IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 1, pp. 420–449. DOI:10.46586/tches.v2025.i1.420-449

Dash: Accelerating Distributed Private Convolutional Neural Network Inference with Arithmetic Garbled Circuits

Jonas Sander[1], Sebastian Berndt[2], Ida Bruhns[1] and Thomas Eisenbarth[1]

[1] University of Luebeck, Luebeck, Germany,
{j.sander,ida.bruhns,thomas.eisenbarth}@uni-luebeck.de
[2] Technische Hochschule Luebeck, Luebeck, Germany, sebastian.berndt@th-luebeck.de

Abstract. The adoption of machine learning solutions is rapidly increasing across all parts of society. As the models grow larger, both training and inference of machine learning models is increasingly outsourced, e.g. to cloud service providers. This means that potentially sensitive data is processed on untrusted platforms, which bears inherent data security and privacy risks. In this work, we investigate how to protect distributed machine learning systems, focusing on deep convolutional neural networks. The most common and best-performing mixed MPC approaches are based on HE, secret sharing, and garbled circuits. They commonly suffer from large performance overheads, big accuracy losses, and communication overheads that grow linearly in the depth of the neural network. To improve on these problems, we present Dash, a fast and distributed private convolutional neural network inference scheme secure against malicious attackers. Building on arithmetic garbling gadgets [BMR16] and *fancy-garbling* [BCM+19], Dash is based purely on arithmetic garbled circuits. We introduce *LabelTensors* that allow us to leverage the massive parallelity of modern GPUs. Combined with state-of-the-art garbling optimizations, Dash outperforms previous garbling approaches up to a factor of about 100. Furthermore, we introduce an efficient *scaling* operation over the residues of the Chinese remainder theorem representation to arithmetic garbled circuits, which allows us to garble larger networks and achieve much higher accuracy than previous approaches. Finally, Dash requires only a single communication round per inference step, regardless of the depth of the neural network, and a very small constant online communication volume.

Keywords: Garbled Circuit, Inference, GPU, TEE

1 Introduction

The recent progress of machine learning (ML) has resulted in broad adoption across almost all industries and government institutions. Consequently, ML is also being applied to security and privacy-sensitive domains like healthcare, law enforcement, finance, public administration, logistics, and many more. As ML models usually become very large and computation intensive, machine learning as a service (MLaaS) is a growing market in which data owners send their data to an inference device (or server) which produces the output. However, the owners (often called clients) of input and output may not want to or are not allowed to share their data due to privacy concerns or due to regulations such as the General Data Protection Regulation (GDPR) of the European Union. This can be solved by providing private inference protocols.

Many different cryptographic approaches have been used to protect the data in distributed ML applications, including secure multiparty computation (MPC) [TB19], homomorphic encryption (HE) [CBL+18], garbled circuit (GC) based techniques [RRK18],

Received: 2024-07-15 Accepted: 2024-09-15 Published: 2024-12-09

or combinations of the former [MZ17, JVC18, LMSP21, CGOS22]. While HE and MPC already provide quite efficient solutions for linear operations, the non-linear components of artificial neural networks (ANNs) lead to very large overheads in these schemes. Newer approaches thus split up the operations of ANNs and compute the linear components via HE or MPC and the non-linearities via GCs [CL01, BOP06, OPB07, BFL$^+$11, MZ17, LJLA17, RWT$^+$18, MR18, MLS$^+$20]. Unfortunately, switching the cryptographic techniques seems to always require communication between the data owner(s) and the inference device or between multiple inference devices which jointly perform the inference. This results in a linear number of communication rounds in the number of transitions from linear to non-linear layers (and vice versa) in the model architecture. Consequently, the computation of a linear layer that follows a non-linear layer must always wait for the completion of the required communication, slowing down the computation significantly. Even worse, in many approaches, the communication volume massively grows with the depth of the network [LJLA17, CBL$^+$18], turning communication volume into a massive bottleneck compared to the computational cost of the schemes. Excessive volumes of communication can be avoided by staying within a single technique, as long as all layer types can be computed efficiently within the chosen technique. Compared to other MPC techniques, GCs require the least amount of online communication. Several works have proposed and analyzed the usage of binary GCs for protecting machine learning [SS08, RRK18, RSC$^+$19]. While keeping online communication rounds low, binary GCs are not well-suited for the arithmetic of ANNs, resulting in huge and slow circuits. In their fairly recent seminal work, Ball et al. generalized the concept of GCs to *arithmetic GCs* [BMR16], which can be tailored to accommodate ANN-specific arithmetic [BCM$^+$19], significantly outperforming previous binary GC approaches. Dash builds on the cryptographic building blocks of these works, extends them with efficient power-of-two scaling in the Chinese remainder theorem representation, demonstrates their applicability in the offline-online scenario and supports GPU acceleration.

Non-private ML settings massively benefited from the use of accelerators like GPUs. Hence, using acceleration-friendly techniques for distributed private inference (DPI) is a logical step. While binary GCs were accelerated on GPUs already a decade ago [HMSG13], to the best of our knowledge, Dash is the first framework demonstrating GPU acceleration of arithmetic GCs. In modern works on accelerating cryptographic inference using GPUs such as GForce [NC21], GPU-accelerated GCs are not common. In fact, the authors [NC21] formulate efficient GPU acceleration of GCs as an open challenge. Dash accelerates highly optimized arithmetic GCs for CNN inference including the new garbled scaling gadget on the GPU and outperforms the previous state-of-the-art in outsourced inference.

An alternative to purely cryptographically protected distributed inference is to rely on TEEs such as Intel SGX or AMD SEV for the protection of data and/or model [MTH22, NCW$^+$21]. Slalom used a TEE combined with a simple stream cipher to protect input data during inference while leveraging a co-located GPU to accelerate linear layers [TB19]. Combining lightweight cryptography with efficient TEE solutions results in small runtimes for Slalom. However, Slalom needs to communicate the entire layer output between TEE and GPU after each switch from linear to non-linear layer (and the other way around), making the GPU's co-location to the TEE mandatory. Dash can also use a TEE but only needs to communicate the model in- and outputs with the inference device (e.g., GPU accelerated) at online time, making the computation of the hidden layers non-interactive.

Our contribution

In this work, we propose a novel arithmetic garbling framework based on the ANN-optimized arithmetic GCs from Ball et al. [BCM$^+$19]. Dash uses heavily optimized GCs to offer strong security guarantees for confidentiality of the inputs and outputs and the integrity of the computation even in the presence of malicious parties. To achieve security against such malicious parties, we can either rely on purely cryptographic techniques such as cut-and-choose and zero-knowledge proofs or use TEEs. Furthermore, the whole ANN can be garbled securely while the performance-critical GC evaluation (the protected ANN inference) can be performed by an untrusted device. We introduce the notion of *LabelTensors* that allow massive acceleration of this inference on GPUs, resulting in significantly improved performance during inference compared to previous solutions. Dash thus combines the advantages presented by GCs (very low communication complexity) with the advantages presented by GPUs (massive parallelism). Furthermore, we introduce an efficient *scaling* operation to arithmetic GCs, which allows us to further push the limit of the size of ANNs that can be efficiently garbled and thus the accuracy of the inference. Compared to secret-sharing-based MPC frameworks like Piranha [WWP22], where each party contributes a share and the memory requirement increases linearly, our solution has a constant memory requirement independent of the number of input providers. In summary, our contributions are:

- Introducing *LabelTensors* which leverage the intrinsic parallel nature of ANNs and allows us to evaluate them very efficiently on highly parallel GPUs. As a result, Dash outperforms inherently serial state-of-the-art solutions.
- Enabling modern quantization schemes by developing a *scaling* operation to achieve much higher accuracy on larger networks than previous GC frameworks as well as increasing the ANN size a GC can handle effectively.
- Regardless of the number of layer-transitions from linear to non-linear and vice versa in the model architecture, Dash requires only a constant number of communication rounds and a constant communication volume.
- Combining GCs with TEE-enabled modern hardware to guarantee *input privacy*, *integrity of the computation*, and *output privacy.*
- Dash can also be extended to guarantee security in the presence of an actively *malicious* attacker controlling multiple parties.
- Dash has a constant computation-, memory-, and communication-overhead independent of the number of participating input owners.
- Dash is open-source and source code will be released upon acceptance.[1]

2 Related Work

In 2017, Mohassel and Zhang [MZ17] introduced SecureML, a scheme for private inference and training in a setting with two untrusted but *non-colluding* servers, which then train the desired model using 2-party MPC techniques. The activation functions sigmoid and softmax are approximated due to their high complexity in an MPC setting, and GCs are used to further speed up the computation of these functions. DeepSecure [RRK18] preprocesses the networks and encodes them into a *binary* circuit for garbling. During the evaluation, optimized oblivious transfers are used for the evaluation of non-linear operations. Liu et al. introduced MiniONN for secure inference on ANNs in the one-server setting [LJLA17]. It also approximates activation functions and combines secret sharing with GCs. The evaluation of all these schemes shows that ANNs with seven or more activation layers generate so much overhead that the scheme is not usable any more.

Gazelle builds on MiniONN and uses HE in the linear layers and GCs in the activation layers [JVC18]. Every part of Gazelle was optimized for performance, including evaluating the linear layers and transforming the data between the linear and non-linear layers. Gazelle supports inference only and advanced the state-of-the-art runtime at the time by several orders of magnitude, mainly by optimizing the communication.

All these schemes use the CPU only. Since ML has benefited from GPUs, it is a natural direction to explore when trying to improve the performance of private ML schemes.

[1] https://github.com/UzL-ITS/dash

Delphi [MLS+20] transferred the techniques used by Gazelle to the GPU [MLS+20]. It accelerates the linear layers by moving the HE computations via additive secret sharing into the offline phase. In addition, a performance-accuracy tradeoff for ReLUs is calculated and hyperparameter optimization is used for the resulting architecture. This architecture remodeling of the ANN is timely and requires extensive retraining of the model.

Continuing with the GPU approach, Tramèr and Boneh introduced the Slalom scheme in 2019 [TB19], providing verified and private inference on ANNs in a one-server setting. The server must feature a CPU with a TEE and a co-located GPU. Several attempts to secure the one-server by leveraging a TEE have been introduced [OSF+16, TGS+18, HSS+18, HZG+18, LLP+19, ZHC+20], but Slalom is the first protocol to privately outsource the expensive linear operations of ANN inference from the TEE to a GPU via masking. The communication overhead increases linearly with the depth of the ANN. Like Dash, Slalom can also protect against a malicious attacker.

Faster CryptoNets [CBL+18] is a scheme for encrypted inference in the one-server setting which follows the two previously proposed CryptoNets [GDL+16, XBF+14] approaches and uses HE to perform ANN inference over encrypted inputs. The activations are approximated by quantized polynomials, which works best in the small interval $[-1, 1]$. Outside this interval, more significant errors occur. The practicable multiplicative depth of the HE scheme limits the ANN to three layers, after which the authors delegate some computations back to the clients, which contradicts the MLaaS setting and induces a large communication overhead of several hundred GBs to TBs.

