e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

FIXED-POINT ARITHMETIC FOR EDGE DEVICES: A COMPARATIVE STUDY

Muhammad Nouman[*1]

[*1]University Of Engineering And Technology, Peshawar, Khyber Pakhtunkhwa, Pakistan.

ABSTRACT

In the current era of rapid growth in Internet of Things (IoT) devices and the demand for low power, high efficiency computation solutions in edge computing have has encouraged research efforts in alternative arithmetic suitable for resource-constrained environments. This paper investigates the implementation in fixed-point arithmetic over edge devices to achieve more computational efficiency and comparatively less power consumption compared to floating-point arithmetic. Experiments were performed on three different hardware platforms which are PIC32MZ2048EFG, Node MCU 8266 and Raspberry Pi zero. The findings show that fixed-point arithmetic consumes 50% less energy and executes 66% faster in terms of processing speed with minor accuracy trade-offs. The paper further demonstrates that two or threefold reductions in training time and memory usage can be achieved at the expense of acceptable accuracy loss when examining the performance of MNIST dataset neural network models in fixed-point versus floating point computation. These findings of this paper highlight a significant lower cost on edge computing is reached via fixed-point instead of floating-point that allows AI deployment on devices with limited computation and storage resources.

Keywords: Edge Computing, Floating-Point Arithmetic, Fixed-Point Arithmetic, Neural Network Models, AI.

I. INTRODUCTION

Edge computing has proved to be an instrumental element in solving the challenges with processing and sending large amounts of data over long distances as it moves closer towards the edge of a network, where data is generated. This transition helps decrease latency, save bandwidth and enable fast decision-making for various Internet of Things (IoT) devices and embedded systems, particularly real-time applications [1]. On the other hand, due to power limits, memory costs and processing constraints of edge devices, hardware friendly arithmetic methods are also required to handle workloads.

Floating-point is a common form of arithmetic in many computing applications, however it has limitations for edge devices due to its computational complexity and high power consumption. On the contrary, using fixed point simplifies the calculations due to its use of limited decimal places offers solution for low-power applications reducing energy requirements at the cost of decreased numerical precision. While prior work has shown that fixed-point formats can improve resource utilization and performance in areas such as signal processing and machine learning [2], a general comparison of the strengths and weaknesses of fixed-point across many different embedded applications has not been performed.

The objective of this work is to present an extensive comparison between floating-point and fixed-point arithmetic performed in vector operations as well as in Neural Network inference through a sequence of experiments carried out on three different hardware platform PIC32MZ2048EFG, Node MCU 8266 and Raspberry Pi Zero and neural network models respectively. Floating-point versus fixed-point arithmetic affects speed, power consumption and memory usage but also accuracy. This will help by resolving the appropriate trade-offs between each arithmetic format, thus helping determine if a fixed-point method can be used to deploy AI models for devices that are constrained to very low-power and resource-constrained hardware.

II. LITERATURE REVIEW

The 1990s was the era of computing based on single or multiple CPU-based cores and companies like WinTel scaled its CPU centric architectures according to Moore law scaling rows of CPU cores and RAM to boost performance [3]. These CPUs improvements were great but they still fell short for the requirements of parallel processing needed in AI applications. The introduction of GPUs, initially developed for rendering graphics opened up a strong alternative with thousands of cores capable of executing parallel computations required in deep learning [4]. This transition was accelerated even more with NVIDIA development of CUDA programming language that optimized GPU for AI tasks [4]. Latest GPUs along with new plug and play accelerators such as Intel® Movidius™ Neural Compute Stick, Laceli AI compute sticks from Gyrfalcon Technology and Raspberry Pi

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

Model B are now the heart of high-performance AI infrastructure [5][6]. Even with these hardware improvements, we still need better algorithms and methods to make proper use of existing computational power.

Neural network training can benefit from both algorithmic and hardware optimizations to fully maximize performance. Some of these Algorithmic optimizations are like AdaGrad which controls the learning rates based on previous gradients and RMSProp which adjusts the learning rate based on recent gradient information to reduce excessive fluctuations [8][9]. Another popular optimization method for the model tuning problem are population based optimization methods which simulate evolutionary processes [10]. Methods such as fixed-point arithmetic are used to facilitate calculation by uniformly using a constant number of bits for representation (real value), which speeds up the process and requires less memory that can always be advantageous in embedded systems.

In traditional computing environments, which are often cloud and central data-center based, systems have predominantly favored floating-point used for the precision needed for diverse numerical ranges have higher computational complexity and power consumption that is not suitable for an edge device. But edge devices usually work under strict power and resource constraints which makes them consider alternative arithmetic. On the other hand, fixed-point arithmetic uses a much simpler and faster method that preserves a fix number of digits both before and after the decimal point. This approach minimizes the processing load and energy consumption with a minimal accuracy loss as a trade-off.

Edge computing reduces reliance on centralized networks by processing data on device, which can reduce latency and bandwidth issues [1]. Through this strategy, latency is minimized which is particularly important for real-time applications such as autonomous vehicles and medical monitoring [10]. Edge computing minimizes network traffic by sending only data that has been processed at the source itself and thus ease congestion much-needed for scaling systems to be cheaper. That improves the privacy as well by reducing data moving across networks especially useful in use cases like smart home and industrial IOT.

