The 25th Winona Computer Science Undergraduate Research Symposium

May 1, 2025

11:30am to 13:30pm

https://minnstate.zoom.us/j/94382378997
Zoom Meeting ID: 943 8237 8997

Winona State University
Winona, MN

Sponsored by the Department of Computer Science

at Winona State University

WINONA
STATE UNIVERSITY
Computer Science Department
http://cs.winona.edu

Table of Contents

Using Keystroke Dynamics Behavioral Biometrics to Identify Users

Bradley Budach
Winona State University
Winona, Minnesota
bbudach7@gmail.com

ABSTRACT

This study explores the use of keystroke dynamics as a behavioral biometric for user identification. Unlike physiological biometrics, such as fingerprints or facial recognition, keystroke dynamics leverages the unique typing patterns of individuals to create a distinctive signature. This goal of this research was to develop a machine learning-based system that utilizes keystroke dynamics for continuous and unobtrusive user authentication. By collecting and analyzing keystroke data from multiple users, relevant features were extracted and used to train a machine learning model to identify users with high accuracy. This study shows that using keystroke dynamics for behavioral biometrics is possible for creating a scalable authentication system that can provide an additional layer of security on top of traditional security measures.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**; **Neural networks**; • **Security and privacy → Biometrics**; • **Human-centered computing** → *Keyboards*.

KEYWORDS

Behavioral Biometrics, Machine Learning, Security, Authentication, Keyboard Behavior, Siamese Network, Feature Extraction

ACM Reference Format:
Bradley Budach. 2025. Using Keystroke Dynamics Behavioral Biometrics to Identify Users. In *Proceedings of Proceedings of the 25th Winona Computer Science Undergraduate Research Seminar*. ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

Biometrics is a type of authentication that is commonplace in the current technological landscape. There are two primary types of biometrics, physiological and behavioral. Physiological biometrics are things like fingerprints, facial recognition, and other static information about a person. These types of biometrics are extremely common in practical applications. The less common category is behavioral biometrics, which identifies people by their dynamic actions such as how they interact with devices, how they speak or how they move [7].

One of these Behavioral Biometrics is Keystroke Dynamics. Every person has a unique way they interact with the keyboard, which in turn creates a unique signature that can be used to identify them [7]. Many different behavioral biometrics can be used for identification, such as screen touches on mobile devices, mouse movement, keystroke dynamics, voice, gait, and various other behaviors, as well as combinations of these measures. The objective of this study is to create a biometrics system that uses only keystroke dynamics to authenticate users.

Many similar studies focus on keystroke dynamics in combination with other data sources. Like on mobile devices where the keystroke dynamics are used alongside things like motion sensor readings and finger movements [10, 12]. Another study used a camera pointed at the keyboard to study hand movement patterns rather than keystroke dynamics [1]. This information is not available on all devices like computers, however.

Using purely keystroke dynamics is advantageous due to its accessibility. Almost anyone interacting with a computer will have to use the keyboard for some tasks. It is also unobtrusive, as keystroke dynamics information can be collected while doing ordinary tasks on a device, and the user does not need to go out of their way to perform specific actions like using their voice, using the mouse, or doing something within view of a camera. Keystroke dynamics can also be continuous, allowing for authentication throughout sensitive processes, rather than only at the beginning.

With these considerations in mind, this study aims to implement a behavioral biometrics system with machine learning using only keystroke dynamics. This method should be unobtrusive for the user while adding an extra layer of security. Another primary goal of the study is to have this authentication system be continuous. That means having authentication that can run in the background over the course of a process that is constantly checking the users' biometrics. Lastly, this model should ideally be scalable. If the model can easily adjust to new user biometrics, it will be more flexible to implement in real-world applications.

To ensure that these goals are met, the final model should achieve a Equal Error Rate (EER) of 5% or below, with greater than 90% accuracy. Meeting these metrics will ensure ease of use for the authentication system and indicate a successful result. This is not as strict as it would be for a primary authentication system, but keyboard authentication would likely be a tertiary security measure in practice.

2 BACKGROUND RESEARCH

Behavioral biometrics has an interesting history, with the identification of unique typing patterns going back all the way to WWII. "The first mention of the use of the typing method to identify an individual can be dated to the period of the Second World War. Back then, a common way of communication was to send messages using the Morse code, and operators quickly learned how to write correspondence" [3]. Since the invention of machine learning, various behavioral biometrics have been studied for authentication purposes.

Proceedings of the 25th Winona Computer Science Undergraduate Research Seminar, April 29 and May 1, 2025, Winona, MN, US
2025.

One study [2] was done that simulated an office environment for 80 workers where information like keystroke dynamics, mouse dynamics, browsing history, and applications used were collected. Users also had to take numerous psychometric tests where further data was collected. Features were extracted from this data and used to train models to identify individual users with reasonable levels of accuracy.

Later a study [1] took a different approach, using computer vision with a camera pointed at the keyboard to identify users. This method extracts features from the video stream and uses a regression model to compare to a saved "probe". This method shows promise with a very high accuracy result, however implementing such a system practically has the obvious drawback of requiring a camera set up above a keyboard implementing this system.

More recently a method was proposed using a similar feature subset to this study and gives an outline for how such a keystroke authentication system could be implemented [7]. With mobile device security becoming more important, behavioral biometrics was studied for use on mobile devices using both keystroke dynamics as well as other device sensors like tilt and swipe patterns [8]. This is similar to studies [9, 10, 12] which used similar features on a mobile device to classify users.

Some common models used are RNNs as used in [11], LSTMs, SVMs as used in [4], Decision Trees, and Random Forests. The proposed model for this study, however, is a Siamese Network. Siamese networks were first proposed in 1993 but have found significant use in recent years when combined with more powerful networks and deep learning [5]. The most similar research to this study is [6], where a Siamese Network was used to identify users based on keystroke dynamics when entering a password with an 88% accuracy. The aims of this study are to create a model that is more broadly applicable to any typing situation. With additional features as well as novel model architecture, a more generally applicable and accurate model can be created, furthering the research on this approach.

3 METHODOLOGY

3.1 Method Outline

For this study, a machine learning model was created that can identify users using their unique keyboard dynamics. This model was built using Python TensorFlow and Keras. The model was designed with scalability in mind, which means it needs to be highly adaptable to different typing patterns and not specifically tuned to typing a specific phrase or password. The features extracted from the data were selected with this in mind and should be applicable in most typing situations across a wide range of durations. To facilitate this model, a Siamese Network (SSN) was used to compare stored profiles to target probes.

3.2 Data Collection

The first step of the process is to collect data from multiple different users. For this, a web based front-end interface was created to collect keyboard data and store results in a database. The application was then given to multiple individuals who were asked to complete a typing task for 60 seconds while keyboard data is collected. In total, data was collected on 15 different users. This data collected includes the key that was pressed, the timestamp when it was pressed, and the timestamp when it was released. This keyboard data is used for the next step, feature extraction.

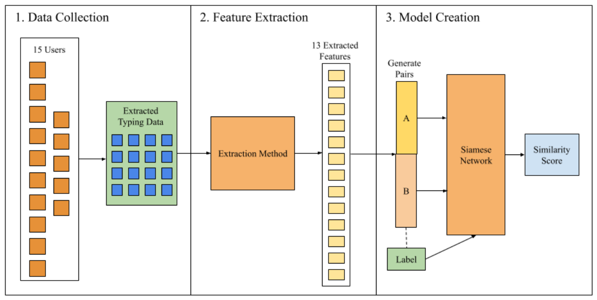


Figure 1: Methodology phases outline.

3.3 Feature Extraction

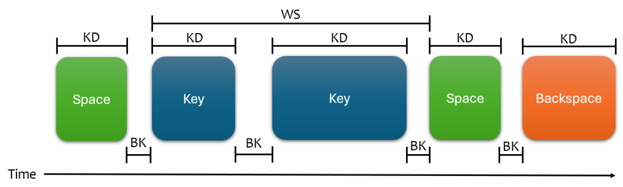


Figure 2: The features extracted based on the timing results of various keyboard actions.

A total of 13 features were extracted from the data to get a unique profile of each typist. The mean, median, range, and standard deviation were collected on the following:

- Time between each key press (BK)
- The ratio of the time taken to write a word to its length (WS)
- The time each key was down (KD)

Additionally, the ratio of backspace keys to total keys pressed was measured. These features are common across several studies like [6, 7]. WS and backspace ratio were additions made for this study. These features were selected because they provide useful information to differentiate between users while also not restricting the possible typing scenarios too much by broadening the required data collection window.

Another step performed after feature extraction is normalization. Normalization was done using Z-Score Normalization.

$$Z = \frac{x - \mu}{\sigma}$$

- Z - the standardized feature
- x - the original feature
- μ - the average for that feature
- σ - standard deviation

Normalizing the features ensures that training can be more consistent when used as an input to a machine learning model, especially ones like an SSN that rely on distance calculations.

Feature vectors were extracted on the full 60 second dataset for each user to represent their profile or signature. To gain data representing dynamic user typing, data was extracted on individual 20 second splits from their overall dataset. To combine this data into usable examples for the machine learning model. Pairs were created with every user's signature paired with data from every user's 20 second splits (probe).

$$Pair = [F_{signature}, F_{probe}]$$

The label is set to 1 if $F_{signature}$ and F_{probe} come from matching users, and 0 if they are from different users.

3.4 Model Creation

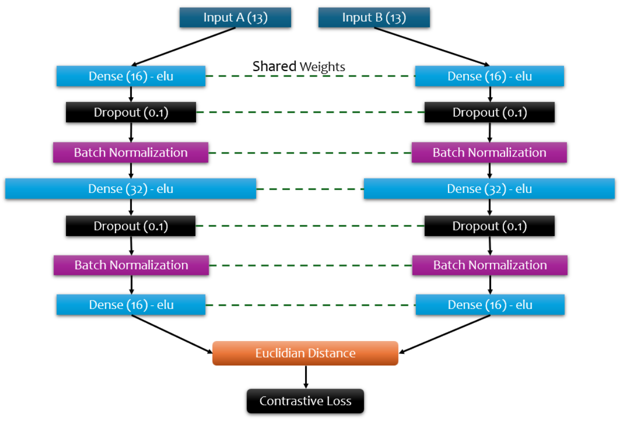


Figure 3: Siamese network architecture.

Many of the past solutions implemented for keystroke dynamics behavioral biometrics were trained to either identify a single user in a binary classification task, or a trained set of users. This requires additional training with each person added. Also, training these models requires a subset of data from multiple other people which makes it difficult if you are starting with a small set of individuals. For scalability reasons, having to re-train the model or set of models with every new user is not ideal for a production environment. That is why, for this study, a Siamese Network is used to learn a loss function on the data which can be generalized to new predictions. The structure of this network, shown in fig. 2, uses two identical Deep Neural Networks (DNNs). Input A is the saved profile (signature) of a user which is a normalized feature vector. Input B is the target feature vector that the saved profile will be compared against. Each DNN will output a feature vector:

$$InputA \longrightarrow [F_1], InputB \longrightarrow [F_2]$$

Euclidean distance is then calculated on these features with D dimensions:

$$L2(F_1, F_2) = \sqrt{\sum_{i=0}^{D} (F_{2i} - F_{1i})^2}$$

Contrastive loss is used on the L2 distance to optimize the output distance to be larger when profiles come from different users and smaller when profiles come from the same user. Contrastive loss is measured on N outputs from the current batch:

$$Contrastive\ Loss(y_{true}, y_{pred}, m) =$$
$$\frac{1}{N}\sum_{i=0}^{N}[y_{true}{}^{i} \cdot (y_{pred}{}^{i})^2 + (1 - y_{true}{}^{i})$$
$$\cdot max((m - y_{pred}{}^{i}), 0)^2]$$

The value of $y_{true}{}^{i}$ is either 1 or 0 based on whether there is a match or not. $y_{pred}{}^{i}$) is the L2 distance calculated on the input feature vectors. m is a margin value that was set to 1 for this study. If there is a between the signature and the probe, the first term is in play. $(y_{pred}{}^{i})^2$ This will punish the network for having a large distance between feature vectors from the same user. If the label is 0, the second term is in play. $max((m - y_{pred}{}^{i}), 0)^2$ This will punish the network for having a small distance between non-matching users. As a result, each DNN sub-network will learn weights that make distances for matching inputs very small, and distances for non-matching inputs very large.

To ensure the model is scalable. Measures were taken to prevent overfitting on the data. Dropout and Batch Normalization layers are used in conjunction with early stopping on a 20% validation split when training to reduce overfitting. Due to the small dataset, there are a lot more examples of the negative case (users not matching) than the positive (users matching), to prevent bias while training, random oversampling was used to duplicate cases of the positive class to match the negative. After oversampling, the training set was an even split across 1476 examples.

The model was trained with a batch size of 64 over a maximum of 200 epochs, halted by early stopping.

4 EXPERIMENTS

Various experiments with different models and hyperparameters were done during the development process that helped narrow down the methodology.

4.1 Experiment 1

A non-machine learning method was tried using normalized signatures that were compared with cosine similarity. This method showed some level of correlation between matching signatures, but the false positive and negative rate was very high, and it became clear that a machine learning method would be needed to learn more complex relationships within the data.

4.2 Experiment 2

Larger and more layers of dense networks were tested for the individual DNNs, however this seemed to make no difference in quality while slowing down processing significantly. Therefore, a smaller network was chosen.

4.3 Experiment 3

Cosine similarity was tested as a distance function in place of Euclidean distance, however Euclidean distance was by far the best performing.

4.4 Experiment 4

Various hyperparameters were tuned for better results. Initially higher dropout rates (0.3-0.5) were used, but that seemed to hurt performance too much with the smaller size of the dense layers, so a smaller dropout rate of 0.1 was chosen. Various activation functions were also tested like ReLU, Leaky-ReLU, and ELU. ELU had the best performance and was chosen as the activation function.