Recent approaches, such as Piranha in 2022, focus on the usability and performance of the suggested solution [WWP22]. The Piranha platform allows developers of secret-sharing-based MPC schemes to use GPUs efficiently without knowledge of GPU programming. The models do not need to be retrained. Besides that Piranha does not add any features to the existing schemes. Dash also focuses on usability: Model and input owners can use the framework without knowledge of GCs or TEEs. They do not need to modify their ML Model since Dash supports loading models in ONNX format.

3 Preliminaries

We introduce the basics of ANNs, GCs, and TEEs to facilitate the explanation of Dash.

3.1 Neural Networks

For classification tasks, ANNs are used to map an input to an output-class, e.g., a picture of a handwritten digit to the output 0 - 9. Usually, this is done by a variety of layers that are consecutively applied to the input. For our purposes, we will consider *linear layers* such as dense layers and 2D convolutions, and *non-linear layers* like ReLU and sign activations.

3.2 Garbled Circuits

GCs were introduced by Yao [Rab05] and allow two-party MPC computations of binary circuits against a semi-honest or malicious attacker [LP07]. To garble a gate with two input wires x and y and an output wire z, the garbler generates two random labels $l^0, l^1 \in_r \mathbb{Z}_2^\kappa$ of lengths κ representing the 0- and 1-bit semantic for all in- and outputs. For a given gate functionality $f\colon \mathbb{Z}_2^2 \to \mathbb{Z}_2$, the garbler produces the garbled gate as a table of 4 ciphertexts $\mathrm{EN}_{l_x^a, l_y^b}(l_z^{f(a,b)})$ for all $a, b \in \mathbb{Z}_2$. Here, l_u^a corresponds to the label of wire u with semantic a and EN_k is a suitable encryption function with key k.

To garble an acyclic circuit of several gates, the above procedure is applied successively from the input to the output gates. Since the evaluator only knows the wire labels for a single input combination, he learns only the output label corresponding to this input. To prevent the evaluator from distinguishing between labels with 0- and 1-bit semantics, GCs can only be used for a single run. Besides, the ciphertexts must be shuffled per gate, as a canonical ordering exposes the key-bit-mapping. In the classical setting, the evaluator uses an oblivious transfer to obtain the labels of his private inputs.

We follow the conventional notation and note the garbling algorithm, which generates the garbled circuit gC from a given circuit C, as $\mathsf{Gb}(\mathsf{C}) = \mathsf{gC}, \mathsf{e}, \mathsf{d}$. The encoding information e are used by the encoding algorithm to garble the clear input and obtain the garbled input $\mathsf{En}(\mathsf{In}, \mathsf{e}) = \mathsf{gIn}$. We note the evaluation algorithm, which evaluates a garbled input on a GC and outputs the garbled output as $\mathsf{Ev}(\mathsf{gC}, \mathsf{gIn}) = \mathsf{gOut}$. To decode the garbled output, the decoding algorithm is used with the decoding information $\mathsf{De}(\mathsf{gOut}, \mathsf{d}) = \mathsf{Out}$. As Dash deals with neural networks, we also write NN and gNN instead of C and gC.

3.3 Garbled Circuit Optimizations

Optimizations for GCs focus on their size, the computational complexity, and the hardness assumptions needed to achieve appropriate security guarantees. Below, we concentrate on optimizations which generalize to the arithmetical domain and are supported by Dash.

Point-and-permute was introduced by Beaver et al. [BMR90] and reduces the number of necessary decryption operations to one ciphertext per gate. The idea is to append a pointer pair $(p, \overline{p})$ with $p \in_r \mathbb{Z}_2$ randomly chosen to each pair of input labels $(l^0 \parallel p, l^1 \parallel \overline{p})$ to sort the ciphertexts based on these pointer (called color bit). This ordering allows the evaluator to determine the correct output using only one decryption. *Free-XOR* was introduced by Kolesnikov and Schneider [KS08] and enables the evaluation of XOR gates without performing cryptographic operations or transmitting ciphertexts. The garbler chooses the input labels l^0 with 0-bit semantics randomly, and the labels with 1-bit semantics as $l^1 = l^0 \oplus R$, with $R \in_r \mathbb{Z}_2^k$ being a circuit-wide constant. With output label $l_z^0 = l_x^0 \oplus l_y^0$, the output of an XOR gate is simply evaluated as the XOR of the two input labels. *Half Gates* garble AND gates with only two ciphertexts per gate [ZRE15]. We give a detailed description of the arithmetical generalization used in Dash below.

The optimizations described so far mainly focus on the communication complexity of GC-based protocols. Another line of work also introduces computational improvements regarding the gate-level encryptions [NPS99, LPS08, HEKM11, KSS12]. The state-of-the-art was presented by Bellare et al. [BHKR13]. They propose to encrypt wire labels using a cryptographic permutation instantiated by fixed-key AES. By instantiating AES with a fixed and public key, the scheme only needs to perform a single key-derivation for a whole circuit. While avoiding key derivations drastically improves the computation time, it comes at the cost of introducing non-standard assumptions about AES to enable a security-proof in the random-permutation model (for a more details, see [GLNP15, GKWY20]).

3.4 Arithmetic Garbled Circuits

Implementing arithmetic operations via conventional binary GCs is expensive, especially compared to other MPC approaches, such as secret-sharing-based MPC. Ball, Malkin, and Rosulek [BMR16] introduced garbling gadgets for efficient garbling of arithmetic circuits over large finite fields. We will refer to their approach as BMR scheme. Considering our use case, their gadgets allow free addition, free multiplication with a constant, and efficient projection gates for arbitrary unary functions (w.r.t. communication complexity).

Starting from Free-XOR, Ball et al. [BMR16] consider labels as vectors of components from $\mathbb{Z}_p$. The encoding of a value $a \in \mathbb{Z}_p$ is given through $l^a = l^0 + aR$, with l^0 (chosen individually per input wire) and R (circuit-wide constant) being vectors of random elements from $\mathbb{Z}_p$. We call l^0 a *base label*. The construction considers wires with different moduli, called *mixed-moduli circuits*, and leverages different *offset labels* R_p for each module p. This allows for a Free-XOR generalization and free addition-gates as shown, e.g., in [MPS15].

Point-and-permute is generalized by using an element from $\mathbb{Z}_p$ instead of a single bit and choosing $1 \in \mathbb{Z}_p$ as the point-and-permute component of each base label R_p. A mixed-modulus-circuit consists of an acyclic structure of wires together with their moduli and gates constructed as follows:

- **Unbounded Fan-In Addition** To compute $(a, b) \mapsto (a+b) \bmod p$, we set $l_x^a + l_y^b \equiv (l_x^0 + l_y^0) + (a+b)R_p \bmod p$.
- **Multiplication by a public constant** To compute $a \mapsto ac \bmod p$, we compute $c \cdot l^a \equiv c \cdot l^0 + caR_p \bmod p$ with c being co-prime to modulus p (needed for technical reasons in the security proof, see also [BMR16]).
- **Projections for unary functions** To compute an arbitrary unary function $\varphi\colon \mathbb{Z}_p \to \mathbb{Z}_q$, we construct a garbled projection gate with value a on input wire x that consists of p ciphertexts of the form $\mathrm{En}_{l_x^a}(l^{\varphi(a)})$. Note that leveraging a generalization of the garbled row reduction [NPS99], one ciphertext could be removed. For simplicity, we ignore this optimization, as the saving is negligible for larger moduli.

While the first two gate types are free, the projection gate is not practical for large moduli p. Therefore, Ball et al. use a *composite primal modulus* (CPM) $P_k = 2 \cdot 3 \cdot \ldots \cdot p_k$ which is the product of the first k primes. They leverage the Chinese remainder theorem to represent the label values in a *residue (or CRT) representation*. Using CRT representations for optimizations in GCs was proposed before, e.g., in [AIK11]. From a circuit perspective, every value of a conventional wire is now represented by a bundle of wires, with each of the wires in a bundle corresponding to one residue in the CRT representation and one prime factor of the CPM P_k. Hence, instead of having a projection gate with P_k possible inputs, we can now represent this projection gate via k gates where the i-th gate has only p_i possible inputs. Both addition and multiplication (by a constant) gates remain free.

Sign Gadget Leveraging the mixed-modulus HG (see Section 7) with new optimizations targeting ANN operations, Ball et al. [BCM+19] demonstrate the practicability of garbled ANN inference. They propose a new approximated garbled sign gadget (see also Section 7) working over CRT representations with $\mathrm{sgn}_{l,h}(x) = l$ for $x \leq 0$ and $\mathrm{sgn}_{l,h}(x) = h$ for $x > 0$.

We use the sign function to construct the base-extension of our new scaling gadget. Following Ball et al. [BCM+19], Dash uses the sign gadget to garble the sign- and ReLU activations and provide highly parallel GPU implementations. To garble the sign activation, the gadget is applied on the inputs $\text{sgn-act}(a) = \mathrm{sgn}_{-1,1}(a)$. The ReLU activation is garbled via $\mathrm{ReLU}(a) = a \cdot \mathrm{sgn}_{0,1}(a)$, using the mixed modulus half gate for the multiplication.

3.5 Trusted Execution Environments

TEEs such as Intel SGX [MAB+13, CD16, Int21], AMD SEV [Kap16], Sanctum [CLD16], ARM CCA [LLD+22] and Intel TDX [SMF21] allow programs, containers, or whole VMs to be executed securely in hardware-sealed *enclaves*. The hardware isolates the enclave from other programs on the host, regardless of privilege level or CPU mode. Enclave code and data are protected even from a compromised operating system or hypervisor.

The key feature of TEEs besides strong memory isolation is *remote attestation*, a process that guarantees that a specific enclave, specific in the sense of the code and data it contains, has been deployed on a protected system. One distinguishes between local and remote attestation. Using remote attestation, an enclave authenticates itself (i.e., the code, identity, and the fact it is executed in a trusted environment) to a remote third party. This makes TEEs particularly attractive for use in cloud computing offerings like MLaaS. The cloud provider creates an enclave to which the cloud customer establishes a secure connection and authenticates the enclave via remote attestation. If two or more distrustful parties want to perform outsourced computation together, they can outsource their data and computation to a mutually trusted enclave [CB19].

Like Slalom, we use Intel SGX over newer TEEs such as TDX or AMD SEV to minimize the trusted computing base. SGX applications are divided into a trusted and untrusted component. The trusted component runs inside the enclave and communicates with the untrusted part outside the enclave through an interface defined by the developer. The trusted part of the application and the communication between the two components should be kept small to reduce the potential attack surface and improve performance.

Intel SGX was introduced in Intel Core CPUs in 2015, supporting 128MB to 256MB of *Processor Reserved Memory* (PRM) shared by all active enclaves. The original use in consumer devices was discontinued, but SGX remains a heavily supported feature in server-grade CPUs such as the 4rd Gen Intel Xeon Scalable server CPUs, providing enclaves with up to 1TB of protected memory. SGX's enclave memory is encrypted and integrity protected outside the CPU package. The memory management unit (MMU) decrypts and integrity checks data before loading it from memory into caches or registers. Nearly all server-level machines support TEEs (SGX by Intel or SEV by AMD), and they are used in practice, e.g. by the Signal messenger servers. As the main motivation for Dash are MLaaS scenarios in which the sensitive computation is performed on a server level device, it is valid to assume a TEE can be used.

