SLIP: Securing LLM's IP Using Weights Decomposition

Anonymous authors
Paper under double-blind review

Abstract

Large language models (LLMs) have recently seen widespread adoption, in both academia and industry. As these models grow, they become valuable intellectual property (IP), reflecting enormous investments by their owners. Moreover, the high cost of cloud-based deployment has driven interest towards deployment to edge devices, yet this risks exposing valuable parameters to theft and unauthorized use. Current methods to protect models' IP on the edge have limitations in terms of practicality, loss in accuracy, or suitability to requirements. In this paper, we introduce a novel hybrid inference algorithm, named SLIP, designed to protect edge-deployed models from theft. SLIP is the first hybrid protocol that is both practical for real-world applications and provably IP-preserving, while having zero accuracy degradation and minimal impact on latency. It involves partitioning the model between two computing resources, one secure but expensive, and another cost-effective but untrusted. This is achieved through matrix decomposition, ensuring that the secure resource retains a maximally sensitive portion of the model's IP while performing a minimal amount of computations, and vice versa for the untrusted resource. Importantly, the protocol includes guarantees that prevent attackers from exploiting the partition to infer the model weights. Finally, we present experimental results that show the robustness and effectiveness of our method, positioning it as a compelling solution for protecting LLMs.

1 Introduction

Large Language Models (LLMs) are increasingly being commercialized due to their superior performance and vast applications Zhao et al. (2023). Alongside this trend, many researchers have been tackling challenges related to their deployment – balancing costs, latency, performance and security. Companies which develop LLMs invest substantial capital to create these models. Not only in extensive computing resources for training, up to hundreds of millions of dollars Perrault & Clark (2024), but also on high-end talent, datasets and labeling, proprietary training procedures and unique network architectures. Consequently, the parameters of a pre-trained foundational model can be considered *highly valuable* intellectual property (IP) for its owner, who would therefore be highly motivated to secure it from theft. Cloud services offer various security solutions against model theft and extraction attacks, however they often come with a high price tag with recent growth in demand Mok (2023). As a result, there is a growing financial incentive to offload as much computational workload as possible from secure computing resources, like the cloud, to more cost-effective, but *less secure* edge devices, such as servers and laptops. This motivates us to develop a security solution that protects the model owner's intellectual property, despite the security gap in edge devices.

Recently, *hybrid inference* has emerged as a framework that aims to address various challenges related to deployment of LLM's Chen & Ran (2019). In this framework, tokens are generated using two computing resources that differ in their properties. Most strategies focus on cost and performance Ding et al. (2024); Bang et al. (2024); Kang et al. (2017), or on privacy Hou et al. (2023); Chaopeng et al. (2023); Rathee et al. (2020), rather than on model security. Few have aimed to protect model IP; For example, in Schlögl & Böhme (2020) they propose splitting the model to a feature extractor and task specific classifier, securing the final layers of deep neural networks (DNNs) in an enclave, but their approach degrades the model accuracy, supports feed-forward DNNs only, requires model retraining, and does not have provable guarantees against information leakage. In contrast, Zhou et al. (2023) use RL for model splitting, but their obfuscated parameters are inefficiently spread across the model, and they lack security guarantees against model restoration through observing inferences. Finally, Li et al. (2024) employ weight permutation and an authorization module on a trusted execution environment (TEE), but their approach is vulnerable to hardware attacks, impractical for consumer devices due to the cost of TEEs, and not provably secure.

In this paper, we propose SLIP, a novel *hybrid inference* approach to IP protection of the model that uses a cheap and low security computing resource in tandem with an expensive but secure resource. SLIP balances budget constraints, latency and security standards, but has no loss in model accuracy. We achieve this by offloading the majority of computations to the cost-effective but less secure computing resource, while protecting the most valuable yet computationally undemanding information on the secure but expensive resource. Moreover, we demonstrate that SLIP is applicable to various DNN architectures, including fully-connected multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), and transformers.

1.1 Our contributions

We make the following contributions,

1. We propose a **model decomposition** technique that strategically selects weight matrices across the model (as observed in Meng et al. (2024); Wang et al. (2024); Sharma et al. (2023)), and identifies the components within these matrices that embody the model IP, and safeguards them on the secure resource. Consequently, the remainder of the model contains most of the computations, but is essentially useless for an attacker aiming to steal the model IP.

2. Since low and high security computing resources participate in each model inference, it incurs additional latency and introduces new potential attacks that could exploit the communication between them. Thus, our paper introduces an **inference protocol** to mitigate these security risks in hybrid inference. The protocol does not reveal the secure resource's intermediate outputs to the low-security resource, by **masking** these intermediate outputs with precomputed **uniformly distributed and input-independent** noise. Later, we efficiently cancel out this mask's effect without degradation in accuracy. We provide **theoretical proofs** for the correctness of the protocol and IP-preservation of our approach, and in a companion paper Jain et al. (2025) provide even more formal definitions and proofs focusing on the security aspects of of the protocol.

3. We give **experimental evidence** for our method on LLMs. We show that our decomposition exposes a portion of the model with most of the computations but insignificant utility for an attacker aiming to steal the model IP. Moreover, we demonstrate that even when the attacker fine-tunes the exposed portion of the model in an attempt to restore its original performance, the resulting model is very weak compared to the original model, and the smaller the exposed portion, the weaker the resulting model.

1.2 Background and related works

Threat Model. In our setting, we aim to protect from an attacker who aims to obtain the parameters, i.e., the IP Oliynyk et al. (2023) of our model. We assume the attacker knows the model architecture, has full knowledge of our protocol and any associated hyper-parameters Petitcolas (2023), and has indefinite physical access and full admin privileges in the low-security compute resource.[1] Conversely, we assume that the secure resource is a trusted entity, inaccessible to the attacker except through the query API. Thus, our protocol focuses on protecting from an attacker who has access to the exposed model portion and attempts to restore the secret portion in the secure resource (or the complete model directly) through model restoration techniques Chen et al. (2023); Ma et al. (2023).

Model Extraction Attacks. Model *extraction* attacks are a specific type of security threat where attackers aim to create a duplicate or approximation of a machine learning model by using its public API (Birch et al. (2023); Shamir et al. (2023); Truong et al. (2021); Kariyappa et al. (2021) see Yao et al. (2024); Jagielski et al. (2020) for review). Attackers manipulate the model's input-output pairs to learn about its structure

[1]We treat this resource as if the attacker has possession of it, bypassed any existing security measures, has direct control over the exposed model parameters and can freely query the secure resource.

and functionality, allowing them to create an approximation of the model. This can enable them to use the model for their own purposes, sidestep any usage restrictions or fees, or even uncover sensitive information about the original training data. Hence, this type of attacks could compromise compute resources at any security level, whether the inference system is hybrid or not, since it exploits the intended use of the service. Note that we focus on extraction attacks rather than *stealing* Oliynyk et al. (2023), since our setting is low-security resource edge devices.

Protecting DNNs IP from Theft. Several existing methods aim to secure DNNs. One obvious approach is *Model Obfuscation*, the process of intentionally obscuring or hiding the sensitive information of a neural network while retaining its functionality. However, despite astonishing breakthroughs in recent years Jain et al. (2024), current solutions for cryptographically secure obfuscation either degrade model accuracy, or are extremely impractical even for simple functions Chen et al. (2018); Ye et al. (2017). Other approaches use knowledge distillation Xu et al. (2018), architecture obfuscation Zhou et al. (2022b), intentional model poisoning Grailoo et al. (2022), increase model sensitivity to parameters perturbation Szentannai et al. (2020), locate model tampering for user authorization Huang et al. (2023), or various other means Zhou et al. (2022a). Additionally, passive protection techniques, such as *watermarking* Uchida et al. (2017); Adi et al. (2018); Yan et al. (2023); Yang et al. (2021); Zhang et al. (2018) (see Li et al. (2021) for survey) or *Attestation* Chen et al. (2019), help verify ownership and copyrights, but they cannot effectively *prevent* unauthorized access and usage to demotivate attackers. Lastly, cryptographic techniques have been extensively applied in the area of Privacy Preserving Machine Learning (PPML). Such techniques include Homomorphic Encryption (e.g., Acar et al. (2018), Onoufriou et al. (2021); Lee et al. (2022)), Secure Multiparty Computation (e.g., Knott et al. (2021); Dalskov et al. (2020); Cheng et al. (2023); Liu et al. (2017); Gupta et al. (2023)), and Differential Privacy (e.g., Abadi et al. (2016)). However, in PPML, the emphasis is on user data privacy or training data rather than protecting model weights, and existing frameworks are not designed for this purpose, or do not account for the computational trade-offs between asymmetric parties with varying costs and security levels Rouhani et al. (2018); Juvekar et al. (2018); Huang et al. (2022); Rathee et al. (2020).

2 General Framework: Model Decomposition and Hybrid Inference Protocol

We now describe our framework for an IP-preserving model decomposition and hybrid inference, as detailed in Figure 1. Our goal is to offload most of the inference workload to a low security (but computationally efficient) resource, while ensuring that the intellectual property of the model remains secure, even through repeated inferences. To achieve this, we introduce a model decomposition scheme and a hybrid inference protocol. We consider two entities:

1. A *secure* resource controlled by $\mathcal{C}$harlie ('C' as in Cloud), which has limited compute power but is trusted to keep sensitive data confidential.

2. A *low security* resource controlled by $\mathcal{D}$avid ('D' as in Desktop), which is computationally powerful but fully exposed to the attacker.

