Characterizing and Optimizing End-to-End Systems for Private Inference

Karthik Garimella
kg2383@nyu.edu
New York University
New York, New York, USA

Zahra Ghodsi
zahra@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Nandan Kumar Jha
nj2049@nyu.edu
New York University
New York, New York, USA

Siddharth Garg
sg175@nyu.edu
New York University
New York, New York, USA

Brandon Reagen
bjr5@nyu.edu
New York University
New York, New York, USA

ABSTRACT

In two-party machine learning prediction services, the client's goal is to query a remote server's trained machine learning model to perform neural network inference in some application domain. However, sensitive information can be obtained during this process by either the client or the server, leading to potential collection, unauthorized secondary use, and inappropriate access to personal information. These security concerns have given rise to Private Inference (PI), in which both the client's personal data and the server's trained model are kept confidential. State-of-the-art PI protocols consist of a pre-processing or offline phase and an online phase that combine several cryptographic primitives: Homomorphic Encryption (HE), Secret Sharing (SS), Garbled Circuits (GC), and Oblivious Transfer (OT). Despite the need and recent performance improvements, PI remains largely arcane today and is too slow for practical use.

This paper addresses PI's shortcomings with a detailed characterization of a standard high-performance protocol to build foundational knowledge and intuition in the systems community. Our characterization pinpoints all sources of inefficiency – compute, communication, and storage. In contrast to prior work, we consider inference request arrival rates rather than studying individual inferences in isolation and we find that the pre-processing phase cannot be ignored and is often incurred online as there is insufficient downtime to hide pre-compute latency. Finally, we leverage insights from our characterization and propose three optimizations to address the storage (Client-Garbler), computation (layer-parallel HE), and communication (wireless slot allocation) overheads. Compared to the state-of-the-art PI protocol, these optimizations provide a total PI speedup of $1.8 \times$ with the ability to sustain inference requests up to a $2.24 \times$ greater rate. Looking ahead, we conclude our paper with an analysis of future research innovations and their effects and improvements on PI latency.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; **Distributed systems security**.

KEYWORDS

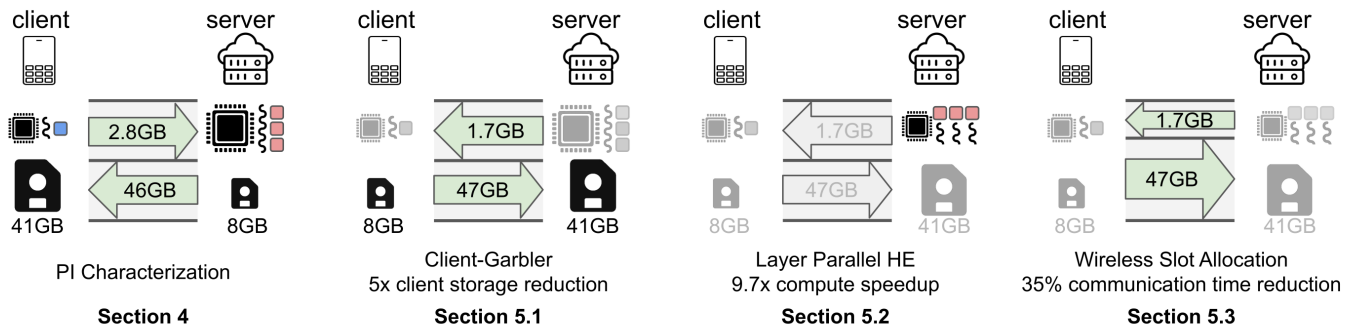machine learning, cryptography, private inference protocols, systems for machine learning

1 INTRODUCTION

Privacy continues to increase in significance as users steadily demand more protection and control over how and when their data is used. Addressing these concerns will have a profound effect on how many popular cloud services are implemented, which rely heavily on precise user data for high-quality experiences. One solution is to leverage privacy-preserving computation to guarantee the privacy and security of user data *during* computation, effectively extending the guarantees of today's methods that only protect communication (and possibly storage) to the entire execution pipeline. Privacy-preserving computation provides the best of both worlds for the user: privacy and security are significantly improved while retaining the ability to use online services now integral to everyday life.

AI is a natural starting point for privacy-preserving computing. These workloads make heavy use of the client-cloud model and require access to user data to be successful. Moreover, AI accounts for a significant amount of all private user data processing. E.g., Meta processes over 200 trillion inferences per day [52]. The ability to process certain essential functions in private (such as convolutions, ReLU, and fully-connected layers) could safeguard a disproportionate amount of users' data and would mark a significant achievement in privacy-preserving computation. Privacy-preserving computation is still a developing field, and as we will show inference is still a ways off from being practical. We focus on private inference (PI) and leave training to future work.

Figure 1: Overview of the paper's organization and contributions in reducing overheads of hybrid PI protocols. Each image highlights a contribution and summarizes the key insight or result.

The solution space of privacy-preserving techniques is vast, and each has its strengths and weaknesses. One way to classify techniques is in how they achieve security. Whether by: system implementation (e.g., SGX [3]), statistics (e.g., differential privacy [2, 21]), or cryptography (e.g., homomorphic encryption [11, 14, 25, 28–30, 69] and multi-party computation [6, 7, 24, 34, 42, 65, 74]). Systems solutions offer the best performance but are also less secure because they assume hardware trust and are vulnerable to attack [80]. Statistical privacy is stronger and provides quantifiable guarantees, e.g., ϵ-DP [22]. However, these methods add noise to the computation, which can result in degraded accuracy, and DP has been most successful in answering questions in aggregate. Their use in computing on individual user data, e.g., a single inference, is unclear. Cryptography-based solutions compute exact inferences with strong security guarantees (e.g., 128-bit) but come at the expense of high computation, communication, and storage overhead. Homomorphic encryption (HE) and secure multi-party computation (MPC), including secret sharing (SS) and garbled circuits (GCs), are the two leading methods for cryptographically-secure computation and the focus of this paper.

There now exists a large body of research on private inference using both HE and MPC. At a high-level, many protocols share a common, hybrid approach that tailors cryptographic primitive selection to the computational needs of each network layer. Most leverage homomorphic encryption and/or secret sharing for arithmetic circuits (i.e., convolutions and fully-connected layers) and multi-party computation for boolean circuits such as ReLUs [16, 17, 43, 55, 57]. (Non-hybrid approaches have been proposed but typically sacrifice accuracy, see Section 7.) Recent work has sought to further improve hybrid protocols with better neural architecture design [16, 32], protocol optimization [17, 43, 55, 57], and hardware acceleration [66, 71, 72]. These optimizations have largely focused on reducing specific components of the overall PI protocol, and it is unclear how they combine to improve the overall goal of fast end-to-end private inference.

This paper provides the first characterization of end-to-end private inference using state-of-the-art hybrid protocols. The characterization encompasses both the online and offline phases, accounting for compute, communication, and storage overheads. We are also the first to consider inference request arrival rates (workloads of inferences) in PI; all known prior work focuses on individual inferences in isolation. We find that this is a vital distinction as the offline costs are so large that they do not always remain offline, even at low arrival rates.

Our characterization provides the following insights:

- The computation time of the offline phase cannot be ignored when processing PI requests. It involves (HE and GC) computations that run on the order of minutes.
- Hybrid protocols incur large offline storage costs (e.g., tens of GBs). These must be stored on the client device, limiting the number of PI pre-computations that can be buffered and quickly overwhelm storage constraints.
- ReLUs introduce both large communication and computation overheads. For example, ReLU transmission and computation times for TinyImageNet using ResNet-18 can take up to 11 and 21 minutes, respectively.

Leveraging our insights, we propose three optimizations to improve inference latency that address each of the system limitations: compute, communication, and storage. An overview is shown in Figure 1. First, to overcome the limits of client storage, we propose a protocol optimization that we refer to as the **Client-Garbler** protocol. This reverses the client and cloud roles in GC, enabling pre-computed GCs to be stored on the server, which we assume has significantly more storage than a client's smartphone. Second, we identify a form of compute parallelism termed **layer-parallel HE** (LPHE). The observation is that each neural network layer's offline HE computation is independent and can be run completely in parallel. LPHE significantly reduces the offline computational costs of hybrid protocols, reducing their impact on inference latency even when they are incurred online. Furthermore, LPHE implies that hybrid solutions using a combination of protocols (i.e., SS+HE) for linear layers can be faster than single-primitive solutions (i.e., HE) even though hybrid protocols entail more total computation. This is because LPHE provides an opportunity to run all offline HE layer computations in an embarrassingly parallel fashion. Finally, we show how the default provisioning of wireless bandwidth between upload and download is sub-optimal for PI and propose **wireless slot allocation** (WSA). WSA reduces communication time by optimally dividing wireless bandwidth between upload and download based on the protocol's needs. Combined, the proposed optimizations improve the mean inference latency by 1.8× over the state of the art [57].

This paper makes the following contributions:

(1) The first end-to-end characterization of private inference using arrival rates. Our analysis reveals the key sources of inefficiency with respect to storage, communication, and computation.

(2) A protocol optimization (Client-Garbler) to reduce the significant storage pressure on the client by 5×.

(3) Identifying a new form of parallelism (LPHE) that enables each HE linear layer computation to run simultaneously, significantly improving the performance of one of the slowest legs of PI by 9.7×. LPHE is favorable when compared to Request-Level Parallelism when client storage is limited (e.g., 32 GB and less).

(4) Wireless bandwidth slot allocation (WSA) to reduce communication latency by up to 35%.

We conclude with a discussion of our results and their implications in Section 6, including the tradeoffs of different PI protocols with respect to systems and by estimating PI performance as advances to each part of the protocol are inevitably made, e.g., from HE and GC accelerators. Our discussion points to open problems that require further research attention.

2 CRYPTOGRAPHIC PRIMITIVES AND PI

This section provides a primer on the cryptographic primitives used to achieve private inference: homomorphic encryption (HE), secret sharing (SS), garbled circuits (GC), and oblivious transfer (OT). We provide technical details and highlight their system implications. We then outline the entire process taken to achieve high-performance private inference, including all phases of execution. The threat model assumed is consistent with other work: semi-honest, two-party computation with no hardware and software trust.

2.1 Cryptographic Primitives

2.1.1 Homomorphic Encryption. HE is a type of encryption where encrypted data is malleable. This enables functions to be computed directly on ciphertexts, preserving data confidentiality during computation.

Details: HE involves three steps. First, the client encrypts data x using an encryption function E and its public key k_{pub}, i.e., $c = E(x, k_{pub})$ (or just $c = E(x)$ for short). Then, given two ciphertexts $c_1 = E(x_1)$ and $c_2 = E(x_2)$, the server homomorphically computes $x_1 \odot x_2$ using a function $\star$ that operates directly on ciphertexts, i.e., $c' = c_1 \star c_2$. The client can then decrypt c' using a decryption function D and secret key k_{sec} to obtain $x_1 \odot x_2 = D(c', k_{sec})$ (or just $D(c')$ for short).

