CHEX-MIX: Combining Homomorphic Encryption with Trusted Execution Environments for Two-party Oblivious Inference in the Cloud

Deepika Natarajan
University of Michigan
Ann Arbor, MI

Andrew Loveless
University of Michigan
Ann Arbor, MI

Wei Dai
Microsoft Research
Redmond, WA

Ronald Dreslinski
University of Michigan
Ann Arbor, MI

Abstract

Data, when coupled with state-of-the-art machine learning models, can enable remarkable applications. But, there exists an underlying tension: users wish to keep their data private, and model providers wish to protect their intellectual property. Homomorphic encryption (HE) and multi-party computation (MPC) techniques have been proposed as solutions to this problem; however, both techniques require model providers to fully trust the server performing the machine learning computation. This limits the scale of inference applications since it prevents model providers from leveraging shared public cloud infrastructures.

In this work, we present CHEX-MIX, a solution to the problem of privacy-preserving machine learning between two mutually distrustful parties in an untrusted cloud setting. CHEX-MIX relies on a combination of HE and trusted execution environments (TEEs), using HE to provide clients with confidentiality guarantees and TEEs to provide model providers with confidentiality guarantees and protect the integrity of computation from malicious cloud adversaries. Unlike prior solutions to this problem, such as multi-key HE, single-key HE, MPC, or TEE-only techniques, our solution assumes that both clients and the cloud can be malicious, makes no collusion assumptions, and frees model providers from needing to maintain private online infrastructures. Our implementation results show that CHEX-MIX can execute at high efficiency, with low communication cost, while providing security guarantees unaddressed by prior work.

1 Introduction

The rise of machine learning (ML) has enabled a host of improvements in nearly all aspects of life, from medical diagnosis [72, 83], to finance [31], personal assistants [67], and more [26, 88]. Alongside the rise of cloud computing, these technologies reach users across the globe at increasingly large scales. However, users often incur a high privacy cost when using inference services since users must provide their personal data to model providers to obtain inference results. This threat is further exacerbated by attacks on datacenters leading to direct access to user data.

A simple solution to this problem is having ML providers deploy their models close to users, such as directly on client devices. However, training robust and useful models is difficult and monetarily expensive, and models are thus often considered key intellectual property which model owners are reluctant to share [68]. Moreover, sharing the details of a model can increase the ability of attackers to perform re-identification or membership inference attacks [6, 98] that violate the privacy of user data in the model training set.

This problem of providing privacy to both clients and model providers, known generally as *oblivious inference* [19, 81], is often addressed with cryptographic techniques such as multi-party computation (MPC) [55, 62, 81], homomorphic encryption (HE) [17, 39], or hybrid HE-MPC techniques [51, 57, 69]. These existing techniques provide cryptographic guarantees that user data is completely hidden from model providers. However, they assume that model providers host their services themselves, negating the cost, scalability, and robustness benefits enabled by public cloud infrastructure. If an ML provider hosted their service in the cloud instead, they would lose guarantees over the secrecy of their trained model.

Prior work has proposed hardware isolation using trusted execution environments (TEEs) as a solution for secure machine learning [74, 80, 86, 95, 96]. TEEs allow model providers to host their models on public clouds with privacy and integrity guarantees against cloud attackers. But, TEEs provide only marginal benefits to end users since users must still trust the model provider's code running inside of the TEE. Even the attestation property of TEEs, whereby a client can precisely verify the code running inside of the TEE, is of little use, as providers are still unwilling to share their intellectual property (IP) for independent auditing and inspection [10, 43, 52, 85, 93]. Further, independent and trustworthy auditing of model provider code becomes infeasible as the number of model providers continues to scale. What is lacking is a solution that provides scalability, flexibility, and security to model providers while si-

multaneously guaranteeing end-user privacy.

To address these concerns, we hypothesize that a careful integration of cryptographic *and* hardware isolation techniques can enable protection for both providers and users. We propose the CHEX-MIX protocol for oblivious inference in the cloud, based on a combination of HE and TEEs, as a concrete instantiation of this solution. The general idea ends up being quite simple: model providers run their model inside a TEE on their cloud provider of choice, and clients use HE when sending their data to the cloud for inference. The privacy of the client's data is protected by HE, while the privacy of the ML model provider's model parameters and the correctness of computation, including the integrity of the input and intermediate and final results, is protected by the TEE.

We note that a naive implementation of this solution would not satisfy the problem and would require assumptions incompatible with the problem setting. In particular, a combined solution requiring a client to attest the enclave would assume a client *can* validate the security of enclave code. Instead, we show how this requirement can be removed through the use of a new trust framework. This trust framework, which treats the model provider as a rational party, is stronger and more realistic than the passive adversary models of prior solutions in this setting. We show how our solution provides privacy guarantees to clients and model providers under a strong adversary model, tolerating malicious clients, cloud adversaries, and model providers, and correctness guarantees to clients tolerating malicious cloud adversaries and rational, actively adversarial ML model providers, while making no assumptions about collusion between the parties.

Next, we seek to understand the feasibility of an HE-TEE hybrid approach for oblivious inference in today's infrastructure. State-of-the-art HE-libraries are large and complex, while modern TEEs are resource-constrained and difficult to program. These conflicting characteristics suggest that an implementation of an HE-TEE hybrid solution would be impractical, if not impossible. In spite of this, we show through our implementation of CHEX-MIX on a commodity cloud server that not only is an HE-TEE hybrid solution possible, but it can also significantly *improve* computation and communication costs compared to prior solutions. Compared to a recent multi-key HE work [19], CHEX-MIX achieves a $3\times$ smaller communication cost and $3\times$ faster computation time, while providing additional privacy and security benefits and enabling providers to fully utilize the cloud.

Finally, we discuss how CHEX-MIX is able to offer model providers confidentiality of their inference code, which may in turn allow model providers more control over their intellectual property. We show how this variant of our solution is fully compatible with the baseline setup and how CHEX-MIX can offer this protection essentially *for free*.

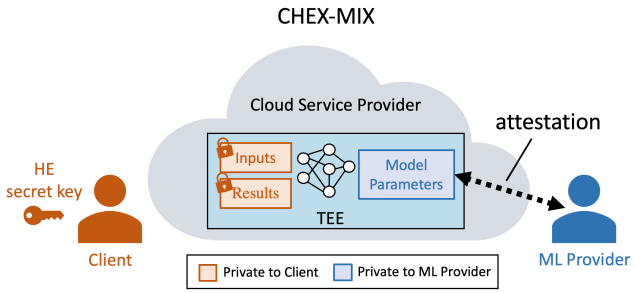

Figure 1: Overview of CHEX-MIX. The ML model provider attests and sends the model parameters to the cloud TEE. The client homomorphically encrypts and sends their inputs to the TEE, which homomorphically computes the inference results.

2 Background

2.1 Trusted Execution Environments

The shared nature of cloud environments often results in cloud computing stacks that are large and difficult to verify. Hidden vulnerabilities in complex stacks provide malicious actors more opportunities to access user data. TEEs help solve this problem by allowing users to define private regions of memory, called *enclaves*, in which users can store and operate on sensitive data with added protection. Several types of TEEs exist to date, including AMD SEV [101], ARM TrustZone [42], and Intel SGX [66]. We focus on SGX in this paper, but the principles discussed can be extended to similar TEEs.

To utilize an enclave, a user partitions their application code into trusted and untrusted components and loads the trusted component of the code into the enclave through a series of enclave setup procedures [28]. The user confirms that the enclave was created securely and loaded with the expected code using *(remote) attestation*, which provides protection against all attacks assumed by the SGX threat model [1, 66]. TLS channel establishment can be integrated with the attestation procedure for protection of data transferred between a user and an enclave [28, 54].

Still, SGX is limited in what it can guarantee; It does not guarantee protection of enclaves containing code written in an insecure manner (e.g., containing buffer overflow vulnerabilities or side-channel-producing computation). In order to trust the attested enclave, SGX users must also trust Intel attestation services to correctly identify trustworthy platforms.

2.2 Homomorphic Encryption

HE schemes are cryptographic schemes that enable computation on encrypted data. This property enables a user to outsource computation to an untrusted entity without requiring the user to reveal their private input or output values.

HE schemes consist of multiple stages. During parameter selection, HE parameters (namely, the ring degree and coef-

ficient moduli of the polynomial quotient rings) are typically chosen by users to be within the bounds of the HE Security Standard [18] consistent with the desired security level and the maximum multiplicative depth of the target computation circuit. During key generation, a user generates a secret key ($\mathbf{SK}$) for encryption/decryption using these parameters, as well as certain encryptions of the secret key called *evaluation keys* ($\mathbf{EK}$) that assist with outsourced computation. Encoding and encryption stages transform a vector of input values into a plaintext and then a ciphertext, while decryption and decoding reverse this process. Evaluation refers to the computation on ciphertexts performed by the untrusted party.

Homomorphic Inference. Prior works [17, 19, 25, 39, 48] have shown how it is possible to use HE for private neural network inference by interpreting linear network layers as a series of (homomorphic) additions, multiplications, and rotations between fixed-length vectors, and modeling non-linear network layers, such as ReLU, by low-degree polynomials. Efficient HE schemes such as BGV [15], BFV [14, 35], and CKKS [22] represent plaintexts and ciphertexts as elements in polynomial quotient rings and can operate on vectors of input values in a single-instruction-multiple-data (SIMD) fashion by using special encoding techniques [91]. Using these methods, prior works achieved an accuracy of $\geq$ 98.4% for inference over the MNIST dataset for HE evaluation of a 5-layer Convolutional Neural Network (CNN) [17].

The CKKS scheme allows users to efficiently discard unwanted precision in results of HE computation, essentially preserving an *approximate* computation on the input vector. Since this type of approximate computation is a good fit for machine learning tasks, which are already approximate in nature, CKKS is widely considered as the scheme of choice for ML computations [19, 48, 53, 58]. Given our target application of oblivious inference, we focus on the CKKS scheme in this work. However, our solution is not limited to the CKKS scheme and is applicable to other efficient HE schemes such as BFV and BGV as well.

Security of HE and CKKS. The CKKS scheme is based on the Ring Learning with Errors (RLWE) assumption and is IND-CPA secure. Though a recent work [58] demonstrated a passive key-recovery attack on the CKKS scheme, this attack requires access to a decryption oracle. A known simple mitigation for this attack, which we apply in this work, is to have the decryptor (user) not share the raw decryption results with any untrusted entity [21].[1]

Additionally, HE schemes are *malleable* by nature, so they offer no guarantee that an untrusted party computes a function $f()$ instead of a different function $f'()$ over an input during evaluation. Thus, HE alone cannot provide computational integrity. A lack of computational integrity in ML inference can be severe; a manipulation of the result of a medical image inference, for example, could lead to an incorrect diagnosis.

[1] Note that the general outcome of a decryption, such as the predicted class in a classification network, is not considered raw decryption.

2.3 Adversary Modeling

Before discussing our problem statement, we motivate the need for a model that more accurately captures real world adversarial behavior in the ML-as-a-service paradigm.

Traditionally in the field of multi-party computation and related works, adversaries are classified into one of two categories: *semi-honest* (or *honest-but-curious*) and *malicious*. A *semi-honest* adversary refers to an adversary that, given a prescribed protocol, will passively follow the protocol as described; thus, this adversary may only infer information about other parties from the protocol messages it receives. A *malicious* adversary, by contrast, may actively and arbitrarily deviate from the protocol specification. This adversary seeks to compromise either the privacy of participant data or the correctness of protocol execution.

We argue that these two models do not adequately reflect the behavior of real-world adversaries in the ML-as-a-service use case. The semi-honest model assumes that adversaries can only act passively, while many real-world adversaries typically actually posses the means to act actively [7, 60]. On the other hand, the malicious adversary model makes the overly cautious assumption that an adversary may deviate randomly and arbitrarily from the protocol, even if the adversary has no incentive to do so.

We need a model that considers the active capabilities of the adversary but can benefit from realistic assumptions on the adversarial party. To this end, we consider a third type of adversary known as a *rational* adversary that has capabilities nearly equivalent to that of a malicious adversary (i.e., is actively adversarial), but is bounded by their incentive to maximize their utility over a set of utility-providing actions. This adversary is described in several works at the intersection of multi-party computation and game theory as a realistic yet powerful assumption [4, 41, 65, 104]. In later sections, we show how we can utilize this model to provide clients with correctness of results during oblivious inference (without requiring a rational assumption for privacy of data).

3 Problem Statement

We consider the scenario of two-party oblivious neural network inference in the cloud. Specifically, we assume that there exists an ML model provider $\mathcal{M}$ with a pre-trained neural network model with weights and bias values, and a client $\mathcal{C}$ with inputs $\mathbf{X}$. We refer to $\mathcal{M}$'s weights and bias values collectively as $\mathbf{W}$ and $\mathcal{M}$'s model architecture (i.e., the number, ordering, types, and sizes of layers) as $\mathbf{F}$. We refer to inputs $\mathbf{X}$ and results $\mathbf{Y}$=$\mathbf{F}$($\mathbf{X}$, $\mathbf{W}$) collectively as *client data*.

Importantly, we wish to enable model providers to take advantage of the benefits of using the public cloud. These include, for example, the ability to scale a service to a large number of clients across the globe, as well as the flexibility and cost-effectiveness of dynamically scaling a service's resources as

needed based on demand. Thus, we introduce an additional entity $\mathcal{S}$ as the cloud service provider (e.g., Amazon AWS or Microsoft Azure). We aim to allow $\mathcal{M}$ to host their inference service on a server (or collection of servers) hosted by $\mathcal{S}$.