5 RESULTS AND ANALYSIS

For testing the final model, the standard accuracy metric was used in addition to the false acceptance rate (FAR) and false rejection rate (FRR). The equal error rate (EER) is calculated from the FAR and FRR. All these metrics vary based on the threshold (θ) chosen.

$$FAR = 100 \cdot \frac{False\ acceptances \mid \theta}{Total\ Fraudulant\ Attempts}$$
$$FRR = 100 \cdot \frac{False\ rejections \mid \theta}{Total\ Legitimiate\ Attempts}$$
$$EER = FAR\ \cap\ FRR$$

To find the optimal threshold (θ), FAR and FRR are calculated on the range $\theta : 0 \longrightarrow 1$ and θ is set at the point of the EER. A lower equal error rate means both better security and less intrusiveness on the user side with a lower rate of both false acceptances and rejections. For this study, a target EER of 5% was chosen. While this is on the higher side for important security operations, keystroke behavioral biometrics is meant to be an additional layer of security, not the primary source, so a higher target EER is an acceptable goal.

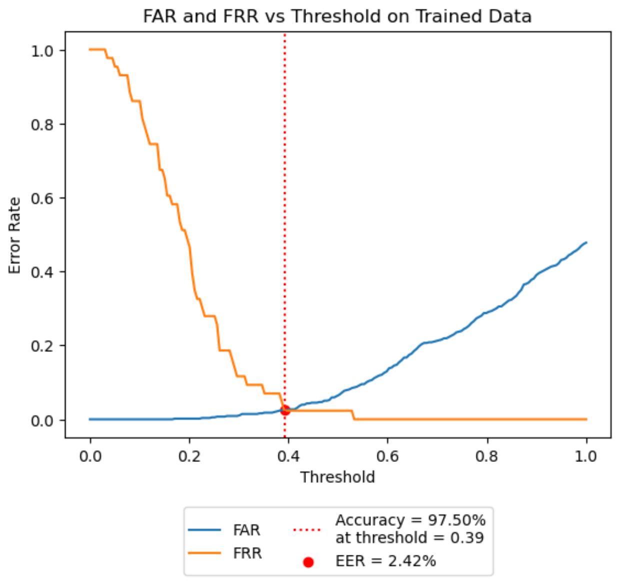


Figure 4: Accuracy FAR, FRR, and EER at best threshold on trained data.

Figure 4 indicates the EER, and accuracy of the final model on the training set. The threshold selected (0.39), is the optimal position to minimize the FAR and FRR with a resulting EER of 2.42%. This means, of the negative examples, only 2.4% were able to gain false access to the system, and only 2.4% of positive examples were rejected. This is below the EER target of 5% and indicates a robust system. The overall accuracy of this model on the testing data was 97.5%.

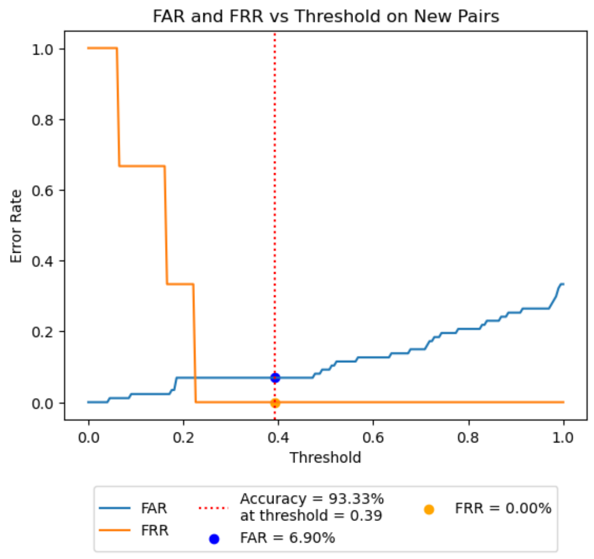


Figure 5: Accuracy FAR, and FRR, at best threshold on new data.

Figure 5 shows the same threshold but used on data that the model hasn't seen before. This means new user profiles and new positive and negative examples. At the optimal threshold, a false acceptance rate of 6.9% was measured with a 0% false rejection rate. Overall, the model performed with 93.33% accuracy on new data. These results outperform the primary comparison for this study [6] which achieved an accuracy of 89% and EER of 27% on new data, however the window for data collected in this study is much higher than a small password window which may account for some of the differences.

These results show some falloff for new data, but it is not significant. The model still performs with a greater than 90% accuracy even on new data which is within the goals for the study. Though the FAR is slightly above the target goal of 5%, which indicates some falloff in quality.

6 CONCLUSION

6.1 Findings

The results from this study show that keystroke behavioral biometrics by themselves are enough to accurately identify users. Furthermore, using a Siamese Network to differentiate user signatures on a generalized feature set shows high effectiveness and scalability, with similar quality results on new data without the need for retraining.

6.2 Future Study

Further study could be done to optimize the model for a production environment. More testing would need to be done on scalability with a much larger set of users as the data set collected for this study was small. Extracting additional features or combining with other biometric types, such as mouse dynamics, could improve model performance at scale while remaining unobtrusive.

There are some practical issues that still need to be solved before a system like this can be implemented in a production environment. One potential issue is people's typing behavior changing over time. They could be using a new keyboard, have an injury, be getting more skilled at typing, or just be tired that day. For more gradual changes, the saved profile could be updated over time with more recent data, keeping the signature close to a user's current typing style. More drastic changes like an injury may require generating a new signature. Different signatures could also be stored and used based on which device, and therefore keyboard, a user is currently using.

Privacy concerns are also a relevant issue when implementing a system like this. Users would need to be made aware of the data being collected on them, and any implementation would have to conform to relevant privacy laws.

The implementation of this system would also need to remain out of the way for normal users. False rejections have a high cost if it means locking the user out of whatever they were doing or causing annoyance. One possible way to reduce this issue is explored in the study by Sağbaş where a circular queue was used, and a user was only rejected if all checks within the queue indicated the user was invalid [9]. The length of the queue used could be optimized for either faster intrusion detection or lower risk of false rejection based on the use case.

REFERENCES

[1] Joseph Roth et al. 2014. On Continuous User Authentication via Typing Behavior. *IEEE Transactions on Image Processing* 23, 10 (2014). https://doi.org/10.1109/TIP.2014.2348802

[2] Patrick Juola, John I. Noecker, Ariel Stolerman, Michael V. Ryan, Patrick Brennan, and Rachel Greenstadt. 2013. Keyboard-Behavior-Based Authentication for Security. *IT Professional* 15, 4 (2013), 8–11. https://doi.org/10.1109/MITP.2013.49

[3] Pawel Kasprowski, Zaneta Borowska, and Katarzyna Harezlak. 2022. Biometric Identification Based on Keystroke Dynamics. *Sensors (Basel, Switzerland)* 22, 9 (Apr 2022), 3158. https://doi.org/10.3390/s22093158

[4] Sowndarya Krishnamoorthy, Luis Rueda, Sherif Saad, and Haytham Elmiligi. 2018. Identification of User Behavioral Biometrics for Authentication Using Keystroke Dynamics and Machine Learning. In *2nd International Conference on Biometric Engineering and Applications (ICBEA '18)*. 50–57. https://doi.org/10.1145/3230820.3230829

[5] Yikai Li, C. L. Philip Chen, and Tong Zhang. 2022. A Survey on Siamese Network: Methodologies, Applications, and Opportunities. *IEEE Transactions on Artificial Intelligence* 3, 6 (Dec 2022), 994–1014. https://doi.org/10.1109/TAI.2022.3207112

[6] Kamila Lis, Ewa Niewiadomska-Szynkiewicz, and Katarzyna Dziewulska. 2023. Siamese Neural Network for Keystroke Dynamics-Based Authentication on Partial Passwords. *Sensors* 23 (2023), 6685. https://doi.org/10.3390/s23156685

[7] Rohit Patil and Amar Renke. 2016. Keystroke Dynamics for User Authentication and Identification by using Typing Rhythm. *International Journal of Computer Applications* 144, 9 (Jun 2016), 27–33. https://doi.org/10.5120/ijca2016910432

[8] Dmytro Progonov, Valentyna Cherniakova, and Pavlo Kolesnichenko et al. 2022. Behavior-based user authentication on mobile devices in various usage contexts. *EURASIP Journal on Information Security* (2022), Article 6. https://doi.org/10.1186/s13635-022-00132-x

[9] Ensar Arif Sağbaş and Serkan Ballı. 2024. Machine learning-based novel continuous authentication system using soft keyboard typing behavior and motion sensor data. *Journal of Neural Computing & Applications* 36 (Jan 2024), 5433–5445. https://doi.org/10.1007/s00521-023-09360-9

[10] Ioannis Stylios, Andreas Skalkos, Spyros Kokolakis, and Maria Karyda. 2022. BioPrivacy: A Behavioral Biometrics Continuous Authentication System based on Keystroke Dynamics and Touch Gestures. *Information and Computer Security* 30, 5 (2022), 687–704. https://doi.org/10.1108/ICS-12-2021-0212

[11] Lichao Sun, Yuqi Wang, Bokai Cao, Philip S. Yu, Witawas Srisa-an, and Alex D. Leow. 2017. Sequential Keystroke Behavioral Biometrics for Mobile User Identification via Multi-view Deep Learning. In *ECML PKDD 2017*, Vol. 10536. 228–240. https://doi.org/10.1007/978-3-319-71273-4_19

[12] Ka-Wing Tse and Kevin Hung. 2019. Behavioral Biometrics Scheme with Keystroke and Swipe Dynamics for User Authentication on Mobile Platform. In *IEEE 9th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. 125–130. https://doi.org/10.1109/ISCAIE.2019.8743995

An implementation and comparison of NAT64 eBPF NAT64 technologies with the Jool Kernel Module

Arvinder Dhanoa
ArvinderDhan@gmail.com
Winona State University
Winona, Minnesota, USA

ABSTRACT

Internet Protocol v4 (IPv4) exhaustion has been a prevalent problem for years, as organizations and service providers fought against the scarcity of IPv4 address space available on the Internet. NAT64 is increasingly being deployed as a solution to this problem. As a result, it becomes increasingly important that the deployment of NAT64 technologies is easy and performant. Numerous implementations of NAT64 technologies already exist, and some new implementations use extended Berkeley's Packet Filter (eBPF) as well. In this research, we implemented a customer side translator (CLAT) with an eBPF Traffic Control (TC) classifier and compared its performance to Jool, a widely used kernel module. We did this using a series of virtual machines on two networks and a VyOS router. Using iperf3, we compared throughput when using the Transmission Control Protocol (TCP) and analyzed throughput and performance loss. Deployment is trivial, essentially loading the program to the appropriate interfaces, and the overhead is minimized because both approaches stay entirely within kernel space.

CCS CONCEPTS

• **Computer systems organization → Maintainability and maintenance**.

KEYWORDS

NAT64, Networking, eBPF, TC

ACM Reference Format:
Arvinder Dhanoa. 2025. An implementation and comparison of NAT64 eBPF NAT64 technologies with the Jool Kernel Module. In *Proceedings of Proceedings of the 25th Winona Computer Science Undergraduate Research Seminar*. ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

1.1 The history of the internet, and NAT

The Internet is made up of devices routed through a protocol known as the Internet Protocol (IP). IP is a protocol focused on hierarchical routing between networks, and using that hierarchical routing and protocols such as the Border Gateway Protocol to locate hosts on the scale of the Internet. Internet Protocol version 4 (IPv4), officially created in 1981 under RFC 791[8], was a protocol created that gave a total of 32 bits of address space for any device on the Internet. At the time, the internet was considered a research network, and as ARPANET was not meant to scale to the extent it largely has today[3]. Government, DoD and academic networks were the first

Proceedings of the 25th Winona Computer Science Undergraduate Research Seminar, April 29 and May 1, 2025, Winona, MN, US
2025.

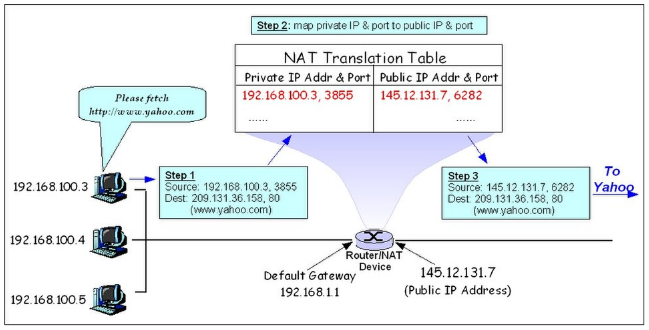


Figure 1: Depiction of NAT, with a translation table, and packet flow.

to connect to ARPANET, and gradually more commercial entities followed. Because of the expenses of computing, and network hardware at the time, it was thought that it would be unlikely that most countries would have more than 1 or 2 networks on them, with a potential of 16 million or so computers at most[3]. As a result, 32 bit addresses were chosen at the time, figuring that would be more than enough addresses to serve the entire Internet. What followed was a large-scale proliferation of networks and computational hardware. Ethernet became prolific and cheap, and computers slowly started becoming commodities. Realizing this, two standards were created in order to stifle the rapidly depleting address space in IPv4. Internet Protocol version 6 (IPv6), a new protocol increasing the size of addresses from 32bit to 128 - and a trick to keep the IPv4 internet working for a while longer called Network Address Translation (NAT)[4]. NAT (pictured in Figure 1) was a hack in which a router would designate a special IP range to the devices in it's network (specified by RFC1918), commonly referred too as private address space. This address space was not allowed to be advertised on the internet, and was used only for internal routing purposes for an organization or network. When a device wished to communicate with another device on the internet, the device would traverse through the router. This router would swap the host's source address (which was private) with its own, public address. This address was advertised on the internet. When a packet came back, it would swap in the original device's private address before forwarding. In order to correctly do this mapping, a router would inspect data lying on top of the IP protocol (typically the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP)), and send traffic to a device on the internet from a corresponding outgoing port. When traffic came back, it would use that port to figure out which device was the originator.

Although this scheme works, there are several issues with doing NAT. Because it relies on the metadata of what was sent on top of IP, in practice, the only protocols that can traverse the internet for home users is TCP and UDP. Further, end-to-end communications are severely hampered by this model. Devices can't know their own IP, and even when they do, forming an end-to-end connection is non-trivial as the NAT gateway won't know who to forward to when receiving a packet from a host on the internet. STUN/TURN are protocols to work around this, but end up plagued with their own issues. Finally, this was a relatively temporary band aid. It allowed you to put several devices behind one router, but eventually we ran out of IPv4 addresses once again.

