Hardware Acceleration of Explainable Machine Learning

Zhixin Pan and Prabhat Mishra
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—Machine learning (ML) is successful in achieving human-level performance in various fields. However, it lacks the ability to explain an outcome due to its black-box nature. While recent efforts on explainable ML has received significant attention, the existing solutions are not applicable in real-time systems since they map interpretability as an optimization problem, which leads to numerous iterations of time-consuming complex computations. To make matters worse, existing implementations are not amenable for hardware-based acceleration. In this paper, we propose an efficient framework to enable acceleration of explainable ML procedure with hardware accelerators. We explore the effectiveness of both Tensor Processing Unit (TPU) and Graphics Processing Unit (GPU) based architectures in accelerating explainable ML. Specifically, this paper makes three important contributions. (1) To the best of our knowledge, our proposed work is the first attempt in enabling hardware acceleration of explainable ML. (2) Our proposed solution exploits the synergy between matrix convolution and Fourier transform, and therefore, it takes full advantage of TPU's inherent ability in accelerating matrix computations. (3) Our proposed approach can lead to real-time outcome interpretation. Extensive experimental evaluation demonstrates that proposed approach deployed on TPU can provide drastic improvement in interpretation time (39x on average) as well as energy efficiency (69x on average) compared to existing acceleration techniques.**

I. Introduction

Machine learning (ML) techniques powered by deep neural networks (DNNs) are pervasive across various application domains [1]–[4]. Recent advances in ML algorithms have enabled promising performance with outstanding flexibility and generalization. However, most of the existing ML methods are not able to interpret the outcome (e.g., explain its prediction) since it produces the outcome based on computations inside a "black-box". This lack of transparency severely limits the applicability of ML. Explainable ML provides interpretation of input-output mapping as well as clues for importance ranking of input features [1], [3]. As a result, explainable ML acts like a supervisor to guide the learning process and provides additional information to users.

Due to the inherent inefficiency in explainable ML algorithms, they are not applicable in real-time systems. These algorithms treat the explanation process as an extra procedure, and performs the interpretation outside the learning model, which makes them inefficient in practice. In this paper, we propose an efficient framework to achieve fast explainable ML utilizing various hardware accelerators, including Graphic Processing Units (GPU) and Tensor Processing Units (TPU). Specifically, we transform explainable ML procedure to computation of matrix operations, and exploit the natural ability of hardware accelerators for fast and efficient parallel computation of matrix operations. GPU consists of a large number of cores and high-speed memory for performing efficient matrix computation and parallel computing. Similarly, TPU is an Application Specific Integrated Circuit (ASIC) developed specifically to accelerate the computations in deep neural networks [5]–[7], with extremely high throughout and fast performance with low memory footprint. It is a compatible choice to deploy in our proposed framework.

This work was partially supported by the NSF grant CCF-1908131.

Our proposed approach effectively utilizes the synergy between matrix based representation of interpretation procedure and hardware-based acceleration of matrix operations. Specifically, this paper makes the following major contributions:

1) To the best of our knowledge, our proposed approach is the first attempt in hardware-based acceleration of explainable machine learning. Specifically, we investigate the effectiveness of Tensor Processing Unit (TPU) as well as Graphics Processing Unit (GPU) based architectures in accelerating explainable ML algorithms.
2) We propose an efficient mechanism to convert explainable machine learning task to linear algebra computation. As a result, it can exploit the inherent advantages of hardware accelerators in computing ultra-fast matrix operations.
3) Experimental evaluation using two popular ML models demonstrate that our proposed approach can achieve real-time explainable machine learning by providing fast outcome interpretation with hardware accelerators.

The rest of this paper is organized as follows. Section II surveys related efforts. Section III describes our proposed hardware acceleration framework. Section IV presents experimental results. Finally, Section V concludes the paper.

II. Background and Related Work

In this section, we survey related efforts on TPU/GPU-based acceleration. Both TPU and GPU architectures are widely used in accelerating diverse applications. For example, in [8], the author applied GPU-based accelerator to solve the optimal scheduling problem fast to achieve efficient energy storage in distributed systems. Similarly, many computationally expensive mathematical algorithms can be accelerated by TPU/GPU based implementation. The popularity of AI algorithms in recent years has led to many research efforts in hardware acceleration of machine learning algorithms. In [9], TPU was utilized on various Convolutional Neural Networks (CNN) to demonstrate its advantage in performing convolution operations. This advantage was further exploited in ML-based tasks like image reconstruction [10] and automatic path-finding [11]. *While GPU and TPU have been successfully used for accelerating machine learning algorithms, there are no prior efforts in utilizing GPU or TPU architectures in accelerating explainable machine learning.*

