Towards a Framework for Object Tracking using Heterogeneous In-Pixel Machine Learning Hardware

***Abstract*—Object tracking systems are constrained by the rate at which image data can be transmitted from image sensors to machine learning hardware. Contemporary imaging systems transmit magnitudes of data on the order of 30 gigabits per second, while back-end intelligent processing systems operate at a level significantly below this. The issue is exacerbated by the communication latency induced from data transmission between hardware devices. As a result, recent works have proposed in-pixel processing, a technique which uses custom ASIC designs that can perform intelligent processing on raw image data. While in-pixel processing has been demonstrated as an effective solution, this approach is limited by the long design cycle of custom ASICs and limited space on these ASICs for all intelligent processing. Heterogeneous architectures offer an alternative approach to this issue by leveraging the flexibility and low cost of FPGA and GPU platforms in parallel with processing in-pixel technology.**

In this paper, we propose a framework for designing and implementing a heterogeneous machine learning system that combines the advantages of in-pixel processing, FPGA, and GPU hardware. Our resulting design uses a custom in-pixel computing ASIC and a Zynq UltraScale MPSoC to model a ResNet-50 CNN for object detection on raw image data from the BDD100K data set. Furthermore, an NVIDIA Jetson AGX Xavier is included to perform multi-object tracking using PyTorch-based tracking software. An emulation of our system demonstrates an estimated peak power dissipation of 39.43 watts, a throughput of 13 frames per second, and an IDF1 tracking score of 72.3%. In this paper, we further discuss the challenges in developing this heterogeneous system and provide insights to guide future approaches.

I. Introduction

Contemporary image sensors operating at high frame rates and high resolutions generate significant amounts of data. High-end imaging sensors found in devices such as the iPhone 14, GoPro Hero9, and Sony Alpha 7S III capture 1080p resolution video at rates of 240 frames per second, achieving an effective throughput of nearly 30 gigabits per second [1]–[3]. This magnitude of data makes it it difficult for embedded computing systems to utilize this data in real-time with machine learning algorithms. Edge GPUs like the NVIDIA Jetson TX1 struggle to execute large state-of-the-art object detection algorithms (e.g., ResNet-50) at frame rates higher than 30 FPS [4].

In response to this performance gap, in-pixel processing (IP2) has been proposed as a method to enable computation on raw image data without a dedicated image signal processor (ISP) [5]. IP2 systems are characterized by their ability to bring high performing convolutional neural network (CNN) algorithms closer to the image sensors of video processing systems. This technique eliminates the need for an ISP to read and filter raw image data, thus providing improvements in latency, energy efficiency and bandwidth reduction. However, implementing an entire CNN in an IP2 circuit can be an expensive undertaking with limited flexibility to adapt to emerging CNN architectures. Heterogeneous hardware systems offer a unique solution to this issue by combining the strengths of reprogrammable FPGAs and GPUs with the performance of IP2 circuits.

Our work proposes the RPIXELS framework (Recurrent Neural Network Processing In-Pixel for Efficient Low-energy Heterogeneous Systems), a methodology for developing a heterogeneous system to perform multi-object tracking, illustrated in Figure 1. By following this framework, a user can transform a Python model of an object tracking system, into an embedded hardware system that uses a processing in-pixel ASIC and a Xilinx FPGA for neural network inference, and an embedded GPU for object tracing. Our contributions include:

- A systematic approach for implementing a CNN across custom IP2 and FPGA hardware, and interfacing it with a GPU tracking system.
- An estimation of the performance of our demonstrative heterogeneous system to support RPIXELS' feasibility.
- Suggestions for improvements and future approaches for realizing heterogeneous machine learning hardware.

II. Background

A. In-Pixel Computing

In recent years, in-pixel computing has been proposed as a technique to bring neural network inference closer to image sensors. [6] proposes a novel in-pixel computing technique, namely Processing-In-Pixel-in-Memory (P^2M), for convolutional neural networks that achieves competitive object tracking performance on the BDD100K data set. The solution works by integrating the first convolutional layer of typical CNNs in the pixel array of an image sensor circuit. Since the convolutional operations compress the amount of data sent from the sensor, the sensor's bandwidth is reduced. Furthermore, the larger system's throughput and energy efficiency are improved by offloading demanding convolutional operations into high-speed, low-power analog circuitry. This approach has additional benefits compared to similar works due to its ability to be manufactured with standard CMOS fabrication technology and to handle high resolution video feeds.

