Fusion: Efficient and Secure Inference Resilient to Malicious Servers

Caiqin Dong[†], Jian Weng[†,✉], Jia-Nan Liu[†], Yue Zhang[†], Yao Tong[§], Anjia Yang[†], Yudan Cheng[†], and Shun Hu[†]

[†]Jinan University, [§]Guangzhou Fongwell Data Limited Company

E-mails: {caiqindong611, cryptjweng, zyueinfosec, anjiayang, gzhushun}@gmail.com,

{j.n.liu, yudan_cheng}@foxmail.com, melody@fongwell.com

Abstract—In secure machine learning inference, most of the schemes assume that the server is semi-honest (honestly following the protocol but attempting to infer additional information). However, the server may be malicious (e.g., using a low-quality model or deviating from the protocol) in the real world. Although a few studies have considered a malicious server that deviates from the protocol, they ignore the verification of model accuracy (where the malicious server uses a low-quality model) meanwhile preserving the privacy of both the server's model and the client's inputs. To address these issues, we propose *Fusion*, where the client mixes the public samples (which have known query results) with their own samples to be queried as the inputs of multi-party computation to jointly perform the secure inference. Since a server that uses a low-quality model or deviates from the protocol can only produce results that can be easily identified by the client, *Fusion* forces the server to behave honestly, thereby addressing all those aforementioned issues without leveraging expensive cryptographic techniques. Our evaluation indicates that *Fusion* is 48.06× faster and uses 30.90× less communication than the existing maliciously secure inference protocol (which currently does not support the verification of the model accuracy). In addition, to show the scalability, we conduct ImageNet-scale inference on the practical ResNet50 model and it costs 8.678 minutes and 10.117 GiB of communication in a WAN setting, which is 1.18× faster and has 2.64× less communication than those of the semi-honest protocol.

I. INTRODUCTION

Machine Learning as a Service (MLaaS) [61], [20], [10] is rapidly emerging as a dominant computing paradigm in the past few years. In such a paradigm, the servers provide cloud-based machine learning services for the clients. As such, the clients now do not need to train their own models (which is computationally expensive, and requires large datasets), but consume the services on demand: the clients can simply feed the server inputs, and let the server make inferences based on the inputs. For example, a patient can provide the raw medical data to the server, and the server can then generate diagnosis results using the pre-trained models.

While MLaaS has brought the enormous convenience, it is also subject to privacy risks. For example, the inputs of clients may be highly sensitive (e.g., raw medical data) with confidentiality concerns [14], and it is urgently necessary for the server to provide inference services without breaking their confidentiality. Currently, many efforts have been made toward this end [13], [55], [28], [43], [30], [7], [60], [46], [59], [58], [3], [18], [34] via various cryptographic techniques such as homomorphic encryption (HE), garbled circuits (GC), and secret sharing (SS). However, there could be plenty of large matrix multiplications, non-linear operations, and secure conversions back and forth between them in the neural network inference, which are relatively expensive to achieve with cryptographic techniques. To achieve practical and privacy-preserving MLaaS, most of those works have no alternatives but to settle for less by adopting a weaker threat model — assuming the servers and the clients will follow the protocol, but they will also try to obtain additional information (a.k.a, semi-honest security).

However, there is no reason to believe that such an assumption will always hold: the server can be completely malicious, and does not follow the protocols at all (e.g., the server returns the client random results to trick the clients) [17], [70], [16]. Furthermore, even if the server follows the protocol, there is no guarantee that the server will produce high-quality results as promised, as few protocols can verify whether the inference results are produced by a high-quality model. Those incorrect or inaccurate results can have grave consequences (e.g., misleading patients). As such, solutions that solely consider the semi-honest security are obsolete, and we have to deal with a much worse scenario, where (i) the server is completely malicious (i.e., malicious security) to deviate from the protocol and provide incorrect results, or (ii) although the server follows the protocol, it uses a low-quality model as input, and (iii) the server and client have the requirements of protecting their highly sensitive data (including the server's model and client's query input).

To our best knowledge, there is no solution that can satisfy all the criteria mentioned above. For example, *LevioSA* [18] achieves the maliciously secure arithmetic computation by following the high-level approach of the IPS compiler [24], [41] (IPS compiler is designed to construct secure protocols in the presence of malicious adversaries) and applies it to privacy-preserving machine learning. CRYPTFLOW [34] converts TensorFlow [2] inference code to MPC protocols. By

using the trusted hardware, their solution also satisfies malicious security. Although those solutions [34], [18] guarantee computation correctness (which ensures the server follows the protocol correctly) and preserve privacy, they do not verify the model accuracy (e.g., the malicious server could still use a low-quality model). Moreover, some schemes [17], [36], [70], [16], [44], [64] utilize zero-knowledge (ZK) proofs to compel the server to provide correct inference results for the client, but they only protect the privacy of either the server's model or the client's input.

This paper aims to propose a solution for MLaaS that satisfies all three criteria (including verification of model accuracy, computation correctness, and privacy preservation) at the same time. To that end, there could be multiple ways. For example, we can directly apply maliciously secure two-party computation (2PC), which preserves privacy and ensures computation correctness, but extra expensive cryptographic approaches (e.g., ZK proofs) are required to verify the model accuracy. Alternatively, we can make changes on the existing efficient semi-honest inference schemes [43], [30], [46] to achieve malicious security. Given that these schemes usually use multiple cryptographic techniques such as HE and GC simultaneously, it is also challenging to customize them to let them meet all three criteria (particularly, enabling the verification of model accuracy may require significant modifications) effectively.

Fortunately, we observe that the client can know the computation results (public samples' labels) of some inputs (or query samples) in advance, which is not possible in most secure computation applications. These pre-collected public samples with known (or expected) computation results can be used to verify the model accuracy meanwhile force the server to perform computations correctly by mixing them with real query samples. Based on this observation, we customize a *mix-and-check* method that combines the verification of model accuracy with computation correctness for batched inference queries. Specifically, we design a mixed dataset by preparing query samples (non-public, and each with multiple copies) and a number of public samples, and then using a random permutation to shuffle them. If the server attempts to cheat the client without being noticed, the server has to provide incorrect-but-consistent results for all copies of a particular query sample. Given that the server will not know how the samples (the public ones and non-public ones) are mixed according to our design, it is extremely challenging for the server to cheat successfully, and the clients can easily notice the misbehavior with overwhelming possibility.

By using this *mix-and-check* method, we propose a maliciously secure inference scheme, named *Fusion*, which can convert a semi-honest inference protocol into a maliciously secure one. *Fusion* is superior because it fulfills the aforementioned three security requirements effectively. It preserves the privacy of both the server and the client by utilizing a semi-honest secure inference protocol, and ensures the computation correctness and model accuracy effectively (it uses simple-but-effective local checks, not expensive cryptographic techniques). We have also implemented *Fusion* and compared

its performances with existing maliciously secure solutions and semi-honest inference protocols. Moreover, we conduct ImageNet-scale inference on practical ResNet50 model. When compared with CRYPTFLOW2 [59], *Fusion* is $1.30\times$ and $1.18\times$ faster in the LAN setting and the WAN setting respectively, and uses $2.64\times$ less communication (when the total number of query samples is 512 and the copies for each query sample is 5 that ensure the statistical security of 2^{-40}). Finally, we also enable *Fusion* to defend against model extraction attacks (by integrating a prior solution), which proves that *Fusion* has good scalability.

In short, our contributions are twofold:

- First, we propose *Fusion*, a maliciously secure inference scheme. To our best knowledge, *Fusion* is the first solution that preserves the privacy of both the server and the client, and ensures the model accuracy of the server's input and computation correctness at the same time in the *server-malicious* threat model. Particularly, *Fusion* can be used as a general compiler that converts a semi-honest inference scheme into a maliciously secure one. As a consequence, the proposed scheme can benefit from the existing efficient and practical inference schemes.
- Second, we implement *Fusion*, and compare its performance with the state-of-the-art maliciously secure work *LevioSA* [18]. The results are encouraging: *Fusion* is $48.06\times$ faster and uses $30.90\times$ less communication than *LevioSA* [18] (which currently does not support verification of the server's model accuracy). We also compare *Fusion* with multiple semi-honest inference protocols such as DELPHI [46] and ABY [13], and better performance is also observed on *Fusion* when they all adopted the same settings (e.g., the number of query samples is large enough such as 32).

II. PRELIMINARIES AND BACKGROUND

In this section, we describe some background information about neural network inference, followed by privacy-preserving neural network inference, where we introduce a few popular hybrid 2PC-based privacy-preserving inference protocols.

A. Neural Network Inference

Being one of the important types of deep learning, convolutional neural network (CNN) inference computations mainly contain four components, i.e., convolutions, which extract different features from a dataset, pooling layer, which mainly attempts to reduce the dimensionality, activation function, which is used as a feature transformation method to increase the expression ability, and fully connected layer, which takes the outputs of other components and produces the final outputs. All those components are connected, and ultimately form a multi-layer network structure, where the outputs of one layer can be the inputs of other layers. CNN inference usually contains a lot of computationally expensive linear and non-linear computations. For example, convolutions usually contain multiple linear matrix multiplications; activation functions such as

rectified linear unit (ReLU) [48] are nonlinear transformations. Pooling functions (e.g., max pooling functions) usually are nonlinear operations as well.

B. Privacy-Preserving Neural Network Inference

In MLaaS without considering privacy preservation, the client's inputs are directly sent to the server who produces the inference results, and the highly sensitive inputs (e.g., raw medical test results) can be leaked to the server. As such, the privacy-preserving neural network inference is introduced, which generates the outputs without leaking sensitive inputs to the server. At a high level, the client and the server adopt cryptographic techniques to perform inference computations for protecting their own inputs. Among all the privacy-preserving inference frameworks, semi-honest privacy-preserving inference gained popularity due to its practical efficiency.

An intuitive way to implement privacy-preserving inference is through two-party computation (2PC). 2PC is a sub-problem of secure multi-party computation (MPC), which allows two parties to jointly compute a function without sacrificing their input privacy. However, due to the complexity of operation types (e.g., nonlinear comparison operations, matrix multiplication) and plenty of conversions between these operations in neural networks, directly using general 2PC schemes faces efficiency challenges.

Having seen the efficiency challenges, researchers proposed hybrid 2PC-based privacy-preserving inference protocols [43], [30], [46], [21], which improve the overall efficiency by using appropriate cryptographic techniques for different kinds of computation or designing new efficient subprotocols. In the following, we would like to introduce a few popular hybrid 2PC-based privacy-preserving inference frameworks:

- **ABY** [13] is a semi-honest mixed-protocol framework that combines Arithmetic (A) sharing, Boolean (B) sharing, and Yao's (Y) garbled circuits, and designs efficient conversions between every pair of the three. It also adopts a set of existing state-of-the-art optimizations in a novel fashion and provides instantiations.

- **DELPHI** [46] combines the additive homomorphic encryption with garbled circuits (HE is used to perform linear matrix multiplication while GC is used to perform non-linear operations) and connects them by additive secret sharing (e.g., the output of the linear layer are additive secret shares of computation results that are ultimately fed into the GC for non-linear operations). Besides, it allows users to automatically generate neural networks that mix these two methods and navigate the trade-off between accuracy and performance.