III. Hardware Acceleration of Explainable ML

Figure 1 shows an overview of our proposed framework for hardware acceleration of explainable machine learning. For a specific ML task, we apply traditional training scheme to construct a well-trained model and respective input-output dataset. Then we build a corresponding distilled model, which is able to provide reasonable explanation for target model's behavior. In this work, we consider three major tasks to achieve fast model distillation. First, we perform *task transformation* to map the explanation problem to Fourier transform computation by utilizing the inherent property of matrix convolution (Section III-A). Next, we develop two synergistic activities to accelerate the computation procedure of Fourier transform. The first activity performs *data decomposition* (Section III-B), where the complete computing task is split into multiple sub-tasks, and each sub-task can be executed by a GPU or TPU core without requiring any data exchange between cores (sub-tasks). The second activity fully exploits hardware accelerators' inherent ability in *parallel computation* to process multiple input-output pairs concurrently. Simultaneous execution of these two activities can provide significant improvement in acceleration efficiency, which is demonstrated in our experimental evaluation (Section IV).

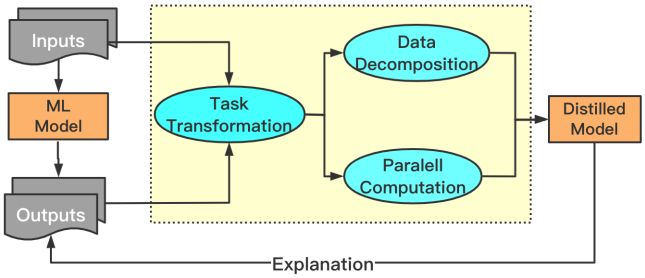

Fig. 1: Our framework consists of three major activities: task transformation, data decomposition and parallel computation.

A. Task Transformation

In this section, we demonstrate how to convert task into a matrix computation. Specifically, we use model distillation as explanation scheme, which aims at distilling features learned by a complex and cumbersome ML model, and use a lightweight "shadow" model to mimic its input-output mapping behavior, which is called as *distilled model*. Model distillation consists of the following three steps: model specification, model computation and outcome interpretation.

Model Specification: In this work, a regression model is applied in order to satisfy the two requirements outlined above. Formally, given input data $\mathbf{X}$ and output $\mathbf{Y}$, we find a matrix $\mathbf{K}$ using Equation 1, where "*" denotes the matrix convolution.

$$\mathbf{X} * \mathbf{K} = \mathbf{Y} \tag{1}$$

Since convolution is a linear-shift-invariant operation, the above regression guarantees the distilled model to be sufficiently lightweight and transparent. Under this scenario, the model computation task boils down to solving for the parameters in matrix $\mathbf{K}$.

Model Computation: To solve for $\mathbf{K}$, one key observation is that we can apply Fourier transformation on both sides of Equation 1, and by discrete convolution theorem, it gives

$$\begin{aligned} \mathbf{X} * \mathbf{K} &= \mathbf{Y} \\ \mathscr{F}(\mathbf{X} * \mathbf{K}) &= \mathscr{F}(\mathbf{Y}) \\ \mathscr{F}(\mathbf{X}) \circ \mathscr{F}(\mathbf{K}) &= \mathscr{F}(\mathbf{Y}) \end{aligned} \tag{2}$$

where $\circ$ is the Hadamard product. Therefore, the solution is given by this formula:

$$\mathbf{K} = \mathscr{F}^{-1}(\mathscr{F}(\mathbf{Y})/\mathscr{F}(\mathbf{X})) \tag{3}$$

Outcome Interpretation: The primary goal of explainable ML is to measure how each input feature contributes to the output value. Once $\mathbf{K}$ is obtained, the contribution of each feature can be viewed in an indirect way – consider a scenario where we remove this component from the original input, and let it pass through the distilled model again to produce a "perturbed" result. Then by calculating the difference between the original and newly generated outputs, the impact of the key feature on the output can be quantified. The intuition behind the assumption is that hiding important features are more likely to cause considerable changes to the model output. Formally, assume that the input is $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}..., \mathbf{x}_d]$. We define the contribution factor of $\mathbf{x}_i$ as

$$con(\mathbf{x}_i) \triangleq \mathbf{Y} - \mathbf{X}' * \mathbf{K} \tag{4}$$

where $\mathbf{X}' = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{i-1}, \mathbf{0}, \mathbf{x}_{i+1}..., \mathbf{x}_d]$, which is nothing but removing the target component from the original input.

