Check for updates

OPEN ACCESS

EDITED BY James A. Ang, Pacific Northwest National Laboratory (DOE), United States

REVIEWED BY Peng Chen,

RIKEN, Japan Tekin Bicer, Argonne National Laboratory (DOE), United States Yudong Yao, ShanghaiTech University, China

*CORRESPONDENCE Abhilasha Dave 🖂 adave@slac.stanford.edu

RECEIVED 30 October 2024 ACCEPTED 19 May 2025 PUBLISHED 18 June 2025

CITATION

Dave A, Wang C, Russell J, Herbst R and Thayer J (2025) FPGA-accelerated SpeckleNN with SNL for real-time X-ray single-particle imaging.

Front. High Perform. Comput. 3:1520151. doi: 10.3389/fhpcp.2025.1520151

COPYRIGHT

© 2025 Dave, Wang, Russell, Herbst and Thayer. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

FPGA-accelerated SpeckleNN with SNL for real-time X-ray single-particle imaging

Abhilasha Dave*, Cong Wang, James Russell, Ryan Herbst and Jana Thayer

SLAC National Accelerator Laboratory, Menlo Park, CA, United States

We present the implementation of a specialized version of our previously published unified embedding model, SpeckleNN, for real-time speckle pattern classification in X-ray Single-Particle Imaging (SPI), using the SLAC Neural Network Library (SNL) on an FPGA platform. This hardware realization transitions SpeckleNN from a prototypic model into a practical edge solution, optimized for running inference near the detector in high-throughput X-ray free-electron laser (XFEL) facilities, such as those found at the Linac Coherent Light Source (LCLS). To address the resource constraints inherent in FPGAs, we developed a more specialized version of SpeckleNN. The original model, which was designed for broader classification across multiple biological samples, comprised \sim 5.6 million parameters. The new implementation, while reducing the parameter count to 64.6K (a 98.8% reduction), focuses on maintaining the model's essential functionality for real-time operation, achieving an accuracy of 90%. Furthermore, we compressed the latent space from 128 to 50 dimensions. This implementation was demonstrated on the KCU1500 FPGA board, utilizing 71% of available DSPs, 75% of LUTs, and 48% of FFs, with an average power consumption of 9.4W according to the Vivado post-implementation report. The FPGA performed inference on a single image with a latency of 45.015 microseconds at a 200 MHz clock rate. In comparison, running the same inference on an NVIDIA A100 GPU resulted in an average power consumption of \sim 73W and an image processing latency of around 400 microseconds. Our FPGA-accelerated version of SpeckleNN demonstrated significant improvements, achieving an 8.9× speedup and a 7.8× reduction in power consumption compared to the GPU implementation. Key advancements include model specialization and dynamic weight loading through SNL, which eliminates the need for time-consuming FPGA design re-synthesis, allowing fast and continuous deployment of models (re)trained online. These innovations enable real-time adaptive classification and efficient vetoing of speckle patterns, making SpeckleNN more suited for deployment in XFEL facilities. This implementation has the potential to significantly accelerate SPI experiments and enhance adaptability to evolving experimental conditions.

KEYWORDS

FPGA, machine learning, XFEL, SPI, SpeckleNN, GPU, LCLS

1 Introduction

The rapid advancement of next-generation detectors in scientific research and industrial applications has catalyzed an exponential increase in data generation rates. Ultra-high-rate (UHR) detectors, such as those used in X-ray free-electron laser (XFEL) facilities like Linac Coherent Light Source (LCLS), now operate at frequencies exceeding 100 kHz, producing data throughputs that can surpass 1 TB/s. These cutting-edge technologies have revolutionized the study of nanoscale particles, particularly through X-ray Single-Particle Imaging (SPI). In these experiments, intense femtosecond X-ray pulses generate intricate scattering patterns, commonly referred to as "speckles," from individual particles. These speckles are critical for reconstructing the three-dimensional structures of non-crystalline particles at room temperature. However, the high data rates at these facilities present significant challenges for real-time classification of speckle patterns, which is essential for accurate and rapid identification of single hits required for subsequent three-dimensional reconstructions.

Traditional approaches to speckle pattern classification have relied heavily on either unsupervised learning techniques, which often necessitate post-experimental human intervention, or supervised learning models that require extensive labeled datasets. These datasets are time-consuming to generate and are often impractical to obtain at the scale required for real-time applications in high-throughput environments. Consequently, both approaches are suboptimal for the rapid, on-the-fly analysis needed in XFEL facilities, where computational complexity and latency can significantly hinder experimental progress.

