DPWatch: A Framework For Hardware-Based Differential Privacy Guarantees

Pawan Kumar Sanjaya*‡, Christina Giannoula*‡, Ian Colbert†, Ihab Amer†, Mehdi Saeedi†, Gabor Sines†, and Nandita Vijaykumar*‡
*University of Toronto, ‡ Vector Institute, †Advanced Micro Devices Inc

***Abstract*—Differential privacy (DP) and federated learning (FL) have emerged as important privacy-preserving approaches when using sensitive data to train machine learning models. FL ensures that raw sensitive data does not leave the users' devices by training the model in a distributed manner. DP ensures that the model does not leak any information about an individual by *clipping* and adding *noise* to the gradients. However, real-life deployments of such algorithms assume that the third-party application implementing DP-based FL is trusted, and is thus given access to sensitive data on the data owner's device/server. In this work, we propose DPWatch, a hardware-based framework for ML accelerators that enforces guarantees that a third party application cannot leak sensitive user data used for training and ensures that the gradients are appropriately noised before leaving the device. We evaluate DPWatch on two accelerators and demonstrate small area and performance overheads.**

***Index Terms*—Hardware support, Federated learning, Differential privacy, Accelerators**

I. Introduction

Protecting the privacy of sensitive user data in machine learning training is of critical importance in many domains such as healthcare and finance. For example, training a highly accurate model for medical diagnosis requires the use of sensitive medical records of patients from multiple hospitals. Similarly, sensitive user data on mobile devices are used to train models for text prediction, facial recognition, voice recognition, etc. Federated learning (FL) [1] and differential privacy (DP) [2] have emerged as important techniques to safeguard the privacy of users while using sensitive data for training an ML model. FL enables training a shared ML model using sensitive data from various clients, but trains a copy of the shared model directly on the device/server where the sensitive training data is located. Only the gradients are shared with the central server for aggregation. FL helps address regulations such as GDPR [3] which constrain where data should be stored.

FL alone is insufficient to ensure the privacy of sensitive data. Prior works [4], [5] demonstrate that it is possible to extract sensitive user information from the outputs of the trained model during inference. Differential privacy (DP) [2] techniques can be leveraged in addition to FL to provide formal privacy guarantees. DP algorithms add Gaussian noise to the gradients and clips them before they are used to update the model weights. DP-enabled FL ensures that a global model can be trained without any sensitive user data leaving the owners' devices and formally ensuring that it is extremely difficult to infer the presence of a user or record in the dataset using the outputs of the learned model.

FL and DP together provide strong privacy guarantees; however, existing ML training frameworks that incorporate FL and DP still require trusting a third-party application with access to sensitive data and trusting that the application properly executes the DP-based FL algorithm. As a result, the training application is granted unrestricted access to the sensitive user data on the owner's device, without any guarantees that there is no malicious use of the data by the application. For example, for the guarantees of DP-enabled FL to be met, we must ensure that (a) the sensitive user data is only used to calculate gradients for the global model; and (b) the gradients are appropriately transformed using DP algorithms (*i.e.*, noising and clipping) before being sent to the central server that holds the global model.

It is challenging to ensure any DP-enabled FL application meets these requirements before providing access to sensitive user data. The application performs numerous computations during the training process using sensitive user data. Sharing any resulting or intermediate data can violate privacy guarantees. Identifying any leakage of sensitive information is a non-trivial task. Static or dynamic program analysis techniques [6], [7] can be used to identify information leakage; however, this requires access to the source code, which may not be possible in real deployments. Many tools [6] often require restrictions on how the program is written and on the programming language or framework used. Software frameworks that are designed to allow limited access and operations on data can potentially address these issues, but this limits the kind of applications that can be supported (*e.g.*, PINQ [8] does not support ML). Alternatively, *local* differential privacy can be used, where the user data is noised *before* giving the training application access to it. However, this significantly impacts the accuracy of the model [9].