Upcoming trends in edge computing focus on how modern infrastructure relies more on edge devices, lowering latency and increasing efficiency with applications such as smart cities, connected vehicles and smart grids. AI models are deployable on low-power devices via optimization techniques such as using fixed-point arithmetic, model compression and efficient inference algorithms [16]. Fixed-point optimization is being widely used in domains such as computer vision and NLP etc, making AI suitable for resource-constrained devices [11]. The deployment of machine learning in low-power circumstances by TinyML has turned fixed-point optimization into a key approach [12]. The development of hardware accelerators like Google's TPUs and AI chips for edge computing such as Apple's Neural Engine and Huawei's Ascend AI processor have motivated the need for fixed point arithmetic [13][14][15].

Running neural networks at the edge has brought to light an important tradeoff between model accuracy and computational efficiency. Recent years have seen the adoption of fixed-point arithmetic for neural network inference as it requires less memory and computation. Quantized models have been shown to operate quite well on embedded hardware with acceptable accuracy loss while using small-fixed-point representation. Hence, the fixed-point formats are widely used for performing AI models under a resource constraint setup.

III. METHODOLOGY

The experiments conducted in this study focused on evaluating the performance of floating-point and fixed-point arithmetic using different embedded hardware platforms and neural network models. This section provides an overview of the hardware specifications, software setup and experimental procedures used to measure computational speed, power efficiency and accuracy across different arithmetic formats.

Hardware Platforms

Three embedded hardware platforms were selected to cover a range of computational capabilities and hardware constraints:

1. **PIC32MZ2048EFG Microcontroller**

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

Table 1: Specifications of PIC32MZ2048EFG Microcontroller

Specification	Details
Name	PIC32MZ2048EFG
Bus	Peripheral: SPI, I2C, UART, CAN | System Bus: AHB
Memory	Program Flash: 2 MB, RAM: 512 KB
Processor	MIPS M14K core, 200 MHz, 32-bit, No hardware FPU
Display	No dedicated display controller, SPI/Parallel for external display
Generic Features	144-pin, 134 GPIOs, Multiple Timers, 48-channel ADC, Integrated Ethernet, Low power
Storage	No dedicated storage beyond program flash
Other Details	PWM channels, USB 2.0, High-performance embedded systems

2. Node MCU 8266:

Table 2: Specifications for Node MCU 8266

Specification	Details
Name	Node MCU 8266
Bus	Peripheral: SPI, I2C, UART | System Bus: Internal memory bus
Memory	Flash Memory: 4 MB, RAM: 64 KB instruction, 96 KB data
Processor	Tensilica Xtensa L106, 80 MHz (160 MHz possible), 32-bit
Display	No dedicated display, External via GPIO (SPI/I2C)
Generic Features	11 GPIOs, Built-in Wi-Fi, Single 10-bit ADC, Power-efficient
Storage	Flash for code and data, No expandable storage
Other Details	PWM output, USB-to-Serial, Full TCP/IP stack, IoT applications

3. Raspberry Pi Zero

Table 3: Specifications for Raspberry Pi Zero

Specification	Details
Name	Raspberry Pi Zero
Bus	Peripheral: SPI, I2C, UART, USB OTG | System Bus: AMBA
Memory	RAM: 512 MB LPDDR2
Processor	Broadcom BCM2835, 1 GHz ARM11 core, ARMv6, Integrated FPU
Display	Mini HDMI port (1080p), DSI display interface
Generic Features	40 GPIOs, HDMI and Composite video, Wireless (Zero W), CSI camera
Storage	MicroSD card slot for booting and storage
Other Details	USB OTG, Runs Linux-based OS, Integrated GPU for 1080p

Software Environment

The experiments implemented a combination of programming languages and development tools based on the particular hardware platform:

- **Programming Languages:** C++ was used for low-level hardware interfacing to ensure optimal performance for operations. Python was adopted for building and testing neural network-based models.
- **Development Tools:** Microchip MPLAB X IDE: In the experiment, the PIC32MZ2048EFG microcontroller was employed, programmed in a Microchip MPLAB X IDE environment using an XC32 Compiler. Once the

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

code was compiled, it was loaded into the microcontroller through a PICKit method. Since UART was implemented for data logging, it was possible to store and evaluate the outcomes in the future. Embedded timers within MPLAB were deployed for the purpose of taking time measurements, for instance ensuring that correct time was taken for vector addition. In the course of the experiment, a very accurate real-time power monitoring strategy was developed by attaching an INA219 Current Sensor to the power input of the microcontroller for power consumption measurements.