We split the model parameters between these parties, then define a protocol for inference on any input that involves communication between them. The decomposition and protocol must jointly satisfy the desiderata of usefulness, safety and efficiency, which we formally define in Section 2.1, in order to ensure the whole solution preserves model IP and is efficient (Theorem 3).

Definition 1 (Model Decomposition). *Let ϕ_Θ be a model with parameters Θ. A* model decomposition *is a pair $(\phi_{\Theta_\mathcal{C}}, \phi_{\Theta_\mathcal{D}})$, where $\Theta = (\Theta_\mathcal{C}, \Theta_\mathcal{D})$ denotes the parameter partition assigned to Charlie and David, respectively.*

Definition 2 (Hybrid Inference Protocol). *Given a model decomposition $\Theta = (\Theta_\mathcal{C}, \Theta_\mathcal{D})$ where Charlie and David only have access their respective portion of the model, a* Hybrid Inference protocol $\Pi := \Pi(\Theta_\mathcal{C}, \Theta_\mathcal{D})$ *is an interactive protocol, that given an input x, jointly computes $\phi_\Theta(x)$.*

We now formalize what makes a good decomposition and protocol.

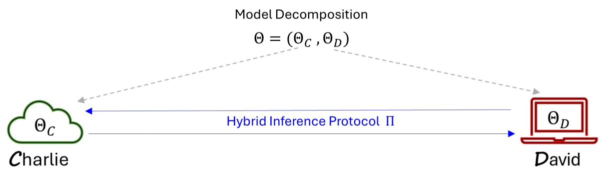
Model Decomposition
$\Theta = (\Theta_C, \Theta_D)$
Hybrid Inference Protocol Π
Θ_C
Charlie
Θ_D
David

Figure 1: The proposed framework, consisting of the Model Decomposition and the Hybrid Inference Protocol

2.1 What Makes a Good Decomposition and Protocol

In order to protect a model's intellectual property (IP) while enabling efficient inference, we define four key properties that a decomposition and hybrid inference protocol should satisfy: **IP-preservation** and **Efficiency**. As a vehicle to prove IP-preservation, we analyze additional properties called **Usefulness** and **Safety**. These properties form the foundation of our framework.

We begin with intuitive explanations and follow with formal definitions. **Usefulness** ensures the model IP isn't leaked through the David's portion of the model. **Safety** ensures that these input/output pairs are the major risk for David reconstructing the model (beyond David having his model portion). **IP-preservation** ensures that through the model decomposition, and through repeated hybrid inferences of the model, David does not learn the model IP. [2] **Efficiency** means the protocol runtime for Charlie is much quicker than running the full model inference.

We encompass model IP through the notion of risk. For a model $\mathcal{M}$ and a loss function ℓ, the model true risk is given by $\mathsf{risk}(\mathcal{M}) := \mathbb{E}_P\left[\ell(y, \mathcal{M}(x))\right]$, where $P(x, y)$ is the data distribution. We turn to the formal definitions.

Definition 3 (Usefulness)**.** *Let ϕ_Θ be the original model with parameters Θ, and let $\Theta = (\Theta_\mathcal{C}, \Theta_\mathcal{D})$ be its decomposition. For an attacker algorithm A (an adversarial David) that produces a model $A(\Theta_D)$ from Θ_D, define its usefulness to be:*

$$\kappa_A := \frac{\mathsf{risk}(\phi_\Theta)}{\mathsf{risk}(A(\Theta_D))}. \tag{1}$$

Note that lower κ_A indicates the risk of the reproduced model is high, i.e., less useful. Also, for the decomposition in Section 3, the best $A(\cdot)$ just naïvely returns Θ_D.

Definition 4 (Safety)**.** *We say a decomposition $\Theta = (\Theta_\mathcal{C}, \Theta_\mathcal{D})$ is* safe *if, for any Probabilistic Polynomial Time adversarial David A that uses $\Theta_\mathcal{D}$ and black box access to ϕ_Θ , there exists a Probabilistic Polynomial Time adversarial David B using only black box access to ϕ_Θ, such that B's reconstructed model has roughly the same risk as A's reconstructed model.*

Definition 5 (IP Preservation)**.** *We say a model decomposition and hybrid inference protocol preserves model IP, if An attacker cannot learn the model IP through the decomposition and repeated inference.*[3]

Definition 6 (Efficiency)**.** *Let $T_{\Pi,\mathcal{C}}(\Theta_\mathcal{C})$ be the runtime of Charlie during the execution of a hybrid inference protocol Π, and let $T_\mathcal{C}(\Theta)$ be the runtime of Charlie for the full model inference. For $\epsilon > 0$, we say Π is* ϵ-efficient *if:*

$$T_{\Pi,\mathcal{C}}(\Theta_\mathcal{C}) \;\leq\; \epsilon \cdot T_\mathcal{C}(\Theta). \tag{2}$$

In the following sections, we demonstrate how our proposed decomposition and protocol satisfy these properties both theoretically and empirically.

[2]Note that a black-box adversary which sees sufficiently many input-output pairs of the model can recover the model Θ Canales-Martínez et al. (2024); Carlini et al. (2024), and protecting from such attacks is a long term research challenge, So our definition of IP-preservation is such that the solution (decomposition and inference) does not give the attacker $\mathcal{D}$ much extra power beyond black-box access.

[3]Since with repeated inference in any possible inference scheme (even if the model was completely on the cloud), the attacker gains the respective model input/output pairs for those inferences, we only require that the attacker cannot learn the model IP beyond what they could learn in that setting, i.e., through performing a black-box model inversion attack, using the inferred model inputs and outputs.

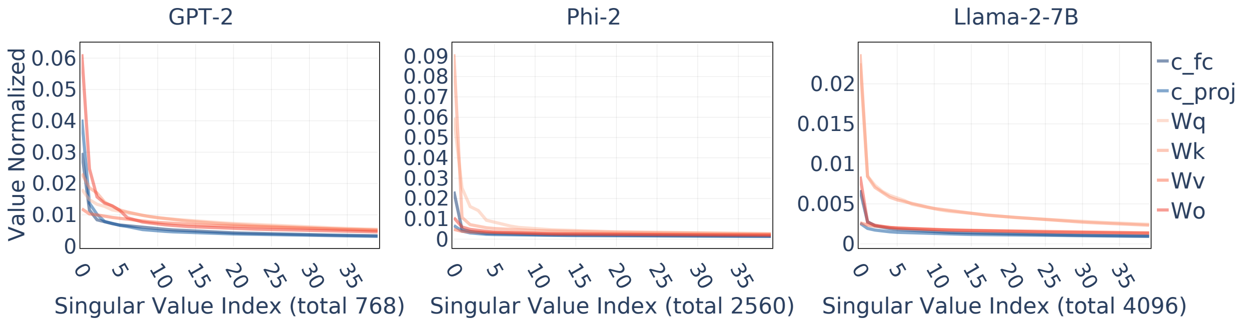

Figure 2: Singular values of the weight matrices of each model, quickly decreasing from largest to smallest.

2.2 Applying Our Framework to LLMs: A Roadmap

We now instantiate our framework to protect large language models (LLMs) during edge deployment. Our approach satisfies the desiderata introduced in Section 2.1 through two core components:

1. **Model Decomposition**: We identify sensitive layers in transformer-based LLMs and apply singular value decomposition (SVD) to their weight matrices. The most informative singular components are kept on the secure side, while the remainder is offloaded.
2. **Hybrid Inference Protocol**: To prevent the low security resource from recovering sensitive weights, we design a masking protocol that ensures secure interaction during inference.

In Section 3, we detail the decomposition strategy using SVD. In Section 4, we introduce and analyze the protocol. Section 5 evaluates the effectiveness of our full system on several LLMs, validating all four desiderata in practice.

3 Model decomposition via SVD

We now describe our method for securely decomposing the model parameters, satisfying the desiderata of usefulness and efficiency (Section 2.1). Our strategy leverages a key empirical observation: in neural network layers which perform matrix-vector multiplications (or layers which contain such operations, e.g., Linear Layers, Attention), the singular values of these matrices quickly decrease, as illustrated in Figure 2. This suggests that most of the information of the matrix (corresponding to the IP within the layer) is concentrated in a few top singular components, so Charlie can hold these singular component, and efficiently compute the linear portion only using his components. This makes singular value decomposition (SVD) a natural tool for balancing efficiency with usefulness.

3.1 Layer-wise Decomposition

Many layers in transformer-based LLMs and also other modern architectures, contain one or many matrix-vector multiplications, e.g., fully connected layers and attention layers (Appendix E), and convolutional layers (Appendix D). We apply SVD to decompose these matrices into two parts:

1. a small number k of top singular components, retained in the secure side (Charlie), and
2. the remaining components, offloaded to the insecure device (David).

For simplicity of presentation, to assume each layer has a single weight matrix, for attention layers we consider each matrix-vector operation within it as a separate linear layer for the decomposition above. Also for convolutions, we transform the convolution to its equivalent matrix-vector product representation.