IPv6 was a longer term solution for this problem, but adoption was slow. Much of the problems facing IPv6 boil down to all networks already supporting IPv4. There may be problems, with many countries having low enough address space where connections suffer, but for wealthier nations, address space concerns are far more theoretical still. Because every endpoint has IPv4 in some form - degraded or not - you must support IPv4 in order to participate with everyone on the internet. Having IPv6 does not preclude you from the pain of supporting IPv4, so in practice IPv6 is less important to support as a result. Further, supporting both IPv6 and IPv4 in a dual-stack environment means running two sets of networks. Two sets of routing rules, two sets of firewall rules, and two sets of problems to deal with. Although IPv6 adoption is slowly increasing[1], being at 48.30% as of April 12th, 2025 - IPv4 compatibility is still paramount.

1.2 Introducing NAT64

In order to reduce IPv4 address usage, some network operators, and hyperscale networks have started to employ something known as NAT64. NAT64 is in many ways like NAT described above. The primary distinction is that the "inside" network no longer uses RFC1918, and instead uses each device's IPv6 global address. Whenever the gateway receives a packet starting with some prefix (commonly 64:ff9b::), a NAT64 gateway strips the prefix, and understands the suffix as an IPv4 address. Normal NAT is performed, and when a response comes back, the prefix is inserted back before routing to the original host. Under this scheme, an IPv6 host can communicate with IPv4 hosts, without any internal IPv4 network. Crucially, this also means that your entire network infrastructure can operate over IPv6 - simplifying deployment and reducing your network's deployment.

The problem with this approach is how to keep compatibility with IPv4 applications under this approach. As far as hosts are concerned, they have no IPv4 networking capability, and so IPv4 applications can't access the IPv4 internet. There's two solutions to this problem commonly deployed. Something called DNS64, and 464XLAT. DNS64 is a simple scheme where a DNS server stuffs an IPv6 prefix in front of domain lookups whenever it would normally respond back with an A record containing an IPv4 address. 464XLAT is more complex. Using awareness of the prefix (communicated either by DHCP, or through some other mechanism), a host creates a fake IPv4 interface. This interface synthesizes the IPv6 packet, and routes it over the network. The network then routes it to the NAT64 gateway as normal, which translates it back to IPv4 before routing to the IPv4 internet. The false IPv4 interface is known as a Customer Side Translator (CLAT). Currently, CLAT's can be found on Apple devices[2], with Microsoft committing to adding one in the future as well[11]. There is also interest in implementations in the common Linux endpoint networking stacks[7, 10]. On Linux, Jool is a common implementation for NAT64 technologies, and uses a kernel module, which is an extension for the Linux kernel. As such, Jool has no security guarantees or boundaries, due to having direct and full access to hardware. Recently, eBPF technologies have also been gaining popularity, which offer stronger isolation and guarantees about programs written using eBPF. They're guaranteed to terminate, more isolated from the kernel, and also run in kernel space. As of April 15th, 2025 - Network Manager has a Merge Request open to implement a CLAT using eBPF[9].

Our goal was to implement a NAT64 CLAT, using eBPF, and compare throughput and CPU overhead with Jool, a popular kernel module. We believed that throughput would be comparable (within 10%), while having not much more CPU overhead (5-10%).

2 BACKGROUND RESEARCH

Although prior research exists, as well as prior methodologies to benchmark NAT64 technologies, these are largely focused on benchmarking the Provider side translator (PLAT)[5, 6]. This research also does not include eBPF implementations. These benchmarks are cohesive however, testing with millions of connections, with far more technologies than just Jool, and a variety of schemes for doings things such as allocating source ports. Other research focusing on the performance of eBPF (and especially Express Data Path (XDP) programs) exist as well, however none implement nor compare results to NAT64, which is the specific goal of this paper.

3 METHODOLOGY

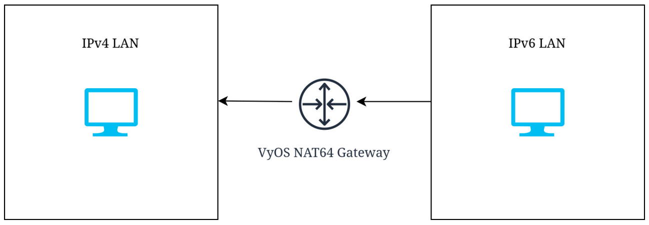


Figure 2: Network Diagram of the test network

Our tests consisted of using a virtual network. Using 3 Virtual Machines, one running VyOS, and two running conventional Linux (fedora 41), we set up a NAT64 translator on the VyOS router. The two Linux machines on each side were assigned an IPv6 and IPv4 address respectively. Using iperf3, we ran two minute tests of throughput using varying amounts of TCP streams, as well as tracked CPU usage when limiting by throughput using iperf3's throughput flag.

Although eBPF programs are commonly written in C, we chose Rust due to familiarity. We used Aya, and wrote a program which did the following steps for initialization:

(1) Assign an IPv4 address in the 192.0.0.0/29 range to the interface.
(2) Lower the IPv4 MTU by 20 bytes relative to the real MTU.

(3) Insert a fake neighbor in the ARP table within the 192.0.0.0/29 range.
(4) Create a default route to that neighbor.

The MTU was lowered in order to account for the fact that the IPv6 packet would be 20 bytes bigger than IPv4, and would have to be able to traverse the interface. The neighbor was added to the ARP table in order to get the host to be able to successfully resolve layer2, so the packet may actually be sent out. As we were working directly on the ethernet stack of the virtual machine, the packet would not be sent if this was not done, as the host would send ARP messages in order to resolve the IP in the route. An eBPF tc program is loaded in for ingress and egress, which does the following:

(1) Check if it's IPv4 or IPv6.
(2) If it's IPv6, and it's egress traffic, we let it pass.
(3) If it's IPv6, and it's ingress traffic *not* starting with the NAT64 prefix, we let it pass.
(4) If it's IPv4, we translate it to IPv6, calculate the new PSEUDO HEADER, checksum, and forward it.
(5) If it's IPv6, and it's ingress traffic starting with the NAT64 prefix for the destination address, translate it to IPv4, and stuff in the interface source address

4 RESULTS AND ANALYSIS

In our testing, we noted throughput to be nearly identical between Jool and our custom made eBPF module, although it's worth noting that we also have far higher CPU utilization. Our explanation for this is that the limiting factor is likely in a different part of the stack. It could be our benchmarking tool (iperf3), VirtIO, or some other limiting factor. Despite this, although it's clear that our implementation isn't bad performing in absolute terms - still able to keep up with 14Gb/s speeds without a problem on relatively old hardware (a Ryzen 5 3600) - it is far worse in relative terms. We didn't record Jool's overhead largely because we couldn't. The CPU overhead of Jool was minimal enough to be masked by the rest of the kernel. On the other hand, at 10Gb/s we had more than 40% cpu usage on one core. Although the hardware we tested on had 12 threads, using 40% of one thread is far slower than what Jool accomplished. Our explanation for this is likely due to the implementation details of our eBPF program. Although we needed to use something called a Traffic Control (TC) program for egress traffic (traffic leaving our network interface), a better implementation would have been able to use something called XDP instead for ingress traffic (traffic arriving on our interface). XDP offloads computation onto the Network Interface Card (NIC) and as such removes overhead on the CPU. Further, a better implementation would have also implemented our program using some form of a virtual interface, in order to eliminate Ethernet driver overhead. The current program does more work, and has layer2 overhead that is largely tossed out once the IPv4 information is translated into IPv6 or vice versa. Ideally, the layer2 overhead would only be computed once, when first arriving or leaving the ethernet interface.

5 CONCLUSION

As NAT64 becomes more predominantly used as a solution to the IPv4 compatibility problem, having high performing endpoints becomes more and more important. Although Apple platforms already ship a CLAT, and Microsoft promises to in the future, the Linux landscape is still in flux. eBPF seems to be a solid candidate for ease of deployment, and security boundaries. Although our implementation was not as optimal as it could be, the performance was still largely acceptable, and likely could get even higher performing as a result. In the future, it may also be worth implementing the NAT64 PLAT (or gateway) using eBPF as well. Having more state, observing the performance of such an implementation would put more computation on the eBPF side rather than on the rest of the networking stack - letting more of the differences shine through in the context of NAT64.

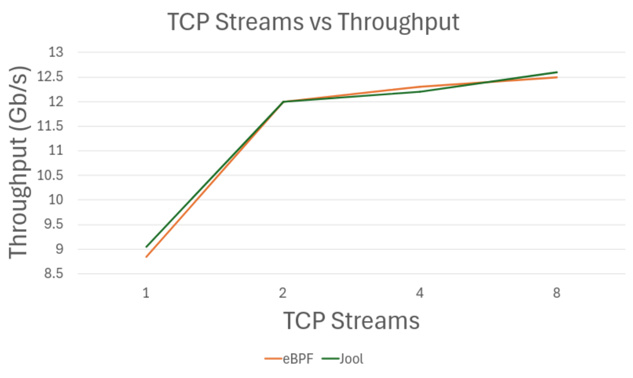


Figure 3: Throughput of the TC program and Jool

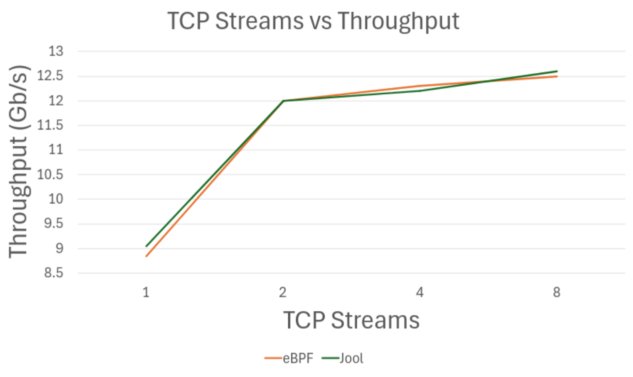


Figure 4: Core utilization of eBPF

REFERENCES

[1] [n. d.]. IPv6 Statistics. https://www.google.com/intl/en/ipv6/statistics.html
[2] Ondřej Caletka. 2022. Deploying IPv6-mostly access networks. https://blog.apnic.net/2022/11/21/deploying-ipv6-mostly-access-networks/
[3] TWiT Hangouts. [n. d.]. Hangout with Vint Cerf. https://www.youtube.com/watch?v=17GtmwyvmWE&feature=share&t=26m18s
[4] Matt Holdrege and Pyda Srisuresh. [n. d.]. RFC 2663: IP Network Address Translator (NAT) terminology and considerations. https://datatracker.ietf.org/doc/html/rfc2663
[5] Gábor Lencse, Keiichi Shima, and Kenjiro Cho. 2023. Benchmarking methodology for stateful NAT64 gateways. *Computer Communications* 210 (2023), 256–272. https://doi.org/10.1016/j.comcom.2023.08.009
[6] Gábor Lencse and Gábor Takács. 2012. Performance analysis of DNS64 and NAT64 solutions. *Infocommunications Journal (ISSN 2061-2079)* 4 (06 2012), 29–36.
[7] Petr Menšík. [n. d.]. Capability to (auto)configure 464XLAT CLAT part [RFC 6877]. https://gitlab.freedesktop.org/NetworkManager/NetworkManager/-/issues/1435

[8] J. Postel. 1981. https://datatracker.ietf.org/doc/html/rfc791
[9] Mary Strodl. [n. d.]. Add support for CLAT using a BPF program. https://gitlab.freedesktop.org/NetworkManager/NetworkManager/-/merge_requests/2107
[10] Systemd. [n. d.]. Built-in 464XLAT implementation · ISSUE 23674 · SYSTEMD/Systemd. https://github.com/systemd/systemd/issues/23674
[11] tojens. 2024. https://techcommunity.microsoft.com/blog/networkingblog/windows-11-plans-to-expand-clat-support/4078173

Using Multilayer Perceptron (MLP) to predict crop yields

Thomas Donnelly
Winona State University

tommymdonnelly@gmail.com

ABSTRACT

This study aims to develop a Multilayer Perceptron (MLP) that accurately predicts crop yields within 10% of ground truth in 80% of cases using weather data, region, soil type, temperature, fertilizer, irrigation, days taken to harvest, and rail fall. The dataset has 1 million unique data points and 10 columns. The model will be trained using a random selection of 80% of the data for training and 10% for testing. The effectiveness of the model will be derived from the accuracy and the Mean Square Error (MSE) of the model.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Contructs and Features – *abstract data types, polymorphism, control structures.* This is just an example, please use the correct category and subject descriptors for your submission.

General Terms

Your general terms must be any of the following 16 designated terms: Algorithms, Management, Measurement, Documentation, Performance, Design, Economics, Reliability, Experimentation, Security, Human Factors, Standardization, Languages, Theory, Legal Aspects, Verification.

Keywords

Keywords are your own designated keywords.

1. INTRODUCTION

As technology advances, it is important to explore as many ways it can be used to improve the world and the lives of people around us. One area where this could have a great impact is farming. Farming is notoriously difficult to predict. On average, crop yields in the US have increased since 1995, with 2023 yielding 179.3 bushels of corn per acre, compared to 113.5 in 1995 (USDA Crop, 1). This, however, can vary year to year depending on how much fertilizer is used, yearly rainfall, and the general region and climate. To help with this prediction, this study plans to create a neural network to assist in the prediction. This study plans to create a multilayer perceptron (MLP) to achieve this goal.

2. Hypothesis

Multilayer Perceptron (MLP) accurately predicts crop yields within 10% of ground truth in 80% of cases using weather data, region, soil type, temperature, fertilizer, irrigation, days taken to harvest, and rail fall.

3. Methods

3.1 Model Basics

This project will focus on developing software. This study plans to use Python, a coding language with a large library of neural network creation tools, to create the model using the Pandas library. The model I will be creating is a Multilayer Perceptron (MLP). An MLP is made up of three layers. The first layer receives all input data, including weather data, region, etc. The second layer is all the fillings or the hidden layers. These processes take the input data and try to make predictions based on it. While additional hidden layers can improve performance, having more does not necessarily result in a better model. Lastly, the output layer takes all the info the hidden layers give it and makes a final prediction or guess on the correct answer. See Figure 1 for reference.