- **Arduino IDE:** the NodeMCU 8266 was configured as an ESP8266 board in the Arduino IDE. The connection of the Node MCU via micro-USB facilitated the uploading of C++ code which streamlined the development process. Results in real time could be viewed through the Serial Monitor of the IDE. The execution time could be measured using the millis() function without causing latency in the microcontroller as it timed the elapsed time at an appropriate rate. Changes in the code did not affect power consumption, logged by a DROK USB Power Meter placed between the power supply and the NodeMCU; thus, the monitoring of power remained unaffected by the computing activities.
- **GNU Compiler Collection (GCC):** For the purpose of compiling C++ programs on the Raspberry Pi Zero, it was accessed remotely using SSH or terminal. Detailed data gathering was performed by logging the output on the console directly to perform an analysis of the implementation. The duration of the performance was measured via the C++ chrono library which provided clear specification on the timing of carrying out the vector operations. Power draw was determined with the help of an external UM25C USB power meter positioned in-line between the power supply and the Raspberry Pi Zero to provide a clean and clear power draw measurement unobstructed by other processing activities.
- **TensorFlow and TensorFlow Lite** for neural network experiments with additional quantization tools to facilitate fixed-point model deployment.

Experimental Design

The experimental design was divided into two major parts as follows; vector operations and neural network inference. Each component sought to evaluate the performance and accuracy of floating-point and fixed-point arithmetic in different data types, tasks and scenarios.

1. Vector Operations

The objective of the vector operations experiments is to evaluate the effect of various arithmetic representation formats on computational speed, power consumption and numerical accuracy. Vector addition was performed using five different data types which includes: double (64-bits precision floating point), float (32-bits precision floating point), fixed int (16 bits integer & 16 bits fraction), fixed short (8 bits integer & 8 bits fraction) and fixed char (4 bits integer & 4 bits fraction). In each operation, the number of elements was 10,000 and the operations were timed in milliseconds using internal timers affixed on the respective platforms. Power data was captured in milliwatts using external USB power meters to capture energy usage in real time during computations. Computational accuracy of fixed point arithmetic was investigated against its double precision floating points and variances in percentage of error were recorded for each numerical datatype.

2. Neural Network Inference

The objective of these experiments is to examine and investigate the effectiveness of floating-point and fixed-point formats and what computation advantages and limitations can be observed in neural network inference tasks. A convolutional neural network (CNN) was developed based on the MNIST dataset, which contained training and 10,000 testing images of handwritten digits. The dataset was preprocessed and normalized for both floating-point and fixed-point experiments to ensure consistency. Floating-point model was developed incorporating the use of 64 bit precision. However, fixed point model was further quantized to 8 bit using full integer quantization of TensorFlow lite. Performance metrics included training time, validation loss, test accuracy and memory consumption.

IV. RESULTS AND ANALYSES

This section provides a thorough evaluation of the results of the experiments conducted with particular emphasis on the performance differences associated with computational speed, power consumption and

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

accuracy when using floating point and fixed point arithmetic. The discussion highlights the implications of these findings for embedded systems and edge computing.

Vector Operations

The results of the experiments are displayed in the format of bar charts that present a comparative analysis of the performance of the floating and fixed point arithmetic on the PIC32MZ2048EFG, NodeMCU 8266 and Raspberry Pi Zero after performing 10,000 operations in such aspects as vector addition time, power consumption and error percentage for different data types. Where floating point variables are colored blue while fixed point variables are colored orange.

