MACHINE LEARNING-BASED DYNAMIC VOLTAGE AND FREQUENCY SCALING ERROR DETECTION

An Undergraduate Research Scholars Thesis

by

JOHN C. MUSCHINSKE

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Faculty Research Advisors:	Dr. Jiang Hu Dr. Paul Gratz

May 2022

Major:	Electrical Engineering

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, John C. Muschinske, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisors prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Machine Learning-Based Dynamic Voltage and Frequency Scaling Error Detection

John C. Muschinske
Department of Electrical and Computer Engineering
Texas A&M University

Research Faculty Advisor: Dr. Jiang Hu, Dr. Paul Gratz
Department of Electrical and Computer Engineering
Texas A&M University

Modern microprocessors are more and more optimized for speed and power efficiency. A system that regulates these parameters is Dynamic Voltage and Frequency Scaling (DVFS). Generally, all bugs or errors in a microarchitecture fall under two categories: performance and logical. These errors can apply to any component of the microarchitecture. A performance error is an error that results not in a logically incorrect output of a system, but a slowdown in the production of that output. Most broadly, errors related to DVFS would be not increasing voltage and frequency leading to slower execution in real time, or the inverse (increasing voltage and frequency) leading to wasteful power consumption and chip degradation. The first slows down the machine unnecessarily, and the second decreases expected battery time. Using machine learning to analyze data extracted from gem5 to detect these errors is the objective of this project.

DEDICATION

To my Advisors, for their direction, concepts, and guidance. To my Family, my mother, Janet for support in communication. To my father, Steve for being there when nothing was working and I thought this would go nowhere. To my aunt, Susan for advice on writing along the way. To the rest of the group, for help along the way.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisors, Dr. Jiang Hu and Dr. Paul Gratz, and the remainder of the group Chrysanthos Peipei and Dr. Erick Barboza, for their guidance and support throughout the course of this research.

Thanks also to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to my parents and aunt for their encouragement and for reading all of my roughest of drafts.

The materials and models used for Machine Learning-Based Dynamic Voltage and Frequency Scaling Error Detection were provided by the Computer Engineering and Systems group . The analyses depicted in Machine Learning-Based Dynamic Voltage and Frequency Scaling Error Detection were conducted in part by Dr. Jiang Hu and Dr. Paul Gratz and these data are unpublished.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Undergraduate research was supported by the Electrical and Computer Engineering Department at Texas A&M University.

No external funding was required.

NOMENCLATURE

ALU	Arithmetic Logic Unit
ANN	Artificial Neural Network
ARM	A RISC instruction set architecture developed and licensed by ARM LTD.
B/CS	Bryan and College Station
CNN	Convolusional Neural Network
CPU	Central Processing Unit
DVFS	Dynamic Voltage and Frequency Scaling
gem5	Computer-system architecture and processor microarchitecture simulator
GPU	Graphics Processing Unit
IPC	Instructions Per Cycle
IR	Image Recognition
ISA	Instruction Set Architecture
ML	Machine Learning
P-state	Processor Power Performance State
PID	Proportional Integral Derivative
RISC	Microprocessor architecture that utilizes a small, highly-optimized set of instructions rather than the highly-specialized set of instructions
ROC	Receiver Operator Characteristics
SimPoint	Simulation Point
SPEC	Simulation Specification
TAMU	Texas A&M University
V-F pair	Voltage Frequency Value Pair

1. INTRODUCTION

1.1 Overview of Dynamic Voltage and Frequency Scaling

Modern computing systems have come to rely upon various power systems. This is made inevitable by both increasing miniaturization and base clock rates, and the desire to produce a long battery life and reasonable power draw. In extreme cases where raw performance is concerned, the heat produced to power these computers further increases the power draw to provide cooling. To remedy this, systems commonly called Dynamic Voltage and Frequency Scaling (DVFS) have been gaining prevalence. A DVFS system controls which frequency/voltage state pairs [1] a computational system such as GPUs or CPUs (also called cores) operating parameters. As time has advanced, DVFS systems have also gained complexity, ranging from proportional integral derivative (PID) based control systems to Artificial Neural Network (ANN) based systems [2] depending on modernity and designer.

Depending on the driver, there commonly exist between 0 and 255 individual voltage-frequency states. That is 0-255 voltage operating points with both a software (driver) side control scheme and a hardware (core) side system, and an identical number of frequency operating states, also divided between driver and core. A pair of voltage and frequency can be called a power state or P state or V-F operating point [1] , among several other analogous names. The software side system is common to the driver used and is independent of the microarchitecture, while the hardware side system is the opposite, usually changing chip to chip. Only one of the two systems, driver or core, can control the power state transition of the core in question at any time.

