Hijacking the Embedded GPU: Accelerating Neural Network Inference on Raspberry Pi

1st Eric Rippey
Virginia Tech Computer Science Dept.
Blacksburg, U.S.A
erippey3@vt.edu

***Abstract*—In the evolving landscape of Industry 4.0, real-time processing and intelligent decision-making are paramount to optimizing industrial automation and smart manufacturing. However, the reliance on cloud-based computation introduces undesirable latency in time-sensitive applications. This proposal explores an unconventional approach: repurposing the embedded GPU found in the Raspberry Pi 5 for neural network inference. Traditionally dedicated to rendering graphics, this GPU exhibits untapped computational potential that can be redirected to accelerate machine learning tasks at the edge. We outline a methodology to adapt GPU functionalities beyond their conventional scope, evaluate performance trade-offs, and present preliminary experimental insights. We intend to realize this functionality by porting OpenCL kernels to Vulkan, which the Pi 5's GPU supports.**

***Index Terms*—Embedded Systems, Neural Network, GPU Acceleration, Edge Computing, Raspberry Pi, OpenCL, Vulkan**

I. Introduction

The current era of industry is often referred to as Industry 4.0, or the fourth industrial revolution. It is marked by the shift in industry operations through the use of automation and smart technologies. This transformation has seen the emergence of smart factories and intelligent machines that significantly improve productivity and operational efficiency. Technology is at the center of this revolution, including the emergence of Internet of Things (IoT), Cloud Computing, Artificial Intelligence, Machine Learning, and Edge Computing. These technologies provide a combination of real time data processing and adaptive decision making capabilities.

In industry today a popular edge device commonly used is the Raspberry Pi. This is a low cost, low power, edge device with high configurability which makes it desirable for deployment and embedded applications. A common use case for the Raspberry Pi is for monitoring fields such as, temperature, humidity, light intensity, water level, power consumption, current, and voltage [2], [3], [7].

Smart sensors and IoT devices are used in conjunction with Artificial intelligence to recognize patterns, detect anomalies, do classification, and make predictions to improve industry output and increase efficiency. Artificial Neural Networks are particularly strong at these tasks. Artificial Neural Networks have many subclasses with specific strengths, such as recurrent neural networks, used in text or speech recognition; convolution neural networks, used in feature extraction and image recognition; and deconvolutional neural networks, used in image denoising, and image upscaling [5].

In this study we propose using the popular edge device, the Raspberry Pi, to compute for more computationally expensive tasks, such as running inference using ANNs. Specifically, we want to look at extracting performance from the GPU on the Raspberry Pi, which is typically used only for graphics rendering. The ultimate goal is to determine the efficacy of using the Pi to perform more processing at the edge before sending data to

the cloud.

II. Motivation

The current standard is to do data collection at the edge, and send the data to the cloud where all of the analysis is run. A literature review of Industry 4.0 strategies found that edge devices where most commonly used for data collection and that data was sent upstream for processing, machine learning, decision making, etc [4].

There are two problems with this approach. The first is that it has a high latency cost associated with moving data from the edge to the cloud for analysis. For most tasks, this is a good approach as the cloud is far more computationally capable and analysis is expensive. In time sensitive applications, high latency can be unideal. Examples of time sensitive tasks where decision making needs to be near instantaneous are in real time quality control, broad area traffic control, predictive maintenance, autonomous vehicle object detection, and intelligent patient monitoring [18]..

The other issue is that while, digitizing manufacturing has the benefits of improving productivity, and recourse efficiency, the automation of this work is associated with high energy consumption and high recourse use. If it is possible to do computations on low powered devices, this may lead to lower overall power consumption [1].

The idea behind this research is to investigate the viability of running inference on common edge devices. This will enable real time decision making for time sensitive tasks at a reasonable power consumption. One domain where bringing inference computation the edge could be especially transformative is image deconvolution, particularly the deconvolution of Low Dose CT images. This has applications such as giving accurate COVID-19 diagnosis within minutes of scanning [10] and classify and flag mobile CT, X-ray, MRI scans for TB, pneumonia, microcalcifications, and stroke [19].

