ASYMMETRICML: AN ASYMMETRIC DECOMPOSITION FRAMEWORK FOR PRIVACY-PRESERVING DNN TRAINING AND INFERENCE

Yue Niu
Dept. of Electrical and Computer Engineering
University of Southern California
Los Angeles, CA 90007, USA
yueniu@usc.edu

Salman Avestimehr
Dept. of Electrical and Computer Engineering
University of Southern California
Los Angeles, CA 90007, USA
avestimehr@ee.usc.edu

ABSTRACT

Leveraging specialized parallel hardware, such as GPUs, to conduct DNN training/inference significantly reduces time. However, data in these platforms is visible to any party, which, in certain circumstances, raises great concerns of data misuse. Trusted execution environments (TEEs) protect data privacy by performing training/inference in a secure environment, but at the cost of serious performance degradation. To bridge the gap between privacy and computing performance, we propose an *asymmetric* model-splitting framework, AsymmetricML, to (1) exploit computing power in specialized parallel hardware; and (2) preserve data privacy in TEEs during DNN training/inference. AsymmetricML asymmetrically splits a DNN model into two parts: the first part features most sensitive data information but less computation; while most computation is performed in the second part. Evaluations on typical models (VGG, ResNet) shows the framework delivers $5.9\times$ speedup in model inference, and $5.4\times$ in model training compared with TEE-only executions.

1 INTRODUCTION

Deep Neural Networks (DNNs) no doubt are emerging as an essential building block in various applications, such as computer vision (Simonyan & Zisserman, 2014; He et al., 2016; Zhu et al., 2017), natural language processing (Devlin et al., 2018). Efficiently training an large-scale DNN model usually requires (1) sufficient training dataset; (2) high-end parallel hardware, which are usually met in high-end distributed systems. In distributed or cloud-based applications, DNN training/inference is outsourced to remote computing servers, which requires data to be shared among untrusted platforms. This practice brings serious concerns that data, especially if it contains sensitive information, might be leaked to public and misused.

This poses a new level of challenge when training DNN models: keep computing services unaware of what input data is when using the services to speed up model training/inference. Current methods such as differential privacy (Abadi et al., 2016) and federated learning (Konečnỳ et al., 2015), though to some extent protect privacy, do not directly attack this issue. Trusted execution environments (TEEs), like Intel Software Guard Extensions (SGX) (Costan & Devadas, 2016) and Arm TrustZone (Alves, 2004), is a practical platform which can directly addresses this challenge in hardware. By conducting all computation in TEEs, the server can effectively protect data privacy from malicious attackers. However, computing performance and training/inference speed is severely affected. Tramer & Boneh (2018) though proposes a framework to preserve privacy using Intel SGX platform and GPUs during inference, private DNN training is still an open problem.

To efficiently perform private DNN training and inference in a *heterogeneous system with high-end parallel hardware and security-enabled processors*, this paper proposes a novel framework, asymmetricML, to *asymmetrically* split DNN models into two parts. The first part keeps most data information while only involves a small amount of computation; and the second part holds most computation with little information potentially leaked. In this way, the DNN model can be executed

by combining security-enabled processors and untrusted high-end parallel hardware, in which the TEE protects privacy, while high-end hardware guarantees computing performance.

Our contributions are detailed in the following aspects:

- A model-splitting algorithm is proposed to decompose a model into two parts which effectively decouples sensitive information from computation;
- To measure performance, the AsymmetricML framework is implemented in a heterogeneous system with security-enable CPUs and high-end GPUs which supports various DNN models.

Experiments on VGG16/19 and ResNet variants show this framework delivers $5.9\times$ speedup in model inference, and $5.4\times$ in model training than in TEE-only executions.

2 ASYMMETRIC MODEL-SPLITTING ALGORITHM

For a convolutional layer with input $\mathbf{X} \in \mathbb{R}^{b\times c_\text{i}\times H_\text{i}\times W_\text{i}}$ and kernel $\mathbf{W} \in \mathbb{R}^{c_\text{o}\times c_\text{i}\times k\times k}$, each output channel is calculated as $\mathbf{Y}_{i_b,i_c,:,:} = \text{Conv}(\mathbf{X}_{i_b,:,:,:}, \mathbf{W}_{i_c,:,:,:}) = \sum_{j=1}^{c_\text{i}} \mathbf{X}_{i_b,j,:,:} \circledast \mathbf{W}_{i_c,j,:,:}$ (bias is omitted), where b is batch size, c_i, c_o is the total number of input/output channels, and H_i, W_i denotes image sizes. To simplify notations, the slice of tensors such as $\mathbf{X}_{i_b,:,:,:}$ is reduced to $\mathbf{X}_{i_b}$.