In the context of this work, only the software/driver-based system is observed due to simulation model constraints. To finish and clarify the explanation of the single core mode in the software side control scheme, there can be at most 255 V-F pairs. Adding multiple cores to the power management scheme increases DVFS complexity significantly. For instance, when multiple cores are introduced, modern DVFS schemes often incorporate the throttling down of cores not actively working on the main load of a multithreaded system. An example of this is Intel's Turbo

Boost.

This innate and increasing complexity in the DVFS system leads to the motivation of this work. Because performance errors are hard to localize or even notice, it is valuable to provide the capability to confirm the presence of this type of error rapidly and accurately. Thus, the goal herein is to find a method to determine mostly autonomously with Machine Learning (ML) if there is a performance error in a DVFS system.

1.2 Overview of Performance v. Functional Errors

To differentiate between functional errors and performance errors a functional or logical error is, in a broader sense, when a system somehow fails to execute its function. An example of this would be that an Arithmetic Logic Unit (ALU) does not properly execute the AND or OR operations due to some faulty bus connection. Generally, these errors are more detectable due to a difference in expected output and actual output. Commercial software as well as humans tend to be more proficient in detecting this type of error. That is not to trivialize this type of error, as functional errors can still be difficult to detect.

Conversely performance errors are more difficult to detect. A performance error is a fault in the design of a system such that the expected output is still reached; however, various performance parameters, such as power consumed or Instructions Per Cycle (IPC), are negatively affected. Some prior analysis exists on performance error detection and localization [3], [4], [5], [6], [7]. However, the focus is again on IPC across cores as well as non-DVFS performance improvement. In the context of DVFS an example of this would be over-stability, that is the DVFS system does not switch states when it would be power/performance efficient to do so. To match the more ephemeral nature of the errors, humans and commercial software tend to have a harder time detecting performance errors.

An example of a performance error would be the snoop filter eviction error [8]. It took several months to detect and remove this error, which had occasionally caused a greater than 10% loss in performance. This finding once again suggests that first detecting these performance errors, and then being able to localize them would be useful.

In the context of DVFS these errors would, in general, lead to increased energy consumption or run time. The source of these as hypothesized throughout this work is in the nature of how a DVFS system changes states, to be mentioned in the methods section. A non-DVFS related, and possibly clearer example, would be a scheduling error in an out of order scheduler. Suppose that the scheduler for a specific command would in certain circumstances continuously over-prioritize the execution of that command, such as OR, while other commands, such as ADD, would be delayed in execution. Eventually this can accumulate in a section of idle time due to some aspect of the test program requiring the execution of inefficient scheduling, resulting in a drop of IPC.

The takeaway is that while a performance error rarely leads to a failure of the overall core, they still lead to a drop in the value of a product, all while being more abstract and harder to detect than logical errors.

2. BACKGROUND

Performance error debugging focused on IPC and register-based issues in Barboza et. al. [3], where the concept was to measure datapoints with several probes, and develop an ML model for each probe. Then, this set of models was used to detect performance errors in the core. While doing this Barboza et. al. [3], directly implemented performance errors/bugs that could be used through using a variety of SPEC2006 SimPoints [7] across several x86 Intel microarchitectures.

Older works on performance debugging, such as Surya et. al. [4], refer to a set of performance bugs/errors called timer/timing errors. Here the objective is to classify the behavior of such errors in IPC/pipelining sense. To do so, Surya et. al. develops functions to evaluate timing errors, and by finding and stating these errors are removable, conclude that by limiting timing errors, so to are future functional errors limited.

While ML has been applied to the debug space of design, it has even more so been applied to design problems, as demonstrated by the depth of spaces described in [9]. These applications are predominantly in any method that has to do with decision based or prediction based systems, including DVFS and schedulers systems. Additionally, there are the various forms of machine learning. Most relevant to this and works in automated performance error detection are supervised learning methods. This is a type of method that uses known inputs and their outputs to predict the output, or in this case, presence of a performance error. The opposite of supervised is unsupervised, where inputs are provided without expected outputs. Instead, the ML system clusters them in a provided or unprovided number of outputs based on similarity. Semi-Supervised uses a mix of labeled supervised inputs and unlabeled inputs, this allows for easier access to training data. Lastly, is reinforcement learning, where a desired result is given an incentive and the system optimizes itself for that incentive. All of these have potential uses in microarchitecture design, though supervised learning and reinforcement learning have seen the most results.