Figure 2 illustrates the mechanisms used to implement the first layer of ResNet-50 within an in-pixel computing ASIC. As described in [6], when a new feature map needs to be generated, the following procedures occur:

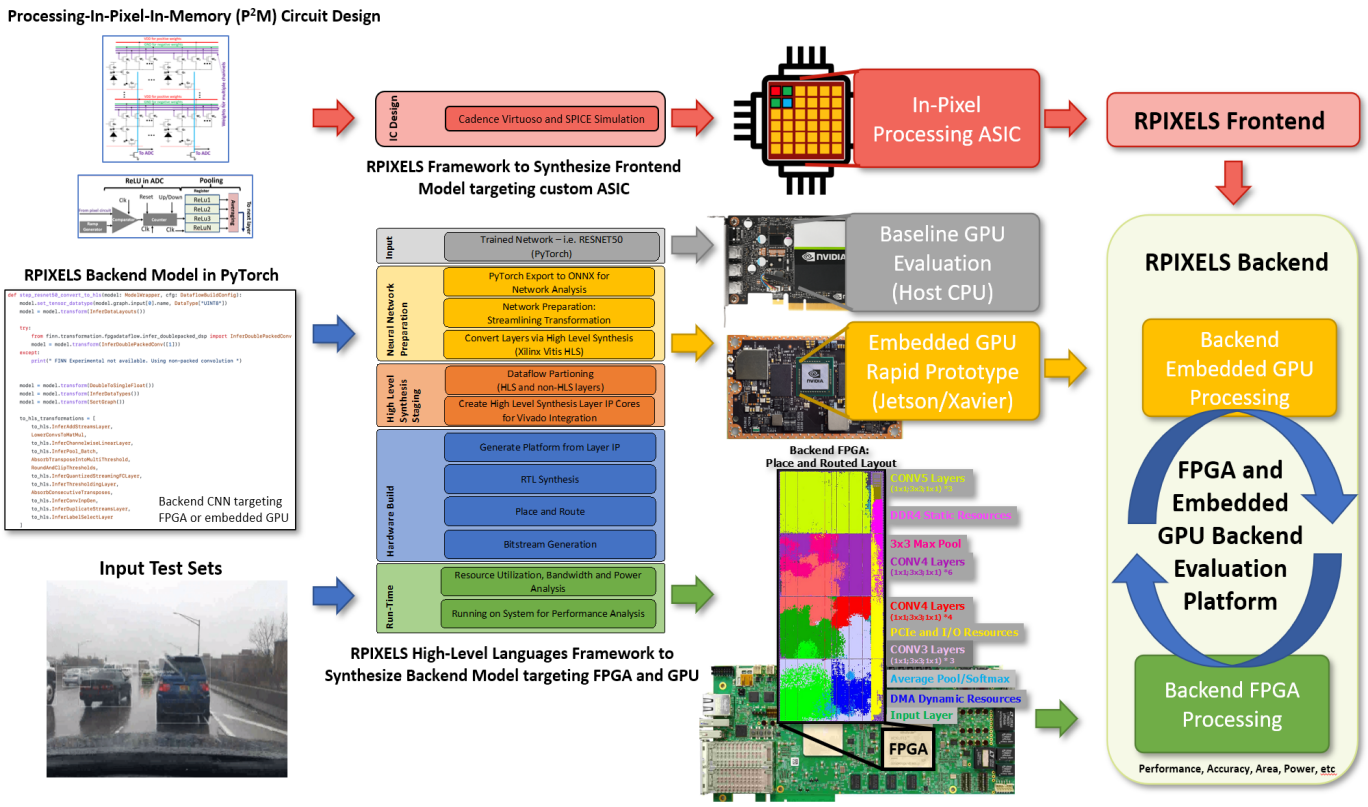

Fig. 1: An overview of the hardware, phases, and tools used in the Recurrent Neural Network Processing In-Pixel for Efficient Low-energy Heterogeneous Systems (RPIXELS) framework. Tools such as PyTorch, Vitis HLS, FINN and Cadence Virtuoso form the backbone of this framework.

1) The gate voltages of a specific input channel's weight transistors (W_i) are set to VDD. Their output drives the voltage at each G_S transistor.
2) Light exposed to each pixels' photodiodes (M) modulates current through each G_S.
3) A subset of resulting pixel outputs from each G_S is collected by setting the gate voltages of G_H to VDD.
4) The pixel output voltages are converted into a quantized digital representation via an ADC. This emulates the ReLU activation function.