Several options exist for such a tasks. We have chosen to use the Raspberry Pi Compute Module 5 for this proposal. The Raspberry Pi 5 and Compute Module Variant are a line of versatile single board computer that run between 2.5 and 12 watts. Raspberry Pi is the choice for this research in specific for a few reasons. For one, it was chosen for its wide availability and acceptance. The Raspberry Pi 5 did over a million sales in its first 11 months of production [6]. It is a widely available between $70-120 based on the memory capacity. Other options are available such as the NVidia Orin Jetson Nano and its Super variant as well as specialized devices such as the Google Coral TPU. Both of these devices boast more computational power. The Jetson Orin Nano Super, for example, specifically targets AI tasks. It has a 6 core Arm CPU, 1024 CUDA cores, and 32 tensor cores, capable of 67 TOPS at 7-2k watts [20]. However, the NVidia line of products are far more costly and consume more power. The Google TPU, while capable of 2 TOPS, is a specialized device whereas the Pi can do a myriad of tasks. Further, the Raspberry Pi is of interest because of its GPU, which for the most part has seen few use cases outside of display. We are interested in leveraging the compute power of the Pis GPU which has the potential to gain hold of untapped potential.

III. Background

The power of edge devices has improved drastically over the last decade. This has enabled more computation to happen closer to the edge. Comparing the first Raspberry Pi, released in 2012 compared to the Raspberry Pi 5, released in 2023, shows over a 150x improvement in CPU performance and 60x improvement in GPU performance in the last decade. These results where compiled from running Sysbench and GLMark2 on the CPU and GPU respectively.

The Raspberry Pi 5 is powered by the Broadcom BCM2712. This die consists of a 4 core, 4 thread CPU using the ARM architecture. Despite its impressive processing capabilities, the Pi 5 does not include a discrete GPU. Instead, it relies on the integrated Broadcom VideoCore VII, a graphics processor that supports OpenGL ES 3.1 and Vulkan 1.2. This chip is engineered with 12 cores, 128 shader units, and 8 execution units. The VideoCore VII is situated on the CPU die and utilizes dynam-

ically allocated SDRAM, which is not shared with the CPU.

Compared to a discrete GPU, the VideoCore VII is limited by both its operation support and its limited support of common heterogeneous programming languages. Both OpenGL ES and Vulkan are powerful products of the Khronos Group. However, compared to standards like CUDA, OpenACC, and OpenCL, which are standards for parallel programming on heterogeneous systems, OpenGL ES and Vulkan are specifically graphics APIs. Being restricted to the graphics standards can be a difficult challenge when trying to perform general purpose compute on a GPU. As a result, using the GPU for computationally heavy tasks such as neural network inference are not natively supported.

To circumvent this limitation, tools like clvk can be used. clvk is an implementation of OpenCL 3.0 which uses clspv as its compiler. OpenCL 3.0 is a commonly used and portable compute language that can be used on many types of heterogeneous systems. clspv is a open sourced compiler for OpenCL built by Google. It transforms OpenCL C code into SPIR-V modules. SPIR-V modules are the common intermediate binary language that can be consumed by many Khronos Group standards including Vulkan and OpenCL. We plan to implement our code in OpenCL 3.0 standard using clvk and compile it into SPIR-V modules to be interpreted by Vulkan 1.2 which the VideoCore VII supports.

There is little in depth information available about the VideoCore VII architecture. However, the architecture guide for the VideoCore IV, which powers the Raspberry Pi 3 has been released [13]. the VideoCore IV 12 Quad Processors (QPUs). These are floating point shader processors that are 4-way SIMD multiplexed four ways to compute 16-way 32-bit SIMD processing. In other words, the processor executes the same instruction for four cycles on four different 4-way vectors, called 'quads'. The QPUs are organized into groups of four processors called slices. Slices share Instruction cache, Special Function Units, which compute reciprocals, reciprocal square root, logarithm, and exponential operations, as well as one or two Texture and Memory lookup Units. All four QPUs in a slice execute the same set of instructions and resynchronize afterward. The QPUs support packing and unpacking 8 bit unsigned integers, 16 bit signed integers, and 16 bit floats into a single 32 bit value for vector arithmetic.

Preliminary testing of the Raspberry Pi 5 found most of the information to be consistent between the VideoCore IV and the VideoCore VII. The Raspberry Pi 5 has two rendering devices at its disposal. The VideoCore VII and a llvm pipe which runs on the CPU as a fallback. The Raspberry Pi supports 32 bit integer and floating point operations.