To address these challenges, we previously introduced SpeckleNN (Wang et al., 2023), a unified embedding model specifically designed for the real-time classification of speckle patterns with limited labeled examples. SpeckleNN employs a contrastive learning approach using twin neural networks (Bromley et al., 1993; Chopra et al., 2005), which map speckle patterns to a unified embedding vector space. Within this space, classification is performed based on the Euclidean distance between embedding vectors, enabling robust few-shot classification even in scenarios with sparse labeling or missing detector areas. This model is particularly suited for SPI experiments, where rapid and accurate classification is critical to the success of data collection and subsequent analysis.

Despite its effectiveness, maintaining ML inference at high data rate presented significant challenges. While GPUs are powerful tools for parallel processing, their performance in real-time, edge deployments can be influenced by various factors, such as data transfer overhead, memory constraints, and the need for batch processing to optimize throughput. This is particularly problematic when data must be continuously streamed from sensors or detectors, and immediate feedback is required.

To overcome the challenge of real-time inference in highthroughput environments, there is a growing interest in computing platforms that can provide low-latency processing at the edge, in close proximity to the instrumentation. Field-Programmable Gate Arrays (FPGAs) have emerged as a promising solution, leveraging their advanced parallelism and configurability. FPGAs can be optimized for processing individual data points with minimal latency, making them particularly suitable for applications that require immediate feedback and real-time analysis. This capability is particularly advantageous in high data rate XFEL facilities.

The primary motivation of this work is to enable low-latency, edge-based inference at the point of data acquisition in highthroughput environments such as LCLS. Real-time identification of critical patterns (e.g., single-particle hits) is essential for optimizing data collection, reducing storage demands, and supporting online decision-making. While demonstrated here for X-ray SPI, the approach is equally relevant to other data-intensive scientific domains, including high-energy physics and electron microscopy, where similar constraints—bandwidth, latency, and compute limitations—necessitate adaptive, FPGA-based inference solutions.

However, the promise of FPGAs comes with its own set of challenges, particularly related to its resource constraints. These devices, while powerful, are typically smaller and offer limited computational resources compared to platforms like GPUs. This presents a significant hurdle when deploying neural network models, which often contain millions of parameters. Our research confronted this challenge with the original SpeckleNN model, which comprised nearly 5.5 million parameters. Fitting such a large model into single FPGA without exceeding its capacity was not feasible.

To address this, we undertook a rigorous optimization process that successfully reduced the model size by \sim 98.8%. However, this reduction came at a cost to accuracy. While the original model achieved around 98% accuracy, the newer, lighter version has an accuracy of 90%. Trading off model size with accuracy in this case is appropriate, enabling the SpeckleNN model to run efficiently on edge devices like FPGAs, thus facilitating real-time data processing directly at the site of data collection. Despite the challenges posed by the reduction in model size, the benefits of deploying these optimized models on FPGAs are profound. By enabling inference on a per-image basis with substantially reduced latency, FPGAs eliminate the bottlenecks associated with batch size constraints in GPUs, allowing for continuous, real-time data analysis without the burdensome overhead of data transfer and storage.

To deploy this optimized model on FPGAs, we leveraged the SLAC Neural Network Library (SNL) (Herbst et al., 2022), a powerful tool designed specifically to address the challenges of high-throughput, low-latency environments. SNL enables the seamless translation of machine learning architectures into FPGAcompatible code, allowing for the construction of data processing pipelines characterized by ultra-low latency and high throughput. This capability is particularly crucial for managing the immense data velocities generated by modern detectors.

One of the most significant advantages of SNL is its ability to dynamically load weights and biases onto the FPGA. This feature eliminates the need for resynthesizing the entire neural network code for FPGA when the model is retrained, enabling immediate inference runs with updated models. This feature greatly enhances the flexibility and efficiency of using FPGAs in environments where models may need to be frequently updated or fine-tuned to adapt to new data or experimental conditions. As a result, deploying machine learning models on FPGAs via SNL represents a significant advancement, offering a robust solution for real-time data processing across a wide range of scientific domains.