Systems implications: HE introduces a large computational overhead of 4-6 orders of magnitude slowdown [66, 71, 72]. However, its storage and communication costs are minimal.

2.1.2 Additive Secret Sharing. Additive secret sharing (SS) allows a value x to be shared amongst two (or more) parties such that neither party learns anything about x from its share. The parties can compute arithmetic functions of secret shared values to obtain shares of the result. These can then be combined to reveal the result.

Details: The shares of a value x are $\langle x \rangle_1 = r$ and $\langle x \rangle_2 = x - r$, where r is a random value and the arithmetic operations are modulo a prime p. Since $x = \langle x \rangle_1 + \langle x \rangle_2$, the value x can be recovered if the two parties add their shares together. However, since r is random, neither party has enough information to reconstruct x by itself. Given shares of two values x and y, shares of $x + y$ can be computed by as follows: $\langle x + y \rangle_1 = \langle x \rangle_1 + \langle y \rangle_1$, and $\langle x + y \rangle_2 = \langle x \rangle_2 + \langle y \rangle_2$. That is, the shares of a sum of two values equal the sum of the shares. Multiplying two secret shared values is more challenging, however. One solution is to generate so-called "Beaver triples" in a pre-processing phase. Beaver triples are secret shares of values a, b and c where a and b are random and unknown to the parties, and $c = a \odot b$. Beaver triples can be generated using offline HE and leveraged online for fast private multiplications; see [57] for more details.

Systems implication: Additions in SS add minimal storage and communication overheads compared to plaintext computation. However, multiplications are costlier and require expensive HE computations in a pre-processing phase. Hybrid protocols leverage offline HE followed by online SS for linear layer computations, thus incurring large offline costs that cannot always be hidden.

2.1.3 Garbled Circuits. The protocols described so far enable private arithmetic computations; additions and multiplications over integer values. GCs enable two parties to compute a *Boolean* function on their private inputs without revealing their inputs to each other. Unlike SS and HE, operating over Boolean gates enables the parties to jointly compute any function. (We note that binary constructions of HE and SS exist, we discuss them in Section 7.)

Details: In GCs, a function is first represented as a Boolean circuit. One party (the garbler) assigns two random labels (keys) to each input wire of each gate, labels correspond to the values 0 and 1. For each gate, the garbler then generates an encrypted truth table that maps the output labels to the gate's input labels. The garbler sends the generated garbled circuit to the other party (the evaluator), along with the labels corresponding to its inputs. The evaluator then uses the OT protocol (see below) to obtain the labels corresponding to its inputs without the garbler learning the input values. At this point, the evaluator can execute the circuit in a gate-by-gate fashion, without learning intermediate values. Finally, the evaluator shares the output labels with the garbler who maps them to plaintext values.

System implications: GCs, like SS, involve both the client and the server in the actual computation; one must be the garbler and the other evaluator. The process of evaluating and garbling gates is compute-intensive, involving up to two AES computations (and additional pre-/post-processing logic) per truth table entry. Recent optimizations have been made to improve GC compute (i.e., FreeXOR [49] and HalfGate [90]); we employ these optimizations and note that run-times are still long. In addition, the wires and tables must be transmitted between the parties and stored by the evaluator prior to being used. As we will show, this introduces substantial communication and storage overheads on the system.

2.1.4 Oblivious Transfer. OT [65] enables a receiver to learn only one of two values sent by a transmitter without the transmitter learning which value was learned. It is a fundamental building block in MPC and the foundation for many constructions including GCs.
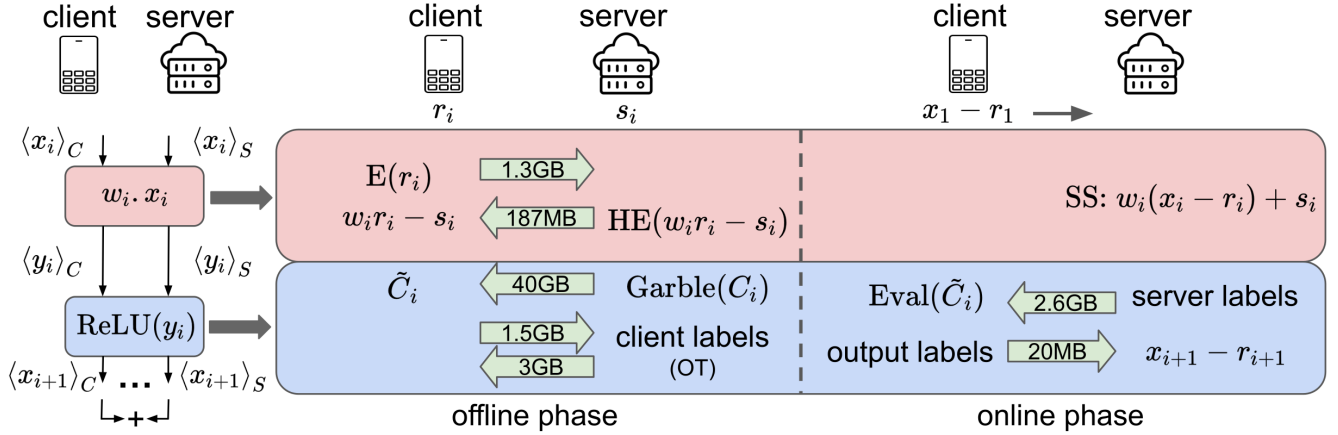
Figure 2: Baseline PI protocol using HE and SS for linear layers and GC for ReLU layers for ResNet-18 on TinyImageNet.

Details: In 1-out-2 OT [65] a sender has two input values x_0, x_1, and the receiver has a selection bit r. The OT protocol allows the receiver to obtain x_r without learning x_{1-r} and without the sender learning r. The simplest OT protocol requires expensive public-key cryptography, but can be modified to use cheaper symmetric key cryptography by *extending* a small number of *base OTs* [42].

System implications: The computational and storage costs of OT are relatively small, especially with OT extensions. OTs do add moderate communication overheads, but even these are small compared to the communication costs of transmitting garbled circuits. Some of our protocol optimizations add extra OT costs, but given that OTs themselves are relatively cheap, the overall impact on our optimizations is small as well.

2.2 Private Inference

We begin by reviewing a class of hybrid PI protocols that use HE and SS for linear layers, and GCs for ReLU layers [54, 57]. These protocols have achieved state-of-the-art results in the 2PC setting. In this setting, we assume that a client wishes to obtain an inference on her private data using the server's proprietary deep learning model. The server should not learn the client's input and the client should not learn the parameters of the server's model. Following prior work, we assume both parties are honest-but-curious, i.e., they follow the protocol faithfully but try to extract information during protocol execution. We protect against the same threat models and refer to prior work, e.g., DELPHI [57], for details.

We briefly describe DELPHI [57], one such hybrid protocol (see Figure 2) that we assume as a baseline in this paper. The protocol consists of an offline phase, which only depends on the network architecture and parameters, and an online phase, which is performed after the client's input is available. (One of the major insights in DELPHI was to move many of the expensive computations to the offline phase to improve online inference latency.) We represent model parameters at layer i with $\mathbf{w_i}$ (omitting biases in the description for simplicity). The layers after each linear and ReLU operation

are represented by $\mathbf{y_i}$ and $\mathbf{x_i}$, respectively. At a high level, the protocol allows the client and the server to hold additive shares of each layer value during inference evaluation.

Offline Phase. During the offline phase, the client generates the encryption keys and sends them to the server. The client and the server then sample random vectors per each linear layer from a finite ring, denoted by $\mathbf{r_i}$ and $\mathbf{s_i}$ respectively. The client homomorphically encrypts its random vectors and sends them to the server. After obtaining $E(\mathbf{r_i})$, the server computes $E(\mathbf{w_i}.\mathbf{r_i} - \mathbf{s_i})$ homomorphically and sends it back to the client. The client decrypts the ciphertext received from the server and stores the plaintext values. For each non-linear ReLU operation, the server constructs and garbles a circuit that implements $\text{ReLU}(\mathbf{y_i}) - \mathbf{r_{i+1}}$, where $\mathbf{r_{i+1}}$ is the next layer randomness (provided by the client). The server sends the garbled circuits to the client and the two parties engage in OT after which the client obtains labels corresponding to its inputs. At the end of the offline phase, the server stores its random vectors $\mathbf{s_i}$ and the garbled circuit input and output encodings, and the client stores its random vectors $\mathbf{r_i}$, garbled circuits, and the labels corresponding to its inputs. The communication and storage requirements for ResNet-18 inference on a single input from TinyImageNet are shown in Figure 2.

Online Phase. The client's input $\mathbf{x_1}$ is available in the online phase. The client computes $\mathbf{x_1} - \mathbf{r_1}$ and sends it to the server. At the beginning of ith linear layer, the server holds $\mathbf{x_i} - \mathbf{r_i}$ and the client holds $\mathbf{r_i}$. The server computes its share of layer output as $\langle \mathbf{y_i} \rangle_s = \mathbf{w_i}(\mathbf{x_i} - \mathbf{r_i}) + \mathbf{s_i}$. The client holds $\langle \mathbf{y_i} \rangle_c = \mathbf{w_i}.\mathbf{r_i} - \mathbf{s_i}$ from the offline phase, and the client and the server hold additive shares of $\mathbf{y_i} = \mathbf{w_i}.\mathbf{x_i}$. To evaluate the ReLU layer, the server obtains the labels corresponding to its share of ReLU input $\langle \mathbf{y_i} \rangle_s$ and sends them to the client. The client now holds labels for the server's input, as well as labels for its inputs $\langle \mathbf{y_i} \rangle_c$ and $\mathbf{r_{i+1}}$, which were obtained during the offline phase along with the garbled circuits. The client evaluates the circuit and sends the output labels to the server who is then able to obtain $(\mathbf{x_{i+1}} - \mathbf{r_{i+1}})$. At this point, the client and server hold additive shares of $\mathbf{x_{i+1}}$ and will similarly evaluate subsequent layers.
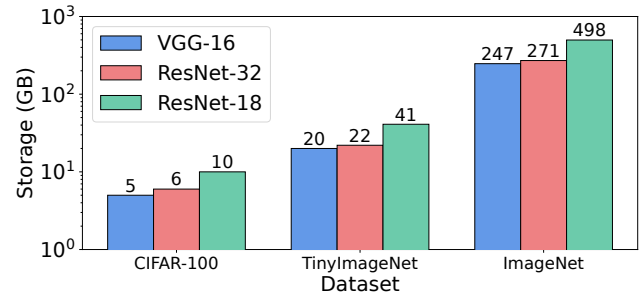
Hybrid versus HE-only PI. In contrast to hybrid protocols, HE-only PI protocols do not have an offline phase; all PI computation and communication takes place once the client's input is available [33]. Specifically, the client encrypts their input x_1 and sends it to the server to process the inference under HE. The server then sends the encrypted prediction to the client who can decrypt the ciphertext using their private key. However, in HE-only protocols, ReLU (and other non-linearities) must be replaced by polynomial functions in order to be supported by HE operations. This use of polynomials has been shown to greatly reduce network accuracy, especially for deeper networks [27]. The hybrid PI protocols used in this paper preserve accuracy by implementing ReLU using garbled circuits rather than using polynomial approximations.

