Errant Beam Detection Using the AMD Versal ACAP and Vitis AI

Anthony M Cabrera[*†], Yigit A Yucesan[*], Frank Y Liu[*], Willem Blokland[*], Jeffrey S Vetter[*]
Oak Ridge National Laboratory[*], Washington University in St. Louis[†]
{cabreraam, yucesanya, blokland, liufy}@ornl.gov, vetter@computer.org

***Abstract*—The prevalence of ML and AI-powered solutions along with the slowing of Moore's Law has given rise to novel hardware platforms aimed at accelerating ML and AI. While programming these hardware platforms can be difficult, particularly for non-hardware experts, hardware vendors provide high-level tooling in an effort to address this difficulty. The Versal ACAP is an SoC designed by AMD that combines CPU cores, FPGA fabric, and a tiled, vector architecture called an AI engine all on the same socket. In an effort to more easily program this heterogeneous system, AMD has provided the Vitis AI development stack. In this work, we leverage Vitis AI to program a Versal ACAP to perform errant beam detection in the Spallation Neutron Source at Oak Ridge National Laboratory. Our initial work shows that after quantization and compilation of the model for the Versal ACAP, the classification accuracy, as measured by the AUC metric, is over 95% accurate while achieving this accuracy in 46 microseconds on average.**

***Index Terms*—Heterogeneous Computing, Edge Computing, Machine Learning, Artificial Intelligence, Versal ACAP, Vitis AI**

I. Introduction

The prevalence of machine learning and artificial intelligence is marked, in part, by the ubiquity of neural network (NN) models being leveraged to solve high impact problems in spaces such as object recognition [1] and natural language processing [2]. In concert with the end of Dennard's Scaling [3] and Moore's Law [4], this rise in these NN models has been accompanied by an increase in the amount of specialized hardware architectures and accelerators for deploying these models.

Historically, shrinking transistor process size has enabled better performance on traditional CPUs through additional compute capability being packed into the same square area. However, scaling feature sizes is becoming increasingly difficult as foundries are pushing the boundaries of just how small the process size can get. Thus, a need to rethink traditional architectures has given rise to a golden age of computer architecture [5].

While architectural innovation has enabled development of new microarchitectures and hardware accelerators for NN inference, harnessing the compute capability of these new hardware platforms is hampered by programming difficulty. This observation is especially apparent for domain scientists, e.g., physicists, chemists, who are experts and architect solutions for high impact problems in their respective fields, but face a programming barrier when realizing these solutions on leading-edge hardware systems. To this end, hardware vendors e.g., Groq, Cerebras, and GraphCore, often supply high-level Python or C++ APIs and tools that aim to ease the burden of programming these systems.

In this work, we explore a new hardware platform – the AMD Versal Adaptive Compute Acceleration Platform (ACAP) – and its respective tooling – Vitis AI – provided by AMD that aims to make the utilization of the Versal ACAP for ML and AI more amenable to non-hardware experts. The Versal ACAP is a system-on-chip (SoC) that combines hardened CPU cores, programmable logic, and a 2D-tiled architecture called an AI engine all on the same socket. The Versal ACAP has both PCIe card (VCK5000) and edge compute (VCK190) form factors, and Vitis AI can target both variants, as well as other AMD hardware backends. Vitis AI enables developers to take NN models trained in mainstream, high-level ML frameworks like PyTorch and TensorFlow and create a binary that can be executed on a desired AMD hardware platforms Common use-cases for Vitis AI

The common use of Vitis AI, as evidenced by provided examples in AMD's Vitis AI software release, is centered around image-based tasks. Our work, instead, focuses on first steps towards using the Versal ACAP to detect abnormal operating modes of the Spallation Neutron Source (SNS), located at Oak Ridge National Laboratory (ORNL), using a neural network trained on sensor data produced during the instrument's operation. To the best of our knowledge, our work is the first in the literature to leverage Vitis AI to target the VCK190 for a non-image based ML task. Specifically, our contributions are:

- Designing an NN model to predict errant beams at the ORNL SNS
- Deploying a non-image-based machine learning model for the AMD VCK190 platform using Vitis AI
- An end-to-end methodology for using Vitis AI to target a Versal ACAP platform
- Lessons learned using the Vitis AI and Versal ACAP-enabled platform

The rest of the paper is structured as follows. Section II does background. Section III does related work. Section IV talks about creating the high-level float model. Section V talks about creating the model for the VCK190. Section VI talks about board and host stuff. Section VII talks about experimental

Funding redatcted for anonymity.

setup. Section VIII talks about results. Sections IX and X talks about future work and conclusions, respectively.