4 Design of Dash

Dash provides an efficient implementation of GCs for BMR circuits that outperforms previous approaches significantly and allows larger models to be garbled effectively. Improvements are achieved through three main contributions. Firstly, the implementation contains state-of-the-art techniques for GCs (see Section 3) and makes use of the approximated ReLU technique of Ball et al. [BCM$^+$19], which gives a significant speedup.

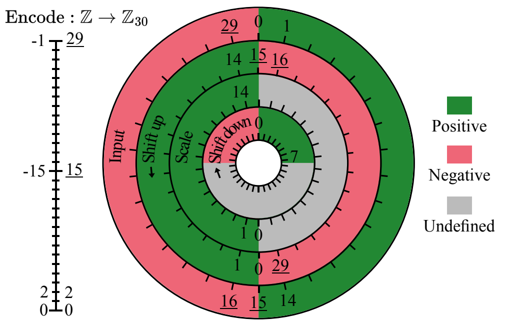
Secondly, we make use of specialized ML hardware such as GPUs. Previous implementations of GCs for ANNs, such as those by Ball et al., use a *scheduler* to choose the next gate in the GC to be evaluated. Due to this strictly linear approach, the massive parallelity of GPUs could not be used properly. In contrast, we do not look at individual gates but treat gates on the same layer as *tensors*, i.e., as multidimensional objects called *LabelTensors*. This allows the evaluator to evaluate a complete circuit layer in parallel. As a result, Dash scales up easily to large circuits with very high width. Classic examples of this kind of circuit are ANNs, but they also appear in many cryptographic use cases (e.g., via parallel encryption of blocks by AES or ChaCha20).

Thirdly, Dash includes a new garbled scale operation, allowing much larger networks to be processed due to being able to use more effective quantization schemes.

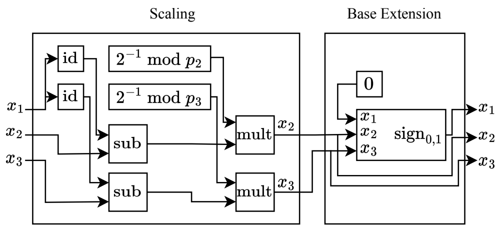
The modular approach taken in Dash's implementation allows for many deployment scenarios. For example, we can use TEEs both for the garbler and the evaluator, allowing the user to work with trusted data on an untrusted platform easily. Our end-to-end framework supports the common interchange format ONNX (used with TensorFlow and PyTorch). Hence, a user wanting to garble an ANN can easily enter a model in this format and will obtain a garbled version of it. This makes Dash usable for everyone.

4.1 Quantization and Encoding

ANNs typically operate over 32-bit floating point numbers. To increase the throughput and reduce the memory requirements of the models during the inference phase, models and inputs are quantized to integers. To garble ANNs, we need to quantize them as well. Dash supports two quantization schemes we call *SimpleQuant* and *ScaleQuant*. SimpleQuant has been used before in GNNP [BCM$^+$19] and ScaleQuant is a more advanced quantization scheme, inspired by Gupta et al. [GAGN15] and also leveraged by Slalom [TB19]. Using



(a) Schematic visualization.



(b) Circuit of the scaling gadget in Dash.

Figure 1: Visualization of our scaling operation in $\mathbb{Z}_{P_3}$.

SimpleQuant to quantize a value x, we multiply with a small quantization constant α and round to the nearest integer: $\text{SimpleQuant}(x, \alpha) = \text{round}(x \cdot \alpha)$.

In ScaleQuant, we extend the forward-pass with down-scaling operations, which allows quantizing larger ANNs without affecting the model's predictive power significantly. Without a garbled scale operation, previous arithmetic garbling schemes were not able to use ScaleQuant. This significantly limited the size of models that could be garbled effectively. We introduce the garbled scale operation in Subsection 4.2. To quantize a floating point number x, we use $\text{ScaleQuant}(x, \ell) = round(x \cdot 2^{\ell})$. In an ANN all weights are quantized with $\text{ScaleQuant}(x, \ell)$ and all biases with $\text{ScaleQuant}(x, 2\ell)$. In the forward pass, the outputs of layers with quantized parameters are then down-scaled with $\text{ScaleQuant}(x, -\ell)$.

Our arithmetic GCs operate on values from the finite ring $\mathbb{Z}_{P_k}$. To be able to garble the quantized ANNs, all integers, including all weights, biases, and inputs, must be encoded in $\mathbb{Z}_{P_k}$. We map all positive integers, including zero, to the *lower* half of $\mathbb{Z}_{P_k}$ and all negative numbers to the *upper* half of the ring: $\text{Encode}(x) = x \bmod P_k$ (see also Figure 1a).

4.2 The Scaling Operation

Scaling a number x by some *scaling factor* s means to compute $y = \lfloor x/s \rfloor$. We only consider $s = 2$, as successive applications of scaling by 2 can be used to implement ScaleQuant. First, we describe our scaling operation from a high-level perspective without considering that the inputs to Dash are in CRT representation. Next, we explain the details regarding the CRT representation and how the scaling operation is implemented in Dash.

4.2.1 High-Level Steps

Figure 1a shows the steps (1) *ShiftUp*, (2) *Scale*, and (3) *ShiftDown* of our scaling operation exemplarily for the case P_3. Table 1 shows an example for $P_2 = 6$ including the outputs of all intermediate steps. The $\text{ShiftUp}(x) = x + (P_k/2)$ step limits the result of the following $\text{Scale}(X) = \lfloor x/2 \rfloor$ step (including the base extension) to the positive value range $[0, P_k/2 - 1]$. Finally, we apply the $\text{ShiftDown}(x) = x - \lfloor P_k/4 \rfloor$ step to move the result to the correct value ranges $[0, \lfloor P_k/4 \rfloor]$ and $[\lceil P_k/(3/4) \rceil, P_k - 1]$, respective $[-\lfloor P_k/4 \rfloor, \lfloor P_k/4 \rfloor]$.

4.2.2 CRT Representation

The shift steps generalize easily to the CRT representation, as addition and subtraction can be performed independently on the individual residues. Figure 1b visualizes the scaling step (with base extension). To scale the residues $\tilde{x}_2, \dots, \tilde{x}_k$ after the ShiftUp in step 2 we use a simple technique described by Jullien [Jul78] and compute $\tilde{y}_i = (\tilde{x}_i - \tilde{x}_0) \cdot 2^{-1} \bmod p_i$, where 2^{-1} is the multiplicative inverse of 2 mod p_i and p_i is the i-th prime number.

Table 1: Step-by-step outputs of the scaling gadget for all inputs in $\mathbb{Z}_{P_2}$. x: Input value to the scaling function. $x^{\pm}$: Sign information of x. $x^{\uparrow}$ and $x^{\downarrow}$: Output of the ShiftUp and ShiftDown operations. b: Scaling operation after the ShiftUp and before the base extension. $\varphi(x)$: Residue representation of x. $\varphi^{-1}()$ leverages the CRT to reconstruct a value from its residue representation. y: Output of our base extension algorithm.

x	$x^{\pm}$	$\varphi(x)$	$x^{\uparrow}$	$x^{\pm\uparrow}$	$\varphi(x^{\uparrow})$	$\alpha = [0, b(\varphi(x))]$	$\varphi^{-1}(\alpha)$	$\varphi^{-1}(\alpha)^{\pm}$	y	$\varphi(y)$	$\varphi(y^{\downarrow})$	$y^{\downarrow}$	$y^{\pm\downarrow}$
0	0	[0, 0]	3	-3	[1, 0]	[0, 1]	4	-2	1	[1, 1]	[0, 0]	0	0
1	1	[1, 1]	4	-2	[0, 1]	[0, 2]	2	2	2	[0, 2]	[1, 1]	1	1
2	2	[0, 2]	5	-1	[1, 2]	[0, 2]	2	2	2	[0, 2]	[1, 1]	1	1
3	-3	[1, 0]	0	0	[0, 0]	[0, 0]	0	0	0	[0, 0]	[1, 2]	5	-1
4	-2	[0, 1]	1	1	[1, 1]	[0, 0]	0	0	0	[0, 0]	[1, 2]	5	-1
5	-1	[1, 2]	2	2	[0, 2]	[0, 1]	4	-2	1	[1, 1]	[0, 0]	0	0

To construct the base extension and compute $\tilde{x}_1$ we leverage the observation that $\tilde{x}_1$ either contributes exactly 0 or exactly $P_k/2$ to the sum of the reconstruction of $\tilde{x}$, since $\tilde{x} \equiv \sum_{i=1}^{k} \alpha_i \cdot \tilde{x}_i \bmod P_k$ where $\alpha_1 = P_k/2$ and $\tilde{x}_1 \in \mathbb{Z}_2$. Since all values must be positive after the scaling operation (see ring 3 in Figure 1a), we can use our sign gadget $\text{sign}_{0,1}$ to test whether $\tilde{x}_1 = 0$ or $\tilde{x}_1 = 1$ (see base extension in Figure 1b).

4.2.3 Garbling

This scaling operation is surprisingly cheap in our setting. For $P_k/2$ and $P_k/4$ we introduce k additional constant garbled inputs respectively (offline). The additions and subtractions of the shift operations are free in terms of needed ciphertexts. To compute $\tilde{y}_i$ for $i \geq 2$ in the scaling step, the number $\tilde{x} \bmod 2$ must first be projected into the space $\{0, \ldots, p_i - 1\}$ via a projection gate with two ciphertexts (see also Figure 1b). Then, the computation $(\tilde{x}_i - \tilde{x}_0)$ can be performed by several free subtractions. The multiplication with $2^{-1} \bmod p_i$ is a multiplication with a constant and thus also ciphertext-free. To finish the garbling of the scaling operation we need to garble a single sign gadget for the base extension. If $m_1, \ldots m_t$ represent the mixed-radix base used in the approximation of the sign gadget and $p_1, \ldots, p_k$ is a base of k primes used for the residue representation, the cost (in terms of number of ciphertexts) to garble the sign gadget is $\alpha = t \sum_{i=1}^{k} pi + 2k \sum_{i=2}^{n} m_i + 2n(k-1) + m_1$. Garbling the scaling gadgets results in $\alpha + (k-1)2$ ciphertexts.

4.3 System-level Architecture

Figure 2 shows a high level overview of Dash's modular system-level architecture and components involved in the garbling process (from the top to the bottom). Garbling the Dense and Conv2D operations is straight-forward, as these linear operations only depend on multiplication with a public constant (the weight) and addition. To add the bias to the resulting label, we model it as a constant circuit input. To garble the ReLU activation, we follow the approach of Ball et al. [BCM+19] see Subsection 3.4 and Section 7. The garbling of the Rescale layer is performed as described in Subsection 4.2.