3. DESIGN

3.1 Resources

3.1.1 gem5

To simulate microarchitectures gem5 [10] will be used. The gem5 simulator is a commonly used platform for computer architecture research. The gem5 simulator has two modes of simulation, full system (FS) and syscall emulation (SE). Full system mode fully simulates the virtual machine and all parts of the operating system, while having higher accuracy to real world systems. Conversely, syscall emulation mode is much faster at simulating but sacrifices fidelity to do so. More precisely, FS mode simulates a physical CPU, and if an offshoot of gem5, gem5-gpu [11] is used, GPUs can be simulated. Consequently, FS mode is more accurate to an actual system, with the operating system being simulated, while syscall emulation mode only simulates the behaviors of the CPU caches and associated systems, and leaves the opperating system to the host machine. Due to the constraints of simulation in gem5 [10], full system ARM simulation is used. This is because there is only a pre-established DVFS simulation option configuration for full system ARM based Instruction Set Architectures (ISAs) for gem5. Further, due to the design of gem5 and the core models available, only the driver based DVFS system has been simulated for the context of this paper, as well as a focus on a single core CPU. For future work, GPUs also use a DVFS like system and by nature are more relevant in the single core case.

The rationale behind the proposed is that should performance errors be detectable from extracted data from the driver based DVFS system, they could be similarly detected from data extracted from core based DVFS systems. To ensure variety and validity of extracted data, various SimPoints [7], mostly extracted from SPEC2006 [12], are used. A SimPoint is a representative collection of instructions from a benchmark application (or several applications) such that the results of running the overall benchmark can be inferred. The benchmark's SimPoints that will be used are those of a SPEC, a collection of applications/programs/executables that are an exhaustive test of an architecture. Upon the completion of a simulation run, various output parameters are received from gem5. Most notable for the case of this paper are cycle time, energy consumed, and cycles in different power states.

Observationally, gem5 implements the simulation of the simulated machine through the following aspects. First, there is the design and layout of the simulation using C code. This is mostly under the hood and handles all the actual implementation of the machine emulation. Second, there are the Python configurations, which handle the input/output and actual structure of the emulated machine. This is usually transformed into C code to operate more quickly. Third, there are the inputs which are the voltage ranges, the modes, the disk image with the necessary SimPoint [7] loaded on it, and the commands to be executed for each disk, comprising a period of basic Linux list commands to make sure the runscript is loaded. Then run the command for the SimPoint contained in the bootscript file. Further, to be as stringent as possible, the only thing loaded on these disks is an identical base file system and the SimPoint to be evaluated in the data gathering run.

3.1.2 ARM

The version of ARM used is arm64. There are ultimately two reasons ARM is used in this line of experiments. First, gem5 only does DVFS with ARM ISA based architectures. It would be possible, but prohibitively time consuming, to try to adapt what exists to the x86 ISA. Further, the ARM chip models are documented and usable under license. ARM chip models have a base ISA similar to Apple Corporation's A series CPUs found in iPhones and iPads. Intel's x86 ISA was not chosen because the models are proprietary and therefore more difficult to attain models for. The ARM architecture should provide ample evidence that the approach can be applied to any other architecture.

To be further detailed, arm64 or aarch64 is a 64 bit Reduced Instruction Set Architecture (RISK). This should not be directly applicable to the metrics used in this work because no instruction related performance errors were used. However, in the process of future work on this research the instruction queue issued and the cone of influence of those instructions are likely to become critical in localization of a performance error. This is because if there can be one period in the set of datamaps that is indicative of a performance error, it shows what period of prior operations could have influenced the occurrence of the negative behavior. An alternative case is that there is no correlation between instructions issued, but instead the DVFS system establishes a poorly distributed array of accessible V-F pairs at start of run, which can be identified as a performance error.

3.1.3 Python and ResNet-50

ResNet-50 is a convolutional neural network (CNN) that is 50 layers deep. This research uses PyTorch library containing a pre-trained network in the TorchVision module for Image Classification. The TorchVision package contains common image transforms for computer vision.

The datasets for the input to gem5 and transforming gem5 output for input to ResNet-50 are written in Python. Python is a general-purpose programming language that interfaces with gem5 and ResNet-50.

3.2 Machine Learning and Simulation Scheme

The premise of this research is that machine learning can identify DVFS errors. Machine Learning (ML), specifically models like those used for Image Recognition (IR), were selected for this research. Upon the completion of data extraction, the next step is ML. The use of ML is chosen because modern algorithms, especially Image Recognition (IR) algorithms, are proven to be very good at recognizing minute differences in images. For reference, a well-trained image recognition model, say facial identification, can have upwards of 99% accuracy.