B. Heterogeneous Neural Network Systems

While the P^{2}M technique introduced in [6] provides gains in energy and throughput, it does not support programmable weights. This is acceptable at early stages in the CNN where pre-trained layers can serve as generic high-level feature extractors, but poses a limitation for more specialized later stages. This is an important consideration since new network architectures continue to emerge each year, which boast higher accuracy than previous iterations, as shown by publications recorded on *Papers With Code* [7]. As opposed to a system solely consisting of an analog circuit with fixed network weights and layers, a heterogeneous approach that leverages a combination of in-pixel computing technology with reprogrammable hardware can provide a balance between application performance and adaptability to emerging algorithms.

In [8], the primary challenges of creating heterogeneous deep learning hardware are summarized. In particular, distributing computation, synchronizing data and communicating information between platforms are the most urgent tasks that must be resolved to achieve ideal heterogeneous systems. Recent works have successfully tackled these issues for FPGA-GPU and FPGA-CPU systems. [9] proposes FARNN, a GPU-FPGA RNN training accelerator that offloads recurrent weight matrix storage and matrix-vector multiplication to a Xilinx UltraScale+ VU9P FPGA. FARNN allows for RNN training in 60% of the time of a standalone GPU approach, and reduces energy consumption by up to 13%. [10] proposes a architecture that uses an NVIDIA Jetson TX2 to perform floating point convolutional operations, while a Xilinx Artix-7 FPGA executes fixed point fully connected layer operations. This approach has a low peak power dissipation at 3.7 watts, and can potentially execute up to 24% faster than the standalone NVIDIA Jetson TX2.

While FPGA-GPU co-processing techniques have been studied extensively, embedded ASIC-FPGA systems are rarely explored for neural network inference acceleration. Heterogeneous ASIC-FPGA systems have already proven to be

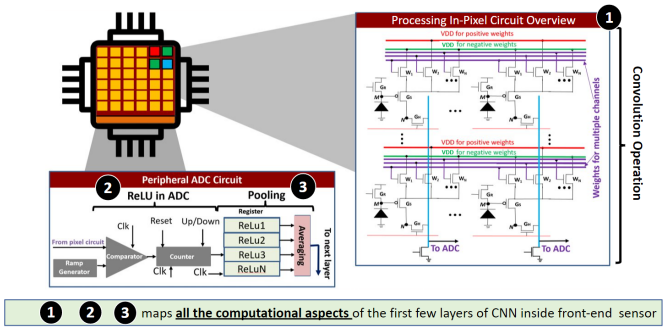

Fig. 2: Processing-In-Pixel-in-Memory (P^2M) circuit used to design the ASIC.

viable solutions for emerging embedded applications. For example, contemporary software defined radios, such as Ettus Research's USRP devices, combine RF front-end ASICs and Xilinx MPSoC FPGAs [11]. The FPGA in these software radios are capable of being reconfigured to match the application requirements of a given RF task, while the RF front-end provides high speed amplification, filtering and ADC functions. With these capabilities, the USRP has become the de facto platform for rapid prototyping and development of new RF systems, while maintaining comparable performance and competitive cost to other state-of-the-art approaches. Similar advancements can likely be achieved in deep learning-focused platforms.

C. Neural Network Compilers

Given the diverse skill set required to develop heterogeneous machine learning systems, it is valuable to use existing frameworks to streamline development. Existing neural network frameworks like PyTorch provide robust support for GPUs, with several supplemental tool kits to support applications such as multi-object detection and tracking, including QDTrack and GTR [12], [13]. Additionally, several teams have recently proposed high-level synthesis frameworks to translate PyTorch neural networks into a synthesizable RTL representation for implementation on FPGAs. Xilinx's Vitis AI integrated development environment allows developers to use deep learning processing units (DPUs) to implement neural networks [14]. In addition to this, Xilinx has also released FINN, which uses a more fine-grained approach to implementing neural networks on FPGAs [15]–[17]. Intel has also contributed to these tools by providing OpenVINO, a deep learning toolkit that optimizes neural network models across a variety of Intel processors and FPGAs [18].