Formally, let $\mathbf{W} \in \mathbb{R}^{m \times n}$ be a weight matrix. Its Singular Value Decomposition (SVD) is given by

$$\mathbf{W} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top ,$$

where $\{\sigma_j \in \mathbb{R}_{\geq 0}\}_{j \in [n]}$ are the singular values in decreasing order, $\{\mathbf{u}_j \in \mathbb{R}^m\}_{j \in [m]}$ and $\{\mathbf{v}_j \in \mathbb{R}^n\}_{j \in [n]}$ are the left and right singular vectors, and $r \leq \min\{m, n\}$ is the rank of $\mathbf{W}$. We define the sensitive portion $\mathbf{W}^{\mathcal{C}}$ retained by Charlie to be the top k singular components of $\mathbf{W}$,

$$\mathbf{W}^{\mathcal{C}} = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top ,$$

and the remaining portion $\mathbf{W}^{\mathcal{D}}$ offloaded to $\mathcal{D}$ to be the remaining components

$$\mathbf{W}^{\mathcal{D}} = \sum_{j=k+1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top .$$

This decomposition achieves two goals. It is useful, since the exposed portion $\mathbf{W}^{\mathcal{D}}$ contains little signal, and it is efficient for David. Specifically, for a weight matrix $\mathbf{W}$, computing $\mathbf{W}x$ requires $O(mn)$ floating point operations, whereas if we store $\mathbf{W}^{\mathcal{C}}$ in its decomposed form, computing Charlie's part $\mathbf{W}^{\mathcal{C}}x$ requires only $O(k(n + m))$ operations.

We choose $k \ll r$, typically $k < 10$, to minimize Charlie's compute, but k must be large enough to ensure usefulness, specifically $k > 1$ to prevent a trivial reconstruction attack described in Appendix A.2.

In Section 5, we empirically demonstrate that this method is far superior than other decomposition methods, such as offloading large norm columns or random singular component.

3.2 Full-Model Decomposition

To decompose a full model Θ, we strategically select a set of "sensitive layers" among the layers defined by weight matrices $\mathbf{W}$, and decompose them as in Section 3.1 (see Appendix B.2 for the selection process). The weight matrix $\mathbf{W}^{\mathcal{C}}$ and the remaining computation within the layer (e.g., the non-linearity portion of a fully connected layer) are assigned to $\Theta_{\mathcal{C}}$, and $\mathbf{W}^{\mathcal{D}}$ is assigned to $\Theta_{\mathcal{D}}$. Non-sensitive layers are assigned entirely to $\Theta_{\mathcal{D}}$.

Formally, the overall decomposition becomes:

$$\begin{aligned}
\Theta_{\mathcal{C}} &= \{\mathbf{W}_i^{\mathcal{C}}\}_{i \in sensitive} \\
\Theta_{\mathcal{D}} &= \{\mathbf{W}_i^{\mathcal{D}}\}_{i \in sensitive} \cup \{\mathbf{W}^j\}_{j \notin sensitive}.
\end{aligned}$$

In Section 5, we show that only decomposing a few layers, typically both at the beginning and end of the model using the SVD decomposition from Section 3.1 yields a useful and safe decomposition.

4 Hybrid Inference Protocol

In Section 3, we saw how leverage SVD to decompose a model Θ into $\Theta = (\Theta_C, \Theta_D)$ such that Charlie's part Θ_C preserves most of the model's IP and David's part Θ_D contributes minimally to a malicious David which aims to extractact the original model Θ. In this section, we present a *hybrid inference protocol* where Charlie and David collaborate to compute the inference on this decomposed model by exchanging information on computations on their respective parts. We assume that Charlie owns the model input x and is also responsible for producing the final output of the model.

4.1 Why Naïve Hybrid Inference Leaks IP

Consider a sensitive layer i with weight matrix $\mathbf{W}_i$ decomposed as $\mathbf{W}_i = \mathbf{W}_i^{\mathcal{C}} + \mathbf{W}_i^{\mathcal{D}}$. A naïve inference protocol to compute $\mathbf{W}_i\mathbf{a}_{i-1}$ might proceed as follows. Charlie sends the input activation $\mathbf{a}_{i-1}$ to David, receives $\mathbf{W}_i^{\mathcal{D}}\mathbf{a}_{i-1}$ from David, locally computes $\mathbf{W}_i^{\mathcal{C}}\mathbf{a}_{i-1}$, and applies the nonlinearity σ to obtain $\mathbf{a}_i = \sigma(\mathbf{W}_i^{\mathcal{C}}\mathbf{a}_{i-1} + \mathbf{W}_i^{\mathcal{D}}\mathbf{a}_{i-1})$. If David also observes $\mathbf{a}_i$ (e.g., since the next layer $i+1$ is also sensitive) and if σ is invertible (e.g., a sigmoid or a ReLU over non-negative inputs), he can reconstruct the linear equation $b_i = \mathbf{W}_i^{\mathcal{C}}\mathbf{a}_{i-1} + \mathbf{W}_i^{\mathcal{D}}\mathbf{a}_{i-1}$, in which he knows all variables except $\mathbf{W}_i^{\mathcal{C}}$. By observing sufficiently many pairs $(\mathbf{a}_i, \mathbf{a}_{i-1})$, David can solve a system of linear equations to discover $\mathbf{W}_i^{\mathcal{C}}$, thereby leaking the sensitive information (IP) $\mathbf{W}_i^{\mathcal{C}}$ of the model Θ to David.

This attack shows that, without any additional security measures, the naïve inference protocol is inherently leakes model information about the input/output of the layer, on which an attacker can much more easily perform reconstruction attacks, compared to a protocol in which they learn only the model input/output. We next describe a solution that protects against such leakage.

4.2 IP Protection via Masking

Intuitively, to prevent information leakage about $\Theta_{\mathcal{C}}$ in the inference protocol, e.g., as in the attack above, we *mask* Charlie's messages (internal model activations) so they appear random to David. The tricky part is to recover David's matrix-activation product $\mathbf{W}_i^{\mathcal{D}}\mathbf{a}_{i-1}$ from the output on David's part of the model on the masked input. Fortunately, since David's computation in sensitive layers is *linear* (either in Linear layers or Attention layers which are a composition of linear operations), Charlie can later cancel the masks via *precomputed mask cancellations* to recover David's matrix-activation product locally. Below we describe the main technical ideas of the protocol and its security guarantees. For more formal details of the protocol and proofs, see the companion paper Jain et al. (2025).

To realize this intuition in a concrete protocol, we assume that the model is quantized to integers, and our protocol uses modular arithmetic over *integers mod L* for a sufficiently large L. For each sensitive layer i, before inference, Charlie generates a uniformly random mask $\mathbf{r}_{i-1}$ of suitable length (as the layer input dimension) over integers mod$L$, and precomputes the corresponding cancellation mask $c_i := \mathbf{W}_i^{\mathcal{D}}\mathbf{r}_{i-1}$ and stores it. During inference, for a layer input $\mathbf{a}_{i-1}$, Charlie sends the masked input $\widetilde{\mathbf{a}}_{i-1} = \mathbf{a}_{i-1} + \mathbf{r}_{i-1}$ (mod L) to David, who computes $\widetilde{\mathbf{a}_i^{\mathcal{D}}} := \mathbf{W}_i^{\mathcal{D}} \cdot \widetilde{\mathbf{a}}_{i-1}$ (mod L) and sends it to Charlie. In parallel, Charlie computes $\mathbf{W}_i^{\mathcal{C}}\mathbf{a}_{i-1}$. Upon receiving $\widetilde{\mathbf{a}_i^{\mathcal{D}}}$, Charlie then recovers the output $\mathbf{W}_i\mathbf{a}_{i-1}$, using the precomputed cancellation mask c_i via

$$\mathbf{W}_i\mathbf{a}_{i-1} = \mathbf{W}_i^{\mathcal{C}}\mathbf{a}_{i-1} + \widetilde{\mathbf{a}_i^{\mathcal{D}}} - c_i,$$

which holds since $\mathbf{W}_i = \mathbf{W}_i^{\mathcal{C}} + \mathbf{W}_i^{\mathcal{D}}$ and since $\widetilde{\mathbf{a}_i^{\mathcal{D}}} - c_i = \mathbf{W}_i^{\mathcal{D}}a_{i-1} + \mathbf{W}_i^{\mathcal{D}}\mathbf{r}_{i-1} - c_i = \mathbf{W}_i^{\mathcal{D}}a_{i-1}$ by the definition of c_i. Finally, Charlie derives the next layer input by computing $\mathbf{a}_i = \sigma(\mathbf{W}_i\mathbf{a}_{i-1})$ from the recovered $\mathbf{W}_i\mathbf{a}_{i-1}$.

To apply this intuition for Transformer models, for which the basic layer beyond Matrix-vector multiplication is attention, we give a formal and slightly more involved protocol in Appendix E.

We now analyze the different properties of this protocol.

First, observe that the masking gives Charlie uniformly random values, independent of the activations.

Theorem 1 (Perfect masking). *Let a discrete random variable $s \in \mathbb{Z}^d$ and random noise $n \sim \mathbf{U}[0, L-1]^d$, and denote the masked variable by $s_n = mod(s+n, L)$. Then $s_n \sim \mathbf{U}[0, L-1]^d$, and s_n and s are independent.*

Next, we observe that the computation is correct (assuming David follows the protocol),

Lemma 2. *Assuming $L \geq \|\mathbf{W}_i^{\mathcal{D}}\mathbf{a}_{i-1}\|_\infty$, Charlie's computation correctly computes $\mathbf{W}_i\mathbf{a}_{i-1}$, over the integers modulo L.*

Finally, note that this protocol is efficient since Charlie performs a constant number of vector addition, subtraction and sigmoid computations, but they are dominated by the two heavy operations: **1.** $\mathbf{W}_i^{\mathcal{D}}\mathbf{r}_{i-1}$, which is independent of the input and is therefore pre-computed before the protocol, and **2.** $\mathbf{W}_i^{\mathcal{C}}\mathbf{a}_{i-1}$ which is efficient since $\mathbf{W}_i^{\mathcal{C}}$ contains only the top k SVD components (see Section 3.1).