This paper details the implementation of a streamlined version of SpeckleNN on the KCU1500 FPGA board using Xilinx Vitis (Kathail, 2020) and the SLAC Neural Network Library (SNL) (Xilinx, Inc., 2024). This approach demonstrates the board's capability to meet the stringent demands of high-throughput, lowlatency environments. Key advancements include extensive model pruning and the novel application of dynamic weight loading

10.3389/fhpcp.2025.1520151

through SNL, eliminating the need for FPGA re-synthesis and enabling online retraining for continuous model improvement. These innovations enable real-time adaptive classification and efficient vetoing of speckle patterns, optimizing SpeckleNN for deployment in XFEL facilities. Consequently, this implementation accelerates SPI experiments and enhances system adaptability to evolving experimental conditions.

The primary objective of this study is to demonstrate the feasibility and effectiveness of deploying a highly optimized version of the SpeckleNN model on Field-Programmable Gate Arrays (FPGAs) for real-time speckle pattern classification in X-ray single-particle imaging (SPI) at X-ray free-electron laser (XFEL) facilities. Specifically, the study aims to:

- 1. **Optimize the SpeckleNN model:** achieve a significant reduction in the model size, from 5.6 million parameters to 64.6K parameters (98.8% reduction), while maintaining high classification accuracy (90%), making it suitable for deployment on resource-constrained FPGA devices.
- 2. **Implement and evaluate on FPGA:** implement the optimized SpeckleNN model on the KCU1500 FPGA board using the SLAC Neural Network Library (SNL) and evaluate its performance in terms of resource utilization (DSPs, LUTs, FFs), power consumption, and inference latency.
- 3. **Compare FPGA performance to GPU:** compare the FPGAaccelerated SpeckleNN's performance with that of a GPU (NVIDIA A100), focusing on improvements in speed (latency reduction) and power efficiency.
- 4. **Consider arithmetic precision:** conduct a detailed consideration of arithmetic precision for each layer of the FPGA-based implementation, comparing the outcomes of FPGA-based C-Simulation (CSIM) with those from PyTorch-based simulations. This ensures that the precision and accuracy of the model are maintained across different computational environments, which is crucial for reliable real-time inference.

2 Methodology

The methodology for this study is structured around four key objectives mentioned in the introduction section earlier, each aimed at ensuring the successful deployment of the SpeckleNN model on FPGA hardware while maintaining performance, efficiency, and accuracy.

2.1 SpeckleNN model architecture and optimization

To optimize our neural network for deployment on resourceconstrained devices such as FPGAs, we systematically reduced the model size while rigorously evaluating its impact on classification accuracy. The architecture of both the original and optimized models used in our experiments is illustrated in Figure 1.

• Original model architecture: the original SpeckleNN model (Figure 1a) featured two convolutional layers followed by

two dense layers, which, while delivering high accuracy, also resulted in a computationally intensive design. Specifically, the model's architecture included 32 filters of size 5×5 in the first convolutional layer, paired with a 2×2 max-pooling layer. The second convolutional layer expanded to 64 filters, again followed by a max-pooling layer. These convolutional layers fed into a dense layer with 512 units, culminating in a final dense layer of 128 units. This design achieved a classification accuracy of 98%, but the model's ~5.6 million parameters posed significant challenges for deployment on FPGAs due to their constrained computational resources.

- Optimized model architecture: recognizing the necessity for a more resource-efficient design, we optimized the SpeckleNN architecture with a focus on minimizing the parameter count while maintaining high accuracy. The optimized model (Figure 1b) incorporated several modifications: the first convolutional layer was reduced to 7 filters of size 5×5 , followed by a 2 \times 2 max-pooling layer, and the second convolutional layer was minimized to 3 filters, again with max-pooling. The dense layers were similarly scaled down to 100 and 50 units, respectively. Through this process, we achieved a dramatic reduction of 98.8% in the number of parameters-from 5.6 million to ~64.6K-while maintaining a robust classification accuracy of 90%. This demonstrates that the original larger model, with its 5.6 million parameters, had redundancy that wasn't necessary for effective learning. By using fewer parameters, the optimized model has become more efficient, with each parameter contributing effectively to the network's final accuracy. This compact architecture ensures that the model not only retains strong predictive performance but also operates more efficiently, making it ideal for deployment on resource-constrained FPGAs where rapid, real-time processing and resource efficiency are paramount.
- 32-bit inference implementation: we implemented the inference run using 32-bit floating-point precision to strike a balance between computational efficiency and maintaining high accuracy. Although lower-bit precision could further reduce resource usage and power consumption, it often introduces quantization errors that can degrade accuracy, especially in deep neural networks (PyTorch Contributors, 2024; Dave et al., 2023). Given the dramatic reduction in model size, where we achieved 90% accuracy, our priority was to preserve this accuracy rather than sacrifice it by aggressively quantizing the parameters (Hashemi et al., 2017). Since the 32-bit version fit successfully on the KCU1500 FPGA board, we opted to keep this precision for now. However, in future studies, we plan to explore the potential of parameter quantization to further optimize the model's performance without compromising accuracy.
- **Trade-off analysis:** the optimization process inevitably involved a trade-off between model size and accuracy. The reduction in parameters led to a slight decrease in accuracy from 98% to 90%. However, this trade-off is often justified in scenarios where deployment constraints, such as limited computational resources or the necessity for real-time processing, are paramount. The significant reduction in model size not only accelerates inference times but also