Given the high cost of performing HE computations during the online phase, recent HE accelerators have been proposed to reduce compute latency [66, 71, 72]. These HE accelerators are evaluated with shallow networks (6, 10, and 20 layers) and small-input datasets such as MNIST and CIFAR-10. Cheetah [66] looks at deeper networks (e.g., ResNet50), and follows Gazelle's [44] hybrid protocol. However, Cheetah focuses on only accelerating the convolution and fully connected layers using HE and does not consider the processing time of ReLU, see Section 2A in paper [66]. For example while Cheetah reports an HE processing time of 198ms for ResNet-50 on ImageNet (Table 6 in [66]), the total end-to-end inference latency would be 215 seconds when considering both the online computation and communication required for evaluating the 9.6 million ReLUs in ResNet-50 (we observe 83 µs and 2.05 KB per ReLU which is in agreement with Table 3 from [57]). This calculation optimistically assumes 4 threads for ReLU evaluation as Gazelle and Delphi do as well as 10 Gbps bandwidth (equivalent to LAN, as used in [10]). Cheetah could be used to speed up the HE compute in this work, however, we optimize beyond accelerators such as Cheetah by analyzing system-level bottlenecks caused by not only HE latency, but also GC latency, communication, and storage. As we consider and optimize the full system, the PI latency reported here is much higher; ReLUs are expensive in PI.

3 METHODOLOGY

We perform experiments using ResNet networks [39] (ResNet-32 and ResNet-18) and VGG-16 [75] on the CIFAR-100 [50] and Tiny-ImageNet [89] datasets. As in prior work, we remove downsampling and replace max-pooling with average-pooling [31, 43]). CIFAR-100 has images of spatial resolution 32×32 while TinyImageNet's images are 64×64. Generally, from ResNet-32, VGG-16, and to ResNet-18 the number of parameters, FLOPs, ReLUs, and accuracy increases.

We stand apart from prior work by modeling the client device after an Intel Atom Z8350 embedded device (1.92 GHz, 4 cores, 2 GB RAM) and the server as an AMD EPYC 7502 (2.5 GHz, 32 cores, 256 GB RAM). Prior work runs the client-side computation on server hardware. We measure the latency, storage, and communication costs for the online and offline phase using the DELPHI codebase [57]. For all networks, we averaged these costs over 10 and 5 runs for CIFAR-100 and TinyImageNet, respectively. We implement LPHE using DELPHI's implementation of Gazelle's [44] HE evaluation algorithm and benchmark the HE computations for each linear layer used. These HE operations are implemented using

Figure 3: Total amount of pre-processing data that must be stored on the client's device *per inference*. Garbled Circuits make up a majority of this storage cost.

SEAL [73]. When using LPHE, we set the number of threads to the number of linear layers.

We use these measurements to construct a model of a system for PI and a simulator using Simpy [56] to explore and evaluate tradeoffs under different system conditions. We model a single-client and single-server system where inference requests are queued and served in a FIFO manner. We generate inference requests following a Poisson distribution, as in prior work [26, 36, 38, 45, 53, 67]. We validate our simulator against DELPHI by configuring our simulator to match their system and measure the compute latency of GC garbling, GC evaluating, and HE linear evaluation: we report a relative error of 0.9% for TinyImageNet on ResNet-18. Our code is available at the following GitHub repository: https://github.com/kvgarimella/characterizing-private-inference.
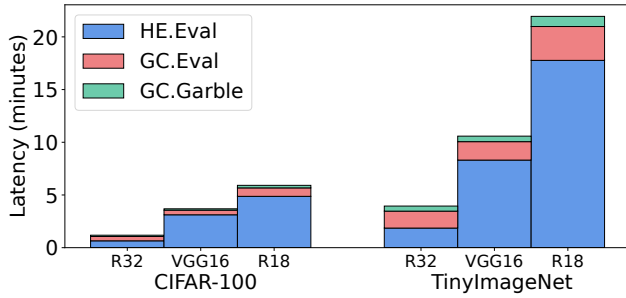
4 PERFORMANCE CHARACTERIZATION OF PI

This section presents our PI profiling results for multiple datasets and networks. We start by characterizing the overheads of individual inferences and then study the impact of a stream of requests on mean inference latency. We note that all results in this section use the baseline Server-Garbler protocol.

4.1 Single Inference Overheads

4.1.1 Storage: ReLUs dominate the storage overheads of PI. ReLU GCs are generated by the server in the offline phase and stored by the client for later use during the online phase. Profiling the fancy-garbling library [5, 12] we find that the server (garbler) incurs a modest storage penalty of 3.5 KB per ReLU to store the encoding information for inputs to the GCs. The client (evaluator), however, incurs a 5× greater storage cost of 18.2 KB per ReLU from storing the GCs themselves.

Figure 3 shows the total client storage costs of PI for different datasets and networks. For networks run on smaller inputs (CIFAR-100), the client must have at least 5 to 10 GB of storage available for a *single inference* (note that these pre-computed values cannot be reused or amortized over multiple inputs). Larger inputs and larger networks have more ReLUs and storage costs go up proportionally. For TinyImageNet, which has been used extensively in PI research, the most accurate model (ResNet-18) requires 41 GB *per inference*. This is a major obstacle for PI: The global average

Karthik Garimella, Zahra Ghodsi, Nandan Kumar Jha, Siddharth Garg, and Brandon Reagen

Figure 4: Latency of homomorphic evaluations for linear layers (HE.Eval), GC garbling (GC.Garble) and GC evaluation (GC.Eval) *per inference.* **GC.Eval occurs client-side and the rest are processed on the server.**

Figure 5: Communication latency *per inference* **on ResNet-18 with TinyImageNet inputs as a function of total bandwidth. GC communication accounts for a majority of this latency.**

smartphone storage capacity is nearly 100GB [85], meaning that an individual would need to allocate nearly half of their total storage for just a single inference pre-compute. Prior work, however, has not contended with this limitation.
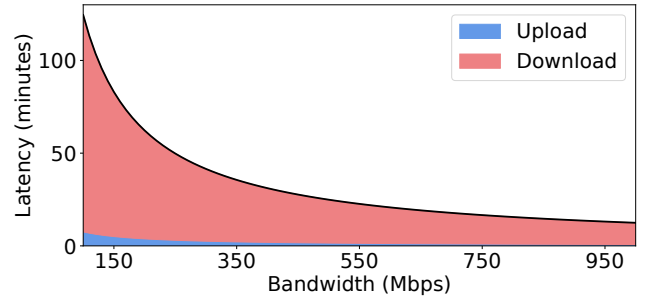
Figure 3 also reports the storage overheads of ImageNet, which are on the order of hundreds of GBs. Storing (and communicating) roughly half a terabyte of data per inference is beyond the capabilities of current technology; ImageNet is therefore not commonly studied in the PI literature.

4.1.2 Computation: In the baseline Server-Garbler protocol, the server is responsible for executing HE computation, GC garbling, and SS evaluation while the client must process GC evaluations. Secret shares generated via HE and GC garbling are performed in the offline phase, leaving only GC evaluation and the server's SS evaluation as the computation that must happen online.

Figure 4 shows the latency breakdown of these cryptographic primitives for a single inference. As can be seen, offline HE computations of linear layers (blue) dominate the total compute cycles for PI protocols. Though smaller than HE, GC evaluations (red) are a sizeable fraction of overall computational cost (and dominate online costs), while offline GC garbling (green) cost is almost negligible. The asymmetry between GC evaluation and garbling is apparent because we characterize the client (evaluator) using a realistic client device that is much less powerful than the server (garbler). These observations highlight two reasons prior work is overly optimistic in its evaluation of PI costs: (1) by assuming that offline costs can be hidden via pre-computation, prior work ignores an otherwise dominant component of the overall cost; and (2) online costs are also underestimated if the computational asymmetry between client and server is ignored. We do not plot SS evaluation as its run-time is negligible when compared to HE and GC (e.g., 0.61 seconds for ResNet-18 on TinyImageNet).

4.1.3 Communication: Hybrid PI protocols require multiple rounds of interaction between the client and server during both the offline and online phases, as shown in Figure 2. The dominant communication costs are from the GC protocol, including its OT sub-protocol.

Figure 5 shows the total communication latency of PI for ResNet-18 on TinyImageNet. Bandwidth sensitivity is shown by sweeping total wireless capacity from 100 Mbps to 1 Gbps; the results assume

Table 1: Total time (seconds) for the Server-Garbler protocol running ResNet-18 on TinyImageNet.

	GC	HE	SS	Comms	Total
Offline	25.1	1080	0.00	704	1809
Online	200	0.00	0.610	42.5	243
Total	225	1080	0.610	747	2052

an even split between the upload and download of the total bandwidth. The data highlights a stark imbalance between the amount of information sent to the server (upload) and the amount of information received from the server (download). Most of the data transmission (81.5% of the total) is in download due to the server sending GCs to the client device during its offline phase. (Recall that the ReLU GCs are on the order of tens of GBs for TinyImageNet). Assuming a state-of-the-art 5G connection of 1 Gbps, we can see the total communication time is 11 minutes, which is on par with the total HE computation costs. The data also shows that much of the available wireless bandwidth is wasted: 50% of the 1 Gbps is reserved for upload even though upload accounts for only 5.7% percent of the total transmitted data. In Section 5 we show how better provisioning the wireless bandwidth can provide significant improvement.
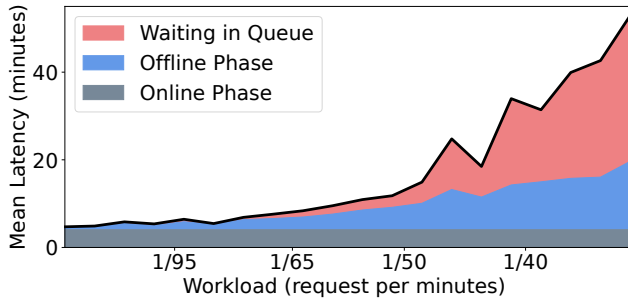
4.2 PI with Streaming Inference Requests

Table 1 presents the total PI time breakdown for a single ResNet-18 inference on TinyImageNet. Prior work has assumed that the entirety of the offline latency, accounting for 88% of the total inference run-time, can be hidden and assumed to have completed sometime prior to when the actual inference must take place. While this may be possible when computing a single inference, it is not the case when considering a rate of inferences to process.

