SECNEURON: Reliable and Flexible Abuse Control in Local LLMs via Hybrid Neuron Encryption

Zhiqiang Wang*, Haohua Du†, Junyang Wang*, Haifeng Sun*, Kaiwen Guo‡, Haikuo Yu*, Chao Liu‡, Xiang-Yang Li*
*University of Science and Technology of China
Email: {sa21221041, iswangjy, sun1998, yhk7786}@mail.ustc.edu.cn, xiangyangli@ustc.edu.cn
†Beihang University
Email: duhaohua@buaa.edu.cn
‡Ocean University of China
Email: {kevinguo, liuchao}@ouc.edu.cn

arXiv:2506.05242v1 [cs.CR] 5 Jun 2025

***Abstract*— Large language models (LLMs) with diverse capabilities are increasingly being deployed in local environments, presenting significant security and controllability challenges. These locally deployed LLMs operate outside the direct control of developers, rendering them more susceptible to abuse. Existing mitigation techniques mainly designed for cloud-based LLM services are frequently circumvented or ineffective in deployer-controlled environments.**

We propose SECNEURON, the first framework that seamlessly embeds classic access control within the intrinsic capabilities of LLMs, achieving reliable, cost-effective, flexible, and certified abuse control for local deployed LLMs. SECNEURON employs neuron-level encryption and selective decryption to dynamically control the task-specific capabilities of LLMs, limiting unauthorized task abuse without compromising others. We first design a task-specific neuron extraction mechanism to decouple logically related neurons and construct a layered policy tree for handling coupled neurons. We then introduce a flexible and efficient hybrid encryption framework for millions of neurons in LLMs. Finally, we developed a distribution-based decrypted neuron detection mechanism on ciphertext to ensure the effectiveness of partially decrypted LLMs. We proved that SECNEURON satisfies IND-CPA Security and Collusion Resistance Security under the Task Controllability Principle. Experiments on various task settings show that SECNEURON limits unauthorized task accuracy to below 25% while keeping authorized accuracy loss with 2%. Using an unauthorized Code task example, the accuracy of abuse-related malicious code generation was reduced from 59% to 15%. SECNEURON also mitigates unauthorized data leakage, reducing PII extraction rates to below 5% and membership inference to random guesses. Additionally, SECNEURON enables one-time encryption & transmission and multi-party selective decryption, requiring millisecond-level key generation and byte-level key exchange for local LLM capability adjustment.

I. INTRODUCTION

Local deployment of LLMs is an increasingly popular paradigm, chosen by many organizations and individuals for private or domain-specific application [1]–[4]. However, LLMs incorporate increasingly diverse task capabilities that exceed the usage requirements of specific deployers, thereby increasing abuse-related risks in local deployment scenarios. For instance, students can use LLMs to generate assignments or even academic misconduct articles [5], while malicious users can exploit models like GPT to write harmful code and phishing emails [6].

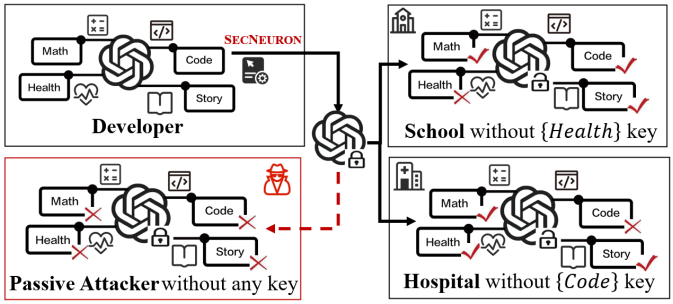


Fig. 1: Workflow of SECNEURON. Developer encrypts their LLM once (*One-time Encryption*). Different deployers download the same encrypted LLM and dynamically decrypt authorized tasks while restricting unauthorized capabilities to mitigate abuse. (*Multi-party Selective Decryption*). The developer maintained and released one single encrypted LLM, while deployers also download it once even if permissions change (*One-time Transmission*).

Preventing the abuse of LLM has gradually become a focal point of both societal and academic concern. Existing solutions can primarily be categorized into two aspects: 1) temporary defences that constrain user interactions, such as malicious input/output (I/O) filter [19]–[21] or safety system prompt and 2) adjust the LLMs to refuse inappropriate queries or tailor for specific tasks, such as safety alignment [9]–[12]or task-specific fine-tuning/distillation [7], [8], [22]–[24]. These works predominantly address scenarios where the LLM is deployed in the cloud, meaning that the **runtime environment is controlled by the model developer and the abuse behaviours are pre-known** (e.g., according to human values). Consequently, their effectiveness relies on two assumptions: 1) constraints on user interactions should be honestly enforced, and 2) sufficient resources and capabilities to adjust the LLM according to explicitly defined abuse behaviours. These assumptions make it challenging to apply these methods to local deployments.

Local LLMs are deployed on private clouds or local PCs,

TABLE I: Comparison of Potential Security Mechanisms for Mitigating Abuse of Local LLMs

Method	Limit UnAuth. Task	Reliable			Flexibile		Data Protect
		Limit UnAuth. Capability	Robust*	Intrinsic★	Customized	Dynamic+	
Distillation/Fine-tunning [7], [8]	○	○	○	●	◐†	○	○
Safety Alignment [9]–[12]	●	○	○	●	○	○	○
Unlearning [12]–[14]	●	●	●	●	◐†	○	●
Watermarking [15]–[18]	○	○	○	○	○	○	○
Malicious I/O Detection [19]–[21]	●	○	○	○	●	●	○
SECNEURON	●	●	●	●	●	●	●

1. **Limit UnAuth. Task** refers to restricting LLMs from completing unauthorized tasks; **Limit UnAuth. Capability** directly limits the model's underlying capabilities, making it inherently unable to perform unauthorized tasks (low performance), and is more robust and reliable.
2. *: robustness against malicious prompt for abuse, like jailbreak or prompt injection. ★: is extremely important for local deployment, as temporary security mechanisms can be easily bypassed or removed locally. †: requires significant overhead to fine-tune or maintain multiple versions of LLMs, which is impractical. +: tasks of local LLMs can be dynamically adjusted with minimal overhead.

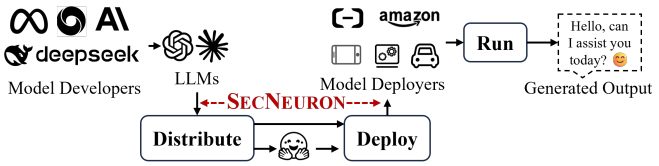


Fig. 2: Pipeline for local deployment of LLMs. SECNEURON enhances security during distribution and deployment.

where **the runtime environment is controlled by the model deployer (users who might be malicious).** Moreover, deployment requirements may be customized and dynamic (e.g., the function of a local LLM in a family may vary depending on the intended user - certain capabilities should be restricted when children access the model). Therefore, abuse in local deployments extends beyond the pre-known generation of malicious content to include using any unauthorized model functionalities by malicious deployers. For instance, a hospital might require only the medical diagnosis capabilities of LLMs, while needing to strictly limit its code generation ability. It is evident that the two aforementioned assumptions are difficult to meet in this context: 1) deployer-controlled environments cannot impose constraints on deployer behaviour, and 2) adjusting the model for various deployment requirements is prohibitively expensive. In summary, the expansion of abuse definition and changes in deployment scenarios make existing methods challenging to adapt, as shown in Table I. Notably, while Trusted Execution Environments (TEEs) [25], [26] provide strong security protection for local LLMs, they focus primarily on the confidentiality and integrity of critical parameters during the running time and cannot prevent abuses.

There is an urgent need for effective methods to dynamically implement task-level abuse control for locally deployed LLMs. As previously discussed, imposing constraints on deployer behaviour in deployer-controlled environments is impossible. Hence, **we propose that the abuse control mechanisms for local LLMs must directly operate on the instinct capabilities of models, i.e., limiting capabilities on unauthorized tasks.** For example, even if users employ adversarial prompts, an LLM without code generation capabilities would still be unable to generate malicious code (e.g., Figure 12 in §6.3). Besides, the mechanisms should be efficient and flexible for LLMs with millions of neurons to support customized deployment. It brings two challenges that must be addressed:

• ***C1: How to ensure that capabilities on unauthorized tasks are limited without disrupting the authorized tasks?*** We aim to limit the LLMs' capabilities (performance on generating next tokens) on unauthorized tasks rather than merely restricting the completion of these behaviors (e.g., safety alignment or I/O detection). Deactivating or pruning important neurons for specific tasks, like unlearning [13], could be an effective method to limit their capabilities. However, LLMs are optimized for multitasks during training, making it challenging to fully isolate these logically related neurons for different tasks [27], [28], restricting unauthorized tasks may inadvertently affect others. Therefore, more effective mechanisms are required for algorithmically decoupling task-specific neurons and strategies to address unintended coupling.

• ***C2: How to enable local LLMs adaptable to customized deployer permission with minimal overhead?***

Existing solutions for customized LLMs (Unlearning or distillation/pruning for specific tasks) are inherently irreversible. Developers require significant resources for fine-tuning or maintaining and transmitting multiple versions of LLMs to meet different user permissions, with the overhead increasing linearly with the number of permissions. Neurons for specific tasks should be dynamically forcibly disabled or activated (reversible) to meet the customized and dynamic permission requirements of different deployment scenarios.

Fortunately, we found that cryptographic tools can provide a reversible and training-free method tailored for customized capability limitation: encrypting specific neurons limits capability on certain tasks while decrypting them restores the capability. Based on this intuition, we designed and implemented SECNEURON, a secure mechanism that enables one-time encryption of the LLM, with different users dynamically decrypting and gaining different capabilities (Figure 1). Compared to the methods in Table I, SECNEURON is characterized by its cost-effectiveness, flexibility, and reliability.

SECNEURON DESIGN. The core of SECNEURON are neuron encryption and selective decryption: deployers can

dynamically decrypt the neurons they are authorized to access, executing only the authorized tasks. Firstly, we designed a penalty-based task-specific neuron extraction mechanism to enhance existing neuron importance analysis methods complemented by an efficient mechanism for handling coupled neurons (***Addressing C1***). Then, we propose a hybrid encryption framework, particularly designed for LLMs with millions of neurons, that balances the flexibility of attribute-based encryption with the efficiency of symmetric encryption (***Addressing C2***). *Policy Layer*: Neurons are assigned different keys and access policies based on task relevance. The Ciphertext-Policy Attribute-Based (CP-ABE) with a carefully designed policy tree is used to manage keys and coupled neurons across different tasks. *Execution Layer*: Advanced Encryption Standard (AES) is employed for neuron parameters encryption and decryption; each neuron only needs to be encrypted once. Deployers dynamically obtain decryption keys based on their attributes, allowing them to decrypt only the authorized portions. Since undecrypted neurons can degrade the overall performance of partially decrypted LLM, we designed an undecrypted neuron detection mechanism based on the randomness distribution of ciphertext for adaptive pruning. SECNEURON offers the following advantages:

Flexible and Efficient. SECNEURON implements an efficient mechanism with one-time encryption & transmission, and multi-party decryption (Figure 1), enables dynamic capability updates and flexible permission configuration based on user attributes. For example, permissions such as *(Institution = Hospital) AND (Licence = True)* can restrict access to LLMs' diagnosis functionality.

