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

Deepika Natarajan
University of Michigan, Ann Arbor

Wei Dai
Microsoft Research

Ronald Dreslinski
University of Michigan, Ann Arbor

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) and leverages the benefits of each to counter the drawbacks of the other. In particular, we use 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. In this paper, we analyze our solution from a security perspective and detail the advantages that our solution provides over prior works, including its ability to allow model providers to maintain privacy of their software IP. We demonstrate the feasibility of our solution by deploying Chex-Mix in an Azure confidential computing machine. Our results show that Chex-Mix can execute at high efficiency, with low communication cost, while providing security guarantees unaddressed by prior work.

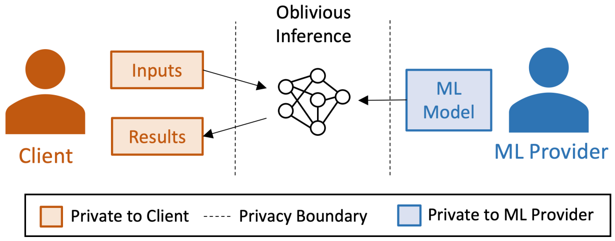


Figure 1: Two-party oblivious inference problem. Solutions should preserve both client and ML model provider privacy.

1 Introduction

The rise of machine learning (ML) has enabled a host of improvements in nearly all aspects of life, from medical diagnosis, to finance, personal assistants, and more. Alongside the rise of cloud computing, these technologies have had the potential to reach users across the globe at increasingly large scales. However, deployments of machine learning inference services often come at a high cost to user privacy since users must provide their personal data to model providers in order to obtain inference results. This threat is further exacerbated by the prevalence of third-party attackers, who target vulnerabilities in the infrastructure of large server deployments to obtain direct access to user data.

As a simple solution to this problem, ML model providers can choose to deploy their models close to users, such as directly on client devices. However, companies often spend significant time and resources developing and training models for their inference services. Highly tuned models are often thus considered vital pieces of intellectual property, which companies are generally unwilling to share. Moreover, sharing the details of a model can increase the ability of attackers to perform re-identification or membership inference attacks [3,71] that violate the privacy of user data in the model training set. The problem we are faced with then is this: *how can we* *provide privacy to both clients and model providers in a cloud setting?*

This problem, known more generally as *oblivious inference* in the literature [12, 58] and shown in Figure 1, has been the subject of several prior works, with solutions typically consisting of multi-party computation (MPC), homomorphic encryption (HE), or hybrid HE-MPC techniques. However, these techniques ultimately fall short of enabling scalable private machine learning since they still require ML model providers to maintain a private online infrastructure, effectively preventing model providers from taking full advantage of the public cloud.

More recently, trusted execution environments (TEEs) such as Intel Software Guard Extensions (SGX) have been proposed as a solution for secure cloud computing. TEEs allow ML model providers to deploy services in the cloud without needing to trust the host operating system of the cloud server. This property allows both ML model providers and their users to achieve security guarantees against third-party attackers. However, these environments are either not able to provide users with protection from the model providers themselves or require clients to possess an unrealistic level of security expertise in order to thoroughly verify the trustworthiness of TEE code.

1.0.1 Our solution

We present a novel hybrid solution to the problem of two-party oblivious inference in the cloud. Our solution, shown in Figure 2, relies on a combination of HE and TEEs and works as follows: An ML model provider establishes a TEE in the cloud and securely transfers the sensitive parameters of its model (e.g., its weights and bias values) to the TEE. The ML model provider then goes offline, and a client who wishes to use the established ML service homomorphically encrypts and sends their inputs to the TEE. The TEE performs a homomorphic evaluation over the client's inputs and returns the still HE-encrypted results to the client, who can then decrypt and obtain the final results. The privacy of the client's inputs and results 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. In this work, we make the following contributions:

- We present the CHEX-MIX protocol for two-party oblivious inference based on a combination of homomorphic encryption and trusted execution environments. Compared to prior works, our solution better enables ML model providers to utilize the scale of the public cloud, freeing model providers from needing to maintain a private online infrastructure and freeing clients from needing to possess expert-level knowledge to verify the security of TEE-code. Our solution provides privacy and correctness guarantees to both clients and model providers under a strong adversary model, tolerating malicious clients, malicious cloud adversaries, and rational, actively adversarial ML model providers.

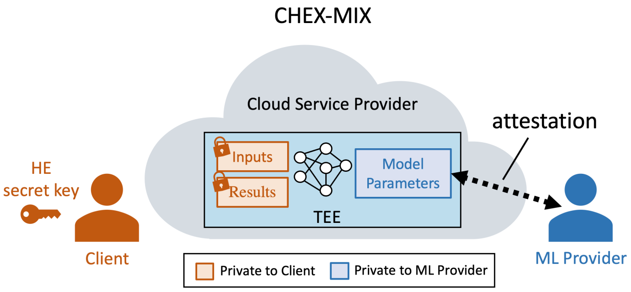


Figure 2: Overview of CHEX-MIX. The ML model provider attests and sends the model parameters to the cloud TEE. The client homomorphically encrypts and sends their inputs to the TEE, which homomorphically computes the inference results.

- We demonstrate the CHEX-MIX baseline protocol targeting a five-layer CNN for inference over the MNIST dataset. Our results show that CHEX-MIX is orders of magnitude more efficient than prior solutions for two-party oblivious inference over the target workload and provides security guarantees not addressed by prior works.
- We further adapt the CHEX-MIX baseline protocol to provide the model provider with confidentiality of their inference code. This version of our protocol may be most interesting for real-world deployments since it allows ML model providers to maintain privacy of their software IP while still providing both client and model provider parties with privacy and correctness guarantees in an untrusted cloud.

2 Background

2.1 Trusted Execution Environments

In a cloud environment, where computing resources are naturally shared and computing stacks are large and difficult to verify, malicious actors have abundant opportunities to access sensitive data. In particular, attackers are often able to co-opt the higher privilege level of OS/hypervisor layers by compromising vulnerabilities in bloated software stacks, thereby gaining direct access to user data. TEEs help solve this problem by allowing users to define trust boundaries around private regions of memory, called *enclaves*, in which users can store and operate on sensitive data. Several types of TEEs exist to date, including AMD Secure Encryption Virtualization, ARM TrustZone, and Intel SGX. We focus on SGX as our TEE of choice for the remaining descriptions, but the principles discussed in this paper can be extended to any TEE of a similar nature.

To utilize an enclave, a user must first partition their application code into trusted and untrusted components. Then, the user loads the trusted component of the code into an enclave through a series of enclave setup procedures. The process by which a user can confirm that an enclave was created securely and loaded with the expected code is called *(remote) attestation*, and it guarantees security against all attacks assumed by the SGX adversary model [1].

TLS channel establishment can be integrated with the attestation procedure, as discussed in [20, 41], to further guarantee confidentiality, integrity, and freshness of values transferred between the user and the enclave against all attacks assumed by the TLS threat model.

SGX cannot solve all problems of secure code execution. In particular, SGX does not guarantee protection of enclave code that is written in an insecure manner (e.g., contains buffer overflow vulnerabilities or side-channel-producing computation). Intel contends that it is the enclave developer's responsibility to write secure and side-channel-free code [37]. Finally, users of SGX must trust Intel attestation services to correctly identify trustworthy platforms in order to trust the attested enclave.

2.2 Homomorphic Encryption

Homomorphic encryption schemes are a class of cryptographic schemes that enable computations on encrypted data. This property enables a user to outsource function computation to an entity that may not be fully trusted, without revealing the underlying input or output values to the entity. Efficient HE schemes such as BGV [9], BFV [8, 27], and CKKS [15] 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 encoding techniques introduced in [65].

HE schemes consist of 7 main stages: parameter selection, key generation, encoding, encryption, evaluation, decryption, and decoding. HE parameters (e.g., the ring degree and coefficient moduli of the polynomial quotient rings) are chosen during parameter selection to be within the bounds of the Homomorphic Encryption Security Standard [11] and determine both the security level of the scheme and the maximum multiplicative depth of the target computation circuit. During key generation, a user generates a secret key ($\mathbf{SK}$) for symmetric encryption and decryption and certain encryptions of the secret key known as *evaluation keys* ($\mathbf{EK}$) that assist with any ciphertext-ciphertext multiplications or ciphertext rotations performed during evaluation.