The central part in the model-splitting algorithm is to break down Conv into Trusted and Untrusted parts, $\mathbf{Y}_{i_b,i_c} = \text{Conv}_\text{T}(\mathbf{X}^{(\text{T})}_{i_b}, \mathbf{W}_{i_c}) + \text{Conv}_\text{U}(\mathbf{X}^{(\text{U})}_{i_b}, \mathbf{W}_{i_c})$, with Conv_T performed in TEEs, and Conv_U offloaded onto GPUs. Furthermore, this decomposition is proceeded in an *asymmetric* manner, in which information in $\mathbf{X}^{(\text{T})}_{i_b}$ is much more than in $\mathbf{X}^{(\text{U})}_{i_b}$, while rank of $\mathbf{X}^{(\text{T})}_{i_b}$ is much smaller than rank of $\mathbf{X}^{(\text{U})}_{i_b}$. Hence, the trusted platform keeps most sensitive information from potential attackers with manageable computation overhead (explained in Sec 2.1). On the other hand, most computation is offloaded onto untrusted platforms to accelerate both inference and training.

2.1 SVD-BASED MODEL-SPLITTING ALGORITHM

The cornerstone in the algorithm is to find an *asymmetric* method to decompose input $\mathbf{X}$, which is achievable due to high correlation among input channels in $\mathbf{X}$.

To begin with, each $\mathbf{X}_{ib}$ is flattened into a matrix $\overline{\boldsymbol{X}_{i_b}} \in \mathbb{R}^{c_\text{i}\times H_\text{i}W_\text{i}}$. By applying SVD, we get transformed "input channels", $\boldsymbol{V}_{i_b}$, and corresponding singular values in $\boldsymbol{S}_{i_b}$, as shown in Eq (1).

$$\overline{\boldsymbol{X}_{i_b}} = \boldsymbol{U}_{i_b} \cdot \boldsymbol{S}_{i_b} \cdot \boldsymbol{V}^T_{i_b}. \tag{1}$$

By reshaping each column $j^{'}$ in $\boldsymbol{V}_{i_b}$ back to a matrix $\mathbf{X}^{'}_{i_b,j'} \in \mathbb{R}^{H_\text{i}\times W_\text{i}}$, $\mathbf{X}_{i_b,i_c}$ is rewritten as:

$$\mathbf{X}_{i_b,i_c} = \sum_{j'=1}^{c_\text{i}} \sigma_{i_b,j'} \cdot \boldsymbol{u}_{i_b,j'}(i_c) \cdot \mathbf{X}^{'}_{i_b,j'} \equiv \sum_{j'=1}^{c_\text{i}} a_{i_b,i_c,j'} \cdot \mathbf{X}^{'}_{i_b,j'}, \tag{2}$$

where $\sigma_{j'} = \boldsymbol{S}_{i_b}(j^{'}, j^{'})$, and $\boldsymbol{u}_{i_b,j'}(i_c) = \boldsymbol{U}_{i_b}(j^{'}, i_c)$ and $a_{i_b,i_c,j'}$ is the combined coefficient.

Then, according to the sigular value of each channel, $\mathbf{X}_{i_b,i_c}$ can be further split as in Eq (3).

$$\mathbf{X}_{i_b,i_c} \equiv \mathbf{X}^{(\text{T})}_{i_b,i_c} + \mathbf{X}^{(\text{U})}_{i_b,i_c} \equiv \sum_{j'=1}^{r} a_{i_b,i_c,j'} \cdot \mathbf{X}^{'}_{i_b,j'} + \sum_{j'=r+1}^{c_\text{i}} a_{i_b,i_c,j'} \cdot \mathbf{X}^{'}_{i_b,j'}, \tag{3}$$

where $r \ll c_\text{i}$ denotes the number of *most* "principle channels" stored in $\mathbf{X}^{(\text{T})}$.