4.3 Full Model Inference Protocol Π

To extend the protocols above from a single layer to a full model, the full model protocol Π proceeds through the model layer by layer.[4]

For each sensitive layer, whose weights must be protected, we run the masking scheme from Section 4.2.

For each non-sensitive layer which is fully exposed to David, we still need to mask (since otherwise David can leverage internal activations to recover the model easily using reconstruction attacks between activations), which is equivalent to running the masking and scheme for a layer decomposition in which Charlie retains $k = 0$ components.

For both sensitive and non-sensitive layers, all operations which are not Matrix-vector multiplications (e.g., nonlinearities, softmax, etc.) are computed completely by Charlie, so that Charlie can mask the next layer input. Note that the majority of computation is within Matrix-vector multiplications (or in attention layers, for which we give a modified efficient protocol per layer in Appendix E), so our scheme is efficient.

Our main result is that this full model protocol does not leak information about the model, beyond the model input/output for the inference (which in any inference scheme would have leaked, by the definition of inference). Specifically, we apply Theorem 1 for each layer separately to conclude that the masked messages from Charlie to David are independently and uniformly random vectors over integers mod$L$ of the corresponding length (even regardless of David's messages in the protocol). Hence Charlie's view of the protocol is just a set of independently and uniformly random vectors over integers (together with the model output which is shared), as required.[5] By the definition of safety, it follows that if the decomposition $\Theta = (\Theta_{\mathcal{C}}, \Theta_{\mathcal{D}})$ is safe, then the protocol does not leak the model IP to David.

4.4 Putting things together: Hybrid Inference on a Decomposition

Now that we have a general method to decompose a general model (Section 3) and a general protocol to perform inference (Section 4), we select them to ensure the desiderata from Section 2.1. Specifically, in Section 5 we find the layers which are most essential for the inference, and experimentally find the number of sensitive layers and the smallest number of k of retained singular components (which we demonstrate is relatively very small), to ensure the decomposition is both Useful and Safe. We follow with our main result of this paper, that when following the above procedure, the resulting decomposition and hybrid inference protocol are efficient and preserve the model IP.

Theorem 3. *Our solution, including the model decomposition and hybrid inference protocol, preserves model IP (Definition 5) and is efficient.*

Proof. By the decomposition definition above which is useful, first David cannot reconstruct the model from his portion of the model (usefulness). Next, David performs inferences over the model through the hybrid inference protocol, which by Theorem 1, the only thing David learns from any inference about $\Theta_{\mathcal{C}}$ is the model input/output, which don't help him beyond regular model black-box attacks (safety), thus preserving model IP (to the largest possible extent given that David needs the output of each model inference). Finally, efficiency follows thanks the low number of sensitive layers and k, together with our hybrid inference protocol which is communication and computation-efficient for Charlie. For a practical efficiency-IP preservation trade-off, see Section 5. $\square$

5 Experiments

In this section, we evaluate the SLIP framework and explore different decomposition strategies using three LLMs: GPT-2 Small Radford et al. (2019), Phi-2 Microsoft Research (2023), and LLaMA2-7B Touvron et al. (2023). We run experiments on an NVIDIA A100 GPU and use the WikiText benchmark Merity et al. (2016)

[4] We observe that our solution can be extended to trade-off usefulness with efficiency by splitting not the whole model, but rather parts of it which, informally, contain most of the model knowledge.

[5] In our companion paper Jain et al. (2025), we give complete definitions and a formal proof for this.

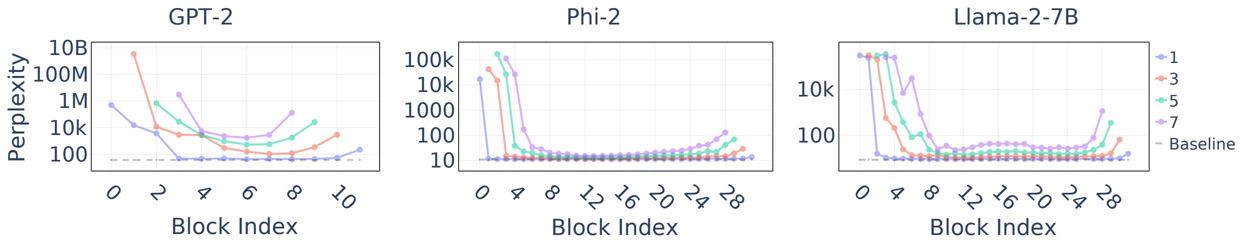

Figure 3: Performance degradation of the LLMs when bypassing windows of 1, 3, 5, or 7 consecutive decoder blocks, compared to the baseline model. The x-axis represents the center block of each window, while the y-axis shows the perplexity (lower perplexity is better).

for perplexity evaluation. Perplexity is computed using EleutherAI's evaluation suite Gao et al. (2023). Our experiments aim to assess the usefulness, safety, and efficiency of the decomposition and protocol.

Our first 4 experiments show how to find a safe and useful decomposition. Specifically, we first find which decoder blocks in the model are best to offload and select as sensitive (contain much IP), then find which layers in these blocks are best to offload, then how to decompose each such layer (SVD over the alternatives), and finally show the trade-off between offload % and IP preservation (measured by perplexity). Our last experiment shows that our decomposition and protocol is efficient.

Which Decoder Blocks are Sensitive. Recall that for efficiency, we want to identify the smallest set possible of sensitive layers which contain the model IP. For LLMs, the high-level layers are decoder blocks, each which contain internal layers (e.g., $\mathbf{W}_q$ for attention query, projection etc.). To identify sensitive decoder blocks, we consider sequences of consecutive decoder blocks with various sequences positions (indexes) and length (1,3,5 or 7), omit them from the model to simulate the portion moving completely to Charlie, and measure the usefulness via perplexity. Consistent with the findings of Gromov et al. (2024), Figure 3 demonstrates that sensitive layers in the first decoder blocks damages David's model the most (i.e., most useful), in last ones slightly less, and the least in the middle. Hence, we experiment with a class of decompositions for models with ℓ, in which the first a layers (1 to a), and the last a layers ($\ell - a + 1$ to ℓ) are taken as sensitive, all with the same value k (for simplicity).

Which Decoder Block Internal Layers are Sensitive. Now that we know which decoder blocks should be taken as sensitive, we must decompose them, i.e., identify their internal sensitive layers. There are several different internal layers which contain Matrix-vector multiplication, specifically MLP layers ($\mathbf{W}_{c\_fc}$, $\mathbf{W}_{c\_proj}$) and attention layers ($\mathbf{W}_q$,$\mathbf{W}_k$,$\mathbf{W}_v$,$\mathbf{W}_o$). We empirically show that the decomposition usefulness greatly varies with the layer type above that we decompose, and the number of top singular components of the weight matrix retained by Charlie (usually $k = 10$ is enough). In Figure 4, we plot the usefulness measured by the resulting perplexity of David's model as a function of both, where we average the results across all the blocks of the model of each layer type to see which is best for each model on average. We find that the taking the internal layer type $\mathbf{W}_{c\_fc}$ as sensitive is useful for GPT-2 and Phi-2, and the layer $\mathbf{W}_k$ is for Llama-2-7B.

We now study Safety, that is, to what extent can an adversary David with his portion of the model and a set of model input/output pairs (taken from a public dataset), recover model performance (through fine-tuning Han et al. (2024) resources). We use the public dataset Alpaca Taori et al. (2023) and LoRA Hu et al. (2021), to fine-tune Phi-2 (or Llama-2-7B) for 10 epochs (to not overfit on the relatively small dataset), across five decomposition configurations and a baseline with randomized model parameters.

How to Decompose a Sensitive Layer Safely. We compare our method **Largest Singular Components (LSC)** to four alternatives:

1. **RSC** - randomly retains 200 singular components,
2. **Norm CS** - retains 200 columns with largest L2 norm,

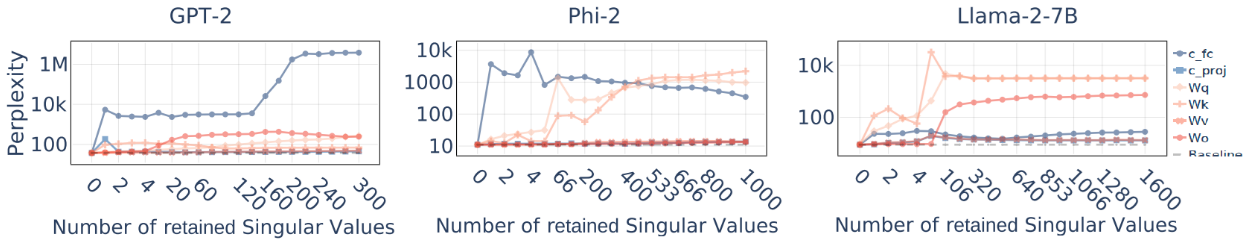

Figure 4: Perplexity scores for all three models, after removing various numbers of singular components from different layer types in a single decoder block. Results are averaged across all possible blocks.