The remaining stages represent the main components of the HE protocol that contribute to its running time. Encoding converts a vector of input values (integers or real numbers) into a plaintext, while encryption transforms a plaintext into a ciphertext. Decryption and decoding reverse this process, transforming a ciphertext into a plaintext, then plaintext into a vector of values. Evaluation refers to the computation on ciphertexts, in the form of addition, multiplication, or rotation operations, that is typically performed by an untrusted party. For addition and multiplication operations, the computation may be between two ciphertexts (`CT-CT`) or between a ciphertext and a plaintext (`CT-PT`).

2.2.1 Homomorphic Inference

In their CryptoNets work [30], Bachrach et al. first described how to use HE to achieve oblivious neural network inference for an MNIST classification network. The main idea behind this work and others that followed [10, 12, 18, 36] was to interpret the linear network layers as a series of additions, multiplications, and rotations between fixed-length vectors. Non-linear network layers, such as ReLU, were approximated by linear operations such as square. Using these methods, prior works achieved 98.4% or higher accuracy for inference over the MNIST dataset for homomorphic evaluation of a 5-layer CNN [10].

While CrypoNets demonstrated these techniques using the BFV scheme, more recent works [12, 36, 40] prefer the newer, more efficient CKKS scheme for ML-related tasks. The CKKS scheme has the unique advantage that it can efficiently discard unwanted precision in results of HE computation, essentially preserving an *approximate* computation on the input vector [1]. 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. Given our target application of oblivious inference, we focus on the CKKS scheme in this work.

2.2.2 Security of HE and CKKS

The CKKS scheme is based on the (Ring) Learning with Errors assumption and is IND-CPA secure. Though a recent work [44] proposed a passive key-recovery attack on the CKKS scheme, this attack requires access to a decryption oracle. As a simple mitigation, the decryptor can choose not to share the raw decryption results[2] with any untrusted entity. Additionally, HE schemes are *malleable* by nature, meaning that an HE ciphertext can be directly transformed into another valid ciphertext. They offer no guarantee, therefore, that the untrusted party computes a function $f()$ instead of a different function $f'()$ over an input. Thus, HE schemes alone are unable to provide users with computational integrity.

2.3 Adversary Modeling

Traditionally, in the field of multi-party computation and related works, adversaries are classified into one of two categories: *semi-honest* (a.k.a., *honest-but-curious*) and *malicious*. A semi-honest adversary refers to an adversary that,

[1] Discarding precision has a high computational cost in BFV and BGV [13].

[2] Sharing the general outcome of a decryption, such as the predicted class in a classification network, however, is not considered insecure.

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, on the other hand, may actively and arbitrarily deviate from the protocol specification and may seek to compromise either the privacy of participant data or the correctness of protocol execution.

While useful for modeling certain scenarios, however, the above two designations fail to adequately capture how many adversaries actually behave in practice. Some adversaries, for example, cannot be trusted to only act passively (as in the semi-honest case) but may still follow certain components of the prescribed protocol if they are incentivized to do so. For example, a model provider may ultimately wish to provide a user with a functional service that provides the user with some benefit, perhaps to prevent the client from using the service of a competitor, or it may be that the provider would not wish to deviate in a manner that would compromise the security of their own private data. Put another way, the provider would not be expected to act maliciously unless there were a clear incentive to do so.

To deal with the above problems and more effectively model real-world adversary behavior, we employ a *rational* adversary assumption in this work. This adversary is described in several works in the intersection of multi-party computation and game theory [2, 32, 48, 77] as a realistic yet powerful assumption upon which efficient MPC schemes can be based. Such an adversary can be thought of as having capabilities nearly equivalent to that of a malicious adversary, but bounded by the adversary's incentive to maximize their utility according to some list of preferences[3].

We point out that the rational adversary model considers an additional problem not typically captured by semi-honest or malicious models regarding the usefulness of any private values the adversary may provide to the protocol. Consider, for example, a protocol for computation that takes some private value $\mathbf{V}$ as input from a semi-honest or malicious party P_1 and provides another party P_2 with the result. Since value $\mathbf{V}$ constitutes P_1's private data, and since there is often no way of describing in the protocol whether P_1 provides some input $\mathbf{V} := \texttt{Input\_1}$ over some other $\mathbf{V} := \texttt{Input\_2}$ to the protocol, there is no restriction[4] on what P_1 defines as $\mathbf{V}$. Since the value of $\mathbf{V}$ likely affects the computation result, a solution that claims security for P_2 against a semi-honest or malicious P_1 may still allow P_2 to receive incorrect results undetected. On the other hand, a rational model can allow us to reason about why such an outcome may or may not be possible under the assumptions, which may be a valuable feature for real-world deployments.

Table 1: Summary of Terms / Symbols

$\mathcal{S}$	Cloud server / Cloud model provider
$\mathcal{E}_\mathcal{S}$	Enclave established on $\mathcal{S}$
$\mathcal{C}$	Client / Feature Provider
$\mathcal{M}$	ML model provider
$\mathcal{A}_\mathcal{S}$	Adversary $\mathcal{S}$ (malicious)
$\mathcal{A}_\mathcal{C}$	Adversary $\mathcal{C}$ (malicious)
$\mathcal{A}_\mathcal{M}$	Adversary $\mathcal{M}$ (rational)
π	HE parameters
$\mathbf{W}$	$\mathcal{M}$'s neural network weights and biases
$\mathbf{X}$ / $\mathbf{Y}$	$\mathcal{C}$'s neural network feature inputs / outputs
$\overline{\mathbf{X}}$, $\overline{\mathbf{Y}}$	HE-encrypted $\mathbf{X}$, $\mathbf{Y}$
$\mathbf{F}$	Neural network inference circuit
$\mathbf{I_F}$	An implementation / code of $\mathbf{F}$
$\overline{\mathbf{I_F}}$	$\mathbf{I_F}$ encrypted with Intel PCL [33]
σ	Communication channel

Finally, we note that, in order to protect against any of the adversaries mentioned above, a solution must either prevent the adversary from acting in a way that would compromise the security of the protocol or allow parties to identify adversarial actors and abort the protocol before the adversary is able to compromise sensitive data[5]. We do not distinguish between the two types of security in this work since either is sufficient for our problem setting.

3 Problem Statement

We consider the scenario of two-party secure 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, type, and sizes of layers) as $\mathbf{F}$. We would like to allow the client to submit their inputs to provider $\mathcal{M}$ to obtain result $\mathbf{Y}$=$\mathbf{F}$($\mathbf{X}$, $\mathbf{W}$) in a secure manner that preserves the privacy of $\mathbf{X}$ and $\mathbf{Y}$ (as well as any intermediate ciphertexts) and guarantees no corruptions of the result $\mathbf{Y}$. We refer to $\mathbf{X}$ and $\mathbf{Y}$ collectively as "client data" and provide a summary of all symbols used in Table 1.

Importantly, we also wish to free the ML model provider from having to maintain a private online infrastructure; that is, we

[3] This adversary is similar in concept to a *covert* adversary [4], who may be willing to (passively or actively) cheat, but would only do so if it can ensure it would not be caught. The covert adversary model specifies a deterrence factor ε that indicates the probability with which other parties would be able to catch a covert adversary attempting to cheat. A rational adversary model, on the other hand, does not require a specific deterrence factor or cheating detection method and instead simply assumes certain protocol deviations may or may not occur according to the preferences of the rational adversary.

[4] Some MPC models describe a notion of input consistency, which seeks to verify that if a party P_1 inputs $\mathbf{V}$ into some portion of the protocol, and the same input is required elsewhere in the protocol, the same value of $\mathbf{V}$ is used in both places. This does not prevent P_1 from defining $\mathbf{V}$ as `Dataset_1` instead of `Dataset_2` in both places, however.

[5] Prior works sometimes refer to these two cases as *robust security* and *security with abort*.

would like $\mathcal{M}$ to be able to take full advantage of all the benefits of using a public cloud. Thus, we introduce an additional entity $\mathcal{S}$ as the cloud model 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 $\mathcal{S}$ in a secure manner that preserves the privacy of $\mathbf{W}$ (as well as the privacy of client data) and prevents cloud adversaries from tampering with data execution.