With the decomposed input, the forward and backward operation in a convolutional layer can be accordingly *decomposed*. During forward, convolution operation can be decomposed as:

in trusted platform,

$$\mathbf{Y}^{(\text{T})}_{i_b,i_c} = \sum_{j=1}^{c_\text{i}} \mathbf{X}^{(\text{T})}_{i_b,j} \circledast \mathbf{W}_{i_c,j} = \sum_{j'=1}^{r} \mathbf{X}^{'}_{i_b,j'} \circledast \sum_{j=1}^{c_\text{i}} a_{i_b,j,j'} \mathbf{W}_{i_c,j} = \sum_{j'=1}^{r} \mathbf{X}^{'}_{i_b,j'} \circledast \mathbf{W}^{'}_{i_b,i_c,j'}; \tag{4a}$$

in untrusted platform,

$$\mathbf{Y}^{(\mathrm{U})}_{i_b,i_c} = \sum_{j=1}^{c_\mathrm{i}} \mathbf{X}^{(\mathrm{U})}_{i_b,j} \circledast \mathbf{W}_{i_c,j}. \tag{4b}$$

For backpropagation, given $\frac{\partial L}{\partial \mathbf{Y}}$ obtained from previous layer, $\frac{\partial L}{\partial \mathbf{X}}$ can be calculated as:

$$\frac{\partial L}{\partial \mathbf{X}_{i_b,i_c}} = \sum_{j=1}^{c_\mathrm{o}} \frac{\partial L}{\partial \mathbf{Y}_{i_b,j}} \circledast \mathbf{W}_{j,i_c \,\mathrm{rot}180^\circ} \tag{5}$$

Since computing $\frac{\partial L}{\partial \mathbf{X}}$ usually will not reveal information in X, it can be fully offloaded to untrusted platform.

On the other hand, computing $\frac{\partial L}{\partial \mathbf{W}}$ involves both private and public data:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{W}_{i_{c_\mathrm{o}},i_{c_\mathrm{i}}}} &= \sum_{i_b=1}^{b} \mathbf{X}^{(\mathrm{T})}_{i_b,i_{c_\mathrm{i}}} \circledast \frac{\partial L}{\partial \mathbf{Y}_{i_b,i_{c_\mathrm{o}}}} + \sum_{i_b=1}^{b} \mathbf{X}^{(\mathrm{U})}_{i_b,i_{c_\mathrm{i}}} \circledast \frac{\partial L}{\partial \mathbf{Y}_{i_b,i_{c_\mathrm{o}}}} \\ &= \sum_{i_b=1}^{b} (\sum_{j'=1}^{r} a_{i_b,i_{c_\mathrm{i}},j'} (\mathbf{X}^{'}_{i_b,j'} \circledast \frac{\partial L}{\partial \mathbf{Y}_{i_b,i_{c_\mathrm{o}}}})) + \mathbf{X}^{(\mathrm{U})}_{i_b,i_{c_\mathrm{i}}} \circledast \frac{\partial L}{\partial \mathbf{Y}_{i_b,i_{c_\mathrm{o}}}} \end{aligned} \tag{6}$$

The first part in Eq (6) is done in trusted platform, in which $\mathbf{X}^{'}_{i_b,j'} \circledast \frac{\partial L}{\partial \mathbf{Y}_{i_b,i_{c_\mathrm{o}}}}$ for $j' = 1, \cdots, r$ is pre-computed in TEEs; then $\mathbf{X}^{(\mathrm{T})}_{i_b,i_{c_\mathrm{i}}} \circledast \frac{\partial L}{\partial \mathbf{Y}_{i_b,i_{c_\mathrm{o}}}}$ is obtained by multiplying with the coefficient $a_{i_b,i_{c_\mathrm{i}},j'}$. Therefore computation complexity in TEEs is less than $\frac{r}{c_\mathrm{i}}$ in GPUs.

2.2 Light-weight SVD approximation

One issue is that "principle channels" change after non-linear operations, which needs to be re-calculated from output activations. It is, however, unwise to perform exact SVD in TEEs. In this section, we present a light-weight SVD approximation to obtain "principle channels".

