Enc^2:Privacy-Preserving Inference for Tiny IoTs via Encoding and Encryption

Hao-Jen Chien
hjchien91090@ucla.edu
UCLA
Los Angeles, CA, USA

Hossein Khalili
hkhalili@ucla.edu
UCLA
Los Angeles, CA, USA

Amin Hass
amin.hassanzadeh@accenture.com
Accenture
Washington, DC, USA

Nader Sehatbakhsh
nsehat@ucla.edu
UCLA
Los Angeles, CA, USA

ABSTRACT

Privacy-preserving machine learning (PPML) techniques have allowed remote and private inference for resource-constrained internet-of-things (IoT) devices on the cloud. The main challenge in most of the existing PPML technologies is a severe slowdown in inference latency mainly due to the use of *encryption* during the computation. To combat this, an emerging method is to leverage *encoding* as an alternative. While this results in a significant speedup, it imposes the burden of encoding to the resource-constrained IoT/edge device. Despite being feasible for simple workloads where encoding is lightweight, devices with very limited computational capabilities face a tradeoff between latency and privacy when performing complex tasks.

This work proposes an alternative strategy for privacy-preserving inference. Our main contribution is to propose a hybrid method that uses both encoding and encryption. Our key insight is to employ a *cloud-side* encoder achieved by leveraging homomorphic encryption. Since most computations are performed on plaintext, our method enjoys *better latency* than existing encryption-based methods. Additionally, the method eliminates the burden of encoding on the IoT device, resulting in *improved latency AND privacy*. We implement our system, `Orient`, on real-world setups and measure various important metrics including *accuracy*, *privacy*, and *end-to-end latency*. Further, we compare our system with state-of-the-art (SOTA) to highlight its advantages. We show that our method could improve SOTA inference latency by more than 400%, on average, and is an excellent candidate for PPML on low-end resource-constrained devices.

CCS CONCEPTS

• **Computer systems organization** → *Embedded systems*; • **Security and privacy** → **Mobile and wireless security**.

KEYWORDS

privacy-preserving, collaborative edge-cloud computing.

ACM Reference Format:
Hao-Jen Chien, Hossein Khalili, Amin Hass, and Nader Sehatbakhsh. 2023. Enc^2:Privacy-Preserving Inference for Tiny IoTs via Encoding and Encryption. In *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3570361.3592501

1 INTRODUCTION

Recent advancements in artificial intelligence have enabled IoT devices with many new capabilities. However, most of these machine-learning-based tasks are extremely compute-intensive, and consequently, infeasible to be executed locally on a resource-constrained device. To solve this problem, IoT devices rely primarily on a powerful server in the "cloud,"[1] to perform the heavy computation (e.g., matrix multiplication) on the collected data. While this setup provides lots of new opportunities, sensor data sharing creates privacy issues.

To solve this issue, privacy-preserving computation techniques for machine learning (PPML) are proposed, ranging

ACM ISBN 978-1-4503-9990-6/23/10…$15.00
https://doi.org/10.1145/3570361.3592501

[1]For simplicity, in this paper we assume a two-tier client-server computing platform with an IoT/node and a cloud server. The cloud might have multiple sub-tiers, such as edge/fog, but is collectively referred to as "cloud".

from methods such as homomorphic encryption (HE) [4, 13, 22], multi-party computation (MPC) [30, 44], and differential privacy (DP) [1] to hardware-based methods [20, 27, 31, 62]. An ideal solution in a real-world IoT-cloud setting is a method that can achieve high security and privacy, given a threat model, while providing accurate results with a short latency. Additionally, given the rise in the popularity of deep and complex networks, to improve the inference capability of IoT devices, it is ideal to achieve all these metrics even when using a complex deep neural network.

Despite many recent advancements, state-of-the-art methods are not able to achieve all these requirements simultaneously due to the algorithm and/or hardware limitations, while mainly suffering from a severe slowdown. This is particularly problematic for real-time (or even near-real-time) scenarios. There are many examples where real-time response is desired. A few of these include video surveillance, smart home/city, and collaborative robotic/drone systems.

An emerging solution for PPML is using methods based on *encoding* and particularly *adversarial representation learning* (**ARL**) [15, 17, 21, 47, 52, 59, 68]. The key insight is to add an encoding phase in the PPML pipeline such that sensitive information is first removed locally (i.e., on the IoT device), and then (sanitized) data is sent to the cloud for classification. The main benefit is the significant speedup compared to existing PPML methods given that the majority of the operations (i.e., everything besides the encoding) are performed natively in the cloud on raw/plaintext data. However, the major *disadvantage* is that to ensure privacy, the encoder's complexity grows *rapidly* with the network's depth. Since the encoder has to be implemented on the IoT device, a larger encoder hinders the practicality of ARL, especially for tiny resource-constrained IoT devices. Consequently, to balance privacy and feasibility, there is an inevitable *tradeoff* between privacy, latency, and accuracy in ARL methods.

In short, there is a design tradeoff for encoding-based PPML methods. Larger encoders provide better privacy but increase latency. Smaller encoder reduces latency but so as privacy, hence not suitable for more complex networks. Finally, methods based on cryptography provide the best privacy but the worst latency.

To solve this conundrum, ***we propose a hybrid solution*** that builds a bridge between existing PPML solutions and ARL by combining encryption and encoding. Our key insight is to use a *cloud-side* encoder by leveraging homomorphic encryption (HE). The key benefit is that ***(i)*** Our method enjoys better (end-to-end) latency compared to existing HE methods since the majority of computations are performed on the plaintext with only a small reduction of privacy; ***(ii)*** For resource-constrained devices, offloading the encoder to the cloud greatly improves the latency; and ***(iii)*** Our method enjoys superior privacy (especially for complex deep networks) compared to existing ARL methods since it uses a *stronger* encoder that is (securely) offloaded to the cloud.

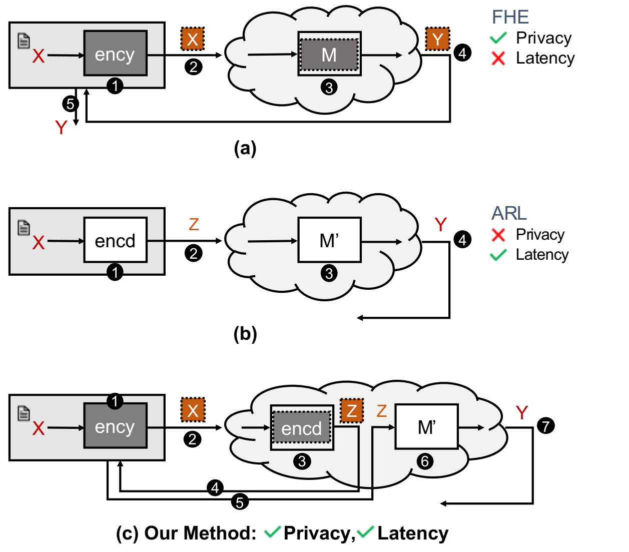


Figure 1: An overview of our method (bottom) vs. existing encryption-based (top) and encoding-based (middle) methods. Our method improves privacy AND latency by using both encryption and encoding.

Note that while in this work we use an ARL+HE method, ***our fundamental observation is that privacy-latency tradeoff can be improved by using ENCoding+ENCryption (enc^2) where various encoding and encryption/HW methods could be leveraged.*** For example, our method could be extended and applied to Encoding+TEE (trusted execution environment) configurations as we will show in §7.

The overview of our system is shown in Figure 1c which improves the existing methods shown in (a) and (b). Briefly, in Figure 1 we show two parties: an *IoT/edge* device (shown on the left), and a *cloud/server* (shown on the right). In conventional HE methods [4, 13, 33, 35], data is first encrypted locally (❶) and then sent to the cloud (❷). An ML model, M, is applied to data *homomorphically* (❸), to create an encrypted output, Y. The result is then sent back to the IoT device to get the final (decrypted) label (❹ and ❺).

For ARL methods [15, 17, 21, 47, 52, 59, 68], instead of encryption, *encoding* is used (Figure 1b). The challenge, however, is that encoding, *encd*, has to be performed locally, hence an inevitable tradeoff between latency and privacy is created.

To address this, we present our method, `Orient`, which instead of encoding the data locally, uses HE (❸). To decrypt the intermediate input, Z, an additional round of communication is needed (❹ and ❺). The rest of the computation is done on plaintext similar to other ARL methods. As we will show in §6, our method could improve the end-to-end latency by **5x** compared to modern HE methods and **3.7x** compared to ARL while providing *similar level of privacy*.

To further improve the applicability of our method, we propose `OrientExpress` in §4. The main insight is that for devices that are energy-constrained but have more computational resources, (small) parts of the encoding could be delegated back to the device. This has two benefits. First, the model's accuracy could be improved by distributing the encoder between the IoT and cloud devices since the error created by the approximation of non-linear layers in HE [13] is mitigated while privacy remains the same. Second, for a more performant IoT device, this results in improved latency. `OrientExpress` is suitable for settings where energy is a constraint thus offloading the computation to the cloud is preferred, however, the IoT device is powerful enough to handle a portion (or all) of the encoding task.