- **CRYPTFLOW2** [59] is also a hybrid inference protocol that shares some similarity with DELPHI. To improve efficiency, it proposes a more efficient millionaires protocol $\Pi_{\mathrm{MILL}}^{l,m}$ for securely computing the Yao's millionaires' problem (which is used for non-linear operations such as DReLU and Maxpool). CRYPTFLOW2 also provides two options, i.e., SCI$_\mathrm{OT}$ and SCI$_\mathrm{HE}$ that use oblivious trans-

fer and homomorphic encryption techniques to perform computationally costly linear operations such as matrix multiplication or convolution.

- **Cheetah** [21] is also a hybrid secure inference scheme that utilizes two lattice-based homomorphic encryptions [8] (i.e., learning with errors (LWE) and its ring variant (ring-LWE)) to perform secure linear layers (e.g., matrix multiplications in convolution), and makes some optimizations on the millionaire protocol [59] for non-linear layer (e.g., activation function). They achieve performance optimizations based on the insightful observation that matrix multiplication results can be represented as the coefficients in specific positions of polynomial multiplication, which can be efficiently performed using ring-LWE. For the non-linear layers, they optimize the millionaire protocol by adopting VOLE-style OT and customizing truncation protocols.

III. THREAT MODEL AND SECURITY GOALS

In privacy-preserving neural network inference, the server and the client jointly perform secure 2PC inference computations with their private inputs (e.g., model parameters or query samples). As such, similar to all other privacy-preserving neural network inference, we consider two entities in the system model, as shown in Figure 1: The server owns a well-trained (in terms of accuracy) model for a specific task (e.g., medical diagnosis) and provides inference services to the clients. The client owns a set of query samples, and it attempts to obtain the correctly computed inference results from the server. During the process, the server is required to use a well-trained model as input, and the client uses query samples as inputs for privacy-preserving inference computations. In the rest of this section, we describe the threat model and security requirements.
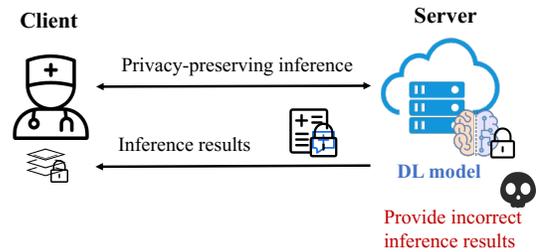

Fig. 1: System model.

A. System Model and Threat Model

Similar to previous efforts (e.g., [16], [64], [38]), in this paper, we consider the server is malicious that may not follow the protocol while the client is honest. In particular, given there are two roles (i.e., the server and the client), we make the following assumptions for each of them respectively:

- **Server:** We assume that the server may trick the clients by using a low-quality model (in terms of accuracy)

as input. Meanwhile, the server may deviate from the protocol specification. However, the server still aims to convince the client that the inference results are correctly computed from the client's query samples using a high-quality model (e.g., high accuracy).

- **Client:** The client honestly follows the protocol but may attempt to obtain additional information (i.e., semi-honest). The client also holds a set of samples to be queried. We additionally assume that the client has access to a subset of public samples whose type is the same as its query samples. This is reasonable since there are plenty of public datasets with different categories (e.g., images, audio, and healthcare data) that can be used for various purposes (e.g., face recognition and medical diagnosis). In particular, we can even find publicly available datasets that are as sensitive as medical records on some large national databases (e.g., TGGA [22] and NCBI [49]) or institutes (e.g., AIMI [6]), as data sharing can help accelerate disease research and improve diagnostic methods. Examples of those medical records datasets include genomic data [50], [23], gene expression profiles [66], [51], COVID-19-Data [45], [63], and medical image data [6], [39]. As such, it is practical for clients to obtain a number of public samples for use.

B. Security Goals

While the semi-honest assumptions (by assuming the server and the client will honestly follow the protocol without cheating and other misbehavior) are practical and widely adopted by a lot of previous efforts, there is no reason to believe that such an assumption will result in expected outputs. For example, the server may be malicious, which feeds the protocol low-quality model to trick the clients as discussed in §III-A. As such, this paper considers a much stronger threat model, which attempts to meet at least the following security goals: (1) we should not leak the input privacy of both the server and the client; (2) we should ensure the computation correctness of the outputs; and (3) we should ensure the accuracy of the inference results (e.g., the server uses high-quality models as inputs).

Formalization. Without loss of generality, we provide security in the simulation paradigm. We consider a *hybrid model* where parties both interact with each other and have access to ideal functionalities. Assume that a two-party hybrid-model protocol π uses ideal calls to ideal functionalities $f_1, \cdots, f_{p(n)}$. Let $\rho_1, \cdots, \rho_{p(n)}$ be protocols that securely compute $f_1, \cdots, f_{p(n)}$ respectively. The composition theorem [4] states that if π securely computes the functionality g in the $(f_1, \cdots, f_{p(n)})$−hybrid model, then $\pi^{\rho_1, \cdots, \rho_{p(n)}}$ in which the ideal functionalities are substituted by the secure sub-protocols securely computes g in the real model.

Definition 1: A protocol Π_{Fusion} between a server having model M which satisfies the accuracy threshold δ and a client having a dataset $\mathrm{X} = (\mathrm{x}_1, \cdots, \mathrm{x}_n)$ as inputs securely achieves a secure inference functionality $\mathcal{F}_{Fusion}$ against a malicious server and a semi-honest client if it satisfies the following requirements:

- **R1. Model Accuracy.** *The accuracy (e.g., η) of the model that the server uses as input should meet the requirement (e.g., δ). Assume that the model accuracy η is calculated. If $\eta \geq \delta$, the verification of model accuracy passes.*

- **R2. Computation Correctness.** *In an execution of Π_{Fusion}, the probability that the client's output on every input vector x_i is not the correct inference result $\mathrm{M}(\mathrm{x}_i)$ is negligible in security parameter λ.*

- **R3. Privacy.** *The server and the client both feed their private inputs to Π_{Fusion}. As such, from the perspective of privacy preservation, the client and the server is said to securely execute Π_{Fusion} against a malicious server and semi-honest client if the following properties are satisfied:*
  - **Malicious Server Security.** *The view of the server during a real execution of protocol Π_{Fusion} is denoted by $\mathsf{View}_{\mathcal{S}}^{\Pi_{Fusion}}$. For any server $\mathcal{S}$, there exists a probabilistic polynomial-time simulator $\mathsf{Sim}_{\mathcal{S}}$ such that for any input M of the server and x_i of the client, we have:*
  $$\mathsf{View}_{\mathcal{S}}^{\Pi_{Fusion}} \approx_c \mathsf{Sim}_{\mathcal{S}}(\mathrm{M})$$
  That is, $\mathsf{Sim}_{\mathcal{S}}$ can simulate a computationally indistinguishable view of the malicious server without knowing the client's private inputs and inference results.
  - **Semi-honest Client Security.** *The view of the client during a real execution of protocol Π_{Fusion} is denoted by $\mathsf{View}_{\mathcal{C}}^{\Pi_{Fusion}}$. For any client $\mathcal{C}$, there exists a probabilistic polynomial-time simulator $\mathsf{Sim}_{\mathcal{C}}$ such that for any input M of the server and x_i of the client, we have:*
  $$\mathsf{View}_{\mathcal{C}}^{\Pi_{Fusion}} \approx_c \mathsf{Sim}_{\mathcal{C}}(\mathrm{x}_i, \mathrm{M}(\mathrm{x}_i))$$
  That is, $\mathsf{Sim}_{\mathcal{C}}$ can simulate a computationally indistinguishable view of the semi-honest client without knowing the server's model.

IV. THE *Fusion* SCHEME

In this section, we propose *Fusion*, a privacy-preserving inference scheme that is secure against a malicious server and a semi-honest client. We first explain our intuition, then we present the overview of *Fusion*, and finally, we describe the current design in greater details.

A. Challenges and Key Ideas

As discussed in §III-B, there are three goals that need to be fulfilled in the presence of a malicious server, i.e., model accuracy, computation correctness, and privacy. To our best knowledge, there is no existing work that satisfies all three goals mentioned above in neural network inference. The latter two requirements (R2 and R3) can be achieved by directly adopting general maliciously secure protocols (e.g., popular SPDZ-style protocols [12], [31]), though they are not efficient enough for performing neural network inference. In addition, maliciously secure protocols do not provide guarantees for the model accuracy (e.g., those protocols cannot force the server to use a high-quality model as input). Even though there are some
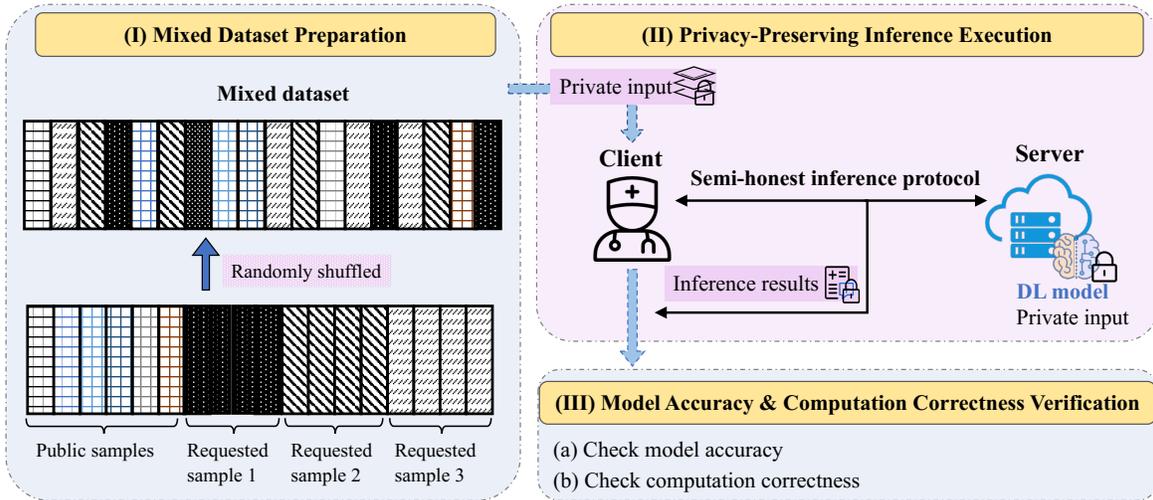
Fig. 2: The workflow of *Fusion*. Particularly, for better illustration, we highlight the local operations of the client as blue, and highlight the interaction between the server and the client as pink.

additional techniques that could potentially be used to verify model accuracy, they usually involved heavy cryptographic techniques (e.g., using ZK proofs and commitments [64] to convert publicly committed data into privately authenticated values). As such, it is challenging to achieve all three goals in neural network inference efficiently.

Our goal is to achieve all three security requirements without using heavy cryptographic techniques such as ZK proofs. Generally, the privacy requirement (R3) can be achieved by adopting efficient 2PC protocols to perform the neural network inference. We now discuss how to fulfill model accuracy (R1) and model correctness (R2) at the same time. Our idea is that if we can know the outputs of specific inputs in advance, we can use such knowledge to ensure the *computation correctness* and *model accuracy*. This is because if the server causes wrong results (violating R2) or uses low-quality inputs (violating R1), the client can notice that by comparing the expected outputs (which are known in advance) with the actual outputs (which are obtained at real time), thereby forcing the server to behave honestly. Particularly, knowing the outputs in advance is possible, since there are samples (e.g., TGGA [22] and NCBI [49] databases) where both the inputs and outputs (labels) are publicly available, as discussed in §III-A. The key question now becomes how we can find a way to appropriately utilize these public samples, and feed them into the server to achieve R1 and R2.