Given this setting, our goal is to allow the client $\mathcal{C}$ to submit their inputs to provider $\mathcal{M}$, who may be hosting their service on $\mathcal{S}$'s servers, to obtain the result of inference $\mathbf{Y}$=$\mathbf{F}(\mathbf{X}, \mathbf{W})$ in a manner that preserves the privacy of $\mathcal{C}$'s data and $\mathcal{M}$'s weights and the correctness of inference result $\mathbf{Y}$, thus satisfying the requirements for oblivious inference in the cloud.

3.1 Adversary Model

We consider three main types of adversaries in this setting: a cloud server adversary $\mathcal{A}_{\mathcal{S}}$, a client adversary $\mathcal{A}_{\mathcal{C}}$, and an ML model provider adversary $\mathcal{A}_{\mathcal{M}}$. The cloud server adversary $\mathcal{A}_{\mathcal{S}}$ is an untrusted cloud provider or a third-party attacker that uses vulnerabilities in the public cloud infrastructure to access $\mathcal{M}$'s or $\mathcal{C}$'s private data or tamper with computation execution. We assume $\mathcal{A}_{\mathcal{S}}$ may have full control of the OS or hypervisor layer of any public cloud-based servers. Meanwhile, the client adversary's goal is to try and learn as much about the model provider's weights and bias values as possible. We assume both $\mathcal{A}_{\mathcal{S}}$ and $\mathcal{A}_{\mathcal{C}}$ may be fully malicious.

We employ two separate adversary models for the model provider adversary $\mathcal{A}_{\mathcal{M}}$. First, we assume a malicious $\mathcal{A}_{\mathcal{M}}$, representing either a model provider themselves or a third-party adversary who is able to take control of $\mathcal{M}$'s servers.

Then, we consider an adversary model for $\mathcal{A}_{\mathcal{M}}$ to match the realistic use case of ML-as-a-service. For this case, we relax the malicious assumption slightly and treat $\mathcal{A}_{\mathcal{M}}$ as a rational adversary that seeks to maximize their utility under the following objectives: 1) learning the values of $\mathbf{X}$ or $\mathbf{Y}$ (i.e., $\mathcal{C}$'s private data) and 2) providing $\mathcal{C}$ with a useful inference service. Here we define "useful" as a service that will allow $\mathcal{M}$ to maintain enough users by some performance metric (see "best performing", Section 3.2). This means we can assume that $\mathcal{M}$ would like to provide $\mathcal{C}$ with a useful inference service and will in general not act in a way that counteracts this purpose *unless doing so might help reveal $\mathcal{C}$'s private data*. This constraint closely models real-world service provider adversaries who are unlikely to compromise the utility of their service, as this would deter users, but who may still seek to learn private user data given the high value of such data today [78, 110].

We make no assumptions about collusion between the adversaries in any of the above cases (i.e., we assume $\mathcal{A}_{\mathcal{M}}$ may collude with $\mathcal{A}_{\mathcal{S}}$, and $\mathcal{A}_{\mathcal{C}}$ may collude with $\mathcal{A}_{\mathcal{S}}$).

3.2 Privacy and Security Goals

Given the aforementioned setting, we target a solution providing the following guarantees:

1. Privacy of $\mathcal{C}$'s data $\mathbf{X}$ and $\mathbf{Y}$ against a malicious $\mathcal{A}_{\mathcal{S}}$ and malicious or rational $\mathcal{A}_{\mathcal{M}}$

Table 1: Summary of Terms / Symbols.

$\mathcal{C}$	Client / Feature Provider
$\mathcal{M}$	ML model provider
$\mathcal{S}$	Cloud server / Cloud service provider
$\mathcal{E}_{\mathcal{S}}$	Enclave established on $\mathcal{S}$
$\mathcal{A}_{\mathcal{C}}$	Adversary $\mathcal{C}$
$\mathcal{A}_{\mathcal{M}}$	Adversary $\mathcal{M}$
$\mathcal{A}_{\mathcal{S}}$	Adversary $\mathcal{S}$
π	HE parameters
$\mathbf{X}$, $\mathbf{Y}$	$\mathcal{C}$'s neural network feature inputs, outputs
$\mathbf{W}$	$\mathcal{M}$'s neural network weights and biases
$\mathbf{N}$	Intermediate inference results
$\overline{\mathbf{X}}$, $\overline{\mathbf{Y}}$, $\overline{\mathbf{N}}$	HE-encrypted $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{N}$
$\mathbf{F}$	Neural network inference circuit
$\mathbf{I_F}$	An implementation / code of $\mathbf{F}$
$\overline{\mathbf{I_F}}$	$\mathbf{I_F}$ encrypted with Intel PCL [45]
σ	Communication channel

2. Privacy of $\mathcal{M}$'s data $\mathbf{W}$ against a malicious $\mathcal{A}_{\mathcal{S}}$ and malicious $\mathcal{A}_{\mathcal{C}}$
3. Correctness of $\mathbf{Y}$ (for $\mathcal{C}$) against a malicious $\mathcal{A}_{\mathcal{S}}$ and rational $\mathcal{A}_{\mathcal{M}}$

To formalize Guarantee 3, we introduce some notation:

- Let P represent the service clients are expecting to interact with (e.g., a handwriting recognition service).
- Let T represent the set of all possible inference applications $\mathcal{M}$ may offer for service P.
- Let $\theta \in T$ be an inference application with some specific weights $\mathbf{W}$ and code $\mathbf{I_F}$ implementing a neural network circuit $\mathbf{F}$. For some input x from a client, $\theta(x) := \mathbf{I_F}(x,\mathbf{W}) = y_\theta$ is the result this application provides to the user.
- Denote as $\theta^* \in T$ the inference application that is considered "best-performing", where best-performing is determined by what ensures $\mathcal{M}$ maintains enough users (i.e., would provide enough utility to users that they would want to use the application[2]). A correct result should return $y_{\theta^*} = \theta^*(x)$ to the user upon receiving input x.

3.2.1 Scope.

Recent works have demonstrated how clients can infer information about models by analyzing the plaintext results of input queries (e.g. through *model-inversion* [97] or *membership inference* attacks [89, 98]). Protection against this class of attacks is not the focus of our work, and several other works detail mitigations that model providers can take against such attacks in practice [38, 50, 62, 89]. We discuss how these techniques can be implemented on top of our solution in more detail in Appendix C.

[2] Examples of a "best performing" implementation may be the fastest implementation of P known by $\mathcal{M}$ above a certain accuracy threshold, or the most accurate implementation of P known by $\mathcal{M}$ above a certain runtime threshold. In reality, there may be several such best-performing applications for any service P. In this case, θ^* may be any one of these applications.

Consistent with the threat model of SGX [49], we do not consider side-channel attacks as part of our threat model. However, we give a more detailed consideration of side channels in Section 8.2.

We also consider denial-of-service attacks out of scope.

3.2.2 Assumptions

We assume that all parties have access to standard network protection mechanisms for transferring private data over an untrusted network. In this work, we use the TLS protocol for secure channel establishment.

Additionally, prior HE works for inference [17, 19, 25, 39] implement any final softmax layers directly on the client device as part of the client decryption and decoding process since this layer is expensive to approximate for HE evaluation [23]. We employ the same approach in our evaluation, and we often refer to the "machine learning model" as the model without this final layer.

To securely share evaluation keys $\mathbf{EK}$ with untrusted parties, we make the usual circular security assumption used by prior HE works [15, 19].

Finally, we assume that $\mathcal{C}$ and $\mathcal{M}$ trust Intel attestation services to correctly identify when an enclave is malformed. (Though we do not require this assumption to ensure the privacy of $\mathcal{C}$'s data, we require it to ensure the privacy of $\mathcal{M}$'s data and the correctness of protocol execution.)

4 Naive Solution: TEE-only

Before we describe our solution, we discuss why a TEE-only solution would not fulfill our problem statement. In a TEE-only solution, a model provider creates an implementation of an inference service $\mathbf{I_F}$ to offer to clients through the public cloud. The model provider deploys $\mathbf{I_F}$ to an enclave in the cloud and releases the code for $\mathbf{I_F}$ somewhere accessible to the client. The client verifies the trustworthiness of $\mathbf{I_F}$, verifies that the enclave is a secure enclave running $\mathbf{I_F}$, and sends its private data to the enclave. When the enclave is done computing the results, it sends the results to the client.

While the above solution seems secure at first, protection for the client and the model provider relies on each party's ability to verify the trustworthiness of $\mathbf{I_F}$. Both parties must ensure $\mathbf{I_F}$ does not contain any vulnerabilities exploitable by $\mathcal{A}_\mathcal{S}$, and the client must additionally ensure $\mathbf{I_F}$ does not contain any opportunities for $\mathcal{A}_\mathcal{M}$ to access the enclave (e.g. through backdoors). It is plausible that the model provider could possess the ability to do this verification to a reasonable extent since software service providers often employ dedicated security teams to analyze code for vulnerabilities.

It is less plausible, however, that a client (who may just be an individual user) could achieve this verification in practice. Clients could rely on some trusted third-party service (e.g., the open-source community) to independently audit the code. However, this would require model providers to give up all secrecy of their inference code for independent auditing, which providers are often unwilling to do for IP protection [10, 43, 52, 93] or even security reasons [85]. Moreover, as the number of application settings and model providers continues to grow at a fast pace,[3] so too will the number of ML services, the complexity of inference backends, and the frequency of algorithmic updates. At this scale, it becomes infeasible for independent trusted third parties to fully verify the security of all inference services, even if their implementations were made publicly available.

We need a solution that removes the verification requirement from the client altogether. In the next section, we show how we can achieve this using our solution, CHEX-MIX.

5 Our Solution: CHEX-MIX

An overview of our solution is as follows: the model provider deploys their inference service in an enclave in the public cloud. A client wishing to use the service homomorphically encrypts their inputs and sends them to the enclave. The enclave then computes the inference result and sends the (still HE-encrypted) result to the client. Finally, the client decrypts the HE-encrypted result to obtain the result of inference.

5.1 Baseline Protocol

We describe our baseline protocol for two-party oblivious inference below, detailed further in Figure 2.

$\mathcal{M}$ begins the setup phase of the protocol by verifying that $\mathbf{I_F}$ securely implements $\mathbf{F}$ (where $\mathbf{I_F}$, along with weights $\mathbf{W}$, implements service P). Next, $\mathcal{M}$ establishes an enclave $\mathcal{E}_\mathcal{S}$ with code $\mathbf{I_F}$. Before provisioning $\mathcal{E}_\mathcal{S}$ with $\mathbf{W}$, $\mathcal{M}$ establishes an attested TLS channel with $\mathcal{E}_\mathcal{S}$ to ensure $\mathcal{E}_\mathcal{S}$ is correctly initialized and that the communication endpoint is the expected enclave. To later allow clients to connect to $\mathcal{E}_\mathcal{S}$ without attestation, $\mathcal{M}$ establishes a key-pair for the TLS channel establishment, obtains a certificate from a certificate authority (CA) trusted by both $\mathcal{M}$ and $\mathcal{C}$, and provisions $\mathcal{E}_\mathcal{S}$ with this certificate. $\mathcal{M}$ establishes a certificate chain by further issuing and provisioning $\mathcal{E}_\mathcal{S}$ with a certificate for $\mathcal{E}_\mathcal{S}$'s public key. At the end of this process, which is only required once across all clients, $\mathcal{M}$ exits the protocol and goes offline.

After $\mathcal{M}$'s setup, $\mathcal{C}$ connects to $\mathcal{E}_\mathcal{S}$ to receive the aforementioned certificates and the HE encryption parameters π to use. After verifying the parameters and certificates, $\mathcal{C}$ establishes a (*non*-attested) TLS channel with $\mathcal{E}_\mathcal{S}$. $\mathcal{C}$ generates HE keys $\mathbf{SK}$ and $\mathbf{EK}$ based on π and sends $\mathbf{EK}$ to $\mathcal{E}_\mathcal{S}$ over the secure channel. The setup phase for $\mathcal{C}$ is executed once per client.

Following the setup phases, $\mathcal{C}$ requests an inference result from $\mathcal{E}_\mathcal{S}$ based on its inputs $\mathbf{X}$. $\mathcal{C}$ encrypts $\mathbf{X}$ to $\overline{\mathbf{X}}$ using

[3] By one estimate, the machine learning market is expected to grow from USD 21 billion in 2022 to USD 210 billion by 2029 (a 10× increase in less than decade) [3].

Setup ($\mathcal{M}$) (One time)

1. $\mathcal{M}$ verifies that code $\mathbf{I_F}$ securely implements $\mathbf{F}$.
2. $\mathcal{M}$ initializes $\mathcal{E}_\mathcal{S}$ with $\mathbf{I_F}$ on a server hosted by $\mathcal{S}$.
3. $\mathcal{M}$ generates a key pair for TLS channel establishment.
4. $\mathcal{M}$ obtains a certificate $\mathbf{Cert}_\mathcal{M}$ for its public key from a CA trusted by both $\mathcal{M}$ and $\mathcal{C}$.
5. $\mathcal{M}$ attests and establishes a secure channel $\sigma_\mathcal{M}$ with $\mathcal{E}_\mathcal{S}$ for all future communication.
6. $\mathcal{M}$ sends $\mathbf{W}$ to $\mathcal{E}_\mathcal{S}$ over $\sigma_\mathcal{M}$.
7. $\mathcal{M}$ issues a certificate $\mathbf{Cert}_{\mathcal{E}_\mathcal{S}}$ for $\mathcal{E}_\mathcal{S}$'s public key and sends $(\mathbf{W}, \mathbf{Cert}_\mathcal{M}, \mathbf{Cert}_{\mathcal{E}_\mathcal{S}}, \pi)$ to $\mathcal{E}_\mathcal{S}$ over $\sigma_\mathcal{M}$.
8. $\mathcal{M}$ exits the protocol and goes offline.