In this work, we propose DPWatch, a hardware-based framework for ML accelerators that guarantees that sensitive data is used safely in DP-enabled ML. DPWatch comprises two key components. First, we propose a lightweight tagging mechanism that can identify and track the noised gradients in hardware to flag data that is safe to leave the device. Any data that is not marked as safe by DPWatch is prevented from being shared out of the device. Second, we implement a hardware noising module that ensures that gradients are noised to satisfy DP guarantees. Only gradients that are noised by this module are tagged as safe for sharing. This ensures that the application developer can only use DP-based FL algorithms to share any information computed using sensitive user data.

We model DPWatch in the Scale-sim [10] simulator for ML systolic array-based accelerators configured as the Genesys [11] and TPU [12] simulators. Our evaluations on popular ML models demonstrate negligible performance slowdown and area overhead of on average 0.35% and 0.08%, respectively. This letter makes the following contributions:

- We motivate the need for additional privacy protection mechanisms in real-life deployments of DP-enabled FL. We propose DPWatch, the first hardware-based framework for ensuring DP guarantees in ML accelerators.

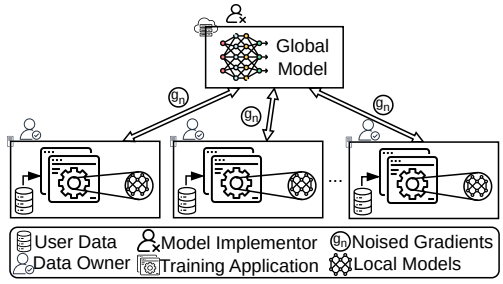

Fig. 1. Overview of federated learning.

- We propose a lightweight tagging mechanism to track these gradients and only allow them to be shared over the network.
- We use a hardware noising module to add noise and identify the noised gradients.
- We evaluate DPWatch in various ML workloads on two accelerators and show small performance and area overheads.

II. Background and Goal

Federated learning (FL): Fig. 1 provides an overview of FL. FL enables training a shared model without sensitive training data leaving the data owners' systems. In an FL setting, each client (*i.e.*, data owner) is sent a copy of the global model, and the clients train the model using their local data for multiple iterations (training round). At the end of each training round, the client sends only the model gradients to the FL server that aggregates the updates from multiple clients. The FL server then sends the clients a new copy of the global model for the next round of training.

Differential privacy (DP) in machine learning (ML): DP [2] aims to prevent malicious actors from inferring the original training data record(s) or the presence of a particular training record(s) in a dataset using the output of a trained ML model. DP aims to post-process outputs of the computation such that the final output does not change significantly with the addition or omission of a single record [2]. Post-processing techniques for DP involve adding noise sampled from a distribution such as the Gaussian. DP in ML is enabled by modifying the gradient descent algorithm for training. Instead of using the gradients (g) calculated for a batch, the ℓ_2-norm of the gradients is clipped to a threshold (C). Gaussian noise with a standard deviation (σ) proportional to C and a user-defined scaling factor (z) is added to the gradients (Eq. 1). Model weights are updated using the noised gradients ($\hat{g}$).

$$\hat{g} = S + \mathcal{N}(0, \sigma^2), \text{where } \sigma = C \cdot z$$
$$\text{and } S = \min\left(1, \frac{C}{\|g\|_2}\right) \cdot g \tag{1}$$

An alternative approach [13] that is used to incorporate DP in a federated setting involves clipping the values of each element of the gradient to C, followed by adding Gaussian noise (Eq. 2) to compute the noised gradients ($\hat{g}$).

$$\forall g_i \in < g_1, g_2, ..., g_k >,$$
$$\hat{g_i} = \min\left(1, \frac{C}{|g_i|}\right) \cdot g_i + \mathcal{N}(0, \sigma^2) \tag{2}$$

Goal: Our goal in this work is to enable the following guarantees in hardware: First, there is no leakage of sensitive training data during DP-based ML training. Second, the gradients that are shared out of the device are noised appropriately. To ensure that the noised gradients are DP, the gradients must be clipped according to Eq. 2. If these conditions are satisfied, an individual can allow third-party applications access to sensitive data, while ensuring that the guarantees provided by DP and FL are met.