From optimization perspective, SVD can be viewed as finding vector $\boldsymbol{u}^{(i)}$ and $\boldsymbol{v}^{(i)}$ to minimize $\left\| \mathbf{X}^{(i-1)} - \boldsymbol{u}^{(i)} \cdot \boldsymbol{v}^{(i)^T} \right\|$, in which $\mathbf{X}^{(i-1)}$ is the remaining $\mathbf{X}$ with $i-1$ most principle components extracted. SVD requires $\left\{\boldsymbol{u}^{(i)}\right\}$ and $\left\{\boldsymbol{v}^{(i)}\right\}$ to be orthogonal. If relaxing this constraint, the optimization for calculating the r principle components can be formulated as Eq (7).

$$\mathbf{X}^{(\mathrm{T})} = \arg\min_{\mathbf{X}^{(\mathrm{T})}} \left\| \mathbf{X} - \mathbf{X}^{(\mathrm{T})} \right\|, \mathrm{rank}(\mathbf{X}^{(\mathrm{T})}) \leq r \tag{7a}$$

$$\mathbf{X}^{(\mathrm{U})} = \mathbf{X} - \mathbf{X}^{(\mathrm{T})} \tag{7b}$$

Using alternating optimization(Bezdek & Hathaway, 2003), each component i in $\mathbf{X}^{(\mathrm{T})}$ can be obtained as in Alg 1. With initial value from before-nonlinear operations, only $1 \sim 2$ iterations are sufficient to reach optimal $\left\{\boldsymbol{u}^{(i)}\right\}$ and $\left\{\boldsymbol{v}^{(i)}\right\}$. The computation complexity just increases linearly with r.

2.3 Design flow

Figure 1 shows a complete procedure when decomposing a model. Each convolutional layer is decomposed and distributed to TEEs (Private flow) and GPUs (Public flow), after which outputs $\mathbf{Y}^{(\mathrm{T})}, \mathbf{Y}^{(\mathrm{U})}$ are merged in TEEs. When a convolutional layers is done, a non-linear is applied, and a light-weight SVD is then performed to prepared inputs for the next convolutional layer. At any time, only data in GPUs is visible to public. Data in Conv_T and merged outputs $\mathbf{Y}$ are always secured in TEEs.

Algorithm 1 Light-weight SVD Approximation

Input: $r, \mathbf{X}, \left\{\boldsymbol{u}^{(0)}, \boldsymbol{v}^{(0)} \quad | \quad i = 1, \cdots, r\right\}$

1: Initialize $\mathbf{X}^{(\mathrm{T})} = \oslash$
2: **for** i **in** $1, \cdots, r$ **do**
3: **for** j **in** $1, \cdots,$ max_iter **do** ▷ max_iter ≤ 2
4: $\boldsymbol{u}_j^{(i)} = \frac{\mathbf{X} \cdot \boldsymbol{v}_{j-1}^{(i)}}{\left\|\boldsymbol{v}_{j-1}^{(i)}\right\|^2}, \quad \boldsymbol{v}_j^{(i)} = \frac{\mathbf{X}^T \cdot \boldsymbol{u}_j^{(i)}}{\left\|\boldsymbol{u}_j^{(i)}\right\|^2}$ ▷ Alternating optimization
5: $\mathbf{X} = \mathbf{X} - \boldsymbol{u}_j^{(i)} \cdot \boldsymbol{v}_j^{(i)^T}, \quad \mathbf{X}^{(\mathrm{T})} = \mathbf{X}^{(\mathrm{T})} \cup \left\{\boldsymbol{u}_j^{(i)}, \boldsymbol{v}_j^{(i)}\right\}$
6: $\mathbf{X}^{(\mathrm{U})} = \mathbf{X}$

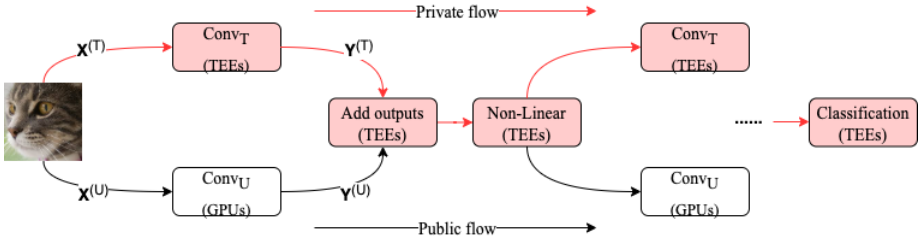

Figure 1: A complete procedure for decomposing a model

3 Experiments

In this section, we evaluate the asymmetric model-splitting algorithm in a hybrid system with Intel SGX (Costan & Devadas, 2016) enabled CPUs and NVIDIA GPUs. NVIDIA QUADRO RTX 5000 GPUs are used as an untrusted platform, while SGX-enabled CPUs as a trusted platform. ImageNet (Deng et al., 2009) is used to measure training/inference performance.