Setup ($\mathcal{C}$) (Once per client)

1. $\mathcal{C}$ receives $(\mathbf{Cert}_\mathcal{M}, \mathbf{Cert}_{\mathcal{E}_\mathcal{S}}, \pi)$ from $\mathcal{E}_\mathcal{S}$.
2. $\mathcal{C}$ verifies $(\mathbf{Cert}_\mathcal{M}, \mathbf{Cert}_{\mathcal{E}_\mathcal{S}}, \pi)$ and establishes a secure channel $\sigma_\mathcal{C}$ with $\mathcal{E}_\mathcal{S}$ upon successful verification.
3. $\mathcal{C}$ generates HE keys $\mathbf{SK}$ and $\mathbf{EK}$ based on π.
4. $\mathcal{C}$ sends $\mathbf{EK}$ to $\mathcal{E}_\mathcal{S}$ over channel $\sigma_\mathcal{C}$.

Evaluation

1. $\mathcal{C}$ encrypts $\mathbf{X}$ to $\overline{\mathbf{X}}$ using $\mathbf{SK}$.
2. $\mathcal{C}$ sends $\overline{\mathbf{X}}$ to $\mathcal{E}_\mathcal{S}$ over channel $\sigma_\mathcal{C}$.
3. $\mathcal{E}_\mathcal{S}$ computes $\overline{\mathbf{Y}} = \texttt{Eval}(\mathbf{I_F}, \overline{\mathbf{X}}, \mathbf{W}, \mathbf{EK})$.
4. $\mathcal{E}_\mathcal{S}$ sends $\overline{\mathbf{Y}}$ to $\mathcal{C}$ over channel $\sigma_\mathcal{C}$.
5. $\mathcal{C}$ decrypts $\overline{\mathbf{Y}}$ with $\mathbf{SK}$ to obtain result $\mathbf{Y}$.

Figure 2: CHEX-MIX baseline protocol.

$\mathbf{SK}$, sends $\overline{\mathbf{X}}$ to $\mathcal{E}_\mathcal{S}$ over the established network channel, and waits for $\mathcal{E}_\mathcal{S}$ to compute the neural network evaluation. $\mathcal{C}$ then receives the (still HE encrypted) result $\overline{\mathbf{Y}}$ from $\mathcal{E}_\mathcal{S}$ and decrypts and decodes $\overline{\mathbf{Y}}$ to $\mathbf{Y}$ using $\mathbf{SK}$ to obtain the result.

5.1.1 Protocol Details

All communication between clients and the enclave and between the model provider and the enclave must be secured by the TLS protocol. Once two entities establish a secure communication channel in the protocol, all further communication between them occurs over that channel. This is essential to protect the integrity of all data, keys, and certificates and the confidentiality and integrity of sensitive metadata/headers associated with network packets themselves.

Since evaluation keys $\mathbf{EK}$ are types of HE ciphertexts, they also require integrity protection during and after use. During evaluation, the enclave $\mathcal{E}_\mathcal{S}$ provides this integrity protection of $\mathbf{EK}$. After evaluation, $\mathcal{E}_\mathcal{S}$ can store each client's $\mathbf{EK}$ in an integrity-protected database and retrieve the client's $\mathbf{EK}$ from the database upon subsequent connections. $\mathcal{M}$ may implement this service within the evaluation enclave itself, by establishing a separate enclave or set of enclaves for this purpose (for example, using an SGX-based database design such as those detailed in prior works [79, 100]), or by using some other integrity-protected database scheme.

Finally, unlike inference backends, the client-side HE code can be open-sourced, reviewed by trusted experts, and standardized, while also remaining relatively stable. It is thus possible for users to trust client-side HE code.

5.1.2 Security Analysis

In Lemmas 1 and 2 below, we show how CHEX-MIX provides Guarantees (1) and (2) from Section 3.2 in the presence of malicious adversaries $\mathcal{A}_\mathcal{S}$, $\mathcal{A}_\mathcal{C}$, and $\mathcal{A}_\mathcal{M}$. Since a malicious adversary can exhibit any behavior (including acting rationally), this analysis also holds for a rational $\mathcal{A}_\mathcal{M}$. Note that Guarantees (1) and (2) imply that the underlying values $\mathbf{N}$ of any intermediate ciphertexts of an HE-inference network should not be revealed to any entity, since doing so could potentially allow the entity to learn the values of $\mathbf{X}$ or $\mathbf{Y}$ if $\mathbf{W}$ is known, or the values of $\mathbf{W}$ if $\mathbf{X}$ or $\mathbf{Y}$ is known, through simple calculations [13] (we omit consideration for the trivial case of all-zero $\mathbf{W}$). Similarly, Guarantee (2) implies that the intermediate ciphertexts $\overline{\mathbf{N}}$ (i.e., the intermediate values still in encrypted form) should not be revealed to any entity other than $\mathcal{M}$, since knowledge of these values and $\overline{\mathbf{X}}$ or $\overline{\mathbf{Y}}$ could also easily allow an adversary to compute the values of $\mathbf{W}$.

In Lemma 3, we show how CHEX-MIX provides Guarantee (3) from Section 3.2 in the presence of a malicious adversary $\mathcal{A}_\mathcal{S}$ and a rational adversary $\mathcal{A}_\mathcal{M}$. If $\mathbf{I_F}$ is an HE inference application, then $x = (\overline{\mathbf{X}}, \mathbf{EK})$ and $y = \theta(x) = \mathbf{I_F}(\overline{\mathbf{X}}, \mathbf{EK}, \mathbf{W}) = \overline{\mathbf{Y}}$. For y to be considered correct in this case, $\texttt{Decrypt}_\mathbf{SK}(y)$ should return some final result $\mathbf{Y}$ to the client to ensure θ satisfies the "best-performing" property (implying $\theta = \theta^*$). We say $\mathbf{Y}$ is correct if $\overline{\mathbf{Y}} = \theta^*(x)$ and $\mathbf{Y} = \texttt{Decrypt}_\mathbf{SK}(\overline{\mathbf{Y}})$.

LEMMA 1. CHEX-MIX ensures privacy of $\mathcal{C}$'s inputs $\mathbf{X}$ and results $\mathbf{Y}$ from a malicious $\mathcal{A}_\mathcal{S}$ and malicious $\mathcal{A}_\mathcal{M}$.

Proof Sketch. $\mathcal{C}$ uses HE to encrypt $\mathbf{X}$ to $\overline{\mathbf{X}}$. $\mathcal{C}$ only sends the encrypted result $\overline{\mathbf{X}}$ to $\mathcal{E}_\mathcal{S}$. By the IND-CPA property of HE, $\overline{\mathbf{X}}$ does not reveal $\mathbf{X}$, $\mathbf{Y}$, or $\mathbf{N}$ to passive or active adversaries. $\mathcal{C}$ performs key generation and encryption locally in a trusted environment and never shares $\mathbf{SK}$, $\mathbf{X}$, or $\mathbf{Y}$ with untrusted parties. Since $\overline{\mathbf{X}}$ does not reveal $\mathbf{X}$, $\mathbf{Y}$, or $\mathbf{N}$, and since no party but $\mathcal{C}$ has the secret key $\mathbf{SK}$ or direct access to $\mathbf{X}$ or $\mathbf{Y}$, CHEX-MIX guarantees the privacy of $\mathbf{X}$ and $\mathbf{Y}$ for $\mathcal{C}$ against all malicious adversaries, including a malicious $\mathcal{A}_\mathcal{S}$ and malicious $\mathcal{A}_\mathcal{M}$. □

LEMMA 2. CHEX-MIX ensures privacy of $\mathcal{M}$'s weights and biases $\mathbf{W}$ from a malicious $\mathcal{A}_\mathcal{S}$ and malicious $\mathcal{A}_\mathcal{C}$.

Proof Sketch. $\mathcal{M}$ verifies that $\mathbf{I_F}$ contains no vulnerabilities exploitable by $\mathcal{A}_\mathcal{C}$ or $\mathcal{A}_\mathcal{S}$ to access enclave data. $\mathcal{M}$ uses remote attestation to guarantee that $\mathcal{E}_\mathcal{S}$ is a valid SGX enclave containing code $\mathbf{I_F}$. Since $\mathcal{E}_\mathcal{S}$ contains $\mathbf{I_F}$, and since $\mathbf{I_F}$ is not exploitable by $\mathcal{A}_\mathcal{C}$ or $\mathcal{A}_\mathcal{S}$, $\mathcal{A}_\mathcal{C}$ and $\mathcal{A}_\mathcal{S}$ cannot access $\mathbf{W}$ or $\overline{\mathbf{N}}$ through $\mathcal{E}_\mathcal{S}$. $\mathcal{M}$ securely transfers $\mathbf{W}$ to $\mathcal{E}_\mathcal{S}$ via an attested TLS channel and does not share $\mathbf{W}$ or $\overline{\mathbf{N}}$ with any untrusted

parties. Since the TLS channel protects the privacy of data against active attacks, $\mathcal{A}_\mathcal{C}$ and $\mathcal{A}_\mathcal{S}$ cannot access $\mathbf{W}$ during data transfer. Since $\mathcal{A}_\mathcal{C}$ and $\mathcal{A}_\mathcal{S}$ cannot access $\mathbf{W}$ or $\overline{\mathbf{N}}$ during data transfer or at either endpoint, they cannot access $\mathbf{W}$ or $\overline{\mathbf{N}}$ at any point in the protocol and the privacy of $\mathbf{W}$ for $\mathcal{M}$ is guaranteed. □

LEMMA 3. CHEX-MIX guarantees the correctness of results $\mathbf{Y}$ for $\mathcal{C}$ against a malicious $\mathcal{A}_\mathcal{S}$ and a rational $\mathcal{A}_\mathcal{M}$.

Proof Sketch. To ensure computation correctness, it is necessary to ensure that $\mathcal{M}$ deploys some $\theta = \theta^*$ as the inference service (Requirement 1) and that the result $\overline{\mathbf{Y}}$ provided to $\mathcal{C}$ is the result y of applying this θ^* to $\mathcal{C}$'s inputs x (i.e., $y = \theta^*(x) = \overline{\mathbf{Y}}$) (Requirement 2). There are three possible scenarios for $\mathcal{M}$'s deployments: $\mathcal{M}$ deploys some $\theta \neq \theta^*$ (Scenario 1), $\mathcal{M}$ does not deploy any service at all (Scenario 2), and $\mathcal{M}$ deploys some $\theta = \theta^*$ (Scenario 3). Let U_i denote the utility of Scenario i to $\mathcal{A}_\mathcal{M}$.

First, assume $\mathcal{M}$ deploys some $\theta \neq \theta^*$ (Scenario 1). This directly contradicts objective 2 of a rational $\mathcal{A}_\mathcal{M}$ as defined in Section 3.1 to provide $\mathcal{C}$ with a useful inference service. Thus, $\mathcal{M}$ would only deploy $\theta \neq \theta^*$ to learn the values of $\mathbf{X}$ or $\mathbf{Y}$ (objective 1). By Lemma 1, $\mathcal{A}_\mathcal{M}$ cannot learn the values of $\mathbf{X}$ or $\mathbf{Y}$ and thus cannot achieve objective 1. Since $\theta \neq \theta^*$ would not be considered a useful service, it would not allow $\mathcal{M}$ to achieve objective 2. Since $\mathcal{A}_\mathcal{M}$ cannot achieve objectives 1 or 2 in Scenario 1, $U_1 = 0$.

Next, assume $\mathcal{M}$ deploys no service at all (Scenario 2). Like in Scenario 1, Lemma 1 ensures $\mathcal{A}_\mathcal{M}$ cannot learn $\mathbf{X}$ or $\mathbf{Y}$ (objective 1). Also, failing to deploy a service is not useful to $\mathcal{C}$ (objective 2). Therefore, $U_2 = 0$.

Finally, assume $\mathcal{M}$ deploys some $\theta = \theta^*$ (Scenario 3). Since θ allows $\mathcal{M}$ to maintain enough users, it is considered a useful service and allows $\mathcal{A}_\mathcal{M}$ to achieve objective 2. Since $\mathcal{A}_\mathcal{M}$ achieves at least objective 2 in Scenario 3, $U_3 > 0$. Since $U_3 > U_1$ and $U_3 > U_2$, $\mathcal{A}_\mathcal{M}$ enacts Scenario 3 and attempts to deploy some $\theta = \theta^*$, satisfying Requirement 1.

In order for $\mathcal{M}$ to ensure the deployed θ is the intended θ^*, $\mathcal{M}$ must ensure that the $\mathbf{I_F}$ constituting θ implements $\mathbf{F}$ using $\mathbf{W}$ for the service P, where $\theta(x) = \overline{\mathbf{Y}} = \mathbf{I_F}(\overline{\mathbf{X}}, \mathbf{EK}, \mathbf{W})$ is a useful enough result for service P to satisfy the "best performing" requirement for $\theta = \theta^*$. Any vulnerability in $\mathbf{I_F}$ that could lead to $\mathcal{A}_\mathcal{S}$ tampering with results such that $\overline{\mathbf{Y}}$ is no longer a useful result would lead to $\theta \neq \theta^*$. Thus, to ensure $\theta = \theta^*$, $\mathcal{M}$ will verify that $\mathbf{I_F}$ is free of vulnerabilities exploitable by $\mathcal{A}_\mathcal{S}$ to corrupt $\overline{\mathbf{Y}}$.