4.4 Feature Set in Comparison

Here, we compare Dash's features and properties to other DPI schemes for ANNs. Table 2 shows that Dash compares positively to these other schemes concerning the provided feature set. We explain the compared features: An activation functions is pure if it is not modified to be suitable for MPC techniques. Off-the-shelf models are conventionally trained models without tuning for the DPI setting. Convenient model-loading means out-of-the-box model usage from frameworks like PyTorch or TensorFlow. CR abbreviates

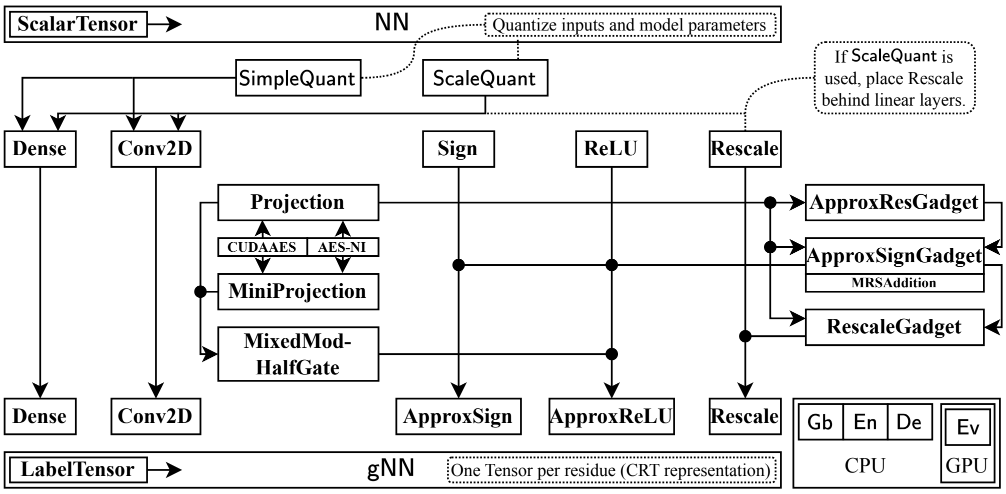


Figure 2: System-level architecture of Dash.

constant number of communications rounds and CV communication volume with respect to the number of layer-transitions — in the model architecture — from linear to non-linear and vice versa. The cryptographic building blocks from GNNP would also fully support Constant CRs and Constant CVs if the scheme supported the offline pre-processing model, and the implementation would sacrifice the streaming feature.

The categorization into Optional, Full, Limited and No support arises from the critical path through the inference process. If a scheme supports offline pre-computation, the categorization is based on the online phase. Optional support means that the Scheme can suppress a feature in favor of performance or other requirements. Thus, optional support is in general stronger than full support. Limited support means that a feature is only partially supported, e.g., Slalom and Goten can only accelerate the linear layers of a model on the GPU and must communicate with the TEE for all non-linear layers.

Many frameworks do not use TEEs, which may be since SGX (one of the most common TEEs) constituted a severe memory bottleneck until recently. This has led to paging artifacts and immense performance losses. However, in the recent versions of SGX, the enclave memory is much larger and paging becomes less of a problem. The security benefits of the TEE for Dash are discussed in the following section.

The GPU support is one of the outstanding features of Dash: Both linear and non-linear layers can be accelerated on a GPU without breaking the underlying security guarantees. Furthermore, due to the introduction of *LabelTensors*, Dash strongly benefits from the massive parallelity of garbled CNNs. Activation functions can either be computed with full accuracy, or as in many other works, in an approximated fashion to speed up the computation. While many frameworks support off-the-shelf models, meaning models that were trained conventionally without the DPI setting in mind, Dash goes a step further: Users can conveniently load their models in the ONNX format, which is widely supported by standard ML frameworks such as TensorFlow or PyTorch. Together with the GPU-support, Dash can be seen as a drop-in replacement for conventional and insecure inference engines in many existing ML applications.

Dash requires only a single round of online communication regardless of the depth of the ANN, or the number of alternating linear and non-linear layers. While gnnp [BCM+19] explicitly does not exploit this outstanding property in its implementation *fancy-garbling* and streams the GC to the evaluator during the online phase, Dash capitalizes on this property and thus significantly accelerates the online phase. Dash can evaluate the garbled ANN layer by layer without being slowed down by network communication at

Table 2: Feature Comparison. CR: Communication rounds, CV: Communication volume.

Scheme	DASH	DASH (w/o TEE)	GNNP [BCM+19]	SLALOM [TB19]	SECUREML [MZ17]	MINIONN [LJLA17]	CryptoNets [CBL+18]	DELPHI [MLS+20]	CRYPTGPU [TKTW21]	CRYPTFLOW [KRC+20]	MP2ML [BCD+20]	PIRANHA [WWP22]	SIMC [CGOS22]	MUSE [LMSP21]	GOTEN [NCW+21]	GAZELLE [JVC18]	DEEPSECURE [RRK18]
TEE	◐	-	-	●	-	-	-	-	-	●	-	-	-	-	●	-	-
GPU	◐	◐	-	○	-	-	-	○	●	-	-	●	-	-	○	-	-
Pure activation fun.	◐	◐	◐	●	-	-	◐	-	●	●	◐	●	●	●	●	●	○
Off-the-shelf models	◐	◐	●	●	●	●	◐	◐	-	○	●	●	○	○	●	●	○
Convenient model-loading	●	●	○	-	-	-	○	●	○	●	●	-	-	-	-	-	-
Constant CRs	●	●	○	-	-	-	●	-	-	-	-	-	-	-	-	-	-
Constant CV	●	●	○	-	-	-	●	-	-	-	-	-	-	-	-	-	-
Malicious security	●	○	○	●	-	-	●	-	-	●	-	-	◐	◐	●	-	-
Supports learning	-	-	-	-	●	-	-	-	●	-	-	●	-	-	●	-	-
Unlimited Input Owner(s)	●	-	-	●	-	●	●	-	-	-	-	○	-	-	-	-	●

◐ Optional, ● Full, ○ Limited, – No support

each transition from linear to non-linear layer (or vice versa). For a fixed input size (sum of inputs from all input providers) and a fixed CPM, DASH has a constant online communication volume regardless of ANN architecture or the number of input providers.

If TEE-Support like Intel SGX is provided and the use-case agrees with the TEE-assumption (see Subsection 4.7), DASH can optimize the OTs for communicating the garbled inputs to the garbling device away and can run the inference with more than two input providers. This distinguishes DASH from secret-sharing-based approaches, which are inherently limited in this aspect as they have to communicate and store a share for each input provider. Furthermore, trough leveraging the TEE DASH achieves malicious security and can thus enable the use of ML inference in security critical domains, where protocol participants are not trustworthy and semi-honest security is not sufficient.

If the hardware does not provide TEE support, then DASH does not achieve security against malicious attackers. However, all other DASH optimizations are preserved and DASH remains secure in the semi-honest outsourced inference scenario. In this scenario, DASH still beats the current SOTA scheme (see also [NC23]) GNNP for the outsourced inference case (GNNP also does not consider network overhead due to OTs in the evaluation).

Like other schemes, DASH does not support training, as this requires constant parameter changes, which does not fit well with offline pre-computation. Secure training with feasible computation requirements is an open research problem and not the focus of this work.

4.5 Use Cases

Using DASH guarantees several important security objectives, depending on the concrete scenario. All scenarios share a common structure with the following participant roles:

- The *model owner* who holds the ANN NN.
- The *input owner* who holds an input In to the ANN (for inference). Our framework supports multiple input owners that each contribute a part of the input.
- The *inference device*, a device aimed to compute the inference of an ANN.
- The *garbling device*, a device to garble the circuit and the inputs.
- The *result owner* who receives the inference result.

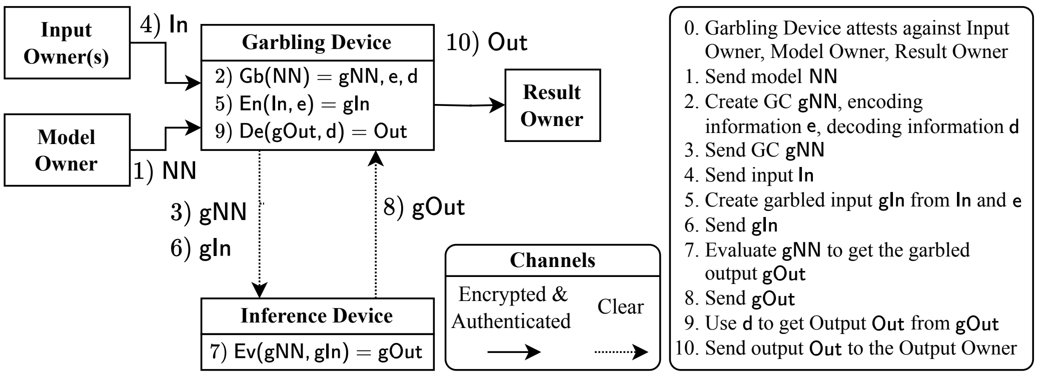

Input Owner(s)
4) In
Garbling Device
2) Gb(NN) = gNN, e, d
5) En(In, e) = gIn
9) De(gOut, d) = Out
10) Out
Result Owner
Model Owner
1) NN
3) gNN
6) gIn
8) gOut
Inference Device
7) Ev(gNN, gIn) = gOut
Channels
Encrypted & Authenticated
Clear
0. Garbling Device attests against Input Owner, Model Owner, Result Owner
1. Send model NN
2. Create GC gNN, encoding information e, decoding information d
3. Send GC gNN
4. Send input In
5. Create garbled input gIn from In and e
6. Send gIn
7. Evaluate gNN to get the garbled output gOut
8. Send gOut
9. Use d to get Output Out from gOut
10. Send output Out to the Output Owner

Figure 3: Example workflow of DASH. The first four steps can be pre-computed in an input-independent offline phase. Note that the inference device works on garbled data.

Dash runs in the typical outsourced inference scenario: A customer outsources its inference workload to a single server in a fire-and-forget manner: Sending input and output is the only communication during the computation in the online phase. Dash allows fine adjustments to the participant roles in this setting. Some examples of conceivable scenarios are:

- The classical outsourced inference scenario, where the model owner also controls the garbling device, the input owner also obtains the result as result owner, and the inference device is its own party.
- A more involved outsourced inference scenario with two different input owners.
- A disjoint setting where each participant is its own entity.

Compared to previous works like SLALOM, GOTEN and the implementation *fancy-garbling* of GNNP, we do not assume a co-location of different devices or parties (especially the garbling and the evaluation device must *not* be co-located). The typical workflow of DASH is illustrated in Figure 3.

Similar to Slalom [TB19] and GAZELLE [JVC18], we also split the computation into an *offline* and an *online phase*. In the offline phase (steps 1, 2, and 3), the model owner can already use the garbling device to prepare gNN. Since this is typically the bottleneck of the computation, moving it to the offline phase allows us to speed up the online phase significantly. In the online phase, the input owner performs the inference with its sensitive data In and profits of the pre-computation. We assume the messages in step 1, 4, and 10 to be sent over an authenticated and encrypted channel. The messages in step 3, 6 and 8 are protected through the garbling properties.