substantially reduces the computational load and memory footprint, making the optimized model more suitable for highthroughput environments like those encountered in XFEL facilities.

In summary, the optimized SpeckleNN model effectively balances accuracy, efficiency, and precision. By reducing the parameter count, we enhanced the effective use of each parameter toward achieving high accuracy, and by implementing a 32-bit inference run, we ensured that the model operates efficiently while maintaining its performance integrity. These optimizations make the model an ideal candidate for deployment in edge computing environments, where rapid and efficient data processing is essential.

2.2 Implementation and Evaluation on FPGA using SNL

After optimizing the SpeckleNN model, as shown in Figure 1b, we moved forward with its implementation on the FPGA platform using the SLAC Neural Network Library (SNL). As illustrated in Figure 2, we began by defining the architecture of the SpeckleNN neural network using

PyTorch (Paszke et al., 2016), a widely-used machine learning framework. The PyTorch model, which provides a highlevel and flexible environment for neural network design, was translated into a C++ parameter template within the Xilinx Vitis platform (Kathail, 2020; AMD, 2024) to facilitate its deployment using the SLAC Neural Network Library (SNL). SNL, designed specifically for FPGA-based machine learning implementations, operates using a C++ template model where weights, and biases are represented as memory-mapped interfaces. This allows for the precise mapping of neural network parameters to FPGA hardware, ensuring that the network structure is preserved while optimizing for performance in resource-constrained environments.

In SNL, the memory-based interface manages the storage and retrieval of weights and biases, allowing the FPGA to access these parameters efficiently during inference. The communication between different layers of the neural network is handled through a streaming interface, where data flows sequentially from one layer to the next without the need for intermediate storage. This streaming interface is particularly advantageous for real-time processing as it minimizes latency and maximizes throughput, both of which are critical for high-performance edge applications like those found in XFEL facilities.



The use of a C++ parameter template in SNL also supports a modular and re-configurable design, which is crucial when implementing complex neural networks such as SpeckleNN on hardware. By directly mapping the PyTorch model into this C++ structure, we ensured that the translation from software to hardware was seamless, preserving the integrity of the model's architecture while making it compatible with FPGA constraints.

Furthermore, SNL's design allows for a clear separation between the computational elements (such as matrix multiplications and convolutions) and the memory management for storing weights and biases. This separation ensures that the FPGA's computational resources are used efficiently, with minimal contention for memory access during the inference process. The streaming interface between layers further reduces bottlenecks, allowing data to flow continuously through the pipeline, thus achieving low-latency performance. This structure enables real-time inference on large datasets without sacrificing accuracy or speed, making it ideal for high-throughput, low-latency environments.

Overall, by leveraging SNL's C++ parameter template and streaming architecture, we successfully implemented the optimized SpeckleNN model on FPGA hardware. This approach allowed us to maintain the integrity of the PyTorch-based design with a reasonable 90% accuracy, while ensuring that the model was adapted to meet the performance demands of FPGA deployment, specifically in scientific applications requiring real-time SPI data processing in XFEL facilities.

2.2.1 Evaluation of each layer type Outcomes Between SNL-Csim and PyTorch

In this section, we conducted a comprehensive evaluation by comparing the output of various layers in the SpeckleNN model across two platforms: SNL's C-simulation (Csim) and PyTorch. The comparison includes a single layer of each type—convolution, ReLU activation, MaxPool, and the final two dense layers to examine the behavior of these layers in both frameworks. While both Csim and PyTorch are designed for deep learning model implementation, they can differ in how they handle numerical precision and arithmetic operations, potentially leading to slight discrepancies in results. These differences primarily stem from variations in floating-point arithmetic, internal optimizations, and the way each framework manages edge cases such as overflow, underflow, and rounding.