Standard certificate verification with attestation ensures $\mathcal{M}$ securely communicates with $\mathcal{E}_\mathcal{S}$ to deploy the verified $\mathbf{I_F}$ and corresponding $\mathbf{W}$ and ensures $\mathcal{A}_\mathcal{S}$ cannot tamper with $\mathbf{I_F}$ or $\mathbf{W}$. Standard certificate verification ensures $\mathcal{C}$ communicates with an entity set up by $\mathcal{M}$. $\mathcal{M}$ verifies this entity is $\mathcal{E}_\mathcal{S}$ to ensure protection against $\mathcal{A}_\mathcal{S}$ and satisfy $\theta = \theta^*$. $\mathcal{C}$ securely sends its inputs $x = (\overline{\mathbf{X}}, \mathbf{EK})$ to $\mathcal{E}_\mathcal{S}$ and receives results y from $\mathcal{E}_\mathcal{S}$ using the TLS channel. The TEE protects all data and code execution from $\mathcal{A}_\mathcal{S}$ once in the enclave. Since a rational $\mathcal{A}_\mathcal{M}$ deploys a useful service, and since the deployed service computation and execution is protected at all points from tampering by $\mathcal{A}_\mathcal{S}$, the deployed service θ is equal to θ^*. Since $\theta = \theta^*$, the result y returned to $\mathcal{C}$ is $\overline{\mathbf{Y}} = \theta^*(x)$, satisfying Requirement 2. Since CHEX-MIX satisfies Requirements 1 and 2, it provides $\mathcal{C}$ with correct results $y = \overline{\mathbf{Y}}$, which $\mathcal{C}$ decrypts to obtain the final correct result $\mathbf{Y}$. □

5.1.3 Takeaways.

CHEX-MIX is beneficial to $\mathcal{C}$ and $\mathcal{M}$ in several ways. First, it relieves $\mathcal{C}$ of the burden of needing to thoroughly analyze code $\mathbf{I_F}$ for security vulnerabilities by removing the client attestation requirement.[4] Second, it guarantees privacy of provider data $\mathbf{W}$, privacy of client data $\mathbf{X}$ and $\mathbf{Y}$, and correctness of inference execution $\mathbf{F}$ even while deployed in the public cloud. Third, it allows $\mathcal{M}$ to remain offline after the initial setup phase of the protocol and offload all subsequent computation. Altogether, CHEX-MIX allows ML model providers to securely utilize the scale made possible by the cloud.

5.2 Achieving Confidentiality of $\mathbf{I_F}$

While the baseline protocol protects the implementation $\mathbf{I_F}$ from integrity attacks during computation, it does not natively protect the privacy of the $\mathbf{I_F}$ binary itself. In particular, the baseline protocol requires $\mathcal{M}$ to send the $\mathbf{I_F}$ binary to the cloud server in-the-clear to instantiate the enclave $\mathcal{E}_\mathcal{S}$, allowing attackers in the cloud direct access to the $\mathbf{I_F}$ binary. Several prior works have shown that it is possible to reverse-engineer binaries to learn details of its original application code [9, 16, 56, 105]. Thus, this setup leaves $\mathbf{I_F}$ vulnerable to confidentiality attacks.

Confidentiality of $\mathbf{I_F}$ code would be beneficial to service providers. Software implementations of services contain decisions based on a provider's expertise, such as algorithm choice or robust coding practices, and are thus often considered valuable IP [10, 43, 52, 93]. Therefore, protecting the details of the software implementation could protect the valuable IP that allows the provider to maintain a competitive advantage.

To solve the problem of maintaining software IP privacy in the public cloud, Intel released a Protected Code Loader (PCL) library [45] for SGX to allow users to treat an enclave binary as private enclave data. PCL encrypts several sections of the generated shared object binary of the enclave code (see: Appendix E for an example of a PCL-encrypted elf file). Intel describes PCL as a tool that can protect a developer's private code IP [47] and asserts that it is the provider's responsibility to ensure that the segments of code which PCL does not encrypt (e.g., the code loader itself, the BSS segment, and

[4] Appendix A provides a discussion on any added protection that including client attestation in the protocol would provide, if desired.

Setup ($\mathcal{M}$) (One time)

1. $\mathcal{M}$ verifies that code $\mathbf{I_F}$ securely implements $\mathbf{F}$.
2. $\mathcal{M}$ generates a symmetric (e.g., AES-GCM) encryption key k for use with Intel PCL.
3. $\mathcal{M}$ uses Intel's PCL to encrypt the shared binary object of $\mathbf{I_F}$ into $\overline{\mathbf{I_F}}$ using k.
4. $\mathcal{M}$ establishes an enclave $\mathcal{E}_{\mathcal{S}_1}$ on a server hosted by $\mathcal{S}$.
5. $\mathcal{M}$ generates a key pair for TLS channel establishment.
6. $\mathcal{M}$ obtains a certificate $\mathbf{Cert}_{\mathcal{M}}$ for its public key from a CA trusted by both $\mathcal{M}$ and $\mathcal{C}$.
7. $\mathcal{M}$ attests and establishes a secure channel $\sigma_{\mathcal{M}}$ with $\mathcal{E}_{\mathcal{S}_1}$ for all future communication.
8. $\mathcal{M}$ sends k to $\mathcal{E}_{\mathcal{S}_1}$ over $\sigma_{\mathcal{M}}$.
9. $\mathcal{E}_{\mathcal{S}_1}$ seals k to server storage and terminates.
10. $\mathcal{M}$ repeats steps 4 through 7 with enclave $\mathcal{E}_{\mathcal{S}_2}$, where $\mathcal{E}_{\mathcal{S}_2}$'s binary consists of $\overline{\mathbf{I_F}}$ and Intel PCL components.
11. $\mathcal{E}_{\mathcal{S}_2}$ unseals k, decrypts $\overline{\mathbf{I_F}}$ to $\mathbf{I_F}$ using k, and executes $\mathbf{I_F}$
12. $\mathcal{M}$ sends $\mathbf{W}$ to $\mathcal{E}_{\mathcal{S}_2}$ over $\sigma_{\mathcal{M}}$.
13. $\mathcal{M}$ issues a certificate $\mathbf{Cert}_{\mathcal{E}_{\mathcal{S}_2}}$ for $\mathcal{E}_{\mathcal{S}_2}$'s public key and sends $(\mathbf{W}, \mathbf{Cert}_{\mathcal{M}}, \mathbf{Cert}_{\mathcal{E}_{\mathcal{S}_2}}, \pi)$ to $\mathcal{E}_{\mathcal{S}_2}$ over $\sigma_{\mathcal{M}}$.
14. $\mathcal{M}$ exits the protocol and goes offline.

Figure 3: CHEX-MIX-PCL protocol for $\mathbf{I_F}$ IP privacy for $\mathcal{M}$

any debugging info, with the full list given in the PCL documentation [47]) do not contain sensitive information. Since we assume $\mathcal{M}$, with its dedicated security team, can adequately vet the security of their own enclave code, we also assume $\mathcal{M}$ can verify that the sensitive components of its private IP are not contained in the unencrypted segments.

To utilize Intel's PCL as part of our solution, we only need to make minor modifications to part of $\mathcal{M}$'s setup phase in the CHEX-MIX baseline protocol. After analyzing the code for vulnerabilities as before, $\mathcal{M}$ uses Intel's PCL to encrypt the shared object file for the code $\mathbf{I_F}$. $\mathcal{M}$ shares this encrypted file with the enclave $\mathcal{E}_{\mathcal{S}}$, which then decrypts and runs the $\mathbf{I_F}$ binary from inside the enclave. We give a revised setup protocol for $\mathcal{M}$ incorporating these modifications in Figure 3 (CHEX-MIX-PCL). Note that the logical enclave $\mathcal{E}_{\mathcal{S}}$ is actually split into two enclaves $\mathcal{E}_{\mathcal{S}_1}$ and $\mathcal{E}_{\mathcal{S}_2}$ in the protocol description, consistent with the actual implementation of the PCL library. $\mathcal{E}_{\mathcal{S}_1}$ shares data with $\mathcal{E}_{\mathcal{S}_2}$ by encrypting the desired shared data using a *sealing key*, which $\mathcal{E}_{\mathcal{S}_1}$ derives from the identity of the enclave developer, and which can be obtained only by enclaves signed by that developer. Since both $\mathcal{E}_{\mathcal{S}_1}$ and $\mathcal{E}_{\mathcal{S}_2}$ are developed and signed by developer $\mathcal{M}$, $\mathcal{E}_{\mathcal{S}_2}$ can then decrypt the file containing the secret data after deriving the sealing key in a similar manner. In the above procedure using PCL, this "secret data" is the key k used to encrypt $\mathbf{I_F}$.

This version of our solution provides the following guarantee in addition to the baseline protocol guarantees:

LEMMA 4. CHEX-MIX-PCL ensures confidentiality of $\mathbf{I_F}$ for $\mathcal{M}$ against a malicious $\mathcal{A}_{\mathcal{C}}$ and malicious $\mathcal{A}_{\mathcal{S}}$.

Proof Sketch. $\mathcal{M}$ verifies via remote attestation that $\mathcal{E}_{\mathcal{S}_1}$ is a secure enclave containing code to receive and seal a key k from $\mathcal{M}$, and that $\mathcal{E}_{\mathcal{S}_1}$ contains no vulnerabilities exploitable by $\mathcal{A}_{\mathcal{C}}$ or $\mathcal{A}_{\mathcal{S}}$ to access enclave code or data. $\mathcal{M}$ securely transfers key k to $\mathcal{E}_{\mathcal{S}}$ via an attested TLS channel. $\mathcal{M}$ verifies that $\overline{\mathbf{I_F}}$ fully encrypts the components of its code that it considers private IP. $\mathcal{M}$ verifies via remote attestation that $\mathcal{E}_{\mathcal{S}_2}$ is a secure enclave containing code $\overline{\mathbf{I_F}}$, code to unseal a key k, and PCL code to decrypt $\overline{\mathbf{I_F}}$ into $\mathbf{I_F}$. The TEEs ensure $\mathcal{A}_{\mathcal{C}}$ and $\mathcal{A}_{\mathcal{S}}$ cannot access code key k. Since $\mathcal{M}$'s private code IP is only decrypted in enclave $\mathcal{E}_{\mathcal{S}_2}$, and since only $\mathcal{M}$ and the TEEs possess the key to decrypt code $\overline{\mathbf{I_F}}$ into $\mathbf{I_F}$, the confidentiality of $\mathcal{M}$'s private code IP is guaranteed. □

6 Related Work

In this section, we describe alternate approaches proposed by prior works for oblivious inference. We intentionally leave this section detailed to provide more context for our evaluation. We provide a summary of the main points discussed in this section in Table 2.

TEE-Only. TEE-only solutions for oblivious inference [74, 80, 86, 95, 96] heavily rely on remote attestation to guarantee privacy of client data. Since all client data in these solutions is processed in-the-clear within the enclaves, *any security vulnerabilities in the enclave code, including any back doors purposely inserted by the model provider, could expose direct access to enclave data.* Malicious security in TEE-only solutions thus requires the impractical assumption that clients can thoroughly verify or blindly trust the security of enclave code, whereas our solution provides malicious security for privacy protection without this assumption.

HE-Only. CryptoNets [39] and LoLa [17] propose HE-based solutions for two-party oblivious inference. In these works, clients encrypt their private inputs and send the resulting ciphertexts to an untrusted server for homomorphic inference. However, these solutions require the server to maintain a private online infrastructure to perform the homomorphic evaluation, preventing the model provider from making use of the public cloud. Alternatively, E2DM [48] proposes having both the client and the model provider homomorphically encrypt their data under the client's key, allowing an untrusted cloud to perform the HE evaluation. However, this technique assumes a *semi-honest* cloud and an *honest* client.

MKHE / Hybrid-MKHE. Multi-key HE (MKHE) [19, 20, 59] enables a client and model provider to encrypt their private values using their respective secret keys and outsource homomorphic evaluation to an untrusted cloud server. The final result ciphertext of the MKHE inference, now encrypted under the keys of both parties, must be partially decrypted by the model provider before the final client device decryption. Thus, this technique still requires the model provider to maintain a private online infrastructure for partial decryption. More importantly, this technique is only secure against even just

Table 2: A combined approach of HE+TEE provides security guarantees unaddressed by prior approaches for two-party oblivious inference, while allowing $\mathcal{M}$ to offload computation to the public cloud. (R) refers to a "rational" adversary as defined in Section 3.1, and (M) refers to a "malicious" adversary type. Dashes (–) indicate that a feature does not apply, since the technique does not allow offload to an untrusted cloud.

Technique	$\mathcal{M}$ can fully offload	Privacy of $\mathbf{X}$, $\mathbf{Y}$ (for $\mathcal{C}$)		Correctness of $\mathbf{Y}$ (for $\mathcal{C}$)		Privacy of $\mathbf{W}$ (for $\mathcal{M}$)	
		$\mathcal{A}_{\mathcal{M}}$ (M)	$\mathcal{A}_{\mathcal{S}}$ (M)	$\mathcal{A}_{\mathcal{M}}$ (R)	$\mathcal{A}_{\mathcal{S}}$ (M)	$\mathcal{A}_{\mathcal{C}}$ (M)	$\mathcal{A}_{\mathcal{S}}$ (M)
TEE	✓	✗	✓	✗	✗	✓	✓
HE	✗	✓	–	✓	–	✓	–
MKHE	✗	✓	✓	✓	✗	✗	✗
2PC	✗	✗[1]	–	✗[1]	–	✗[2]	–
HE+2PC	✗	✗[1]	–	✗[1]	–	✗/✓[2]	–
HE+TEE	✓	✓	✓	✓	✓	✓	✓

[1] Most prior 2PC or HE-2PC solutions such as [51,69] assume only a semi-honest/passive model provider.
[2] Most prior 2PC and HE-2PC solutions, with the exception of MUSE [57], assume a semi-honest client.

semi-honest adversaries if the partial decryption is performed with a secure method such as noise flooding.