		Accuracy				PPL
Model	**Approach**	**ARC**	**Hellaswag**	**MMLU**	**Average**	**Wikitext**
Phi-2	Base Model	53%	49%	55%	X1.00	11
	LSC (ours)	**21%**	**30%**	**25%**	**X0.48**	**709**
	RSC	52%	47%	53%	X0.97	12
	Norm CS	25%	36%	26%	X0.57	73
	Random CS	51%	48%	53%	X0.97	13
	FMS	42%	42%	26%	X0.71	17
Llama-2 7B	Base Model	44%	50%	42%	X1.00	9
	LSC (ours)	22%	**28%**	**23%**	**X0.53**	**46,090**
	RSC	45%	50%	36%	X0.96	11
	Norm CS	**20%**	29%	25%	X0.54	6,503
	Random CS	43%	50%	38%	X0.96	9
	FMS	24%	32%	28%	X0.61	39

Table 1: Benchmark results for each model and decomposition method on perplexity (PPL Wikitext) and classification accuracy (ARC, Hellaswag and MMLU) after restoration through fine-tuning. Blue indicates safe results while the bold is safest. LSC most often outperforms others.

3. **Random CS** - retains 200 random columns, and

4. **FMS** - Offloads complete weight matrices to reach 97.5% parameter offload to David.

All methods are evaluated on decompositions of the same model, offloading 97.5% of parameters and focusing on the 5 initial and 5 final decoder blocks, and after attempting to restore David's model using the Safety experiment mentioned above. Table 1 shows that the LSC method consistently results in the highest perplexity and lowest accuracy after fine-tuning David's model to the PPL wikitext and testing performance on 3 text classification tasks (ARC Chollet et al. (2025), Hellaswag Zellers et al. (2019) and MMLU Liu (2023)), confirming it as the safest decomposition scheme.

Efficiency vs Safety Trade-off. To experiment with various levels of efficiency, we consider decompositions created by varying the number a of sensitive decoder blocks from the beginning and end of the model, and the number k of top singular values retained by Charlie from each such layer (block). Each decomposition gives David a different fraction of the total model parameters, and we use this ratio as a proxy the inference time ratio. This proxy is accurate since Charlie's computation is dominated by matrix-vector multiplication in sensitive layers, and we store Charlie's portion in its decomposed form (See Section 3.1).

Figure 5 shows the perplexity of a decomposed Phi-2 both after decomposition and after fine-tuning using the Safety study mentioned above. For all decompositions, Charlie's portion of the model has perplexity similar to a random weights model, and after retraining, the more computation offloaded to David, the lower the recovered perplexity. Specifically, configurations that offload 92% to 97.5% to David remain robust against recovery, i.e., safe.

Efficiency Analysis. We simulate the latency of our approach decomposed to an edge device (Charlie) and a cloud server (David), considering the computational capacities of these hardware, network bandwith and delay. Our simplified analysis, based on a version of Llama2 Touvron et al. (2023) with only feed-forward layers, decomposed in a third of its layers, shows edge computation dominates latency (150.34 ms), followed

Idx	Offload%	Top Retained SVs	Sensitive Blocks
Rand	0	0	All
1	85	380	All
2	92	200	All
3	95	200	±10
4	97.5	200	±5
5	99.4	50	±5
6	99.9	10	±5

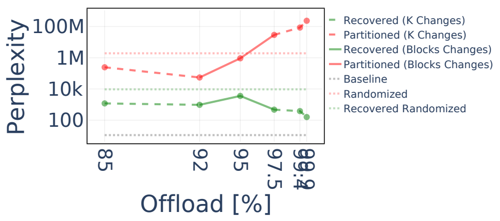

Figure 5: Impact of fine-tuning on various decomposition configurations. Left: list of experimented configurations, with % of offloaded computation to David, number of top singular values retained by Charlie from each layer, and the number of sensitive layers (blocks) in the beginning and end of the model. Right: the perplexity after decomposition (red) and after retraining (green). The experiment starting from randomized weights is dotted, and the baseline perplexity is in gray.

by data transfer (71.31 ms), with cloud computation being minimal (0.66 ms). Our approach is in fact ε-efficient for $\varepsilon = 1.5\%$ (see Appendix C), and one can theoretically prove that for a simplified feed-forward model with ℓ identical linear layers of type $\mathbf{W} \in \mathbb{R}^{d \times d}$, and a sensitive layers in the beginning and the end with k retained eigenvectors, the protocol is $O\left((ak + \ell)/(\ell d)\right)$-efficient.

6 Discussion

In this paper we present SLIP, a novel hybrid inference technique designed to safeguard DNNs, particularly LLMs, against theft when offloaded to low-security resources like edge devices, leaving minimal computations on the secure resource, such as the cloud or a trusted execution environment. To our knowledge, SLIP is the first technique that is both practical for real-world use and provably IP-preserving, without compromising model accuracy (the hybrid inference protocol retains the original model outputs) or introducing significant latency (primarily influenced by network delays which can be mitigated in scenarios such as TEEs). It leverages the rapid decay of singular components in LLM weight matrices to obtain a useful, efficient and safe model decomposition, storing only the most sensitive singular components securely, while the remaining model is useless and robust against fine-tuning restoration, more than alternative partitioning approaches. To protect the secret portion of the model from the low-security resource, we give a provably secure and efficient inference protocol. These properties make SLIP highly promising for industry applications, enabling safe cost-effective model deployment on edge devices, and providing a foundation to further develop the hybrid inference framework. Although our approach is limited to DNNs that are quantized to integers, this is not a significant limitation since the majority of models are quantized pre-deployment to improve latency and memory usage.

For future work, one can improve our framework applicability, and quantify its practical latency, obfuscate the model architecture, and improve the efficiency of the random mask generation and removal. Moreover, evaluating our method to other models and tasks is important. Finally, we believe that developing optimization algorithms to best balance the IP-preservation, latency and computation offload of the decomposition, is very impactful.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318, 2016.

Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.

Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1615–1631, 2018.

Jihwan Bang, Juntae Lee, Kyuhong Shim, Seunghan Yang, and Simyung Chang. Crayon: customized on-device llm via instant adapter blending and edge-server hybrid inference. *arXiv preprint arXiv:2406.07007*, 2024.

Lewis Birch, William Hackett, Stefan Trawicki, Neeraj Suri, and Peter Garraghan. Model leeching: An extraction attack targeting llms, 2023.

Isaac A Canales-Martínez, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, Nitin Satpute, and Adi Shamir. Polynomial time cryptanalytic extraction of neural network models. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 3–33. Springer, 2024.

Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr, Arthur Conmy, et al. Stealing part of a production language model. *arXiv preprint arXiv:2403.06634*, 2024.

Guo Chaopeng, Lin Zhengqing, and Song Jie. A privacy protection approach in edge-computing based on maximized dnn partition strategy with energy saving. *Journal of Cloud Computing*, 12(1):29, 2023.

Cheng Chen, Nicholas Genise, Daniele Micciancio, Yuriy Polyakov, and Kurt Rohloff. Implementing token-based obfuscation under (ring) LWE. Cryptology ePrint Archive, Paper 2018/1222, 2018. URL `https://eprint.iacr.org/2018/1222`. `https://eprint.iacr.org/2018/1222`.

Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. Deepattest: An end-to-end attestation framework for deep neural networks. In *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 487–498, 2019.

Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107 (8):1655–1674, 2019. doi: 10.1109/JPROC.2019.2921977.

Tianyi Chen, Tianyu Ding, Badal Yadav, Ilya Zharkov, and Luming Liang. Lorashear: Efficient large language model structured pruning and knowledge recovery, 2023.

Ke Cheng, Ning Xi, Ximeng Liu, Xinghui Zhu, Haichang Gao, Zhiwei Zhang, and Yulong Shen. Private inference for deep neural networks: A secure, adaptive, and efficient realization. *IEEE Transactions on Computers*, 72(12):3519–3531, 2023. doi: 10.1109/TC.2023.3305754.

Francois Chollet, Mike Knoop, Greg Kamradt, Walter Reade, and Addison Howard. Arc prize 2025. `https://kaggle.com/competitions/arc-prize-2025`, 2025. Kaggle.

Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355–375, August 2020. ISSN 2299-0984. doi: 10.2478/popets-2020-0077.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023.

Mahdieh Grailoo, Zain Ul Abideen, Mairo Leier, and Samuel Pagliarini. Preventing distillation-based attacks on neural network ip, 2022.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers, 2024.

Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. Cryptology ePrint Archive, Paper 2023/1269, 2023. `https://eprint.iacr.org/2023/1269`.

Zeyu Han, Chao Gao, Jinyang Liu, Sai Qian Zhang, et al. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party gpt inference. Cryptology ePrint Archive, Paper 2023/1147, 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

Lin Huang, Gejian Zhao, and Chuan Qin. Recoverable active protection framework for neural network models. In *2023 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, 2023. doi: 10.1109/WIFS58808.2023.10374926.

Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure {Two-Party} deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 809–826, 2022.

Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX security symposium (USENIX Security 20)*, pp. 1345–1362, 2020.

Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. *Commun. ACM*, 67(3):97–105, Feb 2024. ISSN 0001-0782. doi: 10.1145/3611095. URL `https://doi.org/10.1145/3611095`.

Racchit Jain, Satya Lokam, Yehonathan Refael, Adam Hakim, Lev Greenberg, and Jay Tenenbaum. Slip-sec: Formalizing secure protocols for model ip protection, 2025.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX security 18)*, pp. 1651–1669, 2018.

Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pp. 615–629, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450344654. doi: 10.1145/3037697.3037698.

Sanjay Kariyappa, Atul Prakash, and Moinuddin K Qureshi. Maze: Data-free model stealing attack using zeroth-order gradient estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13814–13823, June 2021.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *iEEE Access*, 10:30039–30054, 2022.