We evaluate our methods using real-world IoT-cloud setups. We implement a proof-of-concept (POC) of `Orient` and compare our systems with state-of-the-art ARL, TEE, and FHE methods [4, 17, 21].

We envision `Orient` is a great fit for running various complex deep learning applications including objection detection and/or image classification on a mobile device such as a robot, drone, and autonomous vehicle. In these systems, while there is some computational power on the device which can be used for the encoding, the majority of computation is uploaded to the cloud.

In short, the major contributions of this paper are:

- Design and implementation of a new lightweight privacy-preserving ML framework that is suitable for resource-constrained tiny IoT devices.
- Developing an improved version of our design to expand its use case and applicability by designing a new scheduling algorithm.
- Evaluating our proposed methods by developing proof-of-concept instances of our system and several state-of-the-art privacy-preserving techniques.

The rest of this paper is as follows. We provide a brief background in §2. The details of our design are presented in §3, and in §4, we describe the improved version of our framework. Details of the experiments and setups used in this work are explained in §5. Experimental results are shown in §6 along with their analysis. §7 presents our further experiments and comparisons with other methods. Related work is presented in §8, and conclusions are provided in §9.

2 BACKGROUND AND ASSUMPTIONS

System Configuration. In this paper, we consider a system where an IoT device is collecting data using its sensors (e.g., a camera). To analyze the sensed data, a machine learning (ML) task (e.g., vision) should be performed. We consider a scenario in which the IoT device has some computation capability, but the ML task cannot be fully executed locally on the IoT device due to the ***computational and/or energy constraints***. As a result, the IoT device prefers to leverage an available cloud computing infrastructure to offload the computation (either fully or partially). Privacy concerns, however, prevent the IoT device from directly offloading its data to the cloud, hence privacy-preserving methods need to be employed. Upon receiving the data, the cloud platform computes the ML task and sends back the final result. We assume that the IoT device uses standard wireless connections (e.g., Wi-Fi) to send and receive data to/from the cloud.

Threat Model. We assume an *honest but curious* scenario where the user trusts the IoT device but does not trust the cloud while all parties follow the agreed protocol correctly. We only consider data privacy scenarios, and therefore, `Orient` is unable to verify the integrity of computation on the cloud. `Orient`, however, can be coupled with existing work to verify integrity [62]. We assume that an adversary controls the cloud environment. We consider a ***strong adversary*** where they not only have access to the inputs and outputs of the application running on the cloud but also all *internal* values including partial computations in every layer of an offloaded neural network.

We consider the privacy attacks during the **inference phase** of a neural network as the main focus. The goal of the adversary is to infer the original input data that was collected by the IoT's sensor. We also assume *a stronger notion* of privacy where an adversary wants to infer some *features* about the input data (e.g., age, gender, etc.) and not the entire input data.

Training-time defenses, such as federated learning [42], attacks such as poisoning [49], and attacks that target model privacy or integrity, such as evasion [5] and/or model extraction [63] are out of scope. Membership inference attacks [43, 58] are also out of scope.

Leveraging Homomorphic Encryption for PPML. An emerging technique for preserving privacy is using homomorphic encryption (HE) [13]. HE enables computation over encryption and prevents the server from ever *observing* the plaintext. There are two major challenges with HE. First, HE computation is orders of magnitude slower than regular computation. Second, HE methods could not evaluate non-linear operations (e.g., ReLU, tanh, etc.), hence they need to be approximated with a polynomial function. This approximation, unfortunately, creates errors and decreases accuracy. To address the first problem, recent hardware and software methods have been proposed, ranging from designing customized hardware to new algorithms [6, 7, 26, 33, 50, 56, 64].

To address the latter, specialized networks have been proposed [4, 35] which can tolerate the noise created by the approximation of non-linear functions. Although effective for simpler networks, neither of these solutions is scalable to complex deep neural networks.

Adversarial Representation Learning (ARL). These methods solve the privacy problem in ML by using an *additional* component, called an encoder, to remove sensitive information. In this method, the representation learning task is divided into extracting two types of attributes: *target and sensitive*, where ideally, the target attribute is accurately extracted while the sensitive attribute(s) is fully removed. An example is object detection where detecting objects is the target task while finding additional identifying information about objects (e.g., gender, race) could be sensitive.

To achieve this, ARL forms a *game* consisting of three players during the training phase: an encoder E, a discriminator D, and a predictor P. The encoder maps input data, x, *deterministically*, into a (low-dimensional) feature space. The discriminator, which works as a proxy of an unknown adversary, is associated with a *sensitive* attribute with m classes, $\mathrm{S} = \{s_1, ..., s_m\}$. It uses the encoded input, $z = E(x; \theta_E)$, to identify the sensitive attribute from the encoded input. The predictor is associated with a target attribute with n classes, $\mathrm{Y} = \{y_1, ..., y_n\}$. It makes use of the same encoded input to make predictions of Y as in a typical discriminative model.

Using this setting, instead of directly modeling the system as $p(y \mid x)$, the problem is modeled as a conditional distribution $p(y \mid x, s)$. Formally, there is a *minimax* game:

$$\min_{E,P} \max_{D} V(E, P, D) = \mathop{\mathbb{E}}_{x,s,y \sim p(x,s,y)} [\alpha \log q_D(s|h = E(x, s)) - \log q_P(y|h = E(x, s))], \quad (1)$$

where the hyperparameter, α, creates a trade-off that adjusts the constraint of the invariance (accuracy of the system vs. the level of privacy), and $p(x, s, y)$ is the ground truth distribution that empirical observations are drawn from.

The key challenge in ARL is that to maintain privacy, the encoder's complexity grows as the network becomes more complex. As a result, the existing ARL methods are limited to simpler networks and *a method that can use ARL for large and complex networks is still missing.* Furthermore, ARL is only applicable to a special class of privacy-preserving problems where the target and sensitive tasks can be well-defined. We acknowledge that there are scenarios where this is not applicable, however, we emphasize that there is a large set of problems that could benefit from this approach. Note that the target and sensitive tasks shouldn't be physically separated or visually observable, but there should be minimal mutual information between the two tasks.

Method	Metric				
	Late.	Accu.	Complexity	Privacy	Security
Crypto	○	◕	◑	●	●
TEE	◑	●	◕	◕	◑
Noise	●	◑	◑	◕	●
ARL	●	◕	◑	◑	●
`Orient`	●	◕	◕	◕	●

Table 1: Comparing PPML methods using different metrics. *Crypto* refers to methods based on FHE, MPC, and their hybrid combination. *TEE* is for methods that use trusted execution environments. *Noise* refers to methods that leverage adding noise for privacy and differential privacy.

Comparing Various PPML Methods. PPML techniques can be categorized into *four* main categories: *cryptographic*, *statistical*, *hardware*, and *encoding*. The *fundamental difference* among these methods is a trade-off between the degree of privacy delivered in each method versus computational efficiency. Another important difference between them is the *threat model* and security assumptions (i.e., what should be protected and what can be shared, etc.).

To help us understand why we need a new method in this domain, we compare the existing methods using the five main metrics discussed in §1. Table 1 shows this comparison. As can be seen, neither of the methods can achieve a balanced solution, indicating that a new method that can address these together is needed.

3 SYSTEM DESIGN FOR ORIENT

Overview. To design a new privacy-preserving method that can handle complex deep neural networks we use the following two observations: ***(i)*** Methods based on HE, MPC, and TEE can provide sufficient privacy, but they cannot handle deep neural networks due to the significant slowdown, and ***(ii)*** Methods based on ARL can address the latency issue but in order to provide sufficient privacy they need complex encoders. Combining these two, we propose a new method, called `Orient`, that leverages ARL to ensure privacy, but instead of using a local encoder, the encoding is performed homomorphically using standard HE algorithms. As a result, our model enjoys the benefits of ARL while easing the burden of encoding by leveraging HE.

Our method, `Orient`, has two major components: the encoder and the predictor which are formed by *cutting* the network into two parts where the first part is labeled as the encoder and the rest is the predictor.