Fig 2 shows training time for different methods. During experiments, the number of "principle channels" r in each convolutional layer is chosen to keep $\geq 97\%$ energy of inputs in TEEs. Comparing with the SGX-only method (baseline2), the asymmetric DNN framework (Asym1) achieves $5.2\times$ speedup on VGG16/VGG19, and up to $4\times$ speedup on ResNet variants. If further relaxing privacy constraint (Asym2, 80% energy in TEEs), more performance gains can be squeezed out. Comparing with the GPU-only method (baseline1), the asymmetric DNN framework shows around $20\times$ slowdown. The main bottleneck is not computation cost in SGX, instead, it lies on the context switching between SGX and GPU, along with necessary data movement (parameters/activations/gradients). Comparing Asym1 with Asym2, even though computations in Asym1 is almost as twice as in Asym2, actual training time using Asym1 is very close to the one using Asym2.

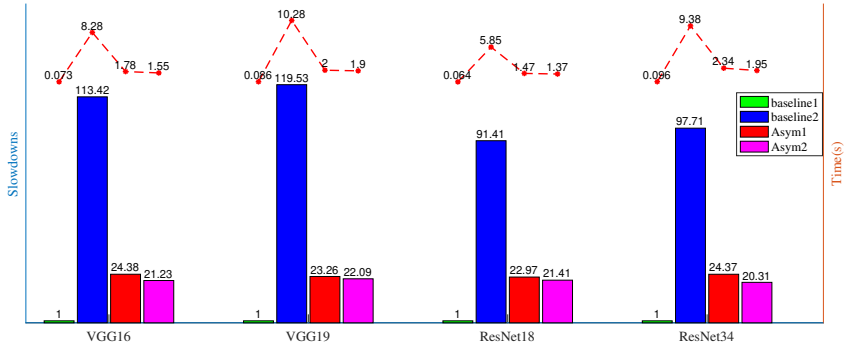

Figure 2: Training performance on ImageNet.Bar plot shows slowdowns compared to GPU-only (baseline1) execution; red dotted lines are the corresponding running time with batch size 32

4 CONCLUSION

In this paper, we propose an asymmetric decomposition framework to split DNN models and distribute models onto trusted platforms and untrusted platforms. The trusted platform is aimed to preserve most information in input data with manageable computation cost; the untrusted platform performs most computation. The asymmetric framework makes the best use of each platform in a hybrid system. Experiments show that the asymmetric decomposition framework achieves up to $5.9\times$ speedup in model inference and $5.4\times$ in training.

ACKNOWLEDGEMENTS

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053 and FA8750-19-2-1005, ARO award W911NF1810400, NSF grants CCF-1703575 and CCF-1763673, ONR Award No. N00014-16-1-2189, and a gift from Intel/Avast/Borsetta via the PrivateAI institute. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318, 2016.

Tiago Alves. Trustzone: Integrated hardware and software security. *White paper*, 2004.

James C Bezdek and Richard J Hathaway. Convergence of alternating optimization. *Neural, Parallel & Scientific Computations*, 11(4):351–368, 2003.

Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86): 1–118, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pp. 8026–8037, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

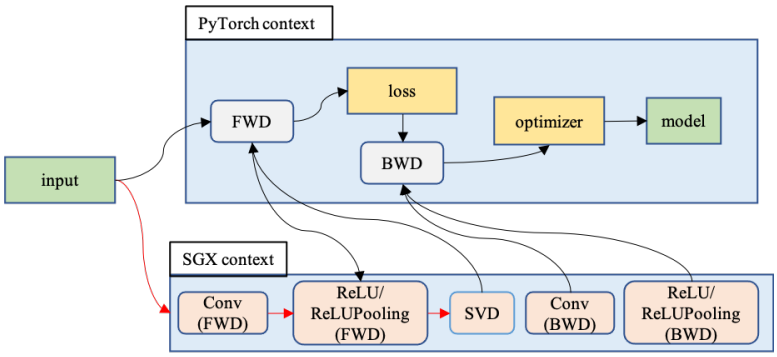

Figure 3: Asymmetric DNN implementation

A Appendix

A.1 Asymmetric DNN Implementation

Due to the fact that there is no efficient framework supported in SGX, a new framework is developed in this paper to automatically split DNN models and distribute computation into SGX enclaves and GPUs. The asymmetric DNN framework in this paper takes input from a PyTorch (Paszke et al., 2019) model definition, splits it into two parts and outsources to SGX enclaves and GPUs respectively. Part of model in SGX is executed using a dedicated DNN library, while the other part is executed using PyTorch, as shown in Fig 3. The DNN library in SGX enclaves currently supports forward (FWD) and backward (BWD) operations on convolution, ReLU, and Pooling. It also contains a light SVD op. Pooling can be further fused with a ReLU op to reduce memory access between SGX enclaves and GPUs, similar as (Tramer & Boneh, 2018).

During runtime, two running contexts are created: PyTorch and SGX context. PyTorch works as a coordinator to distribute computation, activate SGX context and update model parameters. For convolutional layers, GPUs and SGX enclaves work in parallel; for other operations, sequential execution is required.

A.2 Inference Performance

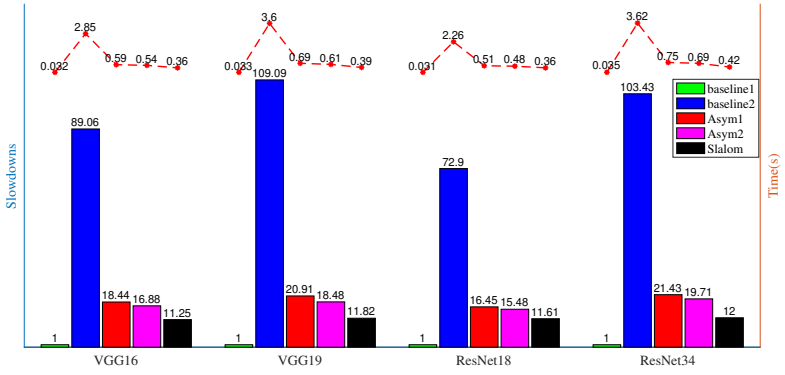

Figure 4: Inference performance on ImageNet. Bar plot shows slowdowns compared to GPU-only (baseline1) execution; red dotted lines are the corresponding running time with batch size 32.

We present inference results and compare the performance with Slalom (Tramer & Boneh, 2018). As shown in Fig 4, performance (slowdowns/running time) for VGG16/VGG19 and ResNet18/34 on ImageNet using different methods is measured. Baseline 1 is GPU-only execution; while baseline 2 is SGX-only execution in which the whole model is processed in SGX Enclave. The asymmetric DNN framework achieves $5.2\times$/$5.9\times$ faster than SGX-only execution on VGG16/VGG19,

and $4.7\times/5.2\times$ faster on ResNet18/ResNet34. Due to additional partial convolution processed in SGX, inference in our asymmetric DNN framework (Asym1/Asym2) is slower than Slalom, which is reasonable since Slalom only privatizes non-linear operations. However, due to the channel-wise low-rank characteristic in activations, the asymmetric DNN framework though compute partial convolutions, does not sacrifice too much performance.

Furthermore, in some scenarios, privacy can be to some extent compromised, in which small part of the information is allowed to be leaked without causing significant damage. In the asymmetric DNN framework, we achieve this goal by parameterizing information leakage: Asym1 in Fig 4 is aimed to keep almost all information in SGX, which incurs more computation cost. Asym2, on the other hand, only keeps half of the principle channels as in Asym1, by which around 20% information is exposed, but with the reward of 10% speedup. Another observations is as models go deep as VGG19 and ResNet34, relative performance using the asymmetric DNN framework decreases. This is due to the fact that more additional layers in SGX incurs more extra cost of calling SGX context, data movement between SGX Enclaves and GPUs.