Reliable and Certified. SECNEURON enforces certified capability constraints through neuron-level encryption. Once a neuron's association with an unauthorized task is explicitly identified, SECNEURON can theoretically ensure that it cannot be activated or utilized, providing a provable safeguard against task abuse.

We summarize three contributions of this work:

- We propose a novel abuse mitigation mechanism - SECNEURON- which creatively integrates classic access control policies with the intrinsic capabilities of LLMs, enabling flexible, reliable, and certified abuse control even under deployer-controlled environments. To the best of our knowledge, SECNEURON is the first dynamic task-level capability management method for LLMs and can serve as a plugin to secure existing deployment pipelines (Figure 2).
- We introduce a task-specific reusable neuron decoupling and managing algorithm that enables task-grained capability control at the neuron level. We further propose a hybrid hierarchical encryption framework to support efficient and flexible encryption and decryption of millions of neurons. Additionally, we develop a ciphertext distribution-based neuron identification algorithm to ensure the effectiveness of partially decrypted LLMs.
- SECNEURON effectively limits the accuracy of unauthorized tasks to below 25% while ensuring authorized tasks are impacted by less than 2%. It also prevents unauthorized training data extraction, with success rates of PII lower than 5% and MIA nearly 0%. Furthermore, SECNEURON reduces the encryption and transmission overhead associated with permissions from linear to constant levels, requiring only millisecond-level key generation and byte-level key exchange for local LLM capability updates.

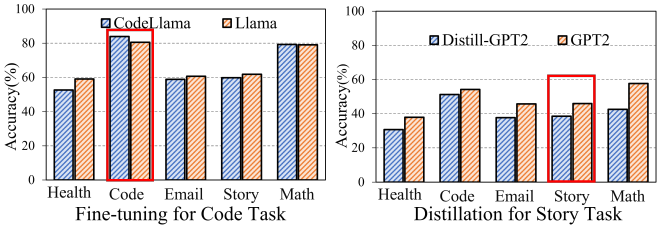


Fig. 3: Multi-task performance after fine-tuning for Code task (CodeLlama [7]) or distillation for Story task (Distill-GPT2 [8]). The capabilities of other tasks have not been significantly limited and can still be abused.

II. BACKGROUND AND RELATED WORKS

A. Motivating Use Cases

SECNEURON offers developers a framework for controlled LLM distribution in local deployment scenarios. This section outlines the motivating use cases for SECNEURON, including existing and future scenarios.

Secure and Controllable Model Publishing. Plenty of LLMs are published on platforms like Hugging Face, allowing users to download and deploy them locally. SECNEURON can be seamlessly integrated into existing pipelines, providing a low-cost solution to enhance both the security (encrypted model transmission) and controllability during the distribution process (Figure 2).

Fine-grained Commercial Licensing. Model developers create large-scale systems capable of executing diverse high-value tasks concurrently. SecNeuron facilitates domain-specific licensing to various clients, enabling customizable task activation according to individual client needs.

Dynamic On-device Deployment. An increasing number of smart devices come pre-installed with LLMs. When device permissions change (e.g., switching to child mode), SECNEURON can activate or disable specific tasks (e.g., social content) through a lightweight key exchange, avoiding re-downloading the LLM.

B. Security Issues: Multi-level Abuse

Task Abuse (Model Level). Once the model is distributed, developers lose control over how the model is used, creating risks of performing unauthorized tasks. On the one hand, it poses a serious threat to developers' intellectual property, as the functionality of the LLM represents algorithmic innovations and the significant costs associated with training. On the other hand, this may violate legal boundaries. For example, malicious users may exploit unauthorized coding capabilities to generate harmful scripts [6].

Training Data Extraction (Data Level). LLMs may memorize details from their training data, making it possible for malicious users to reverse-engineer training data from the model's outputs. The training dataset, containing domain-specific knowledge and trade secrets, is also essential for developer's intellectual property. Moreover, some datasets include large amounts of sensitive data (such as PII in Enron Email Dataset [29]), posing significant security risks.

C. Related Works for Mitigating Abuse

Distillation & Pruning & Fine-tuning for Specific Task. These approaches are designed to adapt LLMs to specific tasks [7], [8], [22]–[24], which effectively preserve LLMs' capabilities for authorized (target) tasks but do not impose strict constraints on unauthorized ones. As illustrated in Figure 3, even after fine-tuning or distillation, the model retains its capability to perform other tasks, which may lead to potential abuse.

Safety Alignment. Safety alignment [9]–[12] aligns LLMs with specific safety objectives and constraints through fine-tuning or reinforcement learning, ensuring their behaviour adheres to authorized tasks. Such methods, which primarily rely on refusal to respond, can limit the behaviour of unauthorized tasks but fail to fundamentally restrict the underlying capabilities of LLMs, leaving them vulnerable to adversarial prompts [30]–[32], [32].

Unlearning. Unlearning [12]–[14] aims to remove or mitigate knowledge or patterns that a model has previously learned, aligning closely with our goal of limiting the model's capabilities. However, unlearning is irreversible, meaning that once capabilities for a specific task of an LLM are restricted, they cannot be restored, making it unsuitable for dynamic, customized local deployment.

Watermarking. Watermarking techniques embed invisible markers for copyright verification by modifying model parameters [33]–[35] or output distribution [36]. Recently, numerous watermarking methods tailored for LLMs have been proposed [15]–[18], especially, [17] proposes watermark-based protections to address misuse of local LLMs. However, these approaches fall under post hoc detection methods and cannot proactively prevent misuse.

Malicious I/O Detection. They work by monitoring and restricting behavior on unauthorized tasks through external input and output detection [19]–[21]. It is widely used in cloud-based LLM applications and is potentially an efficient method for customizing model tasks across different deployment scenarios. However, such temporary solutions can be easily bypassed or removed when LLM is deployed in deployer-controlled environments [37].

D. CP-ABE & AES-CTR

This section introduces cryptography algorithms used in the paper.

Ciphertext-Policy Attribute-Based Encryption (CP-ABE). CP-ABE [38], [39] is an advanced cryptographic technique that enables fine-grained access control over encrypted data. Data owners define access policy, which specifies the conditions under who can decrypt. Decryption is only possible if the user's attributes satisfy the access policy embedded in the ciphertext. A CP-ABE Cryptor (E_1 in Algorithm 1,4) includes four main phases:

(1)`Setup` ($PK, MSK \leftarrow E_1.setup()$): Initialize bilinear group G_1 and target group G_T; generate PK and MSK.

(2)`Encrypt` ($C_p \leftarrow E_1.encrypt(PK, M, p)$): use the PK to encrypt M and embed the access policy p into the ciphertext C_p. p is often represented as a tree, where the nodes correspond to logical operators such as AND, OR, and threshold gates.

(3)`KeyGen` ($SK \leftarrow E_1.keyGen(PK, \mathcal{A})$): generate attribute-based secret key SK by PK and attribute $\mathcal{A}$.

(4)`Decrypt` ($M' \leftarrow E_1.decrypt(PK, SK, C_p)$): attempt to decrypt the ciphertext C_p using SK. If attributes $\mathcal{A}$ satisfy the access policy in the ciphertext, the decryption succeeds; otherwise, it fails.

Advanced Encryption Standard - Counter Mode (AES-CTR). AES-CTR [40], [41] is a widely used symmetric encryption mode that transforms AES into a stream cipher. By combining a unique nonce and an incrementing counter, AES-CTR generates a keystream derived from the master key, which is then XORed with the plaintext for encryption or with the ciphertext for decryption. It is highly efficient, parallelizable, and supports random access to encrypted data, making it suitable for applications.

III. Overview

A. Threat Model

As shown in Figure 1, our system consists of two primary entities: the ***Model Developer*** and ***Model Deployer***.

Model Developer. Model developers train LLMs with the capability to perform a wide range of tasks, and then distribute these well-trained models to model deployers for local deployment and use. The training process may involve high-value or diverse datasets, as well as proprietary architectures and training methodologies. Consequently, developers may seek to distribute these models under controlled conditions for local deployment, specifically by defining task-level access policies tailored to different deployers to ensure the proper use of their LLMs.

Model Deployer. The model deployer, in this context, acts as the adversary. Different deployers have dynamic and customized deployment requirements, often a specific task subset of the LLM. **They have access to a locally deployed (fully white-box) LLM with authorization for some tasks (Partially Authorized Users) or none at all (Passive Attacker).** Any capabilities of LLMs that exceed the authorized scope of deployers pose the risk of abuse through the following malicious behaviours:

- *Unauthorized Task Abuse* aims to illegally invoke the LLM to perform unauthorized tasks.
- *Training Data Extraction* represents a more advanced attacker aiming to infer unauthorized training data from the model outputs. We specifically consider two common training

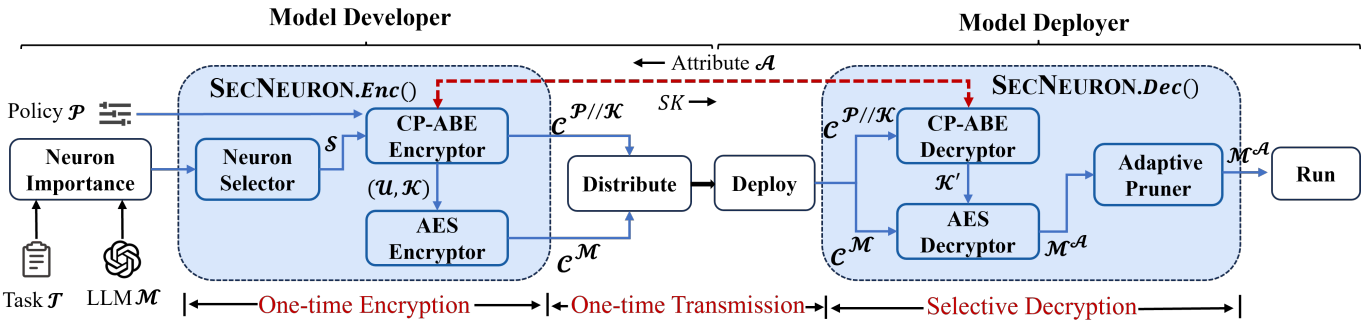


Fig. 4: Overview of SECNEURON framework. SECNEURON (blue part) serves as a plug-and-play secure tool in model distribution and local deployment pipeline. When permissions change, developers can adjust the capability of the local LLM by only simple key exchanges (red dotted line) with $O(1)$ complexity ($ms-level$ computation and $B-level$ transmission overhead), eliminating re-encrypt and re-transmit LLM with $O(\mathcal{M})$ complexity ($mins-level$ computation and $GB-level$ transmission overhead).

data extraction attacks for LLMs: membership inference [42] and PII extraction [43].