4.6 Security Objective

We distinguish three security objectives: *input privacy*, *integrity of the inference*, and *output privacy*. For many applications, the most sensitive information is the input In. Following the notion of Tramèr and Boneh [TB19], this security objective is called *input privacy*. We always guarantee input privacy, even if all other participants cooperate (including other input owners). Another essential property, called *integrity of the inference*, ensures that, at the end of the protocol, the result owner really obtains NN(In), i.e., the inference is computed correctly. GCs inherently guarantee the integrity of the computation and the inputs [BMR16]. However, in the case of a malicious model owner, we can't prevent the attacker from submitting a different network NN*. In this case, we will thus assume that the model owner first *commits* to NN in order to allow verification that the correct network was used. Analog to the input privacy, the output of the computation is also sensitive information. It should only be obtained by the result owner, so *output privacy* can be guaranteed. We will always guarantee output privacy.

While other works also aim to protect the model, many model extraction attacks (e.g., [CJM20, PMG+17, TZJ+16]) that reverse the architecture and the weights of a model from various sources, like input-output behavior, make model privacy an impossible goal for many scenarios. We thus do not provide it in our standard scheme. As we work in the outsourced inference setting, we always assume that the garbling device and the inference device do not collude.

4.7 Security of Dash

Our security analysis against attackers is built on two assumptions: the *garbling assumption*, which assumes that GCs are secure, i.e., they are private, oblivious, and guarantee authenticity [BMR16]. More formally, privacy guarantees that an attacker can not learn anything from $(\mathsf{gNN}, \mathsf{gIn}, d)$ except for the correct output Out. Obliviousness means that $(\mathsf{gNN}, \mathsf{gIn})$ do not reveal information about In. Finally, authenticity means that an attacker knowing $(\mathsf{gNN}, \mathsf{gIn})$ can not generate $\mathsf{g\tilde{O}ut} \neq \mathsf{gOut}$ such that $\mathsf{De}(\mathsf{g\tilde{O}ut}, \mathsf{d}) \neq \perp$. A long line of research has established that one can base this assumption on different well-established cryptographic assumptions. While this assumption guarantees security against a wide range of attacks, it does not prevent all attacks performed by malicious parties. To prevent the remaining attacks, we use our second assumption, called the *device assumption*, which guarantees that the garbling device acts as a trusted third party. After our security analysis, we will present two approaches on how to obtain the device assumption: one will be based on a TEE and the other will use purely cryptographic means. We do not make additional assumptions about the system or the behavior of other parties.

A quick inspection of Figure 3 shows that the security guarantees of GCs along with the device assumption guarantee security: Regarding the input (and output) privacy, by the device assumption, we only need to consider a scenario where some input owners, the model owner, and the inference device collaborate maliciously to obtain information about the input of an input owner or the output of the computation, as the garbling device acts as a trusted third party. The confidentiality of each input follows directly from the privacy property of GCs and the fact that the garbling device is trusted. Hence, given gNN and garbled inputs gIn, an attacker is unable to learn anything about the plain input. Similarly, nothing can be learned about the plain output without the decoding information d due to the obliviousness of the garbled circuit. Due to the device assumption, it is only sent to the result owner.

If the correct ANN is handed to the trusted garbling device, computation integrity again follows from the authenticity property of GCs and the fact the garbling device is trusted. An attacker cannot generate a manipulated output whose decoding is a valid output given the GC and garbled inputs. Hence, the computation result will always be an encryption of $\mathsf{NN}(\mathsf{In})$, as the garbling device is only able to work on the garbled inputs gIn presented by the input owner and the gNN. We only need to guarantee that the model owner does indeed hand out the correct model to the garbling device. As discussed above, we ensure that the model owner first needs to commit to NN, e.g., by publishing a hash of it. As the garbling device is trusted due to the device assumption, after having obtained the model, it can verify that the given network matches the commitment.

On the Device Assumption

In our security discussion above, we used the device assumption that guaranteed that the garbling device behaves honestly. We now discuss two ways to guarantee such a behavior.

In the first, purely cryptographic, way we consider the classical setting where the garbling device and the input owner are one entity. Input privacy and output privacy are trivially maintained by the privacy and the obliviousness of the GC. Finally, in order to prevent the garbling device from manipulating the circuit, we can either use the cut-and-choose approach [Cha82, LP15], make use of zero-knowledge proofs [GMW86], or use authenticated garbling schemes [NNOB12, WRK17]. For example, when using zero-knowledge proofs, all parties first need to commit to their private information (such as the inputs or the model) and to their randomness. Whenever a party now send some value to another party, it also sends a zero-knowledge proof along to convince the other parties that the sent message is consistent with the protocol, the earlier messages, the committed private information, and the committed randomness. While such approaches have been dismissed as purely theoretical in the past, the last decade has seen the design of very efficient and small zero-knowledge proofs, see e.g., [BBB+18, AHIV23, YSWW21]. These developments have been used to show that this approach does in fact allow for a relatively efficient implementation [ASH+20]. To guarantee a private exchange of the input and output information, *oblivious transfer* protocols [Rab05] need to be used. For a more in-depth discussion see [EKR18].

The second approach uses a TEE on the garbling device and assumes that it is secure. This assumption has been used already in different works, e.g., by Tramèr and Boneh [TB19]. While TEEs have been studied for a much shorter time period, there are certain robust designs that have withstood attacks, e.g. [CLD16, BGJ+19]. We stress here that even for the systems already in production, like Intel SGX or AMD SEV, the vast majority of attacks does not concern the cryptographic black-box guarantees, but make use of side-channel attacks. While TEEs are conceptually secure, side-channel attacks affect the implementation. Similarly, cryptographic primitives are also conceptually secure, but can be affected from side-channel attacks, if the implementation is not hardened. In protocol development, it is generally assumed that the underlying cryptographic primitives are implemented securely. It is therefore a reasonable assumption that the TEE implementation used for Dash also does not suffer from side-channel leakage. Due to the remote attestation feature, third parties can be assured that code and data inside an enclave are protected and that the code is executed as expected. However, this can be completely done in the offline phase and thus does not influence the performance of the online phase.

5 Implementation

Dash is implemented in C/C++ and CUDA 12.2 and uses the TEE assumption to guarantee security against malicious attackers, i.e., we assume that the TEE on the garbling device acts as a trusted third party. To parallelize CPU computations, we leverage OpenMP. We use the Linux Intel SGX SDK version 2.21 (with in-kernel drivers) to support the latest Intel Saphire Rapids Scalable Xeon CPUs for our TEE implementation. For high-quality randomness in the wire label generation, we use Intel's Digital Random Number Generator (DRNG). As a driver for our random-permutation-engine, we use the hardware-accelerated AES-NI instructions on the CPU and the OpenSSL AES (ECB mode) implementation on the CUDA-enabled GPU with T-Tables in constant memory. To send the wire labels made of $\mathbb{Z}_p$ elements to the AES-based fixed-key permutation, they first have to be compressed to 128-bit chunks. Like Ball et al. [BMR16], we use the Horner method for compression. The compression compress(l) of a label l with n components and wire modulus q is simply computed as compress$(l) = (\ldots(l_n q + l_{n-1})q + \ldots)q + l_1$.

5.1 Dash-as-a-Framework

From the perspective of a model owner, the only thing needed to use Dash is an ANN in the standard ONNX format. Then, the general procedure to garble the ANN is as follows, where encoding of weights and inputs into $\mathbb{Z}_{P_k}$ is handled automatically.

```
// 1. Step: Create circuit from
    model-file
auto circuit = load_onnx_model("path",
    q_method);
// Optional: Optimize quantization on
    data
circuit->optimize_quantization(crt_size,
    example_data);
// 2. Step: Garble quantized circuit,
    sign accuracy: ‘acc‘
auto gc = new GarbledCircuit(circuit,
    crt_size, acc);
// Optional: Move GC to GPU
gc->cuda_move();
```

(a) Garbling a CNN given as ONNX model-file.

```
// Step 1. Quantize input
auto q_in = quantize(input, q_method);
// Step 2. Garble input
auto g_in = gc->garble_inputs(q_in);
// Optional: Move q_in to GPU
auto g_dev_in =
    gc->cuda_move_inputs(g_in);
// Step 3: Evaluate GC on CPU
auto g_out = gc->cpu_evaluate(g_in);
// Or on GPU
gc->cuda_evaluate(g_dev_in);
auto g_out = gc->cuda_move_outputs();
// Step 4: Decode outputs
auto out = gc->decode_outputs(g_out);
```

(b) Evaluation of a GC.

Figure 4: C++-interface of Dash

1. Import the model as a circuit to Dash and quantize weights and biases using SimpleQuant or ScaleQuant. If SimpleQuant is used, optionally optimize quantization of the circuit with representative example data (the quantization constant is chosen based on a given CRT base P_k and the maximal computed value during the inference).
2. Garble the quantized circuit (and optionally move it to a GPU).

Independently of the chosen use case, the following steps must be performed to infer the plain output from an inference with the GC.

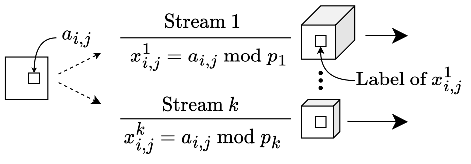
1. Quantize the inputs to integers.
2. Garble the inputs (optionally move them to a GPU).
3. Evaluate the GC on the garbled inputs (optionally move the result back to the host).
4. Decode the garbled outputs.

Like ANNs, our circuits and GCs consist of layers such as Dense, Conv2d, ReLU, Sign, Flatten, or Rescaling. All operations are implemented in a garbled and non-garbled variant to facilitate testing, experimentation, and quantization tuning. In addition to the ONNX model loader, users of Dash can also construct circuits directly in code with an intuitive interface inspired by the sequential models of PyTorch and TensorFlow. The interface of Dash's SGX implementation behaves analogously to the implementation without a TEE.

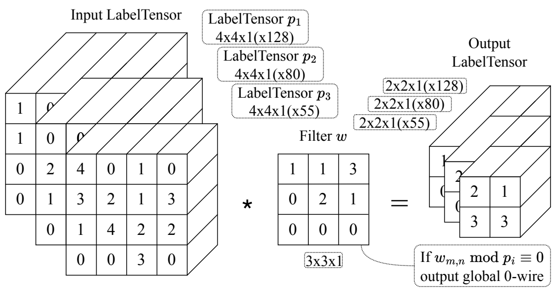
5.2 LabelTensors

In general, arbitrary arithmetic GCs are not particularly well-suited for the evaluation on CUDA-enabled GPUs, as they do not fit nicely into the concept of single-instruction-multiple-thread (SIMT). In CUDA threads are grouped into grids of blocks and blocks are executed in warps of 32 threads on the streaming multiprocessors of the GPU. As all threads of a warp share the same program counter, the parallel evaluation of heterogeneous gates, meaning the execution of threads with diverging control flows, will cause warp convergence, i.e. a serialization of those threads. Furthermore, the arithmetic GCs handled by Dash are neither strictly hierarchical nor uniform due to their global circuit wires and the modulus-based adaptive label length. To contribute enough entropy to the fixed-key AES permutation after packing, the labels with smaller modulus are longer (have more components) as the labels with larger modulus [BMR16]. This makes memory coalescing difficult, since labels of gates that can be executed in parallel are not necessarily consecutively arranged in the global memory of the GPU.