The fundamental problem that has been largely overlooked is that the storage costs are so extreme that systems can realistically only store pre-computes for a few inferences, if any. These pre-computes serve as the buffer between online and offline cost. As requests start to arrive, this buffer is rapidly depleted, as each takes GBs of storage limiting what the client can hold. And even though the offline phase can run in parallel with the online phase as requests arrive, the high offline latency (1808 seconds) can only be hidden

Figure 6: Proposed Client-Garbler protocol. The server (rather than the client) stores the garbled circuits ($\tilde{C}$). In addition, server labels must now be communicated via OT in the online phase since they are input dependent.

Figure 7: Mean PI Inference latency with increasing inference requests rate (requests/min).

at extremely low arrival rates. In this light, the performance claims of hybrid systems, which consider isolated inferences and optimize only for online latency, have been overly optimistic and sweep large overheads (up to 88% of end-to-end latency) under the rug.

In Figure 7 we vary the arrival rate of inference requests from 1 request every 3 hours to 1 request every 15 minutes for ResNet-18 on TinyImageNet. We assume a generous 128 GB of client-side storage is available to hold pre-computes. The workload is simulated for 24 hours and we average the inference latencies over 50 runs of the simulation where each run is a unique sample from the arrival rate distribution. At the near-zero arrival rate region (far left of Figure 7), the mean inference latency is attributed completely to the online phase (grey) of the PI Protocol, as expected. As the arrival rate increases, and starting at 1 request every two hours, mean average latency is impacted by the offline phase (blue), implying that the arrival rate has surpassed the rate of pre-compute refill. And at 1 request every 70 minutes, we observe inference rates beginning to queue up (red). By one request every 30 minutes, each inference must pay the full PI latency (online+offline) per request in addition to waiting in the queue for previous requests to complete. As request pile up, we see the queue waiting time increase and quickly dominate the total latency.
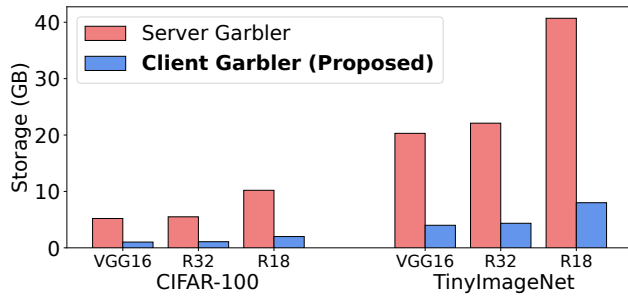
5 PROPOSED PI OPTIMIZATIONS

This section presents our proposed optimizations for PI: the Client-Garbler protocol, layer-parallel HE, and wireless slot allocation. We explain and quantify the benefit of each optimization and conclude with an evaluation against prior work to demonstrate the total improvement.

5.1 The Client-Garbler Protocol

As previously shown, the client-side storage requirements of existing protocols are prohibitively high, reaching 41 GB for a single ResNet-18 inference on TinyImageNet.

Solution: To overcome this limitation, we propose the Client-Garbler protocol. Here, the GC roles of the client and server are reversed: the client becomes the garbler and the server the evaluator (see Figure 6). In the offline phase, the client garbles ReLUs and sends the GCs along with the client labels to the server which stores them for use during the online phase. The parties engage in base OT offline so that in the online phase, the server can obtain their inputs to the GC using extended OT. The function to be garbled remains exactly the same as before: combine the shares of the prior linear layer, perform ReLU, and mask the output with the client's random share for the following linear layer (as shown in Figure 5 of [44]). Furthermore, there are no security implications of this optimization since the security guarantees of GC hold regardless of the role each party plays.

Using the Client-Garbler protocol, the large client-side storage requirements are now shifted to the server, thus taking advantage of the server's ample storage resources. Furthermore, the Client-Garbler protocol also reduces the online phase latency as the powerful server, rather than the client, performs GC evaluations. A drawback is that the parties must now engage in OT during the online phase; however, we observe OT communication time to be minimal and that Client-Garbler provides a net benefit to reducing online latency. In particular, the server-side evaluation of garbled ReLUs in the online phase outweighs the added communication latency from OT. For example, on TinyImageNet on ResNet-18,

Karthik Garimella, Zahra Ghodsi, Nandan Kumar Jha, Siddharth Garg, and Brandon Reagen

Figure 8: Client-side storage requirements for baseline Server-Garbler and proposed Client-Garbler protocols.

Figure 9: Baseline sequential vs. proposed layer parallel HE (LPHE) latency on the server.
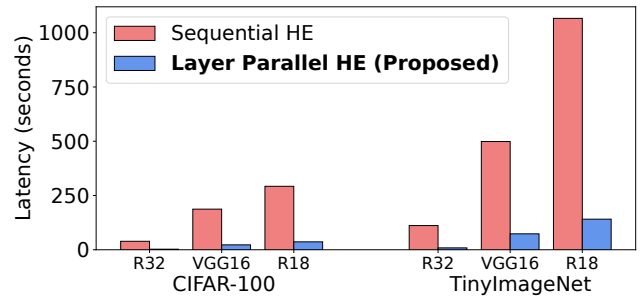
Server-Garbler (Client-Garbler) has an online communication latency of 27.1 seconds (101 seconds) and a GC evaluation time of 200 seconds (11.1 seconds). Thus, Client-Garbler provides an online speedup of 2.02×. The Client-Garbler also increases the offline run-time since GC garbling now runs on the slower client device. However, this is acceptable because the ability to store more precomputes and mask offline costs altogether outweighs the increase in offline compute latency.

We evaluate the energy implications of switching GC roles using `powertop` [81] to measure the energy consumption of garbling and evaluating 10,000 ReLUs (averaged over 30 independent runs on the Atom board). Evaluation and garbling consume 1.25 and 2.33 Joules, respectively. Thus, the Client-Garbler protocol increases client energy consumption by 1.8× as garbling requires additional encryptions compared to evaluating (Figure 1 in [35]). The compute latencies of both roles are also non-negligible (see Figure 4). These energy and performance profiles motivate research into custom hardware to accelerate both roles of GCs (discussed further in Section 6.2). As noted in FASE [40], accelerators can be built for both garbling and evaluation, as the computations are similar, and one simply has additional encryptions.

Nonetheless, the storage constraints of the Server-Garbler protocol prohibit the parties from engaging in an offline phase, leading to a significant increase in PI latency. For example on ResNet-18 on TinyImageNet, with 16 GB client storage, the Client-Garbler has an online latency of 2.3 minutes while the Server-Garbler's is 34.0 minutes. Thus, the Client-Garbler protocol results in a significant decrease in PI latency although the client consumes more GC energy.

In the absence of a full-system perspective, i.e., ignoring storage constraints, the asymmetry between the client and server, and assuming single inferences, Server-Garbler is indeed a natural choice since it moves OT to the offline phase and reduces client-side energy consumption. Accounting for these factors, however, we find that Client-Garbler significantly outperforms Server-Garbler.

Benefits of Client-Garbler: Figure 8 highlights the benefits of the Client-Garbler protocol over the Server-Garbler protocol in terms of client-side storage. On average we find the optimizations reduces the storage requirement of the client by 5×. As a concrete example, the Client-Garbler protocol reduce the client-side storage footprint of ResNet-18 on TinyImageNet from 41 GB to 8 GB. A

discussion of the benefits of Client-Garbler on mean PI latency is deferred to Section 5.4.

5.2 Layer-Parallel Homomorphic Encryption (LPHE)

In order to sustain high inference request arrival rates using PI, it is crucial to not only optimize the online phase but also the offline latency. During the offline phase, hybrid protocols incur a large HE run-time in order to generate additive secret shares for use during the online phase. For example, the HE run-times for ResNet-18 on TinyImageNet is 17.76 minutes, which makes up over 80% of the entire protocol's compute latency.
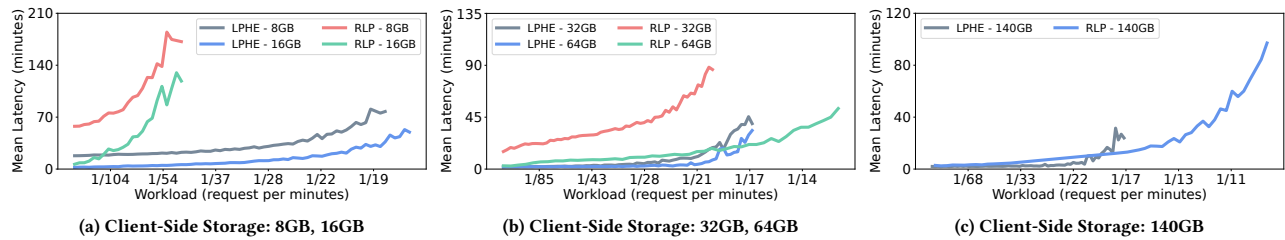
Solution: We identify a novel parallelization opportunity for HE, when used in hybrid, protocols termed layer-parallel HE (LPHE). In hybrid protocols, HE is used server-side to securely evaluate the linear layers of the neural network on the client's randomly sampled secret shares (as shown in Figure 2). We observe that these secret shares are *generated independently* for each linear layer of the network. This means that the secure evaluation of each linear layer need not be performed in-order and can be run in an embarrassingly parallel manner on the server. LPHE provides a general mechanism for speeding up HE computations as a function of the number of layers in the network.

In PI protocols like Gazelle, which do not use a combination of SS and HE for linear layers, HE evaluation is performed directly on network inputs to linear layers and LPHE cannot be leveraged as layers must be processed sequentially. Thus, PI protocols that combine HE and SS *can be faster than protocols that use HE alone.* This is counter-intuitive as HE+SS protocols are more complex and require more work. However, the opportunity to parallelize HE across layers more than makes up for the extra SS work.

Benefits of LPHE: Figure 9 shows the performance benefit of LPHE compared to prior work. The impact of LPHE is substantial on the overall HE speedup and we observe that the LPHE run-time is bounded by the longest-running HE linear layer. Therefore, we expect LPHE to scale well for even deeper networks. For ResNet-18 on TinyImageNet, LPHE reduces the HE evaluation run-time from 17.76 minutes to just 2.35 minutes. Across all datasets and networks, LPHE speeds up HE by 9.7×.