The first solution that comes to our mind is *cut-and-choose* technique [42], a leading technique used to convert Yao's garbled circuit [69] to be maliciously secure: in our scenarios, we need a method to pinpoint the server's malicious behavior, while the insight behind the *cut-and-choose* technique is that if a malicious generator (which takes responsible for constructing garbled circuits for computing a function, such as a server) constructs incorrect garbled circuits, the evaluator (which evaluates the garbled circuits to obtain outputs, such

as a client) would detect the malicious behavior with high probability. We now provide more details of batched *cut-and-choose* (that can amortize the cost across many executions of the same protocol): the generator first constructs many garbled circuits and sends them to the evaluator, then the evaluator asks the generator to open some of the garbled circuits (e.g., revealing the decryption keys corresponding the chosen garbled circuits) for checking the correctness of these garbled circuits (meaning that the circuit indeed computes the expected function). If all opened circuits are correctly constructed, the remaining circuits are randomly grouped into various buckets of a specific size, and all garbled circuits in the same bucket are evaluated with the same inputs from two parties. As a result, the evaluator would identify the malicious behavior (by checking the construction correctness of opened circuits and the result consistency of all copies in the same bucket) with high probability if the generator constructs incorrect garbled circuits.

However, the *cut-and-choose* strategy cannot be directly applied to compile the hybrid 2PC-based inference protocol (which typically utilizes additive secret sharing to connect two different cryptographic protocols such as HE and GC) to a maliciously secure one. The reasons for that are multiple: First, the *cut-and-choose* technique for GC only ensures the correctness (meaning that the garbled circuit is correctly constructed for computing the expected function). However, in our case, we need to verify the server's inputs (not the produced garbled circuits) which the *cut-and-choose* technique itself cannot achieve. Second, the *cut-and-choose* technique directly opens the generated circuits for checking correctness because the garbled circuits themselves do not relate to specific inputs (if they are never used for evaluation), and therefore can be opened publicly without sacrificing the privacy of both parties' inputs. However, in our scenarios, we customize the *mix-and-check* method to convert semi-honest

inference protocols into maliciously secure counterparts, while the underlying hybrid 2PC-based inference protocols generally do not have the structure that can be directly opened for checking correctness without sacrificing input privacy.

Key Insight. Fortunately, inspired by the *cut-and-choose* technique, we propose a novel technique named *mix-and-check*, which forms our technical foundation, and allows us to overcome all challenges above. Our idea is to let the client mix a set of public samples (imitating the evaluator's choice of randomly opening some garbled circuits) together with the samples to be queried, and use them as inputs for executing privacy-preserving inference (R3). If the client observes any inconsistency (e.g., the outputs of these public data records do not equal to their expected outputs, or inconsistent results are observed across some of the copies), we detect that the server does not honestly use a high-quality model or the server does not follow the specified protocol. These public samples are used to verify the model accuracy (R1) and computation correctness (R2) at the same time in a novel fashion.

B. Overview of Fusion

Based on our newly proposed *mix-and-check* method, we design a new maliciously secure inference protocol, *Fusion*. We assume that before the server and client run *Fusion*, the server trains a model and uses it to provide the inference services. As shown in Figure 2, *Fusion* works as follows:

1) **Mixed Dataset Preparation.** The client locally prepares a mixed dataset (with public samples, and the samples to be queried) and utilizes the mixed dataset as input for the privacy-preserving inference computations.
2) **Privacy-Preserving Inference Execution.** The server and the client jointly perform the semi-honest privacy-preserving inference protocols, and the inference results are only revealed to the clients.
3) **Model Accuracy and Computation Correctness Verification.** The client verifies the model accuracy on public samples, and computation correctness through the consistency check of inference results for all copies of every query sample. The client accepts the inference results if the above two checks pass.

C. Detailed Design of Fusion

We summarize the maliciously secure inference protocol Π_{Fusion} in Figure 3 and describe the process in detail as follows.

(I) Mixed Dataset Preparation. In this phase, the client locally prepares the *mixed dataset*. To that end, the client first selects R *query samples*, duplicates B copies of every *query samples* (in total, there are $R * B$ copies), and prepares T *public samples* for future inference computations. Next, the client uses a randomly chosen permutation to shuffle all public samples and copies of all query samples.

The random shuffle of the query samples and public samples is easy to design and implement, and therefore, in the following, we would like to discuss how to select optimal B and T. Particularly, we have two requirements. First, the server should

not successfully trick the client into believing that the incorrect or low-quality results are correctly produced by a high-quality model. The statistical security parameter is denoted by λ. The security requirement is to guarantee that the server succeeds in cheating with a probability (which depends on specific R, B, and T) at most $2^{-\lambda}$. We denote this requirement as *security requirement*. Second, in order to achieve the best efficiency, the goal of this phase is to pick the concrete numbers of B and T (given a specific R) that minimize the cost per query sample while satisfying the *security requirement*. We denote this requirement as the *cost requirement*. Specifically, computation and communication costs per query sample is proportional to the number of public samples T and the number of copies B for each query sample. As such, we have to solve a parameter optimization problem that satisfies the *security requirement* meanwhile meeting the *cost requirement*.

We are now starting with the *security requirement*. The server can succeed in cheating the client when (1) the server passes the **accuracy check of public samples**: accuracy is calculated based on the T public samples; and (2) the server also passes the **consistency check of query samples**: the inference results of all copies for every query sample are consistent and the inference results of some query samples are incorrect (produced by a low-quality model or incorrect inference computations). Without loss of generality, we further define those two types of passes mathematically:

- **Accuracy Check of Public Samples:** To simplify the problem, we assume that the server knows T and B. Assume that the server attempts to corrupt i query samples by providing iB incorrect-but-consistent inference results for all copies of every query sample. If the server attempts to pass the check of model accuracy (accuracy check of public samples), it should use the high-quality model as input for the T public samples. Let E_T denote the event in which the server uses a high-quality model to correctly perform the inference computations on all T public samples. As such, we have the probability $\Pr[E_T]$ that event E_T happens:

$$
\begin{aligned}
\Pr[E_T] &= \frac{\binom{RB + T - iB}{T}}{\binom{RB + T}{T}} \\
&= \frac{(RB + T - iB)!(RB)!}{(RB - iB)!(RB + T)!}.
\end{aligned}
\tag{1}
$$

- **Consistency Check of Query Samples:** Let E_B denote the event in which the iB incorrect inference results chosen by the server are exactly the incorrect-but-consistent results for i query samples. There are $(RB)!$ ways to permute the RB query samples. Again, if the server attempts to cheat the client successfully, it should provide incorrect-but-consistent inference results for iB copies of the i query samples, and use the high-quality model to perform correct inference computations for the remaining

Input: The server inputs a model M for a specific inference task, the client inputs R query samples $\mathrm{X} = (\mathrm{x}_1, \cdots, \mathrm{x}_R)$ and T public samples $\{(\mathrm{x}_1^*, y_1^*), \cdots, (\mathrm{x}_T^*, y_T^*)\}$, and an accuracy threshold δ is set.
Output: The inference results $\mathrm{M}(\mathrm{x}_i')$ $(i \in \{1, \cdots, RB+T\})$.
1) **Mixed Dataset Preparation (only the client).**
   a) Searches optimal (B, T) that minimize $cost(B, T, R) = \frac{RB+T}{R}$ by using the search protocol Π_{Search} (Figure 4).
   b) Duplicates B copies for each query sample to obtain $\{(\mathrm{x}_1^1, \cdots, \mathrm{x}_1^B), \cdots, (\mathrm{x}_R^1, \cdots, \mathrm{x}_R^B)\}$, uses a random permutation π to mix them together with $(\mathrm{x}_1^*, \cdots, \mathrm{x}_T^*)$, and finally obtains a mixed dataset $\mathrm{X}' = (\mathrm{x}_1', \cdots, \mathrm{x}_{RB+T}')$.
2) **Privacy-Preserving Inference Execution.**
   Using $\{\mathrm{x}_1', \cdots, \mathrm{x}_{RB+T}'\}$ and M as inputs respectively, the client and the server jointly invoke the semi-honest inference protocol and reveal the computation results $\mathrm{M}(\mathrm{x}_i')(i \in \{1, \cdots, RB+T\})$ to the client.
3) **Model Accuracy and Computation Correctness Verification (only the client).**
   a) For $i \in \{1, \cdots, T\}$, if $\mathrm{M}(\mathrm{x}_i^*) = y_i^*$, sets $y_i = 1$. Calculates the **model accuracy** η as follows.

$$\eta = \frac{\sum y_i}{T}.$$

   If $\eta < \delta$, the client aborts.
   b) Verifies the **computation correctness** by checking whether $\{\mathrm{M}(\mathrm{x}_i^1), \cdots, \mathrm{M}(\mathrm{x}_i^B)\}(i \in \{1, \cdots, R\})$ are all the same. If there is any inconsistency, the client aborts.

Fig. 3: **Protocol Π_{Fusion} for batched secure inference**.

samples. Similarly, the probability $\Pr[E_B]$ that event E_B happens is as follows:

$$\Pr[E_B] = \frac{\binom{R}{i}(iB)!(RB - iB)!}{(RB)!}. \quad (2)$$

Given that the server can succeed in cheating when it can pass both *accuracy check of public samples* and *consistency check of query samples* at the same time, combining Equation 1 and Equation 2, the probability $\Pr_{success}$ that the server succeeds in cheating is as follows:

$$\begin{aligned} \Pr_{success} &= \Pr[E_T \wedge E_B] \\ &= \Pr[E_T] \times \Pr[E_B] \\ &= \binom{R}{i}\binom{RB+T}{iB}^{-1}. \end{aligned} \quad (3)$$

As long as $T \geq B$, we have the following equation (which is proved in §V):

$$\binom{R}{i}\binom{RB+T}{iB}^{-1} \leq R\binom{RB+T}{B}^{-1}$$

The *security requirement* states that $\Pr_{success}$ should be no more than $2^{-\lambda}$ for every choice of i by the server and of R, B, and T by the client.