II. Preliminaries

A. Vitis

Before we can understand the Vitis AI platform, we must first explain Vitis, which is the platform that Vitis AI is built on top of. Vitis is a unified software development platform that enables hardware and software development that targets a range of AMD platforms, ranging from edge, on-premise, and cloud deployments. Vitis encompasses all of the tooling necessary for traditional and high-level synthesis FPGA development, as well as tooling for the Versal ACAP. Additionally, Vitis contains the Xilinx runtime library, which contains the implementation for host and device communication. While Vitis allows developers to create bespoke hardware solutions, Vitis also provides a library[1] of IP solutions for a wide variety of domains, including high-performance computing, data analytics, and quantitative finance. These solutions are also composable with other designs in the library, or custom, user-defined IP.

B. Vitis AI

Vitis AI[2] is a platform that builds on top of Vitis through providing tooling that streamlines the deployment of deep learning models trained in high-level ML frameworks, e.g., PyTorch and TensorFlow. We will now breifly describe relevant pieces of this framework, i.e., the quantizer, compiler, and Deep-learning Processor Unit (DPU).

The ideal flow of Vitis AI is to take a trained, floating point model and use the Vitis AI toolchain to create the components necessary to execute inference on the desired AMD hardware target. As we describe in later sections, there is more developer effort that must take place in order to facilitate this flow. But the overall goal of the Vitis AI platform is to provide another layer of abstraction between the application developer and cutting-edge hardware in order to make deployment easier.

Two major components of the flow that enable model deployment onto a Vitis AI supported hardware target are the Vitis AI quantizer and compiler. The floating point model serves as input to the Vitis AI quantizer, and the output of the quantizer serves as input to the compiler.

The Vitis AI quantizer is responsible for quantizing the weights and activations of a `float`-precision model trained from a high-level ML framework into corresponding fixed-point representations, i.e., 8-bit integers. On first principles alone, using smaller, less complex types enables higher throughput and better power efficiency. However, these benefits come potentially at the expense of prediction accuracy.

The Vitis AI compiler takes the quantized model and compiles it for the desired hardware target, e.g., VCK190, VCK5000, or U250. The model gets compiled into an special DPU instruction set architecture (ISA) from AMD that was designed with ML and AI operations in mind. DPU microarchitectures and performance benefits vary based on what the hardware target is, e.g., the DPU implementation and latency profile for an edge-based Versal ACAP device is different than that of a data center-grade FPGA PCIe card. Though the instructions for the DPU ISA are proprietary, the documentation states that it it is designed at the tensor level, with the goal of facilitating the efficient implementation of deep learning networks.

We describe the use of the quantizer, compiler, and DPU in more detail in Section V.

C. Versal ACAP

The Versal ACAP is an SoC that combines both scalar and vector processing elements, as well as programmable logic, all on the same socket [6]. The scalar component refers to the hardened, dual-core ARM Cortex-A72 CPU. The programmable logic is refers to the reprogrammable FPGA fabric. Finally, the vector processing elements refer to the AI Engine, which is a 2D array of processor tiles that consist of a 2D array of processors called AI Engine tiles that are optimized for AI/ML Workloads. Each AI engine consists of a very long instruction word (VLIW), single instruction multiple data (SIMD) processor. All of these processing elements are interconnected using a network-on-chip (NoC). Because of constantly changing workloads, there is often not one flavor of processing element that can performantly execute all workloads. Thus, one of the benefits of the Versal ACAP is the flexibility in choosing which type of processing element is the right one for the task at hand. There is both a PCIe card and edge deployment variant of the Versal ACAP. In this work, we focus on the latter variant, known as the VCK190.