Nevertheless, CNNs typically have quite large homogeneous layers consisting of many similar operations, which allows for optimizations with regard to a GPU evaluation. We


$a_{i,j}$
Stream 1
$x^1_{i,j} = a_{i,j} \bmod p_1$
Stream k
$x^k_{i,j} = a_{i,j} \bmod p_k$
Label of $x^1_{i,j}$

(a) Leveraging LabelTensors in the CRT domain in independent streams on GPUs.


Input LabelTensor
LabelTensor p_1 4x4x1(x128)
LabelTensor p_2 4x4x1(x80)
LabelTensor p_3 4x4x1(x55)
Filter w
3x3x1
Output LabelTensor
2x2x1(x128)
2x2x1(x80)
2x2x1(x55)
If $w_{m,n} \bmod p_i \equiv 0$ output global 0-wire

(b) Conv2D operation over LabelTensors in Dash (for simplicity without bias, stride=1).

Figure 5: Visualization of the LabelTensor approach in Dash. The length l of a label with modul p_i is defined as $l = \lfloor 128/\log_2(p_i) \rfloor$, since a longer label can not be packed into a 128bit chunk which is needed in the permutation used for garbling.

introduce the notion of *LabelTensors* to handle the labels in arithmetic GCs for CNNs efficiently. LabelTensors in Dash are the counterpart of conventional tensors heavily used in machine learning frameworks like PyTorch or TensorFlow. The basic idea behind LabelTensors is to add another dimension to a conventional tensor to explicitly model the arithmetic GC labels. LabelTensors structure their values such that all labels of the same length lie along the tensor-width, -height and number of input-channels, consecutively in memory. Since all components of a label now lie consecutively in memory optimized CUDA kernels can coalesce memory accesses in warp granularity.

All garbled operations and utility functionalities of our implementation, such as label en-/decryption and label de-/compression for CPU and GPU are based on these LabelTensors and work in the typical SIMT-style of CUDA without any further abstraction directly over the corresponding memory areas (see Figure 5b for a visualization of the Conv2D operation). For examples of the encountered operations over the LabelTensors see Subsubsection 6.2.2. This optimization concept removes the need for expensive gate-object scheduling. In some sense, our approach can be seen as an arithmetical generalization of the JustGarble approach [BHKR13], which implements binary GCs with only memory blocks and indices and thus eliminates the need for expensive gate objects. As opposed to the implementation of gnnp (called *fancy-garbling*) [BCM+19], which generates a large overhead due to gate-object scheduling with only eight threads, LabelTensors enable Dash to efficiently leverage thousands of cores simultaneously on hardware accelerators like GPUs.

We use one LabelTensor per residue in the CRT representation and perform the operations on them in independent CUDA streams. This way, the operations over the residues of different CRT moduli and thus over labels of different length are processed in independent blocks on the streaming processors of the GPU and warp convergence is prevented (see also Figure 5a). The performance gain from the LabelTensor architecture was evaluated with several microbenchmarks, which will be presented in the next section.

6 Evaluation

Intel offers a trusted library version of OpenMP `libsgx_omp.a` for parallelization within SGX Enclaves. This library requires an OCALL and an ECALL to create a thread and an OCALL to wake up and pause threads and therefore has a considerable application-specific overhead compared to the conventional OpenMP library. For the model architectures in our evaluation, this overhead exceeds the performance gain achieved by our parallelization. Meaning, the usage of OpenMP inside the enclave is rendered useless for our model architectures. Thus, Dash only parallelizes CPU operations outside of the SGX enclave. However, the inference device is untrusted, and we can evaluate the inference in the online phase using the parallel CPU implementation without violating Dash's security properties. The server features an Nvidia Geforce RTX 4090 GPU to evaluate Dash's speedup.

6.1 Microbenchmarks

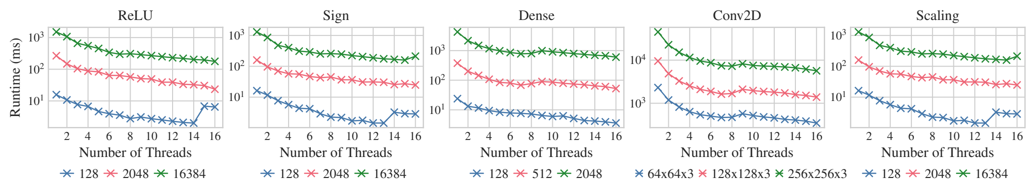
For the microbenchmarks, we garbled all layers with CPM P_8, meaning the CRT representation consists of the residues modulo the first eight prime numbers. Furthermore, we excluded host-device transfer overheads since the hidden layers in CNNs do not suffer from them during the online phase. To evaluate the scalability of the garbled CNN layers over Dash's LabelTensors on CPUs, we measured the runtimes for different input dimensions with 1-16 threads. The results in Figure 6a clearly show that every layer type, including the activation functions and our proposed scaling mechanism, benefit significantly from our parallelization-friendly LabelTensors. Apart from the transition from 8 to 9-11 threads (our CPU has eight physical cores), almost every additional thread brings a clearly measurable performance gain. The convolution operation benefits most from additional threads, but also, the other layers achieve speedups of up to an order of magnitude.

To evaluate the benefit of Dash's GPU extension, we measure the achieved speedup against the CPU implementation, leveraging 16 threads. As shown in Figure 6b, all garbled layers, even the activation layers and the scaling layer, achieve a speedup when evaluated on the GPU. Starting with an input size of 256, our scaling layer can benefit from the GPU's parallelism and achieves a speedup of up to 6x for 2^{14} inputs. Above a dimension of 256 inputs, the GPU implementation of the activation functions shows a speedup-factor of 1.3 and 1.1 for the 100% accurate ReLU respective sign activation functions. At 2^{14} inputs, the speedup grows to a factor of 5.8 respective 5.3, clearly outperforming the CPU implementation. Our implementation of the activation functions is based on the approximated sign gadget. For completeness, we also measured the speedup at a reduced precision. At 99% precision, our GPU implementation can achieve a speedup of up to one order of magnitude. Since most of the computational effort in our garbled CNNs lies in the linear layers and we replaced max-pooling with strided convolutions compared to gnnp, the speedup we achieve in CNN inference due to lower precision is negligible. In the following, our evaluation exclusively uses garbled activation functions with full precision. Nevertheless, the achieved speedup can make a difference if RAM-/GPU-Memory-Usage is a concern or in use cases in predictive modeling for logistic regression or small fully-connected ANNs. As expected, the garbled linear layers leverage the parallelism of the GPU best and achieve speedups of up to two orders of magnitude.

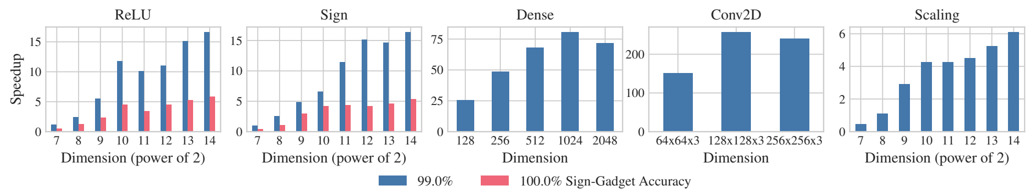
Figure 6c shows the memory usage of Dash's GPU implementation after the GC and the garbled inputs are transferred to the device memory. As expected, large inputs require the most memory, especially in the non-linear layers. If it is possible to compute with reduced accuracy, the memory requirement for the ReLU and Sign can be significantly reduced up to a factor of 7. To further reduce the compute and memory requirements, it would be interesting to explore the use of shorter CRT bases with larger moduli in future research. A shorter CRT base would result in less GC labels per circuit input and larger moduli in shorter labels with fewer components.

6.2 Model Benchmarks

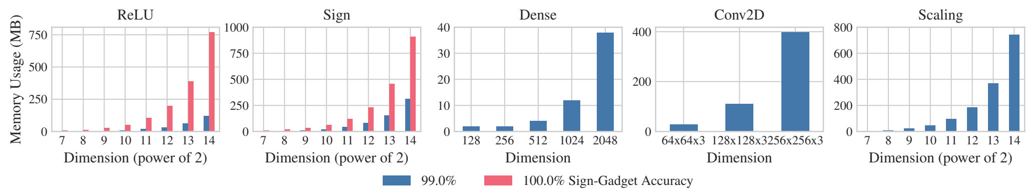
Dash supports the preprocessing paradigm, and we exclude the offline phase from our evaluation, as the cloud provider will pre-compute it before the customer data arrives.



(a) CPU-Scalability of DASH. The ReLU and Sign functions are computed with 100% accuracy.



(b) Speedup of DASH's GPU against its CPU implementation.



(c) GPU memory usage after the GC and the garbled inputs are transferred to the device memory.

Figure 6: Microbenchmarks of DASH. Conv2D uses 16 filters of size 4x4 and a stride of 2.

6.2.1 Model training

We trained all models with PyTorch over 100 epochs and selected the final model based on the minimal validation loss during the training. The models were trained on the MNIST and the CIFAR-10 data set. The MNIST data set consists of 70.000 black and white images of handwritten digits from zero to nine. 60.000 images serve as training data, and 10.000 images serve as test data. We excluded 5.000 images from the training data for validation. Each image is represented by 28×28 integer pixel values from the range [0,255]. The CIFAR-10 data set consists of 60.000 RGB-Images of size 32×32 with color values in the range [0, 255] and ten classes. There are 50.000 training images and 10.000 test images, and we excluded 5.000 images from the training data set for validation.

6.2.2 Model architectures

Similar to Ball et al. [BCM+19], we evaluate several model architectures that have been used for evaluation of previous DPI frameworks [MZ17, GDL+16, RRK18, LJLA17] (see Table 3). In some models, Ball et al. replace ReLU activations with tanh activations during training and those tanh with sign activations at inference time to maintain model performance despite quantization. Thanks to our garbled scaling gadget and the ScaleQuant mechanism, we do not need such replacements. Springenberg and Dosovitskiy et al. [SDBR15] observed that replacing max-pooling with strided convolution in modern CNNs leads to competitive or even better predictive power. Following their approach enables us to heavily reduce the memory footprint of our garbled CNNs compared to GNNP since garbling a single $\max(x, y) = x + \text{ReLU}(y - x)$ operation results in the same large ciphertext overhead as garbling the ReLU function, while garbling the convolution is ciphertext-free.

Table 3: Model architectures. R: ReLU, (a): dense layer with a outputs, (a, b, c, d): 2d convolution with a input-channel, b output-channel, filter size c and a stride of d.