IR's capabilities lead to the central thrust of the approach used, that is to transform the extracted data into images that are sufficiently different for an IR model to differentiate between a well running and a poorly running system. To implement what is a well running and a poorly running system is by its nature, somewhat arbitrary. However, what is being looked for is characteristic differences between the DVFS in the designated 'good' state against an arbitrary number of definitely 'bad' states. To do so, one reasonable unbiased P-state scheme is selected as being the designated good state scheme, that is the scheme is stable in state transitions and P-State distribution, as well as having a reasonable minimum and maximum voltage and frequency. Further, it is proposed that given that a DVFS scheme can be considered stable and not overtly flawed, that scheme is usable in this state to generate a baseline of what center (though not expected in real world) performance should be. Contrasting this is the determination of ‘bad’ state schemes, with the goal of emulating the state behaviors of DVFS performance errors to generate an image of that unfavorable state behavior. This was decided as the course of action due to the relative untouchability of the core side DVFS system, as well as the difficulty of directly designing such a performance error.

Changing the driver version to directly emulate a performance error would be possible, and much more subtle, by forcing the clustering of voltage states to emulate either not transitioning quickly enough, the inverse, or under/over ramping. The behaviors of a performance error can still be emulated, though grossly overstated. Put more favorably, it is behaviorally representative, but not functionally. The same methodology could be expanded to a physical system if one can measure accurately cycles, change in P-state, energy consumption, as well as faults in processing. However, there is the threat of measurements of the real system being of insufficient granularity (small enough timesteps) to detect accurately. The methodology for each of the DVFS performance errors follow.

3.3 Dynamic Voltage and Frequency Scaling Performance Errors

For the purposes of this paper DVFS performance errors have been clustered into three dominant modes. The first is Edge Favored, that is the power states are dominantly clustered near the upper or lower boundary voltage and frequency range being simulated. This mode is to emulate the behavior of the system statistically favoring ramping up or down in general circumstances or as called in the prior section under/over ramping, resulting in either a loss of energy performance or a slowdown of the system depending on if the upper or the lower boundaries are preferred. This is done by step by step shortening the distance between V-F pairs, that is to narrow the power difference between each state on the non-favored end, and by using less or identical numbers of V-F pairs than the default. This generates a region of more rapid transition, and a region of preference for either extreme being a more probable state of residence, on the assumption that the V-F states and DVFS scheme do not have an innate preference. It would be best if under/over ramping was not generalized through all circumstances, such as if it only happens under an arbitrary circumstance. However, by mixing in other poor behaviors in the 'bad' training set, the flaw of non-randomly implementing these error behaviors in training the algorithm this way can be lessened. The prior rationale applies to all errors discussed herein.

The second is excess stability, which is to emulate the DVFS system deciding not to transition states aggressively enough. This is done by increasing the number of adjacent states near each baseline state for the 'good' DVFS scheme. That is, there are two or three other states that drive the DVFS system, specifically the Linux cpufreq driver, which in default configuration will favor cycling to adjacent P-states before ramping towards the next normal P-state region. The consequences of a performance error like this would be a general loss of performance or a slowdown of the machine making it take more time to complete the assigned task. While this is a reasonable estimate, it should emulate depending on the number of associated P-state region different degrees of state stability, the most extreme case of over-stability being the DVFS system being locked to effectively one V-F pair, meaning that there may as well be no DVFS in the system and instead an optimal pair for general performance would be favored. The last proposed performance error is only relevant in systems that have the ability to go above the normal operating range. The method for this type of performance error was to have a singular dense cluster of P-states above the normal operating mode, with a natural grouping of V-F states in the non-overdrive range. This, similar to over ramping, it is to mimic the DVFS system favoring the overdrive portion excessively when it reaches it, leading to wasted energy. However, these are different in that instead of a general favorability of increasing V-F it is to lock into one set of V-F above all else at the detriment to the core. While core damage from this behavior is non-detectable with the used methodology, if given enough residency in this state it stands to reason it would be possible as it is outside of steady state design parameters. Finally, the idea behind these modes of performance errors are that fundamentally, unless there is a logical error in the cores overall power system, all DVFS errors can be generalized into poor behaviors of state transitions.

3.4 Data Processing Methodology

Among several options, a continuous projection of normalized, natural log scaled data was used, with zeros unplotted until they are no longer zero (these correspond to -∞ for the normalization method).

$$b_{ij} = \begin{cases} -\infty, & \text{if } a_{ij} = 0 \\ \sum_{i=0}^{n} \sum_{j=0}^{m} ln(\frac{a_{ij}}{3*max(\mathbf{A})}), & \text{otherwise} \end{cases} \quad \text{(Eq. 1)}$$