D. Spallation Neutron Source and Errant Beam Detection

The Spallation Neutron Source at ORNL is the world's most powerful proton accelerator to date. The accelerator provides high-intensity proton beam to a liquid mercury target for neutron production to be used for various experiments in research fields such as drugs [7], batteries [8], and engines [9]. In the design phase of the accelerator, the availability requirement is set to be at least 90%, and ultimately achieving 95% with continuous beam on target [10]. Unexpected beam failure, then, becomes very costly and disrupts the operation schedule. Unscheduled maintenance ultimately leads to loss of production and postponing the experiments that take months and in some cases years to prepare for. Another disadvantage of an errant beam is the potential to damage the expensive equipment located across the linear accelerator, which can further increase the downtime and additional maintenance costs.

Predicting impending failures only by utilizing data collected from diagnostic equipment already on board can help operators to avoid installing expensive sensors, unscheduled downtime and associated costs. To this end, we exploit the predictive power of ML to detect faulty beams at the SNS linear accelerator prior to failure. As part of our contribution,

[1] https://github.com/Xilinx/Vitis_Libraries
[2] https://github.com/Xilinx/Vitis-AI

we propose an ML model to detect abnormal operating modes trained on data from a pair of sensors located across the accelerator. While the model is trained to construct normal operation, we evaluate the predictive performance on known faulty beam pulses. We will discuss the creation of this model further in Section IV.

III. Related Work

The Cerebras Wafer Scale Engine 2 [11], GraphCore Colossus Mk2 Intelligence Processing Unit [12], and GroqChip Processor [13] are all examples of hardware vendors that have designed hardware for ML/AI workloads. Similar to the Versal ACAP, each of these solutions relies on a tiled approach to accelerating ML/AI workloads. Additionally, each vendor offers a path to take models trained a high-level ML framework and deploy them onto their respective hardware platform. There are two salient differences between these hardware platforms, and the Versal ACAP, though. The first is that none of these vendors offer an edge compute form factor for their respective offerings. Second, the Versal ACAP includes programmable logic on the same socket, creating a path for low-latency communication between heterogeneous processing elements.

To our knowledge, there is one other instance in the literature that uses Vitis AI to target the VCK190. Ibrahim et al. modify the YOLO object detection model for ship detection, and deploy this model on the VCK190 using Vitis AI [14]. However, our work uses Vitis AI to deploy a non-image-based ML model. Both works, though, contribute to the sparse literature on leveraging Vitis AI to target the VCK190.

IV. Errant Beam Detection ML Model

A. Model Architecture and Standard Operating Procedure

The information that we to train an ML model to detect faulty beam operation comes from the Beam Position Monitors (BPMs) located along the beamline data path in the proton accelerator in the SNS. BPMs are crucial pieces of diagnostic equipment located across the linear accelerator tunnel. One of the variables BPMs measure is the phase of the beam, which provide valuable information about acceleration process and potential anomalies.

We utilize BPM phase signals for anomaly prediction, by training a multi-layered perceptron to map phase signal from an upstream BPM (HEBT-BPM01) into a downstream BPM (HEBT-BPM32). In other words, we are using output data from the upstream sensor to predict the output data from the downstream sensor.

The architecture of the model is a multi-layer perceptron (MLP), and the layers are shown in Figure 1. The model is trained on data that represents normal beamline operation.

The trained model is then given both normal and abnormal phase signals – that represent normal and faulty operation, respectively – where the output given a normal phase signal should return an output that reflects what the downstream sensor would output if operating normally. However, the model is not trained on faulty data – only normal operating data.

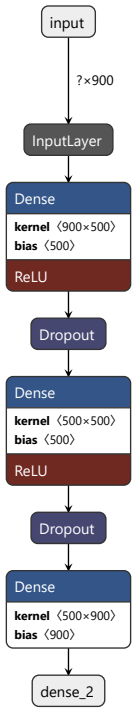

Fig. 1. Architectural overview of the layers in the MLP model trained for errant beam prediction.