More formally, a deep neural network, M, with L layers (linear and non-linear combined) is defined as $y = f(x; \theta_M)$ s.t. $f_M = f_1 \circ f_2 \circ ... \circ f_L$, where $f_i = (.; \theta_i)$. To create the encoder and predictor, M is *cut* into two parts, f_{M_1} and f_{M_2}

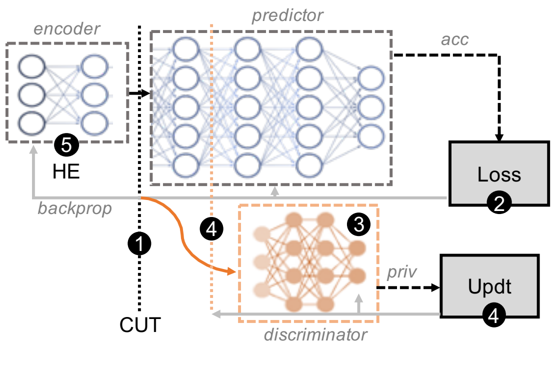


Figure 2: Designing `Orient` has several components: encoder, predictor, discriminator, and loss function. Five steps need to be followed to implement `Orient`.

at layer C. The encoder module, $E = f_{M_1}$, is defined as $z = f_E(x;\theta_E) = f_1 \circ f_2 \circ ... \circ f_C$ (where $C < L$), and the predictor, $P = f_{M_2}$, is defined as $y = F_P(z;\theta_P) = f_{C+1} \circ f_{C+2} \circ ... \circ f_L$.

The overview of our design is shown in Figure 2. The overall goal is to maximize an optimization problem with three metrics: *privacy*, *accuracy*, and *latency*. There are three main steps in designing `Orient`. First, the baseline network is divided into an encoder and a predictor by *cutting* the network (❶). Second, the encoder and predictor are trained by designing a proper loss function (❷). Then, privacy is evaluated, and the network is adjusted if needed (❸-❹). Finally, a homomorphic encryption algorithm is implemented for the encoder (❺). In the following, we describe each one by one.

3.1 Network Design

As described earlier, our focus is convolutional deep neural networks, hence `Orient` takes a baseline CNN model as an input. The baseline model could be a custom or standard model. We will provide a few examples used in our evaluations in §5. The baseline model then needs to be divided to form the encoder and predictor. To find an optimal point (to optimize *latency*, *accuracy*, and *privacy*), `Orient` uses an *iterative* algorithm.

Intuitively, to decide the optimum layer, we observe that the encoder's size (i.e., the number of layers which is assigned to the encoder), creates a **tradeoff** between *privacy* vs. *latency* and *accuracy*. On one hand, a larger encoder could improve privacy since as we proceed through the deeper layers, the feature becomes more specific to the main task, and irrelevant information (including sensitive information) will be gradually lost [60, 69]. On the other hand, larger encoders increase latency since in our method the encoder is evaluated homomorphically. Further, due to the usage of an HE method for the encoder, larger encoders also decrease accuracy because HE computation introduces errors for non-linear operations. As a result, the optimization algorithm tries to find the *minimum* size for the encoder such that it satisfies the privacy requirement.

To measure privacy, the network has to be trained first. We will describe the training algorithm shortly in §3.2. Assuming that the network is trained, to measure privacy, we use a utility network, called a *discriminator*. The role of the discriminator is to evaluate how well the encoder hides private information. More specifically, the discriminator is a classifier (e.g., a multilayer perceptron) that tries to infer a sensitive label, s, from the intermediate variable, z (see the bottom of Figure 2). The discriminator is trained *separately* from that of the encoder and predictor and outputs the *accuracy* of inferring the private label. The *update* function (❹) uses this information to compute the overall loss and privacy and adjusts the encoder.

3.2 Training

There are several proposals for training an ARL model [15, 17, 21, 40, 68]. The key idea is to use a proper loss function that considers *both* accuracy and privacy. A common method is to use an adversarial *three-player* minimax game between the encoder, predictor, and discriminator as described in §2. An alternative method is to minimize the *mutual information* between the intermediate variable, z, (i.e., the output of the encoder), and the sensitive attribute(s). The main advantage of the latter is that no assumption about the exact attacker's classifier is needed during the training. We find that the latter method is more appropriate for `Orient` and hence used in this work. Particularly, we develop our algorithm based on the method proposed by Gupta *et al.* [17] but *extend it to address the unique needs of* `Orient`.

Recall that in the ARL problem setting, there are two types of attributes, private and public. To preserve privacy, the goal is to minimize the information in z that is relevant to the private task, s, while maximizing the information relevant to the public task, y. More formally, this can be shown as

$$\max_E I(y : z) - \alpha I(z : s), \tag{2}$$

where $I(\cdot : \cdot)$ is the mutual information and α is the tradeoff coefficient (between privacy and accuracy).

The fundamental challenge in ARL is the potential *correlation* between public and private attributes. In one extreme, the two attributes could be independent (i.e., $I(s : y) = 0$), providing an ideal case since the private information could be potentially fully removed without impacting the accuracy of the public task. On the other extreme, the two attributes could be highly correlated which significantly impacts the feasibility of ARL (because removing sensitive information significantly decreases the accuracy).

Algorithm 1: Building and training `Orient`

```
Input: $M$ ;      // $L$-layer baseline neural net
Input: $D$;       // $K$ layer baseline classifier
Input: $Loss$;        /* A training function for
        $E, P, and\ D$ */
Input: $(X, Y)$;                  // Training data
Input: $TH$;          // A threshold for privacy
Output: $E, P$;
1  $E = \{\}, P = M, i = 0$;
2  while i<L do
3      $E = \{E, M_{i+1}\}$;
4      train$_{loss}$ $E$ and $P$;
5      train$_{loss}$ $D$;
6      $Priv = D(E(X))$;
7      if $Priv \geq TH$ then
8          break;
9      end
10     $i++$;
11 end
```

For more common scenarios where there is *some* correlation between the two variables, the accuracy of Equ. 2, could be further improved by this fundamental insight that the goal of the encoder should not be maximizing *all* information about y, instead, it should maximize the mutual information between z and information about y that is *not* related to the private attribute, z. More formally, we have:

$$\max_{E} \ I(y : z|s) - \alpha I(z : s), \tag{3}$$

where $I(y : z|s)$ indicates the mutual information between z and y that is not related to s.

While intuitive, direct measurement of Equ. 3 is not straightforward and is time-consuming. As a result, an *estimation* is needed. We use the method proposed by Gupta *et al.* [17] to achieve this. However, *unlike their method*, we apply it to ***multiclass non-binary public attributes***. This is achieved by adjusting the loss function by using the *maximum* cross-entropy among ALL classes. Similar to other conventional learning systems, this calculated loss is then backpropagated to the encoder and predictor to update their parameters. This process is repeated until either the loss is minimized or a certain epoch/iteration is reached. We will provide more details about the hyperparameters used for training in §5.

Putting it all together, `Orient` leverages Algorithm 1 to create a model and train it (see ❶-❹ in Figure 2). Briefly, the baseline network is first cut into an encoder *with one layer* and a decoder with the rest. The network is then trained using the method described above. Then, the accuracy of the encoder is measured, using a classifier (i.e., the discriminator). Using this measurement, the *update* function then decides to either terminate the training (if privacy is acceptable) or to add a layer to the encoder and repeat. Recall that the goal is to have the smallest encoder without sacrificing privacy.

Note that for the training we use mutual information to calculate the loss, but Algorithm 1 leverages a classifier (discriminator) for evaluating the encoder. We find that this is a better choice (as opposed to existing methods that use only one method) due to two reasons. First, this allows us to evaluate the encoder *independently* using the average performance by evaluating ALL records, hence increasing the robustness. Second, it allows us to improve privacy by using various classifiers for the discriminator (details of the model used in §5).

3.3 Homomorphic Evaluation

As mentioned earlier, to ensure privacy, deep networks require complex encoders, hence their latency increases. This is especially amplified when a low-end IoT is used. To combat this, we propose offloading f_E entirely to the cloud. However, plain offloading would violate privacy given the threat model described in §2 since the adversary has full control of the cloud and hence could access the internal values in the encoder. Alternatively, our proposal is to use homomorphic encryption to evaluate f_E directly on the cloud. This approach can significantly ease the burden on the IoT device.

More formally, let $Ey(\cdot, pk)$ and $Dy(\cdot, sk)$ be encryption and decryption functions, respectively, based on a homomorphic encryption algorithm, HE, with key pair (sk, pk). Let $HE_f(\cdot, sk)$ be an evaluation function that computes a polynomial function, f, on a vector input homomorphically. Then our model in `Orient` can be evaluated as:

$$y = f(x; \theta_M) = P(Dy[HE_E(Ey(x, pk)), sk]). \tag{4}$$

There are two main components for enabling HE evaluation on the encoder network. First, designing an efficient HE algorithm that is suited for our application. Second, creating a controller to orchestrate the data movement between the IoT and cloud devices. This is particularly challenging given that an additional round of communication is needed between the devices once the output of the encoder is ready.

Over the last decade, prior work has proposed multiple HE schemes with different capabilities and performance tradeoffs. BGV [3], BFV [9], GSW [11], TFHE [7], and CKKS [6] are among popular HE schemes. For `Orient`, we choose BFV [9] as the baseline since it provides an opportunity to balance performance and accuracy using state-of-the-art (SOTA) optimization techniques [35, 64].

To design our homomorphic encryption evaluation engine (HE3), we make ***two fundamental observations*** that are unique to low-end IoT devices application domain where data communication between the device and cloud is not continuous and then propose a novel technique to address our unique needs.