Final Values

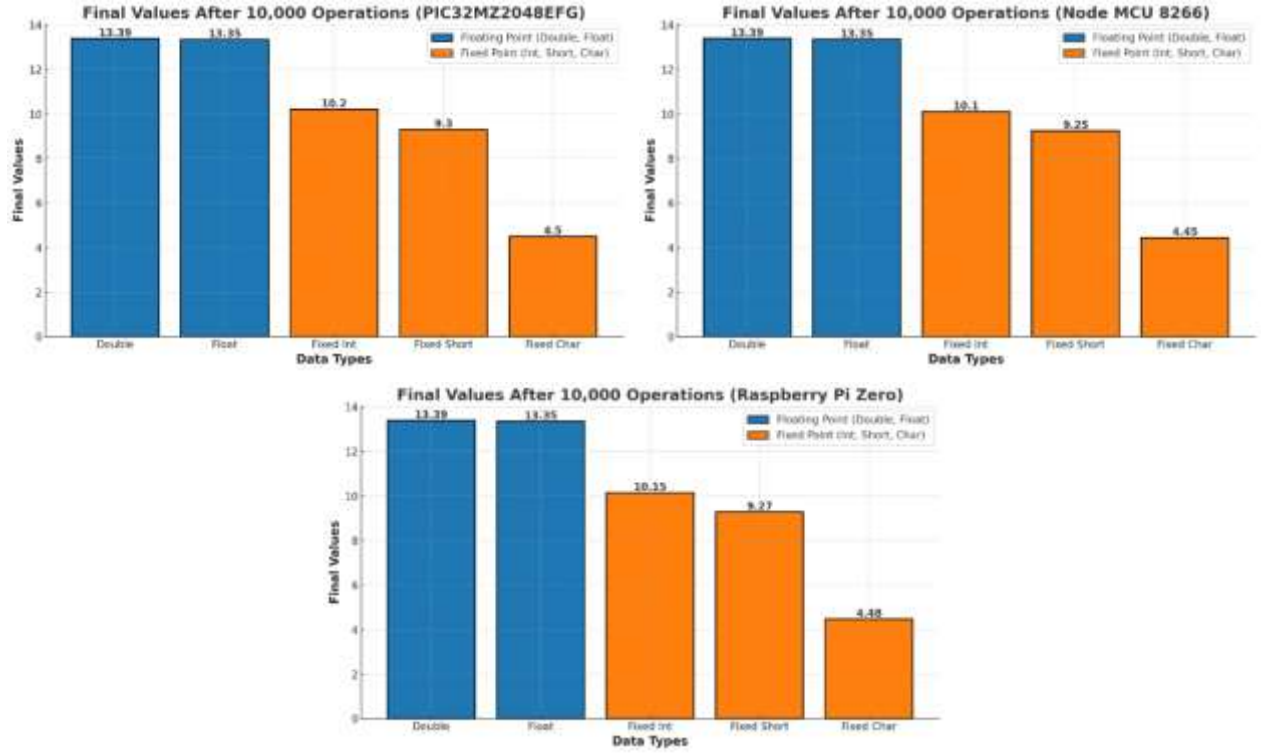
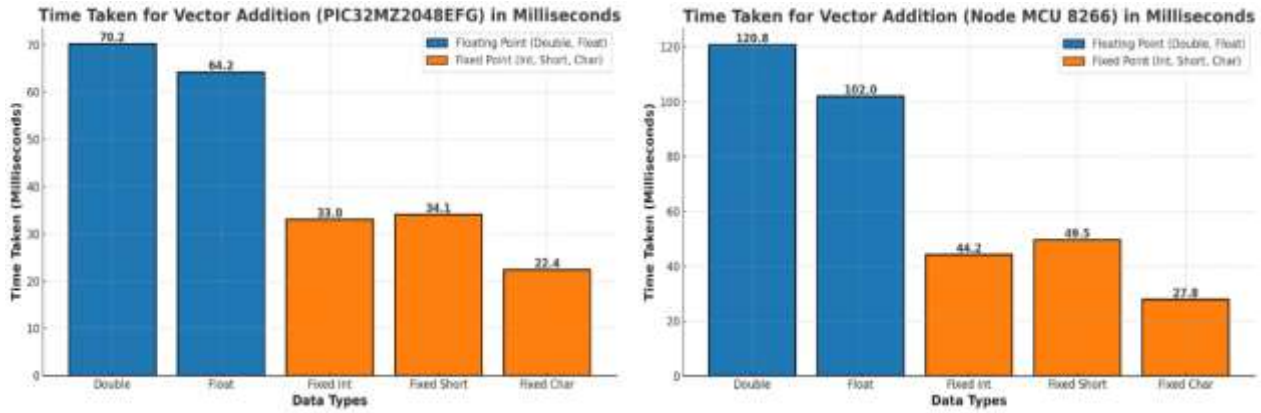


Figure 1, 2 & 3:

The outcomes illustrated in Figure 1, 2 & 3 clearly illustrate that fixed-point arithmetic significantly improves time efficiency in comparison to floating-point arithmetic across all three platforms. Efficiency improvements range from 55.6% to 71.8% for fixed-point data types. In embedded systems where both computational speed and energy efficiency are essential, fixed-point arithmetic is the preferable choice over floating-point arithmetic.

Computational Speed:



e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

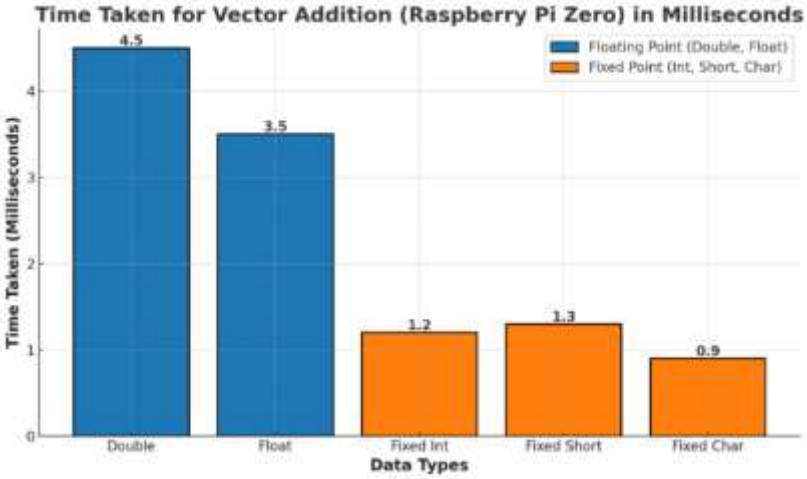


Figure 4, 5 & 6:

The outcomes in figure 4, 5 & 6 demonstrate that fixed-point arithmetic significantly improves time efficiency in comparison to floating-point arithmetic across all three platforms. Efficiency improvements range from 55.6% to 71.8% for fixed-point data types. In embedded systems where both computational speed and energy efficiency are essential, fixed-point arithmetic is the preferable choice over floating-point arithmetic.