When given an abnormal signal – a signal that occurs before an errant beam signal is measured – we distinguish that this is indicative of an incoming faulty state by calculating the root mean squared error (RMSE) between predicted and actual downstream BPM signal. Smaller RMSE values indicate that the model has correctly predicted the downstream signal. Larger RMSE values indicate the opposite. We use a simple threshold to determine when a caluclated RMSE value is large enough such that is in indicative of an abnormal signal.

B. Floating Point Model Training

We use TensorFlow2 as the high-level ML framework to create this model. To build the MLP model, we use the Keras Functional API (`tensorflow.keras.Functional`) to construct and connect the layers. Note that the Vitis AI user guide makes sure to specify that the Sequential API is not supported [15]. Additionally the user guide also mentions that the Vitis AI toolchain expects the model format to be `.h5` when using TensorFlow2. Model creation must be configured manually to specify this format because the ProtoBuf (`.pb`) format is what is generated by default. As mentioned in Section IV-B we use BPM phase signals as training data for the model.

V. Creating the Model for VCK190 Deployment

A. Using the Vitis AI Tools

Once the floating point model has been trained, the model can be ingested by the Vitis AI toolchain to create a binary that can program the VCK190. The Vitis AI toolchain is intended

to be used in a Docker container. The user can either create their own Docker container to target their local host machine, or source a pre-built Docker container from DockerHub[3]. In this work, we take the latter approach.

B. Quantization and Compilation

The way that Vitis AI invokes the quantizer depends on which ML framework (and version, if applicable) is being used. For the TensorFlow2 tools, the quantizer is invoked through an API call from a Python library – `quantizer.quantize_model`. It is worth noting here that, if using Microsoft VSCode as the development environment, the configuration file for the Docker container must set the remote user as `vitis-ai-user`. Otherwise, the user will not have the right permissions to step through the Python code in Vitis AI.

When invoking the quantizer, there are different parameters that can be toggled to configure the quantizer. Of note is the choice of quantization strategy – either a post training quantization (default setting) or quantization aware training approach – that may better performance. In our work, we found that the the default settings for each parameter provided satisfactory performance, but we leave the exploration of tuning these parameters to future work.

Once the model has been quantized, we verify that the effect of model quantization by calculating the area under the curve (AUC) for the quantized model and comparing it to the performance of the float-precision model. In order to perform inference using the quantized model, the user must import the

```
tensorflow_model_optimization.quantization.keras
```

module from Vitis AI. In our case, the AUC of the float and quantized model were 0.9099 and 0.9149, respectively. This indicated that the performance of the model through the quantization phase was performing with high accuracy.

Once the performance of the quantized model can be shown to be satisfactory, we now use the quantized model as input to the Vitis AI compiler. The compiler is invoked as a utility on the command line using the command `vai_c_tensorflow2`. One of the parameters of the compiler is `--arch`, which takes a `.json` file that contains the intended hardware target, and subsequently, the intended DPU, for execution. The result of compilation is a `.xmodel` file, which is a binary that contains all of the information needed to perform inference on the VCK190. As mentioned in Section II-B, the Vitis AI compiler creates a binary for the DPU ISA, but DPU microarchitectures depend on the hardware target. For the VCK190, the DPU microarchitecture is shown in Figure 2.

The DPU implementation for the VCK190 takes advantage of both the programmable logic and the AI engine on the Versal ACAP. The programmable logic side is primarily used to transfer data between the NoC and the AI engine. The AI engine is responsible for performing the heavy mathematical operations, such as convolutions.