First, in an IoT-cloud setup, we observe that inference requests are mostly *sparse* since the IoT device has to occasionally *sense* the inputs and then forward them to the cloud for inference. As a result, *input packing* techniques, often referred to as SIMD (single instruction multiple data) operations, commonly used in SOTA are not beneficial in our setting, since multiple inputs are rarely available simultaneously. On the contrary, *bursty* (i.e., multiple consecutive but delayed samples) requests are quite common as once the sensor is activated, it is very likely that *multiple* requests are sent to the cloud consecutively. Given this observation, our main insight is that `Orient` should be ideally designed such that it is ***optimized for single but consecutive inferences*** rather than multiple but one-time inference requests which are commonly used in existing work.

Second, given that *only* the encoder (i.e., the first few layers) is evaluated homomorphically, HE3 should be optimized to evaluate *convolutional* layers (recall that we focus on deep CNNs and typically the first few layers of these networks are always convolutional networks), rather than being optimized to handle *both* convolutional and fully-connected layers. It is important to mention that, to speed up computation, in HE-based methods weights/filters in each layer are stored in plaintext and only inputs/channels are encrypted[2].

Combining these observations, we propose ***weight redundant mapping*** for `Orient`. The key insight is that instead of packing multiple channels into one ciphertext and applying the same filter (i.e., batch processing commonly used in conventional models), the filter itself could be packed (i.e., multiple instances of the same filer can be *packed* into one ciphertext), allowing more parallel elementwise channel-filter multiplications, and more importantly, fewer rotations. Furthermore, *filter-packing* allows us to process consecutive requests more efficiently.

While this may not be beneficial in non-homomorphic convolutions because (a) there is no need for rotations and (b) a larger filter means more data movement, for our specific HE-based convolutions scenarios where requests are rate and the *multiplications* and *rotations* are very slow, this would result in speedup.

It is important to mention that batching multiple encrypted channels still outperforms our new proposal if multiple requests are available since it needs fewer data movements and has more useful computation. However, if data collection is sparse, weight redundancy could improve the overall latency. Consequently, one could opt for a dynamic method to choose between input packing or weight redundancy and/or leverages a profiling phase to find the best setting.

[2]The assumption is that the privacy of the inputs needs to be protected but not the weights as they are safely residing on the cloud. See our discussion in Threat Model (§2 for more details.

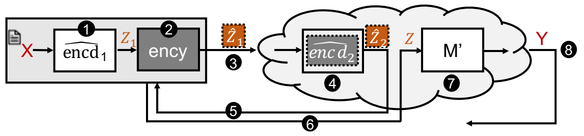


Figure 3: Overview of `OrientExpress`. The key difference is dividing the encoder into two parts to tradeoff between latency/accuracy and energy efficiency.

4 DESIGNING ORIENT EXPRESS

In the previous section, we described the details of our system, `Orient`. We explained three important components of our system namely network, training, and homomorphic encryption evaluation engine (HE3). In designing `Orient`, we always assumed that the IoT device is only responsible to collect inputs, encrypt them using a given HE algorithm, and transmit these input(s) to the cloud. Additionally, the IoT device is responsible to *decrypt* the intermediate data, *z*, and sending back the data to the cloud to continue the computation.

This *client-optimized* design assumes that the IoT device has minimal computational power and hence relies heavily on the cloud. In this section, we explore how our method could be extended to more sophisticated IoT devices where the overhead for computation is much lower. We propose a new design called `OrientExpress` that is more suitable for such a setting. Our fundamental insight for `OrientExpress` is to delegate *some* of the computation back to the IoT device. We, however, assume a scenario where the IoT device is energy-constrained, so it still prefers to offload its computation to the cloud entirely. However, unlike the setting in `Orient` (cf. §3), the IoT device is performant thus it can handle all or parts of the encoding. Note that for scenarios where the IoT device is powerful and energy is not a constraint, neither `Orient` nor `OrientExpress` would be suitable, and conventional ARL techniques are a better fit.

The overview of this system is shown in Figure 3. The idea is to use a pipeline of *encoder$_1$-encoder$_2$-predictor* where encoder$_1$ is implemented on the IoT device *in plaintext*, while encoder$_2$ is on the cloud and uses HE (similar to `Orient`) (see ❶ and ❹). The rest of the system (e.g., encryption, predictor, etc.) is the same as can be seen in Figure 3.

Comparing `Orient` and this variant, `OrientExpress`, there are two important observations. The first is that the encoder in `Orient` should be exactly the same as encoder$_1$+encoder$_2$ in `OrientExpress` because the same rationale used for privacy-latency tradeoff in `Orient` applies to `OrientExpress` (see §3.1). This means that the privacy guarantees of both systems are (exactly) similar.

Algorithm 2: Designing `OrientExpress`

```
Input: $M, D, Loss, X, Y, TH$;
Input: $B$ ;     // Energy budget for the device
Input: $TH_{La}, TH_{Acc}$;
Output: $E = \{e_1, e_2\}$;
1  $\{E, P\} = Alg_1(\cdot)$ ;               // See Algorithm 1
2  $e_1 = \{\}, e_2 = E$;
3  $Bud = 0, i = 0$;
4  while $Bud < B$ do
5      $e_1 = \{e_1, E_{i+1}\}$;
6      $e_2 = \{E_{i+2}, E_k\}$ ;            // $E$ has $k$ layers.
7      $Bud = Calc_E(e_1)$;
8      $Acc = P(e_{2_h}(e_1(X))$ ;   // $e_h$ means when $e$ is
         evaluated homomorphically.
9      $Lat = Calc_{Lat}(e_1, e_{2_h}, P)$;
10     if $Lat \le TH_{La}$ AND/OR $Acc \ge TH_{Acc}$ then
11         break;
12     end
13     $i++$;
14 end
```

A more interesting observation is that latency and accuracy could be different between the two methods. More specifically, ***this delegation creates a new tradeoff*** between latency/accuracy and *energy consumption*, where in one extreme, all computation is performed homomorphically on the cloud (slow and approximate but energy-efficient), and in the other, encoding is entirely performed on the IoT device (faster but not efficient). Consequently, `OrientExpress` provides an additional *control knob* to the user where latency/accuracy can be traded with more energy efficiency.

Using the observations above, we improve our algorithm described in the previous section to support an additional Cut. Specifically, we present Algorithm 2 for `OrientExpress`. The output of this algorithm is a new CUT inside the encoder to decide which part should be on the IoT (e_1) and which part should be evaluated homomorphically on the cloud (e_2). There are three criteria for finding the desired cut. First, the assignment should be such that the energy overhead on the IoT is less than some predefined threshold. This can be *estimated* by counting the number of parameters in e_1. The algorithm iteratively adds an additional layer to e_1 until either/both accuracy and/or latency requirements (second and third criterion) are satisfied. This is a knob defined by the user (e.g., one user may only care about latency, while the other needs both, etc.). Similar to the energy budget, *latency* can be estimated by the number of parameters. Measuring *accuracy* is straightforward (we will formally define it in §5).

5 IMPLEMENTATION AND EVALUATION SETUP

Implementation. We implement a proof-of-concept for `Orient` and `OrientExpress` frameworks. For `Orient`, we use an Arduino Uno (with a HiLetgo ESP8266 Wi-Fi module). For `OrientExpress`, we use a Raspberry Pi 4 (ver. B). The two devices are selected such that they represent the low and med/high-end spectrum of an IoT device computational power consistent with scenarios we assumed for `Orient` and `OrientExpress` (i.e., resource-constrained and energy-constrained for `Orient`, and only energy-constrained for `OrientExpress`). To model the cloud, we use a server equipped with a 10-core Intel X-series Core i9, 256 GB DDR4 memory, and two Nvidia A6000 GPUs connected to the network using a 1 gigabit LAN cable. We use a separate script running on a laptop (MacBook Air, 2020) to control the test and measure the latency and other metrics (details later). We observe around 6.3 MB/s average data rate (upload) between the nodes in our setting. In §7, we analyze the impact of changing the network bandwidth on the overall latency.

For our homomorphic encryption evaluation engine (HE3), we use BGV [3] algorithm. We use Microsoft Seal open-source libraries (SEAL [38] and Cryptonets [37]) for implementing the encryption, decryption, and evaluation modules. We modify the codes to implement our proposed *weight redundant* mapping method (cf. §3.3), and make minor modifications to implement the encryption/decryption modules on Arduino. Given that the encoder is typically shallow, `Orient` enjoys much smaller sizes for the ciphertext (compared to SOTA) hence reducing the overhead. Particularly, to achieve > 128 bit security (similar to SOTA), we use smaller sizes for polynomial modulus (N) and coefficient modulus (Q), while using larger plaintext modules (t) since a smaller noise budget is needed. Details of the parameters are shown in Table 2. We use a simple Python code controlling the process on the cloud (encoding, sending/receiving data, etc.)