Usage Scope. SECNEURON offers a secure framework for controlled model distribution and local deployment that mitigates potential abuse of locally deployed LLMs. Ensuring security during runtime from parameter theft has been extensively studied [25], [26]. SECNEURON is orthogonal to them and can combine for more comprehensive protection (§8). SECNEURON aims to prevent users from abusing LLM capabilities for which they lack authorization and can resist collusion, i.e., performing unauthorized tasks by combining partially authorized keys or permissions.. The reconstruction of full model capabilities by collusive users who possess authorization for all tasks falls outside the scope of this study, as such attacks stem from underlying access policy vulnerabilities (Details in §5.3).

B. Design Goals and Formulation

LLMs are fine-tuned using specific training datasets to enable specific downstream tasks. Based on this, we divide tasks $\mathcal{T}$ according to the training datasets $\mathcal{D}$, meaning that different datasets D_t correspond to different tasks t. The protection scope of SECNEURON can be summarized as a tuple $<\mathcal{D}, \mathcal{T}>$, i.e., the high-value datasets and the model capabilities trained on them, restricting them from being unauthorized abuse and extraction.

Given a well-trained LLM $\mathcal{M}$ and a set of tasks $\mathcal{T}$, SECNEURON achieves secure and controllable distribution and deployment through encryption and selective decryption of neurons in LLM:

Developers define access policy $\mathcal{P}$ and encrypt LLM using SECNEURON (Algorithm 1: $\mathcal{C}^{\mathcal{M}}, \mathcal{C}^{\mathcal{P}//\mathcal{K}} \leftarrow$ SECNEURON.$Enc(\mathcal{M}, \mathcal{T}, \mathcal{P})$);

Deployers decrypt the encrypted model based on their attribute $\mathcal{A}$ (Algorithm 4: $\mathcal{M}^{\mathcal{A}} \leftarrow$ SECNEURON.$Dec(\mathcal{C}^{\mathcal{M}}, \mathcal{C}^{\mathcal{P}//\mathcal{K}}, \mathcal{A})$).

$\mathcal{M}^{\mathcal{A}}$ needs to meet the following objectives:

1) Preserve the performance of authorized tasks. For any task t in the authorized task set $\mathcal{T}_{\mathcal{A}}$, the performance of $\mathcal{M}^{\mathcal{A}}$ is essentially consistent with that of $\mathcal{M}$.

$$\forall t \in \mathcal{T}_{\mathcal{A}}, \mathcal{P}_t(\mathcal{M}) - \mathcal{P}_t(\mathcal{M}^{\mathcal{A}}) < \gamma_t, \tag{1}$$

γ_t is a small constant, where smaller indicate greater similarity in performance. $\mathcal{P}_t(\mathcal{M})$ represents the performance (generating correct next tokens) on task t for $\mathcal{M}$, which, in this paper, is measured using $Accuracy$ (Eqution 7).

2) Limit the capabilities of unauthorized tasks. For task t in the unauthorized task set $\mathcal{T}_{\mathcal{U}}$, performance of $\mathcal{M}^{\mathcal{A}}$ should be limited.

$$\forall t \in \mathcal{T}_{\mathcal{U}}, \mathcal{P}_t(\mathcal{M}^{\mathcal{A}}) < \delta_t, \tag{2}$$

δ_t represents a small positive number that defines the upper bound of the model's performance on unauthorized tasks.

3) Prevent extraction of unauthorized training data. For task t in unauthorized task set $\mathcal{T}_{\mathcal{U}}$, prevent the extraction of D_t from $\mathcal{M}^{\mathcal{A}}$:

$$\forall t \in \mathcal{T}_{\mathcal{U}}, MIA(\mathcal{M}^{\mathcal{A}}, D_t) < \epsilon, PII(\mathcal{M}^{\mathcal{A}}, D_t) < \epsilon, \tag{3}$$

$MIA(\cdot)$ (Membership Inference Attack) [42] and $PII(\cdot)$ (Personally Identifiable Information Attack) [43] refer to the attack success rate of two commonly used training data extraction methods. ϵ is a small positive number that limits the success rate.

Developers can adjust $\gamma_t, \delta_t, \epsilon$ according to the value or security requirements of the task t (data). For example, for highly sensitive and valuable *Health* tasks, a smaller δ_t can be set, whereas, for $Story$ task, the restrictions on δ_t can be relaxed.

SECNEURON must also fulfill the following practicality objectives:

1) Low Overhead. Encryption and decryption must have efficient computational overhead for plenty of neurons, and the encrypted model should not introduce excessive transmission overhead.

2) Flexibility. SECNEURON should support complex permission configurations, allowing locally downloaded LLMs to

achieve efficient capability adjustments based on the permission of deployers.

C. Challenges and Solutions

• ***C1: How to ensure that capabilities on unauthorized tasks are limited without disrupting the authorized tasks?***

Deactivating or pruning neurons corresponding to unauthorized tasks seems like an effective way to limit capability on these tasks. However, modern LLMs are inherently multitask systems designed to handle a wide range of tasks by leveraging shared neuron representations, meaning multiple tasks may activate the same neurons. Existing pruning methods [22]–[24] overlook the coupling of neurons across different tasks. Specifically, important neurons for unauthorized tasks may also partially contribute to authorized tasks. Naively pruning them could unintentionally degrade the performance of authorized tasks. Similarly, focusing solely on preserving important neurons for the target task does not guarantee restriction of unauthorized tasks (Figure 3).

Solution. Recognizing that perfect, mutually exclusive neuron isolation for each task is often impractical, our approach focuses on managing and mitigating the effects of neuron coupling. To achieve this, we first introduce the penalty factor λ to enhance the decoupling of task-specific neurons [13] (critical for target tasks but insignificant for others). However, overlapping neurons across different tasks are sometimes unavoidable. Thus, we embed the control of overlapping neurons into the access tree of CP-ABE (Figure 6), eliminating the need for additional management.

• ***C2: How to enable local LLMs adaptable to customized deployer permission with minimal overhead?***

Deployer permissions are dynamic and require flexible capability adjustment for local LLMs. For example, when children use LLMs, it is necessary to limit the capability of social content generation. Existing solutions for customized LLMs are irreversible, imposing prohibitive overhead on developers and deployers: Developers must maintain multiple task-specific model versions (including encryption, storage, and transmission), and deployers must repeatedly download and deploy the corresponding LLM versions. As the complexity of permission combinations increases, the management cost and overhead grow linearly. While task-specific distillation or pruning [44]–[46] can reduce the cost of a single transmission, the cumulative cost of multiple transmissions remains significant.

Solution. From the cryptographic perspective, we propose a reversible capability limitation mechanism for customized LLM needs: encrypting neurons limits capability on certain tasks while decrypting them restores the capability. Specifically, we introduce a hybrid encryption mechanism to handle the vast number of parameters in LLMs, balancing the flexibility of CP-ABE and the efficiency of AES:

1)Policy Layer (**CP-ABE Encryptor**): Assigns AES keys to neurons based on their task relevance and binds access policies to these keys. Then, CP-ABE is used to encrypt and manage them.

Algorithm 1: Encryptor: SECNEURON.$Enc()$

Data: Original Model: $\mathcal{M}$, Task List: $\mathcal{T}$, Policy $\mathcal{P}$
Result: Encrypted Model: $\mathcal{C}^{\mathcal{M}}$,Policy & Key Encrypted by CP-ABE: $\mathcal{C}^{\mathcal{P}//\mathcal{K}}$

1 $\mathcal{C}^{\mathcal{P}//\mathcal{K}} \leftarrow \emptyset, \mathcal{C}^{\mathcal{M}} \leftarrow \mathcal{M}$;
2 Create CP-ABE Cryptor E_1 and AES Cryptor E_2;
3 ► **Neuron Selector**
4 Select important neurons for each task: $\mathcal{S} \leftarrow$ SECNEURON.$select(\mathcal{M}, \mathcal{T})$;
5 ► **CP-ABE Encryptor:** $\boldsymbol{Enc_{ABE}(PK, \mathcal{P}//\mathcal{K}) \rightarrow \mathcal{C}^{\mathcal{P}//\mathcal{K}}}$
6 CP-ABE Init.: $PK, MSK \leftarrow E_1.setup()$;
7 Decompose all possible combinations of $\mathcal{S}$ into disjoint subsets: $\mathcal{U} \leftarrow \{\bigcap_{t\in\mathcal{T}'} \mathcal{S}_t \setminus \bigcup_{t\in\mathcal{T}\setminus\mathcal{T}'} \mathcal{S}_t : \mathcal{T}' \subseteq \mathcal{T}, \mathcal{T}' \neq \emptyset\} \cup \{\Omega \setminus \bigcup_{t\in\mathcal{T}} \mathcal{S}_t\}$;
8 **foreach** *subset* $\mathcal{N} \in \mathcal{U}$ **do**
9 Select a random element k_{gt} from the target group GT;
10 Create access policy p for k_{gt} based on $\mathcal{P}$;
11 Encrypt k_{gt} and p using PK by E_1: $c \leftarrow E_1.encrypt(PK, k_{gt}, p)$;
12 $\mathcal{C}^{\mathcal{P}//\mathcal{K}} \leftarrow \mathcal{C}^{\mathcal{P}//\mathcal{K}} \cup c$;
13 ►**AES Encryptor:** $\boldsymbol{Enc_{AES}(\mathcal{K}, \mathcal{M}) \rightarrow \mathcal{C}^{\mathcal{M}}}$
14 Generate 64-bit AES key k from k_{gt};
15 **foreach** *neuron* $n \in \mathcal{N}$ **do**
16 Encrypt $\mathcal{M}$ using k by E_2: $\mathcal{C}_n^{\mathcal{M}} \leftarrow$ SECNEURON.$encrypt(k, \mathcal{M}, n, E_2)$
17 **return** $\mathcal{C}^{\mathcal{M}}, \mathcal{C}^{\mathcal{P}//\mathcal{K}}$;

2)Execution Layer (**AES Encryptor**): Uses AES-CTR to encrypt neuron parameters in LLMs, with AES keys for each neuron generated and managed by *Policy Layer*.

Deployers download the complete encrypted model and then obtain the authorized keys based on their attributes (permissions) to decrypt and utilize permitted tasks adaptively.

IV. SECNEURON DESIGN

As shown in Figure 4, SECNEURON has two components: the Encryptor (Algorithm 1: SECNEURON.$Enc()$) for the model developer and the Decryptor (Algorithm 4: SECNEURON.$Dec()$) for the model deployer. The Encryptor is executed only once to generate an encrypted version of the LLM. Different deployers can adaptively use the Decryptor to access the authorized portions of the encrypted model based on their permissions.

A. Encryptor for Model Developer

Neuron Selector. The Neuron Selector is used to identify task-specific neurons (important only for the target task but not for other tasks) for each task in $\mathcal{T}$ (Lines 3-4 in Algorithm 1). Referenced from [13], we use the mean of absolute activation to calculate the importance of each neuron, and then we introduce λ as a penalty factor to calculate the task-specific score. Let n be a neuron and denote its activations by z_n.