Where in this case a$_{ij}$ is the raw numerated output of gem5, and $\boldsymbol{A}$ is the matrix form of a$_{ij}$, and b$_{ij}$ is the output to be plotted and then fed into ResNet-50. Each a$_j \in \boldsymbol{A}_i$ is a raw output of gem5 [10]. Among the 61 outputs used are: energy consumed in each time step, global cycles in each time step, cycles in residency in non-default V-F states, energy consumed in different voltage-frequency residencies, and voltage residency at the end of the measurement step. Each row a$_i \in \boldsymbol{A}$ corresponds to the measurement timestep of 25μs between each of the a$_j$ datapoints in the corresponding ith row. Measured data was chosen to be scaled in this manner to narrow the final values due to a large normalization factor (number of cycles per timestep). After the normalization and scaling step, the data is divided into square, that is an equal number of timesteps to different datapoints extracted from simulation and plotted into effectively a heatmap. This is done arbitrarily with the logic that it is likely that any rectangular projection would not evenly divide the total number of timesteps, and the datamaps should be consistant to each other. This image is a gray scale intensity map of the values. To correct for unsuitable numbers of datapoints and to retain uniqueness of time instance per run, several data vectors are discarded from the middle of the data for each respective non-square SimPoint. To be absolutely accurate, this not an image of a real world item, but is simply transforming the raw data to a form the ResNet-50 base can be trained with and evaluated. The motivation behind having square heatmaps is twofold. One, it makes feeding the ML algorithm simpler, reducing preprocessing and any aberration induced by it. Second, it is indefinitely expandable to the precision of the reading, as in the granularity can be increased or decreased with little change. A note is that if given sufficient funding, one could use a similar method with a high detail heat camera upon actual microarchitectures with

DVFS systems. Then periodically take images of the chip while SimPoints are running and during boot. Depending on what core load distribution the DVFS system chooses to use, group cores by projected similarity to past assignment schemes, and use heat gain and deterioration to create data images for IR ML.

3.5 Methodology Summary

In summary, the steps taken to perform the AI simulations are as follows:

1. Various machine learning models were considered and IR and ResNet-50 were selected because of its well tested ability to accurately classify images.

2. gem5 was selected for the simulator since it is a commonly used platform for successful computer architecture research.

3. The simulator was run four times initially to ensure the process was robust and to ensure the scheduled time for the simulations was sufficient to attain meaningful results.

4. The following performance errors were defined: Overamp/Underamp, Overstability, and dwelling in overdrive.

5. Additional simulations of each SimPoint were run for each archetype of performance error for a total of eight SimPoints for each performance error type. This results in a total of 24 sources for error type datapoints and eight for non error datapoints.

6. The results were analyzed to determine if the simulator reliably identified performance errors.

7. Datamaps were designed to graphically display the results of the performance error simulations.

4. RESULTS

The results are promising. At the time of writing there are two separate sets of data sets used. The first is of more equivalent size and has closer to similarly sized train sets, to be specific the scale of the sets error to no error is 341MB to 105MB, or 32005 to 10029 images. This first set will be referred to as the small or partial set. Conversely, the second set is simply all the extracted data per a specific simulation run gathered into the separate train sets. This will be referred to as the large or complete set, the specific sizes are error to no error, 609MB to 105MB, or 57397 to 14281. Each image corresponds to between 7-14KB. Both sets are produced from a singular run of sets of V-F pairs across Eight SPEC2006 [12] SimPoints [7]. This was done due to issues with accuracy declining in disparately size datasets for image classifier training. This corresponds to a test set size of 22742 images and a training set of 90970 images. The large set was included due to the desire to know without sacrifice of parity of the number of performance error cases, the accuracy of this method, and how a more general set applies to a subset with this methodology. Both sets were randomly split to 80% train, 20% validation of which the validation subsets were combined. To do so the primary methods used are F1 score, accuracy, ROC area under curve, and the confusion matrix, and the percentage TPR and FPR. Those mean in order: accuracy, confidence, relative classifier randomness, numerical classifier performance, percentage of performance that is well performing, and the most poorly performing percentage. True positive is based off of the detection of the presence of a performance error.

4.1 Large Set Results

First of note is that the large sets no error set is unique enough such that the system can semi-accurately detect the lack of the behavior of the three performance errors in a set independently. It would be preferable, however, if the system ramped to a near 1.0 true positive rate (TPR) more cleanly from a higher initial value instead of starting from a 0.2 TPR as demonstrated in Figure 4.1. A near 1.0 TPR from initial point, the ROC plot would indicate the system is very non-random in selecting the lack of a performance error. As such and expected in Figure 4.1, the orange curve shows the verification set for no error is as it should be irrespective of prior knowledge of what is. The blue curve represents a worthless system, that is it completely randomly classifies results.

The large set, as seen in Figure 4.1, has a similar final performance in the negative case. That is, the error set is similarly detectable as the no error set. This is a mark of the data having consistently definable features in the no added error, that is the ”good” set, and added error set, or the ”bad” set, as described in Section 3 as is ideal.