3.1 CHEX-MIX Adversary Model

We consider three main types of adversaries in our solution: 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 $\mathcal{A}_\mathcal{S}$ may represent an untrusted cloud model provider themselves or a third-party attacker that is able to take advantage of 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 operating system or hypervisor layer of any public cloud-based servers. The client adversary $\mathcal{A}_\mathcal{C}$ may act maliciously to try and learn as much about the model provider's weights and bias values as possible. We classify the model provider adversary $\mathcal{A}_\mathcal{M}$ as a rational adversary (see Section 2.3) that seeks to maximize their utility under the following preferences: 1) learning the values of $\mathbf{X}$, $\mathbf{Y}$, or any intermediate ciphertext results, and 2) providing $\mathcal{C}$ with a useful inference service. In other words, we assume that $\mathcal{M}$ would like to provide $\mathcal{C}$ with a useful inference service, and $\mathcal{M}$ will not act in a way that counteracts this purpose *unless doing so might help reveal $\mathcal{C}$'s private data*. In Section 5.1, we further note that, while we consider $\mathcal{A}_\mathcal{M}$ to be rational, our solution offers $\mathcal{C}$ data privacy protection *even if $\mathcal{A}_\mathcal{M}$ were considered arbitrarily malicious*. We make no assumptions about collusion between the adversaries (i.e., we assume $\mathcal{A}_\mathcal{M}$ may collude with $\mathcal{A}_\mathcal{S}$, and $\mathcal{A}_\mathcal{C}$ may collude with $\mathcal{A}_\mathcal{S}$).

3.1.1 Scope

Recent works have demonstrated how, by treating a machine learning model as a black-box oracle, client adversaries may be able to extract a local model that is the same or similar to the deployed model (called a *model-stealing* attack), or may be able to infer information about the training set used to develop the model (a *model-inversion* or *re-identification* or *membership inference* attack [63, 70, 71]). Protection against these types of attacks is not the focus of our work, though several other works detail mitigations that providers can take against such attacks in practice [29, 38, 47, 63]. These techniques can be added on top of our approach for added protection if desired, as discussed in more detail in Appendix B.

Additionally, consistent with the threat model of SGX (see Section 2.1), we do not consider side-channel attacks as part of our threat model. Nevertheless, we provide a more detailed consideration of our solution with respect to side-channel adversaries in Section 7.2. We also consider denial-of-service attacks to be out of scope for our work.

Finally, we address a concern pointed out by a prior work [58] regarding the lack of a property known as *circuit privacy* present in the 2PC-HE hybrid solution for oblivious inference described in [39]. While no prior work has demonstrated how an attacker can utilize a lack of robust circuit privacy to break the security of an HE-based oblivious inference protocol in practice, we provide a more detailed discussion on this topic in Section 7.1. We note in particular how our work differs significantly from the construction of [39] and how these differences render the concerns regarding circuit privacy less applicable to our work.

3.1.2 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, though any similar secure network channel establishment process can be used in its place. In order to securely share evaluation keys $\mathbf{EK}$ with untrusted parties, all prior RLWE-based HE works utilize a circular security assumption to assume it is secure to share encryptions of the secret key $\mathbf{SK}$ under itself. We make this assumption in our work as well. Additionally, prior HE works for neural network inference [10, 12, 18, 30] 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 homomorphic evaluation [16]. We employ the same approach in our evaluation and often refer to the "machine learning model" as the model without this final layer. Finally, since our solution leverages SGX, we assume that Intel attestation services are trusted to correctly identify when an enclave is malformed. Though this assumption is not required to ensure the privacy of $\mathcal{C}$'s data, it is necessary to ensure the privacy of $\mathcal{M}$'s data and the correctness of protocol execution.

4 Related Work

In Section 4.1, we first describe alternate approaches proposed by prior works for oblivious inference. We provide a summary of the main points discussed in this section in Table 2. In Section 4.2, we discuss how our work differs from prior discussions of HE-TEE hybrid solutions.

4.1 Prior Approaches for Oblivious Inference

4.1.1 TEE-Only

TEE-only solutions for oblivious inference [52, 57, 61, 68, 69] heavily rely on remote attestation to guarantee privacy of client

Table 2: Summary of features of our solution compared to prior approaches for the problem of two-party oblivious inference. A combined approach of HE+TEE provides security guarantees unaddressed by prior work while allowing $\mathcal{M}$ to offload computation to the public cloud. "HE" techniques (excluding MKHE) refer to techniques where input feature values are encrypted while weights and bias values remain in plaintext form (i.e., `CT-PT` computations). Dashes (–) indicate that a feature does not apply, since the technique does not allow offload to an untrusted cloud. (R) refers to a "rational" adversary type, and (M) refers to a "malicious" adversary type.

Technique	$\mathcal{M}$ can fully offload	Privacy of $\mathbf{X}$, $\mathbf{Y}$ (for $\mathcal{C}$)		Integrity of $\mathbf{Y}$ (for $\mathcal{C}$)		Privacy of $\mathbf{W}$ (for $\mathcal{M}$)	
		$\mathcal{A}_{\mathcal{M}}$ (R)	$\mathcal{A}_{\mathcal{S}}$ (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 [39, 49] assume only a semi-honest/passive model provider.
[2] Most prior 2PC and HE-2PC solutions, with the exception of MUSE [43], assume a semi-honest 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 from TEEs requires an impractical assumption that clients must thoroughly verify or blindly trust the security of enclave code. While ML model providers may possess the level of security expertise and code review resources necessary to thoroughly analyze enclave code for security vulnerabilities, especially in the common scenario where model providers develop their own inference code in-house, most clients would not possess the same level of resources or expertise necessary to do so. Therefore, clients would not be able to fully trust the enclave code to maintain confidentiality of their private data. This problem becomes especially critical as the number of ML providers – and thus the number and diversity of offered services – continues to grow, and robust community review of even publicly available enclave code becomes infeasible.

4.1.2 HE-Only

CryptoNets [30] and LoLa [10] proposed HE-based solutions for two-party oblivious inference. In these works, clients are expected to encrypt their private inputs and send the resulting ciphertexts to an untrusted server for homomorphic evaluation of ML inference. However, this solution requires the server to maintain a private online infrastructure to perform the homomorphic evaluation in order to maintain privacy of their parameters, preventing the model provider from making use of the public cloud. Alternatively, E2DM [36] 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, their technique assumes a *semi-honest* cloud and an *honest* client.

4.1.3 Multi-Key HE (MKHE)

Multi-key HE [12, 14, 45] enables a client and model provider to encrypt their private values using their respective private 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 infrastructure for the partial decryption process. More importantly, this technique is not secure against active adversaries and is only secure against semi-honest adversaries if the partial decryption is performed with a secure method such as noise flooding.

4.1.4 2PC / Hybrid-2PC

Here we exclusively 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 technologies such as HE.

Several 2PC and hybrid-2PC protocols have been proposed for the problem of oblivious inference [39, 43, 47, 49, 51, 55, 58]. However, all of these works require the model provider to maintain a private infrastructure for protocol execution and thus do not utilize the public cloud. Furthermore, with the exception of [43], these works make a much weaker semi-honest adversary assumption for both model provider and client adversaries, which prior work [43] has demonstrated can lead to devastating results in the client-malicious setting. Additionally, while some works have claimed 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 to reduce computation/communication costs [43, 49]. An interesting MPC-TEE hybrid solution was proposed in [42], though unfortunately, without any experimental neural network inference results for the 2PC setting.

Known 3PC solutions for oblivious inference [42, 50, 59, 75, 76] 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. These solutions also include a large communication cost to share the model to all three parties, which must be incurred *per inference*, and cannot provide security guarantees when all public cloud servers collude or act maliciously.

4.2 Prior HE-TEE works

The few prior works [19,28,79] that propose hybrid HE-TEE solutions for privacy-preserving inference propose using TEEs to perform extremely sensitive stages of the HE flow (e.g., HE encryption and/or decryption, involving knowledge of the client's secret key). These works, therefore, still share the drawback of regular 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 the code is free from vulnerabilities (including any intentionally-inserted backdoors) in order to guarantee their data privacy. The authors of these works additionally only assume a use-case of single-party outsourced computation and do not discuss their solutions for the problem of collaborative machine learning. Furthermore, since these works ultimately propose having HE computations occur in the non-enclave portion of the untrusted cloud, they offer no integrity protections against a malicious server.

Still fewer prior works have proposed executing homomorphic evaluation routines inside a TEE. Drucker and Gueron [25, 26] suggest applying HE to a TEE-based solution for outsourced computation. However, their solution uses the addition-only Pallier HE scheme, which is significantly less computationally powerful and less complex than modern HE systems such as CKKS that support both addition *and* multiplication on ciphertexts. Thus, their technique is only applicable to a small range of problems and cannot be applied to neural network inference. Though the authors present an evaluation of their system for a simple database query, they do not justify why a TEE-only solution would not be sufficient for their problem, present a security analysis, compare with prior work, or provide critical details of their implementation (e.g., the encryption parameters, libraries, or TEE-allocation size used). The authors briefly suggest that TEEs can be used to provide integrity to an MKHE-based voting system but do not provide any implementation results for this extension.