Several prior works have optimized HE leveraging the available parallelism within the core kernels and across ciphertexts *within*

(a) Client-Side Storage: 8GB, 16GB   **(b) Client-Side Storage: 32GB, 64GB**   **(c) Client-Side Storage: 140GB**
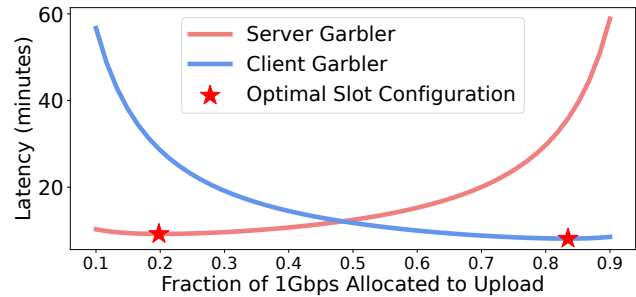
Figure 10: LPHE versus Request-Level Parallelism (RLP) for low (a), medium (b), and high (c) client-side storage. At 140GB, both LPHE and RLP utilize the same amount of server and client resources.

a layer, often for significant speedup as custom hardware accelerators [66, 71, 72]. We note that LPHE is a completely orthogonal source of parallelism, and it can be combined with the prior work for even more performance. These recent papers report that HE computations can be brought within a constant factor of plaintext performance; LPHE could be a significant step towards closing the remaining performance gap.

Comparison with Request-Level Parallelism: LPHE distributes the independent HE layer computations for a single inference pre-processing task across available cores. An alternative approach is for each core to handle independent inference pre-processing tasks, which we call Request-Level Parallelism (RLP). We now explore the trade-offs of LPHE and RLP in the context of varying the client-side storage capacity and inference workload for our proposed protocol (Client-Garbling and wireless slot allocation discussed in Section 5.3) on the TinyImageNet dataset for ResNet-18. For RLP, a single core is used on both devices to run a pre-processing phase. Thus, if the client-side storage is sufficient to buffer k inference precomputes, the two parties can engage in k pre-processing phases concurrently.
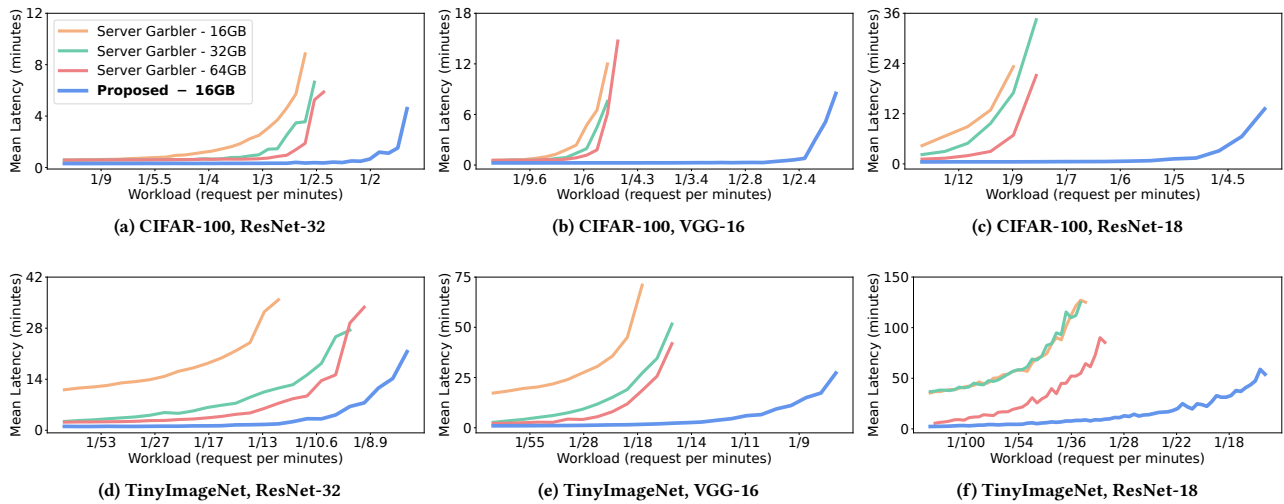
Figure 10 presents the results of our experiments and assumes 17 cores for server-side HE (there are 17 linear layers in ResNet18) and 4 cores for the client-side GC evaluation. At 8 GB of client storage (Figure 10a), pre-processing cannot be engaged for either LPHE or RLP and the entire PI protocol must be processed online. Here, LPHE leverages all 17 server cores for an end-to-end latency of 1053 seconds. Meanwhile RLP is limited to a single core, running in 3126 seconds. With 16 GB (Figure 10a), both RLP and LPHE inference latency improve as the pre-processing phase can complete offline during the downtime between requests. However, RLP can only engage in a single pre-processing phase, given storage constraints, and under-utilizes the cores, taking 3013 seconds per inference compared to LPHE's 936 seconds. We repeat these experiments using 32 and 64 GB (Figure 10b) for completeness, as these were assumed in other experiments. Here, the client device can store 3 and 7 precomputes, respectively, and RLP is better able to leverage the server resources with independent pre-processing. In the case of 32 GB, RLP now sustains a higher workload at 1 request per 21 minutes, but still has a higher mean inference latency when compared to LPHE. When the client device has 64 GB for precomputes, RLP is able to handle a higher workload (1 request per 13 minutes) compared to LPHE (1 request per 17 minutes), since RLP has a higher throughput for precomputes than LPHE. Finally, we set the client-side storage to 140 GB (Figure 10c); this allows for the client

Figure 11: Impact of wireless slot allocation (WSA) on communication latency for Server-Garbler and Client-Garbler protocols. Optimal WSA points that minimize communication for the two protocols are highlighted.

to store 17 precomputes and so RLP and LPHE utilize the same number of resources on both devices: all 17 server cores and all 4 client cores. Now, RLP can handle a workload of 1 request per 10 minutes while LPHE sustains a maximum workload of 1 request per 17 minutes, since RLP has a higher pre-compute throughput (1 precompute every 627 seconds) compared to LPHE (1 precompute every 939 seconds).

Without restricting storage, RLP results in a higher throughput of precomputes compared to LPHE since RLP distributes an equal amount of work across available cores. Meanwhile, LPHE distributes an unbalanced amount of work across cores since each linear layer's homomorphic evaluation varies in latency, thus leaving cores partially under-utilized. However, RLP only realizes its potential when the client is able to devote an immense amount of storage to precomputes, which is unlikely given the dozens to hundreds of GBs required and limited storage of clients' devices, e.g., a smartphone. Moving forward it is likely that the two approaches will be combined and adapt to the available storage to maximize throughput while minimizing latency. RLP also provides benefits in a setting where multiple clients use a single server to process their independent requests. Here, the total client storage scales with the number of clients (i.e., the number of requests). For example, if there were 9 clients each with 16 GB (as in Figure 10a), there would be a total of 144 GB net client storage, similar to the assumption in Figure 10c (140 GB). As a result, the server can exploit RLP and sustain a high throughput without needing LPHE. However, each client still has only enough storage for a single precompute, (which

(a) CIFAR-100, ResNet-32   (b) CIFAR-100, VGG-16   (c) CIFAR-100, ResNet-18

(d) TinyImageNet, ResNet-32   (e) TinyImageNet, VGG-16   (f) TinyImageNet, ResNet-18

Figure 12: Comparison of baseline Server-Garbler and proposed optimizations. In each case, the proposed optimizations (with limited client-side storage) exhibit a lower mean inference latency and sustain a higher arrival rate.

must be retained on the client device)—hence the cost of performing this precompute will still be incurred online at higher arrival rates, increasing client latency. From the perspective of each client, the latency would still be similar to the 16 GB results shown in Figure 10a.

5.3 Wireless Slot Allocation

Our final optimization seeks to reduce the communication latency of hybrid PI protocols. As we observed in Section 4, there is significant asymmetry in the amount of data sent from the client to server and vice versa. Therefore, we propose to minimize communication latency by matching the upload and download bandwidths to the data transferred from client to server and server to the client, respectively. Fortunately, 5G wireless standards provide exactly this flexibility.

Solution: Current 5G wireless standards use time division duplexing (TDD) to partition bandwidth between upload and download [1]. A 10 ms frame is divided into 10 sub-frames, each of which can be allocated to either upload or download, allowing significant flexibility in the fraction of bandwidth allocated in each direction. This flexibility can be, and has been, used to support autonomous driving and virtual reality applications, for instance, where almost all communication is in the upload [23]. The sub-frame structure can also be dynamically changed at a ms granularity [78]. We propose to exploit the flexibility in 5G wireless standards to match the asymmetric communication between client and server in hybrid PI protocols, allocating more upload slots for the Client-Garbler protocol and more download slots for Server-Garbler.

Benefit: Figure 11 shows the benefit and potential of the wireless slot allocation (WSA) optimization. Using our simulator, we set the total available bandwidth to 1 Gbps and calculate the total offline and online communication latency for both protocols while sweeping the percentage of slots allocated to upload. First, we note that as more slots are allocated for upload (download),
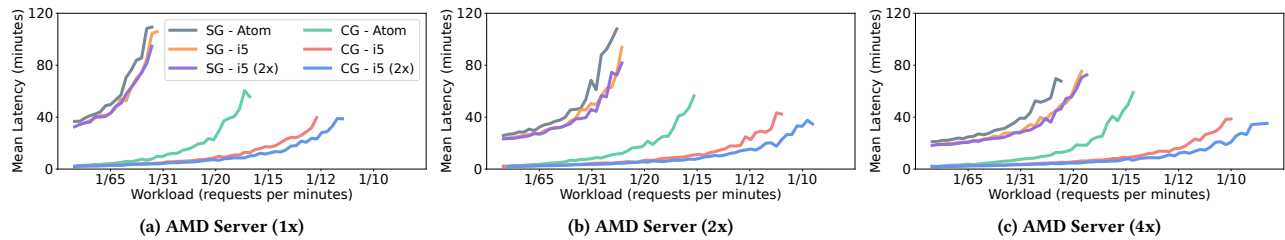
we approach an optimal slot configuration for the Client-Garbler (Server-Garbler) protocol. Second, we find that an even bandwidth split works well as a starting point when compared to sub-optimal settings. However, selecting the optimal point can provide up to a 35% communication time reduction.

We note that the ideal bandwidth partitions are not perfectly symmetric across protocols: Server-Garbler is optimal with 802 Mbps configured for download bandwidth and Client-Garbler is optimal with 835 Mbps set for upload bandwidth; this is due to the Client-Garbler protocol performing OT during the online phase.