A. Related Work

Related to programming for the Raspberry Pis GPU, there is has been little work done. The book "Raspberry Pi GPU Audio Video Programming" by Jan Newmarch looks at writing programs for the GPU through OpenGL, OpenMAX, and OpenVG [11]. It looks at GPU programming through the audio and video points of view rather than programming to the GPU as a general purpose device. There are also a myriad of Papers that look Fast Fourier Transformations (FFTs) utilizing the GPU. These studies utilize GPU_FFT, a program written by Andrew Holme in 2014 [12]. This implementation was written directly in bytecode to utilize the V3D block of the BCM2835, the chip of the Raspberry Pi 1. A spinoff paper also looked at a Vulkan FFT implementation to leverage the GPU [8]. There have been papers which look at Optimizing OpenCL code for the Raspberry Pi. However, these papers look at writing OpenCL code that efficiently maps to the Pi's CPU.

One paper which takes our proposed approach of compiling OpenCL code to Vulkan compute shaders using clvk [14]. Their research was on using heterogeneous systems to reconstruct 3D microwave tomography images from measuring electromagnetic signals. Their paper states their research can benefit from relatively low power portable technology. When running on the Raspberry Pi 4, their algorithm took over twice as long to run on the VideoCore VI as it took to run on the Raspberry Pi's CPU. This suggests there are roadblocks to extracting high performance using OpenCL and

clvk despite the additional compute resources they provide.

This area of research heavily reflects research done in the mid 2000's to bring general purpose compute to graphics hardware. During that time, GPUs where advancing from 8 bit chanel color value pipelines to fully programmable vectorized floating point operation pipelines [15]. GPUs of this time structure their computation into heavily parallelized graphics pipelines consisting of vertex operations, privative assembly, rasterization, fragment operations, and compression into an image. The common languages to interface with the GPU at the time where all based on the idea that the GPU would generate images. Such examples of languages included Cg, HLSL, OpenGL, and Sh. At this time there where early languages and libraries which attempted to map mathematical operations on memory (kernels) to shader streams. Such examples included Accelerator by Microsoft Research, CGiS, and Brook Programming Language. Early papers from this time explored crucial data parallel tasks and data structures such as map, reduce, scatter and gather, sorting, dense arrays, sparse arrays, and dynamic sparse arrays. Combining these ideas, complex algorithms are explored on GPUs such as differential equations, linear algebra, and data queries.

As seen in the case of the microwave tomography images study, simply having access to heavily parallel hardware does not guarantee an improvement in performance. Since the hardware of the Raspberry Pi is specifically tailored to rendering graphics similar to early GPUs, we hypothesize that modern optimizations may not all work. Looking further into specific optimizations made during the era of early general purpose GPU programming will likely be the best source help when programming for the Pi.

One final paper which is the backbone for lots of this research is a paper titled "ComputeCOVID19+: Accelerating COVID-19 Diagnosis and Monitoring via High-Performance Deep Learning on CT Images" [10]. This paper outlines their work enhancing the resolution of low dosage CT images through their deep learning approach called DenseNet Deconvolution Network, or DDnet. This paper goes over the entire pro- cess of enhancing the image, from the architecture of the network, to the data sets used for training, to the training methods, to inference optimization, which is vital to this study.

IV. Approach

Our original plan was to implement a neural network based on DenseNet Deconvolutional Network from the ComputeCOVID19 paper. Most important to our proposal, is their implementation of the neural network layers using OpenCL. This includes OpenCL kernels for convolution, pooling, unpooling, denseblock, and deconvolution. This gives us a good starting point to implement and port neural network layers to the Raspberry Pi. It has the added benefit of being able to lead into further research regarding mobile CT scanning and upscaling, another task which is potentially time sensitive or remote, where access to the cloud is limited.

Our ultimate goal is to port the layers of DDnet from OpenCL to Vulkan using clvk. The reason we are choosing to port OpenCL kernels rather than writing proprietary neural layers in API languages the Pi natively supports is for the sake of future portability. OpenCL is a widely adopted platform which is able to be run or ported to most architectures. This allows for future work and research on other embedded architectures.