Machine learning frameworks like PyTorch and Keras (Northcutt, 2019; Viso, 2024; PyTorch Forum, 2017; Oudad, 2021) are known to produce slightly different results even when performing the same operations. These differences stem from how they manage floating-point precision and internal optimizations (Johnson, 2018; Galal et al., 2013; Pietrołaj and Blok, 2022). While such discrepancies may seem minor in the early layers of a neural network, they can accumulate and propagate through subsequent layers (Dave et al., 2023), potentially affecting the final model output. Therefore, a layer-by-layer comparison is crucial to understand how these numerical differences affect the SpeckleNN model's behavior when deployed on an FPGA using SNL compared to the PyTorch reference.



2.2.1.1 Layer 0 convolutional feature maps analysis (Pre-ReLU activation)

As shown in Figure 3 we compared the outputs from Csim and PyTorch for Layer 0 across seven channels, immediately following the convolution operation. The key observations were:

- High consistency: both frameworks produced nearly identical feature maps across all channels. Minor differences were observed, most notably in Channel 3, where the difference between the two frameworks was slightly larger but still within a minimal range (around -0.0004 to +0.0004).
- Minor numerical differences: the pixel wise differences were very small, mostly due to floating-point precision differences, with the majority of the differences falling within the range of -0.0005 to 0.0005, indicating that the two frameworks are well-aligned in their convolution computation.

2.2.1.2 Layer 0 ReLU activation applied to the convolutional feature maps

Figure 4 presents the comparison of the resulting feature maps from both frameworks after applying the ReLU activation function to the convolutional outputs. The key findings are as follows:

- Sparse activations: as expected, ReLU zeroed out all negative values in the feature maps, resulting in sparse activations, where only positive values remained.
- Consistency maintained: the ReLU-activated feature maps remained highly consistent between Csim and PyTorch across all channels. The visual similarity between the frameworks was nearly identical, reflecting the same activation patterns.
- Channel 3 differences: while differences remained minimal, Channel 3 again showed slightly larger deviations compared

to the other channels. These differences, in the range of -4×10^{-5} to 4×10^{-5} , were more noticeable than in other channels but still very small and insignificant in terms of model performance.

2.2.1.3 Connecting the two observations for layer 0 Pre-ReLU and Post-ReLU

- ReLU's impact on differences: the differences we observed in the convolutional outputs (particularly in Channel 3) persisted post-ReLU, though they remained very small. ReLU did not amplify these differences, which suggests that the inconsistencies seen in Channel 3 are inherent to the convolutional computation itself, rather than being introduced by the ReLU activation.
- Consistency across layers: the high consistency between Csim and PyTorch in both the convolutional output and post-ReLU activation maps indicates that both frameworks handle the convolution and activation operations similarly, with only minor numerical variations. These variations are within an acceptable range and do not suggest any significant divergence in the implementation of these operations.
- The convolutional outputs from Csim and PyTorch are highly consistent, with only minor numerical differences that carry through after the application of ReLU. These small differences, particularly in Channel 3, suggest that some channels may be more sensitive to precision, but these deviations are unlikely to affect the overall model performance.

2.2.1.4 Analysis and connection between convolutional and dense layer outputs

• In our previous analysis of layer 0, we first observed the convolutional output feature maps, which revealed that the



FIGURE 4

Convolutional Layer 0 outcome after passing through ReLU activation for all 7 output featuremap of SNL Csim, PyTotch, and difference between SNL Csim and PyTorch.



major activity or significant patterns were concentrated in the central regions of the feature maps across multiple channels. This indicates that the convolutional layers are capturing prominent features, particularly in the middle of the input space, where activations are more pronounced. • When these activations are propagated through the network, they have a direct influence on the dense layer outputs, as observed in the subsequent analysis. As shown in Figure 5 the comparison of dense layer (Layer 4) before ReLU activation outputs from SNL Csim and PyTorch shows that both



frameworks exhibit similar neuron activation patterns, with the most significant spikes in neuron values occurring around neurons 60 to 70. These spikes can be attributed to the convolutional layer's feature extraction process, where the highly active regions in the middle of the feature maps are likely contributing to stronger connections to specific neurons in the dense layer. The dense layer is designed to aggregate the most relevant information from the convolutional layers, and the prominent spikes observed in neurons 60 to 70 are a direct reflection of the amplified signals derived from the high-activation areas in the convolutional outputs.