[3]https://hub.docker.com/u/xilinx

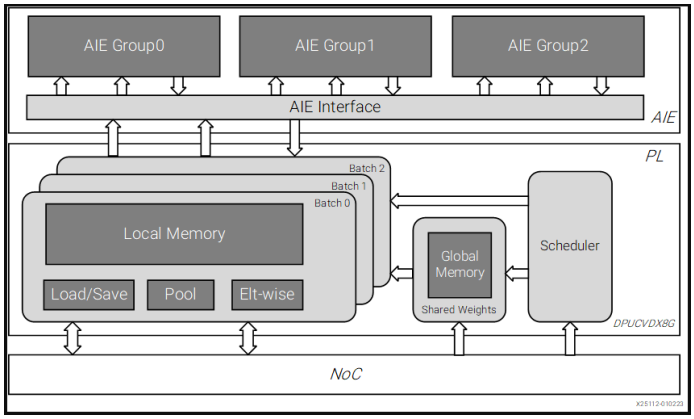

Fig. 2. Block diagram of the VCK190 DPU microarchitecture.

1) Understanding the Quantized and Compiled Model: Part of what the Vitis AI compiler does is partition the necessary computation of the model into nodes of computation. These nodes then get grouped together to form subgraphs. These subgraphs then get mapped to either the CPU or DPU for execution. To generate a graph of how the subgraphs have been partitioned is to use the following utility provided in the Vitis AI docker container to query the model for the structure:

```
xdputil xmodel <model_name> -l
```

where `model_name` is the output model from the Vitis AI compiler, and `-l` is the option for outputting the subgraph structure. A graphical depiction of this form can be obtained by replacing `-l` with `-p` to generate a PNG image of the computation graph and its subgraph.

In our case, running `xdputil` command shows that there are are 3 subgraphs that make up the entire model of computation. The first and third subgraphs are mapped to the CPU and the second subgraph is mapped to the DPU. While this utility shows us the flow of data between CPU and DPU, the user is still responsible for facilitating the communication between the two compute components. We will explain how to achieve this communication in Section VI.

VI. VCK190 and Host Side Setup

A. VCK190 Board Setup and Cross-Compilation

To help facilitate the programming of the VCK190, AMD provides a generic board image equipped with the Vitis AI Runtime at Vitis AI version 3.0[4]. This image must be flashed to a mini SD card and physically installed into the VCK190.

While AMD has provided a script[5] to enable cross-compilation for the ARM architecture-based CPU, we create a CMake Toolchain file that embeds the same information sourced in the script and allows us to use CMake to make building of the host binary easier.

In order to transfer files and serially send commands to the VCK190, it is connected to a host side system via Ethernet and

[4]Link to VCK190 board image
[5]Link to cross-compilation script

USB, respectively. Specifically, we use SSH for data transfer and `minicom` for serial communication with the VCK190.

B. Host Side Code

The host side code is written in C++ and compiled using the cross-compilation setup from Section VI-A. The host-side code's main components can be broken down into the following components: creating a runner to handle data transfer and model execution on the DPU, setting up the input and output buffers on the VCK190 side, pre-processing the raw data into the format expected by the model on the VCK190, batching the data appropriately for consumption by the model, then post-processing the data. In our case, post-processing is done in order to compare the predicted output with the expected output data.

1) Data Integration for DPU Input: From Section V-B, recall that the Vitis AI compiler has partitioned the model into three subgraphs: pre-processing of the input data, execution of the ML model, and post-processing of the output data. In this section we will focus on the pre- and post-processing of the input and output data.

There are two transitions between the CPU and DPU in the compute graph: one for transferring the input data to the VCK190 for consumption by the model, and one for transferring the prediction results back to the host-side CPU. In this case, the outgoing edge from the CPU to the DPU contains the shape information for what is expected by the DPU. Similarly, the outgoing edge from the DPU back to the CPU gives the shape of the data that the CPU should expect. This information is necessary when constructing the input and output buffers on both the host- and device sides. At runtime, we use the Vitis AI runtime (VART) API to query the compiled model in order to create CPU- and DPU-side buffers for incoming and outgoing data.

The pre- and post-processing of the input and output data that must be performed is the conversion between floating and fixed point precision. As noted in Section II-B, the quantizer transforms the weights and activations of the original `float`-precision model into corresponding fixed-point, 8-bit representations. The conversion of the inputs, though, must be authored in the host-side code because the DPU expects 8-bit inputs-bit inputs. Additionally, once the prediction is completed, the output data is still in its fixed-point, 8-bit representation. The scaling factors for conversions in both directions are embedded into the `.xmodel` file, and can be queried at runtime in order to scale the input to and the output of the DPU.