We train our models (details later) using Python and PyTorch v 1.8.1 library. The models are trained on the cloud. For inference on Raspberry Pi (only for `OrientExpress`), we use the available PyTorch library on Raspberry Pi. We do not implement `OrientExpress` on Arduino since it is not suitable for this scenario (cf. §4). We did not observe a significant change in training time (only a small overhead) for `Orient` as training time is mainly dominated by a large number of input iterations and backpropagation.

For comparison against the state-of-the-art (SOTA), we pick *three* methods: LoLA [4] to represent a SOTA homomorphic encryption method and two ARL methods: Adversarial Forgetting [21] and FCRL [17]. We use their respective open-source implementations publicly available on GitHub [16, 37], and make minor changes so that the codes

Table 2: Configuration used in this work.

Comm. (avg.)	6.25 MB/s
HE Params	N=13, Q=175 t=23 (all in bits)
Datasets (networks)	CIFAR100 (Resnet-18) UTK (VGG16)

can be compiled and executed on our machines. For each method, we use the optimized encoder size (cut) and hyperparameter suggested by the authors in their respective works. To run the encoder on the IoT device, for Arduino we use the available TF-Lite library[3]. We use *int8* quantization for all models implemented on the IoT device.

Datasets and Models. We use two datasets to evaluate the effectiveness of our frameworks. First, we use CIFAR100 dataset to train a Resent-18 neural network. The dataset labels 100 classes with 600 images each and all classes are grouped into 20 *superclasses* [25]. In our problem setting, the 20 coarse/superclasses are defined as the *public* attribute and the 100 'fine' classes are defined as the *private* attribute that needs to be protected. Each image in the dataset belongs to one class in the public domain and one class in the private domain (e.g., an image could be in *superclass* 'PEOPLE', while its *private* class is one of the corresponding fine classes: {'BABY', 'BOY', 'GIRL', 'MAN', 'WOMAN'}). This means that the public and private tasks are slightly correlated.

We use Algorithm 1 to find the CUT and train the network. We find that the output of the second CNN layer is the best location for CUT. For each epoch, the encoder and the predictor are trained first, and then the discriminator is trained. We train the target model with 150 epochs using an ADAM optimizer for the optimization. We set the discriminator learning rate to 10^{-2}, the encoder learning rate to 10^{-4}, and the predictor learning rate to 10^{-2}.

To evaluate our methods on a more complex network, we also use Gray-Scale UTKFace [70] dataset to train a VGG16 network. UTKFace is a large-scale RGB face dataset consisting of over 20,000 face images, labeled on the basis of age, gender, and ethnicity. The gray-scale version is cleaned and processed to have images in the gray-scale. The age ranges from 0 to 116, the gender has 2 classes, and the ethnicity has 5 classes. We select *gender classification* as the public task and *ethnicity classification* as the private. Gender and ethnicity are **independent** and **uncorrelated**. The ethnicity is unbalanced, with a dominant label of 42.5%. The model is trained with 20 epochs and the learning rate is 10^{-4} for the predictor and encoder. The discriminator learning rate

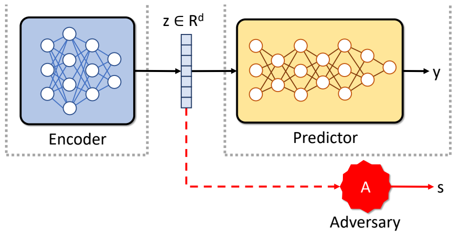


Figure 4: The setup used for measuring privacy. We train a customized classifier, *Adv*, and measure its accuracy in inferring the *private* label.

is 10^{-3}. ADAM optimizer is used for optimization with a momentum of 0.9. Similar to Resnet-18, we use Algorithm 1 to find the CUT which is the end of the first residual block. For both workloads, we use $\alpha = .5$ as the default.

Metrics. As extensively discussed in this paper, the overall goal is to maximize an optimization problem with three metrics: *privacy*, *accuracy*, and *latency* defined as follows:

DEFINITION 1 (ACCURACY). *For a multi-class classification problem, accuracy is defined as the ratio when comparing ground-truth labels versus model predictions, i.e.,* $Acc = CorrectPredictions/AllPredictions$.

This is a conventional metric that is commonly used in literature. Measuring `latency` is also straightforward as it is the *end-to-end* delay from the first operation in IoT to receiving the final classification label back from the cloud. We define `privacy` as:

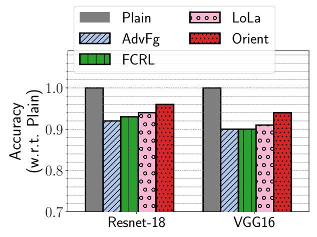
DEFINITION 2 (PRIVACY). *The accuracy of an independent classifier, called Adv, in predicting the private label during inference is defined as privacy.*

Figure 4 shows how privacy is calculated in our setup. We opt for this method instead of purely relying on mutual information (or other approaches, such as anonymity) to prevent underestimation of privacy as shown by recent work [10, 67]. This method of measuring privacy is also more common in the literature. For each network, we use an ensemble (four) of network architectures and report the one with the highest accuracy. For UTKFace, the best classifier has 3 CNN layers and 4 fully connected (FC) layers. For CIFAR100, the best classifier has (only) 7 FC layers. The attacker's classifier and `Orient` datasets are independent and separate.

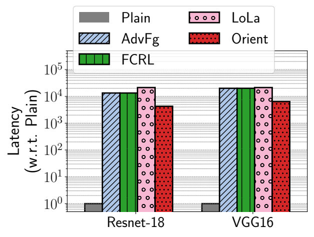
6 RESULTS

Overview. We report three sets of results using the setup described in the previous section. First, we present the comparison between `Orient` and SOTA. Then, we compare the results for `Orient` and `OrientExpress`. Finally, energy consumption evaluation is discussed.

[3]We first train the entire model using Pytorch, and then convert the encoder to a Tensorflow model and implement it on Arduino.

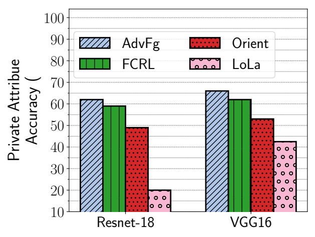


(a) *Accuracy* (higher is better)

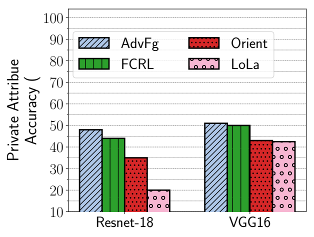


(b) *Latency* (lower is better)

Figure 5: Results for `Orient` and comparison with SOTA (numbers are normalized w.r.t. the *plain* model). All setups used the *same* encoder size.



(a) α = .5



(b) α = .75

Figure 6: Private label inference accuracy for two trade-off coefficients (see Equ. 3). Lower accuracy means more privacy. ARL setups and `Orient` have the same *latency* (i.e., different encoders).

Comparison with State-of-the-Art. We compare the performance of `Orient` with state-of-the-art (SOTA) using the three metrics described in §5. In Figure 5, we show the comparison for latency and accuracy. Specifically, we compare the latency and accuracy of `Orient`, implemented on Arduino+server, to the baseline (i.e., the actual network with no privacy-preserving method implemented, called *plain*), and three SOTA methods. To provide a fair comparison, we use the same encoder size for all methods. We repeat this experiment for both networks (Resnet-18 and VGG16) where each has its unique encoder which is similar across methods. For all tests, α is set to 0.5.

Analyzing the results for accuracy (Figure 5a), it is clear that all methods have lower accuracy than the baseline (i.e., the network with no privacy). This is expected since the baseline model uses neither edge-side computing nor leveraging HE, while all the other methods use either of them. Overall, there are two sources of errors, one is the *quantization error* on the IoT device and the other is the *approximation error* caused by approximating non-linear layers in HE. Both ARL methods (AdvFg [21] and FCRL [17]) suffer from the quantization error since the encoder should be entirely implemented on the IoT device. LoLa [4] and our method, `Orient`, incurs error due to approximation. They do not incur any quantization errors as the encoder is offloaded to the cloud. `Orient` has higher accuracy compared to LoLa since a much smaller part of the encoder is evaluated homomorphically. Comparing `Orient` with ARL methods show that the quantization error is dominant since approximation error affects only non-linear layers while an aggressive quantization (e.g., int8) impacts all layers. We did not investigate larger integers (e.g., int16) due to their significant overhead in our setup.

Latency results are shown in Figure 5b. All methods incur a significant overhead compared to the baseline since they either use HE or need to compute the encoder on a resource-constrained IoT. `Orient` is the fastest (about **5.2x** faster than LoLa and **3.12** than ARL). One important reason for better performance in `Orient` compared to ARL methods is that encryption/decryption is relatively faster than computing a few layers locally. We also examine `Orient` latency with and without *weight redundant* mapping and find that it contributes to **2.81x** of speedup. Also note that the results reported here are based on the network latency observed in our setting (cf. Table 2). Significantly slower communication networks could result in higher end-to-end latency as `Orient` introduces an additional communication round. We investigate this further in §7.