In their master's thesis [64], Singh details the process of porting the TFHE homomorphic encryption library [17, 67] to the Rust programming language and evaluates their solution on a "fused" millionaire problem inside an enclave. As noted by the author, however, this problem would actually require a *multi*-key HE scheme to be used to be deployed in practice, unlike the *single*-key version that the author deployed. Similar to the above works, the author also does not propose using an HE-TEE combined approach for protecting the privacy of two separate parties, nor does the author provide evaluation results for an ML benchmark. Additionally, TFHE ciphertexts are orders of magnitude larger than the underlying messages, implying a high communication cost.

5 Our Solution: CHEX-MIX

5.1 Baseline Protocol

We describe our baseline protocol for two-party oblivious inference below, summarized in Figures 3, 4, and 5.

5.1.1 Setup ($\mathcal{M}$)

$\mathcal{M}$ begins the setup phase of the protocol by verifying that code $\mathbf{I_F}$ securely implements $\mathbf{F}$. While SGX does provide certain guarantees for code and data protection (see Section 2.1), the SGX threat model considers it the enclave developer's responsibility to verify that enclave code is free of all vulnerabilities. As discussed in Section 4.1.1, $\mathcal{C}$ cannot be expected to possess the resources or expertise necessary to thoroughly vet any segment of code. $\mathcal{A}_\mathcal{M}$ may ordinarily try to exploit this property by inserting backdoors into $\mathbf{I_F}$ that would thus be undetected by $\mathcal{C}$. However, since the client's data remains homomorphically encrypted throughout the enclave computation (described in Section 5.1.3), the rational $\mathcal{A}_\mathcal{M}$ can recognize that inserting a backdoor to the enclave will not allow it to compromise $\mathcal{C}$'s data privacy. Thus, having no other rational reason to do so, $\mathcal{A}_\mathcal{M}$ will not attempt to add a backdoor to the enclave code. Since $\mathcal{A}_\mathcal{M}$ cannot achieve its primary objective (to compromise $\mathcal{C}$'s data privacy, see Section 3.1), it falls back to ensuring its second preference: to provide $\mathcal{C}$ with a useful service. We can be sure $\mathcal{M}$ will thoroughly analyze and secure $\mathbf{I_F}$ from any vulnerabilities potentially exploitable by the cloud adversary $\mathcal{A}_\mathcal{S}$ since attacks by $\mathcal{A}_\mathcal{S}$ could potentially compromise the utility of the service (e.g., by affecting the integrity of the computation). $\mathcal{M}$ then proceeds with the protocol if $\mathbf{I_F}$ passes all internal security reviews.

Next, $\mathcal{M}$ establishes an enclave $\mathcal{E}_\mathcal{S}$ with code $\mathbf{I_F}$. Before provisioning $\mathcal{E}_\mathcal{S}$ with private data $\mathbf{W}$, $\mathcal{M}$ establishes an attested TLS channel (see Section 2.1) with $\mathcal{E}_\mathcal{S}$ to ensure $\mathcal{E}_\mathcal{S}$ was initialized correctly and that the communication endpoint is the expected enclave. In order to later allow clients to connect to $\mathcal{E}_\mathcal{S}$ without an attestation requirement (discussed in more detail below), $\mathcal{M}$ 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

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 code $\mathbf{I_F}$ 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.

Figure 3: CHEX-MIX baseline setup protocol for $\mathcal{M}$

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 σ_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 σ_C.

Figure 4: CHEX-MIX baseline setup protocol for $\mathcal{C}$

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 5: CHEX-MIX baseline evaluation protocol

the end of this process, which is only required once across all clients, $\mathcal{M}$ can exit the protocol and go offline.

5.1.2 Setup ($\mathcal{C}$)

After $\mathcal{M}$ completes the above setup procedure, $\mathcal{C}$ connects to $\mathcal{E}_\mathcal{S}$ to receive the aforementioned certificates and the HE encryption parameters π to use. After verifying the parameters[6] 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. This setup phase for $\mathcal{C}$ must be executed once per client.

5.1.3 Evaluation

Following the one-time setup phase, $\mathcal{C}$ can request an inference result from $\mathcal{E}_\mathcal{S}$ based on its inputs $\mathbf{X}$. $\mathcal{C}$ encrypts $\mathbf{X}$ to $\overline{\mathbf{X}}$ using $\mathbf{SK}$, sends $\overline{\mathbf{X}}$ to $\mathcal{E}_\mathcal{S}$ over the established network channel, and waits for $\mathcal{E}_\mathcal{S}$ to compute the HE evaluation over the neural network. $\mathcal{C}$ then receives the (still HE encrypted) result $\overline{\mathbf{Y}}$ from $\mathcal{E}_\mathcal{S}$. Finally, $\mathcal{C}$ decrypts and decodes $\overline{\mathbf{Y}}$ to $\mathbf{Y}$ using $\mathbf{SK}$ to obtain the inference result.

We further elaborate on a few points pertaining to the above protocol. First, we require all communication between clients and the enclave and between the model provider and the enclave to be secured by the TLS protocol upon network channel establishment. Once a secure channel is established between two entities in the protocol, all further communication occurs over that secure channel. Transmitting client data via a secure channel might appear redundant since the HE ciphertexts and evaluation keys do not necessarily need this second encryption for confidentiality. However, this step 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.

A stipulation that is generally not addressed in prior HE works is the protection required for evaluation keys $\mathbf{EK}$. Since these keys are themselves types of HE ciphertexts, they require integrity protection during and after use. During evaluation, integrity protection of $\mathbf{EK}$ can be provided by the enclave $\mathcal{E}_\mathcal{S}$ itself. After evaluation, $\mathcal{E}_\mathcal{S}$ can opt not to store $\mathbf{EK}$ between subsequent requests from the same client (i.e., $\mathcal{E}_\mathcal{S}$ could maintain a *stateless* service, requiring the client to retransmit $\mathbf{EK}$ for each subsequent set of inference requests), or $\mathcal{E}_\mathcal{S}$ can store $\mathbf{EK}$ in an integrity-protected database (i.e., $\mathcal{E}_\mathcal{S}$ could maintain a *stateful* service and retrieve the client's $\mathbf{EK}$ from the database upon subsequent connections). The implementation of such an integrity-protected key retrieval database is not the focus of this work. However, we note that $\mathcal{M}$ may implement this service within the evaluation enclave itself (e.g., for a small number of clients) or by establishing a separate enclave or set of enclaves for this purpose (e.g., for a large number of clients, using an SGX-based database design such as those detailed in [56, 74]), or use some other integrity-protected database scheme. For simplicity, we model a stateful service that stores evaluation keys in the evaluation enclave itself in our evaluation results, discussed in the next section.

Finally, we note that our solution does not require[7] $\mathcal{C}$ to attest $\mathcal{E}_\mathcal{S}$ because $\mathcal{M}$ already verifies the security of $\mathbf{I_F}$ with respect to $\mathcal{A}_\mathcal{S}$ in its setup phase and uses remote attestation to verify that $\mathcal{E}_\mathcal{S}$ is a valid SGX enclave with code $\mathbf{I_F}$. $\mathcal{C}$ can assume $\mathcal{M}$ would have completed this step correctly since a vulnerability that would allow $\mathcal{A}_\mathcal{S}$ to modify or view the contents of $\mathcal{E}_\mathcal{S}$ could compromise the correctness of the inference

[6] Verifying the security of HE parameters is computationally simple and only involves checking that the parameters are within the ranges recommended by the Homomorphic Encryption Security Standard [11] for a given desired security level (and can therefore be accomplished by a simple comparison).

[7] In Appendix A, we provide a discussion of any additional guarantees that could be provided with attestation enabled, along with additional mechanisms $\mathcal{C}$ can use for integrity assurance.

service (without providing $\mathcal{A}_{\mathcal{M}}$ with any meaningful way to collude with $\mathcal{A}_{\mathcal{S}}$ to learn the values of $\mathcal{C}$'s private data since $\mathcal{C}$'s data privacy is protected by the IND-CPA of HE). Since $\mathcal{M}$ is rational, $\mathcal{C}$ can thus ensure an endpoint endorsed by $\mathcal{M}$ (i.e., the enclave $\mathcal{E}_{\mathcal{S}}$) is secure with respect to $\mathcal{A}_{\mathcal{S}}$, and that $\mathcal{M}$ would not themselves attempt to modify the inference result or circumvent enclave deployment to break the correctness of computation. Therefore, so long as $\mathcal{C}$ can verify that they are communicating with an endpoint verified by $\mathcal{M}$, which $\mathcal{C}$ can achieve through standard certificate verification, $\mathcal{C}$ can be assured of secure communication with $\mathcal{E}_{\mathcal{S}}$.