To the best of our knowledge, an MKHE-TEE hybrid solution has not been proposed or evaluated in any prior work. However, our results in Tables 3 and 4 indicate that this solution would be more expensive both in terms of performance and communication cost than CHEX-MIX since Eff.-MKHE is less efficient in both respects even when no part of the protocol is executed inside a TEE.

2PC / Hybrid-2PC. Here we use the terms 2PC or MPC to refer to protocols built with secret sharing, oblivious transfer, and/or garbled circuits. We further use the terms hybrid-2PC or hybrid-MPC to refer to works that use MPC in conjunction with additional techniques such as HE.

Several 2PC and hybrid-2PC protocols have been proposed for the problem of oblivious inference [2,44,51,57,62,69,71,73,77,81]. However, all of these works assume the model provider maintains a private infrastructure for protocol execution and thus do not utilize the public cloud. Furthermore, with the exception of MUSE [57], these works make a much weaker semi-honest adversary assumption for both model provider and client adversaries, which MUSE demonstrates can lead to devastating results in the client-malicious setting. Additionally, while some works claim that the benefit of 2PC techniques over HE is their ability to evaluate "unmodified" non-polynomial activation functions, we note that prior works nevertheless choose to implement truncated versions of these activation functions [57,69] or prefer using square activations for some or all of the activation layers anyway [69] to reduce computation and communication costs.

An interesting MPC-TEE hybrid solution was proposed in prior work CryptFlow [55], though unfortunately, without any experimental results for inference in the 2PC setting. Their protocol also assumes the *client* has a TEE for the model provider to attest in addition to requiring the client to attest the TEE of the model provider. The authors claim their solution's benefit over a TEE-only solution is that they do not assume confidentiality of TEE state; however, they still require integrity of TEE state, and it is not clear whether a TEE that could provide one without the other would exist in practice.

Other than the aforementioned work, to the best of our knowledge, no prior work has proposed or evaluated a 2PC-TEE or hybrid-2PC-TEE solution. Our evaluation suggest that comparable 2PC solutions and 2PC-HE solutions have a larger communication cost and runtime than CHEX-MIX, and the runtimes of these works will further increase if moved to a TEE. Additionally, naive placement of a semi-honest 2PC solution inside a TEE would not necessarily guarantee more than semi-honest security, and as we previously noted with respect to prior work [55] above, it is not clear whether such a solution would be more beneficial than a TEE-only solution. Still, this may be an interesting direction for future work.

3PC. Known 3PC solutions for oblivious inference [55,70,82,102,103] require two of the three parties to act honestly. This requirement is difficult to set up in practice since it requires either an honest third-party server or a non-collusion assumption between public cloud servers. In addition, these solutions also include a large communication cost to share the model with all three parties.

Zero-knowledge proofs. Known zero-knowledge proof solutions for secure machine learning [36,63,108] can only either provide privacy to $\mathcal{M}$ *or* $\mathcal{C}$, but not both. Therefore, they do not address the problem of oblivious inference.

Other HE-TEE. Most prior HE-TEE works [27,37,107] propose using TEEs to perform sensitive (e.g., requiring access to the client's secret key) stages of the HE flow. Thus, they share the drawback of TEE-only solutions that the enclave code must be thoroughly verified by a client, who must possess an unrealistic level of expertise to guarantee that enclave code is free from vulnerabilities for data privacy. Further, since HE evaluation still occurs outside the enclave, they offer no integrity protections against a malicious server. These works also target single-party outsourced computation rather than two-party oblivious inference.

Drucker and Gueron [33,34] suggest combining HE and TEEs for outsourced computation. However, their evaluation over a database query uses the significantly less computationally powerful Pallier HE scheme. Thus, their demonstration is

only applicable to a small range of problems and cannot be applied to neural network inference. The authors also do not present a security analysis of their solution or specify when TEEs can be trusted to provide integrity but not confidentiality to justify their need for HE.

In their master's thesis [90], Singh evaluates a version of the TFHE HE library [24,94] for a "fused" millionaire problem inside an enclave for a single party. The author does not address protecting the privacy of two separate parties, nor present an evaluation of an ML benchmark. Since this work requires bootstrapping after each boolean arithmetic operation and does not use SIMD packing, it is also less communication and computation efficient for our problem setting.

7 Evaluation

7.1 Implementation Details

To evaluate the feasibility of our solution, we implement CHEX-MIX across three experiments on a commodity cloud server using Microsoft Azure. In all experiments, we use the Microsoft SEAL library [87] to implement the HE-component of our solution.

Experiment 1: Baseline Protocol. We first demonstrate a proof of concept of CHEX-MIX for inference over a CNN. We target the same CNN as used in the Eff.-MKHE [19] work, which consists of 1 convolutional, 2 fully connected, and 2 square activation layers (see Table 6 for further details).

For this experiment, we use the OpenEnclave SDK [76] to develop the enclave code. We prefer the OpenEnclave SDK to other enclave SDKs since OpenEnclave provides extensive documentation on how to integrate secure channel establishment (i.e., TLS) with attestation. We use the `attested_tls` sample code with the mbedTLS networking library [5] as a starting point for our experiment, and we disable any optional performance optimization flags in the enclave-side instance of Microsoft SEAL that are incompatible with OpenEnclave.

We do not integrate an integrity-protected database for $\mathbf{EK}$ into our experiment since it is not the focus of our work. Any effort required to retrieve the $\mathbf{EK}$ for a client can be done between an initial "client hello" and subsequent client communication. Furthermore, consecutive requests from the same client would not require additional database access. Thus, we assume this cost can be hidden from the effective runtime.

Experiment 2: $\mathbf{I_F}$ Privacy Protocol We also implement the version of our solution to provide $\mathcal{M}$ with privacy of code $\mathbf{I_F}$ using Intel PCL [45]. We use a similar setup in this experiment as in Experiment 1 and target the same CNN. However, since OpenEnclave does not yet support Intel PCL integration, we instead use the Intel SGX SDK for this implementation. For simplicity, and since we already implement the baseline protocol using full attestation and secure channel establishment procedures, we do not replicate these components in this proof of concept. Instead, we fill the HE plaintext and ciphertext objects used by the server with random values. We run this proof of concept both with and without PCL to understand the overhead of adding PCL-provided protection.

Experiment 3: Evaluating Scalability Given the relatively recent introduction of efficient HE schemes such as CKKS, a benchmark suite (or even another efficient stand-alone network implementation) is not yet available for homomorphic inference. While some compilers exist for auto-generating HE evaluation code [11,12,29,30], the efficiency of their compiled outputs is still much lower than that of hand-optimized implementations by HE experts. The state-of-the-art HE compiler EVA [29] (which subsumes prior work CHET [30]), for example, does not implement known essential optimizations such as merging of adjacent linear network layers [39], or using the baby-step-giant-step algorithm for fully connected layers [40]. Even using *56 threads*, EVA incurs a latency of 0.6 seconds – over *triple* the latency of the *single*-threaded hand-optimized HE neural network implementation we use in this work – to evaluate a similarly sized network.

Given the currently limited ability of modern HE compilers to produce efficient inference code, it would be unfair to use the output of these compilers to analyze the efficiency of our solution against other (non-HE-only) approaches for oblivious inference. However, we still wish to evaluate the ability of our solution to scale to larger efficient HE networks as they are developed. To this end, we evaluate CHEX-MIX over a version of the SqueezeNet CNN used to evaluate the CHET homomorphic compiler [30] over the CIFAR-10 dataset. To the best of our knowledge, *this is the largest neural network evaluated using HE to date* and contains a total of 10 convolutional layers. For this experiment, we follow a similar strategy to Experiment 2 and randomize the values for all inputs inside the enclave prior to evaluation.

7.2 Experimental Setup

We run Experiments 1 and 2 on an Azure `Standard DC8ds_v3` VM with 64 GB of RAM with a maximum of 32 GB of Enclave Page Cache and Experiment 3 on an Azure `Standard DC48ds_v3` VM with 384 GB of RAM with a maximum of 256 GB of Enclave Page Cache. All our experiments are compiled with GNU CC (version 7) on Ubuntu 20.04 and executed with a single thread at 2.8 GHz.

In all experiments, we use the Microsoft SEAL library v3.7.1 [87] for the HE portion of our solution. For developing the server-side enclave, we use the Open-Enclave SDK v0.17.6 [76] in pre-release mode for Experiments 1 and 3 and the Intel SGX SDK v2.15 [46] in pre-release mode (with `SGX_MODE=HW`) for Experiment 2.

HE Parameters. Eff.-MKHE uses a degree of 16384 to obtain an optimal SIMD packing strategy for the CNN in Experiments 1 and 2. We use this as well for a fair comparison with their work. Since we do not need to perform as many

ciphertext-ciphertext operations as Eff.-MKHE (as we do not require weights in ciphertext form), we can use only 5 modulus primes of bit lengths {60, 57, 57, 57, 60} rather than the 8 primes used in Eff.-MKHE without any change to the underlying HE algorithm, SIMD packing strategy, or accuracy. We use encoding scales of $2^{53}, 2^{34}, 2^{27}$, 2^{30}, and 2^{20} for the input, layers 1-3 weights, and the masking plaintext in the final fully connected layer, respectively. Both our work and Eff.-MKHE achieve an accuracy of 97.95% over the MNIST test dataset, which is the same as the accuracy the model achieves for evaluation in-the-clear.

The authors of CHET instantiated SqueezeNet with parameters with $<$ 128-bits of security. We analyzed the computation and discovered that we could reduce parameter selection from a total modulus bit length of 940 to a bit length of 840 for a degree of 32768, which does provide 128-bit security. We use these improved parameters and scales of 2^{10} for weights, 2^{20} for masking plaintexts, and an initial input scale of 2^{53}. We evaluate this version of SqueezeNet over 100 random inputs in the CIFAR-10 test set and observe an accuracy of 77%, which is close to the 81.5% accuracy that the initial CHET work obtained over all test examples for less secure parameters. We emphasize that the purpose of Experiment 3 is to demonstrate scalability rather than efficiency or accuracy of HE, and recent work [64] has already demonstrated methods to significantly improve both the performance and accuracy of HE evaluation of this network using even smaller parameters.

Measurement Methodology. OpenEnclave and the Intel SGX SDK do not provide access to runtime counters (e.g., via the C++ `chrono` library) that developers typically use to measure code performance. Instead, we opt to call custom functions in the untrusted host through enclave OCALLs (a term used for function calls that call the untrusted host process from the enclave), and we use these host functions to implement the measurement checkpoints. Since our measurements include the runtime of executing these OCALL switches, our results may slightly overestimate the enclave code runtime.

Since communication latencies can vary widely between network infrastructures, user devices, and application types (e.g., high-performance use cases vs. IoT, cellular vs. ethernet, etc.), we do not include these in our overall runtime results.

SGX Memory Specification. Two versions of SGX have been released to date, which differ in their handling of memory allocation. The first, SGXv1, requires the user to specify the maximum enclave memory size required for the workload prior to enclave initialization, while SGXv2 allows the enclave to dynamically allocate memory as needed. For our experiments, we target the more widely available SGXv1. We configured Experiments 1 and 2 to use up to 128 KB of stack and up to 512 MB of heap enclave memory, and Experiment 3 to use up to 2 GB of stack and up to 310 GB of heap enclave memory.

We measured the total memory consumption of our modified SqueezeNet CNN benchmark and found that it consumes more than 300 GB. This is larger than the largest Enclave Page Cache offered by Azure VMs, and additionally causes the OS to terminate the enclave process. However, our solution can easily scale to larger network sizes by dividing the evaluation across multiple enclaves. We therefore evaluate this benchmark by splitting the computation roughly in half and evaluating each half separately, and we report the results for this experiment as the sum of these halves. We omit the communication cost between the enclaves since this cost can be hidden in parallel with the main evaluation.

7.3 Results

Table 3 shows the results of Experiment 1 for evaluating the baseline CHEX-MIX protocol over the Eff.-MKHE CNN benchmark, along with the performance costs of prior solutions for oblivious inference. In the next section, we compare this version of our solution to prior works in more detail.

For Experiment 2, we found that both the PCL and non-PCL versions of the benchmarks had the same runtime of 0.38 seconds (and we verified with the developers of the Intel SGX library that this is the expected behavior). Thus, CHEX-MIX can offer this protection for *no added performance cost*.

Finally, Table 5 shows the results of Experiment 3 for evaluating CHEX-MIX over our modified SqueezeNet benchmark. Compared to an HE-only execution of the same network, CHEX-MIX is only 2.28$\times$ slower. Since this is approximately the same slowdown we observed compared to HE-only for Experiment 1, these results suggest that CHEX-MIX maintains its scalability across a range of network sizes.

7.4 Comparison with Prior Work

We emphasize that it would be misleading to compare solely the performance of our solution with prior work since prior solutions do not address the same problem statement and/or assume a weaker threat model than ours. Nevertheless, we compare our baseline Experiment 1 proof of concept to prior approaches for oblivious inference for inference over the MNIST dataset for similarly-sized CNNs (see Appendix D for details of the CNNs) to provide some perspective of our results. We list a subset of the problem statement differences between works alongside these results to remind the reader of some of the key advantages of our approach over prior solutions, with more information given in Section 6 and below.