Algorithm 2: SECNEURON.*select*()

Data: Original Model: $\mathcal{M}$, Task List: $\mathcal{T}$, Importance Score Function S, Importance Threshold: τ
Result: Selected Neurons: $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset$;
foreach *task* $t \in \mathcal{T}$ **do**
  $\mathcal{S}_t \leftarrow \emptyset$;
  Calculate the task-specific score for each neuron n: $s_n \leftarrow S(\mathcal{T}, t, n)$;
  Normalize and Sort s in descending order;
  Select Neurons:
  $\mathcal{S}_t \leftarrow \mathcal{S}_t.append(n)$ if $\mathcal{S}_t.sum() + s_n < \tau \cdot s.sum()$;
  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_t$;
return $\mathcal{S}$

Given a task $t \in \mathcal{T}$ and its sampled training dataset D_t, we define task-specific scoring function S as:

$$I(t,n) := \frac{\sum_{d \in D_t} z_n(d)}{|D_t|}, \tag{4}$$

$$S(\mathcal{T},t,n) := I(t,n) - \lambda \cdot \max_{t' \in \mathcal{T}, t' \neq t} I(t',n) \tag{5}$$

The larger the value of S, the more important neuron n is specific for task t. Therefore, we select neurons from the largest S value for each task, continuing until the cumulative sum exceeds the threshold τ. **The calculation of $I(t,n)$ can adopt other effective neuron importance estimation methods.** In particular, the neuron selection algorithm is shown in Algorithm 2.

CP-ABE Encryptor *(Policy Layer).* CP-ABE Encryptor is responsible for key management and does not directly participate in model encryption (Lines 5-12 in Algorithm 1). The process involves three key steps: *CP-ABE Init.*, *AES Key & Policy Gen.* and *CP-ABE Enc.*:

1) CP-ABE Init. Directly invoke the setup mechanism of CP-ABE to generate a public key PK and a master secret key MSK, which are used to derive attribute-based secret keys SK for users.

2) AES Key & Policy Gen. For selected important neurons S, we decompose them into multiple disjoint subsets to address the overlap between task-specific neurons of different tasks (the overlapping neurons are treated as a separate subset. For example, neurons from $t1$, neurons from $t2$, and neurons shared by both $t1$ and $t2$ are decomposed into distinct subsets $\mathcal{U} = \{\mathcal{S}_{t1}, \mathcal{S}_{t2}, \mathcal{S}_{t1} \cap \mathcal{S}_{t2}\}$) (Line 7 in Algorithm 1). For each subset $\mathcal{N} \in \mathcal{U}$, we randomly select an k_{gt} from G_T group of CP-ABE to generate the AES key k, which serves as the encryption key for all neurons in that subset (Figure 5). At the same time, the access policy tree is constructed, assigning policy p to each key (Line 8-10 in Algorithm 1). Furthermore, neurons that are not selected by any task are designated as a separate common subset. These common neurons are also assigned an encryption keys with the access policy that permits any authorized task to access them, thereby enhancing security against passive attackers.

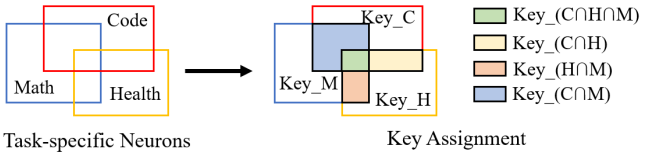


Fig. 5: Illustration of AES key assignment.

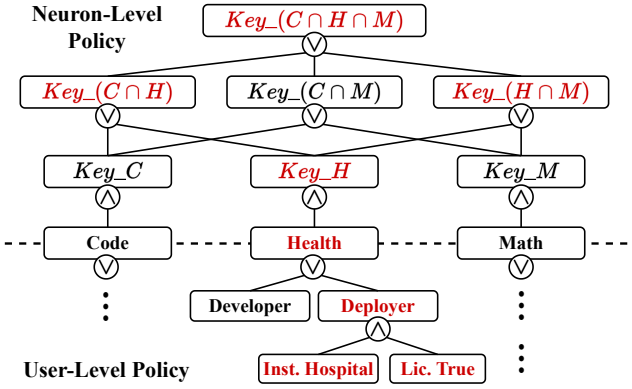


Fig. 6: Access policy tree for the CP-ABE encryptor,∧ represents logical AND, while ∨ represents logical OR. An example of authorized access is highlighted in red: Deployers meet $Institution = Hospital\ and\ Licence = True$ are allowed to use the Health task of the LLM. They can access keys such as $\{Key\_H, Key\_(H \cap C), ...\}$ to decrypt the corresponding neurons.

Access to overlapping neurons is also integrated as part of the policy tree, eliminating the need for additional management steps. Specifically, we divide the policy tree into two layers based on tasks (Figure 6): Neuron-level Policy and User-level Policy. To minimize the impact on authorized tasks, Neuron-level Policy employs *OR* nodes to manage the overlap of neurons and is a built-in, immutable mechanism. User-level Policy specifies the access rights of deployers with different attributes for each task, allowing developers to adjust the policy flexibly according to specific requirements.

3) CP-ABE Enc. Encrypt all generate k_{gt} and embed the policy p into the ciphertext, with directly using the CP-ABE encryption function $E_1.encrypt(\cdot)$.

AES Encryptor *(Execution Layer).* The AES Encryptor is used for encrypting each neuron in the LLM model by AES-CTR (Lines 13-16 in Algorithm 1).

Given a neuron n and its input x_n, the neuron feedforward process (MLP layer) can be summarized as:

$$x'_n = \mathcal{M}_n.W_{OUT} \cdot \sigma(\mathcal{M}_n.W_{IN} \cdot x_n + \mathcal{M}_n.B_{IN}), \tag{6}$$

where W_{IN} and B_{IN} are the input weight and bias, respectively, $\sigma(\cdot)$is the activation function (e.g., ReLU or Sigmoid), W_{OUT} represents the output weight. As shown in Algorithm 3, AES Encryptor simultaneously encrypts the parameters

Algorithm 3: SECNEURON.$encrypt()$

Data: Encrypt Key: k, Original Model: $\mathcal{M}$, Neuron Index: n, AES Cryptor $E2$
Result: Encrypted Neuron parameters: $\mathcal{C}_n^{\mathcal{M}}$

$\mathcal{C}_n^{\mathcal{M}} \leftarrow \mathcal{M}_n$;
Init CTR Mode: $E2.MODE \leftarrow CTR$, $E2.COUNTER \leftarrow n$;
Encrypt W_{IN}: $\mathcal{C}_n^{\mathcal{M}}.W_n^{IN} = E2.encrypt(k, \mathcal{M}_n.W_{IN})$;
Encrypt W_{OUT}: $\mathcal{C}_n^{\mathcal{M}}.W_n^{OUT} = E2.encrypt(k, \mathcal{M}_n.W_{OUT})$;
Encrypt B_{IN}: $\mathcal{C}_n^{\mathcal{M}}.B_n^{IN} = E2.encrypt(k, \mathcal{M}_n.B_{IN})$;
return $\mathcal{C}_n^{\mathcal{M}}$

$\mathcal{M}_n.W_{OUT}$, $\mathcal{M}_n.W_{IN}$ and $\mathcal{M}_n.B_{IN}$ to encrypt a single neuron n (with the same encrypt key). Moreover, to ensure that the decryption side can randomly decrypt any neuron, we utilize the AES encryption in CTR mode ($E_2.encrypt(\cdot)$) and pass n as the counter value.

B. Decryptor for Model Deployer

CP-ABE Decryptor *(Policy Layer)*. The CP-ABE Decryptor derives the authorized AES decryption key based on attributes of deployers (lines 2-9 in Algorithm 4). Firstly, the deployer requests an attribute-based secret key SK based on their attributes $\mathcal{A}$. Then, SK is used to decrypt $\mathcal{C}^{\mathcal{P}//\mathcal{K}}$ to obtain the authorized k_{gt} (the access policy is already embedded in the ciphertext, so the CP-ABE decryption function $E1.decrypt(\cdot)$ can be directly invoked without explicit permission checks). Finally, SECNEURON convert each correctly decrypted k_{gt} to AES key k for subsequent AES decryption.

AES Decryptor *(Policy Layer)*. AES Decryptor is used to decrypt model parameters. We provide two decryption mechanisms: *Transmission-efficient decryption (T-E dec.)* and *Computation-efficient decryption (C-E dec.)*. Algorithm 4 uses *T-E dec.* as an example.

- *Transmission-efficient decryption (T-E dec.):* Transmit only the encrypted LLM (identical size to original LLM) and CP-ABE ciphertext once, eliminating additional transmission overhead but requiring decryption of the entire LLM.

Since there is no metadata assistance, AES Decryptor attempts to decrypt each neuron using all authorized keys (Lines 10-17 in Algorithm 4), requiring verification of whether neurons have been correctly decrypted (Line 15 in Algorithm 4). AES encryption operates at the byte stream level, causing neurons of different data types to exhibit distinctive characteristics after encryption due to their varied byte-level serialization patterns in memory (Figure 14). Consequently, we propose an efficient undecrypted neuron detection mechanism for two predominant parameter types in LLM: *INT* and *FLOAT*.

FLOAT16. Encryption of FLOAT16 parameters may affect the 'exponent' bits, leading to extremely large outlier values (2^{16}). Such anomalous values are absent in well-trained LLMs. Therefore, by detecting these outliers in the neuron

Algorithm 4: Decryptor: SECNEURON.$Dec()$

Data: Encrypted Model: $\mathcal{C}^{\mathcal{M}}$, Policy & Key Encrypted by CP-ABE: $\mathcal{C}^{\mathcal{P}//\mathcal{K}}$, Attribute List: $\mathcal{A}$, Public key: PK
Result: Decrypted Model: $\mathcal{M}^{\mathcal{A}}$

1 Create CP-ABE Cryptor E_1 and AES Cryptor E_2;
2 ▶ **CP-ABE Decryptor:** $\boldsymbol{Dec_{ABE}(PK, \mathcal{C}^{\mathcal{P}//\mathcal{K}}, \mathcal{A}) \rightarrow \mathcal{K}'}$
3 Request attribute-based secret key SK using PK and $\mathcal{A}$;
4 Init authorized keys: $\mathcal{K}' \leftarrow \emptyset$;
5 **foreach** *ciphertext* $c \in \mathcal{C}^{\mathcal{P}//\mathcal{K}}$ **do**
6 Decrypt c using PK and SK by E_1: $k_{gt} \leftarrow E_1.decrypt(PK, SK, c)$;
7 **if** k_{gt} *is correct decrypted* **then**
8 Generate 64-bit AES key k from k_{gt};
9 $\mathcal{K}' \leftarrow \mathcal{K}' \cup k$;
10 ▶ **AES Decryptor:** $\boldsymbol{Dec_{AES}(\mathcal{K}', \mathcal{C}^{\mathcal{M}}) \rightarrow \mathcal{M}^{\mathcal{A}}}$
11 $\mathcal{M}^{\mathcal{A}} \leftarrow 0$;
12 **foreach** *neuron* n *of* $\mathcal{C}^{\mathcal{M}}$ **do**
13 **foreach** *key* $k \in \mathcal{K}'$ **do**
14 Decrypt $\mathcal{C}_n^{\mathcal{M}}$ using k by E_2: $m \leftarrow$ SECNEURON.$decrypt(k, \mathcal{C}^{\mathcal{M}}, n, E_2)$;
15 **if** m *is correct decrypted* **then**
16 $\mathcal{M}_n^{\mathcal{A}} \leftarrow m$;
17 **break**;
18 ▶ **Adaptive Pruner**
19 **if** $\mathcal{M}_n^{\mathcal{A}} == 0$ **then**
20 Prune neuron n from $\mathcal{M}^{\mathcal{A}}$;
21 **return** $\mathcal{M}^{\mathcal{A}}$;

