi, A. S.; Katevas, K.; Demetriou, S.; Leontiadis, I.; Cavallaro, A.; and Haddadi, H. 2020. Darknetz: towards model privacy at the edge using trusted execution environments. In *MobiSys 2020*.

Mu, X.; Wang, Y.; Huang, Z.; Lai, J.; Zhang, Y.; Wang, H.; and Yu, Y. 2024. EncryIP: A Practical Encryption-Based Framework for Model Intellectual Property Protection. In *AAAI 2024*.

Muralidharan, S.; Turuvekere Sreenivas, S.; Joshi, R.; Chochowski, M.; Patwary, M.; Shoeybi, M.; Catanzaro, B.; Kautz, J.; and Molchanov, P. 2024. Compact language models via pruning and knowledge distillation. *NIPS 2024*.

Nicholas, G. S.; Siddiqui, A. S.; Joseph, S. R.; Williams, G.; and Saqib, F. 2021. A secure boot framework with multi-security features and logic-locking applications for reconfigurable logic. *Journal of Hardware and Systems Security*, 5(3): 260–268.

Shannon, C. E. 1949. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4): 656–715.

Sun, M.; Chen, X.; Kolter, J. Z.; and Liu, Z. 2024. Massive Activations in Large Language Models. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

Sun, X.; Zhang, P.; Liu, J. K.; Yu, J.; and Xie, W. 2018. Private machine learning classification based on fully homomorphic encryption. *IEEE Transactions on Emerging Topics in Computing*, 8(2): 352–364.

Takhar, G.; Karri, R.; Pilato, C.; and Roy, S. 2022. HOLL: Program synthesis for higher order logic locking. In *TACAS 2022*.

Tang, X.; Giacomin, E.; Alacchi, A.; Chauviere, B.; and Gaillardon, P.-E. 2019. OpenFPGA: An opensource framework enabling rapid prototyping of customizable FPGAs. In *FPL 2019*.

Tseng, A.; Chee, J.; Sun, Q.; Kuleshov, V.; and De Sa, C. 2024. QuIP #: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks.

Tuyls, P.; Schrijen, G.-J.; Škorić, B.; Van Geloven, J.; Verhaegh, N.; and Wolters, R. 2006. Read-proof hardware from protective coatings. In *8th International Workshop of Cryptographic Hardware and Embedded Systems, CHES 2006*.

Uchida, Y.; Nagai, Y.; Sakazawa, S.; and Satoh, S. 2017. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on international conference on multimedia retrieval*.

Wallis, J. S. 1976. On the existence of Hadamard matrices. *Journal of Combinatorial Theory, Series A*, 21(2): 188–195.

Wang, C.; Deng, Y.; Ning, Z.; Leach, K.; Li, J.; Yan, S.; He, Z.; Cao, J.; and Zhang, F. 2023. Building a lightweight trusted execution environment for arm gpus. *IEEE Transactions on Dependable and Secure Computing*, 21(4): 3801–3816.

Xu, J.; Wang, F.; Ma, M. D.; Koh, P. W.; Xiao, C.; and Chen, M. 2024. Instructional fingerprinting of large language models. *arXiv:2401.12255*.

Yin, L.; Wu, Y.; Zhang, Z.; Hsieh, C.-Y.; Wang, Y.; Jia, Y.; Li, G.; Jaiswal, A.; Pechenizkiy, M.; Liang, Y.; et al. 2024. Outlier Weighed Layerwise Sparsity (OWL): A Missing Secret Sauce for Pruning LLMs to High Sparsity. In *ICML 2024*.

Yu, M.; Wang, D.; Shan, Q.; and Wan, A. 2024. The Super Weight in Large Language Models. *arXiv:2411.07191*.

Zhao, S.; Wen, J.; Luu, A.; Zhao, J.; and Fu, J. 2023. Prompt as Triggers for Backdoor Attack: Examining the Vulnerability in Language Models. In *EMNLP 2023*.

Zhou, T.; Luo, Y.; Ren, S.; and Xu, X. 2023. NNSplitter: an active defense solution for DNN model via automated weight obfuscation. In *ICML 2023*.

Zhuang, H.; Zhang, Y.; and Liu, S. 2023. A pilot study of query-free adversarial attack against stable diffusion. In *CVPR 2023*.

Zou, A.; Wang, Z.; Carlini, N.; Nasr, M.; Kolter, J. Z.; and Fredrikson, M. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv:2307.15043*.

Zuo, P.; Hua, Y.; Liang, L.; Xie, X.; Hu, X.; and Xie, Y. 2021. Sealing neural network models in encrypted deep learning accelerators. In *DAC 2021*.