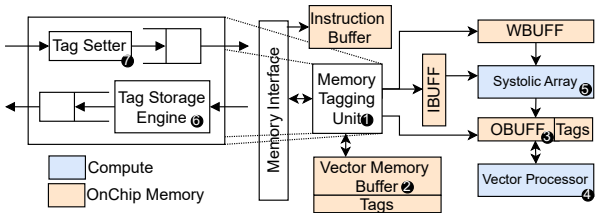

Fig. 2. Block diagram of a systolic array-based accelerator (configured similar to the Genesys [11] accelerator) with DPWatch.

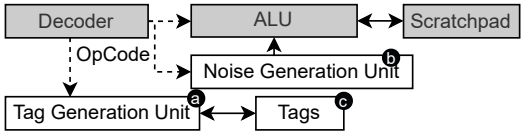

Fig. 3. Block diagram of the vector processor in the accelerator. The modules introduced by DPWatch are indicated using unshaded boxes.

III. Approach: A Framework for Hardware-Based Differential Privacy Guarantees

We propose DPWatch, a framework for ML accelerators that uses hardware mechanisms to identify data that can safely leave the user's device, *i.e.*, gradients that have been appropriately noised. Thus, the user can safely disallow any third-party application from moving out any data that are *not* the noised gradients as identified (tagged) by DPWatch.

A. Key Ideas and Components

Lightweight Information Flow Tracking. DPWatch uses *information flow tracking (IFT)* to monitor how the sensitive user data is used by the application. IFT [14] is a technique that uses metadata to analyze how data flows through the processor and the main memory. Thus, IFT can be used to track both, data that is *sensitive* (*i.e.*, cannot be shared without the DP transformations) and data that was generated using sensitive data (and are thus also sensitive). IFT however typically incurs non-trivial performance and area overheads from storing and accessing metadata [14]. DPWatch leverages the restricted programmability and dataflow common in ML accelerators to design a low-overhead tagging mechanism for accelerators such as TPU [12] and Genesys [11].

In ML applications, only the noised gradients are safe to be shared out of the device. The key idea behind DPWatch's tagging mechanism is to leverage this to tag only the results of the noising operation as *safe*, while all other data is always assumed to be sensitive. Thus, the host system can prevent any data that is not marked as safe by DPWatch from being shared out of the device. This approach has a lower overhead than traditional IFT systems [14] as the tags of the source operands need not be fetched to determine the tag of the result. This approach also allows certain privacy-preserving operations on the noised gradients themselves within the accelerator. For example, gradients across multiple iterations are aggregated in each training round.

Noising Module: We implement a low overhead programmable module that is used to efficiently noise the gradients within the ML accelerator. This module is programmed with new instructions and is seamlessly integrated into DPWatch's tagging mechanism. This module ensures that gradients are appropriately noised as per the required DP guarantees. For the gradients to be tagged as "safe", the application must use this module to noise the gradients.

B. Detailed Design

Fig. 2 illustrates our design using an ML accelerator similar to the Genesys [11] accelerator, which consists of 2 compute

elements: a systolic array (❺) with 64k processing elements for multiply-and-accumulate operations and a 32 lane wide vector processor (VPU) (❹) for other operations. The vector processor uses software-managed buffers (❷ & ❸) which can be directly addressed without registers. Fig. 3 illustrates the changes to the vector processor.

Noising Module: We add a noise generation unit (ⓑ) in the vector processor to generate Gaussian noise. It is based on the Box-Muller transform [15], considered the best approach to balance accuracy vs. area [16]. A custom instruction (*add-noise*) is introduced to add noise (as shown in Eq. 3) to a value from the source location (S). The standard deviation for sampling is calculated using configuration registers (z, C) that store the noise scaling factor and the clipping threshold.

$$S + \max\left(\left|\frac{S}{C}\right|, 1\right) \cdot \mathcal{N}\left(0, \sigma^2\right), \text{ where } \sigma = C \cdot z \quad (3)$$

In order to provide DP guarantees according to Eq. 2, we must ensure in hardware that the application clipped the gradients correctly *i.e.*, $|S| \leq C$. If $|S| \leq C$, then the noising operation is equivalent to Eq. 2. However, if the gradients are not clipped properly, the noising operation protects the privacy of the users by destroying the information captured by the gradients. This is achieved by scaling the noise sampled from the original distribution (with a standard deviation of σ) with $|S/C|$. This operation is equivalent to generating a random number from a Gaussian distribution with mean S and standard deviation of S thereby impacting the accuracy of the model. Thus, any data shared is processed by the noising module in DPWatch and is DP.