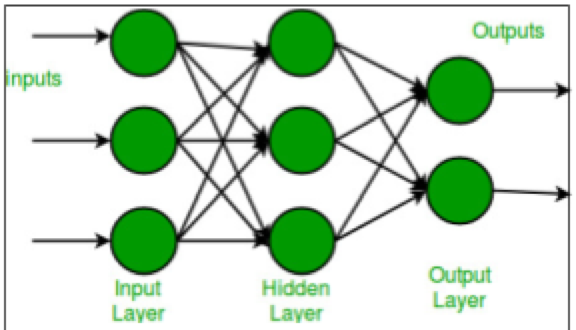


Figure 1. Deep Learning Model Visualization (Taken from Multi-Layer Perceptron Learning in TensorFlow.)

The basic structure of the model using TensorFlow is as follows:

1. Input(shape=(X_train.shape[1],)). Input Layer that just ensures the model fits the data.
2. Dense(x). X is the number of neurons that one layer can have to process data. This can be thought of as the thinking layer.
3. BatchNormalization(). This makes it so multiple Dense layers can read each other's output better.
4. ReLU() helps the model learn complex patterns in the data.
5. Repeat 2-4 with X either increasing or decreasing between layers.
6. Dense(1). This layer is the final output and prediction.

3.2 Geometric Pyrimind Formula

A formula was used to help narrow down the size of the model for training. This is called the Geometric Pyramid Formula. In short, the formula helps prevent creating a model that is too large for the data set given. The version of the formula used goes as follows: # of parameters / 10 > (layer size * previous layer size) + …… (layer(i) * layer(i - 1)).

3.3 Dataset

The dataset that will be used is taken from Kaggle, a site where you can get free datasets to work with. The dataset is titled "Agriculture Crop Yield Dataset" and is a 1 million non-null unique data point set (Agriculture Crop Yield Dataset). From my analysis of the data, it appears to be a generated set, which means it is not taken from the real world but instead created using similar data or trends in the field of interest. This is important to keep in mind as it may affect how the model can predict real-world data. The dataset has ten columns, which are as follows, with date types: Region object, Soil_Type object, Crop object, Railfall_mm float64, Temperature_Celsius float64, Fertilizer_Used boolean, Irrigation_used boolean, Weather_Condition object, Days_to_Harvest int64, Yield_tons_per_hectare float64 (Agriculture Crop Yield Dataset). The first nine columns will be used to predict the ground truth or actual value of the yield per hectare. Figures 2.1-2.5 are visualizations of the data.

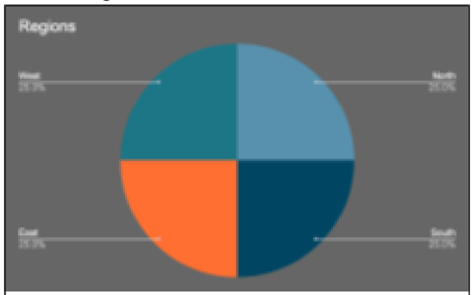


Figure 2.1 Visualization of Regions in the Dataset (Created using Matplotlib)

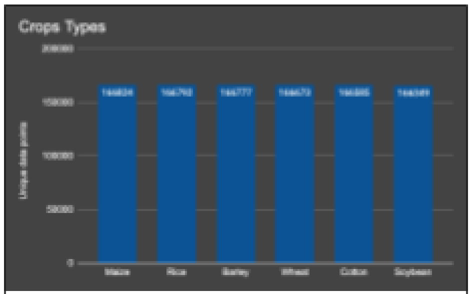


Figure 2.2 Number of Crop Types in the Dataset (Created using Matplotlib)

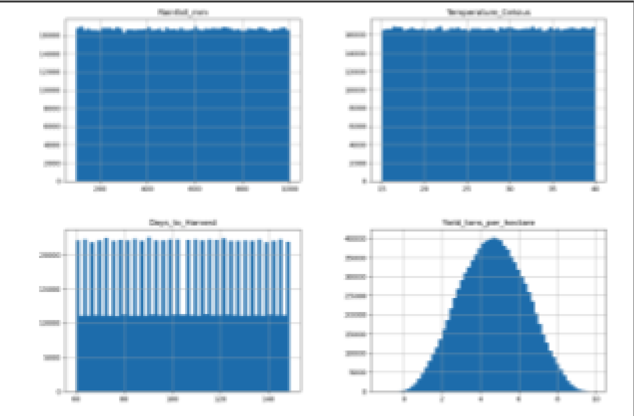


Figure 2.3 Display of the spread of Rainfall, Temperature, Days to Harvest, and Yield values. (Created using Matplotlib)

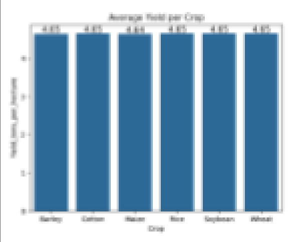


Figure 2.4 The average yield per crop type (Created using Matplotlib)

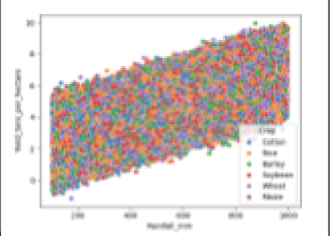


Figure 2.5 How rainfall affects crop yield (Created using Matplotlib)

3.4 Training

The method I will use for the training of the MLP is to divide the dataset into two parts. For the first part, 90% of the data will be available to the model to train on and evaluate its answers. Meanwhile, a random 10% will be set aside for after the model is done training. This data will be used to test how good the model is on previously unseen data. If the model was tested on data it had already seen, the chances would be high that it would recognize the data and learn the answer, and not how to get the answer. This is also assisted by having so many points of data, with 1 million being a large set, making it hard for the model to learn the answer and not the how.

3.5 Evaluation

Two methods will be used to show how effective the model is during and after it is created. The first is accuracy, the goal is for the model to at least 80% of the time be either 10% over or under the testing data. This can be implemented using the following logic:: accuracy = average((predicted >= actual * 0.9) OR (predicted <= actual * 1.1)) >= 0.8. The other method is Mean Square Error (MSE), this formula helps determine how accurate a model is over data, with a lower value meaning the more accurate it is (Mean Squared Error). See Figure 3 below for a similar research using MSE to display the accuracy of models. Their study is a good reference point for the project, as they tested models using only select columns to see which helped get to the ground truth the best. Meanwhile, this study will be using all the columns.

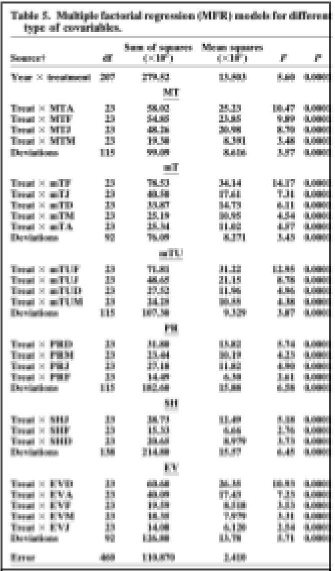


Table 5. Multiple factorial regression (MFR) models for differen type of covariables.

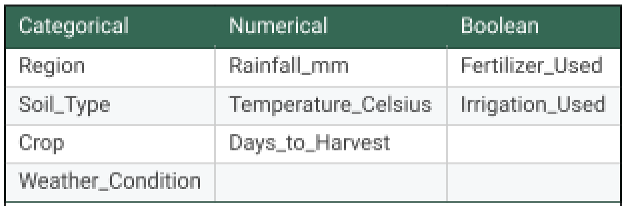
Source†	df	Sum of squares (×10[illegible])	Mean squares (×10[illegible])	F	P
Year × treatment	207	279.52	13.503	5.60	0.000
		MT			
Treat × MTA	23	58.02	25.23	10.47	0.000
Treat × MTF	23	54.85	23.85	9.89	0.000
Treat × MTJ	23	48.26	20.98	8.70	0.000
Treat × MTM	23	19.30	8.391	3.48	0.000
Deviations	115	99.09	8.616	3.57	0.000
		mT			
Treat × mTF	23	78.53	34.14	14.17	0.000
Treat × mTJ	23	40.50	17.61	7.31	0.000
Treat × mTD	23	33.87	14.73	6.11	0.000
Treat × mTM	23	25.19	10.95	4.54	0.000
Treat × mTA	23	25.34	11.02	4.57	0.000
Deviations	92	76.09	8.271	3.43	0.000
		mTU			
Treat × mTUF	23	71.81	31.22	12.95	0.000
Treat × mTUJ	23	48.65	21.15	8.78	0.000
Treat × mTUD	23	27.52	11.96	4.96	0.000
Treat × mTUM	23	24.28	10.55	4.38	0.000
Deviations	115	107.30	9.329	3.87	0.000
		PR			
Treat × PRD	23	31.80	13.82	5.74	0.000
Treat × PRM	23	23.44	10.19	4.23	0.000
Treat × PRJ	23	27.18	11.82	4.90	0.000
Treat × PRF	23	14.49	6.30	2.61	0.000
Deviations	115	182.68	15.88	6.58	0.000
		SH			
Treat × SHJ	23	28.73	12.49	5.18	0.000
Treat × SHF	23	15.33	6.66	2.76	0.000
Treat × SHD	23	20.65	8.979	3.73	0.000
Deviations	138	214.88	15.57	6.45	0.000
		EV			
Treat × EVD	23	60.68	26.35	10.93	0.000
Treat × EVA	23	40.09	17.43	7.23	0.000
Treat × EVF	23	19.19	8.318	3.45	0.000
Treat × EVM	23	18.35	7.979	3.31	0.000
Treat × EVJ	23	14.08	6.120	2.54	0.000
Deviations	92	126.88	13.78	5.71	0.000
Error	460	110.870	2.410		

Figure 3. A similar study that compared multiple models and select data with Mean Square Error being one of the evaluations (Vargas 949)

4. EXPERIMENT

4.1 Data Normalization

The first problem encountered in creating a model was that a third of the data could not be run due to being in an unsupported format. Figure 4 below shows the columns divided by their data formats.



Categorical	Numerical	Boolean
Region	Rainfall_mm	Fertilizer_Used
Soil_Type	Temperature_Celsius	Irrigation_Used
Crop	Days_to_Harvest	
Weather_Condition		

Figure 4. Training columns sorted by data type

Sklearn was used to create a pipeline to separate and manage the data in the format it's in. Since Region, Soil_Type, Crop, and Weather_Condition were strings, they could be run as is. The boolean or yes-no columns Fertilizser_Used and Irrigation_Used had to be transformed to be processed. The method used to achieve this was to convert the data to numbers, with True becoming 1 while False became 0. All the numerical values were able to be run, but a normalizer was applied to help the model read the data, and if missing values were found, the average score of the set was put in their place. After all the data was modified, it was all put back together, and the model was ready to start running.

4.2 Hardware

The expectation was to run the model on a dedicated NVIDIA GPU for its training. This would allow for quicker training, a larger possible model size, and a larger batch size. A larger model size would allow for more neurons; this does not always mean more performance out of a model but would allow the ability to see if a bigger model would be better. The batch size is how much data in a run-over data is stored in memory at a time. A larger batch size generally means higher performance of a model. It was found, however, that the versions of Nvidia GPU driver (Nvidia-smi) and Compute Unified Device Architecture (NVCC) that were installed on the machine were too new for Tensorflow, a library used to create the model, and a different machine was chosen. In the end, a MacBook M2 was used to train the model. This meant that the model had to be trained on a CPU, but given the time constraints, it was good enough.

4.3 Learning Rate

The learning rate, or the rate at which the model can learn the was chosen to be 0.001. A rate of 0.01 was tested but was not used due to the model learning the training data and learning how to predict using the data too quickly. Figure 5 is a quick example of running the model on a 0.01 learning rate. As the epochs increase or the number of times the model goes over the training data, the rate of loss on the testing data starts to reverse. Loss is a measure of how confident the model is in its answer. The reverse in loss in the testing data, but not a similar reverse in training data, shows that the model started to learn the data and not from the data.

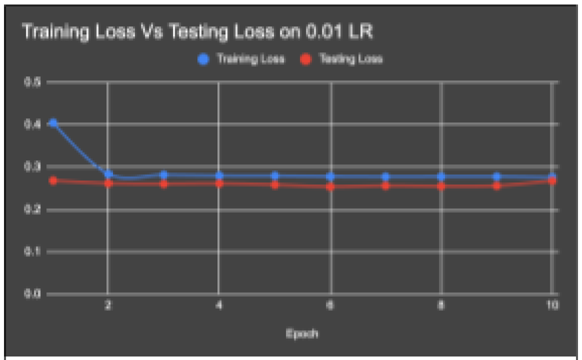


Figure 5. Comparing the model using different learning rates

5. RESULTS

See Figure 6 for the best run as it was trained over its 10 Epochs with a 0.001 learning rate and a best of 0.3979 Mean Squared Error. Figure 7 shows how the best result is structured. It has 10 Epochs, 4 hidden layers with their size in order being 256, 128, 64, and 32 neurons.

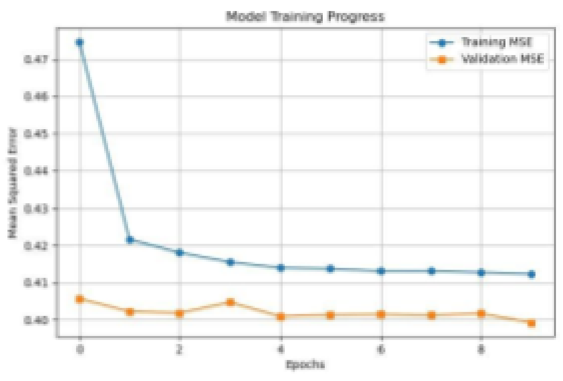


Figure 6. The best model over its epochs showing the Training MSE vs the Testing Validation MSE (Created using Matplotlib)

6. ANALYSIS

Using the best model, the study was able to achieve a total of 62.46% of the predictions being within 10% of the ground truth yield and 87.98 % of the predictions being within 20% of the ground truth yield. This has not met the goal of 80% within 10% of the ground. Training the model with the best parameters over longer epochs would not yield better results, as the model would start to learn the set and could lose accuracy. If you see Figure 8, you can see how little the MAE decreases as the epochs increase. The parameters do, however, have room to increase due to the model being well under the budget of 100,000. The model is 45,825 using the Geometric Pyramid Formula. Though when tested, it was found to slightly decrease the accuracy.