Limited comparisons between these types of neural network implementation frameworks have been made in the past. [19] and [20] identify the strengths and shortcomings of several coarse-grained and fine-grained neural network inference approaches. For example, [19] illustrates that Vitis AI provides the best overall approach for implementing networks in large Xilinx FPGAs, but limits hardware modification with proprietary IP. On the other hand, FINN provides open-source implementations, but suffers from reduced accuracy compared to other approaches. [21] compares FINN and Vitis AI directly by benchmarking their implementations of a network architecture called YoloFINN. The study showed that FINN was able to achieve 55% higher energy efficiency and 208% higher throughput, while operating at a slightly lower power budget and higher quantization level. [22] compares ResNet-8 implemented on FPGA using both FINN and Vitis AI, an NVIDIA Jetson Nano, and a Core i7-11700K processor. This work similarly shows FINN outperforming Vitis AI, Jetson and Core i7. FINN achieves lower latency, higher throughput, and higher power efficiency than Vitis AI without significantly compromising accuracy.

While FINN is capable of efficiently producing FPGA synthesizable neural network designs that can be reconfigured, custom ASIC versions of its outputs would outperform the FPGA implementations. Analog computing techniques, such as P^2M discussed in Section II-A, further highlight their potential advantages. However, custom ASIC designs that use these techniques have a long design life cycle, are expensive to produce, and cannot be updated based on emerging state-of-the-art techniques without redesigns.

Alternatively, no published works appear to explore the idea of dividing a CNN's layers between FPGA and analog integrated circuit components. This approach has the potential to strike a balance between the trade offs of performance, cost and design time for machine learning hardware. In the following section, we propose the RPIXELS framework, a methodology for designing heterogeneous machine learning systems to perform object detection and tracking.

III. RPIXELS Framework

Using the principles established in the background of this paper, we developed the Recurrent Neural Network Processing In-Pixel for Efficient Low-energy Heterogeneous Systems (RPIXELS) framework. Using this framework as a guide, users can transform a PyTorch model of a multi-object tracking system into an embedded hardware system that uses a processing in-pixel ASIC and a Xilinx FPGA for convolutional neural network inference, and an embedded GPU for multi-object tracing.

The output hardware architecture and functional blocks of RPIXELS are depicted in Figure 3. Here, the **front-end** ASIC consists an image sensor to capture a raw video feed, and the analog P^2M circuitry proposed in [6]. Convolution, batch normalization, ReLU activation, and pooling operations are performed on the pixel data before it is sent to the back-end. The **back-end** consists of the FPGA and GPU which use arrays of processing elements and external DRAM to perform inference and tracking operations. Output data from the ASIC first gets received by the FPGA, which executes the remaining CNN feature extraction layers. Output from the FPGA gets sent to the GPU which runs the region proposal network to identify bounding boxes and performs region of interest (ROI) pooling for object detection and tracking.