Power Consumption:

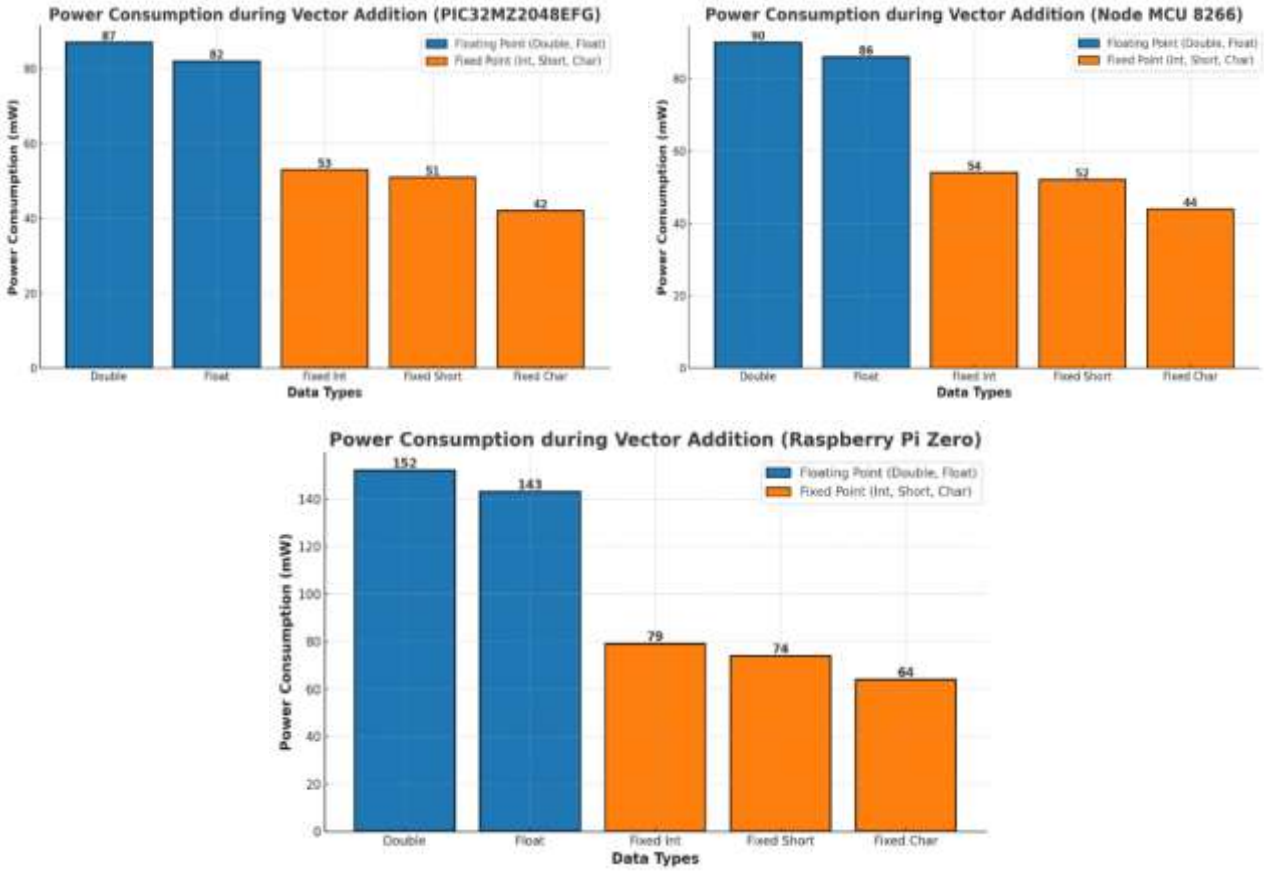


Figure 7, 8 & 9:

The results in figure 7, 8 & 9 demonstrate that fixed point arithmetic offers considerable energy efficiency benefits over floating point arithmetic for all three platforms studied. The power consumption for fixed point types as opposed to floating point types falls within a range of 40% to 50%. Such a notable reduction in power consumption illustrates how efficient the fixed point arithmetic is.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

Precision:

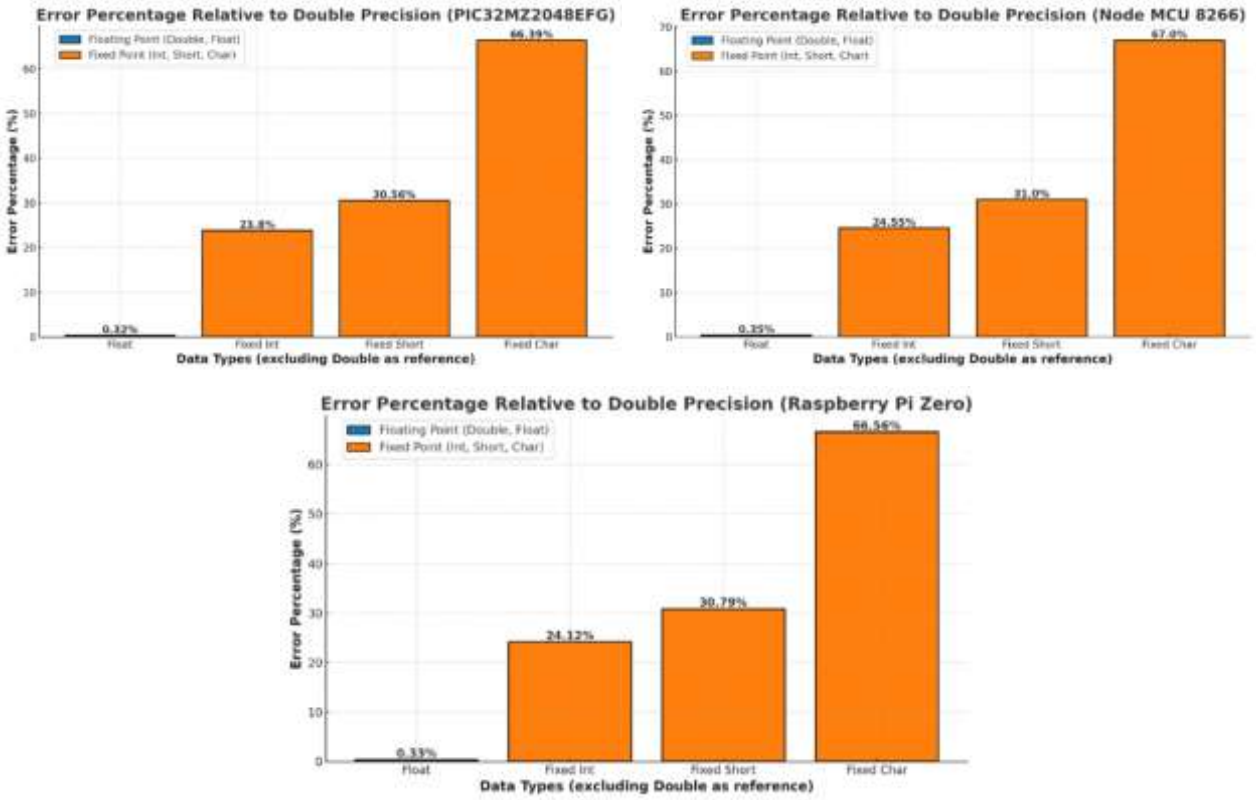


Figure 10, 11 & 12:

The outcomes in figure 10, 11 & 12 show the average estimated error in percentage difference between floating-point and fixed-point variables is observed from 23% to 66%. Upon looking at the percentages of the errors, it can be noted that the floating point types exhibit the least double errors across all platforms though this is the only range where it has the highest accuracy. A similar examination using fixed-point types involves significantly higher error percentages, which portrays the cost of precision in speed. Floating point types are the most suitable in cases that require high level of precision. On the other hand, fixed-point types have the advantage of improved speed of processing but this is at the expense of a reduction in accuracy.

Neural Network Inference: Accuracy and Resource Efficiency

A comparative study of neural network training with floating-point and fixed-point formats indicates that fixed-point arithmetic offers enhanced resource efficiency, achieving performance in accuracy that remains largely equivalent to that of floating-point arithmetic. The corresponding bar charts are presented below where bars in blue color represent floating point variable (64-bit) and bars in orange color represent fixed point variable (8-bit).

Training Time:

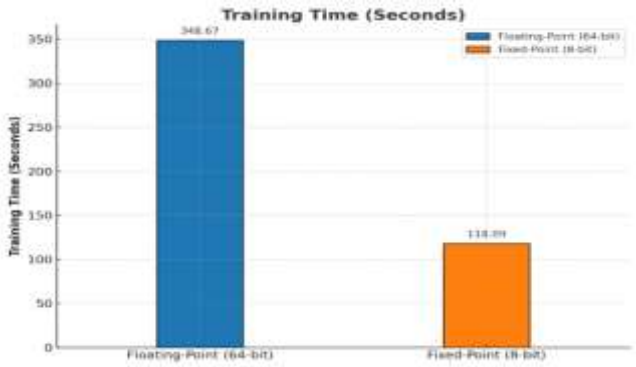


Figure 13:

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

The figure 13 presents training time associated with the 64-bit floating-point model relative to the 8-bit fixed-point model. The floating-point model completed the process in 348.67 seconds, whereas the fixed-point model achieved this in just 118.09 seconds. This represents a significant reduction of 66% in training time for the fixed-point model.