XONN. Like other 2PC solutions, XONN does not allow $\mathcal{M}$ to offload computation to the public cloud. Additionally, XONN assumes a much weaker passive threat model and therefore does not provide privacy guarantees in the presence of a malicious $\mathcal{A}_C$ or malicious or rational $\mathcal{A}_{\mathcal{M}}$. In spite of this, CHEX-MIX has a much smaller ($51\times$) communication cost and is only $3\times$ slower than XONN.

GAZELLE, DELPHI, and MUSE. Similar to XONN, these solutions do not allow $\mathcal{M}$ to offload computation to the cloud. Additionally, they only assume a semi-honest threat model

Table 3: Experiment 1 computation and communication cost of CHEX-MIX compared to prior works for two-party oblivious CNN inference over the MNIST dataset. CHEX-MIX achieves a competitive runtime and communication cost to prior approaches while providing security guarantees not provided by prior work. Measurements below the middle line are from this work.

Work	Technique	$\mathcal{M}$ can fully offload	Privacy ($\mathbf{X}$,$\mathbf{Y}$ / $\mathcal{A}_{\mathcal{M}}$)	Privacy ($\mathbf{W}$ / $\mathcal{A}_C$)	Correctness ($\mathbf{Y}$ / $\mathcal{A}_S$)	Runtime (s)	Comm. (MB)
XONN[3] [81]	2PC	✗	✗	✗	–	0.16	38.28
GAZELLE[1,2,3] [51]	2PC+HE	✗	✗	✗	–	0.15 + 0.05	5.9 + 2.1
MUSE[1,3,4] [57]	2PC+HE	✗	✗	✓	–	22.67 (1-6) + 0.80 (8)	4270.08 + 10.24
DELPHI[1,3,4] [69]	2PC+HE	✗	✗	✗	–	7.41 (1-8) + 0.48 (8)	235.52 + 10.24
DELPHI[1] [69]	2PC+HE	✗	✗	✗	–	1.20 + 0.14	43.65 + 0.15
Eff.-MKHE [19]	MKHE	✗	✓	✗	✗	1.23	2.125
CHEX-MIX	TEE	✓	✗	✓	✓	0.0002	0.00076
CHEX-MIX	HE	✗	✓	✓	–	0.19	0.75
CHEX-MIX	HE+TEE	✓	✓	✓	✓	0.46	0.75

[1] Costs separated into "offline" and "online" components (though both must be repeated per inference). Separation indicated by the (+) symbol.
[2] Work did not explicitly mention evaluation thread count.
[3] Runtime includes computation and LAN communication latency. Authors did not provide separate runtimes for computation only.
[4] Runtime is the sum of multiple parts, each with a different thread count. Thread count listed in parenthesis.

Table 4: Runtime and communication breakdown for DELPHI, Eff.-MKHE, HE-only, and CHEX-MIX for Experiment 1. HE-only and CHEX-MIX solutions enable lower client runtime and communication costs than DELPHI or Eff.-MKHE.

Work	Runtime (s)		Comm. (MB)	
	Client	Server	C–>S	S–>C
DELPHI [69]	0.20	1.20	41.7	2.06
Eff.-MKHE [19]	0.017	1.22	1.75	0.375
HE-only	0.007	0.20	0.5	0.25
CHEX-MIX		0.45		

Table 5: Runtime and communication breakdown for HE-only and CHEX-MIX baseline evaluation of the modified CHET-SqueezeNet CNN over the CIFAR-10 dataset.

Work	Runtime (s)		Comm. (MB)	
	Client	Server	C–>S	S–>C
HE-only	0.21	891	14.25	10
CHEX-MIX		2028		

and thus do not protect against active adversaries (with the exception of client-malicious protection offered by MUSE).

Compared to GAZELLE, CHEX-MIX is only slightly more than twice as slow but achieves a nearly $11\times$ smaller communication cost. Compared to DELPHI [69] and MUSE [57] with *multiple* threads evaluated over a CNN that achieves similar accuracy as our network, our solution with a *single* thread is $17\times$ and $51\times$ faster and has a $328\times$ and $5707\times$ smaller communication cost, respectively. We count the reported so-called "offline" and "online" costs for this comparison together since both of these phases need to repeated for *every* inference.

For an additional comparison point, we implement our target CNN benchmark (with square activations) using the open-source DELPHI code and measure its performance using a single thread on the same platform used to benchmark our first two experiments. We measure the runtime for computation only (omitting time spent on communication) and report this runtime along with the total communication size in Table 3 below the middle line. Even in this case, our solution is nearly $3\times$ faster and has a more than $58\times$ smaller communication cost per inference. Table 4 further shows that DELPHI requires a $29\times$ larger computation time and an $83\times$ larger client upload communication cost than CHEX-MIX, implying it would be more burdensome for constrained client devices.

Eff.-MKHE. To ensure as fair a comparison with Eff.-MKHE [19] as possible, we upgrade the original Eff.-MKHE implementation to use the latest version of Microsoft SEAL by integrating the same modifications the original work made to an earlier version of the Microsoft SEAL library into the newer library version. Compared to Eff.-MKHE, CHEX-MIX is nearly $3\times$ faster and achieves a nearly $3\times$ smaller communication cost. Further, Table 4 shows that Eff.-MKHE has a $2\times$ larger computation cost and a $3\times$ larger upload communication cost for the client compared to CHEX-MIX, making it less ideal for scenarios in which a client device is constrained.

Eff.-MKHE allows the offload of some computation to an untrusted cloud; however, it still requires $\mathcal{M}$ to maintain a private online server to perform partial decryption of results. Eff.-MKHE is also not secure against active adversaries, and it is only secure against a semi-honest server if the model provider performs the partial decryption with a secure method such as noise flooding. The implementation in [19] did not include such a secure method and is therefore vulnerable to a passive key-recovery attack on CKKS [58].

HE-only. The HE-only solution provides no privacy guarantees to $\mathcal{M}$ when $\mathcal{M}$ wants to offload computation to the public cloud. To achieve a comparable threat model to our solution, $\mathcal{M}$ would need to host a private online infrastructure for HE evaluation, losing the scalability and flexibility offered by the public cloud. We measured the runtimes of "HE" and "HE+TEE" in Table 3 with the same code, executed outside and inside an enclave, respectively.

TEE-only The comparison between a TEE-only solution and all prior approaches for oblivious inference is the most stark. However, we emphasize that a TEE-only solution does not provide privacy to clients (see Sections 4 and 6), and thus does not solve the problem of oblivious inference.

8 Discussion

8.1 Circuit Privacy

At a high level, a circuit-private protocol should ensure that clients learn no more about the computation than the outputs of their input queries. HE ciphertexts contain error terms that change during evaluation, potentially leaking information about the plaintext values applied to an encrypted input. XONN [81] noted that the implementation in GAZELLE [51] does not satisfy the circuit privacy property. Indeed, as a hybrid 2PC-HE protocol, GAZELLE requires client decryption of HE ciphertexts after *every* linear layer, making it conceivable that a client adversary could, in theory, derive model weights from the decrypted values.

By contrast, our protocol only involves client decryption after *all* network layers are evaluated. It is not known whether, after several layer computations, especially when including multiple non-linear layers, an attacker could still use the final ciphertext error to learn the weights used in intermediate layers. No prior work has sufficiently demonstrated how an adversary could extract plaintext input from noise in the resulting ciphertexts of a nonlinear circuit in practice. Nonetheless, a simple mitigation for this problem is to use an HE scheme like BFV or BGV with increased parameter sizes rather than CKKS [69, 81]. The defenses listed in Appendix C may also be helpful to defend against circuit privacy attacks.

8.2 Side channels

Although we do not consider side channels as part of our threat model (consistent with Intel's official threat model for SGX [49]), we nevertheless wish to devote some discussion to them given their attention in the literature.

Side-channel attacks on TEEs fall into two main categories: those that the software (enclave) developer can mitigate and those that they cannot. The first type, Intel contends, is the responsibility of the enclave developer to safeguard against [49]. For example, SGX requires developers to take steps to write their enclave code in a secret-data-independent manner if they want protection against timing side-channel adversaries.

The second type of side-channel attacks appears much more complicated to defend against; prior works detail side channel attacks that are either outside of the enclave code developer's immediate purview or are very difficult for the developer to properly prevent [61, 99, 106]. Intel purports to take these attacks seriously and continues to actively issue patches to SGX for numerous side channels as mitigations are developed [49, 92]. It is therefore critical that SGX users use proper enclave attestation to ensure they are using the most up-to-date version of the technology. Additionally, several works [8, 32, 75] propose techniques to mitigate the ability of attackers to perform side-channel attacks on TEEs, which providers can add on top of our solution for added protection.

A significant advantage of our solution is that it natively offers *clients* privacy protection from malicious side-channel adversaries. In particular, since client data is always in encrypted form inside the enclave, any attack on the enclave cannot view the underlying data of the HE ciphertexts. Clients can also use the strategy discussed in Appendix B for added assurance of computational correctness even against side channel attacks.

Additionally, we ensure that our implementation does not leak any information about $\mathcal{M}$'s private values $\mathbf{W}$ through timing side channels by verifying that our inference code and the implementation of operations in Microsoft SEAL do not contain any data-dependent computation, branching, or memory accesses based on $\mathcal{M}$'s private data. While this is possible for any HE inference solution, since HE-inference does not *require* any non-constant-time operations on $\mathcal{M}$'s private data, we note that this not a default property of an HE-TEE hybrid solution but rather results from a secure implementation of both enclave code and SGX technology.

9 Conclusion

In this work, we propose a novel approach for two-party privacy-preserving machine learning inference in the public cloud setting. Our solution, CHEX-MIX, features a hybrid HE-TEE strategy that provides both clients and model providers with confidentiality and integrity guarantees under a strong adversary model, tolerating malicious clients, malicious cloud providers, and rational, actively adversarial model providers. We demonstrate the feasibility of performing homomorphic evaluation inside TEEs by deploying CHEX-MIX on a Microsoft Azure confidential computing virtual machine. Our experiments demonstrate that CHEX-MIX is able to achieve runtime and communication costs comparable to or more efficient than prior approaches, while providing powerful security guarantees not addressed by prior work. We hope that this work will enable more scenarios for HE deployments and that this will in turn encourage the development of more openly-available, hand-optimized HE benchmarks in the future.

References

[1] "Intel® Software Guard Extensions (Intel® SGX)," Jun 2015, reference no. 332680-001. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/332680-001-720907.pdf

[2] "Cheetah: Lean and fast secure Two-Party deep neural network inference," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong

[3] "Machine learning (ml) market size, share amp; covid-19 impact analysis, by component (solution, and services), by enterprise size (smes, and large enterprises), by deployment (cloud and on-premise), by end-user (healthcare, retail, it and telecommunication, bfsi, automotive and transportation, advertising and media, manufacturing, and others), and regional forecast, 2022-2029," Apr 2022. [Online]. Available: https://www.fortunebusinessinsights.com/machine-learning-market-102226

[4] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern, "Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation," in *25th ACM PODC*, E. Ruppert and D. Malkhi, Eds. ACM, Jul. 2006, pp. 53–62.

[5] "Arm mbed tls," https://github.com/Mbed-TLS/mbedtls, Dec. 2021.

[6] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *Int. J. Secur. Netw.*, vol. 10, no. 3, p. 137–150, Sep. 2015. [Online]. Available: https://doi.org/10.1504/IJSN.2015.071829

[7] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," in *TCC 2007*, ser. LNCS, S. P. Vadhan, Ed., vol. 4392. Springer, Heidelberg, Feb. 2007, pp. 137–156.

[8] S. Banerjee, P. Ramrakhyani, S. Wei, and M. Tiwari, "SESAME: software defined enclaves to secure inference accelerators with multi-tenant execution," *CoRR*, vol. abs/2007.06751, 2020. [Online]. Available: https://arxiv.org/abs/2007.06751

[9] T. Bao, J. Burket, M. Woo, R. Turner, and D. Brumley, "{BYTEWEIGHT}: Learning to recognize functions in binary code," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 845–860.

[10] R. Bellan, "Waymo to keep robotaxi safety details secret, court rules," Feb 2022. [Online]. Available: https://techcrunch.com/2022/02/22/waymo-to-keep-robotaxi-safety-details-secret-court-rules/

[11] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "Ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM Workshop on Encrypted Computing Applied Homomorphic Cryptography*, ser. WAHC'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 45–56. [Online]. Available: https://doi-org.proxy.lib.umich.edu/10.1145/3338469.3358944

[12] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "Ngraph-he: A graph compiler for deep learning on homomorphically encrypted data," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, ser. CF '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 3–13. [Online]. Available: https://doi-org.proxy.lib.umich.edu/10.1145/3310273.3323047

[13] F. Boenisch, A. Dziedzic, R. Schuster, A. S. Shamsabadi, I. Shumailov, and N. Papernot, "When the curious abandon honesty: Federated learning is not private," *arXiv preprint arXiv:2112.02918*, 2021.

[14] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *CRYPTO 2012*, ser. LNCS, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, Heidelberg, Aug. 2012, pp. 868–886.

[15] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012*, S. Goldwasser, Ed. ACM, Jan. 2012, pp. 309–325.

[16] D. Brumley, J. Lee, E. J. Schwartz, and M. Woo, "Native x86 decompilation using {Semantics-Preserving} structural analysis and iterative {Control-Flow} structuring," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 353–368.

[17] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 812–821. [Online]. Available: http://proceedings.mlr.press/v97/brutzkus19a.html

[18] M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Security of homomorphic encryption," HomomorphicEncryption.org, Redmond WA, USA, Tech. Rep., July 2017.

[19] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference,"

in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM Press, Nov. 2019, pp. 395–412.