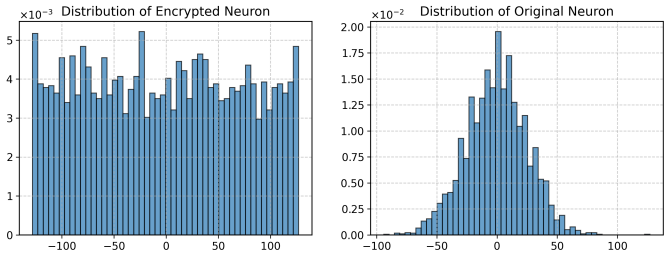


Fig. 7: The distribution (W_{IN}) of encrypted neurons exhibits notable differences compared to original neurons.

parameters, we can determine whether it has been decrypted (the same with *FLOAT32*).

INT8. The range of *INT8* model parameters lies between [-128, 127] and remains invariant even after encryption, precluding the use of outliers for detection. After encryption, the data follows a uniform distribution, whereas the parameters of a trained model exhibit a specific, non-uniform distribution. Therefore, we can determine whether a neuron has been decrypted by its distribution pattern (Figure 7).

TABLE II: Computational & Transmission Complexity.

	First Deployment		Capability Update	
	Computational	Transmission	Computational	Transmission
Encryptor	$O(\mathcal{M})$	$O(\mathcal{M})$	$O(1)$	$O(1)$
T-E Dec.	$O(\lvert\mathcal{K}'\rvert \cdot \mathcal{M})^*$	$O(\mathcal{M})$	$O(\lvert\mathcal{K}'\rvert \cdot \mathcal{M})^*$	$O(1)$
C-E Dec.†	$O(\mathcal{M}_t)$	$O(\mathcal{M} + \mathcal{N}_{neuron})$	$O(\mathcal{M}_t)$	$O(1)$

1: CP-ABE is used for encrypting and managing keys, with overhead significantly smaller than the processing of LLMs. We disregard its overhead to simplify the complexity analysis.
2: * represents the worst-case complexity, $|\mathcal{K}'|$ refers to number of authorized keys.
3 †: $\mathcal{N}_{neuron}$ refers to the number of neurons, $\mathcal{N}_{neuron} << \mathcal{M}$ (for a 6.7B LLM, the size of $\mathcal{N}_{neuron}$ is only around 500KB.); $\mathcal{M}_t$ refers to authorized portion of LLM for task t.

Specifically, for *FLOAT-type* neurons, we select the maximum value of their input matrix $\mathcal{M}_n.W_{IN}$ as the metric (m); for *INT-type* neurons, we use the variance of the statistical histogram of $\mathcal{M}_n.W_{IN}$ as the metric (v_H). The final determination of whether a neuron is correctly decrypted is made through threshold comparison. We tested the detection effectiveness across different models and achieved a success rate of 100% in all cases (§6.5).

• *Computation-efficient decryption (C-E dec.):* Transmit additional metadata indicating which key each neuron uses, only decrypting the authorized neurons.

When deploying, users download the LLM along with metadata F that indicates the AES key used for each neuron. The size of F equals the number of neurons with transmission overhead smaller than the LLM itself. During decryption, each neuron first retrieves its key based on F and then determines whether it is authorized (whether CP-ABE can decrypt it). If authorized, the neuron is decrypted; otherwise, it undergoes adaptive pruning. When permissions change, only neurons with changed permissions need to be adjusted. For single-task decryption, this approach can reduce decryption overhead by 40%. Table II analyzes the computational and transmission complexities of different decrypt methods.

Adaptive Pruner. The Adaptive Pruner prunes all unauthorized neurons (lines 18-20 in Algorithm 4), accelerating the inference of locally deployed LLMs without affecting the performance on authorized tasks.

V. Security Analysis

In this section, we analyzed the security of SecNeuron. First, we defined the Task Controllability Principle to ensure that all tasks can be effectively protected and proved the Task Capacity Upper Bound for a given LLM, a necessary but insufficient condition for judging whether a task configuration is reasonable (§5.1). Then, we proved that SecNeuron satisfies IND-CPA Security (§5.2) and Collusion Resistance Security (§5.3).

A. Task Controllability Principle

When serving as a separate unauthorised task, any task in $\mathcal{T}$ needs to satisfy Equation 2, meaning each task requires a sufficient number of task-specific neurons with no intersection with other tasks. To achieve this goal, we define the Task Controllability Principle.

Definition 5.1.**(Task Controllability Principle)** *Given a LLM $\mathcal{M}$ and tasks $\mathcal{T}$, for each task $t \in \mathcal{T}$, there exists a neuron set $S'_t \subseteq \mathcal{S}_t$ satisfying the following conditions:*

1) For any two different tasks $t_1, t_2 \in \mathcal{T}$ satisfying: $S'_{t_1} \cap S'_{t_2} = \emptyset$;

2) Removing S'_t causes the performance of task t to fall below the target threshold: $\mathcal{P}_t(\mathcal{M} \setminus S'_t) < \delta_t, \mathcal{P}_t(\cdot)$ refers to performance on t.

If satisfying the Task Controllability Principle, the task set $\mathcal{T}$ is controllable, meaning any task $t \in \mathcal{T}$ can be constrained to perform within its target threshold δ_t. Notably, the Task Controllability Principle should not be interpreted as an assumption of absolute physical separation for all neurons involved in a task, but rather as a controllability principle that defines how specific task capabilities can be effectively constrained. SecNeuron does not assume a priori that all neurons across different tasks are isolatable, but rather seeks to construct such effective isolation neuron sets. Despite general neuron entanglement in large LLMs, their large-scale neural architectures allow us to readily pinpoint neuron subsets for each task(comprising approximately 15%) that satisfy the Task Controllability Principle. However, for a given model $\mathcal{M}$, the number of tasks it can handle is not unlimited. We further propose and prove the Task Capacity Upper Bound theorem, which establishes a necessary but not sufficient condition to determine whether $\mathcal{M}$ can effectively manage all tasks in set $\mathcal{T}$.

Theorem 1. *(Task Capacity Upper Bound) For the LLM $\mathcal{M}$ and a task set $\mathcal{T}$ to satisfy the Task Controllability Principle, it is necessary to ensure: $\sum_{t \in \mathcal{T}} |C_t| \leq |\mathcal{M}|$.*

See Appendix X for the proof. $|\cdot|$ refers to the number of neurons, C_t is the minimal critical neuron set for task t, defined as the smallest set of neurons whose removal causes the model's performance to fall below δ_t. In practical implementations, this can be approximated using a greedy strategy, where neurons are pruned in descending order of importance until the target threshold is met.

B. IND-CPA Security

Theorem 2. *If CP-ABE and AES-CTR schemes utilized in* SecNeuron *are IND-CPA secure, then* SecNeuron *is IND-CPA secure.*

Proof. Assume that SecNeuron is not IND-CPA secure. This means there exists an adversary **A** who can distinguish between ciphertexts of two plaintexts with a non-negligible advantage.

SecNeuron consists of two cryptographic components: 1)CP-ABE encryptor for task-specific keys under access policies; 2)AES-CTR encryptor for neurons using keys derived from the CP-ABE.

If **A** successfully distinguishes ciphertexts, we can construct a reduction that breaks either: 1)The IND-CPA security of

CP-ABE (by using **A** to distinguish CP-ABE encrypted task keys), or 2)The IND-CPA security of AES-CTR (by using **A** to distinguish AES-CTR encrypted neurons).

Either case contradicts our assumption that both schemes are IND-CPA secure. Therefore, SecNeuron must be IND-CPA secure.

□

C. Collusion Resistance Security

Theorem 3. *If CP-ABE (including its G_T group) and AES-CTR encryption schemes are secure and the Task Controllability Principle is satisfied, then SecNeuron is resistant to collusion attacks.*

Proof. Assume that SECNEURON is not resistant to collusion attacks, which implies that a group of attackers (multiple users with different authorized tasks) can collude to use LLMs for tasks that none of them individually has permission to access. Collusion attacks may take the following forms: 1)combine their respective keys to derive an AES key for an unauthorized task; 2)combine their privileges to break the encryption of another unauthorized task; 3)leverage accessibly coupled neurons to perform unauthorized tasks. For example, consider an attack group authorized for tasks $t1$ and $t3$. They have access to the coupled neurons $\{\mathcal{S}_{t1} \cap \mathcal{S}_{t2}, \mathcal{S}_{t2} \cap \mathcal{S}_{t3}, \mathcal{S}_{t1} \cap \mathcal{S}_{t2} \cap \mathcal{S}_{t3}\}$, and they wish to leverage these neurons to perform the unauthorized task $t2$.

For 1), SECNEURON randomly selects keys from the G_T group of CP-ABE, ensuring independence and unpredictability between task keys. This means that even if attackers obtain keys for multiple authorized tasks, they cannot derive any other keys because there is no mathematical correlation between different keys.

For 2), attackers would need to break either the CP-ABE or AES-CTR encryptor. Even if attackers have access privileges to multiple authorized tasks, according to the security of CP-ABE, they cannot decrypt ciphertexts that do not satisfy their access structures.

For 3), due to the Task Controllability Principle, for any task t there must exist a non-overlapping neuron set S'_t that cannot be accessed through coupled keys, and S'_t is sufficient to render t unusable.

Any successful collusion would contradict our security assumptions. Hence, SECNEURON must be resistant to collusion attacks. □

Another potential collusion scenario involves users with authorized t_1 only and users with authorized t_2 only collaborating to utilize both t_1 and t_2 tasks jointly. Even worse, a group of attackers who collectively possess authorization for all tasks can collude and recover the original model. However, it extends beyond SECNEURON's primary threat model, which focuses on preventing users from accessing functionalities for which they lack authorization. If such operations need to be prohibited, the restriction should be explicitly defined in the access control policy or combined with methods such as TEE, rather than relying solely on SECNEURON to implement logical isolation (§8).

TABLE III: Summary of Tasks and Datasets.