To measure privacy, we report the accuracy of inferring the private label using a classifier, *Adv* (cf. §5). Lower accuracy indicates higher privacy. Recall that for CIFAR100 (Resnet-18 network) the public labels are the *superclasses* and the private label are the *fine classes*. For UTKFace (VGG16), *gender classification* is the public task, and *ethnicity* is the private task. Results for both datasets are shown in Figure 6. For a fair comparison, we adjust the ARL methods such that they have similar *latency* compared to `Orient`.

For α = .5 in Figure 6a, it can be seen that **`Orient` *has consistently better privacy*** (lower inference accuracy) than other ARL methods since `Orient` can apply a stronger encoder for a given latency budget.

Note that LoLa's accuracy is equal to random guessing (ideal) since values are all encrypted (42.5% for UTKFace and 20% for CIFAR100). To improve privacy, a more aggressive coefficient for privacy could be used. One example is shown in Figure 6b where α = 0.75 (although this would inevitably reduce accuracy). Comparing the two experiments, increasing α improves privacy significantly where `Orient`'s privacy is almost equal to the ideal case for VGG16.

`Orient` vs. `OrientExpress` Comparison. To show the effectiveness of `OrientExpress`, we compare its performance with `Orient`. For this experiment, we use Raspberry Pi as the IoT device. We use Algorithm 2 to find the Cut_1 and Cut_2. We test both networks and report the average values for the three metrics. The same α is used for both. Results are shown in Figure 7.

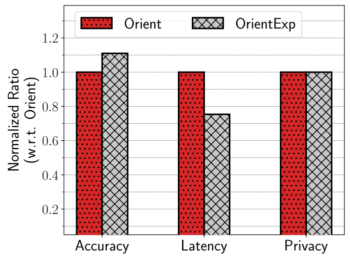


Figure 7: Comparison between `Orient` and `OrientExpress` for three different metrics (on average) with the same α and encoder.

Comparing the two systems, they both have very similar results for *privacy*. `OrientExpress` achieves much better *accuracy* given that the approximation error is *significantly* mitigated while the quantization error is minimum. Our additional investigation shows that the *approximation* error caused by HE is much larger if it is used at the initial layers of the network compared to later layers. In other words, we observe that if we approximate T layers, then $error_{approx}(L_1, ..., L_T) > error_{approx}(L_i, ..., L_{i+T})$ for $i > 1$. Consequently, `OrientExpress` achieves better accuracy since cutting the encoder effectively has the same impact as above. The added bonus for `OrientExpress` is a better latency assuming that the IoT device is performant.

Energy Efficiency. To further investigate the benefits of `OrientExpress`, we compare the energy consumption of the IoT device in each method for the two networks. To measure energy, we use a current sensor (INA169) connected in series with the device. We repeat each experiment five times and report the average to minimize the background noise. Each measurement includes energy for encoding, encryption, and communication. For `Orient` and `OrientExpress`, we also measure the energy for decryption and resend the data. Results are shown in Figure 8.

`Orient` and LoLa have the minimum energy since the minimal work is done on the IoT side (encryption and transmitting). ARL methods have about 17x higher energy compared to `Orient`, while `OrientExpress`'s overhead is about 5x.

Combining the observations made in Figures 8 and 7, `OrientExpress` improves latency and accuracy at the cost of consuming more energy. Comparing `OrientExpress` and ARL shows that for more performant devices, `OrientExpress` offers a better tradeoff for energy, accuracy, and latency while achieving the same or better privacy.

Summary. We combine the comparisons made in this paper in Figure 9. We show five different methods: Plain (baseline with no privacy), ARL (average behavior of the two SOTA methods, AdvFg [21] and FCRL [17]), HE (LoLa [4]), and our

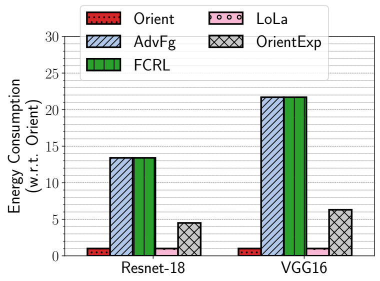


Figure 8: Comparing the energy consumption (only for the IoT device) of various methods.

proposed methods, `Orient` and `OrientExpress`. We compare them using three metrics and report their relative values when averaging on two networks: Resnet-18 and VGG16.

Overall, our methods provide a better balance in the privacy-accuracy-latency space. For *tiny* low-end IoT devices where on-device computation has a large overhead, `Orient` improves the latency and accuracy while maintaining privacy. For more sophisticated IoT devices, `OrientExpress` creates a balance between energy efficiency and other important metrics. It is also useful to mention that shallow networks may not benefit from `Orient` compared to regular ARL methods. However, as the network gets deeper, Orient becomes more beneficial. As the network becomes more complex, the overall latency increases since more layers are added to the encoder. This means all approaches will take a hit on latency. However, there will be a point at which the overhead for ARL won't be negligible anymore since the encoder is too large. The maximum gain for `Orient` happens when the encoder is bigger than that threshold while the ratio between the encoder and the overall network is minimum. This will be dependent on the application scenario.

7 FURTHER EXPERIMENT: COMPARISON WITH TEES AND ACCELERATORS

Overview and Setup. An alternative strategy for offloading the encoder is to use a trusted execution environment (TEE) on the server/cloud side. Several recent studies, such as Occlumentcy [27], Slalom [62], Origami [45], and DarKnight [18] have explored this idea in various forms. Furthermore, another recent trend in privacy-preserving machine learning is to use specialized hardware accelerators to speed up the computation especially for HE algorithms [50, 51, 53, 56, 57, 64]. To compare these trends with our methods and study how `Orient` could benefit from them, in this section, we evaluate `Orient` in the context of TEE-based and hardware accelerator-based methods.

To provide a fair comparison among different methods, we develop a simple *emulator* to *estimate* the performance (end-to-end latency) of each given method. In essence, our

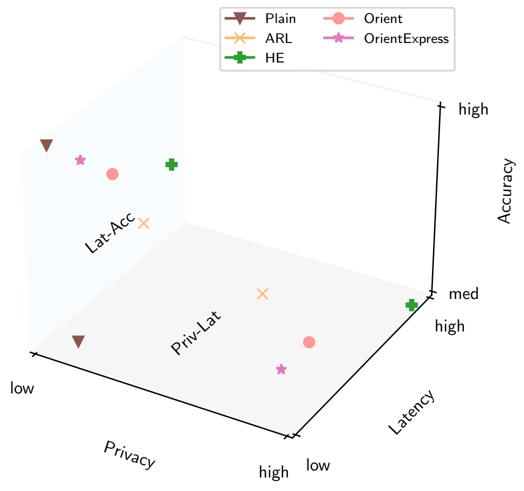


Figure 9: Comparison between various privacy-preserving methods in terms of *accuracy*, *latency*, and *privacy*. 2D projections of the points are shown in latency-accuracy and privacy-latency planes.

emulator takes a network (e.g., VGG16) and a *configuration* file to model the system and estimate the overall latency. The configuration file for each method consists of FLOPs per second, latency for linear and non-linear layers, communication delay between different components (e.g., TEE to GPU, Memory to TEE, etc.), and network delay. Using these files and the input, our code then faithfully models any given method. For each method, we use the numbers provided in each paper. Particularly, we model the following state-of-the-art methods: *1)Occlumency* [27] which improves the performance of the baseline TEE by adding a series of optimization features; *2)Slalom* [62] which uses TEE+GPU by employing blinding; *3)Origami* [45] which builds on top of Slalom and improves it by computing only a portion of the network on TEE; *4)CraterLake* [57] which leverages a specialized hardware accelerator to speed up the homomorphic operations; *5)Orient+TEE* which implements the encoder inside the TEE as opposed to using an HE; *6)Orient+CraterLake* which leverages the hypothetical accelerator for improving the computation speed of the (homomorphic) encoder. Additionally, we model two baselines: *Plain* (i.e., an inference that is running purely on a GPU with no privacy) and *TEE* which runs the entire inference on TEE. The code for our emulator is available online [4].

Results and Analysis. Using the setup described above, we compare the overall latency of each approach for a single inference on VGG16. The results are shown in Figure 10.

Compared to the baseline, Slalom incurs about a 12x slowdown. Origami improves Slalom by (approximately) a factor

[4]https://github.com/ssysarch/orient.git

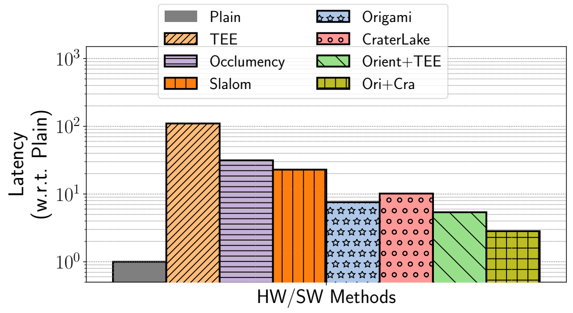


Figure 10: Simulation results for comparing `Orient` +X with other state-of-the-art TEE/HW-based methods.

of 3. Occlumency improves the TEE baseline by about 3.5x, however, it is slower than both Slalom and Origami since *all* of the computation is performed inside the TEE. CraterLake incurs about 10x overhead compared to the baseline which makes it slightly faster than Slalom and Occlumency but slower than Origami.