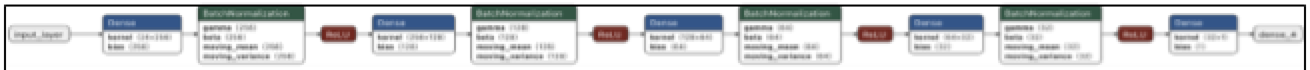
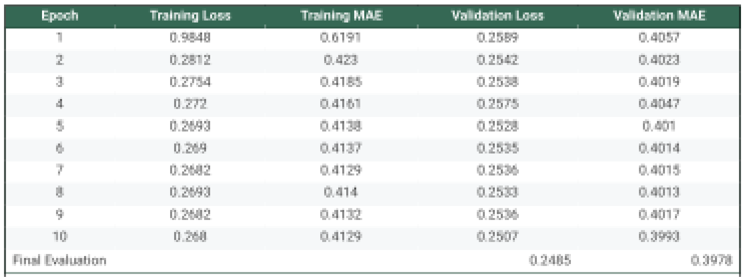


Figure 7. Visualization of the structure of the best model (Created using Netron.app)



Epoch	Training Loss	Training MAE	Validation Loss	Validation MAE
1	0.9848	0.6191	0.2589	0.4057
2	0.2812	0.423	0.2542	0.4023
3	0.2754	0.4185	0.2538	0.4019
4	0.272	0.4161	0.2575	0.4047
5	0.2693	0.4138	0.2528	0.401
6	0.269	0.4137	0.2535	0.4014
7	0.2682	0.4129	0.2536	0.4015
8	0.2693	0.414	0.2533	0.4013
9	0.2682	0.4132	0.2536	0.4017
10	0.268	0.4129	0.2507	0.3993
Final Evaluation			0.2485	0.3978

Figure 8. Table showing the best model's training data

7. CONCLUSION

In conclusion, the study did not achieve the goal it set out to achieve. If the goal post was moved to just 20% within the ground truth, then it would have achieved such. Meanwhile, 62.26% is not nothing. I believe this study has achieved its goal using a network to predict crop yield given the data.

8. REFERENCES

[1] Kaggle. "Agriculture Crop Yield Dataset." Kaggle, 2021, https://www.kaggle.com/datasets/samuelotiattakorah/agriculture-crop-yield/data.

[2] Jha, Dinesh, et al. "Application of Artificial Neural Networks for Predicting Crop Yield: A Review." Frontiers in Plant Science, vol. 10, 2019, https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2019.00621/full.

[3] "Corn Yield." USDA National Agricultural Statistics Service, https://www.nass.usda.gov/Charts_and_Maps/graphics/cornyld.pdf.

[4] "Crop Progress & Condition." USDA National Agricultural Statistics Service, 2024, https://www.nass.usda.gov/Charts_and_Maps/Crop_Progress_&_Condition/2024/index.php.

[5] Subramanian, S., et al. "Early Crop Yield Prediction for Agricultural Decision Making Using Remote Sensing Data." Journal of Water and Climate Change, vol. 14, no. 12, 2023, pp. 4729-4741, https://iwaponline.com/jwcc/article/14/12/4729/99202/Early-crop-yield-prediction-for-agricultural.

[6] Vargas, Mateo. "Interpreting Treatment × Environment Interaction in Agronomy Trials." Agronomy Journal, vol. 93, no. 4, 2001, pp. 949–960.

[7] "Multi-Layer Perceptron Learning in TensorFlow." GeeksforGeeks, https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/.

[8] "Mean Squared Error." GeeksforGeeks, https://www.geeksforgeeks.org/mean-squared-error/.

[9]

Impact of Containerization on the Performance of Web Applications

Amin Elkhalifa
Winona State University
amin.elkhalifa8@gmail.com

ABSTRACT

This study aims to evaluate the impact of running web applications in a Docker container on performance benchmarks when compared to running it natively. As technology becomes more complex and consumer demands increase usage, lightweight and efficient solutions become more valuable. One example is the server strain caused by both the popularity and computational demand of Large-Language Models. Docker attempts to offer a solution with containerization. Due to containers being offered as a deployment solution, understanding their impact on performance benchmarks is important for a diverse set of services and products. The experiment conducted consists of a web application developed for sending ping and a test script for automating the pings and gathering data. The two benchmarks included in data gathering are CPU utilization as a percentage and total roundtrip time as milliseconds. The results can be considered to make informed decisions about whether container-based deployments are viable for performance-sensitive applications.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *Performance measures.*

General Terms

Measurement, Documentation, Performance, Experimentation.

Keywords

Web applications, Docker, Containers, Images, Performance.

INTRODUCTION

Docker is an open-source platform that allows developers to build, deploy, and manage applications in "containers". Containers package an application along with its dependencies which enable it to run consistently in different environments – Linux, MacOS.

Web applications are widely deployed using containerization technologies like Docker due to their portability and convenience. Docker self-reported a 45% year-over-year increase in number of registered users, totaling at 7.3 million [3]. While Docker is already widely adopted, its continued growth makes it an industry standard.

Proceedings of the 25th Winona Computer Science Undergraduate Research Seminar, May 1, 2025, Winona, MN, US.

However, there is merit in configuring an environment to test the performance trade-offs associated with running applications as containers rather than directly on a host system. Gathering and analyzing CPU usage and network response time will provide insight for any organizations leveraging Docker to understand or predict overhead buildup and performance degradation. This study investigates whether Docker containers increase measurable performance benchmarks, particularly CPU memory usage and network response time. By comparing the performance of a Node JavaScript web application running natively versus in a Docker container, this research aims to provide empirical evidence of Docker's impact on web application performance.

1.1 BACKGROUND RESEARCH

Existing literature has explored the impact of containers on machine learning workloads and network-intensive applications, but less emphasis has been placed on front-end-heavy web applications. Nane Kratzke's Docker study focusing on network performance shows a 10 to 20% impact on data transfer rates [4]. These findings inspired this study in order to understand the impact specifically on web applications. Other considerations include studies analyzing docker that measured CPU performance by using computation benchmarks such as High Performance Linpack (HPL). This study does not take this approach in an effort to understand the impact of Docker containers has on the overall system strain. Gathering CPU utilization rather than analyzing HPL performance results provided a clearer answer to the question of performance impact. HPL is a great metric to evaluate Deep Learning tools.

2. METHODOLOGY

2.1 Development of the Ping Tool

The ping tool will be included in the testing portion of the experiment, but not the data gathering portion. The ping tool consists of a front end built with React JS (React) and back end with Node.js (Node) and Node Package Manager (NPM). The front end will have three components minimally: a designated text field to input an IP or URL address, a button to submit request, and a table to display output which will be average ping time of all the requests within a session. The back end of the ping tool will function as an API endpoint. It will resolve URLs (or IP addresses) as a parameter and send ping requests as specified and also perform basic math operations such as averaging the benchmark stats. This endpoint will be used for data gathering but not for testing during this experiment. Once pings are completed, calculations are displayed for total average ping time by tracking total clicks and returned.

2.2 Testing Environment

This experiment will be conducted on MacOS with no instruction for Windows. Node gives the ability to create a runtime environment that will serve as the "Native" implementation (as opposed to the Docker implementation) for this experiment. To run a program, open command line, navigate to its directory and type the command: npm start. When executed successfully, it will populate a response informing which port the program is currently hosted on. This port will also serve the API endpoint. The API is designed to handle simple ping requests by resolving URL or IP addresses. Navigating to localhost:[portnumber] will display the frontend. Alternatively, command line 'curl' command can be used as such:

```
curl 'localhost:[portnumber]/api/ping?address=[url or ip]'
```

2.3 Running Containers & Docker

To test Docker's performance, it is necessary to configure the code to run as a container. To do this, the Docker engine will look for a dockerfile. The main specification will be the first line in the dockerfile "FROM node:lts-alpine" which is a lightweight linux image to be run in the container as its operating system. The remaining lines are for specifying file locations and package managers such as NPM or Yarn. Together, these instructions make up the dockerfile that is used to create an *image*. Within the command line cd to the application's directory and run "docker build -t [image name]". This creates the image that can be run with the "docker run" command. Using the run command on an image creates a running instance that is called a *container*. Other considerations may include installing Docker.desktop to manage, start, and stop containers using a UI.

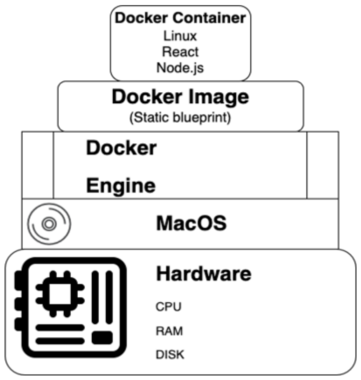


Figure 1. Diagram showing relationship between Docker Engine, Images and Containers.

Docker itself has risen in popularity due to this streamlined process. The benefit of containerizing a program is largely based on the inclusion of all the *dependencies* required to run. Dependencies include the specifications from the paragraph above. Refocusing on the Dockerfile, after the initial line "FROM node:lts-alpine" there is another line with the instruction "RUN npm install". These two lines in conjunction let the Docker Engine know that it needs Node to run the program and to update any Node packages that are used. This saves programmers from chasing bugs caused by deprecated packages or out-of-date client machines.

2.4 Gathering Data

Gathering data with the use of the CLI is straight forward. For the Docker portion, the first step is to "turn on" the API endpoint using the "docker run" command. This will prompt Docker to run the Ping Tool based on the specifications in the Docker File. Next, navigate to the testing script and run it using the python command:

```
Python [test script name].py
```

The testing script for this experiment includes a few basic components. Firstly, there are a set of three websites stored in an array as strings. Next, there is a defined helper function to retrieve current CPU utilization using Python's "OS" library. Lastly, the program loops through each website in the array and repeatedly sends the corresponding ping then calls the CPU utilization helper function 100 times. Each ping returns total roundtrip time in milliseconds which is summed and finally averaged. Each call of the CPU utilization helper function returns CPU utilization as a percentage which also summed and averaged. The testing script also includes a time buffer between pings, so it takes approximately one minute for all data to return. This concludes the testing of the Docker container. Run the command "docker stop" to prevent the container from impacting other tests. Run the command "pnpm start". Navigate to the testing script, run it again and wait for results.

3. RESULTS

Below are two tables showing summarized data of the conducted tests. Time data is continuously averaged while looping, but because CPU utilization is constantly changing, so the testing script was ran twice and averaged the CPU utilization value precisely "after" pings. This experiment attempted to achieve the most accurate CPU utilization reading by continuously averaging the value immediately after sending the ping (similar to the time data).

Table 1. Results from Docker trials.

	Facebook	Google	Winona
Time (avg. ms)	29.82	23.84	14.21
CPU Utilization (%)	4.93	5.64	4.10

Docker data is shown above. After running the docker tests, it is important to keep a computer's state as similar as possible. While testing, ideally there will be a minimal number of applications running. This study focused on keeping the "computer state" as unchanged as possible (i.e., do not close or open any new applications when completing one test and moving to the other). This was done in an effort to gather accurate data and maintain fairness.

Table 2. Results from "Docker-Free" or Native trials.

	Facebook	Google	Winona
Time (avg. ms)	28.32	20.46	13.79
CPU Utilization (%)	2.90	2.53	7.75

Native or "Docker-Free" data is shown above. Considerations include ensuring Docker is shut down and not running programs in the background. While CPU utilization will rarely produce the same results twice, this can be mitigated if the CPU utilization percentage value is averaged over a longer duration of time. The test script checks CPU utilization once with each ping and averages by dividing the total value by the number of successful (successful response) ping requests. The values are results produced from sending ping requests from Rochester, Minnesota, United States.

4. ANALYSIS

4.1 Analysis of Time Results

Using Figure 2 below, it is clear that pings consistently require more time to complete from within a Docker container. Despite the small difference (a single digit number of milliseconds), the pattern clearly establishes that a Docker container requires more time thus introducing overhead. Findings are supported Gamess' study where network response time degraded [2]. This data shows that containerizing a web application can slow down network calls by an average of 7.4%. This data supports the hypothesis of the experiment.

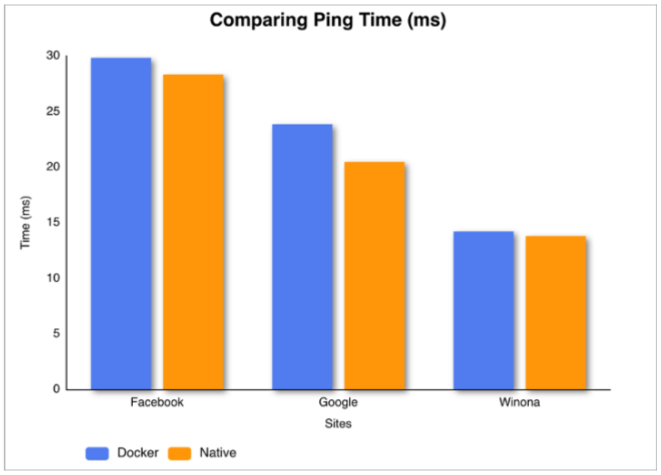


Figure 2. Chart comparing average time results.

4.2 Analysis of CPU Utilization Results

The differences between CPU Utilization in table 1 and table 2 are also small yet consistent. Figure 3 shows the visual comparison. This study finds containerized implementations consistently have an increased CPU Utilization percentage. These small changes can be used to understand patterns in containerized applications consuming CPU space, similarly to Anderson, et. al performance analysis of deep learning tools in docker containers where their research also led them to track the performance of the CPU [1]. Applications of this data analysis could include environmental reports that focus on the added power usage of containerizing increasingly large applications. Another interesting pattern is the spike of CPU utilization when pinging Winona University's home page from table 2. Two possible causes could be Winona State University's firewall due to pings from outside the school intranet as well as Facebook and Google having cutting-edge server systems.