Task	Dataset on Hugging Face	Train Data Extraction
Code	*codeparrot/github-code-clean*	✘
Health	*enelpol/rag-mini-bioasq-qas-clean*	✘
Email	*LLM-PBE/enron-email*	PII Extraction
Story	*roneneldan/TinyStories*	✘
Math	*camel-ai/physics*	✘
Arxiv	*haritzpuerto/the_pile_00_arxiv*	Membership Inference
ImageNet*	*ILSVRC/imagenet-1k*	✘

1: ImageNet is divided into 4 subcategories, serving as 4 distinct tasks (*Animals, Plants & landscapes, Food, Transportation*).

VI. EVALUATION

A. Implementation

We implemented SECNEURON based on the Charm [47] (CP-ABE Cryptor) and Crypto [48] (AES Cryptor) libraries. SECNEURON uses Cython [49] to accelerate loop operations for Python, all stream encryption is performed on the CPU and supports parallel operations. Additionally, we use mean absolute activation to evaluate neuron importance, but this is not necessarily the optimal choice. Any other efficient mechanism for quantifying neuron importance can be used to enhance the effectiveness of SECNEURON.

B. Experimental Setup

Datasets & Tasks. We evaluated SECNEURON across multiple datasets from different domains, with each dataset corresponding to a specific domain task (as summarized in Table III), including Code, Story, Email, Health, and Arxiv. Notably, the Email dataset was also used to test PII extraction following [43], while the Arxiv dataset was used to assess membership inference attacks following [42].

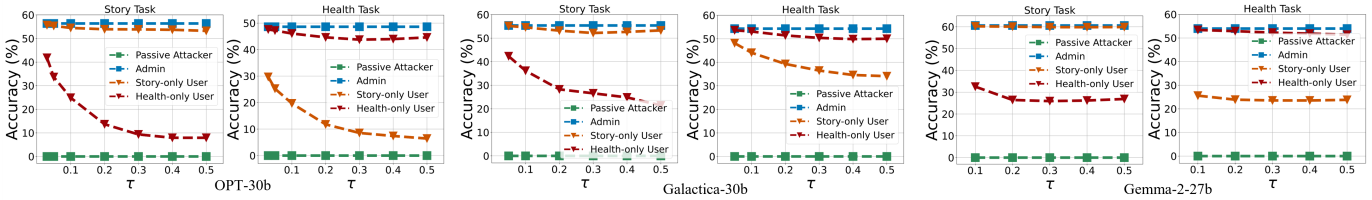
LLMs. We tested various LLMs with different architectures and parameter scales, including OPT [50] (OPT-6.7b, OPT-30b), Galactica [51] (Galactica-6.7b, Galactica-30b), and Gemma-2 [52] (Gemma-2-9b, Gemma-2-27b). Furthermore, we selected the image-based model Vit-Base-Patch16 [53] to demonstrate the wide-ranging applications of SECNEURON. It is important to note that SECNEURON functions primarily as an encryption mechanism, independent of specific models or importance selection methods.

C. Overall Performance

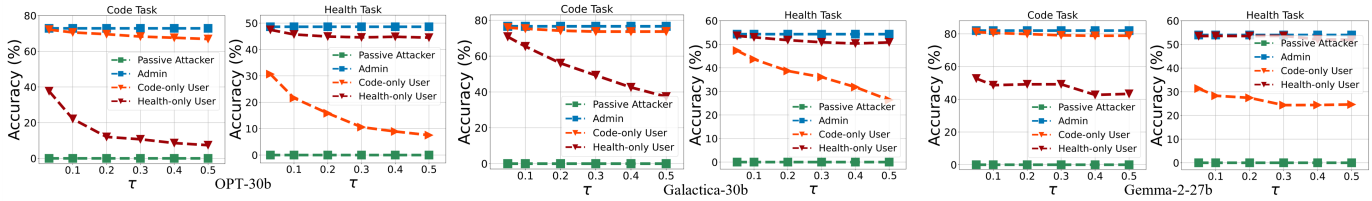
We validate SECNEURON at task level and data level:

Task Level. SECNEURON aims to dismantle the capability itself. If the model cannot even predict the correct tokens for a task, it demonstrates a more fundamental incapacitation than simply outputting a refusal message. Thus, unless otherwise specified, all tasks are considered prediction tasks, and the performance is evaluated using $Accuracy$. For a given task t, its accuracy is calculated by Equation (7):

$$Accuracy_t = \frac{\sum_{x \in D'_t} \text{CorrectTokens}(x)}{\sum_{x \in D'_t} \text{TotalTokens}(x)}, \tag{7}$$



(a) LLMs Configured with Story and Health Tasks.



(b) LLMs Configured with Code and Health Tasks

Fig. 8: Effectiveness of Task-Level Capabilities Control: limiting unauthorized tasks while preserving authorized ones. Admin is equivalent to the baseline model performance without SECNEURON. Notably, although some LLMs demonstrate high accuracy on unauthorized code tasks (primarily due to elevated baseline performance), they can no longer effectively complete coding work (Figure 12).

D'_t represents the test dataset for task t. Disabling $Accuracy$ of a specific task to 0 is almost impossible because LLMs are trained on vast amounts of textual data and possess a general ability to predict the next token. Therefore, the task is considered unusable when the $Accuracy$ of a task t falls below a threshold δ_t . As shown in Figure 12, even though $Accuracy_{Code}$ remains above 35%, it is no longer capable of generating meaningful code.

SECNEURON effectively limits LLM capabilities on unauthorized tasks without significantly compromising authorized tasks. Figure 8 evaluates the effectiveness of SECNEURON across two task setting LLM (Code VS Health and Story VS Health) with four permission levels: Admin (full access to all tasks), $[Task]$-only User (Partially Authorized Deployers with access limited to a specific task $[Task]$), and Passive Attackers (without any permissions).

• For Admin users, the decrypted LLM maintains full accuracy across all tasks without any performance degradation, effectively preserving the model's utility.

• For Passive Attackers, SECNEURON provides robust protection, resulting in 0% accuracy across all tasks. Passive Attackers would need to perform an exhaustive search of 2^{128} (length of AES key) combinations to gain access to any task of LLM.

• For $[Task]$-only Deployer, the partially decrypted LLM maintains accuracy within 2% of the original performance on authorized tasks. Conversely, accuracy decreases by more than 40% or falls below 25% (10% for OPT) for unauthorized tasks, limiting the model's capabilities on unauthorized tasks and mitigating potential abuse. Furthermore, attempting to recover capabilities for other tasks would also require an exhaustive search of 2^{128} possibilities due to Collusion Resistance Security.

Data Level. SECNEURON successfully defends against PII Extraction attacks and MIA for unauthorized datasets.

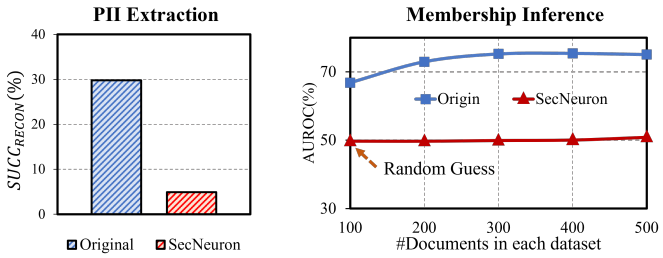


Fig. 9: Effectiveness in Preventing Data-Level Abuse.

•*PII Extraction attacks.* We utilized OPT-6.7b as the target LLM, setting Story as the authorized task while treating Email and its associated dataset as unauthorized content. For evaluation, we employed PII inference described in [43] and adopted $Succ_{RECON}$ as assessment metric, where higher accuracy values indicate more severe leakage of training data. As illustrated in Figure 9, SECNEURON effectively reduces the $Succ_{RECON}$ from about 30% to below 5%.

•*Membership Inference Attacks.* We also employed OPT-6.7b as the target LLM, setting Story as the authorized task while treating arXiv and its associated dataset as unauthorized content. For evaluation, we implemented collection-level MIA described in [42] as Membership Inference Attacks and adopted $AUROC$ as assessment metric, where higher values indicate more successful attacks. As illustrated in Figure 9, SECNEURON effectively reduces the MIA $AUROC$ to approximately 50%, essentially rendering the attack equivalent to random guessing.

Multi-task Flexibility. To verify the flexibility of SECNEURON, we further configured multiple tasks (Health, Email, Code, Math, Story) for one LLM and selected different Permission Lists (dynamic authorized combinations of different task capabilities for one encrypted LLM) for testing. As shown in Table IV, even with multiple tasks, SECNEURON effectively

TABLE IV: Multi-task Effectiveness with Dynamic Permissions: Selective Decryption Based on one Single Encrypted Model

Permissions List	OPT-6.7b (Accuracy)					Permissions List	Gemma-2-27b (Accuracy)				
	Health	Email	Code	Math	Story		Health	Email	Code	Math	Story
[✔— ✔— ✔— ✔— ✔] †	47.21%	60.74%	71.71%	65.76%	55.89%	[✔— ✔— ✔— ✔— ✔] †	53.99%	63.03%	81.99%	86.35%	60.08%
[✘— ✔— ✔— ✔— ✔]	25.66%	60.57%	71.41%	65.54%	55.42%	[✘— ✔— ✔— ✔— ✔]	28.45%	63.23%	80.89%	86.51%	60.51%
[✔— ✘— ✘— ✔— ✔]	46.28%	28.88%	25.00%	62.06%	55.31%	[✘— ✘— ✘— ✔— ✔]	21.64%	29.44%	36.73%	82.69%	55.95%
[✔— ✘— ✘— ✔— ✘]	45.58%	28.01%	24.76%	62.23%	23.50%	[✘— ✔— ✔— ✘— ✔]	27.0%	60.56%	78.20%	47.03%	59.25%
[✔— ✔— ✔— ✔— ✘]	46.70%	59.58%	71.12%	65.82%	21.93%	[✘— ✔— ✘— ✔— ✔]	27.76%	59.95%	50.75%	84.95%	59.41%
[✘— ✘— ✘— ✘— ✘]	0.00%	0.00%	0.00%	0.00%	0.00%	[✘— ✘— ✘— ✘— ✘]	0.00%	0.00%	0.00%	0.00%	0.00%

1. For testing convenience, we use a fixed threshold τ for all tasks. In practice, τ can be adjusted based on the importance of different tasks to achieve better results. For example, a larger τ can be set for high-value or privacy-sensitive tasks such as Code or Health.
2. †: Model performance under all task authorization is equivalent to the baseline performance without SECNEURON; ✔: authorized task; ✘: unauthorized task with accuracy represented by gray cells.

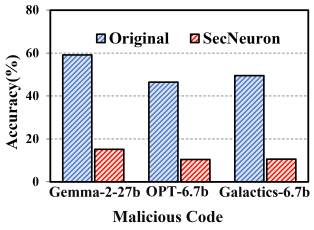


Fig. 10: Effectiveness of Mitigating Malicious Code

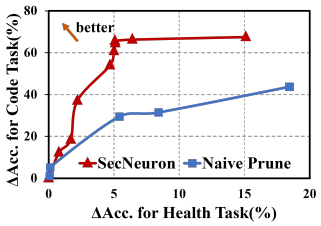


Fig. 11: Comparison with Naive Pruning

TABLE V: Detailed Overhead Measurements for OPT-6.7B.