Qinfeng Li, Zhiqiang Shen, Zhenghan Qin, Yangfan Xie, Xuhong Zhang, Tianyu Du, and Jianwei Yin. Translinkguard: Safeguarding transformer models against model stealing in edge deployment, 2024.

Yue Li, Hongxia Wang, and Mauro Barni. A survey of deep neural network watermarking techniques. *Neurocomputing*, 461:171–193, 2021.

Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pp. 619–631, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3134056.

Peiyuan Liu. Mmlu dataset, 2023. URL `https://www.kaggle.com/ds/3638509`.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21702–21720. Curran Associates, Inc., 2023.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Microsoft Research. Phi-2: The surprising power of small language models. Language model with outstanding reasoning and language understanding capabilities, 2023.

Aaron Mok. ChatGPT could cost over $700,000 per day to operate, 2023. [Accessed 22-05-2024].

Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Comput. Surv.*, 55(14s), jul 2023. ISSN 0360-0300. doi: 10.1145/3595292.

George Onoufriou, Paul Mayfield, and Georgios Leontidis. Fully homomorphically encrypted deep learning as a service. *Machine Learning and Knowledge Extraction*, 3(4):819–834, 2021.

Ray Perrault and Jack Clark. Artificial intelligence index report 2024. *Policy Commons*, 2024.

Fabien AP Petitcolas. Kerckhoffs' principle. In *Encyclopedia of Cryptography, Security and Privacy*, pp. 1–2. Springer, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 325–342, 2020.

Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357005. doi: 10.1145/3195970.3196023.

Alexander Schlögl and Rainer Böhme. ennclave: Offline inference with model confidentiality. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, pp. 93–104, 2020.

Adi Shamir, Isaac Canales-Martinez, Anna Hambitzer, Jorge Chavez-Saab, Francisco Rodrigez-Henriquez, and Nitin Satpute. Polynomial time cryptanalytic extraction of neural network models, 2023.

Pratyusha Sharma, Jordan T. Ash, and Dipendra Misra. The truth is in there: Improving reasoning in language models with layer-selective rank reduction, 2023.

Kálmán Szentannai, Jalal Al-Afandi, and András Horváth. Preventing neural network weight stealing via network obfuscation. In *Intelligent Computing: Proceedings of the 2020 Computing Conference, Volume 3*, pp. 1–11. Springer, 2020.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Jean-Baptiste Truong, Pratyush Maini, Robert J Walls, and Nicolas Papernot. Data-free model extraction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4771–4780, 2021.

Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on international conference on multimedia retrieval*, pp. 269–277, 2017.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression. *arXiv preprint arXiv:2403.07378*, 2024.

Hui Xu, Yuxin Su, Zirui Zhao, Yangfan Zhou, Michael R Lyu, and Irwin King. Deepobfuscation: Securing the structure of convolutional neural networks via knowledge distillation. *arXiv preprint arXiv:1806.10313*, 2018.

Yifan Yan, Xudong Pan, Mi Zhang, and Min Yang. Rethinking {White-Box} watermarks on deep learning models under neural structural obfuscation. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 2347–2364, 2023.

Peng Yang, Yingjie Lao, and Ping Li. Robust watermarking for deep neural networks via bi-level optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14841–14850, 2021.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2): 100211, 2024. ISSN 2667-2952. doi: https://doi.org/10.1016/j.hcc.2024.100211.

Dingfeng Ye, Peng Liu, and Jun Xu. How fast can we obfuscate using ideal graded encoding schemes. Cryptology ePrint Archive, Paper 2017/321, 2017.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pp. 159–172, 2018.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2023.

Mingyi Zhou, Xiang Gao, Jing Wu, John C Grundy, Xiao Chen, Chunyang Chen, and Li Li. Model obfuscation for securing deployed neural networks. *openreview*, 2022a.

Tong Zhou, Shaolei Ren, and Xiaolin Xu. Obfunas: A neural architecture search-based dnn obfuscation approach. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022b.

Tong Zhou, Yukui Luo, Shaolei Ren, and Xiaolin Xu. NNSplitter: An active defense solution for DNN model via automated weight obfuscation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 42614–42624. PMLR, 23–29 Jul 2023.

A Proofs

A.1 Unmasking: Proof of Lemma 2

Assuming $L \geq \|\mathbf{W}_i^{\mathcal{D}}\mathbf{a}_{i-1}\|_\infty$, Charlie's computation correctly computes $\mathbf{a}_i = \mathbf{W}_i\mathbf{a}_{i-1}$, over the integers modulo L.

Proof.

$$\begin{aligned}
\mathbf{a}_{i+1}^{\mathcal{D}} &= \text{mod}\left(\widetilde{\mathbf{a}_{i+1}^{\mathcal{D}}} - \mathbf{c}_i, L\right) \\
&= \text{mod}\left(\mathbf{W}_{i+1}^{\mathcal{D}}\tilde{\mathbf{a}}_i - \mathbf{W}_{i+1}^{\mathcal{D}}\mathbf{r}_i, L\right) \\
&= \text{mod}\left(\mathbf{W}_{i+1}^{\mathcal{D}}\text{mod}(\mathbf{a}_i + \mathbf{r}_i, L) - \mathbf{W}_{i+1}^{\mathcal{D}}\mathbf{r}_i, L\right) \\
&\underbrace{=}_{(1)} \text{mod}\left(\mathbf{W}_{i+1}^{\mathcal{D}}(\mathbf{a}_i + \mathbf{r}_i) - \mathbf{W}_{i+1}^{\mathcal{D}}\mathbf{r}_i, L\right) \\
&= \text{mod}\left(\mathbf{W}_{i+1}^{\mathcal{D}}\mathbf{a}_i, L\right) \\
&\underbrace{=}_{(2)} \mathbf{W}_{i+1}^{\mathcal{D}}\mathbf{a}_i
\end{aligned}$$

where, (1) follows from properties of modulo operator and (2) follows form the assumption that $\|\mathbf{W}_{i+1}^{\mathcal{D}}\mathbf{a}_i\|_\infty < L$. □

Recall **Theorem 1.**[Perfect masking] Let a discrete random variable $s \in \mathbb{Z}^d$ and random noise $n \sim \mathbf{U}[0, L-1]^d$, and denote the masked variable by $s_n = \text{mod}(s + n, L)$. Then $s_n \sim \mathbf{U}[0, L-1]^d$, and s_n and s are independent.

Proof. We first prove it for one-dimensional case $d = 1$. It is enough to show

$$P(x = a) = P(\text{mod}(s + n, L) = a) = 1/L.$$

We separate analysis to two distinct cases: **1.** $\text{mod}(s, L) \leq a$, and **2.** $\text{mod}(s, L) > a$:

1. For $\text{mod}(s, L) \leq a$, the equation $\text{mod}(s + n, L) = a$ holds iff $n = a - s$. In this case,
$$P(\text{mod}(s + n, L) = a) = P(n = a - s) = 1/L.$$
2. Similarly, for $\text{mod}(s, L) > a$, equation $\text{mod}(s + n, L) = a$ holds iff $n = L - (\text{mod}(s, L) - a)$. So also this case,
$$P(\text{mod}(s + n, L) = a) = P(n = L - (\text{mod}(s, L) - a)) = 1/L.$$

Finally, for any dimension $d > 1$, by repeating the same analysis for each dimension, we get

$$P(x = a) = \prod_{i=1}^{d} P(x_i == a_i) = 1/L^d,$$

where x_i and a_i are i-th coordinate of d-dimensional vectors x and a. □

A.2 Minimal number of Singular Vectors that must be decomposed

The following lemma shows that under the use of SVD decomposition, more than one singular vector is needed to be removed from the model and stored separately on the cloud, noted by W_c.

Lemma 4 (The minimal number of required singular vectors to be hidden)**.** *The matrix V_c should contain more than one singular vector, in order to ensure that an attacker cannot reconstruct W_c using only the singular vector available to them on the edge in W_e, or that they are combinatoricaly complex for the user to reconstruct.*

Proof. Let us denote

$$W_e = S_e V_e D_e =$$
$$\begin{pmatrix} -- & S_1 & -- \\ -- & S_2 & -- \\ -- & S_3 & -- \\ & \vdots & \\ -- & S_n & -- \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & \sigma_n \end{pmatrix} \cdot \begin{pmatrix} -- & D_1 & -- \\ -- & D_2 & -- \\ -- & D_3 & -- \\ & \vdots & \\ -- & D_n & -- \end{pmatrix}^t$$

Let us suppose that one singular vector (without the loss of generally the largest one), namely S_1, should be enough to be designated for the matrix W_c, ensuring W_c is unrecoverable to an attacker.. Then, in the edge device, we have $n-1$ orthonormal vectors in $\{S_2, S_3, \ldots, S_n\}$. Since all vectors in it are orthonormal it could be leveraged to construct the following $n-1$ equations,

$$\begin{gathered} S_2^t \cdot S_1 = 0 \\ S_3^t \cdot S_1 = 0 \\ \vdots \\ S_n^t \cdot S_1 = 0, \end{gathered}$$

which results in only one degree of freedom. Now, simply applying the Gram-Schmidt process (or any other orthonormalization techniques) would result in the exact S_1 singular vector. This is in contradiction to the assumption. ☐

Remark 1 (Finding the corresponding missing singular vector)**.** *The sum of the singular values of a matrix equals the trace of that matrix, meaning* $\text{trace}(W_e) = \lambda_1 + \lambda_2 + \ldots + \lambda_{n-1} + \lambda_n$*. Given $\lambda_2, \lambda_3, \ldots, \lambda_{n-1}$, the first singular value could be computed by*

$$\lambda_1 = \text{trace}(W_e) - (\lambda_2 + \lambda_3 + \ldots + \lambda_n)$$

Corollary 4.1. *Directly by Lemma 4, separating $n >> k \geq 2$ singular values and their corresponding singular vectors into their cloud designated matrices S_c, V_c, D_c, results in k equations with n unknown parameters. Thus, in this case, the level of freedom that can not be retrieved (by linear algebraic operations) can given by $n-k+1$.*

B Practical Considerations

B.1 Pseudo-random numbers generation

In the main paper, we analyzed our protocol assuming true randomness from uniform distributions. However, in practice, we would use a Cryptographically Secure Pseudorandom Number Generator (CSPRNG). A CSPRNG has the property that no polynomial time adversary can distinguish between true uniform distribution and the output of such a generator except with a negligible probability. By using such a generator, our security proofs will continue to hold. Otherwise, an adversary that breaks security of our schemes can be used to construct a distinguisher that breaks the security of a CSPRNG.