5.1.4 Security Analysis

As discussed in Section 3, we indicate that our solution must provide the following guarantees in order to be secure: confidentiality of $\mathcal{C}$'s inputs $\mathbf{X}$, results $\mathbf{Y}$, any intermediate ciphertexts; confidentiality of $\mathcal{M}$'s inputs $\mathbf{W}$; and correct evaluation of $\mathbf{F}$. The confidentiality of $\mathbf{X}$ and $\mathbf{Y}$ is secured against malicious adversaries by the IND-CPA security of HE. The confidentiality of intermediate ciphertext results is secured against a malicious $\mathcal{A}_{\mathcal{C}}$ by the TEE and against all other malicious adversaries by the IND-CPA security of HE. The confidentiality of $\mathcal{M}$'s data $\mathbf{W}$ is protected against malicious adversaries by the security properties of the TEE since $\mathcal{M}$ verifies via remote attestation that $\mathcal{E}_{\mathcal{S}}$ is a secure enclave containing code $\mathbf{I_F}$ and verifies that $\mathbf{I_F}$ contains no vulnerabilities exploitable by an adversary to access enclave code or data. $\mathbf{W}$ is protected at transit by the attested TLS channel used by $\mathcal{M}$ to securely transfer $\mathbf{W}$ to $\mathcal{E}_{\mathcal{S}}$,

To ensure computation correctness, which is defined as $\mathbf{Y} = \mathbf{F}(\mathbf{X}, \mathbf{W})$, all computation and associated data must be protected from modification by adversaries. In step 1 of $\mathcal{M}$'s setup protocol, $\mathcal{M}$ verifies that code $\mathbf{I_F}$ implements desired functionality $\mathbf{F}$ using homomorphic evaluation techniques such that $\mathbf{Y} = \mathbf{F}(\mathbf{X},\mathbf{W}) = \mathtt{Decrypt}(\mathtt{Eval}(\mathbf{I_F},\overline{\mathbf{X}},\mathbf{W},\mathbf{EK}))$. Since $\mathcal{M}$ cannot learn $\mathcal{C}$'s private data even if $\mathcal{M}$ actively cheats this verification, $\mathcal{M}$ is expected to be rational and trusted by $\mathcal{C}$ to correctly perform this verification[8]. $\mathcal{C}$ and $\mathcal{M}$ ensure correct endpoints for network communication with $\mathcal{E}_{\mathcal{S}}$ through standard certificate verification and certificate verification with attestation, respectively. When in transit, values $\mathbf{W}$, $\overline{\mathbf{X}}$, and $\overline{\mathbf{Y}}$ are thus integrity protected by the TLS protocol. The security properties of the TEE further guarantee that $\mathbf{I_F}$, while deployed, is executed correctly even in the presence of malicious cloud adversaries. $\mathbf{EK}$ is either kept inside the protected enclave, or stored in and retrieved from an integrity-protected database. Since all computation and inputs are integrity protected against $\mathcal{A}_{\mathcal{S}}$, $\mathcal{A}_{\mathcal{M}}$, and $\mathcal{A}_{\mathcal{C}}$, the correctness of computation is ensured.

5.1.5 Takeaways

The CHEX-MIX protocol is beneficial to $\mathcal{C}$ and $\mathcal{M}$ in several ways. First, CHEX-MIX relieves $\mathcal{C}$ of the burden of having to thoroughly analyze code $\mathbf{I_F}$ for security vulnerabilities. Second, the procedure allows $\mathcal{M}$ to remain offline after the initial setup phase of the protocol. This characteristic differentiates our solution from prior works, which often require $\mathcal{M}$ to maintain a private online infrastructure to communicate with the untrusted cloud server or client directly. Third, our solution guarantees privacy of provider data $\mathbf{W}$ and correctness of inference execution $\mathbf{F}$ even while deployed in the public cloud. Thus, our solution allows ML model providers to more effectively utilize the scale made possible by public cloud infrastructures.

5.2 Achieving Privacy of $\mathbf{I_F}$ IP

Recall from Section 4.1.1 that, in a TEE-only solution, $\mathcal{C}$ requires access to code $\mathbf{I_F}$ in order to attest $\mathcal{E}_{\mathcal{S}}$. This attestation step, as well as $\mathcal{C}$'s thorough verification of the enclave code $\mathbf{I_F}$, is necessary in a TEE-only solution to ensure $\mathcal{C}$ of both data privacy and computational integrity.

By contrast, in our protocol, $\mathcal{C}$ is ensured of data privacy from its use of homomorphic encryption and can assume computational integrity by the assumption that $\mathcal{M}$ is rational. Thus, our solution removes the requirement for $\mathcal{C}$ to have access to code $\mathbf{I_F}$, providing $\mathcal{M}$ the unique opportunity to maintain privacy of $\mathbf{I_F}$. We note that the ability to keep $\mathbf{I_F}$ code private would likely be highly useful to model providers in a real-world setting since many technology service provider companies consider software implementations of their products to be valuable IP. Additionally, protecting the details of $\mathbf{I_F}$ may in turn provide $\mathcal{M}$ with some level of *model architecture privacy*, which may further reduce the likelihood of other model parameter attacks (e.g., model inference or model extraction attacks, see Appendix F for more details).

The standard SGX deployment procedure, however, does not offer confidentiality of enclave *code* since the enclave binary is sent in-the-clear to the remote server and may be reverse-engineered by a cloud attacker. As a solution to this problem, Intel released a Protected Code Loader (PCL) library that allows a user to treat an enclave binary as private enclave data, thereby offering privacy protection of software IP. While Intel markets PCL as a tool that can protect a provider's private IP, it does not claim that PCL can encrypt the entirety of the generated shared object binary. Some sections of the enclave binary that PCL is not able to encrypt include portions specific to the PCL code loader code itself, the BSS segment, and any debugging info, with the full list given in [35]. Intel contends it is the responsibility of the PCL user to verify that unencrypted segments do not reveal private IP. We assume this is possible for $\mathcal{M}$ to do, and thus treat $\mathbf{I_F}$ here as those specific components of the inference code that constitute private IP. Appendix C gives an example of the protection provided by PCL for the

[8] Even if $\mathcal{M}$ acts maliciously and cheats this verification, this will not result in a loss of confidentiality for $\mathcal{C}$'s private data, by the IND-CPA of HE. Therefore, we offer even stronger protection than assumed by the rational model.

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 6: CHEX-MIX protocol including privacy of $\mathbf{I_F}$ IP for $\mathcal{M}$

enclave binary symbol tables.

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 of the CHEX-MIX baseline protocol. In particular, 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 2. We 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[9]. Since both $\mathcal{E}_{\mathcal{S}_1}$ and $\mathcal{E}_{\mathcal{S}_2}$ were 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 $\mathbf{I_F}$.

Table 3: Description of the CNN used to benchmark our solution based on the description given in Eff.-MKHE [12].

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

6 Evaluation

To demonstrate the efficiency and feasibility of our solution, we implement both configurations of CHEX-MIX on a commodity cloud server using Microsoft Azure. We use the `Standard DC8_v2` VM server configuration for our work with 8 vCPUs and 32 GB of RAM. All our experiments are compiled with GNU CC (version 7) on Ubuntu 18.04 and executed with a single thread at 3.7 GHz.