Tagging Mechanism: DPWatch uses tagging to identify data safe to be shared out of the device. In an ML application, only 2 operations can generate results that are safe: the noising of the gradients and the addition of two noised gradients. Thus, the results of all other operations on the accelerators should be marked unsafe. A tag value of 0 is used to indicate that the data is unsafe/sensitive and 1 to indicate safe/noised data. We add a tag generation unit (ⓐ) in the VPU which generates the tags for the result of an operation based on the opcode of the corresponding instruction. A tag value of 1 is generated only for two instructions: *vector-add* and *add-noise*. The result of the *vector-add* instruction is tagged with 1 only if both the source operands have tag values of 1 while the result of the *add-noise* is marked safe unconditionally.

We add additional storage elements to both the on-chip buffers where the results of the operations are written to *i.e. vector memory (vmem)* (❷) and *obuff* (❸). The granularity of tagging should correspond to the minimum access granularity to reduce overheads and avoid false positives [14], which is 128 bytes. Thus, we add 1 bit per 128 bytes for both buffers.

Unlike traditional IFT which sets tags based on the tags of the source operands, we use the opcodes of the instructions to determine tags. This eliminates data movement overheads due to loading tags from off-chip memory for each operation. We modify the memory interface with a *memory tagging unit (MTU)* (❶) to support storing tags in off-chip memory. The *tag setter* (❼) sets the tag of the data loaded from off-chip memory into the buffers. All data loaded from off-chip memory are tagged with 0 unless the *load-tagged* custom instruction is used. The *load-tagged* instruction is used to program the MTU to load the data and the corresponding tags from the off-chip memory. The *tag storage engine* (❻) stores the tags of the data for all writes to off-chip memory. To compute memory addresses for the tags, the MTU contains two configuration registers that store the base address of the tag region (*tag-br*) and data region (*data-br*) in off-chip memory.

C. DPWatch: Operation

All data is initially marked as sensitive and the accelerator is configured by initializing the various registers (z, C, *data-br*, *tag-br*). During each training iteration, once the gradients are computed, the training application uses the *add-noise* instruction to add noise to the gradients. These *noised gradients* are now safe to send out of the device and are automatically tagged as safe. Any further computation using these noised gradients causes those results to be marked sensitive unless two noised gradients are being added together. Thus, the application can only send the noised gradients out of the device. However, multiple training iterations are typically performed locally before sending a model update to the central server. To calculate the local model update for a training round, the application must store a copy of the noised gradients in each iteration and aggregate them using the *vector-add* instruction. The result of this aggregation will also be marked safe and can be shared out of the device.

Malicious applications may attempt to leak information in several ways. First, the application might attempt to send the training data to the server by noising it. During noising, the value of the noise added is scaled with a factor that depends on how big the value is compared to the clipping factor. In practice, the clipping factor is quite small, hence the noise added would render the data unusable. Second, the application might use the raw gradients to update the model weights during each iteration and compute the local model update at the end of a training round. This local model update is marked as sensitive and the application necessarily needs to noise it via the noising module, to share it with the server. If the server uses that noised model update to calculate the new weights of the central model, DP guarantees are preserved due to noising.

Security Analysis: In addition to the hardware modules of DPWatch, we assume that the hardware of the host system and the privileged software (OS) running on the host system are trusted. We also assume that an attacker does not have physical access to the system. To ensure that privacy guarantees are preserved in a system with DPWatch, the host OS has three responsibilities. First, configure the various registers (z, C, *data-br*, *tag-br*) with the correct values. Second, ensure that the application cannot read or write to the memory regions in off-chip memory where the tags are stored. Third, check the corresponding tags when the application attempts to send data out of the device. Only the noised gradients as identified by DPWatch should be allowed to be shared out of the device. In the presence of any security vulnerabilities that allow privilege escalation or manipulation of protected data, a malicious application can circumvent the checks to leak data.