[20] L. Chen, Z. Zhang, and X. Wang, "Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension," in *TCC 2017, Part II*, ser. LNCS, Y. Kalai and L. Reyzin, Eds., vol. 10678. Springer, Heidelberg, Nov. 2017, pp. 597–627.

[21] J. H. Cheon, S. Hong, and D. Kim, "Remark on the security of ckks scheme in practice," *Cryptology ePrint Archive*, 2020.

[22] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT 2017, Part I*, ser. LNCS, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, Heidelberg, Dec. 2017, pp. 409–437.

[23] J. H. Cheon, D. Kim, and D. Kim, "Efficient homomorphic comparison methods with optimal complexity," in *ASIACRYPT 2020, Part II*, ser. LNCS, S. Moriai and H. Wang, Eds., vol. 12492. Springer, Heidelberg, Dec. 2020, pp. 221–256.

[24] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, Jan. 2020.

[25] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," *CoRR*, vol. abs/1811.09953, 2018. [Online]. Available: http://arxiv.org/abs/1811.09953

[26] Z. M. Çınar, A. Abdussalam Nuhu, Q. Zeeshan, O. Korhan, M. Asmael, and B. Safaei, "Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0," *Sustainability*, vol. 12, no. 19, p. 8211, 2020.

[27] L. Coppolino, S. D'Antonio, V. Formicola, G. Mazzeo, and L. Romano, "VISE: Combining Intel SGX and homomorphic encryption for cloud industrial control systems," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 711–724, 2021.

[28] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Report 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[29] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, "Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 546–561. [Online]. Available: https://doi.org/10.1145/3385412.3386023

[30] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "Chet: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 142–156. [Online]. Available: https://doi-org.proxy.lib.umich.edu/10.1145/3314221.3314628

[31] J. De Spiegeleer, D. B. Madan, S. Reyners, and W. Schoutens, "Machine learning for quantitative finance: fast derivative pricing, hedging and fitting," *Quantitative Finance*, vol. 18, no. 10, pp. 1635–1643, 2018.

[32] G. Dessouky, T. Frassetto, and A.-R. Sadeghi, "HybCache: Hybrid side-channel-resilient caches for trusted execution environments," in *USENIX Security 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, Aug. 2020, pp. 451–468.

[33] N. Drucker and S. Gueron, "Combining homomorphic encryption with trusted execution environment: A demonstration with Paillier encryption and SGX," in *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, ser. MIST '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 85–88. [Online]. Available: https://doi.org/10.1145/3139923.3139933

[34] ——, "Achieving trustworthy homomorphic encryption by combining it with a trusted execution environment," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 9, pp. 86–, 03 2018.

[35] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Report 2012/144, 2012, https://eprint.iacr.org/2012/144.

[36] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, "Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences," *Cryptology ePrint Archive*, 2021.

[37] A. Fischer, B. Fuhry, F. Kerschbaum, and E. Bodden, "Computation on encrypted data using dataflow authentication," *PoPETs*, vol. 2020, no. 1, pp. 5–25, Jan. 2020.

[38] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *ACM CCS 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM Press, Oct. 2015, pp. 1322–1333.

[39] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 201–210. [Online]. Available: http://proceedings.mlr.press/v48/gilad-bachrach16.html

[40] S. Halevi and V. Shoup, "Faster homomorphic linear transformations in HElib," in *CRYPTO 2018, Part I*, ser. LNCS, H. Shacham and A. Boldyreva, Eds., vol. 10991. Springer, Heidelberg, Aug. 2018, pp. 93–120.

[41] J. Y. Halpern and V. Teague, "Rational secret sharing and multiparty computation: Extended abstract," in *36th ACM STOC*, L. Babai, Ed. ACM Press, Jun. 2004, pp. 623–632.

[42] A. Holdings, "ARM security technology: Building a secure system using trustzone technology," https://developer.arm.com/documentation/genc009492/latest, 2009.

[43] T. Hunter, "Instagram is touting safety features for teens. mental health advocates aren't buying it." Dec 2021. [Online]. Available: https://www.washingtonpost.com/technology/2021/12/07/instagram-teen-health/

[44] S. U. Hussain, M. Javaheripi, M. Samragh, and F. Koushanfar, "Coinn: Crypto/ml codesign for oblivious inference via neural networks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3266–3281. [Online]. Available: https://doi.org/10.1145/3460120.3484797

[45] "Intel® Software Guard Extensions (Intel® SGX) Protected Code Loader (PCL)," https://github.com/intel/linux-sgx-pcl, May 2018.

[46] "Intel® Software Guard Extensions (Intel® SGX) SDK for Linux* OS (version 2.15)," https://github.com/intel/linux-sgx, Sep. 2021.

[47] "Intel® Software Guard Extensions (Intel® SGX) SDK for Linux* OS," https://download.01.org/intel-sgx/latest/linux-latest/docs/Intel_SGX_Developer_Reference_Linux_2.14_Open_Source.pdf, July 2021.

[48] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM Press, Oct. 2018, pp. 1209–1222.

[49] S. P. Johnson, "Intel® SGX and side-channels," https://software.intel.com/content/www/us/en/develop/articles/intel-sgx-and-side-channels.html, March 2017.

[50] M. Juuti, S. Szyller, A. Dmitrenko, S. Marchal, and N. Asokan, "Prada: Protecting against dnn model stealing attacks," *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 512–527, 2019.

[51] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *USENIX Security 2018*, W. Enck and A. P. Felt, Eds. USENIX Association, Aug. 2018, pp. 1651–1669.

[52] S. K. Katyal, "The paradox of source code secrecy," *Cornell L. Rev.*, vol. 104, p. 1183, 2018.

[53] M. Kim, Y. Song, B. Li, and D. Micciancio, "Semi-parallel logistic regression for GWAS on encrypted data," *BMC Medical Genomics*, vol. 13, no. 7, pp. 1–13, 2020.

[54] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij, "Integrating remote attestation with transport layer security." *arXiv preprint arXiv:1801.05863*, 2018.

[55] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure TensorFlow inference," in *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020, pp. 336–353.

[56] J. Lee, T. Avgerinos, and D. Brumley, "Tie: Principled reverse engineering of types in binary programs," in *NDSS*, 2011.

[57] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "MUSE: Secure inference resilient to malicious clients," in *USENIX Security 2021*. USENIX Association, Aug. 2021.

[58] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *Advances in Cryptology – EUROCRYPT 2021*, A. Canteaut and F.-X. Standaert, Eds. Cham: Springer International Publishing, 2021, pp. 648–677.

[59] N. Li, T. Zhou, X. Yang, Y. Han, W. Liu, and G. Tu, "Efficient multi-key FHE with short extended ciphertexts and directed decryption protocol," *IEEE Access*, vol. 7, pp. 56 724–56 732, 2019.

[60] Y. Lindell, "Secure multiparty computation for privacy preserving data mining," in *Encyclopedia of Data Warehousing and Mining*. IGI global, 2005, pp. 1005–1009.

[61] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "PLATYPUS: Software-based

power side-channel attacks on x86," in *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2021.

[62] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 619–631.

[63] T. Liu, X. Xie, and Y. Zhang, "Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2968–2985. [Online]. Available: https://doi.org/10.1145/3460120.3485379

[64] Q. Lou and L. Jiang, "Hemet: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7102–7110.

[65] A. Lysyanskaya and N. Triandopoulos, "Rationality and adversarial behavior in multi-party computation (extended abstract)," in *CRYPTO 2006*, ser. LNCS, C. Dwork, Ed., vol. 4117. Springer, Heidelberg, Aug. 2006, pp. 180–197.

[66] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." *Hasp@ isca*, vol. 10, no. 1, 2013.

[67] G. McLean and K. Osei-Frimpong, "Hey alexa... examine the variables influencing the use of artificial intelligent in-home voice assistants," *Computers in Human Behavior*, vol. 99, pp. 28–37, 2019.

[68] W. Michiels, "How do you protect your machine learning investment? part 1: Intellectual property aspects of machine learning," Mar 2020. [Online]. Available: https://www.eetimes.com/how-do-you-protect-your-machine-learning-investment/

[69] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *USENIX Security 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, Aug. 2020, pp. 2505–2522.

[70] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM Press, Oct. 2018, pp. 35–52.

[71] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 19–38.

[72] M. A. Myszczynska, P. N. Ojamies, A. Lacoste, D. Neil, A. Saffari, R. Mead, G. M. Hautbergue, J. D. Holbrook, and L. Ferraiuolo, "Applications of machine learning to diagnosis and treatment of neurodegenerative diseases," *Nature Reviews Neurology*, vol. 16, no. 8, pp. 440–456, 2020.

[73] L. K. Ng and S. S. Chow, "{GForce}:{GPU-Friendly} oblivious and rapid neural network inference," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2147–2164.

[74] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *USENIX Security 2016*, T. Holz and S. Savage, Eds. USENIX Association, Aug. 2016, pp. 619–636.

[75] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer, "Varys: Protecting SGX enclaves from practical side-channel attacks," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 227–240. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/oleksenko

[76] "Open Enclave SDK (version 0.15.0)," https://github.com/openenclave/openenclave, Apr. 2021.

[77] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved mixed-protocol secure two-party computation," in *USENIX Security 2021*. USENIX Association, Aug. 2021.

[78] E. Porter, "Your data is crucial to a robotic age. shouldn't you be paid for it?" Mar 2018. [Online]. Available: https://www.nytimes.com/2018/03/06/business/economy/user-data-pay.html

[79] C. Priebe, K. Vaswani, and M. Costa, "EnclaveDB: A secure database using SGX," in *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 264–278.

[80] D. L. Quoc, F. Gregor, S. Arnautov, R. Kunkel, P. Bhatotia, and C. Fetzer, "SecureTF: A secure TensorFlow framework," in *Proceedings of the 21st International Middleware Conference*, ser. Middleware '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 44–59. [Online]. Available: https://doi.org/10.1145/3423211.3425687

[81] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *USENIX Security 2019*, N. Heninger and P. Traynor, Eds. USENIX Association, Aug. 2019, pp. 1501–1518.

[82] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *ASIACCS 18*, J. Kim, G.-J. Ahn, S. Kim, Y. Kim, J. López, and T. Kim, Eds. ACM Press, Apr. 2018, pp. 707–721.

[83] J. G. Richens, C. M. Lee, and S. Johri, "Improving the accuracy of medical diagnosis with causal machine learning," *Nature communications*, vol. 11, no. 1, pp. 1–9, 2020.

[84] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3196023

[85] G. Schryen and R. Kadura, "Open source vs. closed source software: towards measuring security," in *Proceedings of the 2009 ACM symposium on Applied Computing*, 2009, pp. 2016–2023.

[86] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: Trustworthy data analytics in the cloud using SGX," in *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 38–54.

[87] "Microsoft SEAL (release 3.7)," https://github.com/Microsoft/SEAL, Oct. 2021, microsoft Research, Redmond, WA.

[88] A. Sharma, A. Jain, P. Gupta, and V. Chowdary, "Machine learning applications for precision agriculture: A comprehensive review," *IEEE Access*, vol. 9, pp. 4843–4873, 2020.

[89] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 3–18.

[90] I. S. Singh, "Safe and secure outsourced computing with fully homomorphic encryption and trusted execution environments," Master's thesis, UiT Norges arktiske universitet, 2020.

[91] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Designs, codes and cryptography*, vol. 71, no. 1, pp. 57–81, 2014.

[92] R. Stubbs, "Intel® SGX technology and the impact of processor side-channel attacks," https://fortanix.com/blog/2020/03/intel-sgx-technology-and-the-impact-of-processor-side-channel-attacks, Mar 2020.

[93] P. P. Swire, "A theory of disclosure for security and competitive reasons: Open source, proprietary software, and government systems," *Hous. L. Rev.*, vol. 42, p. 1333, 2005.

[94] "TFHE: Fast fully homomorphic encryption library," https://github.com/tfhe/tfhe, Aug. 2016.

[95] S. Tople, K. Grover, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and secure DNN inference," *CoRR*, vol. abs/1810.00602, 2018. [Online]. Available: http://arxiv.org/abs/1810.00602

[96] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJVorjCcKQ

[97] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *USENIX Security 2016*, T. Holz and S. Savage, Eds. USENIX Association, Aug. 2016, pp. 601–618.

[98] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei, "Demystifying membership inference attacks in machine learning as a service," *IEEE Transactions on Services Computing*, pp. 1–1, 2019.

[99] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *USENIX Security 2018*, W. Enck and A. P. Felt, Eds. USENIX Association, Aug. 2018, pp. 991–1008.

[100] D. Vinayagamurthy, A. Gribov, and S. Gorbunov, "StealthDB: a scalable encrypted database with full SQL query support," *PoPETs*, vol. 2019, no. 3, pp. 370–388, Jul. 2019.

[101] A. Virtualization, "Secure virtual machine architecture reference manual," *AMD Publication*, vol. 33047, 2005.

[102] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *PoPETs*, vol. 2019, no. 3, pp. 26–49, Jul. 2019.

[103] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *PoPETs*, vol. 2021, no. 1, pp. 188–208, Jan. 2021.

[104] J. R. Wallrabenstein, "Rational multiparty computation," https://docs.lib.purdue.edu/dissertations/AAI3702881, 2014.

[105] S. Wang, P. Wang, and D. Wu, "Reassembleable disassembling," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 627–642.

[106] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 2421–2434.

[107] W. Wang, Y. Jiang, Q. Shen, W. Huang, H. Chen, S. Wang, X. Wang, H. Tang, K. Chen, K. E. Lauter, and D. Lin, "Toward scalable fully homomorphic encryption through light trusted computing assistance," *CoRR*, vol. abs/1905.07766, 2019. [Online]. Available: http://arxiv.org/abs/1905.07766

[108] C. Weng, K. Yang, X. Xie, J. Katz, and X. S. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 730, 2021.