6.1 Implementation Details

Enclave developers face a unique set of challenges when developing enclave code. First, many standard programming functionalities developers are accustomed to, such as the ability to use standard C/C++ library routines, are not natively available inside enclaves. To cope with lacking these critical components, developers are encouraged to use an SDK such as the official Intel SGX SDK or the OpenEnclave SDK (the latter of which was developed as a TEE-agnostic SDK by multiple companies, including Intel) to access these functionalities in a secure manner. However, there are still several commonly-used programming functionalities that even these SDKs are unable to support, adding an additional layer of difficulty to enclave code development. For example, the SDKs do not support the use of `<fstream>` function calls, which natively involve the untrusted host OS. We compile the server-side enclave component of the Microsoft SEAL library with certain optional optimizations disabled (namely, `SEAL_USE_INTRIN`, `SEAL_USE_EXPLICIT_BZERO`, and `SEAL_USE_MSGSL`, as well as `ZLIB/ZSTD` compression support) to remove calls to functions not supported by the enclave/SDK. Similarly, the SDKs do not provide access to runtime counters (e.g., calls to the C++ `chrono` library) that developers typically use to measure code performance. Instead, we implement these measurements by calling 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 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. We additionally disable C++17

[9] It is also possible to construct a sealing key known only to a specific enclave instance (i.e., not derivable even by another enclave signed by the same developer). We do not require this version for our solution.

optimizations in Microsoft SEAL for server evaluation experiments since C++17 is not supported by the Intel SGX SDK (used for our PCL experiment, described below). We note that disabling these optimizations does not affect the security of our solution and instead at worst only affects the performance or communication efficiency of the library by a small amount. We discuss the cost of removing compression in more detail at the end of this section.

For our baseline protocol evaluation, we utilize the Microsoft SEAL library (version 3.7.1) [62] for the HE portion of our solution, and the Open-Enclave SDK (version 0.15.0) [54] in pre-release mode for implementing the server-side enclave. We chose the Open-Enclave SDK rather than the Intel SGX SDK for our experiments since the Open-Enclave SDK is more user-friendly, and unlike the Intel SGX SDK, provides guidance on the details of integrating standard secure channel establishment (i.e., TLS) with attestation. We disabled the Microsoft SEAL flags listed earlier for the server-side HE code but left all default flags enabled for the client-side HE implementation since the client does not run inside the enclave. For simplicity, we simulate a stateful service (see Section 5.1) for our server deployment. Additionally, since we focus on implementing the performance-critical portions of our solution to provide useful evaluation results, we did not deploy features such as a full client-side attestation result analysis (which would not affect the runtime of the solution). However, we did integrate the server-side pre-release mode attestation quote generation process and server-client TLS channel establishment for proof of concept and to ensure any effect the code had on increasing the enclave binary size (and therefore, potentially, enclave runtime) would be captured. Since communication latencies can vary widely between network connection infrastructures, user devices, and application types (e.g., high-performance use cases vs. IoT, cellular vs. ethernet, etc.), we omit network communication latencies from our overall runtime results.

We also implement the version of our solution described in 5.2 to provide $\mathcal{M}$ with privacy of code $\mathbf{I_F}$. This experiment is as similar to the baseline experiment as possible, with the following differences: 1) we use the Intel SGX SDK (version 2.15) [34] rather than the OpenEnclave SDK for this experiment since OpenEnclave does not yet support integration with Intel PCL; 2) for simplicity, and since we already implement the baseline protocol using full attestation and secure channel establishment procedures, we do not re-implement these components with the Intel SGX SDK; and 3) we implement the server evaluation code in this version with randomly generated data (i.e., not the true weights/bias or input ciphertext values necessary) and/or generate necessary objects (e.g., evaluation keys) from within the enclave itself, and we take care to ensure the sizes of all plaintext, ciphertext, and key objects used are the same as those used by the baseline. Table 4 shows that the runtime achieved by this solution is nearly identical to the non-PCL version, proving that our solution can provide this additional benefit to service providers essentially for free. We emphasize that a TEE-only deployment could *not* offer protection of $\mathbf{I_F}$ IP since such deployments would require clients to have full knowledge of enclave code for attestation.

In all our experiments, we target the same CNN and apply the same inference evaluation structure as the MKHE work [12], which we refer to as Eff.-MKHE. We also use the same HE parameters chosen in Eff.-MKHE of a polynomial ring degree of 16384, and 8 modulus primes with 53 or 60 bits each for a total modulus size of 438 bits. This choice of CNN and parameter set allows us to fairly compare against Eff.-MKHE, which achieved a model accuracy of 98.4% over the MNIST test dataset[10]. Additionally, by using a CNN with parameters previously tuned by HE experts, we can more fairly assess the capabilities of our solution compared to alternate techniques (see Appendix E). We give the details of the target CNN in Table 3.

To ensure as fair a comparison with Eff.-MKHE as possible, we upgrade the original Eff.-MKHE implementation to use the latest version of Microsoft SEAL, including integrating the same modifications the original work made to an earlier version of the SEAL library into the latest SEAL version. We then re-run their workload on the same server as our experiments and report this more efficient runtime in Table 2, which may be of independent interest.

Finally, we note that 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. SGXv2 removes this requirement and allows the enclave to dynamically allocate memory as needed. For our experiments, we target the more widely available SGXv1, though our findings should apply to SGXv2 as well. We conservatively enable the baseline experiment enclave to use up to 512 MB of stack and 2 GB of heap memory, and the PCL-encrypted enclave to use up to 128 KB of stack and up to 512 MB of heap memory. We note that we only perform a rudimentary analysis to determine performant stack and heap memory configurations, so future work may be able to determine tighter bounds for these sizes.

6.2 Results and Comparison with Prior Work

In Table 4, we give a comparison of our runtime and communication costs per inference to those of prior approaches for oblivious inference, for the target workload of inference over the MNIST dataset, with workloads using similarly-sized CNNs [11]. For simplicity, we only include works that have been shown most efficient for the target evaluation (i.e., for multiple

[10] Eff.-MKHE [12] uses public-key CKKS encryption and `CT-CT` homomorphic evaluation operations, as this was necessary for the functionality of their solution. Since our work instead enables symmetric CKKS encryption and `CT-PT` homomorphic evaluation operations, the final HE ciphertext error will be less than in the prior work [12]. Thus, we expect our model to have the same or better final model accuracy as [12] (i.e., $\geq$ 98.4%).

[11] See Appendix D for a description of the CNNs used in other works.

Table 4: Runtime and communication cost efficiency of CHEX-MIX compared to prior work 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. GAZELLE [39], DELPHI [49], and MUSE [43] separate their performance and communication costs into "offline" and "online" components (though both must be repeated per inference). We separate these two costs by a $(+)$ symbol. A number n in parenthesis following a runtime denotes the thread count, where n is the number of threads used when $n > 1$. Bolded measurements are from this work.

Work	Technique	$\mathcal{M}$ can fully offload	Privacy $(\mathbf{X},\mathbf{Y}$ / $(\mathcal{A}_{\mathcal{M}})$	Privacy $(\mathbf{W}$ / $\mathcal{A}_C)$	Integrity $(\mathbf{Y}$ / $\mathcal{A}_S)$	Runtime (s)	Comm. (MB)
XONN[1] [58]	2PC	✗	✗	✗	–	0.16	38.28
GAZELLE[1, 2] [39]	2PC+HE	✗	✗	✗	–	0.15 + 0.05	5.9 + 2.1
DELPHI[2,3] [49]	2PC+HE	✗	✗	✗	–	7.41 (1-8) + 0.48 (8)	235.52 + 10.24
MUSE[3] [43]	2PC+HE	✗	✗	✓	–	22.67 (1-6) + 0.80 (8)	4270.08 + 10.24
Eff.-MKHE [12]	**MKHE**	✗	✓	✗	✗	**1.09**	**2.125**
-	**HE**	✗	✓	✓	–	**0.35**	**1.125**
CHEX-MIX	**HE+TEE**	✓	✓	✓	✓	**1.32**	**1.125**
CHEX-MIX	**HE+TEE (PCL)**	✓	✓	✓	✓	**1.38**	**1.125**

[1] Work did not explicitly mention evaluation thread count.

[2] Runtime includes computation and LAN communication latency. Authors did not provide separate runtimes for computation only.

[3] Runtime is the sum of multiple parts, each with a different thread count.

Table 5: Runtime and communication breakdown for Eff.-MKHE, HE-only, and CHEX-MIX baseline evaluations. HE-only and CHEX-MIX solutions enable lower client runtime and communication costs than Eff.-MKHE.

Work	Runtime (s)		Comm. (MB)	
	Client	Server	Inputs	Results
Eff.-MKHE [12]	0.014	1.08	1.75	0.375
HE-only	0.008	0.34	0.875	0.25
CHEX-MIX		1.31		

works that use the same technique, we list the work with the most efficient runtime).

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. We include a subset of the problem statement differences from Table 2 alongside these results to remind the reader of some of the key advantages of our approach over prior solutions. We also further discuss our solution compared to prior works in more detail below. We omit a comparison with TEE-only works since this method cannot provide client data privacy against even a semi-honest model provider (see Section 4.1.1).