B.2 Model Decomposition Strategy

Technically, decomposition of an entire model begins with a list of triplets that contains indices of sensitive layers and defines how to perform the decomposition. In the case of LLMs, it would appear as follows:

$$[[\mathbf{block}, \mathbf{layer\_type}, \mathbf{K}]_1, \ldots, [\mathbf{block}, \mathbf{layer\_type}, \mathbf{K}]_n]$$

where **block** is any decoder block within the LLM, **layer_type** is any layer out of the possible layers in a decoder block, and **K** is the number of top singular components to hide in Charlie. Therefore, each triplet defines in which block and layer, and how many, singular components should be hidden. Using this list, an LLM can be decomposed into the secret model Θ_C, comprised of the components detailed in the list, and an exposed version, comprised of all remaining components.

Finding the optimal list with which to perform the partition is an optimization problem, that should consider trade-offs between the three following metrics: model security level, computational offload, and latency. For example, increasing security by hiding a larger amount of singular components comes at the cost of less computational offload. Additionally, choosing more layers on which to perform decomposition results in additional networking latency. In this paper, we evaluated several decompositions, taking into account experimental findings in Section 5. For example, we found it is useful to focus the first few and last few blocks of the decoder backbone, as removing them inflicts the most degradation in perplexity. Additionally, we found that sensitive layer types differ between LLMs, so we propose decomposing all layers in a selected block. Finally, we found that it is enough to focus on a small number of singular components (50 out of 2560). Therefore, we recommend that any attempts to apply our approach should begin with the two experiments outlined in Section 5. These experiments are essential for identifying sensitive decoder blocks, key layer types, and the appropriate number of singular vectors for each model under consideration. Nonetheless, more optimized model decomposition could potentially be achieved by incorporating techniques such as Reinforcement Learning, Genetic Algorithms, or additional heuristics. We leave the exploration of these advanced methods to future work.

C Compute & latency analysis details

In this analysis, we evaluate the inference latency of a neural network model that is partitioned between an edge device and the cloud according to the protocol suggested in the paper. This setup necessitates considering both computational latency and data transfer latency between the two compute resources. To accurately measure the total latency, we break down the protocol into distinct phases and calculate the computations in each, which contribute to the overall latency. Key parameters influencing latency include the bandwidth between the edge and cloud, the number of layers, the processor FLOPs capacity, and the size of the data being transferred. The formulas used for latency approximation are given by:

Computation Time ($T_{\mathbf{Compute}}$): The time required to perform the necessary computations on the edge or cloud.

$$T_{\text{Compute}} = \frac{\text{FLOPs}}{\text{HardwareFLOP/s} \times \text{Utilization}} \tag{1}$$

Data Transfer Time ($T_{\mathbf{Transfer}}$): The time required to transfer data between the edge and the cloud.

$$T_{\text{Transfer}} = \lambda_{\text{a}} + \frac{\text{input\_size} \times b \times s}{\text{Bandwidth}} \tag{2}$$

Where λ_{a} is the network latency between the edge and the cloud, b is the batch size, and s is the size of each activation value in bytes.

Table 2: Description of phases and operations

Phase	Op Type	Frequency	Description	FLOPs/Data Size
Upload Input	Transfer	Once	Transfer input data X of size n to the cloud	$n \times b$
Edge-Only Compute	Compute	Each non-decomposed layer	Compute all non-decomposed operations for matrices W_{e_i}	$2 \cdot m \cdot n \cdot b + n \cdot b$
Edge-Partial Compute	Compute	Each decomposed layer	Weights Matrix Multiplication $W_{e_{i+1}} a_{i_{\text{nois}}}$	$2 \cdot m \cdot n \cdot b$
Cloud-Partial Compute	Compute	Each decomposed layer	Decomposed Matrices Multiplication $U_c \Sigma_c V_c a_i$	$2 \cdot k \cdot b \cdot (m + n)$
Upload Edge to Cloud	Transfer	Each decomposed layer	Transfer edge activation $a_{e,i+1_{\text{noisy}}}$ data to the cloud	$n \times b$
Cloud Activation Function	Compute	Each decomposed layer	Remove Noise + Vector addition + activation $a_{i+1} = \sigma\left(a_{c_i} + a_{e_i} - \tilde{\Delta}_i\right)$	$2 \cdot n \cdot b \cdot (L_v + 1)$
Cloud Noise Vector Generation	Compute	Each decomposed layer	Sample L_v noise vectors and create Δ_i	$2 \cdot n \cdot b \cdot L_v$
Cloud Noise Addition	Compute	Each decomposed layer	Vector addition $a_{i_{\text{noisy}}} = a_i + \Delta_i$	$n \cdot b$
Activation Data Download	Transfer	Each decomposed layer	Transfer final activation data to edge $a_{i_{\text{noisy}}}$	$n \times b$

Note that we chose to keep the SVD-decomposed weight matrix W_{c_i} decomposed in the cloud as $U_{c_i} \Sigma_{c_i} V_{c_i}$. This decision is based on the assumption that $k \ll m, n$, which makes the amount of computation lower than performing a full matrix multiplication. Conversely, we decided to reconstruct W_{e_i} after decomposition for use on the edge device because in this scenario, using the reconstructed matrix on the edge results in less computation compared to operating with the decomposed form.

C.1 Total latency calculation

Since the processing of each phase is sequential and cannot be parallelized, the total latency is the sum of the individual latency of each phase. This ensures that the computation and data transfer steps are accounted for in a linear sequence, reflecting the actual flow of operations during inference.

The total latency (T_{Total}) for the purposed protocol, where the model is partitioned between an edge device and a cloud server, can be divided into three main components: edge computations (P^e), cloud computation (P^c), and data transfer (P^t).

Edge Computation are given by

$$\text{FLOPS}^{\text{edge}} = (l - l_{\text{d}}) \cdot P^{\text{e}}_{\text{non-dec}} + l_{\text{d}} \cdot P^{\text{e}}_{\text{compute}} = 2mnb \cdot l + nb \cdot (l - l_{\text{d}}) \tag{3}$$

Where l_{d} represents the decomposed layers and l represents the total layers in the model. With regards to the full model on the edge, we find that

$$\text{FLOPS}^{\text{full}} = 2mnb \cdot l + nb \cdot l \tag{4}$$

So the cloud offload consists of $n \cdot l_{\text{d}}$ activation operations.

Cloud Computation are given by

$$\begin{aligned} \text{FLOPS}^{\text{cloud}} &= l_{\text{d}} \cdot \left(P^{\text{c}}_{\text{compute}} + P^{\text{c}}_{\text{composed}} + P^{\text{c}}_{\text{noise\_gen}} + P^{\text{c}}_{\text{noise\_add}}\right) \\ &= 2l_{\text{d}} \cdot b \cdot (mk + nk + 2nl_v + 1.5n) \end{aligned} \tag{5}$$

Data Transfer is given by

$$N^{\text{transfer}} = P^t_{\text{input}} + l_{\text{d}} \cdot \left(P^t_{\text{upload}} + P^t_{\text{download}}\right) = nb\,(2l_{\text{d}} + 1) \tag{6}$$

Total Latency is given by

$$T_{\text{Total}} = T_{\text{Compute}}\left(\text{FLOPS}^{\text{edge}}\right) + T_{\text{Compute}}\left(\text{FLOPS}^{\text{cloud}}\right) + T_{\text{Transfer}}\left(N^{\text{transfer}}\right) \tag{7}$$

C.2 Selecting parameter values

Next, we will select specific values for the parameters used in the formulas and analyze the results. This will help illustrate the computational and data transfer latency in our protocol, when the weights are split between an edge device and a cloud server. We based many of our numbers on Llama2 model, treating it as if it were a regular feedforward (FF) network to simplify the analysis and make the estimations more straightforward. Additionally, we assumed a single token generation and chose the one of the decomposition strategies mentioned in the Experiments section as an efficient and robust option.