On the basis of satisfying the *security requirement*, another goal is to find the optimal B and T that meet the *cost requirement* (e.g., leading to the lowest amortized cost per query sample). We denote a cost function $cost(B, T, R) = \frac{RB+T}{R}$ which denotes the amortized cost per query sample. Specifically, the parameter optimization problem must ensure the security constraint $\Pr_{success} \leq 2^{-\lambda}$, and should try to minimize the cost function. We additionally set a lower bound β of the

number of public samples for ensuring the reliability of the accuracy check of public samples. Therefore, the parameter optimization problem can be expressed as follows.

$$\underset{B,T}{\arg\min} \frac{RB+T}{R}, \quad (4)$$

subject to

$$T \geq \beta, \quad (5)$$

$$\Pr_{success} \leq 2^{-\lambda}. \quad (6)$$

To find the optimal B and R that satisfy the *security requirement* with minimized amortized cost, we design a search algorithm (shown in Figure 4) based on the probability constraint and cost function. For a given R, we search for a pair (B, T) that minimizes the cost function $cost(B, T, R) = \frac{RB+T}{R}$ while satisfying the *security requirement* (e.g., $\Pr_{success} \leq 2^{-\lambda}$). Specifically, for every choice of B ranging from 2 to λ, we search for T for a given B until we find the smallest T that satisfies the *security requirement* (the amortized cost decreases as T decreases). We continue to explore and update the optimal pair (B^*, T^*) with the current pair (B, T) if the current pair saves more of the amortized cost (according to the cost function) than the optimal pair. Finally, we obtain the optimal (B, T) that minimizes the amortized cost while satisfying the *security requirement* when the protocol terminates.

(II) Privacy-Preserving Inference Execution. In this phase, the client consumes inference services provided by the server by feeding the privately mixed dataset as its input. The server and the client jointly perform the secure inference computations using the known semi-honest privacy-preserving inference protocols (e.g., Cheetah, CRYPTFLOW2, etc). For

Input: R, λ.
Output: (B^*, T^*).

For B from 2 to λ:
1) For T from $\min(\beta, B)$ to $+\infty$, find the smallest T that satisfies $\Pr_{success} \leq 2^{-\lambda}$.
2) If $cost(B, T, R) < cost(B^*, T^*, R)$, set $T^* = T$, $B^* = B$.

Output (B^*, T^*) pair that minimizes the amortized cost function.

Fig. 4: **Protocol Π_{Search} for searching optimized B and T.**

every sample in the mixed dataset, the semi-honest 2PC-based inference protocol is invoked to obtain the inference result. Finally, the inference results on all samples in the mixed dataset are only revealed to the client. At the end of this phase, the correctness of the inference results is not guaranteed and will be checked in the next phase. Please note that *Fusion* provides flexibility in choosing the 2PC-based privacy-preserving inference protocols. Given those are known protocols, we omitted the details for brevity.

(III) Model Accuracy and Computation Correctness Verification. When obtaining inference results on the mixed dataset, the client checks the *model accuracy* and *computation correctness*. Particularly, the model accuracy η is defined as the number of correct inferences on T public samples over T. When η is greater than a threshold (e.g., 0.95), then *model accuracy* is passed. Similarly, the client verifies the computation correctness by checking the consistency of the inference results on B copies of each query sample. If any of the B copies of a query sample are inconsistent, it is considered that the server attempted to deceive the client by giving incorrect inference results. The client accepts the inference results if both checks pass, otherwise, it aborts.

V. Security Analysis

In this section, we give the security analysis of *Fusion*. In particular, we denote our protocol Π_{Fusion}.

Theorem 1: Assuming the existence of Π_{PPML}^{semi} that securely achieves inference functionality $\mathcal{F}_{PPML}^{semi}$ under the semi-honest security, the protocol Π_{Fusion} for executing secure inference securely achieves the ideal functionality $\mathcal{F}_{Fusion}$ (with abort) against a semi-honest client and a malicious server in the $\mathcal{F}_{PPML}^{semi}$-hybrid model.

We prove *Fusion* is secure against a semi-honest client and a malicious server in the ideal or real simulation paradigm where the view in both the ideal and real worlds are indistinguishable. Hence, we need to prove that there exists a simulator in the ideal world that can simulate a view that is indistinguishable from the view in the real world for the adversary who corrupts either the client or the server.

Compromised Client (CC): We first consider the case that the client is corrupted by a semi-honest adversary. As described in Definition 1, we have to prove that there exists a simulator $\mathsf{Sim}_\mathcal{C}$ that can simulate the view that is indistinguishable from the view in the real world for the adversary. Since the client is semi-honest and acts the identical way as the client in $\mathcal{F}_{PPML}^{semi}$ except for additional local checks, there exists a simulator $\mathsf{Sim}_\mathcal{C}$ that can simulate the indistinguishable view by invoking the simulator of $\mathcal{F}_{PPML}^{semi}$.

Compromised Server (CS): We then proceed to the case that the server is corrupted by a malicious adversary and construct a simulator $\mathsf{Sim}_\mathcal{S}$. When the server is corrupted by a malicious adversary who can deviate from the protocol, the final output of the client may be incorrect and cause abort in both worlds. Specifically, we construct a simulator $\mathsf{Sim}_\mathcal{S}$ that simulates the view in the ideal world. We will show that the view in the ideal world is statistically indistinguishable from the view in the real world. In step 2 of Π_{Fusion} (Figure 3), we let simulator $\mathsf{Sim}_\mathcal{S}$ invoke the simulator of $\mathcal{F}_{PPML}^{semi}$ and output whatever it outputs. Specifically, these hybrid inference protocols use additive sharing to connect two layers and the intermediate results that the corrupted server obtains are random secret shares that can be simulated by picking uniformly random values. Next in step 3 of Π_{Fusion}, the simulator $\mathsf{Sim}_\mathcal{S}$ works as follows.

(I) CS-Case 1: If the adversary does not cheat throughout the protocol, there are two probable results, i.e., (1) the model accuracy fulfills the requirement and the simulator will not send abort to the functionality, and (2) the model accuracy does not meet the requirement and the simulator will send abort to the functionality at the end of the protocol execution. The simulator is given the corrupted party's input [40] and it can check the model accuracy, mirroring the behavior of the client. If the model accuracy does not satisfy the threshold, the simulator will send abort to the functionality while in the real world the client rejects the results and aborts. If the model accuracy satisfies the threshold, the functionality will not abort and sends inference results to the client while in the real world the client accepts the results. The view in both the ideal and the real worlds is identical.

(II) CS-Case 2: If the adversary cheats by using inconsistent model parameters for multiple query samples or deviating from the protocol, the simulator will detect and send abort to the functionality. In the real world, if the server cheats as above, it will be caught with overwhelming probability. The adversary can cheat successfully if all copies of the same query sample obtain incorrect-but-consistent inference results. The *mix-and-check* method ensures that the cheating probability can be negligibly small when appropriately choosing the parameters B and T.

We begin by defining the following *mix-and-check* game, which is equivalent to our protocol Π_{Fusion} (Figure 3). The server wins the game if it chooses some of the query samples and provides incorrect inference results for them without being caught by the client. We need to prove that the probability of the server winning the game is negligible,

that is, the server cannot distinguish every two samples in the mixed dataset. If the game's output is 1, the malicious server wins the game and cheats the client successfully. The probability $\Pr_{success}$ that the server succeeds in cheating equals $\Pr[\text{Game}(\mathcal{S}, \mathcal{C}, R, B, T) = 1]$.

> *Definition 2: The probability that the server $\mathcal{S}$ wins the game is negligible by choosing appropriate R, B, and T.*

The game $\text{Game}(\mathcal{S}, \mathcal{C}, R, B, T)$ proceeds as follows. (i) The client $\mathcal{C}$ prepares the mixed dataset containing $RB+T$ samples and uses them as inputs for privacy-preserving inference computations. (ii) The server $\mathcal{S}$ selects iB samples (i is the number of query samples $\mathcal{S}$ chooses to fool) in the mixed dataset and returns incorrect inference results for each of them. (iii) The inference results for the remaining samples are computed correctly using the high-quality model. The output of the game is 1 if there are i query samples such that inference results for each of them and corresponding copies are incorrect-but-consistent, while inference results for remaining $R - i$ query samples and their copies are obtained by correctly performing the inference computations using the high-quality model.

Claim 1: *If $T \geq B$, then for every adversary $\mathcal{S}$, it holds that*

$$Pr[Game(\mathcal{S}, \mathcal{C}, R, B, T) = 1] \leq R \binom{RB + T}{B}^{-1}.$$

We need to show that for every $1 \leq i \leq R$,

$$\binom{R}{i}\binom{RB + T}{iB}^{-1} \leq R\binom{RB + T}{B}^{-1}. \tag{7}$$

At first, it can be observed that when $i = 1$, the left side of the inequality equals the right side, and thus the equation holds. Next, assume that $i \geq 2$, it suffices to show that:
$$\binom{R}{i}\binom{RB + T}{iB}^{-1} \leq \binom{RB + T}{B}^{-1}.$$

It is equivalent to proving that:
$$\binom{R}{i}\frac{(iB)!(RB + T - iB)!}{(RB + T)!} \leq \frac{B!(RB + T - B)!}{(RB + T)!},$$

which can be represented as:
$$\binom{R}{i}\frac{(iB)!}{B!} \leq \frac{(RB + T - B)!}{(RB + T - iB)!}.$$

By multiplying both sides with $\frac{1}{(iB-B)!}$ the above inequality can be transferred to
$$\binom{R}{i}\binom{iB}{iB - B} \leq \binom{RB + T - B}{iB - B}.$$

Considering the assumption that $T \geq B$ and thus $\binom{RB}{iB - B} \leq \binom{RB + T - B}{iB - B}$, it suffices to prove that
$$\binom{R}{i}\binom{iB}{iB - B} \leq \binom{RB}{iB - B}. \tag{8}$$

To prove that the above Equation 8 holds, we can consider the both sides of the inequality as following: The left side

$\binom{R}{i}\binom{iB}{iB - B}$ can represent the process: choosing i query samples among R query samples, then choosing $iB - B$ samples from iB copies of the selected i query samples, and using false model parameters to provide inference results for the $iB - B$ samples. The right side $\binom{RB}{iB - B}$ can represent the process: choosing $iB - B$ samples from RB samples. The above two processes both end with choosing $iB - B$ samples out of RB samples. Since there is no restriction on the selection of the right process, the number of choices in the right process is strictly larger than that in the left process. It is sufficient to conclude that the inequality Equation 8 holds.

Claim 2: *In real execution, if the parameter B and T are properly chosen as in Π_{Search}, then the client aborts with probability at least $2^{-\lambda}$.*

When B and T are properly chosen as in Π_{Search}, it ensures that $R\binom{RB + T}{B}^{-1} \leq 2^{-\lambda}$, then the probability $\Pr[\text{Game}(\mathcal{S}, \mathcal{C}, R, B, T) = 1] \leq 2^{-\lambda}$. That is, if the server cheats in the real protocol, the client will detect and abort with probability at least $1 - 2^{-\lambda}$, while the simulator will definitely detect the server's cheating behavior and abort in the ideal world. Thus, the view in both the ideal and real worlds is statistically close.

To conclude, in all cases, the view in both the ideal and real worlds is computationally indistinguishable. Thus, the protocol Π_{Fusion} (Figure 3) securely realizes the ideal functionality in the $\mathcal{F}_{\text{PPML}}^{\text{semi}}$-hybrid model against the malicious server. This completes the proof.

VI. EVALUATION

In this section, we test *Fusion* using multiple experiments. In particular, we first explain the experimental setup (§VI-A), followed by the experiment results (§VI-B).