5.4 Putting it All Together

Here we evaluate a standard high-performance protocol, namely the Server-Garbler protocol, against our optimizations (Client-Garbler, LPHE-enabled, and WSA-optimal). We analyze both protocols for all network-dataset pairs and set the client-side storage capacity to 16, 32, and 64 GB for the Server-Garbler protocol. For our proposed Client-Garbler protocol we set the client-side storage capacity to the lowest setting (16 GB), as this is sufficient space to store at least a single inference pre-compute. We set the server-side storage capacity to 10 TB and the total available bandwidth to 1 Gbps. We sweep arrival rates to cover the online-only and maximal sustainable throughput regions of all workloads and simulate over a period of 24 hours. For robust data, we calculate the mean inference over 50 independent runs of the simulation. Figure 12 presents the results of these experiments.

For CIFAR-100 (Figure 12a-c) both the Server-Garbler and our Client-Garbler are able to store at least a single inference precompute across all storage capacities considered and are thus able to engage in separate offline and online phases. With Server-Garbler, the mean inference latency quickly rises to the maximum sustainable throughput (as seen by the asymptotes), even when increasing the client-side storage capacity. This is caused by the inference request arrival rates outpacing the latency and resources required

Figure 13: Sensitivity studies of server and client compute capabilities while fixing the client-side storage to 16GB for TinyImageNet on ResNet-18 for both Server-Garbler (labeled SG) and Client-Garbler (labeled CG).

to engage in the offline phase of the Server-Garbler protocol and thus the costs must be incurred online.

Compared to the Server-Garbler protocol, our optimizations maintain a lower mean inference latency due to reduced client-storage pressure and computation/communication latency. Additionally, we observe that our proposed protocol has lower latency than the Server-Garbler protocol in the low arrival rate region (e.g., 1.88 minutes on ResNet-18 for TinyImageNet) as the server performs GC evaluation rather than the client and there is sufficient time between requests to replenish pre-computes. Finally, we note that the maximum sustained throughput of the proposed protocol extends out towards heavier workloads and thus can sustain a higher arrival rate when compared to prior work. For ResNet-18 on CIFAR-100, our proposed protocol can handle a request rate of 1 every 5 minutes while the Server-Garbler protocol is capped to 1 inference request every 12 minutes.

When moving from CIFAR-100 to TinyImageNet (Figure 12d-f), we first observe that not all client-side storage settings allow for the Server-Garbler protocol to engage in the pre-computation or the offline phase. Thus, the offline costs are entirely shifted online, even for the low arrival rate region. For example on ResNet-18 with TinyImageNet, both the 16 and 32 GB settings of the Server-Garbler are unable to engage in an offline phase as 41 GB are required for storing GCs. Hence, both the 16 and 32 GB configurations start with a high mean inference latency of nearly 1 hour. On the other hand, our proposed protocol can perform both the offline and online phase with just 16 GB of storage allocated to the client. Finally, our Client-Garbler protocol is able to sustain higher arrival rates as both LPHE and optimal WSA reduce compute and communication latency.
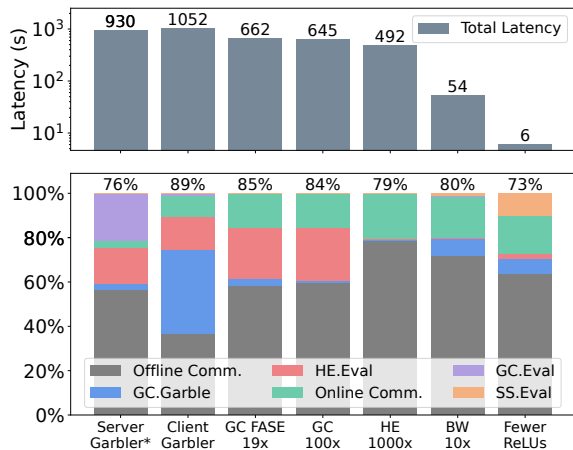
5.5 Sensitivity Study of Compute Capabilities

We now show the sensitivity of the Server-Garbler and proposed Client-Garbler protocol to client/server performance by sweeping device capabilities. We assume the following configurations for the client device: our baseline Intel Atom board, an Intel i5 processor, and a device with 2× the compute capabilities of the i5. For the server: our baseline AMD EPYC 7502 32-core processor and servers with 2× and 4× the compute capabilities of the AMD chip (we initially benchmarked an Intel Xeon Gold 5218 CPU @ 2.30 GHz, but the results were not significantly different from the AMD machine). Figure 13 shows mean inference latencies for TinyImageNet on ResNet-18 with a client-side storage capacity of 16 GB.

Figure 13a uses the standard AMD processor for the server and varies the client device. For the Server-Garbler protocol, the lack of sufficient client-side storage to buffer precomputes increases the end-to-end latency across all sustainable workloads as the entire protocol is executed on-the-fly once an inference is requested. Irrespective of the client device, the Server-Garbler protocol sustains a maximum workload of roughly 1 request per 30 minutes. For the Client-Garbler protocol, the 16 GB client-side storage is sufficient to buffer a single precompute; we observe a decrease in mean inference latency across all workloads as the offline phase can now be executed. When moving from an Atom to i5 client, the maximum sustainable throughput increases from 1 request every 15 minutes to 1 request every 10 minutes as client-side offline garbling reduces from 382.6 seconds (Atom) to 107.2 seconds (i5). A client device with 2× the compute capabilities of the i5 chip helps increase the maximum sustainable workload by further decreasing the garbling latency to 53.8 seconds.

We observe similar trends when increasing the compute capabilities of our server to both 2× and 4× of our standard AMD server (Figure 13b and Figure 13c, respectively). The Server-Garbler protocol is still unable to engage in a precompute phase and must incur the entire PI protocol online. However, since server-side garbling and server-side HE evaluation latency is reduced, the Server-Garbler protocol sustains a higher workload extending to 1 request per 20 minutes in the case of AMD 4×. For the Client-Garbler protocol, a faster server reduces the server-side HE evaluations and server-side GC evaluation, thus reducing both the offline and online time: Client-Garbler extends its sustainable workload to 1 request per 9 minutes in the AMD 4× case. Even in our optimal compute setting (i5 2× and AMD 4×), we observe minutes-long inference latencies, stressing the need for hardware accelerators to reduce the compute burden of both HE and GC. An in-depth analysis of future optimization (including accelerators) for PI latency is shown in Section 6.

6 DISCUSSION AND FURTHER OPTIMIZATIONS

In this section, we discuss opportunities for further improvement in PI latency that can be enabled by future research innovations. To this end, we breakdown the total costs of Server-Garbler and Client-Garbler protocols to identify key bottlenecks, and show how these costs decrease with accumulating optimizations. The results are shown in Figure 14 which plots total PI latency (top) and

Figure 14: Total (offline+online) latency (top) and breakdown of normalized latency (bottom) for different protocols and optimizations. Normalized latency bars are annotated with the percentage of total latency that is offline. * denotes LPHE and WSA are enabled for Server-Garbler.

normalized latency broken down into various components (bottom). The numbers on top of the normalized latency bars represent the fraction of PI latency incurred offline.

6.1 Comparing Protocol Costs

In comparing Server-Garbler and Client-Garbler, we first note that the LPHE and WSA optimization are applicable to Server-Garbler as well, and reduce its end-to-end latency by 54.6%. For a *single* inference, Server-Garbler outperforms Client-Garbler by 13% in terms of total PI latency, as shown in the first two bars of Figure 14 (top). We note that this result is not in contradiction with our findings in the previous sections; with storage constraints and over multiple inference requests, Client-Garbler outperforms Server-Garbler since it is able to better mask offline costs with buffered pre-computes *and* has lower online costs. In particular, even though Client-Garbler increases online communication latency due to OT (27.1 seconds to 101 seconds), Client-Garbler shifts the online GC evaluation from the compute-constrained client (modeled as an Intel Atom embedded device) to the powerful server (an AMD EYPC 32-core machine) and reduces GC evaluation from 200 seconds to 11.1 seconds. Thus, the total online latency decreases for Client-Garbler.

How can we further improve the Client-Garbler protocol? From Figure 14 (bottom), we observe that a key bottleneck in Client-Garbler is the high overhead of circuit garbling on the compute-limited client (the blue component of the Client-Garbler bar). We now discuss the impact of future research innovations that mitigate this and other performance bottlenecks.

6.2 Estimating the Benefits of Future Research

Accelerating GCs: Our data suggest accelerating garbled circuits would significantly reduce PI computational time for both protocols, assuming we accelerate both the evaluation and garbling phases. In the Client-Garbler setting, GC garbling accounts for 38% of the total PI run-time. We first assume the speedups from FASE [40] (shown

in Figure 14 as GC FASE 19×) for both garbling and evaluation. We observe a total PI speedup of 1.59×, from 1052 to 662 seconds, largely due to a decrease in client-side garbling as running GC evaluations on the server is relatively fast. Furthermore, we assume that a future accelerator provides an additional ∼ 5× speedup over FASE. The results are shown in Figure 14 (GC 100×): total PI latency is reduced from 662 to 645 seconds.

Accelerating HE: With the GC computational cost reduced, we see HE (red) emerge as the next major computational bottleneck. Prior work has been published on accelerators for HE as applied to private inference. Recent works have shown that special hardware can achieve 3-5 orders of magnitude speedup relative to CPU execution [66, 71, 72]. Since we are working at a high level, we conservatively assume HE can be sped up by 1000×. HE 1000× in Figure 14 shows that assuming an HE accelerator further reduces PI latency from 645 seconds to 492 seconds, a speedup of 1.31×. Here again, the online latency remains largely the same, as HE is processed entirely during the offline phase. Looking at the remaining time breakdown, we notice that all substantial sources of latency are incurred as communication time due to the MPC protocols.

Next Generation Wireless: One obvious solution to reduce communication time is to increase bandwidth. Here we optimistically assume that a future wireless generation, likely well beyond 6G [4], will provide 10× more bandwidth. The results are shown in Figure 14 as BW 10×, and we find the end-to-end PI latency is sped up by 9.1×, dropping from 492 to 54 seconds with online-only taking 12 seconds. Significant bandwidth savings resurface the high GC garbling times, which now take 10% of the offline latency. Considering both computational and communication overheads we observe that all significant overheads are now caused by GC ReLUs.

PI-Friendly Neural Architectures At this point, the performance limitations are due to ReLU. This is in stark contrast to performance limitations of plaintext inference, where ReLU is often an afterthought. To this end, there has been a recent thrust in the ML community to rethink neural architecture design to make them more PI-amenable, namely by reducing the number of ReLUs [16, 17, 32, 43, 55]. Significant progress has already been made, and by combining techniques achieving a 10× ReLU reduction seems well within reach. In Figure 14 (Fewer ReLUs), we plot the PI runtime assuming 10× ReLUs have been successfully removed. Reducing ReLUs has many positive effects: less garbling and evaluation time, less garbled circuit communication, less OT communication, and less storage pressure. These benefits culminate in a PI speedup of 9×, bringing PI inference down to 6 seconds, of which only 1.63 seconds is online. And while this is a significant speedup from our starting point, the total latency is still far from achieving plaintext-level speeds. Going forward, we encourage and welcome researchers to find innovative research avenues and design novel approaches to private inference that bridge this gap.