Memory Usage:

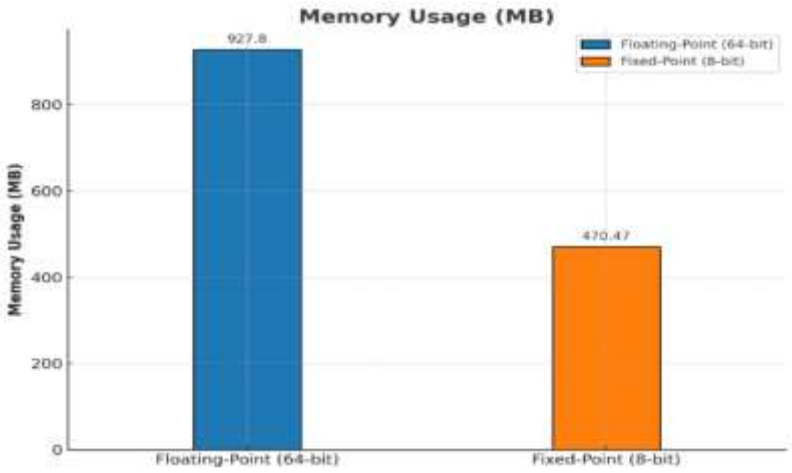


Figure 14

Figure 14 shows the efficiency of memory usage realized with the 64bit floating-point model in comparison to the 8-bit fixed point model. It can be seen that the fixed point model is much more efficient than its floating-point counterpart. 925.80 MB of memory is occupied by the floating-point model but only 470.47 MB by the fixed-point model, which results in almost 50% memory space savings. This is due to the use of fixed-point model as opposed to floating-point model.

Validation Loss and Test Loss:

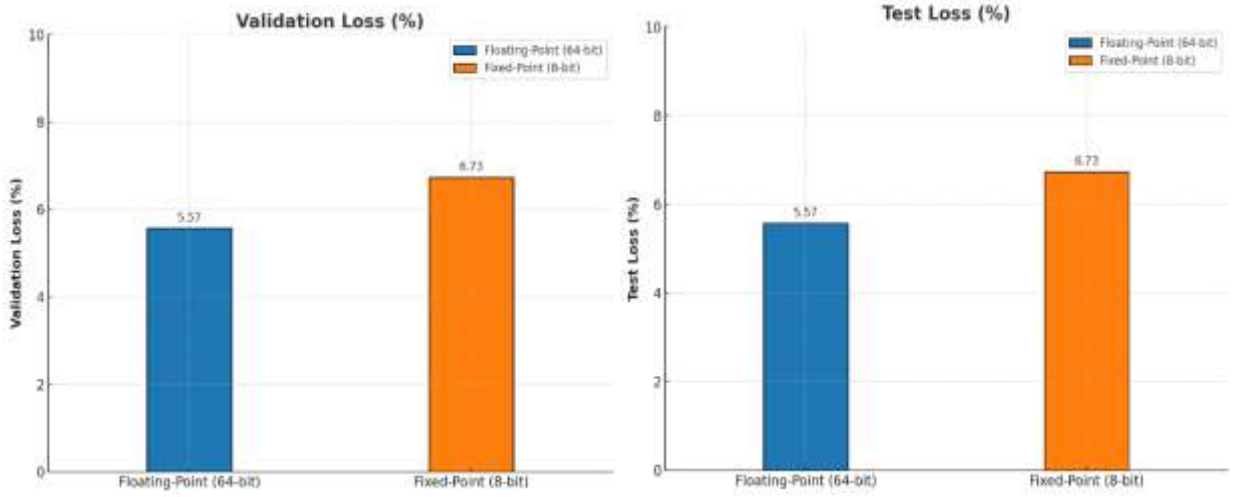


Figure 15 & 16:

The figure 15 presents validation loss and figure 16 presents test loss of the 64-bit floating-point and the 8-bit fixed-point models. A comparison of the validation and test losses between the two models indicates that the floating-point model achieved a validation loss and test loss of 5.57%. In contrast, the fixed-point model recorded slightly higher losses, with both validation and test losses at 6.73%. This results in a relative increase in losses of 1.16% when fixed-point arithmetic is used.

The figure 17 presents test accuracy of the 64-bit floating-point and the 8-bit fixed-point models. It was shown that floating-point model achieved a test accuracy of 98.24% while fixed-point achieved slightly lower accuracy of 98.19%. This drop in accuracy of 0.05% is negligible, which further explains why fixed-point arithmetic can be operated on with reduced precision, yet still able to maintain quite a high degree of accuracy.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

Test Accuracy:

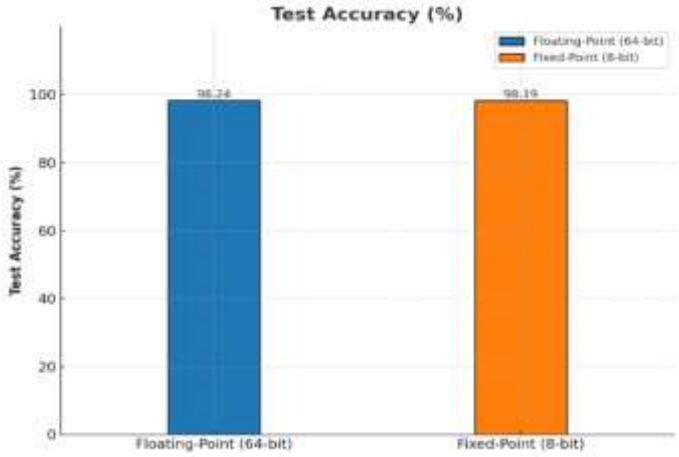


Figure 17:

V. DISCUSSION

The experimental results highlight advantages and limitations of fixed-point versus floating point arithmetic in edge devices and embedded systems. These studies clearly demonstrate that using fixed-point arithmetic can considerably accelerate computation and minimizing power consumption, thus supporting its use in resource-constrained environments.