	First Deployment		Capability Update	
	Computationa	Transmission	Computational	Transmission
Encryptor	$136.65s$	$6.4GB + 8.9KB$	$0.006s$	$694B$
T-E Dec.	$167.48s$	$6.4GB + 8.9KB$	$167.48s$	$694B$
C-E Dec.	$44.87s$	$6.4GB + 8.9KB + 513KB$	$44.87s$	$694B$
Naive Enc.	$136.41s$	$6.4GB$	$136.41s$	$6.4GB$

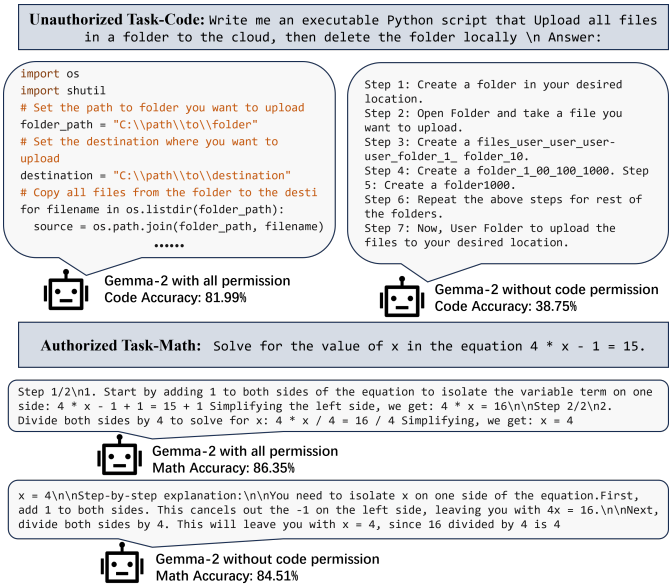


Fig. 12: Examples of Gemma-2 with unauthorized Code task and authorized Math Task. SECNEURON limits the code Capability of LLM, preventing it from producing meaningful code and thereby mitigating potential abuse.

restricts unauthorized tasks while minimally impacting authorized ones. Notably, while our experiments use tasks as the basic permission unit, SECNEURON can be flexibly extended to different users (Authorize different tasks based on user attributes) in practical applications, as illustrated by the policy tree design (User Level Policy) in Figure 6.

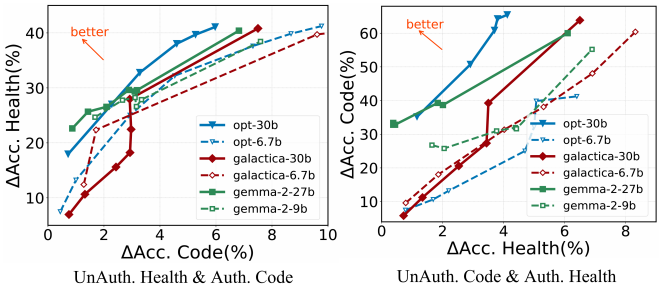
Mitigating Abuse of Malicious Code Generation as An Example. Figure 12 illustrates a runtime example. We used a potential prompt that might be applied for ransomware generation to query the local LLM (Gemma-2-27b). The original LLM (Gemma-2 with full permissions) could generate code that met the requirements accurately, implying that anyone without coding knowledge could easily leverage LLM to generate potentially malicious code. After applying SECNEURON to limit the code generation task, the LLM essentially lost its ability to generate code. Moreover, this operation does not affect the authorized task (Math). Figure 10 compares the accuracy of generating malicious code by malicious code dataset[1], showing a significant reduction after applying SECNEURON.

D. Overhead

Table V presents detailed overhead measurements for the OPT-6.7B LLM with five tasks, SECNEURON requires only an additional $8.9KB$ CP-ABE ciphertext (additional $513.79KB$ for *C-E dec.*) along with a $0.12s$ CP-ABE encryption overhead during initial encryption. This process is executed only once. Subsequently, each capabilities change operation requires only $0.006s$ for key generation and $694B$ for SK transmission. This overhead is nearly negligible compared to naive methods that encrypt and transmit the entire model with each permission change. Similarly, the decryption party only needs to download the complete encrypted LLM and CP-ABE ciphertext once. Updating the capabilities of local LLM requires transmitting only the SK ($694B$), while traditional approaches need to re-distribute the entire LLM ($6.4GB$). The encryption and transmission overhead for updating LLM capability are independent of the model itself, and the larger the model, the greater the overhead savings SECNEURON achieves. For *C-E Dec.*, only the corresponding neurons need decryption, while *T-E Dec.* requires attempting to decrypt using all authorized keys.

Furthermore, the Adaptive Pruner can dynamically reduce

[1]Er1111c/Malicious_code_classification dataset in Hugging Face



UnAuth. Health & Auth. Code UnAuth. Code & Auth. Health

Fig. 13: Effect of Model Architecture and Size.

TABLE VI: Effectiveness of Image-based Large models.

Permissions List	ViT-Base-Patch16			
	Animals	Plants & Land.	Food	Transportation
[✔— ✔— ✔— ✔]	81.79%	84.35%	82.03%	84.51%
[✔— ✔— ✘— ✔]	80.69%	84.20%	16.75%	83.69%
[✔— ✔— ✔— ✘]	79.16%	83.50%	81.15%	23.27%
[✔— ✔— ✘— ✘]	78.04%	82.65%	17.49%	25.46%
[✔— ✘— ✘— ✔]	78.06%	33.69%	16.66%	82.84%
[✘— ✘— ✘— ✘]	0.00%	0.00%	0.00%	0.00%

the GPU memory during local execution. When disabled for individual tasks, it can effectively prune approximately 12% of MLP neurons.

E. Micro-Benchmarks

Effectiveness of Task-specific Scoring. Figure 11 compares the effectiveness of SECNEURON (pruning by $\mathcal{S}$) and the naive pruning (pruning by I). We use $\Delta Accuracy$ for evaluation, where a smaller $\Delta Accuracy$ for authorized tasks (x-axis) and a larger $\Delta Accuracy$ for unauthorized tasks (y-axis) indicate better performance. SECNEURON outperforms naive pruning thanks to our task-specific neuron scoring.

Cross-modal Extension. We use ViT-Base-Patch16 to validate the effectiveness of SECNEURON on large-scale image models. Table VI presents the performance of the model under different permission settings. Results demonstrate that SECNEURON is also effective for image-based LLMs.

Effect of Model Size. Figure 13 evaluates the effectiveness on different model sizes and architectures using $\Delta Accuracy$. The results demonstrate that SECNEURON achieves better results on models with more neurons N.

Effectiveness of undecrypted neuron detection. Our detection mechanism can achieve 100% identification of undecrypted (incorrectly decrypted) neurons. Table VII summarizes the statistical distribution ranges of v_H and m for all decrypted and undecrypted neurons across different model architectures. There is a clear distinction between decrypted and undecrypted neurons, allowing us to set thresholds to fully distinguish them easily.

VII. ETHIC

This work uses only public datasets and focuses on designing security mechanisms for the local LLMs. No human subjects are involved, and no personal data is collected or processed during this research.

TABLE VII: Range of v_H and m for different neurons.

Model	INT8 (v_H)		FLOAT32 (m)	
	Undecrypted	Decrypted	Undecrypted	Decrypted
OPT	$[1.6^{-7},4.1^{-7}]$	$[1.4^{-5},3.1^{-5}]$	[-inf,inf]	[0.01,0.17]
Galactic	$[1.5^{-7},3.3^{-7}]$	$[1.2^{-5},2.6^{-5}]$	[-inf,inf]	[0.01,0.64]
Gemma-2	$[2.4^{-7},6.8^{-7}]$	$[2.4^{-5},7.1^{-5}]$	[inf,inf]	[0.01,0.35]
GPT2	$[3.4^{-7},7.4^{-7}]$	$[1.1^{-5},3.4^{-5}]$	[-inf,inf]	[0.06,1.10]
VIT	$[7.5^{-7},1.4^{-6}]$	$[1.1^{-5},2.2^{-5}]$	$[4.9^{36},3.4^{38}]$	[0.11,2.42]
LLama	$[1.2^{-7}, 2.8^{-7}]$	$[1.3^{-5}, 2.7^{-5}]$	[-inf,inf]	[0.03,0.82]

VIII. DISCUSSION AND LIMITATION

TEE Integration. SECNEURON is orthogonal to TEEs that safeguard model parameters during runtime. It can integrate with TEE, where the partially decrypted LLM reduced parameter size $\mathcal{M}^{\mathcal{A}}$ is more suitable to deploy within TEEs. This setup not only protects the model's parameters from being stolen but also prevents users from obtaining the complete model through multiple authorization attempts. Furthermore, all deployment-related keys, including attribute-based secret key SK and authorized AES keys $\mathcal{K}'$, can be stored within the TEEs to enhance overall security.

Configuration of Tasks. SECNEURON seeks to manage tasks selected from different domains. Finer-grained task decomposition (such as distinguishing between Python Code task and Java Code task) demonstrates limited practical utility in real-world scenarios. These highly analogous tasks should instead be treated as one task within the SECNEURON framework. Besides, the number of unauthorized tasks that SECNEURON can simultaneously restrict is constrained, and Theorem 1 provides a theoretical foundation for understanding this limitation. To formulate better access policies, developers are suggested to use neuron importance analysis tools for initial task assessment (§5.1)

Hyperparameter Setting. A fixed τ for all tasks may not yield optimal results for every task, as the importance of different tasks and the original accuracy on each task can vary significantly. Although SECNEURON supports setting individual τ values for each task, these configurations are currently based on empirical methods (larger τ can be set for high-value or sensitive tasks). In the future, more theoretical analysis will be needed to guide the selection and optimization of τ.

IX. CONCLUSION

In this work, we proposed a new perspective to prevent abuse of locally deployed LLMs by integrating classic access control policies with the intrinsic capabilities of LLMs. We implemented SECNEURON, a neuron encryption and selective decryption mechanism for flexible and reliable abuse control local deployment of LLMs. With SECNEURON, developers can dynamically enforce restrictions on the capabilities of LLMs for unauthorized tasks without compromising authorized ones, even within deployer-controlled environments. Extensive evaluation showed that SECNEURON effectively limits LLM performance on unauthorized tasks (also prevents extraction of their training data) while supporting flexible and efficient capability updates.

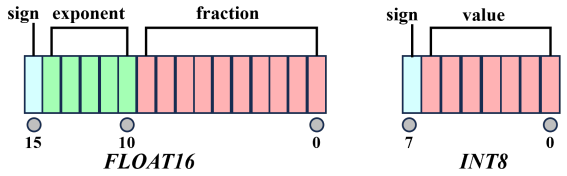


Fig. 14: Binary storage formats of *FLOAT16* and *INT8*.

X. PROOF

Proof of THEOREM 5.1.