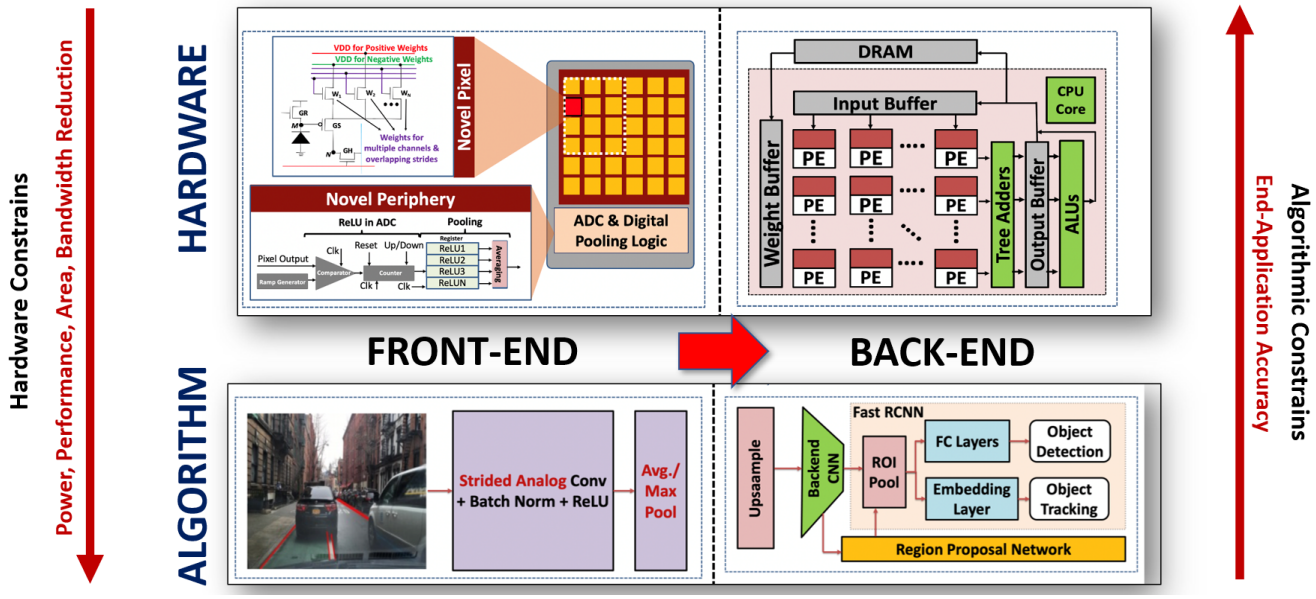

Fig. 3: RPIXELS output hardware architectures and functional blocks. Hardware constraints like power, performance, and area influence algorithmic processes. Similarly, algorithmic constraints like accuracy requirements influence hardware architecture decisions.

The RPIXELS framework consists of three primary phases: ① software modeling, ② model partitioning and ③ hardware mapping.

A. Phase 1: Software Modeling

In the first phase, a designer creates a Python model of the object tracking system, which should reflect the entire system's operation: from obtaining raw image data for processing, to performing object-tracking and visualization. Existing tracking libraries can be used in this phase which typically employ PyTorch neural networks for object detection. Examples of these libraries include Quasi-Dense Tracking (QDTrack) [23], Global Tracking Transformers (GTR) [24], or CenterTrack [25]. It is essential that PyTorch is used to implement the neural network of the functional model because it is supported by other tools used throughout this framework, such as FINN and PYNQ.

B. Phase 2: Model Partitioning

In the second phase, the software model is deconstructed to leverage the advantages of specialized hardware platforms. First, a designer extracts the neural network out of the object tracking library, quantizes it to use a fixed bit width, and partitions it for implementation on the ASIC and FPGA. This allows dedicated hardware implementations to improve the energy efficiency and bandwidth of the system. Here, a designer must consider the trade offs between implementing certain neural network layers in the ASIC versus the FPGA. For example, if several convolutional layers are implemented in the ASIC, the system bandwidth and energy efficiency will improve, but design time will significantly increase. Alternatively, if more convolutional layers are implemented in the FPGA fabric, the system will be quicker to deploy but will not demonstrate peak performance.

C. Phase 3: Hardware Mapping

Once the neural network is partitioned, design of the ASIC and FPGA occur simultaneously. On the ASIC side, an analog circuit designer uses tools like Cadence Virtuoso to create an ASIC design that incorporates the image sensor logic and neural network layers prior to those partitioned for the FPGA.

On the FPGA side, Xilinx's FINN compiler [15] can be used to synthesize and generate a bitstream for a target platform. FINN works by transforming a PyTorch neural network quantized with Brevitas [26] into an FPGA representation using predefined Vitis High-Level Synthesis (HLS) cores [27].

In parallel with ASIC and FPGA development efforts, the tracking portion of the software model is optimized for execution on an embedded GPU. The aforementioned tracking libraries feature support for NVIDIA GPUs through PyTorch's use of CUDA. Therefore, NVIDIA's Jetson products are an ideal choice since they are designed for use in low power embedded systems [28], [29]

Once hardware design is complete, the hardware platforms are interconnected. The ASIC first uses pixel sensor data to run the initial convolutional layers of the network. The output activation maps are then converted using an ADC and sent to the FPGA through a transceiver interface. The Xilinx FPGA uses a high speed transceiver to receive data from the ASIC. The FPGA then processes the remaining network layers and uses Ethernet to transmit results to the NVIDIA Jetson GPU at a peak data rate of 940 megabits per second. Finally, the GPU performs the remaining tracking tasks and provides an

output video feed with bounding boxes corresponding to the tracked objects.

IV. Framework Evaluation

A. Experimental Setup

To evaluate the feasibility of this framework, we measured the peak power consumption and throughput of the ASIC, FPGA and GPU independently. Cadence Virtuoso was used to measure the results for the custom in-pixel processing circuit. Xilinx Vivado was used to measure the results for a ZCU104 FPGA board which implemented a quantized MobileNet CNN synthesized with FINN. An NVIDIA Jetson AGX Xavier running QDTrack, GTR, and CenterTrack was used to measure the GPU results. While the ASIC has yet to be fabricated, we developed software emulations to verify its base functionality for demonstration purposes.