The results highlight a critical trade-off: fixed-point computation provides substantial speed and power advantages compared to its floating-point counterpart but introduces precision loss. This loss in precision was not highly influential in the neural network inference processes which indicated how easily these fixed-point formations can be introduced to edge devices.

VI. CONCLUSION

The study proves the feasibility of fixed-point algorithm as a resource efficient and faster method than floating-point algorithm across edge devices. The experiments and results provide a comprehensive comparison of the two arithmetic methods. Fixed point operations consistently outperformed floating-point operations, achieving processing time reductions of up to 66% and decrease power-consumption of up to 50% while achieving performance nearly equivalent to those of floating-point operations with significantly reduced resource requirements, validating its suitability for low-power AI applications. In the case of neural network models, fixed-point arithmetic can efficiently shrink memory usage and training time, therefore it seems like an appropriate alternative on embedded devices. This advantage is also relevant in edge AI applications that require light-weight models for real-time processing.

VII. FUTURE WORK

Future studies may explore mixed-precision approaches in which combination of floating-point and fixed-point operations are used within the same processing unit. This hybrid approach may provide an optimal compromise between accuracy and efficiency for targeted accuracy tasks. Investigating advanced quantization techniques for neural networks can boost computational benefits and minimize accuracy loss of fixed-point arithmetic. Techniques such as dynamic quantization and post-training calibration should be considered and explored, both of which may improve the efficacy of artificial intelligence models on edge devices. Further research can extend the benchmarking to other domains e.g. audio processing, control systems and sensor networks to give proof of the general validity of the fixed-point arithmetic across a wide range of applications. Creating a standardized toolkit or library that reduces the difficulty of implementing fixed-point arithmetic could help accelerate its use by researchers and developers. These tools may consist of automatic quantization algorithms and accuracy estimators to allow easy conversion from a floating-point format to a fixed-point format. Testing in the field in applications such as healthcare, agriculture and industrial automation would serve practical purposes to bring to light long-term advantages and drawbacks related to fixed-point

arithmetic. These works may concern the stability of fixed-point models in changing environmental conditions and computational requirements.

These results indicate that fixed-point arithmetic is a reliable alternative for improving the energy-efficiency of edge devices and embedded processing systems. The findings confirm the utility of fixed-point arithmetic for a cross-section of edge computing applications, from AI inference to low-level data processing, allowing more intelligent and energy-efficient IoT devices.

VIII. REFERENCES

[1] J.-H. Hub and Y.-S. Seo, "Understanding edge computing engineering evolution with artificial intelligence," IEEE Access, vol. 7, pp. 164229-164245, 2019

[2] Youngmin Kim, et al. "FPGA-Based Convolutional Neural Network Accelerator with Resource-Optimized Approximate Multiply-Accumulate Unit", 2021

[3] B. Metcalfe and K. Foster, "Winning with Wintel business strategies," IEEE Spectrum, vol. 34, no. 5, pp. 7-12, May 1997.

[4] J. MSV, "In the era of artificial intelligence, GPUs are the new CPUs," Aug. 2017. Available https://www.forbes.com/sites/janakirammsv/2017/08/07/in-the-era-of-artificial-intelligence-gpus-are-the-new-cpus/#2fd98b0a5d16.

[5] J. Hochstetler, R. Padidela, Q. Chen, Q. Yang, and S. Fu, "Embedded deep learning for vehicular edge computing," in 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2018, pp. 341-343.

[6] "Gyrfalcon ships AI chip to Samsung, LG." Dec. 2018. Available https://www.gyrfalcontech.ai/news/gyrfalcon-ships-ai-chip-to-samsung-lg/

[7] E. Dogo, O. Afolabi, N. Nwulu, B. Twala, and C. Aigbavboa, "A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks," in 2018 International Conference on Computational Techniques, Electronics, and Mechanical Systems (CTEMS), IEEE, 2018, pp. 92-99.

[8] S. Ruder, "An overview of gradient descent optimization algorithms," Mar. 2020. Available https://ruder.io/optimizing-gradient-descent/

[9] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," Mar. 2017.

[10] M. Nadimpalli, "Artificial intelligence risks and benefits," International Journal of Innovative Research in Science, Engineering and Technology, vol. 6, no. 6, 2017.

[11] Feiyang Chen et al., "Comprehensive Survey of Model Compression and Speed Up for Vision Transformers", 2024.

[12] P. P. Ray, "A Review on TinyML State-of-the-Art and Prospects."

[13] N. P. Jouppi, et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," 2017.

[14] Apple Neural Engine An Overview, Apple Internal Publications.

[15] Huawei Ascend AI Processors for the Era of Intelligent Computing, Huawei Internal Research Teams.

[16] Wenxiao Wang, et al., "Model Compression and Efficient Inference for Large Language Models: A Survey", 2024.