6.2.1 XONN

Compared to CHEX-MIX, XONN [58] is $8\times$ faster but has a $34\times$ higher communication cost. Unlike CHEX-MIX, but similar to other 2PC solutions, XONN does not allow $\mathcal{M}$ to offload computation to a public cloud server. Additionally, XONN assumes a much weaker passive (semi-honest) threat model and therefore does not provide privacy guarantees in the presence of a malicious $\mathcal{A}_C$ or rational $\mathcal{A}_{\mathcal{M}}$.

6.2.2 GAZELLE, DELPHI, and MUSE

Similar to XONN, none of these three solutions allow $\mathcal{M}$ to offload computation to the public cloud. These solutions further only assume a semi-honest threat model and therefore do not protect against active adversaries, with the exception of client-malicious protection offered by MUSE. Though these solutions choose to split their evaluation into an "offline" and "online" phase, as shown separated by a $(+)$ symbol in Table 2, CHEX-MIX has a lower communication cost even compared to just the online phase of all three of these works. Further, the "offline" phase of these works is required for each inference and should also be counted for a fair comparison against our work, rendering our improvement even more significant.

It is difficult to fairly compare the runtime of our solution against DELPHI [49] and MUSE [43] since these works only report multi-threaded computation results. Compared to MUSE with *multiple* threads, which has the closest threat model to our solution amongst the 2PC+HE solutions, our solution with a *single* thread is $16.5\times$ faster and requires $3805\times$ less communication per inference. Though the performance of DELPHI and MUSE in Table 2 are both reported by [43] over a seven layer network, we note that our work would still be much more efficient even if two of the seven layers were removed from DELPHI or MUSE since their costs depend linearly on the number of layers. Finally, CHEX-MIX achieves $\geq$ 98.4% accuracy, while neither DELPHI's nor MUSE's accuracy was reported in [43].

6.2.3 Eff.-MKHE

Eff.-MKHE [12] allows offloading 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 is only secure against a semi-honest server if the model provider performs partial decryption with a secure method such as noise flooding. The implementation in [12] did not include such a secure method and is therefore vulnerable to a passive key-recovery attack on CKKS [44]. Compared to Eff.-MKHE, CHEX-MIX reduces the communication cost by 1 MB (see Section 6.3), achieves a comparable runtime, and since CHEX-MIX uses symmetric HE encryption where Eff.-MKHE requires asymmetric encryption, results in a smaller client computation cost (see Table 5).

6.2.4 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, $\mathcal{M}$ needs to host a private online infrastructure for HE evaluation, losing the scalability and flexibility offered by the public cloud. The performance of "HE" and "HE+TEE" in Table 2 are measured with the same code executed outside and inside an enclave, respectively.

6.3 Communication Cost

For our parameters, we would normally expect a freshly encrypted ciphertext to require 16384 $\times$ 8 bytes per polynomial coefficient $\times$ 8 primes $\times$ 2 ciphertext components = 2 MB. Microsoft SEAL employs a noise reduction technique that consumes a prime during encryption, bringing this cost down to 1.75 MB. When symmetric HE encryption is used, this cost can be further reduced by representing one component of the ciphertext as a random seed to be later expanded by the receiver using an extendable output function (e.g., Blake2xb or SHAKE-256 in Microsoft SEAL), a technique which reduces the communication cost to 0.875 MB (for a 64-byte seed). During HE evaluation, modulus primes are consumed by the operations to reduce noise growth in intermediate results. The final result output ciphertext contains one prime for a total of 16384 $\times$ 8 bytes $\times$ 2 components $=$ 0.25 MB, bringing the total communication cost to 1.125 MB. Since MKHE results in an output ciphertext with three components rather than two, the result ciphertext size in MKHE is 0.375 MB.

Using Microsoft SEAL with our choice of primes ensures that even the maximum values of ciphertext and key polynomial coefficients require fewer bits than the datatype size of a `uint64_t`. Therefore, the upper bits of many coefficients are guaranteed to be zero and can easily be compressed by a standard compression algorithm or even manual omission. Microsoft SEAL contains an interface to use ZLIB or ZSTD compression libraries for this purpose, though we omit these from our experiments. For our solution, applying this compression would result in a new communication size of 0.738 MB for the input ciphertext and 0.242 MB for the output ciphertext, for a new total communication cost of 0.98 MB.

7 Discussion

7.1 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. As discussed in previous works [39, 58], HE ciphertexts contain error terms that change as homomorphic evaluation operations are performed on the ciphertexts, potentially leaking information about the types of operations performed or the plaintext values multiplied or added with the encrypted input. As pointed out in [58], the implementation in [39] did not adequately satisfy the circuit privacy property. As a hybrid 2PC-HE protocol, [39] requires a decryption of HE ciphertexts after *every* linear layer evaluation, making it plausible that an adversary could, in theory, derive the model weights from the decrypted values. By contrast, our protocol only involves client decryption after *all* network layers are evaluated. After several such layer computations, it becomes much less clear that an attacker could plausibly use the final HE ciphertext error to learn the weights used in intermediate layers. We note that 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.

7.2 Side channels

Although we do not consider side channels as part of our threat model (consistent with Intel's official threat model for SGX [37]), we nevertheless wish to devote some discussion to them given their attention in the academic literature. In this context, a *side-channel adversary* is an adversary that exploits paths of unintended information leakage that result from implementations of a technology, such as variations in timing, power, or electromagnetic radiation, to recover a secret value.

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 [37]. For example, SGX requires enclave developers to take steps to write their enclave code in a secret-data-independent manner if protection against timing side-channel adversaries is desired. The second type of side-channel adversary appears much more complicated to defend against; prior works have detailed attacks mounted through side-channel vectors that are either outside of the enclave code developer's immediate purview or are very difficult for the developer to properly prevent [46, 78]. Intel purports to take these attacks seriously and continues to actively issue patches to SGX for numerous side channels, as well as other vulnerabilities, as mitigations are developed [37, 46, 66, 72]. It is therefore critical that users of SGX can ensure they are using the most up-to-date version of the technology, which users can do through proper enclave attestation. Additionally, several works ([5, 24, 53], to name a few) propose approaches that can be taken to mitigate the ability of attackers to perform side-channel attacks on TEEs, which can be added on top of our solution as well.

A significant advantage of our solution is that it does offer clients privacy protection from malicious side-channel adversaries. In particular, since client data is always in HE ciphertext form inside the enclave, any attack on the enclave cannot view the underlying data of the HE ciphertexts. We additionally take care to ensure that our implementation does not leak any information about $\mathcal{M}$'s private values $\mathbf{W}$ through timing side channels. Specifically, we verify that our inference code and the implementation of CT-PT operations in Microsoft SEAL do not contain any data-dependent computation, branching, or memory access based on $\mathcal{M}$'s private data. Still, we note that this is not a default property of an HE-TEE hybrid solution but rather results from a secure implementation of both enclave code and SGX technology.

8 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 solution that provides both clients and model providers with confidentiality and integrity guarantees. We demonstrate the feasibility of performing homomorphic evaluation inside TEEs by deploying CHEX-MIX on a Microsoft Azure confidential computing virtual machine. We compare our solution to prior approaches for two-party oblivious inference and show how our solution can provide security guarantees unaddressed by prior works. 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.

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] 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.

[3] 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

[4] 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.

[5] 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

[6] 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

[7] 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

[8] 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.

[9] 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.

[10] 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

[11] 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.

[12] 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.

[13] H. Chen and K. Han, "Homomorphic lower digits removal and improved FHE bootstrapping," in *EUROCRYPT 2018, Part I*, ser. LNCS, J. B. Nielsen and V. Rijmen, Eds., vol. 10820. Springer, Heidelberg, Apr. / May 2018, pp. 315–337.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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

[19] 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.

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

[21] B. Darvish Rouani, M. Samragh, T. Javidi, and F. Koushanfar, "Safe machine learning and defeating adversarial attacks," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 31–38, 2019.

[22] 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

[23] 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

[24] 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.

[25] 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

[26] ——, "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.

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

[28] 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.

[29] 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.

[30] 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

[31] J. Goodwin, "Elon Musk criticizes OpenAI exclusively licensing GPT-3 to Microsoft," https://www.cnn.com/2020/09/27/tech/elon-musk-tesla-bill-gates-microsoft-open-ai/index.html, Sep 2020.

[32] 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.

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

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

[35] "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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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

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

[55] 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.