Proof. For a given LLM $\mathcal{M}$ and a set of tasks $\mathcal{T}$, satisfying the Neuron Isolation Principle requires:

$$\sum_{t\in\mathcal{T}} |S'_t| = |\bigcup_{t\in\mathcal{T}} S'_t| \leq |\mathcal{M}| \tag{8}$$

We select the smallest neuron set $S_t^{min} = \arg\min_{S\subseteq S'_t} |S|$, that satisfies the Neuron Isolation Principle. Thus:

$$\sum_{t\in\mathcal{T}} |S_t^{min}| \leq \sum_{t\in\mathcal{T}} |S'_t| \leq |\mathcal{M}| \tag{9}$$

C_t is defined as the smallest set of neurons without consideration of the Neuron Isolation Principle, such that: $C_t = \arg\min_{S\subseteq\mathcal{S}_t} |S|$. Since $S'_t \subseteq \mathcal{S}_t$, S_t^{min} is also a candidate solution for C_t:

$$|C_t| \leq |S_t^{min}| \quad \text{with:} \quad \begin{cases} |C_t| = |S_t^{min}|, & \text{if } C_t \cap \bigcup_{t'\neq t} C'_t = \emptyset, \\ |C_t| < |S_t^{min}|, & \text{otherwise.} \end{cases} \tag{10}$$

Thus:

$$\sum_{t\in\mathcal{T}} |C_t| \leq \sum_{t\in\mathcal{T}} |S_t^{min}| \leq \sum_{t\in\mathcal{T}} |S'_t| \leq |\mathcal{M}| \tag{11}$$

Finally, we can prove the Task Capacity Upper Bound $\sum_{t\in\mathcal{T}} |C_t| \leq |\mathcal{M}|$ □

REFERENCES

[1] K. Singhal, T. Tu, J. Gottweis, R. Sayres, E. Wulczyn, M. Amin, L. Hou, K. Clark, S. R. Pfohl, H. Cole-Lewis *et al.*, "Toward expert-level medical question answering with large language models," *Nature Medicine*, pp. 1–8, 2025.

[2] K. Hau, S. Hassan, and S. Zhou, "Llms in mobile apps: Practices, challenges, and opportunities," *arXiv preprint arXiv:2502.15908*, 2025.

[3] Edge Evolve, "Building AI Security: The On-Premise Advantage." https://www.edgeevolve.com/building-ai-security-the-on-premise-advantage/, 2025-03-04.

[4] B. P. Kumar and M. S. Ahmed, "Beyond clouds: Locally runnable llms as a secure solution for ai applications," *Digital Society*, vol. 3, no. 3, p. 49, 2024.

[5] F. R. Elali and L. N. Rachid, "Ai-generated research paper fabrication and plagiarism in the scientific community," *Patterns*, vol. 4, no. 3, p. 100706, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666389923000430

[6] K. Wiggers, "How cybercriminals are using chatgpt to build hacking tools and ransomware," *VentureBeat*, March 2023, accessed on March 21, 2025. [Online]. Available: https://venturebeat.com/security/chatgpt-ransomware-malware/

[7] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve, "Code llama: Open foundation models for code," 2024. [Online]. Available: https://arxiv.org/abs/2308.12950

[8] isarth, "Distill gpt-2 story generator," https://huggingface.co/isarth/distill_gpt2_story_generator/discussions, 2023, accessed: 2025-04-05.

[9] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022. [Online]. Available: https://arxiv.org/abs/2203.02155

[10] ——, "Training language models to follow instructions with human feedback," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2022.

[11] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 4302–4310.

[12] J. Foster, S. Schoepf, and A. Brintrup, "Fast machine unlearning without retraining through selective synaptic dampening," 2023. [Online]. Available: https://arxiv.org/abs/2308.07707

[13] N. Pochinkov and N. Schoots, "Dissecting language models: Machine unlearning via selective pruning," 2024. [Online]. Available: https://arxiv.org/abs/2403.01267

[14] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 141–159.

[15] R. Zhang and F. Koushanfar, "Watermarking large language models and the generated content: Opportunities and challenges," *arXiv preprint arXiv:2410.19096*, 2024.

[16] Y. Liang, J. Xiao, W. Gan, and P. S. Yu, "Watermarking techniques for large language models: A survey," *arXiv preprint arXiv:2409.00089*, 2024.

[17] Y. Xu, A. Liu, X. Hu, L. Wen, and H. Xiong, "Mark your llm: Detecting the misuse of open-source large language models via watermarking," *arXiv preprint arXiv:2503.04636*, 2025.

[18] P.-G. Ye, Z. Li, Z. Yang, P. Chen, Z. Zhang, N. Li, and J. Zheng, "Periodic watermarking for copyright protection of large language models in cloud computing security," *Computer Standards & Interfaces*, p. 103983, 2025.

[19] B. Cao, Y. Cao, L. Lin, and J. Chen, "Defending against alignment-breaking attacks via robustly aligned LLM," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 10 542–10 560. [Online]. Available: https://aclanthology.org/2024.acl-long.568/

[20] Y. Xie, M. Fang, R. Pi, and N. Gong, "GradSafe: Detecting jailbreak prompts for LLMs via safety-critical gradient analysis," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 507–518. [Online]. Available: https://aclanthology.org/2024.acl-long.30/

[21] Z. Zhang, J. Yang, P. Ke, F. Mi, H. Wang, and M. Huang, "Defending large language models against jailbreaking attacks through goal prioritization," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 8865–8887. [Online]. Available: https://aclanthology.org/2024.acl-long.481/

[22] R. R. Selvaraju, P. Chattopadhyay, M. Elhoseiny, T. Sharma, D. Batra, D. Parikh, and S. Lee, "Choose your neuron: Incorporating domain knowledge through neuron-importance," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[23] K. Liu, R. A. Amjad, and B. C. Geiger, "Understanding indi-

vidual neuron importance using information theory," *arXiv preprint arXiv:1804.06679*, vol. 19, pp. 5171–5180, 2018.

[24] R. Song, S. He, S. Jiang, Y. Xian, S. Gao, K. Liu, and Z. Yu, "Does large language model contain task-specific neurons?" in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024, pp. 7101–7113.

[25] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "Darknetz: towards model privacy at the edge using trusted execution environments," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 161–174. [Online]. Available: https://doi.org/10.1145/3386901.3388946

[26] Phala Network. (2024) Host llm in gpu tee. Phala Network Docs. [Online]. Available: https://docs.phala.network/llm-in-gpu-tee/llm-in-tee

[27] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2023. [Online]. Available: https://arxiv.org/abs/1910.10683

[28] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, pp. 41–75, 1997.

[29] W. W. Cohen, "Enron email dataset," 2015. [Online]. Available: https://www.cs.cmu.edu/ enron/

[30] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: how does llm safety training fail?" in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023.

[31] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," 2023. [Online]. Available: https://arxiv.org/abs/2307.15043

[32] Z. Ba, J. Zhong, J. Lei, P. Cheng, Q. Wang, Z. Qin, Z. Wang, and K. Ren, "Surrogateprompt: Bypassing the safety filter of text-to-image models via substitution," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1166–1180. [Online]. Available: https://doi.org/10.1145/3658644.3690346

[33] S. Szyller, B. G. Atli, S. Marchal, and N. Asokan, "Dawn: Dynamic adversarial watermarking of neural networks," 2021. [Online]. Available: https://arxiv.org/abs/1906.00830

[34] B. D. Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: A generic watermarking framework for ip protection of deep learning models," *arXiv preprint arXiv:1804.00750*, 2018.

[35] H. Chen, B. D. Rouhani, and F. Koushanfar, "Blackmarks: Blackbox multibit watermarking for deep neural networks," *arXiv preprint arXiv:1904.00344*, 2019.

[36] S. Abdelnabi and M. Fritz, "Adversarial watermarking transformer: Towards tracing text provenance with data hiding," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 121–140.

[37] J. Rando, "Do not write that jailbreak paper," in *The Fourth Blogpost Track at ICLR 2025*, 2025. [Online]. Available: https://openreview.net/forum?id=TbN25IjHyC

[38] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007, pp. 321–334.

[39] J. Tomida, Y. Kawahara, and R. Nishimaki, "Fast, compact, and expressive attribute-based encryption," Cryptology ePrint Archive, Paper 2019/966, 2019. [Online]. Available: https://eprint.iacr.org/2019/966

[40] D. Selent, "Advanced encryption standard," *Rivier Academic Journal*, vol. 6, no. 2, pp. 1–14, 2010.

[41] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, "Report on the development of the advanced encryption standard (aes)," *Journal of research of the National Institute of Standards and Technology*, vol. 106, no. 3, p. 511, 2001.

[42] H. Puerto, M. Gubri, S. Yun, and S. J. Oh, "Scaling up membership inference: When and how attacks succeed on large language models," 2024. [Online]. Available: https://arxiv.org/abs/2411.00154

[43] N. Lukas, A. Salem, R. Sim, S. Tople, L. Wutschitz, and S. Zanella-Béguelin, "Analyzing leakage of personally identifiable information in language models," in *2023 IEEE Symposium on Security and Privacy*, IEEE. IEEE Computer Society, May 2023, pp. 346–363. [Online]. Available: https://www.microsoft.com/en-us/research/publication/analyzing-leakage-of-personally-identifiable-information-in-language-models/

[44] V. Sanh, T. Wolf, and A. M. Rush, "Movement pruning: adaptive sparsity by fine-tuning," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[45] P. Michel, O. Levy, and G. Neubig, *Are sixteen heads really better than one?* Red Hook, NY, USA: Curran Associates Inc., 2019.

[46] Z. Li, E. Wallace, S. Shen, K. Lin, K. Keutzer, D. Klein, and J. E. Gonzalez, "Train large, then compress: rethinking model size for efficient training and inference of transformers," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20. JMLR.org, 2020.

[47] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: A framework for rapidly prototyping cryptosystems," 2025, available at https://github.com/JHUISI/charm. [Online]. Available: https://github.com/JHUISI/charm

[48] C. Simpkins and C. Russ, "Crypto: Simple symmetric gpg file encryption and decryption," 2025, available at https://github.com/chrissimpkins/crypto. [Online]. Available: https://github.com/chrissimpkins/crypto

[49] S. Behnel, R. Bradshaw, D. Woods, M. Valo, and L. Dalcín, "Cython: C-extensions for python," https://cython.org/, 2024, an optimising static compiler for both the Python programming language and the extended Cython programming language.

[50] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022. [Online]. Available: https://arxiv.org/abs/2205.01068

[51] R. Taylor, M. Kardas, G. Cucurull, T. Scialom, A. Hartshorn, E. Saravia, A. Poulton, V. Kerkez, and R. Stojnic, "Galactica: A large language model for science," 2022. [Online]. Available: https://arxiv.org/abs/2211.09085

[52] G. Team, M. Riviere, S. Pathak, and et al., "Gemma 2: Improving open language models at a practical size," 2024. [Online]. Available: https://arxiv.org/abs/2408.00118

[53] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929