A. Experimental Setup.

Datasets. In our study, we used 7 different datasets, and we now explain each of them in greater details:

- *TIMIT* [15] contains speech data of English recordings from 630 speakers for the development of automatic speech recognition systems.
- *MNIST* [35] is a commonly used dataset that contains (28×28) images of handwritten digits between 0 and 9.
- *CIFAR-10* [33] is a standardized consisting of 60,000 (32×32) color images including 10 classes, e.g., bird, automobile, truck, etc.
- *BC-TCGA* [66] is gene expression data and consists of 17,814 genes and 590 samples (including 61 normal tissue samples and 529 breast cancer tissue samples).
- *GSE2034* [66] is gene expression profiles and includes 12,634 genes and 286 breast cancer samples (including 107 recurrence tumor samples and 179 no recurrence samples).

- *PneumoniaMNIST* [68] contains pediatric chest X-Ray images, and the task is a binary-class classification of pneumonia or normal.
- *DermaMNIST* [68] consists of 10,015 dermatoscopic images of common pigmented skin lesions that are categorized as 7 different diseases.

Environment. We performed experiments on two servers running Ubuntu 16.08 with 2.3 GHz Intel Xeon E5 Broadwell Processors and 244GB RAM. We set the latency as 40 ms which is higher than the latency between two Amazon EC2 machines located in Ohio and Virginia. Other experiments were carried on Intel Xeon CPU E5-2630 v3 @ 2.40GHz with 128GB of RAM and Intel Xeon CPU E5-2680 v4 @ 2.40GHz with 256GB of RAM. The bandwidth between the machines were 382 MBps and 44 MBps, and the echo latency were 0.3 ms and 40 ms in the LAN and the WAN setting respectively.
Availability. Our implementation is publicly available on GitHub: https://github.com/daisy611/Fusion.
Selection of Optimal T. Our *Fusion* requires a number of public samples T to work. Therefore, before testing *Fusion*, we would like to explore an optimized number of public samples. To approximate an optimal T, we then performed experiments: we first denote the model accuracy that is calculated on a large test dataset as *standard accuracy*, and the accuracy obtained on the T public samples as *test accuracy*. We test the *test accuracy* of 50 group non-overlapping and uniformly distributed data on six different datasets (e.g., *MNIST*, *CIFAR-10*, *BC-TCGA*, *GSE2034*, *PneumoniaMNIST*, and *DermaMNIST*) for $T = 80, 85, 90, 95, 100, 105, 110, 115$, and 120. We calculate the variance of *test accuracy* for different numbers of public samples (by setting the *standard accuracy* as its expected value) and show results in Figure 5. The empirical results show that when $T = 100$ the variance is smallest on all the six datasets. When $T = 100$, the *test accuracy* fluctuates around 2% above and below the *standard accuracy* (which is acceptable). Theoretically, based on the analysis in §IV-C, we can notice that T is determined by the *cost requirement* and lower bound of public samples. Since the cost per query sample increases as T increases (when R and B are the same), especially the smaller the R, the larger the cost per query sample. We therefore empirically selected $T = 100$ for trade-offs.
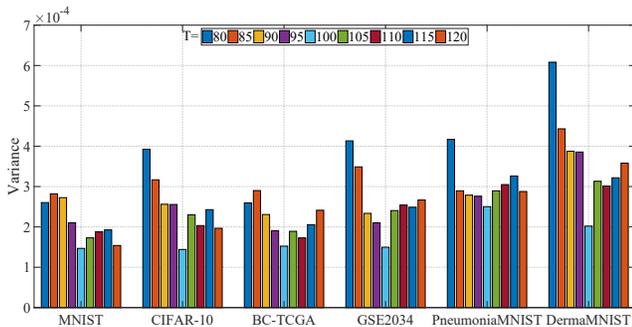
Fig. 5: The variance of *test accuracy* when the numbers of public samples and datasets vary.

B. Experimental Results

Comparison with State-of-the-art *LevioSA*: We compare *Fusion* with the state-of-the-art work maliciously secure neural network inference scheme, i.e., *LevioSA* [18]. To compare *Fusion* with *LevioSA*, we implemented *Fusion* using Cheetah, and adopted the same settings as *LevioSA*: We employed the same neural network framework as *LevioSA*, which consists of a four-layer deep neural network (DNN) with three hidden, fully connected layers with 2000 neurons, quadratic activations, and the final layer is fully connected with 183 output neurons. We trained the DNN model on the *TIMIT* dataset [15] as they used. Since *LevioSA* counted the communication and runtime of inference on 1,845 speech samples, in our experiments, we also set the number of query samples as the same as theirs. When $R = 1,845$, we obtain that optimal parameters $B = 5$ and $T = 100$ (when setting $\beta = 100$) by invoking the search protocol Π_{Search}. Then we let the client prepare the mixed dataset that contains totally $1,845 * 5 + 100$ samples (1,845 query samples (each with 5 copies) and extra 100 speech samples used as public samples). We ran experiments by using the trained DNN model to perform inference computations on the mixed dataset.

The experimental results of communication and runtime between *LevioSA* and *Fusion* are shown in Figure 6. Our results show that *Fusion* has 48.06× less runtime and uses 30.90× less communication compared to *LevioSA*.
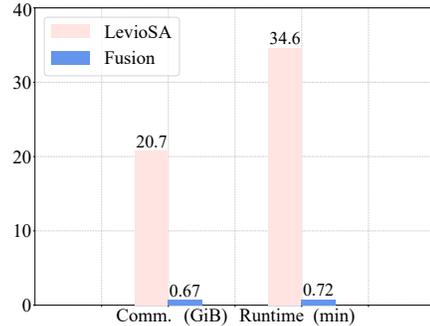
Fig. 6: Comparison between *LevioSA* and *Fusion*.

Evaluation with Different Underlying Building Blocks: The performances of *Fusion* depend on the efficiency of the underlying semi-honest inference protocol (more efficient underlying building block generally improves the performance of *Fusion*), and the choices of R, B, and T (e.g., when T is fixed, the lower B is, the lower the amortized cost of *Fusion* is). As such, to test the performance of *Fusion*, we change the underlying building blocks (e.g., ABY [13], DELPHI [46], CRYPTFLOW2 (SCI$_{HE}$) [59], and Cheetah [21]), and select different R, B, and T. In particular, since the optimal choice of B to minimize amortized cost is influenced by the number of query samples (e.g., see Equation 6), we need to select various R that lead to different B. To that end, for each B ranging from eight to three, we look for the corresponding R and T that satisfy the *security requirement* $\Pr_{success} \leq 2^{-\lambda}$. We set the statistical parameter $\lambda = 40$

and empirically choose the least number of query samples $\beta = 100$. Specifically, for each B from eight to three, we search for the least R, and for simplicity, we select R from 2^x ($x \in \mathbb{Z}$). The search results are shown in Table I. Having decided (R, B, T), we then conducted experiments on two datasets (*MNIST* and *CIFAR-10*) using multiple building blocks. Similarly, we used the four-layer DNN network, and implemented *Fusion* with the settings in [37] to train models on *MNIST* and *CIFAR-10* datasets respectively. We then used those models to perform inference computations in different network settings (i.e., LAN and WAN).

TABLE I: The values of (R, T) with different B.

B	8	7	6	5	4	3
(R, T)	$(2^3, 100)$	$(2^5, 100)$	$(2^7, 100)$	$(2^9, 100)$	$(2^{13}, 100)$	$(2^{19}, 100)$

The total communication and runtime of *Fusion* when using different R (resulting in different B) in the *MNIST* and *CIFAR-10* datasets are shown in Table II and Table III respectively. It can be observed that *Fusion* can achieve the best efficiency when using Cheetah [21] as the building block, while when *Fusion* uses ABY [13] and DELPHI [46], the performance is not as good as that of using others. For example, the Cheetah-based *Fusion* can save around $10\times$ communication, and is roughly $2.5\times$ and $4.8\times$ faster (in the LAN and WAN respectively) compared with those using CRYPTFLOW2 on both the *MNIST* and *CIFAR-10* datasets. When comparing the performance of DELPHI-based *Fusion*, Cheetah-based *Fusion* uses more than $105\times$ less communication and is about $22\times$ faster and $37\times$ faster (in the LAN and WAN settings respectively) on these two datasets.

Comparison with Semi-honest Solutions: We would like to understand whether *Fusion* can be more efficient when compared with semi-honest inference protocols (i.e., ABY [13], DELPHI [46], and CRYPTFLOW2 (SCI$_{\text{HE}}$) [59]). To that end, we conducted experiments using the DNN network on both *MNIST* and *CIFAR-10* datasets and both network settings (the LAN and WAN settings). We change R as 2^3, 2^5, 2^7, 2^9, 2^{13}, and 2^{19}, and the corresponding B for each R ranges from 8 to 3 (with fixed $T = 100$).

In Table IV, we compare the communication and runtime per query sample of *Fusion* with those of ABY [13], DELPHI [46], and CRYPTFLOW2 [59]. Note that the amortized cost of *Fusion* decreases as B decreases, and greater performance improvements will be achieved. The experimental results imply that *Fusion* is more efficient than ABY, DELPHI, and CRYPTFLOW2 in terms of communication when $R \geq 2^5$, $B \leq 7$, and $T = 100$. For instance, when using the *MNIST* dataset and setting $R = 2^5$, $B = 7$, $T = 100$, the communication cost of *Fusion* is $14.13\times$, $10.61\times$, $1.04\times$ lower than those of ABY, DELPHI, CRYPTFLOW2, respectively. When $R \geq 2^5$ and corresponding $B \leq 7$, *Fusion* is faster than ABY and DELPHI in both the LAN and WAN settings. For example, when $R = 2^9$, $B = 5$, and $T = 100$, *Fusion* is about $10.52\times$ and $7.37\times$ faster than ABY and DELPHI respectively on the *CIFAR-10*

dataset in the WAN setting. *Fusion* uses about $2.62\times$ less communication, and is $1.25\times$ faster than CRYPTFLOW2 on *MNIST* dataset in the WAN setting when $R = 2^{13}$ and $B = 4$.

Evaluation on Medical Datasets: Since medical data is highsensitive and has strong privacy preservation requirements, medical data analysis is one of the most important applications of secure inference. To show the applicability of *Fusion* in real-world medical datasets, we evaluated experiments on four publicly available healthcare datasets. We used the same four-layer DNN network with settings [37] to train these models and perform inference. Table V shows the experimental results on four medical datasets *BC-TCGA* [66], *GSE2034* [66], *PneumoniaMNIST* [68], and *DermaMNIST* [68]. Take the *BC-TCGA* dataset as an example, it consists of 17,814 genes (features), and costs 14.579 ms and 0.469 MiB respectively in maliciously secure inference per query sample. As such, the runtime cost and communication cost are acceptable.

Evaluation on Practical DNN ResNet50: To demonstrate the scalability of *Fusion*, we also conducted evaluations of maliciously secure inference on ImageNet-scale deep neural networks (DNN) ResNet50 [19]. We trained a model on ResNet50 using an image dataset [1], and used it to perform inference computation. Since R affects the amortized cost of *Fusion*, we choose R as 2^3, 2^5, 2^7, and 2^9, and the corresponding number of copies for each query sample varies from 8 to 5 (when $T = 100$). We also conducted experiments by using the semi-honest CRYPTFLOW2 protocol and compared it with *Fusion*. Table VI shows the communication and runtime per query sample. It can be observed that when $R \geq 2^5$ and corresponding $B \leq 7$, *Fusion* is more efficient than the semihonest CRYPTFLOW2 (SCI$_{\text{HE}}$) in terms of communication. For example, when $R = 2^9$ and $B = 5$, *Fusion* costs $1.30\times$ runtime and is $1.18\times$ faster in the LAN setting and the WAN setting respectively, and has $2.64\times$ less communication than that of CRYPTFLOW2 (SCI$_{\text{HE}}$).