In order to transfer data and perform inference tasks, the VART API facilitates the submission and collection of jobs between the CPU and DPU. The VART API enables the setup of the runner initialized with the subgraph assigned to the DPU, the transfer of data to and from the DPU, as well as the asynchronous execution of the model on the DPU.

VII. Experimental Setup

At this point, the VCK190 board has been equipped with the image provided from AMD, the errant beam detection model has been quantized and compiled for the VCK190, and the host code has been authored such that it handles data transfers between the host and DPU as well as inference.

The host-side code drives execution. Once the host-side code has been cross-compiled, the resulting binary is transferred to the VCK190 along with the compiled model and input data. The executable run via a command line interface to the VCK190 enabled by `minicom`. The result of running the binary is the predicted output of the model. These predictions are then transferred off of the VCK190 to compare against the expected outputs. Inference is run a total of 10000 times to generate the amount of time elapsed when pre-processing the input data, performing inference, and then post-processing the output data. The results of these experiments are shown in the next section.

VIII. Results

Figure 3 shows the accuracy of the errant beam detection model – through measuring the area under the curve (AUC) – at three stages: when the model is initially trained using the TensorFlow, the model after quantization, and the model when its deployed onto the VCK190. Before exploring this result, it should be noted that the AUC result for the VCK190 needed to be inverted, i.e., the classification of errant beams was backwards. Further exploration into the tools is necessary to understand why this was necessary.

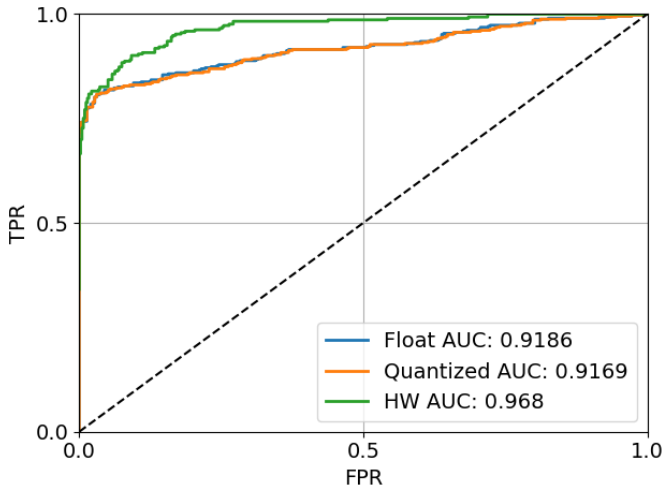

Fig. 3. AUC curves for the float, quantized, and compiled (hw) models.

From the graph, we see that the MLP model for errant beam detection resulted in an effective classifier. The accuracy of the classifier is over 95%. Furthermore, the performance on the VCK190 performed better than the float and quantized models, suggesting that quantization of the weights and activations made the model more accurate. This shows that, for this particular model, 8 bits of precision was enough to make the correct classifications.

In addition to the accuracy of the model deployed on the VCK190, the latency of the model was on the order of tens of

microseconds. The average time for pre-processing the input data, performing inference, and post-processing the data, averaged over 10000 inference instances, was 46 microseconds. In order be considered a viable option for integration into the proton accelerator, the classification must happen, ideally, within 100 microseconds [16].

IX. Future Work

As mentioned in Section V, there are parameters that are exposed during quantization and compilation that can be explored in order to gauge their effect on performance. This parameter tuning becomes a design space that can be explored in order to find the most performant inference solutions. Integrating the Versal ACAP into the beamline data path is another avenue for future work. Right now, the VCK190 ingests data that already exists on the VCK190's SD card. Using this initial exploration as a baseline, we can begin to explore more complex models, their design space, and their performance with respect to accuracy and latency.