We then simulate our model using this setup. When using `Orient` with TEE (i.e., instead of implementing the encoder using HE, the encoding is performed inside the TEE), the simulation results show that *our method improves Origami, Occlumency, and Slalom by at least 40%*. The improvement comes from the fact that `Orient` employs ARL for encoding and further leverages an iterative algorithm to minimize the size of the encoder and hence reduce the overhead.

By combining `Orient` with an accelerator (i.e., offloading the encoder to specialized hardware), the latency further improves by about 90%. Compared to using *only* an accelerator, `Orient` + accelerator has 5x lower latency, further indicating that `Orient` can provide a better privacy-latency tradeoff, and can benefit from these hardware features.

The results reported here are based on an average network latency (cf. §5). Slower network delays could affect the results for all methods. The overhead of a network with higher latency is more noticeable in Orient and OrientExpress, as they both require an extra round of communication between IoT devices and the cloud. However, we believe the impact would be minimal, as the majority of the end-to-end delay is due to computation, not communication.

8 RELATED WORK

There are several methods for achieving privacy. We briefly review the most relevant ones and highlight their differences with this work when needed. We also refer the readers to the discussion in §2 and Table 1 regarding related work.

Encoding for Privacy. An emerging solution is to use *encoding* for hiding private data by adding a pre-processing phase. Adversarial Representation Learning (ARL) is a widely used encoding method [2, 14, 28, 29, 39, 47, 55, 60, 66, 67, 71].

It was first introduced by Xie *et al.* [68]. Madras *et al.* [36] further optimized it by leveraging the notions of fairness and transfer learning. The *Max Entropy* approach [52] proposed a modification to the training algorithm by introducing a new *non-zero-sum* game between three players. Recently, *ARL-Forgetting* [21] uses *masking* to improve the encoder's ability in removing private information. More recently, Singh *et al.* [59] proposed a new filter-based approach to improve the state-of-the-art.

Apart from ARL, there are other methods leveraging encoding for providing privacy. These methods either rely on adding noise to encode the input data [40, 48, 65] or utilizing ad-hoc encoding mechanisms to provide privacy [19].

In §6, we extensively compared our method against state-of-the-art encoding methods and showed that it has superior latency, privacy, and accuracy. It is important to mention that further advancements in encoding can also directly benefit `Orient` since a more effective encoder could improve privacy in our settings too. Thus, while our method used the training algorithm described in §3, a different (newer) loss function could be used during the training in the future.

Hardware-Support for Privacy. Another set of work on privacy focus on leveraging hardware support and particularly trusted execution environment (TEE) [8, 18, 20, 23, 27, 31, 42, 43, 62, 73]. Slalom [62] uses TEE-GPU collaboration to protect data privacy and integrity. Its performance is further improved by using a more sophisticated mapping proposed by DarKnight [18]. Occlumency [27] addresses the performance issues of deep learning in TEEs and proposes a series of solutions to improve the performance. Origami [45] further improves the performance of these two methods by executing only a fraction of inference inside a TEE. Telekine [20] improves the security of TEE-based approaches by using a new scheduling technique.

In §7, we compare the performance of our method with these proposed solutions. We showed that `Orient` can achieve better performance since it fundamentally minimizes the encoder and consequently the overhead.

HE and Cryptography for Privacy. Leveraging cryptography algorithms including homomorphic encryption (HE), multi-party computation (MPC), and their hybrid combinations, is a popular method for achieving privacy [12, 22, 24, 26, 30, 32–35, 41, 46, 54, 61, 72]. At the software-system level, Cryptonets [13] was the first to propose the usage of HE for ML privacy. It has been further improved using various scheduling and encoding techniques [26, 35]. Hybrid methods have improved the performance of HE-based methods. For example, Gazelle [22] proposed a hybrid method where linear layers are computed homomorphically while non-linear layers are evaluated locally using an MPC algorithm. More recently, Delphi [41] improved the state-of-the-art by utilizing a more efficient hybrid algorithm. Further, different cryptography techniques, such as functional encryption [54], have been used to enable a new class of privacy-preserving applications.

At the hardware-system level, the focus has been on designing *hardware-friendly* algorithms [26, 46, 61] Additionally, performance has been further improved by designing specialized accelerators [50, 51, 56, 57].

Compared to these methods, in §7, we showed that using `Orient` + accelerator can significantly improve the latency (by 500%) and bring it very close to the plaintext latency.

9 CONCLUSIONS

In this paper, we proposed a new lightweight method for privacy-preserving machine learning inference. We used homomorphic encryption to implement cloud-side encoding. Since the majority of computations are performed on the plaintext, our method features an advantage over existing encryption-based methods as it has a lower latency. Meanwhile, the method reduces the burden of encoding on the IoT device, thereby improving latency AND privacy.

We implemented our system, `Orient`, on a real-world edge-cloud setup using two popular IoT devices and measured its effectiveness using various important metrics including *inference accuracy*, *privacy*, and *end-to-end latency*.

Additionally, we proposed `OrientExpress` to extend our system's applicability to high-end IoT devices and improve latency, accuracy, and energy efficiency. Further, we compared our systems with state-of-the-art to highlight their advantages. `Orient` and `OrientExpress` provide the right balance in privacy-accuracy-latency space. Our evaluations showed that for low-end IoT devices where on-device computation has a large overhead, `Orient` improves the latency and accuracy while maintaining privacy. Additionally, it showed that for more sophisticated IoT devices, `OrientExpress` creates a balance between energy efficiency and other important metrics. Furthermore, comparisons with hardware-based solutions including TEEs and accelerators showed that `Orient` could benefit from these approaches and a systematic combination of these methods could create a more efficient solution. Collectively, our methods are excellent candidates for protecting data/sensor privacy on both low-end and high-end IoT devices with computing and/or energy constraints.

ACKNOWLEDGEMENT

We thank our shepherd for their guidance. This work has been supported, in part, by NSF grant CNS-2211301 and a gift received from Accenture. The views and findings in this paper are those of the authors and do not necessarily reflect the views of NSF and Accenture.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 308–318.

[2] Mert Al, Semih Yagli, and Sun-Yuan Kung. 2021. Privacy Enhancing Machine Learning via Removal of Unwanted Dependencies. IEEE Transactions on Neural Networks and Learning Systems (2021).

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) 6, 3 (2014), 1–36.

[4] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. 2019. Low latency privacy preserving inference. In International Conference on Machine Learning. PMLR, 812–821.

[5] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp). Ieee, 39–57.

[6] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In International conference on the theory and application of cryptology and information security. Springer, 409–437.

[7] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: fast fully homomorphic encryption over the torus. Journal of Cryptology 33, 1 (2020), 34–91.

[8] Tarek Elgamal and Klara Nahrstedt. 2020. Serdab: An IoT framework for partitioning neural networks computation across multiple enclaves. In 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). IEEE, 519–528.

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

[10] Ruiyuan Gao, Hailong Yang, Shaohan Huang, Ming Dun, Mingzhen Li, Zerong Luan, Zhongzhi Luan, and Depei Qian. 2021. Pripro: towards effective privacy protection on edge-cloud system running dnn inference. In 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE, 334–343.

[11] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Annual Cryptology Conference. Springer, 75–92.

[12] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. 2020. Cryptonas: Private inference on a relu budget. Advances in Neural Information Processing Systems 33 (2020), 16961–16971.

[13] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In International conference on machine learning. PMLR, 201–210.

[14] Xavier Gitiaux and Huzefa Rangwala. 2021. Fair representations by compression. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35. 11506–11515.

[15] Aayush Gupta, Ayush Jaiswal, Yue Wu, Vivek Yadav, and Pradeep Natarajan. 2021. Adversarial Mask Generation for Preserving Visual Privacy. In 2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021). IEEE, 1–5.

[16] Umang Gupta. 2022. Code for "Controllable Guarantees for Fair Outcomes via Contrastive Information Estimation". https://github.com/umgupta/fairness-via-contrastive-estimation

[17] Umang Gupta, Aaron M Ferber, Bistra Dilkina, and Greg Ver Steeg. 2021. Controllable guarantees for fair outcomes via contrastive information estimation. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35. 7610–7619.

[18] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. 2021. DarKnight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. 212–224.

[19] Yangsibo Huang, Zhao Song, Kai Li, and Sanjeev Arora. 2020. Instahide: Instance-hiding schemes for private distributed learning. In International conference on machine learning. PMLR, 4507–4518.