B. Results

1) Emulation Accuracy: As noted in Phase 1 of the RPIXELS framework, a software model of the system was developed in Python to validate its accuracy. QDTrack was used as the tracking method in this emulation, which features a ResNet-50 backbone CNN. For the emulation, we substituted the first convolutional layer of the ResNet-50 layer with a custom layer representing the P^{2}M's hardware functions. QDTrack was then executed with this substitution to record the multi-object tracking accuracy.

The system was evaluated based on its IDF1 score, a commonly used metric to compare correct object detections to the total number of expected detections in tracking benchmarks. Our QDTrack-based emulation was able to achieve an IDF1 score as high as 72.3% which is consistent with SoTA results [30]. We further analyze the possibility of using transformer tracking schemes by evaluating GTR's performance. The GTR emulation yielded an IDF1 score of 70.4%. These experiments show that through the use of the RPIXELS framework different algorithms can be quickly implemented and evaluated on the prototype platform. This allows developers to refine their models and improve accuracy, thus enabling improvement opportunities past-ASIC fabrication through the use of the FPGA reconfigurability.

2) ASIC Hardware Performance: Data was collected to show the estimated performance of the individual hardware platforms developed using the RPIXELS framework. First, we collected performance data for the in-pixel processing circuit from a Cadence Virtuoso SPICE simulation using GlobalFoundries' 22nm FD-SOI process. The attributes of the convolutional layer implemented in the circuit are shown in Table I.

Using the convolutional layer attributes, the bandwidth reduction achieved by using the in-pixel processing circuit compared to a stand-alone image sensor can be estimated using the formula below:

$$\frac{H \times W \times N_{InChannels} \times N_{Bits} \times \frac{4}{3}}{(\frac{H-K}{S}+1) \times (\frac{W-K}{S}+1) \times N_{OutChannels} \times N_{ReLU}} \quad (1)$$

TABLE I: Convolutional Layer Attributes for the In-Pixel Processing Circuit.

Convolutional Layer Attribute	Value
Image Width W	1280
Image Height H	720
Number of Input Channels $N_{OutChannels}$	3
Number of Output Channels $N_{InChannels}$	16
Number of Bits per Pixel N_{Bits}	12
Kernel Size K	7
Stride S	6
Activation Function (ReLU) Output Bits N_{ReLU}	8

In this equation, we calculate the ratio between the number of bits transmitted from a standard image sensor (top), to the number transmitted from our P^{2}M circuit (bottom). The metrics from Table I and Equation 1 indicate the bandwidth can be reduced by a factor of 13.6 compared to the conventional image sensor approach. Furthermore, the simulation results indicated that, given an image size of 1280 pixels by 720 pixels, the circuit achieves a peak power dissipation of 150.96 milliwatts and a frame rate of 17 frames per second.

3) FPGA Hardware Performance: Next, the FPGA performance was evaluated using FINN and Xilinx Vivado. FINN required 26 hours to build the 4-bit quantized MobileNet model for the ZCU104. This model was chosen for preliminary analysis because it was able to fit within the XCZU7EV-2FFVC1156 MPSoC located on the ZCU104 board. The resulting implementation used the majority of LUTs, BRAMs, and URAMs on the device. The complete utilization breakdown is shown in Table II. The MobileNet implementation achieved a throughput of 91 FPS when configured with a batch size of 1 with a 100 MHz clock rate, while the estimated peak power dissipation was 9.723 watts. These measurements are shown in Table III.

TABLE II: MobileNet ZCU104 Implementation Utilization.

Resource	Utilization	Available	Utilization %
LUT	189474	230400	82.24
LUTRAM	9781	101760	9.61
FF	167276	460800	36.30
BRAM	281.50	312	90.22
URAM	69	96	71.88
DSP	50	1728	2.89
BUFG	26	544	4.78

TABLE III: MobileNet ZCU104 Performance.

Metric	Measurement
Throughput	91.443 FPS
Clock Frequency	100 MHz
Dynamic Power	8.974 W
Static Power	0.750 W

4) GPU Hardware Performance: When running the unpartitioned tracking software which included neural network inference, the NVIDIA Jetson AGX Xavier demonstrated poor performance in executing each object tracking algorithm. This

is true even when configured in the highest power mode as shown in Table IV.

TABLE IV: NVIDIA Jetson AGX Xavier Performance.