- Further comparison between SNL Csim and PyTorch reveals high consistency between the two frameworks, as shown in Figure 5 by the overlapping plots in the pre-activation outputs. However, the small numerical differences, particularly the localized spikes in differences around neurons 10, 30, and 60 shown in the difference plot Figure 6, likely coming from floating-point precision variations or minor differences in internal precision handling between the two frameworks. These discrepancies, while noticeable in certain neurons, remain relatively small compared to the overall magnitude of the neuron outcomes, which range from $\sim -20,000$ to +20,000.
- The analysis demonstrates that the dense layer's significant spikes in neuron activations are a direct consequence of the prominent features extracted by the convolutional layers, particularly from the middle part of the feature maps. These high-activation regions are captured and amplified by the dense layer neurons, leading to larger spikes in output. Despite small numerical differences between SNL Csim and PyTorch, both frameworks perform consistently, and the minor deviations observed are not expected to significantly impact

overall model performance. This consistency reinforces the robustness of both frameworks in implementing deep learning models with comparable behavior.

2.2.1.5 Implications and key takeaways

- 1. High framework consistency: the comparison between SNL Csim and PyTorch reveals strong consistency, with only minor numerical differences. These differences, observed around specific neurons (10, 30, and 60), are likely due to floating-point precision variations and frameworkspecific optimizations. However, these discrepancies are minimal and do not significantly affect the overall model behavior, ensuring reliability in implementation across both platforms.
- 2. Convolutional features driving dense layer activations: the spikes observed in the dense layer outputs, particularly around neurons 60 to 70, align with the high-activation regions in the convolutional feature maps. This shows that the convolutional layers effectively extract important features, which are then amplified by the dense layers. This insight reinforces the importance of optimizing the feature extraction process in convolutional layers to improve downstream activations in dense layers.
- 3. Framework interoperability and model robustness: the high degree of similarity between the two frameworks demonstrates their interoperability, allowing models to be trained in one and deployed in the other without significant performance loss. This flexibility is crucial for model development and deployment across different platforms. Furthermore, the model's robustness to small numerical variations indicates that it can produce stable results across frameworks, enhancing its applicability in real-world scenarios.



- 4. Handling arithmetic discrepancies across platforms: despite these differences, the error introduced by the arithmetic handling between SNL-Csim and PyTorch remains within acceptable limits, allowing the model to maintain comparable accuracy between FPGA-based inference and PyTorch simulations. For the SpeckleNN model, the accumulated error is still small enough that it does not significantly impact final accuracy, as demonstrated by the near-identical results between the two platforms. However, this evaluation raises an important question: as models transition between different frameworks and hardware platforms, how can we ensure consistent handling of arithmetic operations? Small discrepancies, especially in deeper networks, could accumulate and affect the reliability of models across platforms. Addressing these concerns is critical for ensuring robustness and consistency when moving from software environments like PyTorch to hardware-based implementations, such as FPGAs via SNL-Csim.
- 5. Need for further exploration of numerical precision: this analysis highlights the need for further exploration into how numerical precision and arithmetic operations are handled across platforms. In edge applications, where minor discrepancies could have significant impact on model performance, ensuring consistency becomes even more crucial. Future research should focus on addressing these precisionrelated issues to improve the reliability and robustness of machine learning models when transitioning between software and hardware platforms.

2.3 Resource Utilization and Performance Analysis of SpeckleNN on KCU1500 FPGA

2.3.1 Resource utilization and performance evaluation on the KCU1500 FPGA board

After successfully implementing the SpeckleNN network on the KCU1500 FPGA board (Xilinx, Inc., 2024), we conducted a detailed evaluation of the resource utilization and inference performance. The results, as depicted in Figure 7, demonstrate the efficient use of FPGA resources. Specifically, the implementation utilized 75% of the available Look-Up Tables (LUTs), 48% of Flip-Flops (FFs), 25% of Block RAM (BRAM), and 71% of Digital Signal Processors (DSPs). This balanced utilization indicates that the network design was well-optimized to fit within the resource constraints of the KCU1500 board.

For the inference run of the neural network on the FPGA, we employed the Integrated Logic Analyzer (ILA) for debugging and performance monitoring. The ILA captured the complete frame of the inference process, tracking the time from the arrival of the first pixel to the generation of the final 50th dimensional latent space output. The total time for this operation was ~9,003 FPGA clock cycles. Given the clock cycle rate of 5 nanoseconds, the total latency for the inference run was measured at ~45.05 microseconds (μ s).