Limitations: DPWatch currently supports applications where all operations are performed on the accelerator. Thus, if any operations need to be performed on the host CPU, DPWatch cannot track operations or modifications to sensitive data. While DPWatch can be extended to enable DP guarantees in GPUs and CPUs, this would require significant changes to the pipeline and memory hierarchy to enable information flow tracking and incur non-trivial overheads. With DPWatch, we leverage the more restricted data flow and limited programmability to design a lightweight tracking mechanism.

IV. Evaluation

Methodology: We implement and evaluate DPWatch using Scale-sim [10], a cycle-accurate simulator for DNN com-

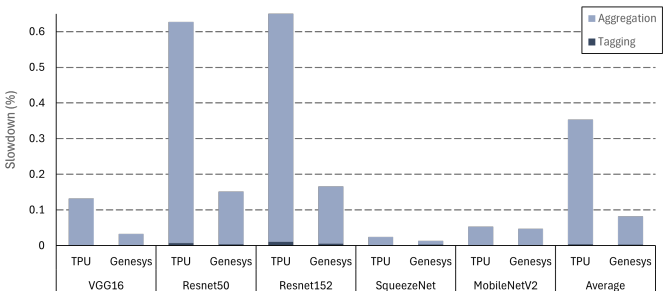

Fig. 4. DPWatch slowdown(%) over the baseline for TPUv1 and Genesys.

putations on systolic array-based accelerators. We evaluate DPWatch for systolic array-based accelerators configured as the Google TPUv1 [12] with 64k MAC units and 24MB of buffers, and Genesys [11] with 1k MAC units and 640KB of buffers.

Results: We evaluated the performance of DPWatch for one communication round for 5 different models: VGG16, ResNet50, ResNet152, SqueezeNet, and MobileNetV2. The experiments were performed on the ImageNet [17] dataset with a batch size of 8 and 10 local training iterations. Fig. 4 presents the slowdown of DPWatch in comparison to the baseline for TPUv1 and Genesys. We observe that DPWatch has an average degradation of 0.35% for TPUv1 and 0.08% for Genesys. The performance overheads from DPWatch are from *(i)* The additional operations and data movement required to calculate the local round update; in the baseline, the round update can be calculated from the final weights by simply subtracting the initial weights. However, in DPWatch, the noised gradients of each iteration need to be fetched and added together. *(ii)* The additional memory traffic due to the tagging mechanism: one additional write request per 1024 requests for Genesys and 8192 requests for TPUv1. The tags of the gradients also need to be fetched while calculating the local round update, generating additional read requests. The slowdown due to tagging is minimal: 0.0045% for TPU and 0.0028% for Genesys. The slowdown due to local round update calculation is the primary cause of the degradation and is dependent on the model size and number of iterations. With larger batch sizes, degradation is reduced, since total execution time for training increases. Thus, we conclude that DPWatch has negligible performance degradation on both accelerators.

Area Overheads: The primary overheads from DPWatch are the SRAM buffers required to store the tags. We evaluate the area and power overheads using CACTI 6.5 [18] for TPUv1, which has a 24MB on-chip buffer. DPWatch requires 1 bit of tag storage for every 128 bytes of data, which causes 0.015% area and 0.0005% power overheads in 22nm technology.

V. Prior Work

To our knowledge, DPWatch is the first hardware-based framework to provide DP guarantees during ML training on specialized accelerators. We briefly discuss prior work.

Privacy Guarantees in Hardware: Prior works [19]–[21] propose approaches that provide guarantees in hardware that sensitive data is noised *before* an untrusted application is allowed to access it, for example at the sensors where the data is collected [19]. These approaches however are limited to certain kinds of sensor data (*e.g.*, accelerometers), and were not designed for text/images/videos. Moreover, directly noising training data is not typically done for ML as it can lead to significant accuracy loss [9].