CPU utilization is gathered using Python's OS library to call Unix's "top" command. This returns overall CPU utilization as well as running processes with their respective CPU utilization. The data in this study is from the returned overall CPU utilization to observe overall impact on the system.

5. CONCLUSION

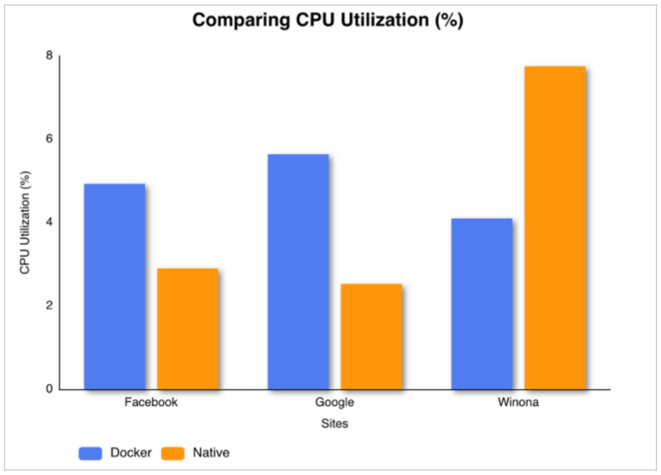


Figure 3. Chart comparing CPU utilization.

This study evaluates Docker's impact on vital benchmarks to analyze the performance of containerization. The experiment confirmed the hypothesis which predicted the added overhead of the Docker container would increase round trip time as well as CPU utilization. While the increases are negligible for some use cases, they may have an impact when developing performance-sensitive applications (e.g., healthcare or military). The data found from this experiment will hopefully provide insight to developers when making implementation decisions. Hopefully the results from this study can assist developers assess the performance impacts of containerization with Docker. Further considerations might include analysis of environmental impact of the widespread use of Docker.

6. REFERENCES

[1] Anderson, Xu, Wei, and Shimin shi. *"Performance Evaluation of Deep Learning Tools in Docker Containers."* arXiv, Nov. 2017, arxiv.org/abs/1711.03386

[2] Eric Gamess. "Performance Evaluation of the Docker Technology on Different Raspberry Pi Models." ResearchGate, Dec. 2024, www.researchgate.net/publication/374706092_Performance_Evaluation_of_the_Docker_Technology_on_Different_Raspberry_Pi_Models.

[3] Docker, Inc. *"Docker Documentation."* Docker, docs.docker.com. Accessed 20 Feb. 2025.

[4] Kratzke, Nane. "About Microservices, Containers and Their Underestimated Impact on Network Performance." *arXiv*, Oct. 2017, arxiv.org/abs/1710.04049.

Comparing Performance of Parallel Implementation of Sorting Algorithms Versus Standard Implementations

Allen Martin*
realallenmartin@gmail.com
uv9380gh@go.minnstate.edu
Winona State University
Winona, Minnesota, USA

ABSTRACT

In this research the implementation and efficiency of sorting algorithms that utilize parallel programming was looked at to determine, if parallel programming can be utilized to create a more efficient algorithm. An implementation of quick sort was developed that utilizes parallel programming and it will then be compared to the standard quick sort algorithm. The comparison was made by utilizing many arrays of different sizes with unsorted integers. They were run through both algorithms. The parallel programmed algorithm was run multiple time with varying threads for each run. This way the speedup could be calculated to determine the best use of parallel programming when sorting with quick sort. The efficiency of the algorithm was then judged based on the calculated speed up. This analysis showed that both the multithread and normal implementation are not always the fastest and depending on array size it is better to use on over the other. The quick sort algorithm in a parallel implementation will have a speedup of 3 when compared to the standard implementation of quick sort.

KEYWORDS

Quick sort, Multi-threading, Java, Thread, Implementation, Speedup, Amdahl, Executor, Parallel program

ACM Reference Format:
Allen Martin. 2025. Comparing Performance of Parallel Implementation of Sorting Algorithms Versus Standard Implementations . In *Proceedings of Proceedings of the 25th Winona Computer Science Undergraduate Research Seminar*. ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

As technology advances, the optimization of some applications becomes less important as we have better computing capabilities. Having better computing capabilities is a great thing, but it allows our code to slack and no longer be programmed to run fast as instead our hardware can make up for it. This loss of optimization has led to a reliance on hardware to make code run efficiently rather than programmers writing efficient code. One solution to achieving more efficient code is to utilize parallel programming. This is where one task is broken up into multiple parts, and the workload is split up among multiple threads who complete their task at the same time rather than having one thread complete everything.

This was implemented on a sorting algorithm to determine whether the same algorithm without parallel programming is of comparable efficiency or if the utilization of parallel programming gives a large boost in efficiency to the algorithm. There are multiple ways to achieve multi-threading, for this research the Java Executors class was used. The executor class was added in JDK 5 and allows for threads to be reused rather than creating a new thread every time. This is done by using threads pool which allow tasks to be assigned to it. If there are more tasks than threads available, the tasks are added to a queue and are completed as threads become available. Other methods include the Thread class and ForkJoinPools. This will determine by running both algorithms on the same machine with the same input parameters and calculating the speedup of the algorithms. This same test will be done on the algorithms many times over many arrays that vary in size, sorted data, unsorted data, and data types. The collected time of execution will then be gathered and used to calculate the speedup.

From *Another View on Parallel Speedup* [2] when increasing the number of processors used, the workload across the other processors is decreased. Eventually though the addition of more processors does not yield a benefit as the sequential part of the program stays the same. This is what is known as Amdahl's Law.

From *Wang Xiang* [3] and *Implementing Quicksort Programs* [1] quick sort at a basic level is selecting a pivot value. The array is then traversed and the values smaller than the pivot are on the left and values greater on the right. Then recursively continue these operations sorting the left and right partitions until the entire array is sorted.

From *An Implementation of Sorting Algorithm Based on Java Multi-thread Technology* [4] the quick sort algorithm was implemented with multi-threading by segmenting the data in parts equal to the number of threads. Then each partition of the main array was sorted by its own thread using quick sort. The two adjacent segments are then merged and the process begins again with the same number of threads as the remaining segments. This continues until there is one remaining segment and that last segment is sorted.

2 METHODOLOGY

To begin the research the implementations of quick sort must first be chosen. For both implementations the median of three was used to select our pivot point in our arrays. This is where the first, middle, and last value are taken, and whichever value is the middle value from sorting them will be our pivot point. This is also how the algorithm starts each time it is called, after the pivot point is selected that value will be moved to the end of the array so it is not altered. Once that is complete, the program iterates through the array with two variables and a temp variable. The temp variable will be used when we need to swap our values. The other two variables can be called left and right. Left will be used to keep track of what values

Proceedings of the 25th Winona Computer Science Undergraduate Research Seminar, April 29 and May 1, 2025, Winona, MN, US
2025.

are less than our pivot and right will be used to iterate over all values and compare if they are less than our pivot. If the value at right is less than our pivot the program will increment left and swap the value at left and right using temp to complete this. This continues until right reaches our pivot at the end of the array. Once this is done, we will use temp one more time to swap our pivot value with the value one to the right of out left variable.

With the array now partitioned with values greater than the pivot on the right and values less than the pivot on the left. The program will now call the quick sort function on the half of the array to the left and the right of the pivot and do the above steps again, once for each side. This process can be seen in Figure 1.

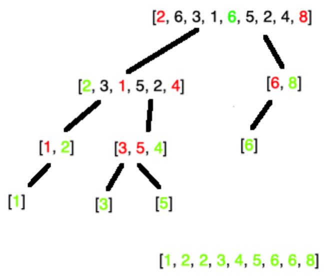


Figure 1: Quick sort algorithm broken out to show the iterations of how the algorithm accomplishes sorting an array of integers, The colored values are the values selected as a potential pivot point. The green number is the selected pivot point

The red numbers are the values not selected as the pivot and the green values are the selected pivot value. It is at this point that the there will be a difference between the parallel and non-parallel implementation. In the non-parallel implementation, the program will have to do each additionally created array one by one until the recursive function reaches the bottom and there is not more to do.

2.1 Multi-thread Implementation

The parallel implementation was done using the Java Executors class. The way this class was utilized is by first allocating a max thread count for the initial call of the program. The max thread count is a variable created in the program that lets the program know how large the inner most for loop should. This is so the for loop only tests the number of threads up to the designated max threads. When quick sort is then called the array passed through checks the current thread count, if there is one thread the default quick sort algorithm is used. Otherwise, the array is portioned and sorted. Once this is done a FixedThreadPool is created with two threads. One for the right sub array and one for the left sub array. The executor then executes on the left and right side with each call getting half the allocated threads. For example, if there are 8 threads in the initial call. The subsequent calls will each only get 4 threads. From there the calls will get 2. And then 1 at the very end, at which point the base quick sort algorithm is used for the remaining unsorted parts of the array.

2.2 Speedup

To assess how the parallel program does in comparison to the non-parallel program, the speedup between the two programs is calculated by taking the time taken of the non-parallel program to complete and divide that by the time taken of the parallel implementation. This division will output a value that will be the amount the parallel program is faster by. The equation to calculate speedup is the old time divided by the new time as seen in (1).

$$speedup = \frac{Time_{Old}}{Time_{New}} \tag{1}$$

. The reason for using this as our unit is that by Amdahl's law a parallel program can only so much faster based on how much of the program can be run in parallel. In the quick sort algorithm the only

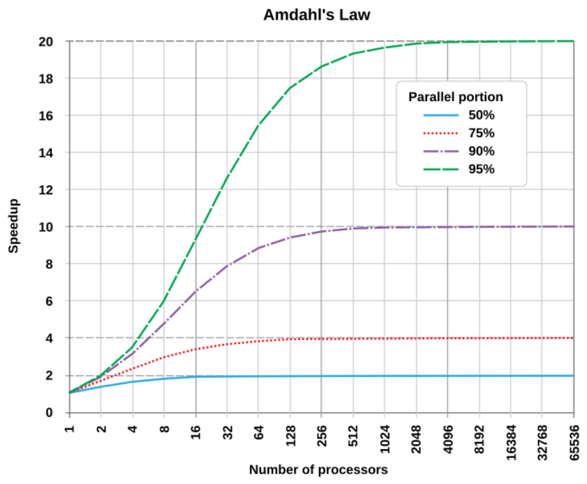


Figure 2: Amdahl's law graph displaying the max speedup based on threads and parallel percentage of the parallel part

part of the algorithm that cannot be run in parallel is the initial pivot and sorting when the algorithm is first called. From here based on how many steps down the recursion will go, determines the percentage of the program that can run in parallel. This means the larger the dataset the higher the speedup and the smaller the dataset the smaller the speedup will be.

Amdahl's law can be seen displayed in Figure 2, where is depicts the max possible speedup.

2.3 Pseudo Code

The exact implementation of the study will be as follows. The programs will be run on a computer running Windows 10 for the operating system and it will utilize a Ryzen 5800x3d for the CPU. The CPU has 16 threads, 8 was the max threads used however as the

```
if low is less than high then
    if available threads is 1 then
        call standard quick sort (Algorithm 2) for current
          partition
    else
        Data: get pivot index and sort current partition
        Data: create executor with two threads
        assign partition (Algorithm 3) to the left of pivot to
          one thread and give half of available threads
        assign partition (Algorithm 3) to the right of pivot to
          the other thread and give half of available threads
        shutdown executor
    end
end
```

Algorithm 1: ParallelQuickSort function

```
if low is less than high then
    Data: get pivot index and sort current partition
            (Algorithm 3)
    call standard quick sort (Algorithm 2) for partition to
      the left of pivot
    call standard quick sort (Algorithm 2) for partition to
      the right of pivot
    shutdown executor
end
```

Algorithm 2: QuickSort function

```
Data: get pivot value and swap it with value at end of array
Data: left pointer equal to low - 1
Data: right pointer equal to low
while right pointer is less than high do
    if value at index right pointer less than or equal to pivot
      then
        increment left pointer swap values at left pointer
          and right pointer
    end
    increment right pointer
end
increment left pointer swap value left pointer and value at
  high return left pointer
```

Algorithm 3: Partition function

CPU only has 8 cores. The program was then run as follows, there was a max tests variable set to 1000 and a max threads variable set to 8. From here the program enters a for loop where the array size is determined, and the array size increases by powers of 10 from 10^1 to 10^7. For each array size a .csv file is created this is where the data is stored to be analyzed later. The program then goes to the tests for loop where each array size has the max tests amount run on them. In this case there were 1000 tests run per array size. In this step a array of size array size is created and will be used for this round of testing. From here the final for loop is reached, this one is used to go through all thread counts from 1 to the max thread's variable. Based on the thread count then either Algorithm 2 is called when there is 1 thread and Algorithm 1 when there is greater than 1 thread. Prior to sorting there is a copy of the master array created so each run will sort the same array. There is a then a time taken in nano seconds before the sorting and a time taken after. Finally, for each run the data must be recorded for analysis later. This is done by adding a row into the csv file. The data needed is the thread count and the speedup. The exact way speedup was calculated is by storing a base time when then the base quick sort algorithm is run and then by dividing the multithread time by that time to get our speedup. The speedup then for the base implementation is always 1. With the code complete the csv files are opened in excel and some calculations are done within excel to get average speed up by thread count. With the resulting data in a new excel sheet. A graph can be created with the x-axis being threads, y-axis being speedup, and a distinct line for each array size. This graph will then show where the data in an easier to understand format.