This low-latency performance is critical for real-time applications in high-throughput environments, such as those found in XFEL facilities. The ability to process incoming data with such minimal delay ensures that the system can keep pace with the

TABLE 1 Resource Utilization Analysis: SNL and HLS4ML for FPGA Inference Runs.

Inference run framework	Latency/image (us)	Power (W)
SNL-based FPGA SpackleNN	45.05 (5ns clock period)	9.4
GPU A100	400	73

rapid data generation rates typically associated with these scientific experiments.

2.3.2 Comparative analysis with GPU

As highlighted in Table 1, the FPGA-based implementation of SpeckleNN provided significant improvements in both speed and power efficiency compared to the GPU-based inference run on the NVIDIA A100. Specifically, the FPGA achieved a remarkable 8.9x improvement in inference speed over the GPU, thanks to its lower latency and more efficient parallelism in handling the data. Additionally, the FPGA consumed $7.8 \times$ less power than the GPU, underscoring its suitability for edge computing environments where energy efficiency is critical.

These improvements demonstrate the effectiveness of deploying the optimized SpeckleNN model on FPGA hardware, not only in terms of computational speed but also in significantly reducing the power overhead, which is a key consideration for continuous, real-time processing in high-data-rate applications.

3 Conclusion

In this study, we successfully optimized and implemented the SpeckleNN neural network on the KCU1500 FPGA platform using the SLAC Neural Network Library (SNL) to meet the high-throughput, low-latency demands of real-time X-ray singleparticle imaging (SPI) at X-ray free-electron laser (XFEL) facilities. The original SpeckleNN model, which initially consisted of 5.6 million parameters, was reduced by \sim 98.8% to 64.6K parameters. This reduction, achieved through architectural modifications and parameter compression, allowed the model to fit efficiently on FPGA hardware while maintaining an impressive 90% classification accuracy.

We chose to implement the network using 32-bit floating-point precision to balance computational efficiency and accuracy. While lower-bit precision could potentially reduce resource consumption further, the 32-bit precision provided a level of reliability and compatibility with industry-standard frameworks such as PyTorch. This decision also ensured that the FPGA implementation closely matched the performance and behavior of the original PyTorch model, minimizing discrepancies due to arithmetic handling differences.

Our evaluation of resource utilization showed that the SpeckleNN model efficiently utilized 75% of LUTs, 48% of Flip-Flops, 25% of BRAM, and 71% of DSPs on the KCU1500 board,

confirming that the design was well-optimized for the FPGA's resource constraints. The inference latency was measured at 45.05 microseconds, with a total of 9,003 clock cycles, meeting the real-time processing requirements essential for high-throughput environments.

A layer-by-layer comparison of the SNL C-simulation (Csim) results with PyTorch revealed small but manageable numerical discrepancies between the two platforms. While the error propagated through the deeper layers of the network, the difference remained within acceptable limits, ensuring that the FPGA's performance closely mirrored the results obtained from the PyTorch simulations.

Finally, the comparative analysis between FPGA and GPU performance demonstrated significant advantages for FPGA-based deployment. The FPGA achieved a $8.9 \times$ speed improvement and consumed $7.8 \times$ less power compared to the NVIDIA A100 GPU, making it a highly efficient alternative for real-time, edge-based machine learning tasks.

In conclusion, our implementation of SpeckleNN on the KCU1500 FPGA, supported by SNL, proved to be a powerful and efficient solution for real-time speckle pattern classification in XFEL facilities. While we achieved substantial speed and power improvements, future work will explore further optimization techniques such as quantization to reduce resource usage while maintaining accuracy. This study underscores the potential of FPGA-based machine learning models for high-performance scientific applications, offering a compelling path forward for the deployment of neural networks in resource-constrained, real-time environments.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: https://www.cxidb.org/id-156.html.

Author contributions

AD: Conceptualization, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing – original draft. CW: Data curation, Formal analysis, Software, Writing – review & editing. JR: Formal analysis, Writing – review & editing. JT: Funding acquisition, Supervision, Writing – review & editing. RH: Resources, Supervision, Writing – review & editing, Funding acquisition.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences under Award Number FWP-100643. Use of the Linac Coherent Light Source (LCLS), SLAC National Accelerator Laboratory, is supported by the US

DOE, Office of Science, Office of Basic Energy Sciences (contract No. DE-AC02-76SF00515).