The largest constraints will be the limited compute power of the Raspberry Pi, and more importantly, its lack of memory. We have recreated the 3D-DDnet model as described in the ComputeCOVID19 paper. This model was trained on a far more powerful system, with two 80 GB A100 GPUs. Training the model took 16 hours and 24 minutes on this system. However, running a single inference on a single 3D CT dataset can complete within seconds on CPUs with less than 32 cores. Running memory profiling on the model while doing inference shows it consumes 15 GB of memory at its peak. This provides a challenge as the Raspberry Pi come in 4, 8, and 16 GB models which will fail to accommodate inference of this model. Especially if we intend to run the inference on the GPU of the Raspberry Pi. Our exploration

thus far leads us to believe the current model will not fit on even the 16 GB Raspberry Pi.

Thus far, we have explored the limitations of using the GPU of the Raspberry Pi 5. This has been done through a mix of examining the outputs of vulkaninfo and clinfo, as well as through compiling and running OpenCL kernels to Vulkan shaders to test the capabilities of the VideoCore VII.

From testing 4GB and 8GB models of the Raspberry Pi 5, the GPU has access to half of the systems memory. The 4GB model has a global memory size of 2 GB and the 8 GB model has a global memory size of 4 GB. Other limitations have been found using OpenCL to Vulkan. OpenCL is unable to compile kernels with shorts, ushorts, longs, ulongs, doubles, halfs, and trigonometric functions such as sin, cos, tan, etc. These errors are attributed to unsupported data types of the VideoCore VII. Other less explainable errors have occurred during preliminary testing. These include kernels that contain booleans and chars causing the graphics pipeline to hang, preventing all future jobs sent to the GPU to fail to complete. This is something that needs further investigation as the VideoCore VII supports 8 bit operations.

Within the context of OpenCL, the Raspberry Pi has a few other limitations, these include a small local memory size of 16 KB, having a maximum work item size of $256 \times 256 \times 256$ and having a maximum work group size of 256. Additional constraints of clvk include that each OpenCL context may only have 1 device and that clvk does not support out of order execution.

Before we are able to port kernels of the DDnet to run on the Pi, further exploration of the VideoCore VII is necessary. Testing basic matrix multiply kernels on the VideoCore VII found that traditional optimizations such as blocking hurt the performance of the kernel. Further understanding of the memory layout of the VideoCore VII is needed to improve operational intensity and performance of simple kernels before moving to more complex ones such as the layers of the DDnet. We plan to further research optimizations made in the early years of general purpose GPU computing to see if any insights there transfer. We will explore per task granularity and group size granularity to find out what runs the best on the underlying hardware. From there we plan to compare the performance of a simple kernel such as matrix multiply against a multithreaded CPU implementation for the A76 CPU as well as against traditional GPUs.

Based on the shortcomings of the initial work we plan to do more benchmarking work before attempting to implement layers of the DDnet. Because of the initial difficulties utilizing traditional optimizations on the VideoCore VII and due to the findings of microwave tomography algorithm running twice as slow on the VideoCore VI compared to the CPU [14] we would like to step back and check the efficacy of running general purpose code on the Raspberry Pi's GPU by using the OpenDwarfs Benchmarking suite. The 13 Dwarfs are a set of fundamental algorithms that capture patterns in computation [17]. They are useful in place of traditional benchmarks to evaluate a hardware and software based on the types of programs expected to be most common. Testing the dwarfs on the Raspberry Pi's GPU for any noticeable performance benefits seems like a logical step to take before attempting to implement the layers of the DDnet architecture for the Pi. This step will also help us further understand what optimizations work well for the underlying hardware. Specifically we plan to test dense linear algebra and sparse linear algebra as they have are the most applicable in the field of artificial neural networks.

Further research towards implementing DDnet on the Raspberry Pi will unfold in stages. First, due to the initial limitations discovered on the Raspberry Pi, we will likely pivot to implementing a 2D-DDnet rather than the full 3D-DDnet as described in the ComputeCOVID19 paper. This has a far smaller memory footprint and can run in orders less time at the cost of accuracy and only computing a single layer of a CT image at a time. We plan to optimize the inference based on the findings of prior tests. If the VideoCore VII of the Raspberry Pi shows promise of performance, we hope to have a working 2D-DDnet inference at the end of this research.