VII. DISCUSSION

In MLaaS, a semi-honest client who correctly follows the protocol still has some other approaches to steal the server's model, and the black-box model extraction attack [62], [53], [5], [25] is one of them. In this type of attack, the client with black-box access keeps querying the model to extract an equivalent ML model. Since this type of attack is getting more attention recently, and the threat model of this attack is consistent with ours (i.e., the client is semi-honest), in this section, we would like to discuss the scalability of *Fusion* (e.g., how hard or easy to integrate prior defenses into *Fusion*) in defending against model extraction attacks. As such, the researchers who would like to use our *Fusion* and also have the requirements of defending against model extraction attacks can benefit from our paper in a timely manner.

While it is true that there are already numerous defense measures that can work against model extraction attacks (e.g., [26], [65], [67]), intuitively, we can directly use any of those off-the-shelf solutions. However, we would like to note there

TABLE II: Comparison of *Fusion* instantiated using ABY [13], DELPHI [46], CRYPTFLOW2 (SCI$_{HE}$) [59], and Cheetah [21] on the *MNIST* dataset in both LAN and WAN settings. We use Comm. to denote communication and is in MiB, and runtime is in seconds.

# of Samples	ABY *(MNIST)*			DELPHI			CRYPTFLOW2			Cheetah		
	Comm.	LAN	WAN	Comm.	LAN	WAN	Comm.	LAN	WAN	Comm.	LAN	WAN
164 (2^3 * 8 + 100)	27.4949	0.1211	0.3843	20.6144	0.0862	0.2583	1.9984	0.0101	0.0333	0.1914	0.0039	0.0068
324 (2^5 * 7 + 100)	54.4965	0.2407	0.7386	40.7033	0.1711	0.5128	3.9787	0.0201	0.0676	0.3782	0.0073	0.0136
868 (2^7 * 6 + 100)	146.0627	0.6393	1.9728	107.6844	0.4619	1.3667	10.5904	0.0541	0.1808	1.0133	0.0198	0.0363
2,660 (2^9 * 5 + 100)	441.7787	1.9620	5.9044	335.8445	1.4615	4.2086	32.6136	0.1651	0.5529	3.1048	0.0605	0.1105
32,868 (2^{13} * 4 + 100)	5,478.9059	24.3705	73.9667	4,124.8922	18.1847	51.7762	400.9943	2.0339	6.7420	38.3905	0.7429	1.3678
1,572,964 (2^{19} * 3 + 100)	262,641.3367	1,166.8446	3,607.8405	197,253.8050	887.0325	2,475.5588	19,340.8286	98.3081	327.7937	1,832.7539	35.7586	65.8347

TABLE III: Comparison of *Fusion* instantiated using ABY, DELPHI, CRYPTFLOW2, and Cheetah on the *CIFAR-10* dataset.

# of Samples	ABY *(CIFAR-10)*			DELPHI			CRYPTFLOW2			Cheetah		
	Comm.	LAN	WAN	Comm.	LAN	WAN	Comm.	LAN	WAN	Comm.	LAN	WAN
164 (2^3 * 8 + 100)	33.1801	0.1405	0.4155	25.4172	0.1016	0.2982	2.4810	0.0121	0.0387	0.2350	0.0046	0.0078
324 (2^5 * 7 + 100)	65.5365	0.2776	0.8237	50.3487	0.1994	0.5907	4.9215	0.0239	0.0768	0.4637	0.0091	0.0154
868 (2^7 * 6 + 100)	175.8464	0.7456	2.1523	134.9016	0.5372	1.5797	13.1828	0.0639	0.2051	1.2436	0.0243	0.0414
2,660 (2^9 * 5 + 100)	537.8853	2.2732	6.8505	409.4743	1.6361	4.8425	40.0124	0.1961	0.6345	3.8084	0.0745	0.1261
32,868 (2^{13} * 4 + 100)	6,659.7936	28.1152	83.2735	5,140.3375	20.1743	59.4230	497.8308	2.4246	7.7918	47.0841	0.9203	1.5706
1,572,964 (2^{19} * 3 + 100)	318,618.4824	1,337.5954	4,075.8356	245,896.7141	971.4180	2,854.9131	23,768.0474	115.9847	374.3836	2,256.4829	44.1961	75.1037

TABLE IV: Performance of Cheetah-based *Fusion* (with different (R, B, T)), and comparison with semi-honest inference protocols. Comm. is in KiB, and runtime is in µs.

	Scheme	MNIST			CIFAR-10		
		Comm.	LAN	WAN	Comm.	LAN	WAN
Fusion	(2^3, 8, 100)	24.499	487.500	850.000	30.080	575.000	975.000
	(2^5, 7, 100)	12.102	228.125	425.000	14.838	284.375	481.250
	(2^7, 6, 100)	8.106	154.688	283.594	9.949	189.844	323.438
	(2^9, 5, 100)	6.210	118.164	215.820	7.617	145.508	246.289
	(2^{13}, 4, 100)	4.799	90.686	166.968	5.886	112.341	191.724
	(2^{19}, 3, 100)	3.580	68.204	125.570	4.407	84.297	143.249
CRYPTFLOW2 [59]		12.591	62.499	208.392	15.473	73.736	238.012
DELPHI [46]		128.412	563.924	1573.818	160.079	617.572	1814.989
ABY [13]		170.980	741.813	2293.657	207.421	850.366	2591.182

TABLE VI: Performance on ResNet50 and ImageNet-scale benchmarks. It shows communication cost (GiB) and runtime (minutes) per query sample with different (R, B, T).

Scheme	Comm.	LAN	WAN
Fusion (2^3, 8, 100)	39.921	20.410	34.241
Fusion (2^5, 7, 100)	19.714	10.082	16.912
Fusion (2^7, 6, 100)	13.205	6.750	11.326
Fusion (2^9, 5, 100)	10.117	5.173	8.678
CRYPTFLOW2 [59] (SCI$_{HE}$)	26.742	3.988	10.204
CRYPTFLOW2 [59] (SCI$_{OT}$)	281.497	4.795	39.466

TABLE V: Performance on medical datasets. It shows amortized communication (MiB) and runtime (ms) per query sample when $R = 2^{13}$, $B = 4$, and $T = 100$.

Dataset	Comm.	LAN	WAN
BC-TCGA [66]	0.469	8.710	14.579
GSE2034 [66]	0.297	6.628	11.342
PneumoniaMNIST [68]	0.694	26.864	44.931
DermaMNIST [68]	0.034	2.353	3.928

should be at least two criteria that need to be fulfilled. First, *Fusion* can verify model accuracy (R1), computation correctness (R2), and preserve privacy (R3). The integrated solution should not introduce weaknesses to break those guarantees. Second, *Fusion* itself is cryptographic-friendly, and as such, we should not introduce heavy cryptographic-based solutions for performance concerns. Fortunately, we adopt a passive defense method [37] that satisfies all those design criteria by

perturbing the model's activation layer to change the output probabilities. As the solution directly makes changes to the neural network, it will not introduce extra overhead on *Fusion*, and allows *Fusion* to achieve the three requirements even when the solution is adopted. Consequently, by integrating the solution, *Fusion* is defense-effective against the model extraction attacks (e.g., the accuracy of the client's stolen model drops by up to 42.7% compared with when not using defense) meanwhile maintaining the *utility* (e.g., reducing the model accuracy with no defense by 1.75%). Moreover, the presence of defense only causes low extra costs, accounting for less than 1% of the overall runtime and communication. We have detailed design and evaluation in Appendix §A for readers of interest.

VIII. RELATED WORK

This paper studies secure inference protocols. We categorize existing works into outsourcing scenarios (§VIII-A) and non-outsourcing scenarios (§VIII-B). Since *Fusion* works in non-outsourcing scenarios, we particularly focus on the later case.

A. Secure Inference Protocols in Outsourcing Scenarios

In an outsourcing scenario, some servers are responsible for performing the secure inference protocol, and the client does not participate in the inference computations. Typically, the client provides query samples to and receives inference results from the servers such that the servers cannot know both the client's inputs and inference results. Based on the adopted threat model, the secure inference protocols in the outsourcing scenario can further have two types:

Semi-honest Security: ABY 1.0&2.0 [13], [55] are mixed 2PC protocols that consider the semi-honest security and provide secure conversions between Arithmetic (A), Boolean, and Yao (Y) sharing. Some schemes [28], [7] adopt HE to perform secure outsourced neural network inference and make efforts to improve the performance.

Malicious Security in Honest Majority Setting: Some schemes [47], [57], [56], [11], [32] achieve honest-majority malicious security (i.e., the number of corrupted parties less than half of the total number of computing parties involved in MPC). These protocols need three [47], [57], [56], [32] or four [11] non-colluding servers to perform secure inference protocols. The aim of these schemes is to achieve better efficiency by various means (e.g., by enabling the communication of dot products in the online phase is independent of the vector size [56]) or achieve stronger security (e.g., output delivery guarantee by using the property of honest-majority setting [32], [11]).

The major difference between *Fusion* and those schemes is the scenarios: those schemes all work in the outsourcing scenario, while *Fusion* work in the non-outsourcing scenario. Moreover, our *Fusion* additionally supports a stronger security level (i.e., malicious security in the dishonest majority). Moreover, our scheme can ensure the computation correctness and verification of model accuracy in the presence of a malicious server.

B. Secure Inference Protocols in Non-outsourcing Scenarios

In the non-outsourcing scenario, the client participates in performing the 2PC-based secure inference protocol by interacting with the server and finally obtains the inference results. Similarly, there are also two types correspondingly:

Semi-honest Security: Due to the efficiency bottleneck, most of the existing secure inference protocols [43], [30], [7], [60], [46], [59], [58] consider semi-honest security (e.g., assuming both the server and client honestly follow the protocol), and focus on improving efficiency to enable practical applications. Specifically, many solutions use a hybrid cryptographic protocol (e.g., adopting different cryptographic techniques for linear and non-linear layers respectively) to achieve better efficiency.

Malicious Security: Recently, some works begin to recognize the significance of security against a malicious server in MLaaS, and try to present solutions for achieving computation correctness in the presence of a malicious adversary.

TABLE VII: Comparison of *Fusion* with others. ◗ represents "semi-honest". ☆ represents "malicious security with honest majority". ★ represents "malicious security with dishonest majority".