ModelA:	$(128), R, (128), R, (10)$
ModelB:	$(1, 5, 5, 1), R, (5, 5, 3, 3), R, (5, 10, 3, 1), R, (10, 10, 3, 3), R, (100), R, (10)$
ModelC:	$(1, 5, 4, 2), R, (100), R, (10)$
ModelD:	$(1, 16, 6, 2), R, (16, 16, 6, 2), R, (100), R, (10)$
Modelf:	$(3, 32, 3, 1), R, (32, 32, 3, 1), R, (32, 32, 2, 2), (32, 64, 3, 1), R, (64, 64, 3, 1), R,$ $(64, 64, 2, 2), (64, 128, 3, 1), R, (128, 128, 3, 1), R, (10)$
ModelF:	$(3, 64, 3, 1), R, (64, 64, 3, 1), R, (64, 64, 2, 2), (64, 64, 3, 1), R, (64, 64, 3, 1), R,$ $(64, 64, 2, 2), (64, 64, 3, 1), R, (64, 64, 1, 1), R, (64, 16, 1, 1), R, (10)$

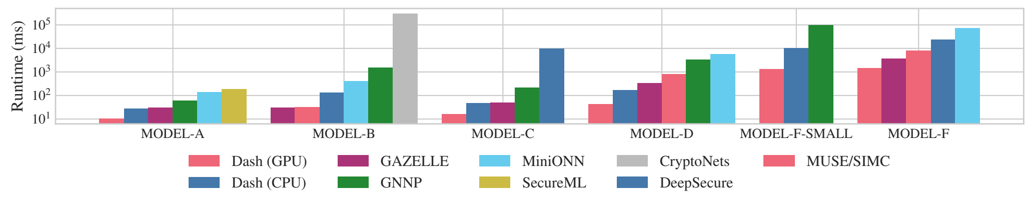


Figure 7: DASH's online model runtime compared to previous DPI frameworks (data is not available for all framework/models, e.g. because not all frameworks support all models).

MODEL-A to -D were trained on the MNIST and MODEL-f and -F on the CIFAR-10 dataset. During garbling of the Models A-F we use CPMs $P_8, P_9, P_9, P_8, P_7, P_7$, such that there are no overflows in the GC during runs on the test sets. We used SimpleQunat for MODEL-A to MODEL-D and ScaleQuant with $l = 5$ for MODEL-f and MODEL-F.

6.2.3 Model Performance

Figure 7 compares the online runtime of DASH to GNNP [BCM+19], MINIONN [LJLA17], CryptoNets [CBL+18], SECUREML [MZ17], GAZELLE [JVC18], DEEPSECURE [RRK18], MUSE [LMSP21] and SIMC [CGOS22] (MUSE and SIMC have the same online runtime). DASH's runtimes incorporate the theoretically determined communication overhead in the worst case in a 1 GBit/s network under 100% protocol overhead (see Subsection 6.3). Achieving malicious security compared to semi-honest security is typically accompanied by a significant performance penalty. DASH is, besides GNNP, the only one of the given frameworks that achieves full malicious security and still delivers the best runtime performance.

Compared to DASH, SECUREML supports training. However, the evaluation of MODEL-A shows that DASH is already over an order of magnitude faster in the case of small fully-connected ANNs. DEEPSECURE is also based on an optimized GC implementation, but in contrast to DASH and GNNP, it implements binary GCs. The results of MODEL-C show the advantage of arithmetic GCs for inference: DASH is over two orders of magnitude faster than DEEPSECURE. While MINIONN is significantly faster than CryptoNets, the evaluation of MODEL-B, -C, and -F clearly shows the speed advantage of DASH.

Table 4 shows a direct performance comparison of DASH against GNNP and GAZELLE. DASH's CPU and GPU implementation beat GNNP concerning all model architectures. In the case of the small fully-connected MODEL-A, DASH is 2x faster on the CPU and 6x faster on the GPU. For all other convolutional models, DASH achieves a 14- to over 100-times speedup against GNNP. Compared to GAZELLE, DASH's GPU implementations are always faster than GAZELLE, except for MODEL-B. For the largest evaluated MODEL-F, DASH is over two times faster than GAZELLE, showing its good scalability.

Table 5 reports the accuracy of our garbled models compared to GNNP. DASH beats

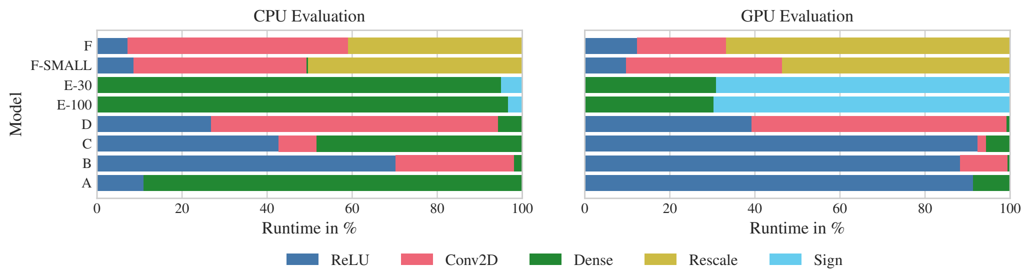


Figure 8: Runtime distribution over layer types and model architectures.

GNNP for each model. For the larger MODEL-f, the power of DASH's ScaleQuant quantization becomes clear, DASH's accuracy is almost 12% higher compared to GNNP.

Table 4: Comparison of online runtimes (in ms). GNNP does not incur communication overhead, GAZELLE is not malicious security.

Model	A	B	C	D	f	F
CPU	27	132	46	169	10263	23959
GPU	**10**	32	**16**	**42**	**1332**	**1443**
GNNP	60	1520	210	3340	97000	-
GAZELLE	30	**30**	50	329	-	3560

Table 5: Comparison of the achieved model accuracy on the test-set in % using DASH and GNNP.

Model	A	B	C	D	f
DASH	**97.76**	**96.73**	**98.10**	**98.84**	**85.67**
GNNP	96.80	86.72	97.21	96.44	73.74

6.2.4 Runtime Distribution

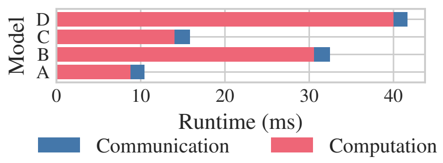
Figure 8 shows what portion of the runtime DASH spends in which layer types. As expected, GPU acceleration shifts the distribution such that the linear portions are minimized, and the non-linear portions account for most of the runtime cost. For optimization, accelerating the activation functions on the GPU even further would be particularly interesting.

For larger models (see F models), the ScaleQuant mechanism is needed to maintain their predictive power. In this case, our garbled scaling dominates the runtime cost. Currently, our scaling gadget only supports scaling with two because otherwise, our base extension mechanism is not able to recover the lower residues. Since the F models use ScaleQuant with $l = 5$ (rescaling by 2^{-5}), the scaling gadget must be applied five times. For further research, it would be interesting to look for ways to enable base extension for residuals of larger CRT moduli to minimize the rescaling overhead.

6.3 Communication

We evaluate DASH's online communication overhead in a theoretical model and assume the worst case, where the input owner(s), the garbling device, the inference device, and the result owner are separate parties that all communicate over a network. This way, we show that DASH outperforms the given DPI schemes in all possible use cases. We incur costs for communicating the plain inputs In, the garbled inputs gIn, the garbled outputs gOut, and the plain outputs Out. We assume that all GC labels are compressed into 128-bit for communication. Plain inputs use a 16-bit, and plain outputs a 64-bit data type. For fairness, we assume a large protocol overhead of 100% regarding the communication.

Figure 9b compares DASH's communication volume to MINIONN and GAZELLE. DASH's online communication volume is constant for a constant CPM and input dimensions. For a



(a) Distribution of the online workload.

(b) Online communication volume (MB) of Dash and Gazelle (single input).

Model	A	B	C	D	f	F
Dash	**0.1**	**0.1**	**0.1**	**0.1**	**0.4**	**0.4**
MiniONN	43.8	12.2	-	636.6	-	6226
Gazelle	0.5	0.5	2.1	22.5	296	-

Figure 9: Online Communication Volume.

CPM P_k, the volume of the garbled input is $k \cdot \text{size}(\mathsf{In}) \cdot 128$ bits, and for the garbled output $k \cdot \text{size}(\mathsf{Out}) \cdot 128$ bits. Typical DPI approaches that combine different MPC techniques for linear and non-linear layers require one communication round for each non-linear layer [CL01, BOP06, OPB07, BFL+11, MZ17, LJLA17, RWT+18, MR18, MLS+20]. Frequent communication slows down the computations because the inference device has to wait for a response before computing the next layer. Dash requires only a single round of communication between the garbling device and the inference device, regardless of the depth of the ANN. These advantages can be seen in Figure 9a, which shows the distribution of communication and computation over the entire runtime. The share of communication work in the total runtime decreases with the model size from 15% for MODEL-A over 5% for MODEL-D to less than 1% for MODEL-F.

7 Conclusion

We present a framework that offers ML inference with security against a malicious attacker by adopting optimized arithmetic GCs. The introduction of LabelTensors enables Dash to efficiently accelerate the inherent parallelism of ANNs with parallel hardware such as multi-core CPUs and large, massively parallel GPUs. Compared to GNNP, leveraging a TEE allows Dash to host inference applications with more than two input providers. With a fixed input dimension, the emerging communication volume and the memory requirement on the inference device remain constant regardless of the number of input providers. Independent of the number of alternating linear and non-linear layers, Dash requires only one round of communication between the garbling and the inference device. As a result, Dash achieves state-of-the-art performance for all evaluated model architectures. Dash can run existing models without retraining and does not require knowledge about GCs.

We exhibit a large feature comparison to many previous approaches and a thorough performance evaluation based on both micro-benchmarks and real-life models to show that Dash outperforms previous works both in regard to resource consumption (runtime and communication) and feature set. This work provides a relevant step towards making secure inference fast and easy to use, which will aid a broader adoption of proposed protection mechanisms, resulting in making secure inference accessible to a wide public audience.

Acknowledgments

We thank the anonymous reviewers for their useful feedback. This work has been supported by the BMBF through the project AnoMed.

References

[AHIV23] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: lightweight sublinear arguments without a trusted setup. *Des. Codes Cryptogr.*, 91(11):3379–3424, 2023.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 120–129. IEEE Computer Society, 2011.

[ASH$^+$20] Jackson Abascal, Mohammad Hossein Faghihi Sereshgi, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Is the classical GMW paradigm practical? the case of non-interactive actively secure 2pc. In *CCS*, pages 1591–1605. ACM, 2020.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society, 2018.

[BCD$^+$20] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. MP2ML: a mixed-protocol machine learning framework for private inference. In *ARES*, pages 14:1–14:10. ACM, 2020.

[BCM$^+$19] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled neural networks are practical. *IACR Cryptol. ePrint Arch.*, 2019:338, 2019.

[BFL$^+$11] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE Trans. Inf. Forensics Secur.*, 6(2):452–468, 2011.

[BGJ$^+$19] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. SANCTUARY: arming trustzone with user-space enclaves. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019.* The Internet Society, 2019.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 478–492. IEEE Computer Society, 2013.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM, 1990.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 565–577. ACM, 2016.

[BOP06] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In Sviatoslav Voloshynovskiy, Jana Dittmann, and Jessica J. Fridrich, editors, *Proceedings of the 8th workshop on Multimedia & Security, MM&Sec 2006, Geneva, Switzerland, September 26-27, 2006*, pages 146–151. ACM, 2006.

[CB19] Joseph I. Choi and Kevin R. B. Butler. Secure multiparty computation and trusted hardware: Examining adoption challenges and opportunities. *Secur. Commun. Networks*, 2019:1368905:1–1368905:28, 2019.

[CBL$^+$18] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *CoRR*, abs/1811.09953, 2018.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

[CGOS22] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. SIMC: ML inference secure against malicious clients at semi-honest cost. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 1361–1378. USENIX Association, 2022.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203. Plenum Press, New York, 1982.