Data Type	addition/subtraction	mult/div	mod	exp/exp2/exp10	log/log2/log10	pow	sqrt	sin/cos/tan/asin/acos/atan	floor/ceil/round/trunc/rint	fabs/fmin/fmax	mix/step/smoothstep	native_sin/native_exp/native_log
bool												
char												
uchar												
short												
ushort												
half												
int												
uint												
long												
ulong												
size_t												
float												
double												

TABLE I
PRELIMINARY TESTS OF SUPPORTED OPERATIONS.

Legend:

- **Green** – Operation is supported
- **Red** – Causes program to hang
- **Orange** – Unsupported or does not compile
- **Gray** – Untested

V. ANTICIPATED RESULTS

We have already gotten programs to compile and run on the GPU of the Raspberry Pi through the use of clvk. We anticipate that we will run many variations of simple kernels such as matrix multiply, dense linear algebra, and sparse linear algebra with different mappings and granularity. We intend to benchmark them all to find the most performant to better understand how to improve the operational intensity on the underlying hardware. We intend to report on the common themes between all kernels that improve performance and be able to explain why certain optimizations work on the VideoCore VII over others.

We anticipate being able to come to a conclusion on the efficacy of hijacking the GPU of the Pi using clvk for future work based on the performance results of our testing. We plan to produce performance results and runtime analysis of the a kernel running on the VideoCore VII versus a discrete GPU, the CPU of the Raspberry Pi, and potentially other devices. From this we will be able to draw conclusions specifically on the performance of OpenCL performance on the Pi using clvk. We do not intend to draw conclusions about the performance of the VideoCore VII as a whole but rather the combination of the VideoCore VII hardware and the clvk software. If we are unhappy with the results, further test could be to compare benchmarks of clvk to a natively implemented solution in Vulkan or OpenGL ES.

If the results of the Dwarfs benchmarks are promising, we intend to move forward with reproducing 2D-DDnet inference kernels in OpenCL. We plan to then gather results on the runtime of inference on the VideoCore VII. From these results, we intend on drawing conclusions on the efficacy of hijacking the GPU of the Raspberry Pi to run inference on ANNs. Further studies could look into implementing inference on a more powerful device such as a TPU or AI board like the Jetson Orin Nano and comparing the runtime results.

VI. CONCLUSION

In summary, this proposal aims to explore the untapped potential of the Raspberry Pi 5's embedded GPU for accelerating neural network inference. By leveraging the Broadcom VideoCore VII through a strategic porting of OpenCL kernels to Vulkan, the research intends to address key challenges related to latency and power consumption in edge computing environments. While preliminary expectations suggest that the performance improvements may be incremental due to inherent architectural constraints, the insights gained will provide a foundation for future optimizations and potential adaptations to other embedded platforms. Ultimately, this work seeks to contribute to the broader field of Industry 4.0 by enhancing the computational capabilities of low-cost, low-power devices, thus opening new avenues for real-time, on-device intelligence.

REFERENCES

[1] M. Ghobakhloo, "Industry 4.0, digitization, and opportunities for sustainability," *Journal of Cleaner Production*, vol. 252, p. 119869, Apr. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652619347390

[2] C. M. Endres, C. Pelisser, D. A. Finco, M. S. Silveira, and V. J. Piana, "IoT and Raspberry Pi application in the food industry: a systematic review," *Research, Society and Development*, vol. 11, no. 1, pp. e0 411 124 270–e0 411 124 270, Jan. 2022, number: 1. [Online]. Available: https://rsdjournal.org/index.php/rsd/article/view/24270

[3] M. D. Mudaliar and N. Sivakumar, "IoT based real time energy monitoring system using Raspberry Pi," *Internet of Things*, vol. 12, p. 100292, Dec. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660520301244

[4] I. Henao-Hernández, E. L. Solano-Charris, A. Muñoz-Villamizar, J. Santos, and R. Henríquez-Machado, "Control and monitoring for sustainable manufacturing in the Industry 4.0: A literature review," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 195–200, 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2405896319308778

[5] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, Nov. 2018, publisher: Elsevier. [Online]. Available: https://www.cell.com/heliyon/abstract/S2405-8440(18)33206-7