As we can see, the original model distillation task has been converted into a matrix computation problem, which consists of matrix convolution, point-wise division and Fourier transform. The first two types of operations can be inherently accelerated by GPU or TPU's built-in structure [12]. The next section describes the details for accelerating Fourier transform.

B. Data Decomposition in Fourier Transform

In this section, we demonstrate how to apply data decomposition to disentangle Fourier transform computation, and further utilize computation resources to significantly accelerate the computing process. The general form of a 2-D Discrete Fourier Transform (DFT) applied on an $M \times N$ signal is defined as:

$$X[k,l] = \frac{1}{\sqrt{MN}} \sum_{n=0}^{N-1} \left[\sum_{m=0}^{M-1} x[m,n] e^{-j2\pi \frac{mk}{M}} \right] e^{-j2\pi \frac{nl}{N}} \tag{5}$$

where $k = 0, ..., M-1$, $l = 0, ..., N-1$.
If we define intermediate signal X' such that

$$X'[k,n] \triangleq \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} x[m,n] e^{-j2\pi \frac{mk}{M}} \tag{6}$$

and plug it into Equation 5, we have

$$X[k,l] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} X'[k,n] e^{-j2\pi \frac{nl}{N}} \tag{7}$$

Notice the similarity between Equation 6 and the definition of 1-D Fourier transform applied on a M-length vector:

$$X[k] = \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} x[m] e^{-j2\pi \frac{mk}{M}} \tag{8}$$

If we treat n as a fixed parameter, then application of Equation 6 is equivalent to performing a 1-D Fourier transform on the n-th column of the original input $M \times N$ matrix. Note that for 1-D Fourier transform, it can always be written

TABLE I: Comparison of accuracy and classification time for various benchmarks

	CPU-based Acceleration			GPU-based Acceleration			TPU-based Acceleration				
Benchmark	Accuracy (%)	Training-time(s)	Testing-time(s)	Accuracy (%)	Training-time(s)	Testing-time(s)	Accuracy (%)	Training-time(s)	Testing-time(s)	Speedup. /CPU	Speedup. /GPU
VGG19	94.06	24.2	10.9	92.08	0.25	0.08	96.37	0.4	0.14	65x	0.61x
ResNet50	78.99	176.2	129.8	86.87	19.1	9.4	87.47	4.3	2.6	44.5x	4.13x
Average	**86.52**	**100.2**	**70.35**	**89.47**	**9.67**	**4.84**	**91.92**	**2.35**	**1.37**	**54.7x**	**3.9x**

as a product of input vector and Fourier transform matrix. Therefore, we can rewrite Equation 6 as:

$$X'[k,n] = \mathbf{W}_M \cdot x[m,n] \tag{9}$$

where $\mathbf{W}_M$ is the $M \times M$ Fourier transform matrix. By varying n from 1 to $N-1$, we get:

$$X' = [X'[k,0], \cdots, X'[k,N-1]] = \mathbf{W}_M \cdot x \tag{10}$$

If we treat k as a parameter and view the definition of $X'[k,n]$ as the 1-D Fourier transform with respect to the k-th row of input x, a similar expression can be obtained using the above derivation steps as:

$$X = X' \cdot \mathbf{W}_N \tag{11}$$

where $\mathbf{W}_N$ is the $N \times N$ Fourier transform matrix. Using Equation 10, the final expression of X can be written as:

$$X = (\mathbf{W}_M \cdot x) \cdot \mathbf{W}_N \tag{12}$$

This transformed expression indicates that a 2-D Fourier transform can be achieved in a two-stage manner. First, transform all the rows of x to obtain intermediate result X'. Second, transform all the columns of the resulting matrix X'. An important observation is that the required computation for each row/column are completely independent. This implies that in real implementation, we can always split the computation process into sub-threads. Given p individual cores involved and a $M \times N$ matrix as input, every core is assigned at most $\frac{max\{M,N\}}{p}$ 1-D Fourier transform workload and can execute in parallel. Our analysis reveals that merging the results matches with the desired 2-D Fourier transform result.

IV. Experiments

A. Experimental Setup

Experiments were conducted on a host machine with Intel i7 3.70GHz CPU, equipped with an external NVIDIA GeForce RTX 2080 Ti GPU, which is considered as the state-of-the-art GPU accelerator for ML algorithms. We also utilize Google's Colab platform to access Google Cloud TPU service. In our evaluation, we used TPUv2 with 64 GB High Bandwidth Memory (HBM), and 128 TPU cores. We developed code using Python [13] for model training and PyTorch 1.6.0 [14] as the machine learning library. We have used the following two populor benchmarks in our study.

1) VGG19 [15] classifier for *CIFAR-100* classification.
2) ResNet50 [16] network for malware detection [1], [17].