Acknowledgments

We acknowledge the assistance of the large language model ChatGPT by OpenAI in refining the language and enhancing the readability of this paper.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative Al statement

The author(s) declare that Gen AI was used in the creation of this manuscript. We acknowledge the assistance of the large language model ChatGPT by OpenAI in refining the language and enhancing the readability of this paper.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

AMD (2024). Vitis High-Level Synthesis (HLS) Tool Documentation: Templates. Available online at: https://docs.amd.com/r/en-US/ug1399-vitis-hls (accessed October 28, 2024).

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a "siamese" time delay neural network. *Adv. Neural Inf. Process. Syst.* 6:25. doi: 10.1142/S0218001493000339

Chopra, S., Hadsell, R., and LeCun, Y. (2005). "Learning a similarity metric discriminatively, with application to face verification," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (Los Alamitos, CA: IEEE Computer Society), 539–546. doi: 10.1109/CVPR.2005.202

Dave, A., Frustaci, F., Spagnolo, F., Yayla, M., Chen, J.-J., and Amrouch, H. (2023). Hw/sw codesign for approximation-aware binary neural networks. *IEEE J. Emerg. Sel. Top. Circuits Syst.* 13, 33-47. doi: 10.1109/JETCAS.2023. 3243267

Galal, S., Shacham, O., Brunhaver, J. S., Pu, J., Vassiliev, A., and Horowitz, M. (2013). "FPU generator for design space exploration," in *2013 IEEE 21st Symposium on Computer Arithmetic* (Los Alamitos, CA: IEEE Computer Society), 25–34. doi: 10.1109/ARITH.2013.27

Hashemi, S., Anthony, N., Tann, H., Bahar, R. I., and Reda, S. (2017). Understanding the impact of precision quantization on the accuracy and energy of neural networks. *Des. Autom. Test Eur. Conf. Exhib.* 2017, 1474–1479. doi: 10.23919/DATE.2017.7927224

Herbst, R., Coffee, R., Fronk, N., Kim, K., Kim, K., Ruckman, L., et al. (2022). "Implementation of a framework for deploying ai inference engines in FPGAS," in *Smoky Mountains Computational Sciences and Engineering Conference* (Springer: New York), 120–134. doi: 10.1007/978-3-031-23606-8_8

Johnson, J. (2018). Rethinking floating point for deep learning. arXiv preprint arXiv:1811.01721. doi: 10.48550/arXiv.1811.01721

Kathail, V. (2020). "Xilinx Vitis unified software platform," in Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (New York, NY: Association for Computing Machinery), 173–174. doi: 10.1145/3373087.3375887

Northcutt, C. G. (2019). *Towards Reproducibility: Benchmarking Keras and PyTorch*. Cambridge, MA: L7. Retrieved from: https://l7.curtisnorthcutt.com/towards-reproducibility-benchmarking-keras-pytorch

Oudad, A. (2021). Comparing Keras and PyTorch on Sentiment Classification. Retrieved from: https://adamoudad.github.io/posts/keras_torch_comparison/ sentiment_classification/

Paszke, A., Gross, S., Chintala, S., and Chanan, G. (2016). *PyTorch: Tensors and Dynamic Neural Networks in Python With Strong GPU Acceleration*. Menlo Park, CA: Facebook AI Research. Retrieved from: https://pytorch.org

Pietrołaj, M., and Blok, M. (2022). Neural network training with limited precision and asymmetric exponent. J. Big Data 9:63. doi: 10.1186/s40537-022-00606-2

PyTorch Contributors (2024). *PyTorch Documentation: Numerical Accuracy*. Available online at: https://docs.pytorch.org/docs/stable/notes/numerical_accuracy. html (accessed October 28, 2024).

PyTorch Forum (2017). *The Same Model Produces Worse Results on PyTorch than on TensorFlow*. Available online at: https://discuss.pytorch.org/t/the-same-model-produces-worse-results-on-pytorch-than-on-tensorflow/5380/2 (accessed October 28, 2024).

Viso, A. I. (2024). *PyTorch vs TensorFlow: A Comprehensive Comparison*. Available online at: https://viso.ai/deep-learning/pytorch-vs-tensorflow/ (accessed October 28, 2024)

Wang, C., Florin, E., Chang, H.-Y., Thayer, J., and Yoon, C. H. (2023). Specklenn: a unified embedding for real-time speckle pattern classification in x-ray single-particle imaging with limited labeled examples. *IUCrJ* 10, 568–578. doi: 10.1107/S205225223006115

Xilinx Inc. (2024). Xilinx KCU1500 Acceleration Development Kit. Available online at: https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/dk-u1-kcu1500-g.html (accessed September 9, 2024).