3 RESULTS AND ANALYSIS

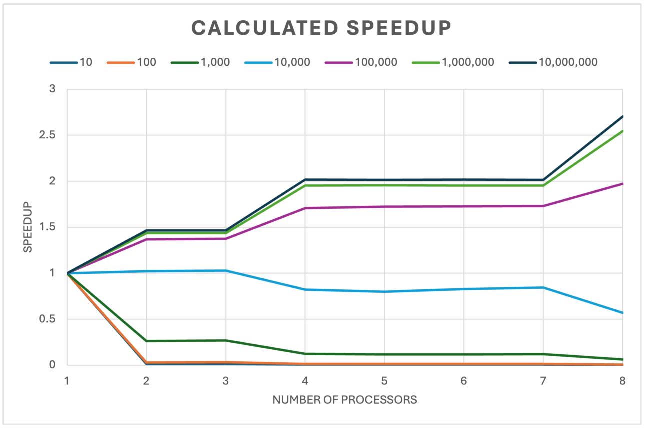
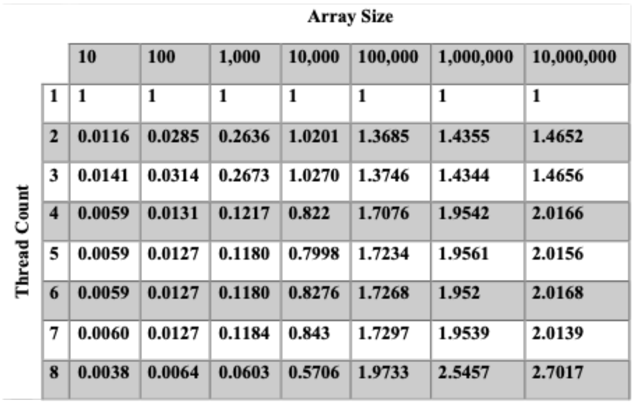


Figure 3: Graph with the average calculated speedup based on array size and tread count

Table 1: Table displaying calculated speedup from tests



Thread Count		Array Size						
		10	100	1,000	10,000	100,000	1,000,000	10,000,000
	1	1	1	1	1	1	1	1
	2	0.0116	0.0285	0.2636	1.0201	1.3685	1.4355	1.4652
	3	0.0141	0.0314	0.2673	1.0270	1.3746	1.4344	1.4656
	4	0.0059	0.0131	0.1217	0.822	1.7076	1.9542	2.0166
	5	0.0059	0.0127	0.1180	0.7998	1.7234	1.9561	2.0156
	6	0.0059	0.0127	0.1180	0.8276	1.7268	1.952	2.0168
	7	0.0060	0.0127	0.1184	0.843	1.7297	1.9539	2.0139
	8	0.0038	0.0064	0.0603	0.5706	1.9733	2.5457	2.7017

Table 1 shows the data gathered from the code on the testing machine. The data in Table 1 is gathered from the initial output of the program. The excel script "=SUMIF(A4:A10000, n, E4:E10000)/ COUNTIF(A4:A10000,n)" is used to get the average speedup of a specific thread count for the array size of that specific sheet. The value n is replaced by the specific thread count, example would be "1". The data then was calculated on a different csv file for each array amount. The resulting data was then copied onto a new file and the Table 1 was the result. The columns represent the array size, and the rows represent the thread count. The main data on the table then is the average speed up from those tests. Figure 3 is constructed using the data from Table 1. It has the thread count on the x-axis, speedup on the y-axis, and each line represents a different array size. Each value for speedup within Table 1 is the average from the 1000 tests for that specific array size and thread count.

Looking at this data multiple things can be seen. The first one is that there are two different main sections for the array size. Those two sections are the half where the speedup is worse indicating it is better to use the standard implementation, for this the point where this start is for any array size of 10,000 and lower. Although the thread counts of 2 and 3 do have a speedup greater than 1, the speedup is minimal considering it is 1.0201 compared to 1. The half then is where the multi-threaded implementation is better. From this research we cannot get the exact turning point for this as the array size went in powers of 10 but we do know it happens somewhere between 10^4 and 10^5. Within those two halves there are more details we can see as well. Looking at the data where the standard algorithm is better, one can see that as the thread count is increased the speedup decreases. This is because of the additional overhead needed to use more threads. This is represented on the table as it can be seen having a lower number of threads can lead to a better speed up over the higher thread count although not being better than the standard implementation. Although based on the data there does a exist a range of array size where multi-threaded is superior to the standard implementation and for a time less threads are also better than more threads. On the opposite end of the spectrum, once the array size gets to be extremely large, the overhead is negligible with additional threads, and we start to see a speedup increase with additional threads.

The recorded speedup for the array size of 10,000,000 does not support the hypothesis however one can assume based on the graph and data that by either increasing the threads to 16 or by increasing the array size another factor of 10, a speedup of 3 is most definitely possible and likely. The reason I say going to 16 is because of the fact that with the way the parallel algorithm is written, there needs to be a factor of 2 otherwise 7 for example gets split to 4 and 3, then to 2 and 2 and 2 and 1 and so no matter what the one layer will always take as long as if there were 4 threads. This is why the largest speedup gain is seen on thread counts with factors of 2.

4 CONCLUSION

From the research conducted it can be concluded that multi-threading does have merit when used for sorting large arrays of integers in Java. Though it does require a large array to make it worthwhile. With all this in mind in some systems where a large array of integers needs to be sorted often in worthwhile to have a multi-thread implementation for when the array size in the millions. Going forward with this line of research, there are multiple things that could be changed or continued. A few that I would like to explore is a global threads variable, this would help with the factor of 2 problem where additional threads can be allocated when available. Another on would be to expand the sorting algorithms used such as Merge sort. The final one that is most interesting is looking at changing from two partitions, left and right, to three partitions, left, middle, and right. This one would most likely suffer from the same problem with factors of threads but would be factors of 3 instead of 2. The final thing I would like to play with is possible using a global thread pool instead of creating a thread pool of size 2 for each branch in the tree. This would solve the problem of the thread count not being of size 2 and could result in some additional speedup gains when there are threads greater than a power of 2. For example, 3 threads could have a slight speedup over 2 threads.

REFERENCES

[1] Robert Sedgewick. 1978. Implementing Quicksort programs. *Commun. ACM* 21, 10 (Oct. 1978), 847–857. https://doi.org/10.1145/359619.359631

[2] Xian-He Sun and Lionel M. Ni. 1990. Another view on parallel speedup. In *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing* (New York, New York, USA) *(Supercomputing '90)*. IEEE Computer Society Press, Washington, DC, USA, 324–333.

[3] Wang Xiang. 2011. Analysis of the Time Complexity of Quick Sort Algorithm. In *2011 International Conference on Information Management, Innovation Management and Industrial Engineering*, Vol. 1. 408–410. https://doi.org/10.1109/ICIII.2011.104

[4] Xiuqiong Zhang, Hongying Qin, Tao Men, Deming Wang, and Minrong Wang. 2012. An Implementation of Sorting Algorithm Based on Java Multi-thread Technology . In *Computer Science and Electronics Engineering, International Conference on*, Vol. 2. IEEE Computer Society, Los Alamitos, CA, USA, 629–632. https://doi.org/10.1109/ICCSEE.2012.152

Comparing Image Scaling Quality Between Photoshop and Clip Studio Paint

Vera Kilpatrick
Winona State University
vera.kilpatrick@go.winona.edu

ABSTRACT

Digital art is used in every aspect of the Internet for eye catching visuals or helpful menu layouts. Photoshop and Clip Studio Paint are two of the most widely used programs for artists to create digital drawings on. In this paper, a set of images will be created and tested in both programs, starting at a base resolution and upscaled to several higher resolutions, as well as downscaled to a few lower resolutions. From the upscaled images, metrics will be qualified based on several aspects of image quality- specifically: sharpness, color preservation, detail preservation, and artifact reduction, all of which will be tested using the Python OpenCV library. The algorithms that will test the images are Histogram: to measure similarities in pixel color, Peak Signal to Noise Ratio (PSNR): to measure if any artifacts were created in the scaling, and Structural Similarity Index Measure (SSIM): to measure sharpness of the scaled image. While the program is running, the impact on the system will be recorded to compare at the end of the testing phase. Photoshop proved to have a slightly better quality that held up over higher resolutions, while Clip Studio Paint had better performance.

Keywords

Upscaling, downscaling, image quality, Image processing, Hardware performance

1. INTRODUCTION

Digital art programs have captured the majority of the market when it comes to creating art. Many styles can be created through various softwares, such as 3D, animation, graphic design, and illustration [4]. Of these softwares available, two have been greatly influential and popular among artists, and are well known for a variety of features and consistent results: Clip Studio Paint and Adobe Photoshop. Digital art is necessary for all websites in some form or another, and with the world having much of its information contained online, having good tools to produce these visuals is key to having a great website. Compressing these images to a smaller scale will help send them across slower networks, and it saves resources to do so, as well as providing a safeguard to protect the integrity of the artists' original works.

To limit the scope of this project to a reasonable scale, the specific features provided by these digital art programs will be image upscaling and downscaling. The programs used will be Adobe Photoshop and Clip Studio Paint. These programs were selected as they are well-known as the most common for digital artists, and are the leaders in their fields.

Photoshop is the most well-known digital art tool and image editing software, and older than Clip Studio, following the release of Illustrator in the 1980s. The specific purpose was to edit digital images. Adobe's Creative Cloud has allowed for a wide range of users to fill their needs with Adobe's product. The product page advertises that Photoshop "can boost colour accuracy, improve contrast and enhance shadows or lighting for better visual appeal" [5]. While the AI upscaling features are still limited in their customization options, the scope of the project was limited to the regular scaling tools.

Clip Studio Paint was created a decade later than Adobe's initial products, catering to a more specific audience of artists to create 2D digital drawings. Clip Studio has been a more accessible program to artists, as the base version does not rely on a subscription service, and provides a comparable amount of features to Photoshop.

Multiple resolutions may be necessary when exporting works from these digital programs. Scaling the image, or "resampling," changes the amount of pixels from a single image and adds or subtracts information to fill in the gaps [3]. The focus of the project had two main questions to answer: which program will provide a better quality output, and how will the performance of the system be affected by the scaling?

1.2 Hypothesis

Photoshop will provide better overall scores for the image upscaling and downscaling compared to Clip Studio Paint, due to its intention of using image processing rather than illustration. This will be coupled with a higher strain on the system.

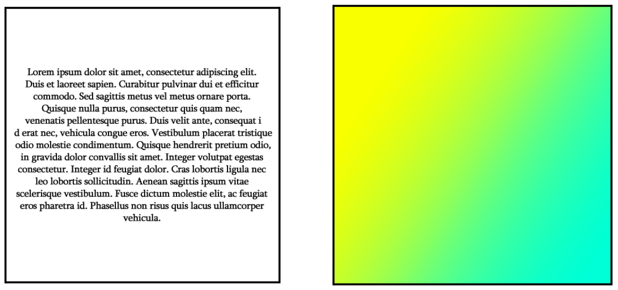
Proceedings of the 25th Winona Computer Science Undergraduate Research Seminar, May 1, 2025, Winona, MN, US.

1.3 Hypothesis Explanation

This expected outcome will be in favor of Photoshop, as it is the most well known program that has a wide range of uses and features. The backing of Adobe allows for more frequent updates to the software, and a larger library of software that is provided for additional creation. It may be that Clip Studio Paint will have a stronger specific application when it comes to certain categories like line sharpness. Clip Studio Paint is a more accessible and less hardware intensive program, making it ideal for those who are only looking to illustrate, while Photoshop will have more strength in creating larger images for those interested in illustration or photography. Clip Studio is expected to have a much lighter load on the CPU compared to Photoshop, as Photoshop has many intense processing features that will require more resources. The proposed hypothesis will be tested in a research study and paper, outlined in the following section.

2. METHODOLOGY

To answer if the hypothesis is correct, images must be created and compared between their initial resolution and the new resolution. Custom images will be created for 2 of the categories, with around 50 of each type. The first is simple, black and white images that contain



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis et laoreet sapien. Curabitur pulvinar dui et efficitur commodo. Sed sagittis metus vel metus ornare porta. Quisque nulla purus, consectetur quis quam nec, venenatis pellentesque purus. Duis velit ante, consequat i d erat nec, vehicula congue eros. Vestibulum placerat tristique odio molestie condimentum. Quisque hendrerit pretium odio, in gravida dolor convallis sit amet. Integer volutpat egestas consectetur. Integer id feugiat dolor. Cras lobortis ligula nec leo lobortis sollicitudin. Aenean sagittis ipsum vitae scelerisque vestibulum. Fusce dictum molestie elit, ac feugiat eros pharetra id. Phasellus non risus quis lacus ullamcorper vehicula.

Figure 1. Examples of "Text" and "Gradient" test images.

text, as seen on the left side of Figure 1. The goal in this category should be to preserve the sharp, defined edges, as well as prevent blurring between the letters, leaving a clean amount of white space. Another category to consider is images with several colors. This can best be tested with a gradient, as seen in Figure 1 on the right. Ideally, the gradient should be preserved between the original and the scaled image, seamlessly transitioning between the colors without mismatched pixels.

Finally, the last category will be images with small details. Since this will be the most desirable category in terms of output, this category will have a higher complexity of images. Instead of being created for this project, the selected images will be photos from datasets collected from Kaggle. Unlike illustrations, photos contain tiny details that are still important to the overall image being observed. One set has roughly 104 types of flowers [6]. These flowers will provide a good variation of colors and natural shapes to the tests. An example is in Figure 2, on the left. To contrast the flower images, a set of traffic camera data was selected [7]. These images feature man made shapes and objects, including roads and cars, and they have straighter lines and more muted color palettes, as seen in Figure 2 on the right.

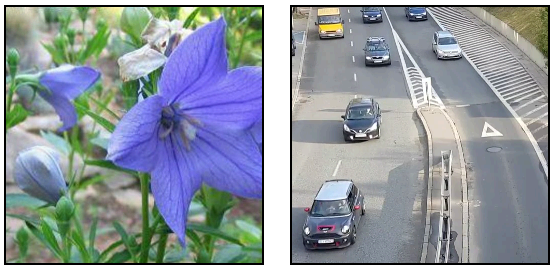


Figure 2. Examples of "Flowers" and "Cars" test images.

If the details can be preserved between the original resolution and the newly scaled image, the goal will be met. These goals are only the quality of the scaling itself. Additional metrics will quantify the performance of the software itself. This will mainly include the speed of processing, processor load, and memory used.

All upscaling tests will be completed on a Windows system. To ensure good testing, starting resolutions will all be 512 x 512 px. All images will be created at this base resolution, so as not to impede the original data from having the base images be altered prior to testing. The base resolution will be changed to multiple sizes up and down, being 0.25x, 0.5x, 2x, 4x, and 8x. The grading for each category will be done on each newly created upscale.

3. EXPERIMENTS

The image categories of quality will be graded for each category with scores from 0 - 100. A higher score is desirable in each metric.