Schemes	Threat Model	Methodology	No Extra Hardware	Non-outsourcing?	R1.Model Accuracy	R2.Correctness	R3. Both Privacy
Jiang et al. [28]	◗	HE	✓	✗	✗	✓	✓
Chen et.al. [7]	◗	HE	✓	✗	✗	✓	✓
ABY [13]	◗	SS+GC	✓	✗	✗	✓	✓
ABY2.0 [55]	◗	SS	✓	✗	✗	✓	✓
ABY3 [47]	☆	SS+GC	✓	✗	✗	✓	✓
Trident [57]	☆	SS	✓	✗	✗	✓	✓
Blaze [56]	☆	SS	✓	✗	✗	✓	✓
FantasticFour [11]	☆	SS	✓	✗	✗	✓	✓
SWIFT [32]	☆	SS	✓	✗	✗	✓	✓
MiniONN [43]	◗	HE+GC+SS	✓	✓	✗	✓	✓
GAZELLE [30]	◗	HE+GC+SS	✓	✓	✗	✓	✓
XONN [60]	◗	GC+SS	✓	✓	✗	✓	✓
DELPHI [46]	◗	HE+GC+SS	✓	✓	✗	✓	✓
SiRNN [58]	◗	SS	✓	✓	✗	✓	✓
CRYPTFLOW2 [59]	◗	HE+SS+OT	✓	✓	✗	✓	✓
Cheetah [21]	◗	HE+SS+VOLE	✓	✓	✗	✓	✓
Veriml [70]	★	ZK	✓	✓	✓	✓	✗
zkcnn [44]	★	ZK	✓	✓	✓	✓	✗
Mystique [64]	★	ZK	✓	✓	✓	✓	✗
CRYPTFLOW [34]	★	SS	✗	✗	✗	✓	✓
LevioSA [18]	★	SS	✓	✓	✗	✓	✓
Fusion	★	Mix&Check	✓	✓	✓	✓	✓

- **ZK Proof-based Protocols:** There are some schemes [70], [44], [64] that adopt zero-knowledge proofs to achieve verifiable inference computations. Verifiable inference services based on ZK proofs allow one party with a secret witness to prove some statement (e.g., the inference results are correctly produced by the expected model) without revealing private information. Schemes based on ZK proofs have the advantage of being publicly verifiable. It usually assumes, however, that the client's input is visible to the server. As a consequence, it is unable to apply to scenarios where both model and client's inputs are required to be protected, while our scheme preserves the privacy of both the server and client's inputs.

- **TEE-based Protocols:** CRYPTFLOW [34] presents a novel technique, called Aramis, that takes any semi-honest secure MPC protocol for computation and converts it into a malicious secure MPC protocol by using hardware Intel SGX. Our scheme is a purely cryptographic protocol and does not require the trust in hardware.

- **MPC-based Protocols:** If there exists a malicious party in 2PC computation, it definitely has to achieve malicious security with the dishonest majority (meaning the number of corrupted parties is greater than or equal to half of the total number of parties involved in MPC). Naturally, MPC protocols that consider malicious security in the dishonest majority are relatively complicated problems compared with those achieve malicious security in the honest majority setting or semi-honest security. Since the malicious security in the honest majority setting

(a weaker security model) naturally bring dramatic performance improvements and is generally used in an outsourcing scenario, it is unfair to compare them [57], [56], [11], [32] with *Fusion* which achieves malicious security in the dishonest majority.

As shown in Table VII, our scheme is the only scheme that satisfies all the three requirements (i.e., R1, R2 and R3) that proposed in this paper at the same time, while other works have their own different focuses. In particular, our work considers the verification of the inputs (R1), and currently, only a few works have considered it. When compared with those works that fulfilled R1, our work does not use ZK proofs, which is more computational friendly. We would like to note that the most similar work to ours is *LevioSA* [18] which achieves maliciously secure 2PC arithmetic computation in the dishonest majority setting and applies it to neural network classification. It proposes a passive-to-active oblivious linear function evaluation (OLE) compiler by following the high-level approach of IPS compiler [24], [41]. Different from their work, our *Fusion* additionally achieves the verification of model accuracy.

IX. Conclusion

We propose *Fusion*, which can achieve security requirements including privacy-preserving, verification of model accuracy and computation correctness. *Fusion* can be used as a general compiler by converting a semi-honest inference protocol (e.g., Cheetah, DELPHI) into a maliciously secure one and thus can benefit from the state-of-the-art efficient 2PC-based inference scheme. Our evaluation shows that *Fusion* is 48.06× faster and has 30.90× less communication than *LevioSA* (the state-of-the-art maliciously secure inference protocol). Moreover, evaluation on the practical ImageNet-scale ResNet50 model indicates that *Fusion* can be more efficient than semi-honest inference protocols.

Acknowledgment

References

[1] Imagenet-sample-images. https://github.com/EliSchwartz/imagenet-sample-images. 2020. 1000 images, one random image per image-net class. For easy visualization/exploration of classes.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[3] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: two-party secure neural network training and prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*, pages 1231–1247. ACM, 2019.

[4] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.

[5] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020*, pages 189–218. Springer, 2020.

[6] Stanford AIMI Center. Center for artificial intelligence in medicine & imaging shared datasets. https://aimi.stanford.edu/shared-datasets.

[7] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 395–412, 2019.

[8] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) lwe ciphertexts. In *International Conference on Applied Cryptography and Network Security*, pages 460–479, 2021.

[9] Zelei Cheng, Zuotian Li, Jiwei Zhang, and Shuhan Zhang. Differentially private machine learning model against model extraction attack. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 722–728. IEEE, 2020.

[10] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*, pages 613–627. USENIX Association, 2017.

[11] Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*, volume 7417, pages 643–662. Springer, 2012.

[13] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[14] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[15] MICHAEL FEKADU. Darpa timit acoustic-phonetic continuous speech. https://www.kaggle.com/datasets/mfekadu/darpa-timit-acousticphonetic-continuous-speech/metadata. Accessed 2019.

[16] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. Technical report, Cryptology ePrint Archive, Report 2021/087, 2021.

[17] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4672–4681, 2017.

[18] Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. Leviosa: Lightweight secure arithmetic computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 327–344, 2019.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[20] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. Privacy-preserving machine learning as a service. *Proc. Priv. Enhancing Technol.*, 2018(3):123–142, 2018.

[21] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *31th USENIX Security Symposium (USENIX Security 22)*, 2022.

[22] National Cancer Institute. The cancer genome atlas program. https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga.

[23] National Cancer Institute. The cancer genome atlas program gdc data portal. https://portal.gdc.cancer.gov/projects/TCGA-BRCA.

[24] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *Theory of Cryptography Conference*, pages 294–314. Springer, 2009.

[25] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium, USENIX Security 2020*, pages 1345–1362. USENIX Association, 2020.

[26] Hengrui Jia, Christopher A Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1937–1954, 2021.

[27] Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *30th USENIX Security Symposium, USENIX Security 2021*, pages 1937–1954. USENIX Association, 2021.

[28] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1209–1222, 2018.

[29] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N. Asokan. PRADA: protecting against DNN model stealing attacks. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019*, pages 512–527. IEEE, 2019.

[30] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.

[31] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.

[32] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. {SWIFT}: Super-fast and robust privacy-preserving machine learning. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[33] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. http://www.cs.toronto.edu/~kriz/cifar.html.

[34] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 336–353. IEEE, 2020.

[35] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/. Accessed September 24, 2017.

[36] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vcnn: Verifiable convolutional neural network based on zk-snarks. Technical report, Cryptology ePrint Archive, Report 2020/584. https://eprint. iacr. org/2020/584, 2020.

[37] Taesung Lee, Benjamin Edwards, Ian M. Molloy, and Dong Su. Defending against neural network model stealing attacks using deceptive perturbations. In *2019 IEEE Security and Privacy Workshops, SP Workshops 2019*, pages 43–49. IEEE, 2019.

[38] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In *30th USENIX Security Symposium, USENIX Security 2021*, pages 2201–2218. USENIX Association, 2021.

[39] Mingchao Li, Yerui Chen, Keren Xie, Songtao Yuan, and Qiang Chen. Octa-500. https://ieee-dataport.org/open-access/octa-500, 2020.

[40] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[41] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The ips compiler: Optimizations, variants and concrete efficiency. In *Annual Cryptology Conference*, pages 259–276. Springer, 2011.

[42] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 52–78. Springer, 2007.

[43] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.

[44] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC conference on Computer & communications security*, pages 1–13, 2021.

[45] Microsoft. Bing-covid-19-data. https://github.com/microsoft/Bing-COVID-19-Data, 2019.

[46] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522, 2020.

[47] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52, 2018.

[48] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[49] National Library of Medicine. National cancer for biotechnology information. https://www.ncbi.nlm.nih.gov/guide/genes-expression/.

[50] National Library of Medicine. National cancer for biotechnology information. https://www.ncbi.nlm.nih.gov/gene.

[51] National Library of Medicine. National cancer for biotechnology information. https://www.ncbi.nlm.nih.gov/gds.

[52] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Prediction poisoning: Towards defenses against DNN model stealing attacks. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, 2020.

[53] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish K. Shevade, and Vinod Ganapathy. Activethief: Model extraction using active learning and unannotated public data. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 865–872. AAAI Press, 2020.

[54] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017*, pages 506–519. ACM, 2017.

[55] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. Aby2. 0: Improved mixed-protocol secure two-party computation. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[56] Arpita Patra and Ajith Suresh. Blaze: blazing fast privacy-preserving machine learning. In *NDSS*, 2020.

[57] Rahul Rachuri and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. In *NDSS*, 2020.

[58] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure rnn inference. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1–18. IEEE, 2021.

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

[60] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. {XONN}: Xnor-based oblivious deep neural network inference. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1501–1518, 2019.

[61] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. Mlaas: Machine learning as a service. In *2015 IEEE 14th International*

Conference on Machine Learning and Applications (ICMLA), pages 896–902. IEEE, 2015.

[62] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium, USENIX Security 16*, pages 601–618. USENIX Association, 2016.

[63] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Doug Burdick, Darrin Eide, Kathryn Funk, Yannis Katsis, Rodney Michael Kinney, Yunyao Li, Ziyang Liu, William Merrill, Paul Mooney, Dewey A. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Nancy Xin Ru Wang, Christopher Wilhelm, Boya Xie, Douglas M. Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. CORD-19: The COVID-19 open research dataset. In *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*, Online, July 2020. Association for Computational Linguistics.

[64] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 501–518, 2021.

[65] Xun Xian, Mingyi Hong, and Jie Ding. A framework for understanding model extraction attack and defense. *arXiv preprint arXiv:2206.11480*, 2022.

[66] Haozhe Xie, Jie Li, Tim Jatkoe, and Christos (2017) Hatzis. Gene expression profiles of breast cancer. *Mendeley Data*, 2017.

[67] Haonan Yan, Xiaoguang Li, Hui Li, Jiamin Li, Wenhai Sun, and Fenghua Li. Monitoring-based differential privacy mechanism against query flooding-based model extraction attack. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[68] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification. *arXiv preprint arXiv:2110.14795*, 2021.

[69] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[70] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems*, 32(10):2524–2540, 2021.

APPENDIX A

A. Defense against Model Extraction Attacks

Existing defense strategies [29], [27], [37], [9], [52], [38] can be divided into two types, i.e., detecting anomalous queries or analyzing query patterns [29], or using perturbation to make it resilient to the model extraction attacks. The former type makes strong assumptions on the attacker's query distributions and requires query pattern analysis on the client's query samples. Since the query samples and the output labels (inference results) are private to the server, it is tricky to defend against the client by detecting the client's query patterns [52] or adding perturbation according to specific inference results [37]. The latter type can be implemented by the server before the inference service, allowing it to collaborate with privacy-preserving inference protocols without compromising the client's input privacy. These strategies, on the other hand, mitigate model extraction attacks by sacrificing model accuracy, necessitating a trade-off between model utility and defense effectiveness.