7 RELATED WORK

We now provide a summary of seminal work and recent developments in privacy-preserving computing and PI.

HE: Since the first successful demonstration of HE in 2009 [28], many advances have been made. Today, two classes of HE exist for computing on integers/fixed-point [11, 14, 25] and Boolean

logic [15, 20]. The former is efficient for addition and multiplication whereas the latter can compute arbitrary logic. Schemes are widely available today thanks to many software implementations [14, 37, 64, 73].

GCs: Garbled circuits were first proposed by Yao in 1986 [88]. They have recently received a resurgence of attention thanks to many significant advances in their performance [8, 48, 49, 59, 63, 90]. The optimizations aim to reduce the amount of work per gate by simplifying the computations and total amount of work. Two of the most widely used optimizations are FreeXOR [48] and HalfGate [90], which are used here. Like HE, many GC libraries exist [5, 19, 58, 61, 86].

SS: Secret sharing is a general technique first proposed by [9, 74]. It has received significant attention privacy-preserving inference, including PyTorch-compatible CrypTen [47], and GPU extension CryptGPU [77] for secure tensor processing.

Protocols for PI: Prior work has explored processing inferences using HE only [33, 72]. These protocols are convenient as privacy primitives are not changed. However, they typically introduce accuracy loss via the approximation of ReLU, even with complex training [27] techniques, and cannot leverage LPHE. To overcome this many have proposed hybrid protocols [44, 54, 57] that we improve on in this work. There has also been work on accelerating private computations assuming a trusted-third party (rather than 2PC, as assumed here), which assumes a weaker security model for higher performance [13, 18, 51, 58, 62, 83, 84]. Finally, prior work has performed a characterization of MPC protocols for Transformer models [87].

PI Amenable Networks: The ML community has begun exploring ways to design neural networks with fewer ReLUs. The general approaches taken so far have been to conduct ReLU aware neural architecture searches [16, 32], prune ReLUs from the networks [17, 43], and approximate ReLU computations for cheaper GC implementations [31]. These methods are orthogonal to the work of this paper and can further reduce GC storage and communication costs along with our proposed optimizations.

DELPHI [57] and AESPA [60] either partly or fully replace ReLUs with polynomial activation functions that are processed using Beaver Triples [7], which are cheaper in both compute and communication than GCs. However, recent results show that replacing ReLUs with low-degree polynomials reduces test accuracy, especially for deeper networks [27]. This is also confirmed in Figure 8 by DELPHI [57]. In this paper, we only consider highly-accurate, ReLU-only deep learning models that are state-of-the-art.

HE accelerators: A variety of hardware accelerators for HW now exist for both FPGAs [68, 70, 79] and ASIC [46, 66, 71, 72]. The design architectures vary and some support fixed or multiple schemes. However, the conclusions are largely the same: specialized logic is able to provide significant speedup over the CPU, easily achieving the 1000× assumed in Section 6. Recently, [82] proposed HE acceleration for resource-constrained clients and massively reduces the PI overheads.

GC accelerators: Prior work has also looked at accelerating GCs. The solution space includes general, re-programmable accelerators [76], more fixed-logic solutions [41], and even application-specific custom GC hardware for multiplication [40]. The designs achieve significant speedup by accelerating the core computations of GC gates and leveraging workload parallelism.

8 CONCLUSION

In this paper, we perform an in-depth evaluation of the system-level characteristics for state-of-the-art hybrid PI protocols. In contrast to prior work, which considers only individual inferences, we consider arrival rates of inference requests. Our experiments reveal excessive offline costs that have largely been ignored, including client-storage pressure, high-latency communication, and long-running HE-GC computations. The characterization using arrival rates reveals these costs cannot remain offline as there is insufficient storage to buffer pre-computations and insufficient time between requests to mask the extreme computation and communication latency. Leveraging insights from our characterization we propose three novel optimizations to overcome the storage limitations of the client, speed up HE computations, and lessen the communication latency. In all, our optimizations reduce the total PI time by 1.8×.

We also evaluate the potential of future optimizations across the PI pipeline and their effect on bringing PI close to practicality. We hope that our analysis encourages researchers to account for end-to-end system effects in PI optimizations.

9 ACKNOWLEDGEMENTS

A ARTIFACT APPENDIX

A.1 Abstract

We open source our private inference simulator at the following GitHub repo: https://github.com/kvgarimella/characterizing-private-inference. We construct a model of a system for private inference and a simulator using Simpy to explore and evaluate tradeoffs under different system conditions. We model a single-client, single-server setting where inferences are queued in a FIFO manner and are generated by sampling from a Poisson distribution.

The repository itself contains four high-level directories. The directory garbled_circuits contains the raw data for benchmarking ReLU Garbling and Evaluation on an Intel Atom Z8350 embedded device (1.92 GHz, 4 cores, 2 GB RAM) and an AMD EPYC 7502 server (2.5 GHz, 32 cores, 256 GB RAM). These two devices represent our client and server, respectively. Next, the directory layer_parallel_HE contains our code and the raw data for applying layer-parallel homomorphic to linear layer evaluations. The directory simulator contains our private inference simulator.

Finally, `artifact` contains scripts to replicate key figures in our paper.

A.2 Artifact check-list (meta-information)

- **Program:** Python Simulator for Private Inference
- **Run-time environment:** Linux OS with Python
- **Output:** Replication of key figures
- **Experiments:** Compute and storage cost analysis of private inference as well as results for simulating workloads of private inference.
- **How much time is needed to prepare workflow (approximately)?:** 15 minutes to clone GitHub repository, install dependencies, and run a simple test.
- **How much time is needed to complete experiments (approximately)?:** 3 hours.
- **Publicly available?:** https://github.com/kvgarimella/characterizing-private-inference
- **Code licenses:** MIT
- **Archived: https://www.doi.org/10.5281/zenodo.7633678**

A.2.1 *How to access.* The code repository can be accessed at: https://github.com/kvgarimella/characterizing-private-inference.

A.2.2 *Software dependencies.* Python is required to run the Private Inference simulator. The required Python packages can be found in the `requirements.txt` file within the GitHub repo. Optionally, Microsoft SEAL, Eigen, and OpenMP are required to benchmark Layer Parallel Homomorphic Encryption within the directory `layer_parallel_HE`.

A.3 Installation

First, clone the GitHub repository. Then, follow the instructions in the README to install the required Python packages. As a basic test, navigate to the subdirectory `simulator/experiments/` and run `python simulate_server_garbler.py`. This should run a single private inference workload experiment and create an output folder named `tmp` containing results of the simulation.

A.4 Experiment workflow

Please refer to the README within the `artifact` directory. In some cases, simply calling a Python script builds a figure presented in this paper. For other experiments involving simulations, shell scripts are provided to first run the simulations and then parse and plot the results from these simulations.

A.5 Evaluation and expected results

The directory `artifact` contains scripts to reproduce Figures 3, 4, 7, 8, 9, 12, and 14. Figures 3 and 4 show the end-to-end storage and compute overheads of running a single network inference for ResNet-32, VGG-16, and ResNet-18 for the image classification datasets CIFAR, TinyImageNet, and ImageNet. Figure 7 moves beyond our single inference analysis and introduced workloads of inferences over a period of time. Figures 8 and 9 show our proposed optimizations for both storage (Client-Garbler) and compute (Layer Parallel Homomorphic Evaluation). Figure 12 compares our optimized protocol to the baseline and is representative of our results for Figures 10 and 13. Finally, Figure 14 examines future optimization on end-to-end latency for private inference.

Navigate to the `artifact` directory. For each of the figures that can be reproduced, either run the provided python or shell script. Running the scripts simulates private inference workloads, parses the data, and reproduces the plots.

REFERENCES

[1] 2015. 3GPP TR 22.885 v14.0.0, Study on LTE Support for Vehicle to Everything (V2X) Services.
[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the ACM SIGSAC conference on computer and communications security.* https://doi.org/10.1145/2976749.2978318
[3] Ittai Anati, Shay Gueron, Simon P Johnson, and Vincent R Scarlata. 2013. Innovative Technology for CPU Based Attestation and Sealing.
[4] Muhammad Zeeshan Asghar, Shafique Ahmed Memon, and Jyri Hämäläinen. 2022. Evolution of Wireless Communication to 6G: Potential Applications and Research Directions. *Sustainability* (2022). https://doi.org/10.3390/su14106356
[5] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. 2019. Garbled neural networks are practical. *Cryptology ePrint Archive* (2019).
[6] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference.* https://doi.org/10.1007/3-540-46766-1_34
[7] Donald Beaver. 1995. Precomputing oblivious transfer. In *Annual International Cryptology Conference.* https://doi.org/10.1007/3-540-44750-4_8
[8] Donald Beaver, Silvio Micali, and Phillip Rogaway. 1990. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing.* https://doi.org/10.1145/100216.100287
[9] George Robert Blakley. 1979. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on.* https://doi.org/10.1109/MARK.1979.8817296
[10] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. 2020. MP2ML: A Mixed-Protocol Machine Learning Framework for Private Inference. In *Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES '20).* Association for Computing Machinery, New York, NY, USA, Article 14, 10 pages. https://doi.org/10.1145/3407023.3407045
[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory* (2014). https://doi.org/10.1145/2090236.2090262
[12] Brent Carmer, Alex J. Malozemoff, and Marc Rosen. 2019. swanky: A suite of rust libraries for secure multi-party computation. https://github.com/GaloisInc/swanky.
[13] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. 2019. ASTRA: high throughput 3pc over rings with application to secure prediction. In *Proceedings of the ACM SIGSAC Conference on Cloud Computing Security Workshop.* https://doi.org/10.1145/3338466.3358922
[14] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security.* https://doi.org/10.1007/978-3-319-70694-8_15
[15] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology* (2020). https://doi.org/10.1007/s00145-019-09319-x
[16] Minsu Cho, Zahra Ghodsi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. 2022. Sphynx: Relu-efficient network design for private inference. *IEEE Security & Privacy* (2022). https://doi.org/10.1109/MSEC.2022.3165475
[17] Minsu Cho, Ameya Joshi, Siddharth Garg, Brandon Reagen, and Chinmay Hegde. 2022. Selective Network Linearization for Efficient Private Inference. In *International Conference on Machine Learning.* https://doi.org/10.48550/ARXIV.2202.02340
[18] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2019. Secure evaluation of quantized neural networks. *arXiv preprint* (2019). https://doi.org/10.2478/popets-2020-0077
[19] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *The Network and Distributed System Security Symposium.* https://doi.org/10.14722/ndss.2015.23113
[20] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques.* https://doi.org/10.1007/978-3-662-46800-5_24
[21] Cynthia Dwork. 2006. Differential privacy. In *International Colloquium on Automata, Languages, and Programming.* https://doi.org/10.1007/11787006_1
[22] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* (2014). https://doi.org/10.1561/0400000042

[23] Fleischer et al. 2022. 5G TDD Uplink v1.0. https://www.ngmn.org/publications/5g-tdd-uplink-white-paper.html.