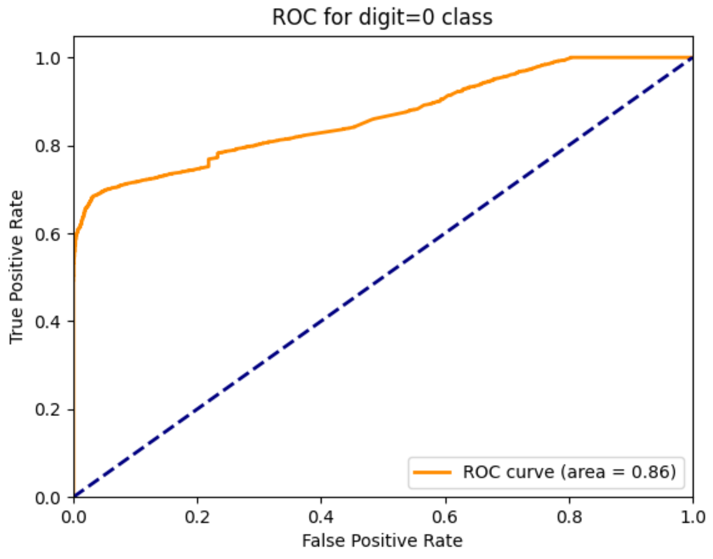

Figure 4.1: Example of large set training ROC including error and no error

Lastly, for the large set is the general data and accuracy of the set as described in Table 4.1. While it may seem odd to differentiate F1 and accuracy, they were measured independently as a sanity check to verify the confusion matrix was indeed working.

Table 4.1: Large Set Data

Large Set		
Confusion	11492	6
Matrix	2287	550
F1	0.84	
Accuracy	0.84	
TPR	0.834	
FPR	0.0108	

4.2 Small Set Results

The small set in its most broad behavior is similar to the large set above, being that the no error case is uniquely detectable. Furthermore, the performance has a higher base TPR and this can indicate that there are some issues with the discrepancy in size of the two data sets or that one of the later SimPoints [7] used produces results more similar between the no error and error subsets.

Figure 4.2 is the best result for ROC at point of writing. There is a very high initial true positive rate and the value converges well to a 1.0 TPR. The area under ROC is above 0.95 also indicating a well performing model. This indicates that by pruning the data to a more balanced set you can get even better performance, though only in subset as indicated later by the cross set tests.

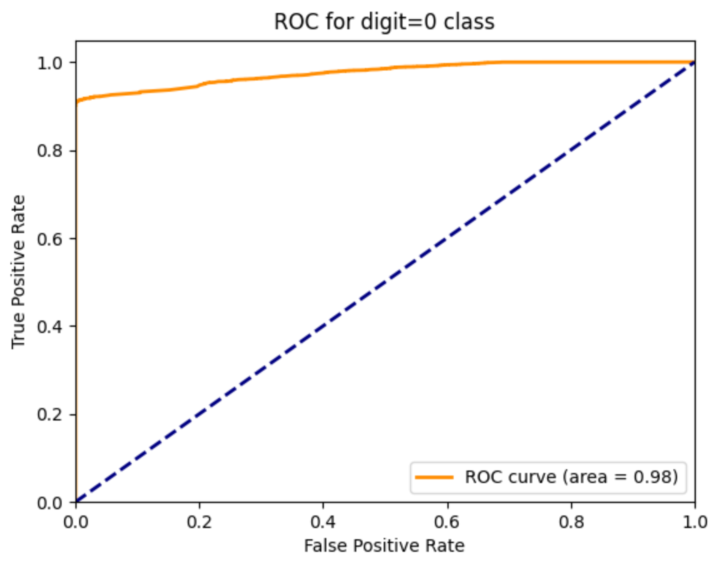

Figure 4.2: Example of small set training ROC

Finally, there is the general data of the small set in Table 4.2. Again the accuracy and F1 are gathered independently to check the confusion matrix and validity of accuracy. Their value should be identical in this case.

Table 4.2: Small Set Data

Small Set		
Confusion Matrix	4991	539
	42	1921
F1	0.931	
Accuracy	0.931	
TPR	0.992	
FPR	0.219	

4.3 Cross Set Test Results

The final significant section of my results is the cross set test results. Cross set tests use each of the two trained IR algorithms on the opposing dataset, that is, using the small dataset to train an ML model, and then using that model to try and detect errors on the large dataset. Conversely, the large to small is using a model trained on the large dataset to try to detect the presence of errors in the small dataset. In the case of a more complete dataset, this should show the ability to transfer the accuracy of the trained model to classify the same subsets of a partial dataset with approximately equal accuracy to the accuracy the model attained on the larger main dataset. More specifically, error detection may be attained past the point of detecting the presence of just performance errors in general. Using similar methods as in this work, it should be possible to use more classes in IR to indicate what archetype of DVFS performance error exists presently. Then from where that performance error is detected a highly trained system should be capable of backtracking from the error to localize the performance error.