[6] R. King. Revenue radar: Supply chain disruption halts raspberry pi from further growth. [Online]. Available: https://the-cfo.io/2024/09/26/revenue-radar-supply-chain-disruption-halts-raspberry-pi-from-further-growth/

[7] E. Kadiyala, S. Meda, R. Basani, and S. Muthulakshmi, "Global industrial process monitoring through IoT using Raspberry pi," in *2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2)*, Mar. 2017, pp. 260–262. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8067944

[8] Q. He, V. Weaver, and B. Segee, "Comparing Power and Energy Usage for Scientific Calculation with and without GPU Acceleration on a Raspberry Pi Model B+ and 3B," in *Proceedings on the International Conference on Internet Computing (ICOMP)*. Athens, United States: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2018, pp. 3–9, num Pages: 3-9. [Online]. Available: https://www.proquest.com/docview/2139478522/abstract/C2EB4E71B3CE455FPQ/1

[9] V. Faerman, V. Avramchuk, K. Voevodin, and M. Shvetsov, "Real-Time Correlation Processing of Vibroacoustic Signals on Single Board Raspberry Pi Computers with HiFiBerry Cards," in *High-Performance Computing Systems and Technologies in Scientific Research, Automation of Control and Production*, V. Jordan, I. Tarasov, and V. Faerman, Eds. Cham: Springer International Publishing, 2022, pp. 55–71.

[10] G. Goel, A. Gondhalekar, J. Qi, Z. Zhang, G. Cao, and W. Feng, "ComputeCOVID19+: Accelerating COVID-19 Diagnosis and Monitoring via High-Performance Deep Learning on CT Images," in *50th International Conference on Parallel Processing*. Lemont IL USA: ACM, Aug. 2021, pp. 1–11. [Online]. Available: https://dl.acm.org/doi/10.1145/3472456.3473523

[11] J. Newmarch, *Raspberry Pi GPU Audio Video Programming*, 1st ed., ser. Technology in Action. Berkeley, CA: Apress, 2017. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-2472-4

[12] A. Holme. (2014) GPU_FFT: Fast fourier transform library for the raspberry pi. [Online]. Available: http://www.aholme.co.uk/GPU_FFT/Main.htm

[13] Broadcom Corporation, *VideoCore IV 3D Architecture Reference Guide*, June 2013, revision 1.0. [Online]. Available: https://docs.broadcom.com/doc/12358545

[14] T. Vasileiou and P. Kosmas, "Accelerated Medical Microwave Tomography via Heterogeneous Computing," *TechRxiv*, 2020. [Online]. Available: https://www.authorea.com/doi/full/10.36227/techrxiv.171198045.57783972?commit=7d

[15] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2007.01012.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01012.x

[16] W.-c. Feng, H. Lin, T. Scogland, and J. Zhang, "OpenCL and the 13 dwarfs: a work in progress," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. Boston Massachusetts USA: ACM, Apr. 2012, pp. 291–294. [Online]. Available: https://dl.acm.org/doi/10.1145/2188286.2188341

[17] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, and S. W. Williams, "The Landscape of Parallel Computing Research: A View from Berkeley," Dec. 2006. [Online]. Available: https://escholarship.org/uc/item/1z50m2xt

[18] C. Swetha, "Edge AI for real-time decision making in IOT networks," *International Journal of Innovative Research*, vol. 12, no. 9, pp. 11 293–11 309, Sep. 2024. [Online]. Available: https://www.researchgate.net/publication/387321332_Edge_AI_for_Real-Time_Decision_Making_in_IOT_Networks

[19] A. Kharat, V. A. Duddalwar, K. Saoji, A. Gaikwad, V. Kulkarni, G. Naik, R. Lokwani, S. Kasliwal, S. Kondal, T. Gupte, and A. Pant, "Role of edge device and cloud machine learning in point-of-care solutions using imaging diagnostics for population screening," *CoRR*, vol. abs/2006.13808, 2020. [Online]. Available: https://arxiv.org/abs/2006.13808

[20] NVIDIA Corporation, "Jetson orin nano developer kit datasheet," NVIDIA Corporation, Tech. Rep. 3575392-R2, 2023, accessed: 2025-04-11. [Online]. Available: https://nvdam.widen.net/s/zkfqjmtds2/jetson-orin-datasheet-nano-developer-kit-3575392-r2