X. Conclusion

In this work, we show an end-to-end example of using Vitis AI from training a model in a high-level ML framework to deployment on a Versal ACAP platform. The trained model was responsible for detecting abnormal operating modes for the beamline in the proton accelerator at the Spallation Neutron Source at Oak Ridge National Laboartory.

We employed the Vitis AI development stack to quantize the model to use 8-bit weights and activations, and then compiled that model for deployment onto the VCK190 variant of the Versal ACAP. Additionally, we disseminated lessons learned when using the Vitis AI toolchain, since, to date, there is only one other instance in the literature that leverages Vitis AI to target the VCK190. In particular, we describe what work needs to be done on the host side, i.e., pre- and post-processing of data and scheduling inference jobs and data transfers between the Versal ACAP's CPU and DPU. We showed that, even after quantization, the model deployed on the VCK190 was able to achieve over 95% accuracy with an average latency of 46 microseconds.

XI. Acknowledgements

This research used resources of the Experimental Computing Laboratory (ExCL) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725

This research was supported by the Defense Advanced Research Projects Agency Microsystems Technology Office Domain-Specific System-on-Chip Program.

The authors would like to thank Joseph Melber and Paul Hartke at AMD for helping to procure the VCK190 and connecting us with technical staff to troubleshoot issues.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] M. Bohr, "A 30 year retrospective on dennard's mosfet scaling paper," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007.

[4] R. R. Schaller, "Moore's law: Past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[5] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[6] AMD, "Versal: The first adaptive compute acceleration platform (acap)," Online, Sep 2020. [Online]. Available: https://docs.xilinx.com/v/u/en-US/wp505-versal-acap

[7] P. S. Langan, V. G. Vandavasi, C. J. Cooper, K. L. Weiss, S. L. Ginell, J. M. Parks, and L. Coates, "Substrate binding induces conformational changes in a class a -lactamase that prime it for catalysis," *ACS Catalysis*, vol. 8, no. 3, pp. 2428–2437, Feb. 2018. [Online]. Available: https://doi.org/10.1021/acscatal.7b04114

[8] H. Zhou, K. An, S. Allu, S. Pannala, J. Li, H. Z. Bilheux, S. K. Martha, and J. Nanda, "Probing multiscale transport and inhomogeneity in a lithium-ion pouch cell using in situ neutron methods," *ACS Energy Letters*, vol. 1, no. 5, pp. 981–986, Oct. 2016. [Online]. Available: https://doi.org/10.1021/acsenergylett.6b00353

[9] M. L. Wissink, Y. Chen, M. J. Frost, S. J. Curran, O. Rios, Z. C. Sims, D. Weiss, E. T. Stromme, and K. An, "Operando measurement of lattice strain in internal combustion engine components by neutron diffraction," *Proceedings of the National Academy of Sciences*, vol. 117, no. 52, pp. 33 061–33 071, Dec. 2020. [Online]. Available: https://doi.org/10.1073/pnas.2012960117

[10] S. Henderson *et al.*, "The spallation neutron source accelerator system design," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 763, pp. 610–673, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900214003817

[11] G. Lauterbach, "The path to successful wafer-scale integration: The cerebras story," *IEEE Micro*, vol. 41, no. 6, pp. 52–57, 2021.

[12] S. Knowles, "Graphcore," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–25.

[13] D. Abts, J. Kim, G. Kimmell, M. Boyd, K. Kang, S. Parmar, A. Ling, A. Bitar, I. Ahmed, and J. Ross, "The groq software-defined scale-out tensor streaming multiprocessor : From chips-to-systems architectural overview," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. Los Alamitos, CA, USA: IEEE Computer Society, aug 2022, pp. 1–69. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HCS55958.2022.9895630

[14] Y. Ibrahim, L. Chen, and T. Haonan, "Deep learning-based ship detection on fpgas," in *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2022, pp. 454–459.

[15] AMD, "Vitis ai overview," Online, Jun 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1414-vitis-ai

[16] N. R. Miniskar, A. Young, F. Liu, W. Blokland, A. Cabrera, and J. S. Vetter, "Ultra low latency machine learning for scientific edge applications," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2022, pp. 01–07.