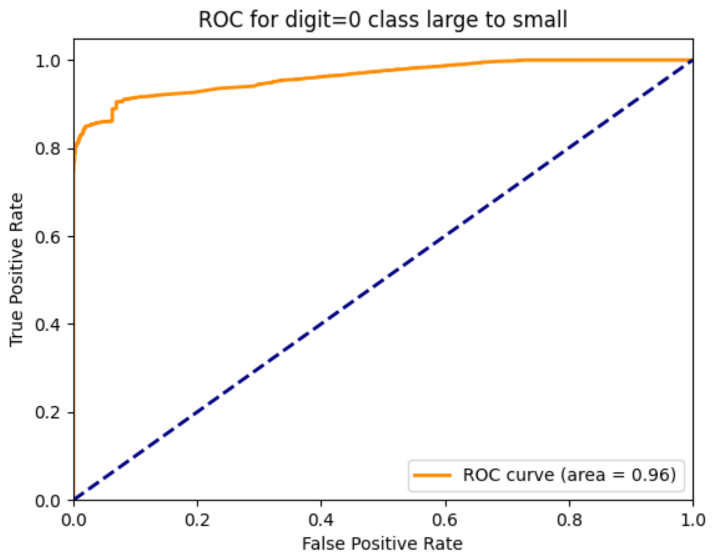

Figure 4.3: Example large to small cross set training ROC

In Figure 4.3 there is a similarity to the ROC curves in the small set cross set circumstance for the no error class, though notably the curve is more jagged, likely the result of the use of less training epochs on the large set after convergence to accuracy. That is, there was minimal improvement in the performance of the model as cycles in the epoch increased. The difference is three epochs vs. seven epochs.

Figure 4.4 demonstrates similarity between itself and Figure 4.2. This reinforces the claim that the data in the small set is more distinct between error and no error classifications. Additionally, the ability of the general case to overlap to the specific case reinforces the thrust that one can go from a broadly trained model to a small number or less broad subset of appropriately formatted data.

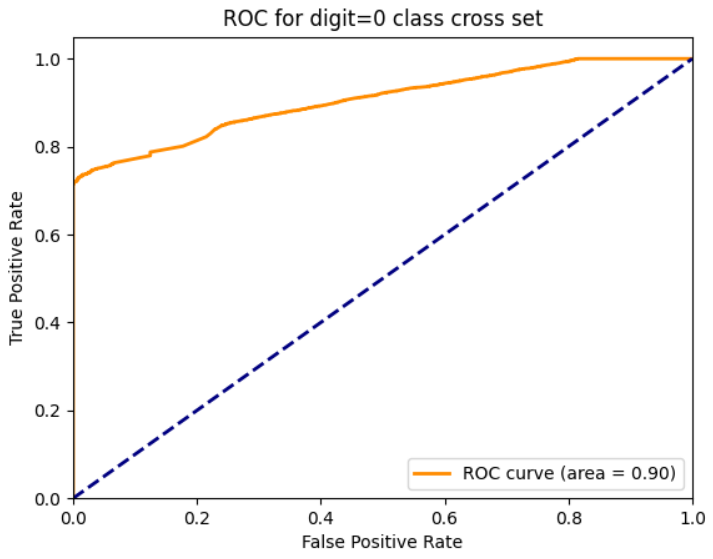

Figure 4.4: Example of cross set small to large training ROC

To verify there is some integrity preserved from the subset to the main set, I anticipated loss of accuracy should be around 19.5%, that is half of the difference in the set size delta. Performance for ROC was surprisingly good, but accuracy suffered due to inability to accurately classify the full sets. Again 4.9 and 4.10 support this. Lastly, cross set stats follow in Tables 4.3 and 4.4.

Table 4.3: Small Set Model on Large Set Data

Small to Large		
Confusion	8494	2992
Matrix	96	2780
F1	0.786	
Accuracy	0.786	
TPR	0.989	
FPR	0.518	

Table 4.4: Large Set Model on Small Set Data

Large to Small		
Confusion	6631	1
Matrix	1442	532
F1	0.828	
Accuracy	0.828	
TPR	0.821	
FPR	0.00188	

5. CONCLUSIONS

5.1 Conclusions

Since the inception of Machine Learning there have been many improvements made to this ever-evolving technology. As systems become more complex, newer solutions are developed to handle operations and optimize performance. This work is Machine Learning-Based Dynamic Voltage and Frequency Scaling Error Detection. The source of DVFS is a simulated software/driver based system which features Performance Errors versus Functional Errors and their effects on energy consumption, run times, and scheduling, to name a few. Additionally, gem5's Full System ARM DVFS simulation was used due to constraints of DVFS simulation in gem5. The basis of this research is that Machine Learning can detect DVFS errors. This research shows that it is possible to refine the structure of an output data in a manner that is detectable to a machine learning algorithm. Specifically, that by implementing differences in V-F state distribution it is possible on a time block to time block set to detect the presence of a set of performance errors.