[56] 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.

[57] 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

[58] 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.

[59] 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.

[60] 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

[61] 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.

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

[63] 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.

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

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

[66] 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.

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

[68] 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

[69] 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

[70] 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.

[71] 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.

[72] 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.

[73] A. Viand, P. Jattke, and A. Hithnawi, "Sok: Fully homomorphic encryption compilers," in *2021 2021 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 1092–1108. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00068

[74] 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.

[75] 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.

[76] 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.

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

[78] 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.

[79] 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

[80] 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

A Additional Correctness Guarantees

CHEX-MIX requires $\mathcal{M}$ to be rational in order to provide $\mathcal{C}$ with a guarantee of correctness. Here we discuss two additions to our solution that can provide $\mathcal{C}$ with additional correctness guarantees for more diverse problem settings if desired.

A.1 Client Attestation

If $\mathcal{M}$ were malicious instead of rational, $\mathcal{C}$ could not assume $\mathcal{M}$ would provide a correct implementation of $\mathbf{F}$, 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 $\mathbf{F}$, $\mathcal{C}$ could, in theory, verify for themselves that $\mathbf{I_F}$ implements $\mathbf{F}$. However, in practice, this places a high burden on $\mathcal{C}$. Moreover, $\mathcal{C}$ would still need to be able to verify that $\mathbf{I_F}$ is free from all vulnerabilities that could affect computational correctness, including any hidden backdoors inserted by a malicious $\mathcal{M}$.

Instead, $\mathcal{C}$ could again apply the rational assumption to guarantee that $\mathcal{M}$ would not insert any backdoors into $\mathbf{I_F}$. In this manner, we can reduce the functionality dependent on the rational behavior of $\mathcal{M}$ and allow $\mathcal{C}$ more control over verifying correct enclave execution. Figures 7 and 8 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 7: CHEX-MIX enhanced correctness verification setup protocol for $\mathcal{M}$

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

A.2 Sensitive Samples

A client may want additional guarantees of computation integrity, for example, from adversaries outside the scope of our threat model (e.g., side-channel adversaries). Furthermore, even the malicious model cannot guarantee that $\mathcal{M}$'s private weights values are not corrupted prior to $\mathcal{M}$'s participation in the defined protocol. In such cases, the following integrity checking mechanism may be helpful: 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 $\mathcal{E}_S$ for evaluation. If the decryption of $\mathbf{Y}$ matches what $\mathcal{C}$ expected, $\mathcal{C}$ can be somewhat 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. in [80] for providing users with integrity assurance for outsourced HE 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.

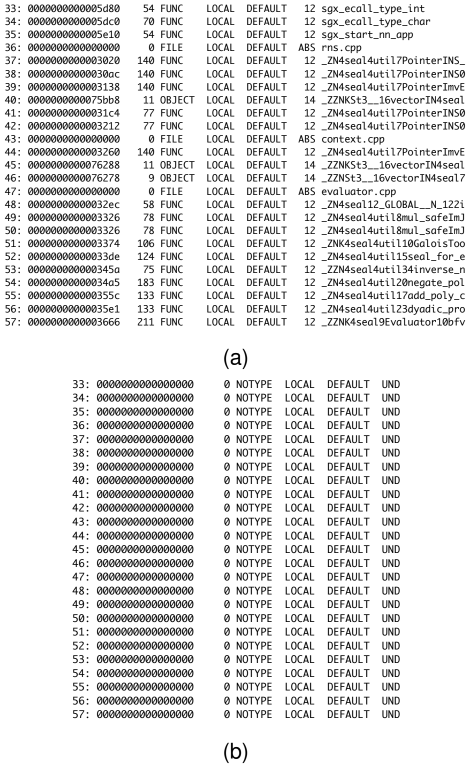
B Additional Defenses for $\mathcal{M}$

Recent works in privacy-preserving machine learning have demonstrated attacks mountable by a client adversary to extract the private weights of the model provider, called *model-stealing* attacks [70], or to learn information about the initial model training data set, including *model-inversion* or *re-identification*/*membership inference* attacks [63, 71]. These attacks involve having the client send several queries to the inference server and analyze the results to infer details about the model. We note that these attacks can be mounted against all prior works in oblivious inference, including prior MPC, Hybrid-MPC, TEE-only, HE-only, and MKHE solutions, and are thus not uniquely applicable to our work. Nevertheless, we follow the approach taken by XONN (see Appendix B of [58]) and discuss here how defense mechanisms for these attacks can be applied on top of our work.

A simple mitigation, suggested by [38, 47], 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 stand-alone network to ensure that the required number of queries to reverse engineer the model parameters is larger than is computationally feasible to analyze by clients. Finally, 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 [29, 63] 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, a rounding filter layer can be easily added to our CKKS-based approach.

C PCL-encrypted ELF File Output

As a small example of the protection provided by the Intel PCL library, Figures 9a and 9b 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.



```
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: 0000000000075bb8    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 9: Excerpt from symbol table output of running the `readelf` command on unencrypted (top) and encrypted (bottom) enclave binaries. The unencrypted symbol table reveals function calls and file names, while the PCL-encrypted symbol table does not.

D Neural Network Models of Prior Works

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

(The CNN used in Eff.-MKHE [12] is the same as used by our work and is given in Table 3.)

Table 6: Description of the CNN used to benchmark GAZELLE [39] (based on the description given in [60].)

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

Table 7: Description of CNN used to benchmark XONN [58]

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

Table 8: Description of the CNN used to benchmark DELPHI [49] and MUSE [43] (based on the description given in [47].). Adjacent linear layers are listed together since it is customary to combine them into a single linear layer during homomorphic inference.

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

E Benchmarking

Ideally, we would like to demonstrate the efficiency of our solution across a range of neural networks to demonstrate its scalability. Unfortunately, especially given the relatively recent introduction of efficient HE schemes such as CKKS, such a benchmark suite (or even another stand-alone network implementation) is not yet available for homomorphic inference. While some compilers exist for auto-generating HE evaluation code, the efficiency of generated output of these compilers still falls severely short of hand-chosen parameters and implementations by HE experts, such as the one we utilize in this work. To the best of our knowledge, EVA [22] (which subsumes prior work CHET [23]) and nGraph-HE [7] / nGraph-HE2 [6] are the only compilers that exist for optimizing CKKS kernels. Even using *56 threads*, EVA's compiled implementation requires 0.6 seconds – nearly *double* the latency of the *single*-threaded implementation we utilize – to evaluate a similarly sized CNN, and nGraph-HE2 requires 2.05 seconds per query (a slowdown of nearly $6\times$) and still requires expert-optimized parameter selection [73]. Given the currently limited ability of these compilers to produce efficient neural network inference code, we do not believe it would be fair to use the output of these compilers to analyze the efficiency of our solution or when comparing the efficiency of our solution against prior works. We note that our solution does not limit the size of HE evaluation networks and can always increase scalability by utilizing multiple TEEs.

F Model Architecture Privacy

As discussed in Section 5.2, CHEX-MIX enables model providers to use Intel PCL to encrypt their inference code $\mathbf{I_F}$, thereby achieving confidentiality of their code IP. Another reason why $\mathcal{M}$ may want to keep $\mathbf{I_F}$ confidential is that the details of $\mathbf{I_F}$ may leak information about the architecture of the deployed model. Here, we define the model architecture as the structural details of the model, such as the type of machine learning computation (e.g., logistic regression vs. neural network inference), the number of layers used, and the ordering, types, and sizes of the layers. This information, which constitutes knowledge of $\mathbf{F}$, is independent of the values of the weights and bias $\mathbf{W}$. Prior 2PC/hybrid-2PC works often naturally reveal the details of a model architecture in their solution, whether directly through the communication of garbled circuit tables or by exposing the exact ordering of linear and non-linear layers through correlated transitions between HE and 2PC protocol evaluation techniques. While earlier works [39, 47] argued that this information leakage is tolerable since some companies have released details of their model architectures, this precedent is already beginning to no longer hold, and model architectures are beginning to be considered vital pieces of intellectual property that provide companies with a competitive edge (see, for example, GPT-3 [31]). Furthermore, revealing details about the model architecture could potentially help adversaries conduct more efficient model stealing or model inversion attacks or help the adversary determine adversarial examples more efficiently (a so-called *gray-box* attack [21]). While CHEX-MIX does not guarantee robust model architecture privacy, it may provide stronger security in this regard than comparable solutions based on 2PC techniques and further motivates its value in practical deployments.