[20] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J Rossbach, and Emmett Witchel. 2020. Telekine: Secure Computing with Cloud {GPUs}. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). 817–833.

[21] Ayush Jaiswal, Daniel Moyer, Greg Ver Steeg, Wael AbdAlmageed, and Premkumar Natarajan. 2020. Invariant representations through adversarial forgetting. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 4272–4279.

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

[23] Kyungtae Kim, Chung Hwan Kim, Junghwan" John" Rhee, Xiao Yu, Haifeng Chen, Dave Tian, and Byoungyoung Lee. 2020. Vessels: Efficient and scalable deep learning prediction on trusted processors. In Proceedings of the 11th ACM Symposium on Cloud Computing. 462–476.

[24] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. Advances in Neural Information Processing Systems 34 (2021), 4961–4973.

[25] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. Technical Report. https://www.cs.toronto.edu/~kriz/cifar.html

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

[27] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. 2019. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In The 25th Annual International Conference on Mobile Computing and Networking. 1–17.

[28] Ang Li, Yixiao Duan, Huanrui Yang, Yiran Chen, and Jianlei Yang. 2020. TIPRDC: task-independent privacy-respecting data crowdsourcing framework for deep learning with anonymized intermediate representations. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 824–832.

[29] Ang Li, Jiayi Guo, Huanrui Yang, Flora D Salim, and Yiran Chen. 2021. DeepObfuscator: Obfuscating intermediate representations with privacy-preserving adversarial learning on smartphones. In Proceedings of the International Conference on Internet-of-Things Design and Implementation. 28–39.

[30] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. 619–631.

[31] Renju Liu, Luis Garcia, Zaoxing Liu, Botong Ou, and Mani Srivastava. 2021. SecDeep: Secure and Performant On-device Deep Learning Inference Framework for Mobile and IoT Devices. In Proceedings of the International Conference on Internet-of-Things Design and Implementation. 67–79.

[32] Qian Lou, Bo Feng, Geoffrey Charles Fox, and Lei Jiang. 2020. Glyph: Fast and accurately training deep neural networks on encrypted data.

Advances in Neural Information Processing Systems 33 (2020), 9193–9202.

[33] Qian Lou and Lei Jiang. 2019. She: A fast and accurate deep neural network for encrypted data. Advances in Neural Information Processing Systems 32 (2019).

[34] Qian Lou and Lei Jiang. 2021. Hemet: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In International conference on machine learning. PMLR, 7102–7110.

[35] Qian Lou, Wen-jie Lu, Cheng Hong, and Lei Jiang. 2020. FALCON: fast spectral inference on encrypted data. Advances in Neural Information Processing Systems 33 (2020), 2364–2374.

[36] David Madras, Elliot Creager, Toniann Pitassi, and Richard Zemel. 2018. Learning adversarially fair and transferable representations. In International Conference on Machine Learning. PMLR, 3384–3393.

[37] Microsoft. 2022. CryptoNets. https://github.com/microsoft/CryptoNets

[38] Microsoft. 2022. Microsoft SEAL. https://github.com/Microsoft/SEAL

[39] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Ali Jalali, Ahmed Taha Taha Elthakeb, Dean Tullsen, and Hadi Esmaeilzadeh. 2021. Not all features are equal: Discovering essential features for preserving prediction privacy. In Proceedings of the Web Conference 2021. 669–680.

[40] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Prakash Ramrakhyani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. 2020. Shredder: Learning noise distributions to protect inference privacy. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 3–18.

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

[42] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. 94–108.

[43] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. DarkneTZ: towards model privacy at the edge using trusted execution environments. In Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services. 161–174.

[44] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In 2017 IEEE symposium on security and privacy (SP). IEEE, 19–38.

[45] Krishna Giri Narra, Zhifeng Lin, Yongqin Wang, Keshav Balasubramanian, and Murali Annavaram. 2021. Origami inference: Private inference using hardware enclaves. In 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). IEEE, 78–84.

[46] Lucien KL Ng and Sherman SM Chow. 2021. {GForce}:{GPU-Friendly} Oblivious and Rapid Neural Network Inference. In 30th USENIX Security Symposium (USENIX Security 21). 2147–2164.

[47] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, Ali Taheri, Kleomenis Katevas, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. 2020. A hybrid deep learning architecture for privacy-preserving mobile analytics. IEEE Internet of Things Journal 7, 5 (2020), 4505–4518.

[48] Seyed Ali Osia, Ali Taheri, Ali Shahin Shamsabadi, Kleomenis Katevas, Hamed Haddadi, and Hamid R Rabiee. 2018. Deep private-feature extraction. IEEE Transactions on Knowledge and Data Engineering 32, 1 (2018), 54–66.

[49] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In 2016 IEEE European symposium on security and privacy (EuroS&P). IEEE, 372–387.

[50] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T Lee, Hsien-Hsin S Lee, Gu-Yeon Wei, and David Brooks. 2021. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 26–39.

[51] M Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. HEAX: An architecture for computing on encrypted data. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 1295–1309.

[52] Proteek Chandan Roy and Vishnu Naresh Boddeti. 2019. Mitigating information leakage in image representations: A maximum entropy approach. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2586–2594.

[53] Sujoy Sinha Roy, Furkan Turan, Kimmo Jarvinen, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data. In 2019 IEEE International symposium on high performance computer architecture (HPCA). IEEE, 387–398.

[54] Théo Ryffel, David Pointcheval, Francis Bach, Edouard Dufour-Sans, and Romain Gay. 2019. Partially encrypted deep learning using functional encryption. Advances in Neural Information Processing Systems 32 (2019).

[55] Bashir Sadeghi, Runyi Yu, and Vishnu Boddeti. 2019. On the global optima of kernelized adversarial representation learning. In Proceedings of the IEEE/CVF International Conference on Computer Vision. 7971–7979.

[56] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A fast and programmable accelerator for fully homomorphic encryption. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. 238–252.

[57] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. 2022. CraterLake: a hardware accelerator for efficient unbounded computation on encrypted data.. In ISCA. 173–187.

[58] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In 2017 IEEE symposium on security and privacy (SP). IEEE, 3–18.

[59] Abhishek Singh, Ayush Chopra, Ethan Garza, Emily Zhang, Praneeth Vepakomma, Vivek Sharma, and Ramesh Raskar. 2021. DISCO: Dynamic and Invariant Sensitive Channel Obfuscation for deep neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 12125–12135.

[60] Congzheng Song and Vitaly Shmatikov. 2020. Overlearning Reveals Sensitive Attributes. In 8th International Conference on Learning Representations, ICLR 2020.

[61] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CryptGPU: Fast privacy-preserving machine learning on the GPU. In 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 1021–1038.

[62] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. arXiv preprint arXiv:1806.03287 (2018).

[63] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction {APIs}. In 25th USENIX security symposium (USENIX Security 16). 601–618.

[64] McKenzie van der Hagen and Brandon Lucia. 2022. Client-optimized algorithms and acceleration for encrypted compute offloading. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 683–696.

[65] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S Yu. 2018. Not just privacy: Improving performance of private deep learning in mobile cloud. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. 2407–2416.

[66] Hao Wu, Xuejin Tian, Minghao Li, Yunxin Liu, Ganesh Ananthanarayanan, Fengyuan Xu, and Sheng Zhong. 2021. PECAM: privacy-enhanced video streaming and analytics via securely-reversible transformation. In Proceedings of the 27th Annual International Conference on Mobile Computing and Networking. 229–241.

[67] Zhenyu Wu, Zhangyang Wang, Zhaowen Wang, and Hailin Jin. 2018. Towards privacy-preserving visual recognition via adversarial training: A pilot study. In Proceedings of the European Conference on Computer Vision (ECCV). 606–624.

[68] Qizhe Xie, Zihang Dai, Yulun Du, Eduard Hovy, and Graham Neubig. 2017. Controllable invariance through adversarial feature learning. Advances in neural information processing systems 30 (2017).

[69] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? Advances in neural information processing systems 27 (2014).

[70] Zhifei Zhang, Yang Song, and Hairong Qi. 2017. Age progression/regression by conditional adversarial autoencoder. In Proceedings of the IEEE conference on computer vision and pattern recognition. 5810–5818.

[71] Han Zhao, Jianfeng Chi, Yuan Tian, and Geoffrey J Gordon. 2020. Trade-offs and guarantees of adversarial representation learning for information obfuscation. Advances in Neural Information Processing Systems 33 (2020), 9485–9496.

[72] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. 2021. Cerebro: A Platform for {Multi-Party} Cryptographic Collaborative Learning. In 30th USENIX Security Symposium (USENIX Security 21). 2723–2740.

[73] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Boyan Zhao, Zhongpu Wang, Yuhui Zhang, Jiameng Ying, Lixin Zhang, et al. 2020. Enabling rack-scale confidential computing using heterogeneous trusted execution environment. In 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 1450–1465.