ResNet-50, an Image Recognition model, was specifically chosen as it is particularly adept at recognizing small differences in images and behaves well with relatively small datasets. This model will seek to detect characteristic differences between DVFS in a 'good state', for example a reasonable unbiased p-state distribution scheme, against ones that are definitely poorly distributed or 'bad states.'

DVFS performance errors were categorized into three main modes. First is Edge Favored, with power states predominately appearing at upper or lower boundary voltage and frequency range being simulated. Next is excess stability whereby the DVFS system does not transition stages aggressively enough. This occurs by increasing the number of adjacent states near the baseline of the pre-designated 'good' DVFS scheme and considering all those clustered states to be the same state. Lastly, the third proposed performance error can appear in systems that have the capability to go above normal operating range. The method in this case of performance error involves a singular dense cluster of P-states above normal operating mode, with a natural grouping of V-F pairs in non-overdrive range.

Data processing involved first enumerates the gem5 output then projects it to a normalized, natural log scaled array. The measured data, after normalization and scaling, is extracted and plotted into a heatmap or datamap. This data is refined for non-plotting of zeros, datapoints and increased magnitude. The resulting image is a gray scale intensity map of values.

There are advantages and disadvantages with this methodology. First as a disadvantage, it is not exactly possible to generalize one trained model across different microarchitectures and expect a good result. The issue here is that the training and data models are designed for the behaviors of one core in simulation. Now if one possesses a model of their microarchitecture pre-silicon, and does a reclassification step to sort what part of the simulation is where, it should be simple enough to implement the performance error sets and see where the system detects members of the error subset. It should be possible to also offset the sampling angle to be more thorough in these tests, that is to shift where the image or IO matrix is in the data vectors say to start at index 30 not index 0. The biggest advantage is that the system concept does not require direct error implementation with the underlying DVFS system, merely how its behaviors are implemented, and to have some way to force it to behave poorly while maintaining the integrity of benchmarks. This means that you do not have to have anything to test it against but itself. For example, if the 'good' test case were to be removed it would only know the bad behavior, and then be able to classify out those behaviors from a general run of the pre-silicon microarchitecture model.

Inspired by the success of this research, there are other interesting ways forward to be pursued and studied. For example, the most direct is to expand to the multicore case, or to localization of performance errors in DVFS. This would enable easier debugging in larger more modern systems. Another way would be to implement the GPU cases, which would allow for more general detection. Also this GPU can be expanded to the multicore case. Further, there is the option to try to design a behavioral data based machine learning system across architectures, similar to the methodology of Barboza et. al. [3]. Lastly, one could more directly implement performance errors in the system, that is to add a performance error in the underlying system, along the same line as the prior example.

In summary, this research shows that simulation of a single core ARM CPU can implement and detect Dynamic Voltage and Frequency Scaling performance errors using a machine learning image recognition approach.

REFERENCES

[1] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED '07)*, pp. 38–43, 2007.

[2] J.-Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 308–319, 2014.

[3] E. C. Barboza, S. Jacob, M. Ketkar, M. Kishinevsky, P. Gratz, and J. Hu, "Automatic microprocessor performance bug detection," *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 545–556, 2021.

[4] S. Surya, P. Bose, and J. Abraham, "Architectural performance verification: Powerpc processors," in *Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 344–347, 1994.

[5] R. C. Ho, C. H. Yang, M. A. Horowitz, and D. L. Dill, "Architecture validation for processors," p. 404–413, 1995.

[6] R. Singhal, E. R. Cohn, D. A. Koufaty, M.-J. Lin, M. Mattwandel, and N. Nidhi, "Performance analysis and validation of the intel ® pentium ® 4 processor on 90 nm technology," 2004.

[7] Y.-M. Wang, Y. Huang, K.-P. Vo, P.-Y. Chung, and C. Kintala, "Checkpointing and its applications," in *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, FTCS '95, (USA), p. 22, IEEE Computer Society, 1995.

[8] J. D. McCalpin, "Hpl and dgemm performance variability on the xeon platinum 8160 processor," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 225–237, 2018.

[9] D. D. Penney and L. Chen, "A survey of machine learning applied to computer architecture design," 2019.

[10] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, p. 1–7, aug 2011.

[11] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A heterogeneous cpu-gpu simulator," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34–36, 2015.

[12] "SPEC CPU2006." https://www.spec.org/cpu2006.