[CJM20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 189–218. Springer, 2020.

[CL01] Yan-Cheng Chang and Chi-Jen Lu. Oblivious polynomial evaluation and oblivious neural learning. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 369–384. Springer, 2001.

[CLD16] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 857–874. USENIX Association, 2016.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Found. Trends Priv. Secur.*, 2(2-3):70–246, 2018.

[GAGN15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1737–1746. JMLR.org, 2015.

[GDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016.

[GKWY20] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 825–841. IEEE, 2020.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 567–578. ACM, 2015.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *FOCS*, pages 174–187. IEEE Computer Society, 1986.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings.* USENIX Association, 2011.

[HMSG13] Nathaniel Husted, Steven A. Myers, Abhi Shelat, and Paul Grubbs. GPU and CPU parallelization of honest-but-curious secure two-party computation. In Charles N. Payne Jr., editor, *Annual Computer Security Applications Conference, ACSAC '13, New Orleans, LA, USA, December 9-13, 2013*, pages 169–178. ACM, 2013.

[HP94] CY Hung and B Parhami. An approximate sign detection method for residue numbers and its application to rns division. *Computers & Mathematics with Applications*, 27(4):23–35, 1994.

[HSS+18] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *CoRR*, abs/1803.05961, 2018.

[HZG+18] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Max Augustin, Michael Backes, and Mario Fritz. Mlcapsule: Guarded offline deployment of machine learning as a service. *CoRR*, abs/1808.00590, 2018.

[Int21] Intel. Intel software guard extensions developer reference for linux* os, 2021.

[Jul78] Graham A. Jullien. Residue number scaling and other operations using ROM arrays. *IEEE Trans. Computers*, 27(4):325–336, 1978.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1651–1669. USENIX Association, 2018.

[Kap16] David Kaplan. AMD x86 memory encryption technologies. In *USENIX*, Austin, TX, August 2016. USENIX Association.

[KRC$^+$20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 336–353. IEEE, 2020.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 285–300. USENIX Association, 2012.

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 619–631. ACM, 2017.

[LLD$^+$22] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. Design and verification of the arm confidential compute architecture. In Marcos K. Aguilera and Hakim Weatherspoon, editors, *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 465–484. USENIX Association, 2022.

[LLP$^+$19] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos, editors, *The 25th Annual International Conference on Mobile Computing and Networking, MobiCom 2019, Los Cabos, Mexico, October 21-25, 2019*, pages 46:1–46:17. ACM, 2019.

[LMSP21] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2201–2218. USENIX Association, 2021.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007.

[LP15] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. *J. Cryptol.*, 28(2):312–350, 2015.

[LPS08] Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *Security and Cryptography for Networks, 6th International Conference, SCN 2008, Amalfi, Italy, September 10-12, 2008. Proceedings*, volume 5229 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 2008.

[MAB$^+$13] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In Ruby B. Lee and Weidong Shi, editors, *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013*, page 10. ACM, 2013.

[MLS$^+$20] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2505–2522. USENIX Association, 2020.

[MPS15] T Malkin, V Pastro, and A Shelat. The whole is greater than the sum of its parts: Linear garbling and applications. In *Workshop talk at Securing Computation Workshop in Berkley*, 2015.

[MR18] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018.

[MTH22] Fan Mo, Zahra Tarkhani, and Hamed Haddadi. Sok: Machine learning with confidential computing. *CoRR*, abs/2208.10134, 2022.

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017.

[NC21] Lucien K. L. Ng and Sherman S. M. Chow. Gforce: Gpu-friendly oblivious and rapid neural network inference. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2147–2164. USENIX Association, 2021.

[NC23] Lucien K. L. Ng and Sherman S. M. Chow. Sok: Cryptographic neural-network computation. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 497–514. IEEE, 2023.

[NCW$^+$21] Lucien K. L. Ng, Sherman S. M. Chow, Anna P. Y. Woo, Donald P. H. Wong, and Yongjun Zhao. Goten: Gpu-outsourcing trusted execution of neural network training. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in*

Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pages 14876–14883. AAAI Press, 2021.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.

[OPB07] Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP J. Inf. Secur.*, 2007:1–11, 2007.

[OSF$^+$16] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 619–636. USENIX Association, 2016.

[PMG$^+$17] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 506–519. ACM, 2017.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2005:187, 2005.

[RRK18] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 2:1–2:6. ACM, 2018.

[RSC$^+$19] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: xnor-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1501–1518. USENIX Association, 2019.

[RWT$^+$18] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 707–721. ACM, 2018.

[SDBR15] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[SMF21] Muhammad Usama Sardar, Saidgani Musaev, and Christof Fetzer. Demystifying attestation in intel trust domain extensions via formal verification. *IEEE Access*, 9:83067–83079, 2021.

[SS08] Ahmad-Reza Sadeghi and Thomas Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, volume 5461 of *Lecture Notes in Computer Science*, pages 336–353. Springer, 2008.

[TB19] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[TGS$^+$18] Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. Privado: Practical and secure DNN inference. *CoRR*, abs/1810.00602, 2018.

[TKTW21] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. Cryptgpu: Fast privacy-preserving machine learning on the GPU. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1021–1038. IEEE, 2021.

[TZJ$^+$16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 601–618. USENIX Association, 2016.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *CCS*, pages 21–37. ACM, 2017.

[WWP22] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A GPU platform for secure computation. In *USENIX Security Symposium*, pages 827–844. USENIX Association, 2022.

[XBF$^+$14] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data. *CoRR*, abs/1412.6181, 2014.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *CCS*, pages 2986–3001. ACM, 2021.

[ZHC$^+$20] Fan Zhang, Warren He, Raymond Cheng, Jernej Kos, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. The ekiden platform for confidentiality-preserving, trustworthy, and performant smart contracts. *IEEE Secur. Priv.*, 18(3):17–27, 2020.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings,* *Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2015.

Appendix

Half-Gate Generalization Ball et al. [BCM+19] introduced a mixed-modulus multiplication gate. We denote the labels on the input-wires x, y with semantic a, b by l_x^a, l_y^b and the point-and-permute value of l_y^0 by r. Hence, the point-and-permute value of l_y^b is given by $(b + r) \bmod p$. The key idea is a reformulation of the multiplication into two gadgets $a \cdot (b + r)$ (the evaluator half gate) and $a \cdot r$ (the garbler half gate). Since r is the color value of the base label of the y-wire, the garbler already knows r at garbling time, and the evaluator knows $b + r$ at evaluation time.

The garbler selects a random base label l_u^0 for the garbler HG and performs $\mathrm{EN}_{l_x^0+aR}(l_u^0 + arR)$ for all $a \in \mathbb{Z}_p$. For the evaluator HG, the garbler selects a random base label l_v^0 and performs $\mathrm{EN}_{l_y^0+bR}(l_v^0 - (b + r)l_x^0)$ for all $b \in \mathbb{Z}_p$. The evaluator knows the labels of both inputs and decrypts two of the above ciphers accordingly. Knowing $b + r$ and $l_x^0 + aR$, he can choose $l_v^0 - (b + r)l_x^0 + (b + r)(l_x^0 + aR) = l_v^0 + (b + r)aR$ and the product-output is $l_v^0 + (b + r)aR - (l_u^0 + arR) = l_v^0 - l_u^0 + abR$ (considering $l_v^0 - l_u^0$ as the base label).

Mixed-Modulus Half-Gate Ball et al. [BCM+19] show how to reduce the mixed modulus multiplication cost to roughly $q + p + 1$ ciphertexts. To encrypt the evaluator HG, the garbler uses an input label $l_y^0 + bR'$ with modulus q. Instead of p ciphertexts, the evaluator HG results in q ciphertexts. Since $b \in \mathbb{Z}_q$, the corresponding color value is also from $\mathbb{Z}_q$. Now, we have to preserve the known input $b + r \in \mathbb{Z}_p$ of the evaluator HG. Therefore, Ball et al. [BCM+19] propose to leverage a single projection gate of q very short ciphertexts, encrypting $b + r$ for all possible b. For the multiplication $\mathrm{ReLU}(a) = \mathrm{sgn}(a) \cdot a$, we can pack all of these ciphertexts into 128 bits. Garbling the evaluator HG for all $b \in \mathbb{Z}_q$ results in the ciphertexts $\mathrm{EN}_{l_y^0+bR'}(l_v^0 - (r + b) \cdot l_x^0)$ and $\mathrm{EN}_{l_y^0+bR'}(r + b)$.

Approximated Garbled Sign The garbled sign function $\mathrm{sgn} : \mathbb{Z}_{P_k} \to \{0, 1\}$ of Ball et al. [BCM+19] expects $\mathbb{Z}_{P_k}$-values and interprets the first half of the ring as negative and the other half as positive numbers, i.e., $\mathrm{sgn}(x) = 0$ if $x < P_k/2$ and $\mathrm{sgn}(x) = 1$ if $x \geq P_k/2$.

The concept[2] is based on the CRT, which describes the reconstruction of the value $x \in P_k$ from its residue representation $[\![x]\!]_{P_k} = (x_1, \dots, x_k)$, where $x_i = x \bmod p_i$: $x \equiv \sum_{i=1}^{k} A_i^{-1} \cdot A_i \cdot x_i \bmod P_k$, with $A_i = \frac{P_k}{p_i} \equiv \sum_{i=1}^{k} \alpha_i \cdot x_i \bmod P_k$. Hence, the sign function can be computed just regarding the fractional part of the last summand by computing $\alpha_i x_i / P_k$ and rounding it to $1/M$ for some M. This approximation can be represented as $\mathbb{Z}_m$-wire and the the error of a k term sum is limited by $k/2M$. Hence, for $M > kP_k/2$, the result is correct. In fact, this $\mathbb{Z}_m$-wire, this is represented as a bundle of wires using a mixed-radix representation (see [BCM+19]). In other situations, correctness is not needed, and M is a trade-off parameter between precision and garbling cost.

Garbled Mixed-Radix Addition For use in the approximated garbled sign function, Ball et al. [BCM+19] introduced a fast *mixed-radix addition.* Consider the summation of $k = 3$ values represented in mixed-radix representation $\mathbb{Z}_{D_n} \cong (\mathbb{Z}_{d_1} \times \ldots \times \mathbb{Z}_{d_n})$ (associated with the integers $\{0, \dots, D_n - 1 = (\prod_i d_i) - 1\}$ and d_1 being the most significant digit). The operation first computes $s = x + y + z + c_i^{\mathtt{in}}$, where x, y, z are the values of the $\mathbb{Z}_{d_i}$-wires and $c_i^{\mathtt{in}}$ is the carry-input. Then, $x, y, z, c_i^{\mathtt{in}}$ are cast using four projection gates

[2]This general approach also appears in earlier works like [HP94].

to $\mathbb{Z}_{3d_i+c_i^{\texttt{max}}-1}$ and all input values are added mod $3d_i + c_i^{\texttt{max}} - 1$. Finally, the carry-out $c_{\texttt{out}} = \lfloor \frac{x+y+z+c_i^{\texttt{in}}}{d_i} \rfloor$ is computed using an unary gate.