[109] G. Xu, H. Li, H. Ren, J. Sun, S. Xu, J. Ning, H. Yang, K. Yang, and R. H. Deng, "Secure and verifiable inference in deep neural networks," in *Annual Computer Security Applications Conference*, ser. ACSAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 784–797. [Online]. Available: https://doi.org/10.1145/3427228.3427232

[110] S. Zoder, "How much is your data worth?" Aug 2019. [Online]. Available: https://www.forbes.com/sites/stephanzoder/2019/08/06/how-much-is-your-data-worth/?sh=4be3f4ed70fc

A Client Attestation

If $\mathcal{M}$ were malicious instead of rational, $\mathcal{C}$ could not assume $\mathcal{M}$ would provide a correct implementation of service P, nor could $\mathcal{C}$ be sure that $\mathcal{M}$ would not insert a backdoor into $\mathbf{I_F}$ for exploitation by $\mathcal{A}_\mathcal{M}$. At first glance, it may seem that this problem can be solved by having $\mathcal{C}$ attest enclave $\mathcal{E}_\mathcal{S}$. Indeed, assuming $\mathcal{C}$ knew the underlying $\mathbf{F}$ that would provide service P, $\mathcal{C}$ could, in theory, verify for themselves that $\mathbf{I_F}$ implements $\mathbf{F}$. However, in practice, this places a high burden on $\mathcal{C}$, and $\mathcal{C}$ would still need to verify that $\mathbf{I_F}$ is free from all vulnerabilities that could affect computational correctness, including any hidden backdoors inserted by a malicious $\mathcal{M}$ (see Sections 4 and 6).

Instead, $\mathcal{C}$ could again apply the rational assumption to guarantee that $\mathcal{M}$ would not insert any backdoors into $\mathbf{I_F}$. This reduces the functionality dependent on the rational behavior of $\mathcal{M}$ and allows $\mathcal{C}$ more control over verifying correct enclave execution. Figures 4 and 5 give the modifications required to the CHEX-MIX protocol to provide these additional properties.

Setup ($\mathcal{M}$) (One time)

1. $\mathcal{M}$ verifies that code $\mathbf{I_F}$ securely implements $\mathbf{F}$.
2. $\mathcal{M}$ uploads $\mathbf{I_F}$ to some public domain accessible by $\mathcal{C}$.
3. $\mathcal{M}$ establishes an enclave $\mathcal{E}_\mathcal{S}$ with code $\mathbf{I_F}$ on a server hosted by $\mathcal{S}$.
4. $\mathcal{M}$ generates a key pair for TLS channel establishment
5. $\mathcal{M}$ obtains a certificate $\mathbf{Cert}_\mathcal{M}$ for its public key from a CA trusted by both $\mathcal{M}$ and $\mathcal{C}$.
6. $\mathcal{M}$ attests and establishes a secure channel $\sigma_\mathcal{M}$ with $\mathcal{E}_\mathcal{S}$ for all future communication.
7. $\mathcal{M}$ sends $\mathbf{W}$ to $\mathcal{E}_\mathcal{S}$ over $\sigma_\mathcal{M}$.
8. $\mathcal{M}$ issues a certificate $\mathbf{Cert}_{\mathcal{E}_\mathcal{S}}$ for $\mathcal{E}_\mathcal{S}$'s public key and sends $(\mathbf{I_F}, \mathbf{W}, \mathbf{Cert}_\mathcal{M}, \mathbf{Cert}_{\mathcal{E}_\mathcal{S}}, \pi)$ to $\mathcal{E}_\mathcal{S}$ over $\sigma_\mathcal{M}$.
9. $\mathcal{M}$ exits the protocol and goes offline.

Figure 4: CHEX-MIX enhanced correctness verification setup protocol for $\mathcal{M}$

B Sensitive Samples

A client may want additional guarantees of computational integrity from adversaries outside the scope of our threat model (e.g., side-channel adversaries). In such cases, the following integrity checking mechanism may be helpful and can be used with our solution: the client $\mathcal{C}$ chooses a value $\mathbf{X}$ to encrypt for which $\mathcal{C}$ knows the expected output $\mathbf{Y}$. (In the case where $\mathcal{C}$ is not able to know the value of $\mathbf{Y}$ ahead of time for even a single $\mathbf{X}$, this pair can be shared directly by $\mathcal{M}$ if $\mathcal{M}$ is trusted to be rational.) $\mathcal{C}$ encrypts this value $\mathbf{X}$ and sends it to $\mathcal{E}_\mathcal{S}$ for evaluation. If the decryption of $\mathbf{Y}$ matches what $\mathcal{C}$ expects, $\mathcal{C}$ can be more confident that the data path is free of the types of integrity violations that would cause the result to be incorrect.

The above method was proposed by Xu et al. [109] for providing users with integrity assurance for outsourced HE

Setup ($\mathcal{C}$) (Once per client)

1. $\mathcal{C}$ verifies that $\mathbf{I_F}$ implements the expected functionality.
2. $\mathcal{C}$ receives $(\mathbf{Cert}_\mathcal{M}, \mathbf{Cert}_{\mathcal{E}_\mathcal{S}}, \pi)$ from $\mathcal{E}_\mathcal{S}$.
3. $\mathcal{C}$ verifies $(\mathbf{Cert}_\mathcal{M}, \mathbf{Cert}_{\mathcal{E}_\mathcal{S}}, \pi)$, attests $\mathcal{E}_\mathcal{S}$, and establishes a secure channel $\sigma_\mathcal{C}$ with $\mathcal{E}_\mathcal{S}$ upon successful verification.
4. $\mathcal{C}$ generates HE keys $\mathbf{SK}$ and $\mathbf{EK}$ based on π.
5. $\mathcal{C}$ sends $\mathbf{EK}$ to $\mathcal{E}_\mathcal{S}$ over channel $\sigma_\mathcal{C}$.

Figure 5: CHEX-MIX enhanced correctness verification setup protocol for $\mathcal{C}$

computation. The authors further discuss how a set of "sensitive samples" – input-output pairs that would detect integrity violations of concern with high probability – could be used to make this technique more robust. We note that this integrity checking property is unique to techniques such as HE that maintain the encrypted form of client inputs throughout the computation, preventing adversaries from simply identifying when inputs are part of the sample set and changing their behavior (e.g., malicious to honest) to evade detection.

C Additional Defenses for $\mathcal{M}$

Recent works have demonstrated how a client adversary can extract the private weights of a model provider through *model-stealing* attacks [97], or learn information about the initial model training data set through *model-inversion* or *re-identification*/*membership inference* attacks [89, 98] by analyzing the results of several queries made to the inference server. We note that these attacks are possible against all prior works in oblivious inference, and thus they are not uniquely applicable to our work. Nevertheless, we follow the approach taken by XONN [81] and discuss how providers can apply defense mechanisms for these attacks on top of our work.

A simple mitigation, suggested in prior work [50,62] involves having the server rate-limit the prediction requests from any given $\mathcal{C}$. We note that this approach requires the server to keep track of the identity of each client, or at least be able to differentiate one client from another. However, it may still be possible for any particular client to masquerade as or collude with a separate client to collect additional query responses.

A related but more difficult approach is to use statistical properties of the network to guarantee that results do not leak information. Here, a model provider can analyze a standalone network to ensure that the required number of queries to reverse engineer the model parameters is larger than is computationally feasible for clients to analyze.

Additionally, since the aforementioned attacks rely on having the server reveal to the client the confidence scores of the result, another mitigation suggested by prior works [38, 89] involves having the model provider apply a rounding filter layer to the result before sharing the result with the client. This technique ensures that, while the maximum predicted class remains the same, the result does not leak additional information about the weights through the precise confidence score values. We note that, unlike BFV or BGV, CKKS is particularly adept at removing the LSB values of a result. Thus, it is easy to add a rounding filter layer to our CKKS-based approach.

D Neural Network Descriptions

We provide a description of the CNNs used in our work and prior works to implement the MNIST inference network in Tables 6, 7, 8, and 9.

Table 6: Description of the CNN used for Experiments 1 and 2 for our solution and Eff.-MKHE based on the description given in Eff.-MKHE [19].

Layer	Description
Convolution	28x28-pixel images, 4×4 windows, (2,2) strides, 5 output channels
Square-1	Squares each of the 845 inputs
FC-1	Fully connects 845 inputs to 64 outputs
Square-2	Squares each of the 64 inputs
FC-2	Fully connects 64 inputs to 10 outputs

Table 7: Description of the CNN used to benchmark GAZELLE [51] in the original work, as described in DeepSecure [84]. Measurements reported in GAZELLE for this network were measured on an AWS `c4.xlarge` instance with 7.5 GB of RAM at 2.90 GHz.

Layer	Description
Convolution	28x28-pixel images, 5×5 windows, (2,2) strides, 5 output channels
ReLU-1	Applies ReLU activation to each of the 845 inputs
FC-1	Fully connects 845 inputs to 100 outputs
ReLU-2	Applies ReLU activation to each of the 100 inputs
FC-2	Fully connects 100 inputs to 10 outputs

E PCL-encrypted ELF File Output

As an example of the protection provided by the Intel PCL library, Figures 6a and 6b show the difference in the output of the `readelf` command on the enclave binaries, with and without using the Intel PCL library to encrypt the binaries, respectively.

Table 8: Description of the CNN used to benchmark DELPHI [69] and MUSE [57] (based on the description given in MiniONN [62].) in the MUSE work [57]. Adjacent linear layers are listed together since they are typically combined for homomorphic inference. Measurements reported in MUSE for this network were measured on an AWS `c5.9xlarge` instance with 72 GB of RAM on an Intel Xeon 8000 series CPU at 3.6 GHz.

Layer	Description
Conv-1	28x28-pixel images, 5×5 windows, (2,2) strides, 16 output channels
Act-1	Applies a truncated ReLU activation to each of the 9216 inputs
Pool-1, Conv-2	Average Pooling, 2x2 windows, 2304 outputs; 5x5 windows, (1,1) strides, 16 output channels
Act-2	Applies a truncated ReLU activation to each of the 1024 inputs
Pool-2, FC-1	Average Pooling; 2x2 windows, 256 outputs; Fully connects 256 inputs to 100 outputs
Act-3	Applies a truncated ReLU activation to each of the 100 inputs
FC-3	Fully connects 100 inputs to 10 outputs

Table 9: Description of CNN used to benchmark XONN [81] in the original work. Measurements reported in XONN for this network were measured on an Intel Core i7-7700k at 4.20 GHz with 32 GB of RAM and achieved an accuracy of 98.4%.

Layer	Description
Convolution	28x28-pixel images, 5×5 windows, (2,2) strides, 5 output channels
BN-BA-1	Binary normalization and binary activation to each of the 845 inputs
FC-1	Fully connects 845 inputs to 100 outputs
BN-BA-2	Binary normalization and binary activation to each of the 100 inputs
FC-2	Fully connects 100 inputs to 10 outputs

```
33: 0000000000005d80    54 FUNC    LOCAL  DEFAULT   12 sgx_ecall_type_int
34: 0000000000005dc0    70 FUNC    LOCAL  DEFAULT   12 sgx_ecall_type_char
35: 0000000000005e10    54 FUNC    LOCAL  DEFAULT   12 sgx_start_nn_app
36: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS rns.cpp
37: 0000000000003020   140 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util7PointerINS_
38: 00000000000030ac   140 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util7PointerINS0
39: 0000000000003138   140 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util7PointerImvE
40: 00000000000075bb8    11 OBJECT  LOCAL  DEFAULT   14 _ZZNKSt3__16vectorIN4seal
41: 00000000000031c4    77 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util7PointerINS0
42: 0000000000003212    77 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util7PointerINS0
43: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS context.cpp
44: 0000000000003260   140 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util7PointerImvE
45: 0000000000076288    11 OBJECT  LOCAL  DEFAULT   14 _ZZNKSt3__16vectorIN4seal
46: 0000000000076278     9 OBJECT  LOCAL  DEFAULT   14 _ZZNSt3__16vectorIN4seal7
47: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS evaluator.cpp
48: 00000000000032ec    58 FUNC    LOCAL  DEFAULT   12 _ZN4seal12_GLOBAL__N_122i
49: 0000000000003326    78 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util8mul_safeImJ
50: 0000000000003326    78 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util8mul_safeImJ
51: 0000000000003374   106 FUNC    LOCAL  DEFAULT   12 _ZNK4seal4util10GaloisToo
52: 00000000000033de   124 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util15seal_for_e
53: 000000000000345a    75 FUNC    LOCAL  DEFAULT   12 _ZZN4seal4util34inverse_n
54: 00000000000034a5   183 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util20negate_pol
55: 000000000000355c   133 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util17add_poly_c
56: 00000000000035e1   133 FUNC    LOCAL  DEFAULT   12 _ZN4seal4util23dyadic_pro
57: 0000000000003666   211 FUNC    LOCAL  DEFAULT   12 _ZZNK4seal9Evaluator10bfv
```

(a)

```
33: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
34: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
35: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
36: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
37: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
38: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
39: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
40: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
41: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
42: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
43: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
44: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
45: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
46: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
47: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
48: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
49: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
50: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
51: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
52: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
53: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
54: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
55: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
56: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
57: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
```

(b)

Figure 6: Excerpt from symbol table output of running the `readelf` command on unencrypted (top) and PCL-encrypted (bottom) enclave binaries. The unencrypted symbol table reveals function calls and file names, while the PCL-encrypted symbol table does not.