Metric	Measurement
QDTrack Throughput	2.19 FPS
QDTrack Throughput (Tracking Only)	12.99 FPS
GTR Throughput	0.95 FPS
QDTrack Throughput	1.925 FPS
Clock Frequency	2100 MHz
Peak Power	30 W

To derive a better estimate of the Jetson's peak tracking performance, a software profiling tool called cProfile was used to identify the method calls within QDTrack that executed neural network functions. These function calls include operations such as 2D convolution, batch normalization, and matrix multiplication. The profiler indicated that out of the 7.47 seconds required to process 30 frames in QDTrack, only 2.31 seconds were spent executing tracking functions while the remaining 5.16 seconds were spent performing neural network inference. If inference is fully executed on the ASIC and FPGA portions of this system, which both achieve higher frame rates, the GPU can achieve a throughput of nearly 13 frames per second.

C. Discussion

The combined results of our experiments suggest that the final system will be able to achieve an effective throughput of 13 frames per second, a peak power consumption of 39.43 watts, and an object tracking accuracy of 72.3% IDF1.

The results show that the GPU bottlenecks the throughput significantly. While the ASIC and FPGA achieve frame rates of 17 and 91 FPS respectively, the GPU is limited to 13 FPS. QDTrack, GTR, and CenterTrack likely need to be better optimized for the NVIDIA Jetson AGX Xavier which only features 512 CUDA cores compared to the thousands found on server grade GPUs [28]. Alternatively, the Jetson AGX Xavier could be replaced with the Jetson Orin. The Orin has a slightly higher power budget at 40 watts with 2,048 CUDA cores: four times as many found on the AGX Xavier [29].

Another challenge experienced in the development of this system was the implementation of larger CNNs on our FPGA. We attempted to synthesize ResNet-50 on the ZCU104 board, but experienced implementation issues because of insufficient FPGA resources. FINN provides a ResNet-50 model that can be successfully compiled, but only for a much larger, server-grade Alveo U250 FPGA. One could resolve the issue of porting the ResNet-50 model to a smaller FPGA by further partitioning the hardware block design in Vivado. Using this approach, the network could be divided among several smaller FPGAs that may be more accessible for prototyping efforts. This approach has been demonstrated by the developers of FINN in previous work [31].

V. Conclusions

In this paper we proposed RPIXELS: a framework for developing heterogeneous in-pixel machine learning hardware. RPIXELS leverages in sensor processing, heterogeneous computing, and neural network compilers to streamline the creation of real-time object tracking systems. The benefits of using the RPIXELS framework include: (1) a organized approach to partitioning object tracking software into a power-constrained heterogeneous system and (2) a bandwidth reduction of data sent from the image sensor. Design tools such as Cadence Virtuoso, Xilinx Vivado, FINN, and Python were used to transfer the software tracking model into the hardware domain.

We validated our methodology by emulating and measuring the power and throughput of a demonstrative design. The design demonstrates a 13 times reduction in data sent from the image sensor to the back-end processing hardware. The design's estimated peak power dissipation was 39.43 watts, and its throughput is estimated to be 13 frames per second at a frame resolution of 1280 pixels by 720 pixels.

This paper reports the detailed multi-object tracking accuracy based on purely software experiments. To more accurately evaluate the system's real-time throughput, we plan to emulate the IP2 ASIC in a placeholder FPGA in future works. This would allow for a nearly complete system to be developed wherein the placeholder FPGA can be substituted with a fabricated IP2 ASIC board. After FPGA emulation, experimental in-pixel hardware will be fabricated to validate the obtained results.

Extensions of this work could seek to incorporate more CNN layers into the front-end ASIC. This would allow for further bandwidth reduction and push the system towards real-time frame rates. Research in dividing large CNNs among several FPGAs could also add more flexibility to solutions created using this framework.

This work provides only a limited example of how heterogeneity can be exploited in machine learning embedded systems. Future work could seek to enhance performance and capability using 3D heterogeneous integration (3DHI) packaging techniques. 3DHI would allow the distinct hardware platforms used in this work to be combined into a single package, thus improving bandwidth and energy efficiency through direct through-silicon via connections [32].

References

[1] iPhone 14. [Online]. Available: https://www.apple.com/iphone-14/specs/

[2] GoPro Hero9 Black. [Online]. Available: https://gopro.com/en/us/shop/cameras/hero9-black/CHDHX-901-master.html

[3] Sony Alpha 7S III. [Online]. Available: https://electronics.sony.com/imaging/interchangeable-lens-cameras/all-interchangeable-lens-cameras/p/ilce7sm3-b

[4] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE access*, vol. 6, pp. 64 270–64 277, 2018.