Accelerators for DP-enabled Training: Prior works propose accelerators for DP for ML: DIVA [22] is a dataflow architecture with a novel reduction module that accelerates the per-example gradient generation. DINAR [23] proposes a novel noise generation algorithm that relies on pre-generated seeds instead of a hardware noise source. While these works improve performance, they still require trusting a third-party application and thus DPWatch is orthogonal to these works.

Software Mechanisms for Protecting User Privacy: Prior works [24] proposed software IFT techniques to detect and prevent leakage of sensitive information. These works rely on a privileged layer of software running between the application and the hardware to instrument tracking and detection. However, these works are not designed for ML applications where gradients computed from sensitive data are sent out of the device. They also rely on a trusted software layer, increasing the attack surface of the system; a software bug can allow malicious applications to leak data. Further, such techniques incur significant performance overheads (*e.g.*, up to 15% [24]).

VI. Conclusion

We propose DPWatch, the first work that enables DP guarantees in hardware for FL on ML accelerators, enabling safe sharing of sensitive data with third party applications. DPWatch uses a lightweight tagging mechanism and a hardware noising module to identify data that is safe to leave the device. We implement and evaluate DPWatch on top of the Genesys [11] and Google TPUv1 [12] accelerators and demonstrate low performance and area overheads.

References

[1] B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data." PMLR, 2017.
[2] C. Dwork *et al.*, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, 2014.
[3] "General data protection regulation (Gdpr) – legal text." [Online]. Available: https://gdpr-info.eu/
[4] L. Melis *et al.*, "Exploiting unintended feature leakage in collaborative learning," in *IEEE S&P*, 2019.
[5] R. Shokri *et al.*, "Membership inference attacks against machine learning models," in *2017 IEEE S&P*, 2017, pp. 3–18.
[6] C. Abuah *et al.*, "Solo: A lightweight static analysis for differential privacy," 2021.
[7] J. Reed *et al.*, "Distance makes the types grow stronger: a calculus for differential privacy," in *ICFP*, 2010.
[8] F. McSherry, "Privacy integrated queries," *SIGMOD*, 2009.
[9] N. Ponomareva *et al.*, "How to dp-fy ml: A practical tutorial to machine learning with differential privacy," in *KDD*, 2023.
[10] A. Samajdar *et al.*, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *ISPASS*, 2020.
[11] S. Ghodrati *et al.*, "Tandem processor: Grappling with emerging operators in neural networks," in *ASPLOS*, 2024.
[12] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," *ISCA*, 2017.
[13] S. Truex *et al.*, "Ldp-fed: federated learning with local differential privacy," in *EdgeSys*, 2020.
[14] W. Hu *et al.*, "Hardware information flow tracking," *ACM Comput. Surv.*, 2021.
[15] D. Lee *et al.*, "A hardware gaussian noise generator using the box-muller method and its error analysis," *IEEE Trans. Computers*, 2006.
[16] J. S. Malik *et al.*, "Gaussian random number generation: A survey on hardware architectures," *ACM Comput. Surv.*, 2016.
[17] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
[18] S. Thoziyoor *et al.*, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," *SIGARCH Comput. Archit. News*, 2008.
[19] M. Maycock *et al.*, "Hardware enforced statistical privacy," *IEEE CAL*, 2016.
[20] A. I. K. Kalupahana *et al.*, "Serandip: Leveraging inherent sensor random noise for differential privacy preservation in wearable community sensing applications," *IMWUT*, 2023.
[21] W.-S. Choi *et al.*, "Guaranteeing local differential privacy on ultra-low-power systems," in *ISCA*, 2018.
[22] B. Park *et al.*, "Diva: An accelerator for differentially private machine learning," in *MICRO*, 2022.
[23] K. Ganesan *et al.*, "Dinar: Enabling distribution agnostic noise injection in machine learning hardware," in *HASP*, 2023.
[24] M. Sun *et al.*, "Taintart: A practical multi-level information-flow tracking system for android runtime," in *CCS*, 2016.