We have compared the following three configurations to highlight the importance of our proposed hardware acceleration approach. To address the compatibility of proposed optimization approach, all the proposed optimization methods (task transformation, data decomposition, parallel computation) are deployed on all 3 accelerators:

1) **CPU**: Traditional execution in software, which is considered as baseline method.
2) **GPU**: NVIDIA GeForce 2080 Ti GPU, which is considered as state-of-the-art ML acceleration component.
3) **TPU**: Google's cloud TPU, a specific ASIC designed to accelerate machine learning procedure.

The model training process consists of 500 epochs in total, with a mini-batch size of 128. As for result evaluation, we first evaluated classification performance by reporting ML models' classification accuracy and execution time. Next, we compare the energy efficiency by measuring the performance per watt on each hardware under different workload. Then we report the average time for completing outcome interpretation step for each configuration. Finally, we present the effectiveness of our proposed method in interpreting classification results.

B. Comparison of Accuracy and Classification Time

Table I compares the classification time and accuracy. Each row represents a specific model structure trained with corresponding hardware configuration. For both training time and testing time, each entry represents time cost of 10 epochs on average. As we can see, with sufficient number of training epochs, all methods obtain reasonable classification accuracy. However, when it comes to time-efficiency, the CPU-based baseline implementation lags far behind the other two, which achieved the slowest speed. On VGG19, GPU provides the best acceleration performance, which provides 65x speedup compared to the baseline implementation. This clearly indicates the great compatability between hardware accelerator and our proposed framework. In case of ResNet50, an even higher speedup was obtained by TPU, showing its acceleration potential in large-scale neural networks by providing around four times speedup than GPU. The drastic improvement (44.5x) compared to the baseline method also leads to significant energy savings, as described in the next section.

C. Comparison of Energy Efficiency

Power consumption is another important aspect of performance evaluation, as power closely affects the thermal, provision and stability of the device. Consequently, designers should consider power consumption for methods deployed on hardware components to ensure their power constraints are satisfied. Figure 2 shows the geometric and weighted mean performance/Watt for the RTX 2080 Ti GPU and Google's TPU relative to the CPU. Similar to [18], we calculate performance/Watt in two different ways. The first one (referred as 'total') computes the total power consumption which consists of the power consumed by the host CPU as well as the actual execution performance/Watt for the GPU or TPU. The second one (referred as 'incremental') does not consider the host CPU power, and therefore, it reflects the actual power consumption of the GPU or TPU during acceleration. As we can see from Figure 2, for total-performance/watt, the GPU implementation

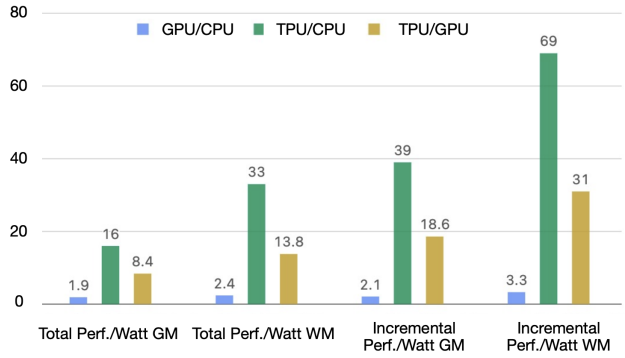

Fig. 2: Relative performance/Watt of GPU (blue bar) and TPU (green bar) over CPU, and TPU versus GPU (yellow bar). The total perf./watt includes host CPU power, while incremental ignores it. **GM** and **WM** are the geometric mean and weighted means, respectively.

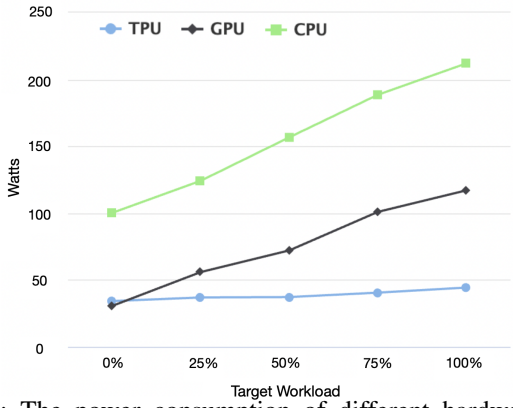

Fig. 3: The power consumption of different hardware with various workload situations.

is 1.9X and 2.4X better than baseline CPU for geometric mean (GM) and weighted mean (WM), respectively. TPU outperforms both CPU (16x on GM and 33X on WM) and GPU (8.4X on GM and 13.8x on WM) in terms of total performance/watt. For incremental-performance/watt, when host CPU's power is omitted, the TPU shows its dominance in energy efficiency over both CPU (39x on GM and 69X on WM) and GPU (18.6x on GM and 31X on WM).