Since the query samples and the final inference results are protected from the server in secure inference, the passive defense methods, without requiring additional knowledge about the client's query samples or output labels, are well-compatible with our scheme. We adopt a passive defense method proposed

in [37] because it mitigates the model extraction attacks by perturbing the server's model network. As the attacker essentially approximates the loss hypersurface to find parameters for the stolen model with the minimum loss value, the defense method leverages a new activation layer that manipulates the estimated loss surface by adding a small controllable perturbation and thus maximizes the loss of the stolen model while preserving the accuracy of the original model. Specifically, this defense method only perturbs the model's final activation layer (e.g., softmax) that maps a vector to a number of probability values.

The *Reverse Sigmoid* perturbation $r(y_j^i)$ is as follows.

$$r(y_j^i) \approx \beta(s(\gamma s^{-1}(y_j^i)) - 1/2), \qquad (9)$$

where y_j^i represents j-th dimension of probability vector y^i for sample x_i, $s(\cdot)$ is a sigmoid function, γ is a positive dataset and model specific convergence parameter, and β is a positive magnitude parameter. The reverse sigmoid perturbation only replaces the final layer. Then the final perturbed probability value is calculated as follows.

$$\hat{y}_j^i = \alpha^i(y_j^i - \beta(s(\gamma s^{-1}(y_j^i)) - 1/2)) \qquad (10)$$

where α^i is a sum-to-1 normalizer for y^i.

B. Security Requirement against Model Extraction Attacks

The goals of model extraction attacks can be divided into two objectives, i.e., accuracy, which measures the extracted model's correctness on test samples, and fidelity, which measures the agreement between the extracted model and original model on any point. We measure the goal of the client's attack by accuracy because it is natural that the clients steal the model for use. Informally, the goal of the defense against the client's model extraction attacks is to reduce the accuracy of the stolen model while maintaining the utility of the server's defense model. Formally, there are two metrics [52] to measure the effectiveness of the defense.

- **Non-replicability.** The *non-replicability* is measured in terms of the accuracy of the client's stolen model. That is, the accuracy of the client's stolen model should be far lower than that of the server's model.
- **Utility.** The *utility* is measured in terms of the accuracy of the server's model. The defense method should have little impact on the server's model to maintain its utility.

To mitigate the model extraction attacks, the server locally trains a defense model and adds perturbations before the secure inference service starts. This method is well compatible with the privacy-preserving inference protocol Π_{Fusion} and can be locally implemented by the server. The server's model with defense is trained using the specific strategy described in §A-A. The server uses the defense model to provide inference service while mitigating the client's model extraction attacks.

C. Evaluation

Experiment Setup: Since the *Fusion* integrated with a defense method can cause extra costs, we conducted experiments to test how much costs the defense will bring compared with those without defense. Specifically, we
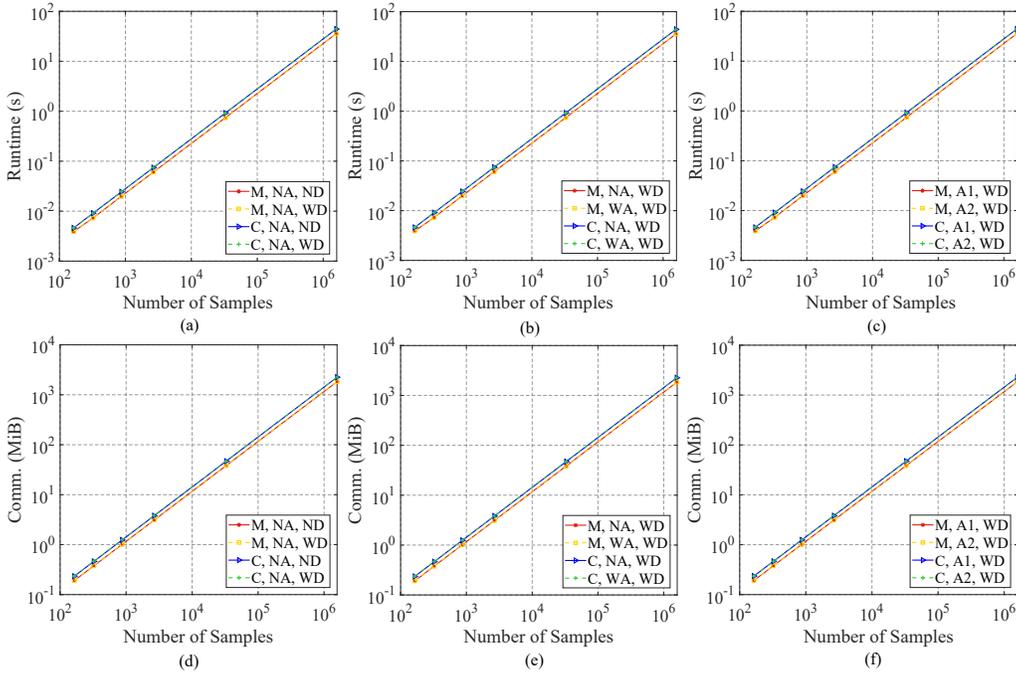
Fig. 7: The impacts of different settings on the performance is depicted in this figure. M and C represent *MNIST* and *CIFAR-10* datasets, respectively. NA denotes no attack while WA denotes with attack. Similarly, ND and WD represent no defense and with defense respectively. A1 and A2 represent attack 1 and attack 2, respectively.

modified the DNN network (in §VI-A) by integrating it with the defense method (§A-A) and used it to train models (with defense ability) on both two datasets (*MNIST* and *CIFAR-10* datasets) respectively. Similarly, we also used the same network without being changed (namely no defense ability) to train models on the two datasets. Then we ran experiments by using these trained models and Cheetah as the underlying building block under different settings. To compare the performance of *Fusion* thoroughly, we ran experiments with extensive settings, i.e., the presence of the server' defense, the presence of the client's attack, as well as different attack methods (attack 1 [54] and attack 2 [5]). We change R as 2^3, 2^5, 2^7, 2^9, 2^{13}, and 2^{19}, and the corresponding B for each R ranges from 8 to 3 (with fixed $T = 100$).

(I) The Integrated Defense Has Negligible Impacts on *Fusion***.** Figure 7 shows the communication and runtime under different settings. First, we compare the performances of *Fusion* with defense (e.g., using the *defense models*, denoted as WD) with that without defense (e.g., using the trained models with no defense, denoted as ND) to measure the ***impact of the existence of defense.*** As Figure 7. (a) and (d) show, the existence of defense brings very small extra costs when compared to without defense. For example, when the total number of samples is 1,572,964, the *Fusion* with defense increases only 3.956/8.488 MiB communication and 0.0007/0.0006 s runtime overhead compared to that without defense using dataset *MNIST* and *CIFAR-10* respectively, which is negligible. The reason for this is that the defense method only changes the final activation layer and adds very little computation

compared to the overall computation. Second, we compare the performances of *Fusion* when there exist attacks (denoted as WA) with when there is no attack (NA) to measure the ***impact of the existence of attacks.*** As Figure 7. (b) and (e) show, the existence of the client's attacks does not affect the running time and communication. Similarly, when the client uses different attack approaches (e.g., attack 1 (A1) [54], and attack 2 (A2) [5]), as shown in Figure 7. (b) and (e), the running time and communication remain nearly the same. The reason for this is that the client only exploits the attack after receiving the inference results, so it has no effect on the privacy-preserving inference computations.

(II) *Fusion* **Is Defense-Effective against Model Extraction Attacks.** We would like to test the defense effectiveness of *Fusion* against model extraction attacks. We conducted experiments by letting the server use the models with defense (WD) and no defense (ND), and the client perform two attack approaches (i.e., attack 1 (A1) [54] and attack 2 (A2) [5]) respectively when the total numbers of query samples vary. To test the *non-replicability* of *Fusion*, we tested the model accuracy of the client's stolen models.

As shown in Figure 8, the accuracy of the client's stolen models significantly decreases when the server used defense compared with when there is no defense, indicating the *non-replicability* of *Fusion*. In other words, *Fusion*'s defense is effective on both two datasets and when the client performed two different attacks. When using the *MNIST* dataset and the numbers of total samples vary, if the server used the models with defense, the accuracy of the client's stolen models de-
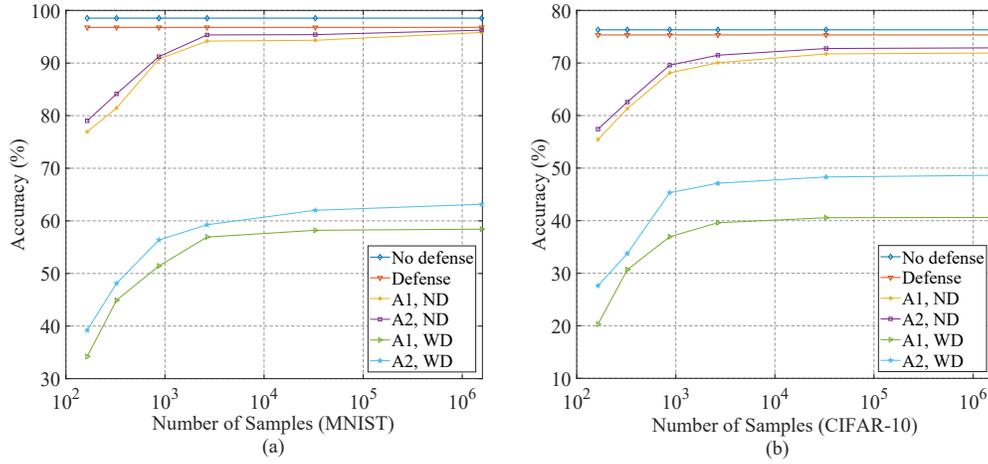
Fig. 8: Accuracy of the client's stolen models when the server uses models with defense and with no defense, the client uses different attack methods (A1 and A2), and the numbers of samples vary.

creases by 36.1-42.7% (A1), while the accuracy decreases by 33.1-39.8% (A2). For instance, when the number of samples is 1,572,964, the accuracy of the client's stolen model drops from 95.81% (A1, ND), 96.25% (A2, ND) to 58.42% (A1, WD), 63.15% (A2, WD), respectively. When using the *CIFAR-10* dataset and the total number of samples is 1,572,964, for example, the accuracy of the client's stolen model drops from 71.9% (A1, ND), 72.88% (A2, ND) to 40.62% (A1, WD), 48.63% (A2, WD), respectively.

(III) *Fusion* **Maintains** *Utility* **When Integrated with the Defense.** To evaluate the *utility* of the server's model with defense, we tested the model accuracy of the models with defense and with no defense on two datasets respectively. As Figure 8 depicts, *Fusion* maintains the *utility*, e.g., the accuracy of the server's models with defense (labelled as Defense in Figure 8) slightly decreases when compared with those with no defense (No defense). For instance, when using the *MNIST* and *CIFAR-10* datasets, the model accuracy of the models with defense is 96.78% and 75.34%, while that with no defense is 98.53% and 76.31%, respectively.