Table 3: Summary of model parameters, hardware specifications, and network details

Parameter	Value	Description
Model Parameters		
l : Total Layers	224	7 layers (4 attn + 3 mpl) per block Llama2 contains 32 blocks
l_{d} : Decomposed Layers	70	7 layers (4 attn + 3 mpl) per block 10 decomposed blocks in configuration X4
n, m : Matrix Dimensions	4096, 4096	Attention Layer
b : batch_size	32	Average input length
k : Secret Singular Values	50	X4 configuration
l_v : Sampled Noise Vectors	50	
s : Activation size in bytes	$4b$	FP32
Hardware Specifications		
Edge GPU FLOP/sec	4 TFlops/sec	Nvidia Jetson Nano spec (estimation for FP32)
Cloud GPU FLOP/sec	14 TFlops/sec	Nvidia V100 spec
Utilization	40%	Realistic estimation
Network Details		
B : Network Bandwidth	25 MB/s	Average bandwidth in the US
λ_{a} : Network Delay	35 ms	Average latency in the US Global IP Network Latency (att.net)

C.3 Results

The results of our calculations are as follows, and are discussed in the paper itself in section 5.

Table 4: Performance

Metric	Value
FLOPs Full Model	240.547 GFLOPs
FLOPs Edge (Hybrid)	240.538 GFLOPs
FLOPs Cloud (Hybrid)	3.697 GFLOPs
Input Transfer	1.848 M
Compute Edge Latency	150.34 ms
Compute Cloud Latency	0.66 ms
Transfer Latency	71.31 ms
Total Latency	**222.31 ms**

C.4 Implications

It is clear that applying our approach incurs additional latency. In use-cases where the added latency does not significantly impact the user experience, such as offline use-cases (i.e summarization, indexing, processing), our approach may be applied with even higher IP-preservation, by decomposing additional layers. In use-cases where the computational offload is not as important, such as when the trusted entity is not costly to employ, an even larger number of selected singular components may be used to increase IP-preservation. However, in scenarios that require real-time responses, our approach should be used sparingly, and only minimal sensitive layers should be selected for decomposition to arrive at feasible latency periods.

D Representing a Convolutional Layer as a Fully Connected Layer

For the purpose of decomposing a convolutional layer, we can represent the convolutional layer as a fully connected layer, and then perform SVD to decompose the layer as detailed in our paper. **Convolutional Layer**. Consider the input tensor $\mathbf{X}$ of shape (H, W, C), the convolutional kernel $\mathbf{K}$ of shape (kH, kW, C, N), and the output tensor $\mathbf{O}$ of shape (H_o, W_o, N). For simplicity, assume stride $s = 1$ and no padding.

The output at position (i, j) for filter n is given by:

$$\mathbf{O}_{i,j,n} = \sum_{c=1}^{C} \sum_{u=1}^{kH} \sum_{v=1}^{kW} \mathbf{X}_{i+u-1,j+v-1,c} \cdot \mathbf{K}_{u,v,c,n}$$

Fully Connected Layer. The equivalent fully connected layer will have an input vector $\mathbf{x}$ and a weight matrix $\mathbf{W}$. The input tensor $\mathbf{X}$ is unrolled into a vector $\mathbf{x}$ of size $H \cdot W \cdot C$. The unrolling process is detailed as follows.

1. **Flatten the Input:**

$$\mathbf{x} = \text{vec}(\mathbf{X})$$

2. **Construct the Weight Matrix:** The weight matrix $\mathbf{W}$ for the fully connected layer will be of shape $(H_o \cdot W_o \cdot N, H \cdot W \cdot C)$. Each row of $\mathbf{W}$ corresponds to a specific position of the kernel applied to the flattened input.

For each output position (i, j) and filter n:

$$\mathbf{W}_{n,\text{index}(i,j,c)} = \mathbf{K}_{u,v,c,n}$$

where $\text{index}(i, j, c)$ maps the 3D position (i, j, c) in the input tensor to the corresponding position in the flattened vector.

Analytical Formulation

For each output position (i, j) and filter n:

$$\mathbf{O}_{i,j,n} = \sum_{c=1}^{C} \sum_{u=1}^{kH} \sum_{v=1}^{kW} \mathbf{X}_{i+u-1,j+v-1,c} \cdot \mathbf{K}_{u,v,c,n}$$

Unroll $\mathbf{X}$ to $\mathbf{x}$ and define the corresponding weight vector $\mathbf{w}_{(i,j,n)}$ in $\mathbf{W}$:

$$\mathbf{w}_{(i,j,n)} = \text{vec}(\mathbf{K}_{:,:,c,n})$$

The output $\mathbf{O}$ is obtained by matrix multiplication:

$$\mathbf{o} = \mathbf{W} \cdot \mathbf{x}$$

where $\mathbf{o}$ is the flattened version of $\mathbf{O}$.

E Hybrid Inference Protocol for Attention Layers

In the vanilla Transformer model, each attention head receives a sequence of tokens embedding denoted by $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, where n represents the sequence length, d is the embedding dimension, and d_h is the number of tokens in a batch. The weight parameters $\boldsymbol{W}_q, \boldsymbol{W}_v \in \mathbb{R}^{d\times d_h}$, and $\boldsymbol{W}^V \in \mathbb{R}^{d\times d_h}$ are used to transform the input features $\boldsymbol{X}$ into query $\boldsymbol{Q}$, key $\boldsymbol{K}$, and value $\boldsymbol{V}$, respectively.

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_k, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_v,$$
$$\mathrm{Attn}(\boldsymbol{X}) = \mathrm{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_h}}\right)\boldsymbol{V}.$$

In the following, we provide the inference protocol for the attention head layer.

Attention Head - Inference Protocol (On-line, per inference instance)

$\mathcal{C}$harlie

$\mathcal{D}$avid

Inner Masking

- Input to this layer: a_{i-1}
- Retrieve pre-computed random masks r^q_{i-1}, r^k_{i-1}
- $\widetilde{a^q}_{i-1} := a_{i-1} + r^q_{i-1}$
- $\widetilde{a^k}_{i-1} := a_{i-1} + r^k_{i-1}$
- ${a_q}^{\mathcal{C}}_i := {W_q}^{\mathcal{C}}_i a^q_{i-1}$
- ${a_k}^{\mathcal{C}}_i := {W_k}^{\mathcal{C}}_i a^k_{i-1}$

$\xrightarrow{\widetilde{a}_{i-1}}$

$\mathcal{D}$'s Local Compute

- $\widetilde{{a_q}^{\mathcal{D}}_i} := {W_q}^{\mathcal{D}}_i \cdot \widetilde{a^q}_{i-1}$
- $\widetilde{{a_k}^{\mathcal{D}}_i} := {W_k}^{\mathcal{D}}_i \cdot \widetilde{a^k}_{i-1}$

$\xleftarrow{\widetilde{{a_q}^{\mathcal{D}}_i}, \widetilde{{a_k}^{\mathcal{D}}_i}}$

Inner Unmasking

- Retrieve pre-computed cancellation mask c_{q_i}, c_{k_i}
- ${a_q}^{\mathcal{D}}_i = \widetilde{{a_q}^{\mathcal{D}}_i} - c_{q_i} = W^{\mathcal{D}}_{q_i} \cdot \left(\widetilde{a}_{q\,i-1} - r^q_{i-1}\right) = W^{\mathcal{D}}_{q_i} \cdot a_{i-1}$
- ${a_k}^{\mathcal{D}}_i = \widetilde{{a_k}^{\mathcal{D}}_i} - c_{k_i} = W^{\mathcal{D}}_{k_i} \cdot \left(\widetilde{a}_{k\,i-1} - r^k_{i-1}\right) = W^{\mathcal{D}}_{k_i} \cdot a_{i-1}$

Compute Midterm Softmax Output

- $a_{s_i} = \text{softmax}\left(\frac{\left({a_q}^{\mathcal{C}}_i + {a_q}^{\mathcal{D}}_i\right) \cdot \left({a_k}^{\mathcal{C}}_i + ({a_k}^{\mathcal{D}}_i\right)^T}{\sqrt{|a_{i-1}|}}\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{|a_{i-1}|}}\right)$

Outer Masking

- Retrieve pre-computed random mask r^v_{i-1}
- $\widetilde{a}_{s_i} := a_{s_i} + r^v_{i-1}$
- $a^{\mathcal{C}}_i := W^{\mathcal{C}}_{v_i} a_{s_i}$

$\xrightarrow{\widetilde{a}_{s_i}}$

$\mathcal{D}$'s Local Compute

- $\widetilde{a^{\mathcal{D}}_i} := W^{\mathcal{D}}_{v_i} \cdot \widetilde{a}_{s_i}$

$\xleftarrow{\widetilde{a^{\mathcal{D}}_i}}$

Outer Unmasking

- Retrieve pre-computed cancellation mask c_{v_i}
- $a^{\mathcal{D}}_i = \widetilde{a^{\mathcal{D}}_i} - c_{v_i} = W^{\mathcal{D}}_{v_i} \cdot \left(\widetilde{a}_{s_i} - r^v_{i-1}\right) = W^{\mathcal{D}}_{v_i} \cdot a_{s_i}$

Compute Attention Head Output

- $a_i = \sigma(a^{\mathcal{C}}_i + a^{\mathcal{D}}_i) = \sigma((W^{\mathcal{C}}_{v_i} + W^{\mathcal{D}}_{v_i})a_{i-1}) = \sigma(W_{v_i} a_{i-1})$
- $a_i = \sigma(a^{\mathcal{C}}_i + a^{\mathcal{D}}_i) = \sigma((W^{\mathcal{C}}_i + W^{\mathcal{D}}_i)a_{i-1}) = \sigma(W_i a_{i-1})$

Figure 6: Double Step Inference Protocol For Attention Head – Online, per Inference Instance