In terms of energy efficiency, TPU outperforms GPU, which outperforms CPU. Our proposed method fully utilizes data decomposition to create a high-level parallel computing environment where both GPU an TPU benefit from it to balance the workloads on every single core. Although both GPU and TPU have the advantage of utilizing parallel computing to fulfill the proposed framework, TPU provides better performance/watt primarily due to the 'quantification' property of TPU. Quantification is a powerful mechanism to reduce the cost of neural network prediction as well as the reduction in memory. The use of integers instead of floating-point calculations greatly reduces the hardware size and power consumption of the TPU. Specifically, TPU can perform 65,536 8-bit integer multiplications in a cycle, while mainstream GPUs used in cloud environments can perform few thousands of 32-bit floating-point multiplications. As long as 8 bits can be used to meet the accuracy requirements, it can bring significant performance improvement. While both TPU and GPU based acceleration can achieve fast explainable machine learning, the TPU-based implementation is the best in terms of energy efficiency.

V. Conclusion

While explainable machine learning techniques are popular, their long running time severely restricts their applicability in many domains. In this paper, we address this fundamental bottleneck using hardware-based acceleration to provide explainability (transparency) of machine learning models in a reasonable time. This paper made several major contributions. We propose an efficient mechanism to transform the model distillation problem to linear algebra computation problem. The transformed model is able to fully exploit the inherent ability of hardware accelerators in computing ultra-fast matrix operations. Moreover, it enables parallel computing by performing data decomposition to break a large matrix into multiple small matrices. Experimental evaluation on a diverse set of benchmarks demonstrated that our approach is scalable and able to meet real-time constraints. Our studies reveal that our proposed framework can effectively utilize the inherent advantages of both TPU and GPU based architectures. Specifically, TPU-based acceleration provides drastic improvement in interpretation time (39x over CPU and 4x over GPU) as well as energy efficiency (69x over CPU and 31x over GPU) for both image classification and malware detection benchmarks.

References

[1] Z. Pan, J. Sheldon, and P. Mishra, "Hardware-assisted malware detection using explainable machine learning," in *ICCD*, 2020, pp. 663–666.

[2] ——, "Test generation using reinforcement learning for delay-based side-channel analysis," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–7.

[3] Z. Pan and P. Mishra, "Automated detection of spectre and meltdown attacks using explainable machine learning," in *HOST*, 2021.

[4] ——, "Automated test generation for hardware trojan detection using reinforcement learning," in *ASPDAC*, 2021, pp. 408–413.

[5] T. Lu *et al.*, "Large-scale discrete fourier transform on tpus," *CoRR*, vol. abs/2002.03260, 2020.

[6] T. Lu, T. Marin, Y. Zhuo, Y. Chen, and C. Ma, "Accelerating MRI reconstruction on tpus," *CoRR*, vol. abs/2006.14080, 2020.

[7] J. Civit-Masot *et al.*, "TPU cloud-based generalized u-net for eye fundus image segmentation," *IEEE Access*, vol. 7, pp. 142 379–142 387, 2019.

[8] D. Sidea *et al.*, "Optimal battery energy storage system scheduling based on mutation-improved grey wolf optimizer using gpu-accelerated load flow in active distribution networks," *IEEE Access*, vol. 9, 2021.

[9] A. Yazdanbakhsh *et al.*, "An evaluation of edge tpu accelerators for convolutional neural networks," *arXiv preprint arXiv:2102.10423*, 2021.

[10] T. Lu *et al.*, "Accelerating mri reconstruction on tpus," in *IEEE HPEC*, 2020, pp. 1–9.

[11] J. Sengupta *et al.*, "High-speed, real-time, spike-based object tracking and path prediction on google edge tpu." in *AICAS*, 2020, pp. 134–135.

[12] C. Y. Kaz Sato and D. Patterson, "An in-depth look at google's first tensor processing unit (tpu)," 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[14] A. Paszke, Gross *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NIPS*, 2019, pp. 8024–8035.

[15] H. Qassim, D. Feinzimer, and A. Verma, "Residual squeeze VGG19," *CoRR*, vol. abs/1705.03004, 2017.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[17] Z. Pan, J. Sheldon, C. Sudusinghe, S. Charles, and P. Mishra, "Hardware-assisted malware detection using machine learning," in *Design Automation and Test in Europe (DATE)*, 2021.

[18] N. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017, pp. 1–12.