[5] Defense Advanced Research Projects Agency, *Artificial Intelligence Exploration (AIE) Opportunity, DARPA-PA-20-02-09, In-Pixel Intelligent Processing (IP2)*, Defense Advanced Research Projects Agency, Arlington, VA, 2021.

[6] "Removed for blind review."

[7] State-of-the-art image classification on imagenet. [Online]. Available: https://paperswithcode.com/sota/image-classification-on-imagenet

[8] Q. Wu, Y. Shen, and M. Zhang, "Heterogeneous computing and applications in deep learning: A survey," in *Proceedings of the 5th International Conference on Computer Science and Software Engineering*, 2022, pp. 383–387.

[9] H. Cho, J. Lee, and J. Lee, "Farnn: Fpga-gpu hybrid acceleration platform for recurrent neural networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, pp. 1725–1738, 2021.

[10] Y. Tu, S. Sadiq, Y. Tao, M.-L. Shyu, and S.-C. Chen, "A power efficient neural network implementation on heterogeneous fpga and gpu devices," in *2019 IEEE 20th international conference on information reuse and integration for data science (IRI)*. IEEE, 2019, pp. 193–199.

[11] M. Ettus and M. Braun, "The universal software radio peripheral (usrp) family of low-cost sdrs," *Opportunistic spectrum sharing and white space access: The practical reality*, pp. 3–23, 2015.

[12] T. Fischer, J. Pang, T. E. Huang, L. Qiu, H. Chen, T. Darrell, and F. Yu, "Qdtrack: Quasi-dense similarity learning for appearance-only multiple object tracking," *arXiv preprint arXiv:2210.06984*, 2022.

[13] X. Zhou, T. Yin, V. Koltun, and P. Krähenbühl, "Global tracking transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8771–8780.

[14] Vitis AI. [Online]. Available: https://xilinx.github.io/Vitis-AI/

[15] Xilinx FINN. [Online]. Available: https://xilinx.github.io/finn/

[16] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 65–74.

[17] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[18] "Intel OpenVINO." [Online]. Available: https://docs.openvino.ai/latest/home.html

[19] M. Lebedev and P. Belecky, "A survey of open-source tools for fpga-based inference of artificial neural networks," in *2021 Ivannikov Memorial Workshop (IVMEM)*. IEEE, 2021, pp. 50–56.

[20] P. Plagwitz, F. Hannig, M. Ströbel, C. Strohmeyer, and J. Teich, "A safari through fpga-based neural network compilation and design automation flows," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 10–19.

[21] M. Machura, M. Danilowicz, and T. Kryjak, "Embedded object detection with custom littlenet, finn and vitis ai dcnn accelerators," *Journal of Low Power Electronics and Applications*, vol. 12, no. 2, p. 30, 2022. [Online]. Available: https://doi.org/10.3390/jlpea6010003

[22] F. Hamanaka, T. Odan, K. Kise, and T. Van Chu, "An exploration of state-of-the-art automation frameworks for fpga-based dnn acceleration," *IEEE Access*, 2023.

[23] Quasi-Dense Tracking GitHub. [Online]. Available: https://github.com/SysCV/qdtrack

[24] Global Tracking Transformers GitHub. [Online]. Available: https://github.com/xingyizhou/GTR

[25] CenterTrack GitHub. [Online]. Available: https://github.com/xingyizhou/CenterTrack

[26] Xilinx Brevitas. [Online]. Available: https://github.com/Xilinx/brevitas

[27] Vitis HLS Library for FINN. [Online]. Available: https://github.com/Xilinx/finn-hlslib

[28] Jetson AGX Xavier. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/

[29] Jetson Orin. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/

[30] J. Pang, L. Qiu, X. Li, H. Chen, Q. Li, T. Darrell, and F. Yu, "Quasi-dense similarity learning for multiple object tracking," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2021.

[31] T. Alonso, L. Petrica, M. Ruiz, J. Petri-Koenig, Y. Umuroglu, I. Stamelos, E. Koromilas, M. Blott, and K. Vissers, "Elastic-df: Scaling performance of dnn inference in fpga clouds through automatic partitioning," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 2, pp. 1–34, 2021.

[32] S. Zhang, Z. Li, H. Zhou, R. Li, S. Wang, K.-W. Paik, and P. He, "Recent prospectives and challenges of 3d heterogeneous integration," *e-Prime-Advances in Electrical Engineering, Electronics and Energy*, p. 100052, 2022.