[24] Shimon Even, Oded Goldreich, and Abraham Lempel. 1985. A randomized protocol for signing contracts. *Commun. ACM* (1985). https://doi.org/10.1145/3812.3818

[25] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[26] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. https://doi.org/10.1145/3297858.3304013

[27] Karthik Garimella, Nandan Kumar Jha, and Brandon Reagen. 2021. Sisyphus: A Cautionary Tale of Using Low-Degree Polynomial Activations in Privacy-Preserving Deep Learning. In *ACM CCS Workshop on Private-preserving Machine Learning*. https://doi.org/10.48550/ARXIV.2107.12342

[28] Craig Gentry et al. 2009. *A fully homomorphic encryption scheme*.

[29] Craig Gentry, Shai Halevi, and Nigel P Smart. 2012. Fully homomorphic encryption with polylog overhead. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. https://doi.org/10.1007/978-3-642-29011-4_28

[30] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*. https://doi.org/10.1007/978-3-642-40041-4_5

[31] Zahra Ghodsi, Nandan Kumar Jha, Brandon Reagen, and Siddharth Garg. 2021. Circa: Stochastic ReLUs for Private Deep Learning. In *Advances in Neural Information Processing Systems*. https://doi.org/10.48550/ARXIV.2106.08475

[32] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. 2020. CryptoNAS: Private Inference on a ReLU Budget. In *Advances in Neural Information Processing Systems*. https://doi.org/10.48550/ARXIV.2006.08733

[33] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*.

[34] Oded Goldreich, Silvio Micali, and Avi Wigderson. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. https://doi.org/10.1145/28395.28420

[35] Chun Guo, Jonathan Katz, Xiao Wang, Chenkai Weng, and Yu Yu. 2019. Better Concrete Security for Half-Gates Garbling (in the Multi-Instance Setting). Cryptology ePrint Archive, Paper 2019/1168. https://doi.org/10.1007/978-3-030-56880-1_28 https://eprint.iacr.org/2019/1168.

[36] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. 2020. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In *ACM/IEEE 47th Annual International Symposium on Computer Architecture*. https://doi.org/10.1109/ISCA45697.2020.00084

[37] Shai Halevi and Victor Shoup. 2014. HElib-An Implementation of homomorphic encryption. *Cryptology ePrint Archive, Report 2014/039* (2014).

[38] Johann Hauswald, Michael A Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ronald G Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, et al. 2015. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. https://doi.org/10.1145/2775054.2694347

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. https://doi.org/10.1109/CVPR.2016.90

[40] Siam U Hussain and Farinaz Koushanfar. 2019. FASE: FPGA acceleration of secure function evaluation. In *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines*. https://doi.org/10.1109/FCCM.2019.00045

[41] Siam U Hussain, Bita Darvish Rouhani, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2018. Maxelerator: FPGA accelerator for privacy preserving multiply-accumulate (MAC) on cloud servers. In *Proceedings of the 55th Annual Design Automation Conference*. https://doi.org/10.1109/DAC.2018.8465770

[42] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *CRYPTO*. https://doi.org/10.1007/978-3-540-45146-4_9

[43] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. 2021. DeepReDuce: ReLU Reduction for Fast Private Inference. In *International Conference on Machine Learning*. https://doi.org/10.48550/ARXIV.2103.01396

[44] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium*.

[45] Harshad Kasture and Daniel Sanchez. 2016. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *IEEE International Symposium on Workload Characterization (IISWC)*. https://doi.org/10.1109/IISW C.2016.7581261

[46] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. 2022. BTS: An accelerator for bootstrappable fully homomorphic encryption. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. https://doi.org/10.1145/3470496.3527415

[47] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems* (2021). https://doi.org/10.48550/ARXIV.2109.00984

[48] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. 2014. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In *Annual Cryptology Conference*. https://doi.org/10.1007/978-3-662-44381-1_25

[49] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages, and Programming*. https://doi.org/10.1007/978-3-540-70583-3_40

[50] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2010. Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html* (2010).

[51] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *IEEE Symposium on Security and Privacy*. https://doi.org/10.1109/SP40000.2020.00092

[52] Kevin Lee, Vijay Rao, and William Christie Arnold. 2019. Accelerating facebook's infrastructure with application-specific hardware. *Facebook Retrieved* (2019).

[53] Jing Li, Kunal Agrawal, Sameh Elnikety, Yuxiong He, I-Ting Angelina Lee, Chenyang Lu, and Kathryn S McKinley. 2016. Work stealing for interactive services to meet target latency. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. https://doi.org/10.1145/2851141.2851151

[54] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. https://doi.org/10.1145/3133956.3134056

[55] Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. 2021. SAFENet: ASecure, ACCU-RATE AND FAST NEU-RAL NETWORK INFERENCE. *International Conference on Learning Representations* (2021).

[56] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* (2017). https://doi.org/10.7717/peerjcs.103/table-1

[57] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. DELPHI: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium*. https://doi.org/10.1145/3411501.3419418

[58] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. https://doi.org/10.1145/3243734.3243760

[59] Moni Naor, Benny Pinkas, and Reuban Sumner. 1999. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*. https://doi.org/10.1145/336992.337028

[60] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. 2022. AESPA: Accuracy Preserving Low-degree Polynomial Activation for Fast Private Inference. https://doi.org/10.48550/ARXIV.2201.06699

[61] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *30th USENIX Security Symposium*.

[62] Arpita Patra and Ajith Suresh. 2020. BLAZE: blazing fast privacy-preserving machine learning. *arXiv preprint* (2020). https://doi.org/10.14722/ndss.2020.24202

[63] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. 2009. Secure two-party computation is practical. In *International conference on the theory and application of cryptology and information security*. https://doi.org/10.1007/978-3-642-10366-7_15

[64] Yuriy Polyakov, Kurt Rohloff, and Gerard W Ryan. 2017. Palisade lattice cryptography library user manual. *Cybersecurity Research Center, New Jersey Institute of Technology (NJIT), Tech. Rep* 15 (2017).

[65] Michael O Rabin. 2005. How To Exchange Secrets with Oblivious Transfer. *IACR Cryptol. ePrint Arch.* (2005).

[66] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T Lee, Hsien-Hsin S Lee, Gu-Yeon Wei, and David Brooks. 2021. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *IEEE International Symposium on High-Performance Computer Architecture*. https://doi.org/10.1109/HPCA51647.2021.00013

[67] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *ACM/IEEE*

47th Annual International Symposium on Computer Architecture. https://doi.org/10.1109/ISCA45697.2020.00045

[68] M Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. HEAX: An architecture for computing on encrypted data. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems.* https://doi.org/10.1145/3373376.3378523

[69] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. 1978. On data banks and privacy homomorphisms. *Foundations of secure computation* (1978).

[70] Sujoy Sinha Roy, Furkan Turan, Kimmo Jarvinen, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data. In *IEEE International symposium on high performance computer architecture.* https://doi.org/10.1109/HPCA.2019.00052

[71] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A fast and programmable accelerator for fully homomorphic encryption. In *54th Annual IEEE/ACM International Symposium on Microarchitecture.* https://doi.org/10.1145/3466752.3480070

[72] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. 2022. CraterLake: a hardware accelerator for efficient unbounded computation on encrypted data.. In *ISCA.* https://doi.org/10.1145/3470496.3527393

[73] SEAL 2022. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[74] Adi Shamir. 1979. How to share a secret. *Commun. ACM* (1979). https://doi.org/10.1145/359168.359176

[75] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint* (2014). https://doi.org/10.48550/ARXIV.1409.1556

[76] Ebrahim M Songhori, Thomas Schneider, Shaza Zeitouni, Ahmad-Reza Sadeghi, Ghada Dessouky, and Farinaz Koushanfar. 2016. Garbledcpu: a mips processor for secure computation in hardware. In *53rd ACM/EDAC/IEEE Design Automation Conference.* https://doi.org/10.1145/2897937.2898027

[77] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CryptGPU: Fast privacy-preserving machine learning on the GPU. In *IEEE Symposium on Security and Privacy.* https://doi.org/10.48550/ARXIV.2104.10949

[78] Fengxiao Tang, Yibo Zhou, and Nei Kato. 2020. Deep reinforcement learning for dynamic uplink/downlink resource allocation in high mobility 5G HetNet. *IEEE Journal on selected areas in communications* (2020). https://doi.org/10.1109/JSAC.2020.3005495

[79] Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede. 2020. HEAWS: An accelerator for homomorphic encryption on the Amazon AWS FPGA. *IEEE Trans. Comput.* (2020). https://doi.org/10.1109/TC.2020.2988765

[80] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium.*

[81] Arjan van de Ven. 2022. The Linux PowerTOP tool. https://github.com/fenrus75/powertop.

[82] McKenzie van der Hagen and Brandon Lucia. 2022. Client-optimized algorithms and acceleration for encrypted compute offloading. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems.* https://doi.org/10.1145/3503222.3507737

[83] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* (2019). https://doi.org/10.2478/popets-2019-0035

[84] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proceedings on Privacy Enhancing Technologies.* https://doi.org/10.2478/popets-2021-0011

[85] Brady Wang. 2001. *Average Smartphone NAND Flash Capacity Crossed 100GB in 2020.* https://www.counterpointresearch.com/average-smartphone-nand-flash-capacity-crossed-100gb-2020/

[86] Xiao Wang, Alex J Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.

[87] Yongqin Wang, G. Edward Suh, Wenjie Xiong, Benjamin Lefaudeux, Brian Knott, Murali Annavaram, and Hsien-Hsin S. Lee. 2022. Characterization of MPC-based Private Inference for Transformer-based Models. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).* 187–197. https://doi.org/10.1109/ISPASS55109.2022.00025

[88] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science.* https://doi.org/10.1109/SFCS.1986.25

[89] Leon Yao and John Miller. 2015. Tiny imagenet classification with convolutional neural networks. *CS 231N* (2015).

[90] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* https://doi.org/10.1007/978-3-662-46803-6_8