For edge sharpness and preservation, simple black and white images of text will be tested. The text will be at several angles and sizes. For color preservation, the images will be of gradients of analogous colors as well as complimentary colors within various ranges. In addition to the characteristics of the images, artifact testing should be done to ensure the images are scaled without defects. Defects include "ringing, staircasing (also known as "jaggies"), and blurring effects" [1]. These qualities are generally known as artifacts, signaling faults of the upscaling algorithms.

3.1 Image Quality Algorithms

For more accurate measuring of quality, three algorithms will be used to compare the input images to their outputs. These will be created in Python using the OpenCV library (cv2).

3.1.1 Histogram

The first of the programs will be the histogram program. The input image and the output image will both be loaded into the program and their RGB values will be recorded, as demonstrated in Figure 3. This will be repeated for the entire category of images, and at every resolution. The upscaled image will be loaded and read as an image object, then scaled back to 512x512 using the cv2 resizing method. One limitation of the image comparison method is that it must be the same size to run the tools, so some differences may be missed by the algorithm. Differences between the inputs and outputs will be recorded and averaged. If a significant difference is found in a certain category, the RGB values will be averaged and output into two histograms to show where the peaks of different values lay on the color spectrum.

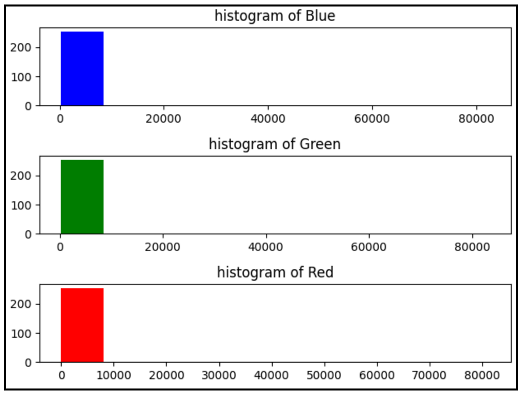


Figure 3. Example "gradient" histogram. The values are directly compared from the original to the scaled version.

By having many types of colors nearby in an image, they may upscale to appear slightly off. Once the algorithm analyzes the color consistency, a percentage of accuracy will be output to show how well the programs preserved color.

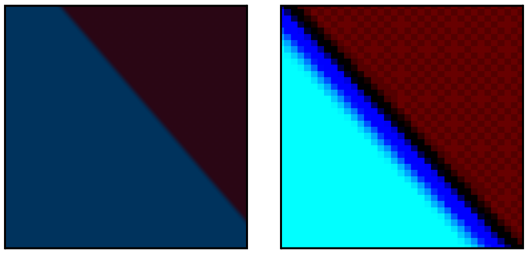


Figure 4. Gradient with adjusted contrast

In Figure 4, when adjusting the altered image's brightness and contrast, it can be seen that an incorrect region of color appears, as well as what should be a solid color that has discoloration throughout due to the upscaling algorithm.

3.1.2 Peak Signal to Noise Ratio

Peak Signal to Noise Ratio (PSNR) is the algorithm that will be used to automatically find any faults in the scaled output image compared to its original input. PSNR has been designed to find how much of the original image has been preserved in the final image by detecting any "noise," or faults in the compared output, fulfilling the artifact reduction requirement. To determine which program succeeds, a higher PSNR will be desirable. When no noise is detected, a perfect score of 100 will output. If some noise was already present in the image, there could be an incorrect value returned, or if the processing within the program results in some lost noise, which is why a variety of images was ideal. This algorithm relies on Mean Squared Error (MSE) and finds the luminance, contrast, and structure from a greyscale of both images to determine how similar the result is to the original. Figure 5 shows how a scaled image with a low PSNR result looks when zooming in. The colors appear to be blurry, and the details are not as easily distinguished.

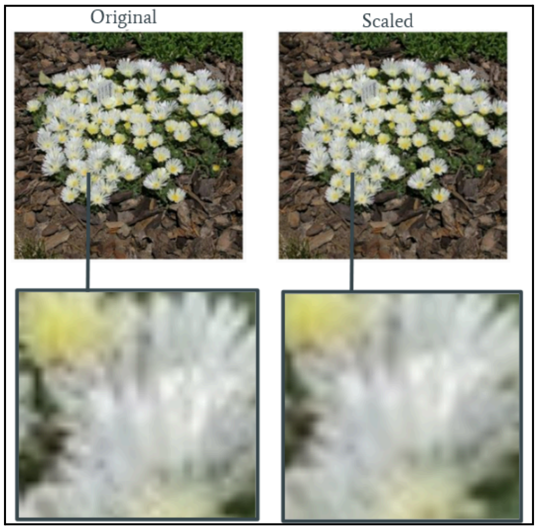
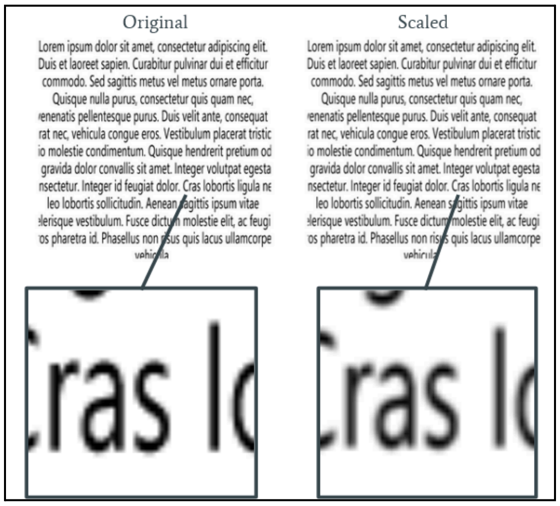


Figure 5. A low PSNR scoring image shows muddied details when zooming in.

3.1.3 Structural Similarity Index Measure

The final algorithm to analyze the scaling quality will be the Structural Similarity Index Measure (SSIM). This finds any potential issues that PSNR may have missed, specifically relating to the "sharpness" of an image. An ideal score would be 1, meaning the input and output images are the same. This algorithm is ideal for the text category, as the concern for color preservation or noise reduction is low, but it still can have many faults when altered to a new resolution, which can be seen in Figure 6.


Original
Scaled
ras l

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis et laoreet sapien. Curabitur pulvinar dui et efficitur commodo. Sed sagittis metus vel metus ornare porta. Quisque nulla purus, consectetur quis quam nec, enenatis pellentesque purus. Duis velit ante, consequat rat nec, vehicula congue eros. Vestibulum placerat tristic io molestie condimentum. Quisque hendrerit pretium od gravida dolor convallis sit amet. Integer volutpat egesta nsectetur. Integer id feugiat dolor. Cras lobortis ligula ne leo lobortis sollicitudin. Aenean sagittis ipsum vitae lerisque vestibulum. Fusce dictum molestie elit, ac feugi os pharetra id. Phasellus non risus quis lacus ullamcorpe vehicula

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis et laoreet sapien. Curabitur pulvinar dui et efficitur commodo. Sed sagittis metus vel metus ornare porta. Quisque nulla purus, consectetur quis quam nec, enenatis pellentesque purus. Duis velit ante, consequat rat nec, vehicula congue eros. Vestibulum placerat tristic io molestie condimentum. Quisque hendrerit pretium od gravida dolor convallis sit amet. Integer volutpat egesta nsectetur. Integer id feugiat dolor. Cras lobortis ligula ne leo lobortis sollicitudin. Aenean sagittis ipsum vitae lerisque vestibulum. Fusce dictum molestie elit, ac feugi os pharetra id. Phasellus non risus quis lacus ullamcorpe vehicula

Figure 6. Some blurriness occurs to the sharp edges at larger resolutions.

3.2 Performance Metrics

The processing power of each program will be recorded between the sizes of the image measured through CPU and memory usage. In order to record the impact of the software on the CPU, the

Windows tool Performance Monitor will be used. All processor related features will be observed as both Clip Studio Paint and Photoshop run at different times. Any significant spikes will be recorded and measured from the average usage these programs have. Finally, their overall usage should be recorded.

3.3 Data Collection

Each image was saved with its score compared to the original in the 3 metrics evaluations. These results were averaged across each resolution and category of image, and then compiled into a single file. While any potential outside variables were prevented from running and affecting Performance Monitor, processing issues related to the system that the programs were being recorded on are a small possibility. This consideration was taken into account for both softwares, and only process information relating to the program was selected to prevent unwanted variables.

4. RESULTS

4.1 Overall Quality Results

The quality between programs was averaged across all categories to compile the average scores, then sorted by scaled size.

The results in Figure 7 showed that as Photoshop's output image increased in size, it did not heavily affect the scores. PSNR showed consistently higher results as image size increased once averaged. This could be caused by the "gradient" category of images having a PSNR that after processing was very effective in retaining quality and decreasing noise. The effectiveness of the SSIM and Histogram test in Photoshop's higher scaling sizes also supplements Photoshop's quality at the higher scores.

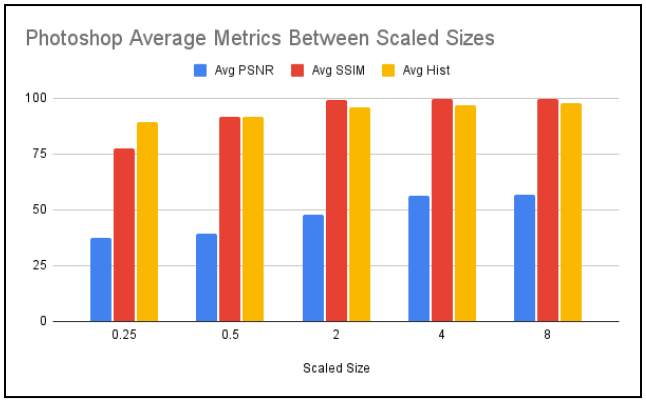


Figure 7. Photoshop's average metric scores across the scaled sizes.

Unlike Photoshop, Clip Studio Paint showed a larger decline in image quality as the output approached very high resolutions. Clip Studio Paint's results are visualized in Figure 8. At lower resolutions, the quality stayed more consistent than Photoshop's decline at those small image sizes. PSNR scores can be seen staying nearly unchanging across the resolutions.

The highest SSIM and Histogram quality remained at 2x for both programs. At lower scores, both programs are shown to have very similar resulting averages.

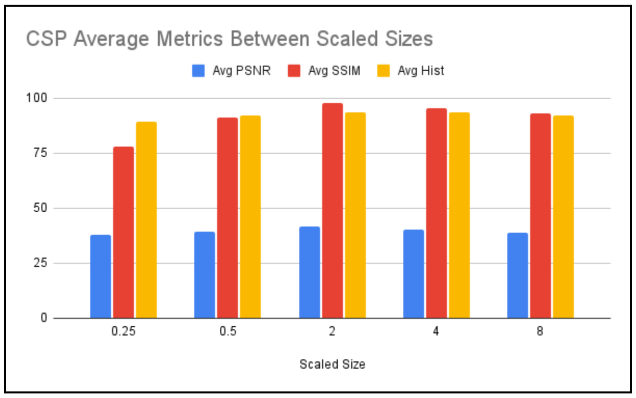


Figure 8. Clip Studio Paint's average metric scores across the scaled sizes.

4.2 Category Specific Results

To further understand the reasoning behind the scores produced by the programs, each individual category was analyzed.

In the following graphs, the categories of images are grouped and the scores are shown as ranges between the programs. From the overall scores seen in Figures 7 and 8 previously, the average PSNR is higher for Photoshop, which is reflected in Figure 9. The text category does not follow this trend, however, and the scores are near-identical between Photoshop and Clip Studio Paint. Photoshop shows a unique output of gradient PSNR scores, as many of the scores were high, while others were close to Clip Studio's. Difference in the noise score for the gradient category in Photoshop could be caused by some kind of smoothing implemented into the scaling process, and being a simple image instead of a photo, provides the alternative effect for some scaled gradients. This range is not reflected across Clip Studio Paint's scores. For the photo categories, Photoshop retains its slight edge.

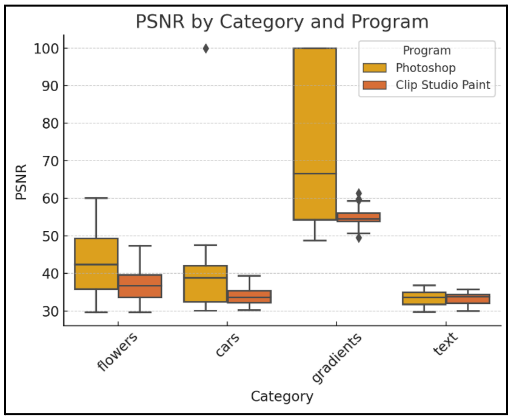


Figure 9. Ranges of PSNR scores across each category and grouped by program.

When analyzing the structure of the images using SSIM, the cars category had the widest range of scores for both programs. These scores can be seen in Figure 10. A large number of outliers were found in the flowers category and text category. These are the categories that are going to have very small details that need to be preserved, such as the ends of petals or gaps between letters respectively. Depending on the image, this could have led to some lower for both programs, especially after downscaling, as the small details become obscured.

Since a gradient does not have a “structure,” the processing algorithm found no structure to assess, therefore leading to the near-perfect scores across both programs.

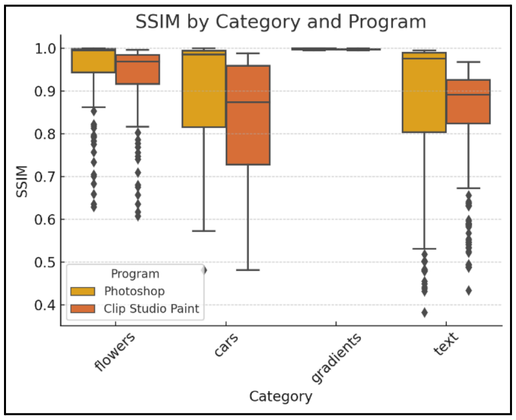


Figure 10. Ranges of SSIM scores across each category and grouped by program.

Color accuracy scores from the Histogram test were very good for the majority of the image tests. A few outliers occurred in the text category at the 0.25x size. Because the image became blurred at the small size, and the image was only monochrome, the histogram found that the image had pixels becoming closer to a gray when only white and black solid colors were in the original.

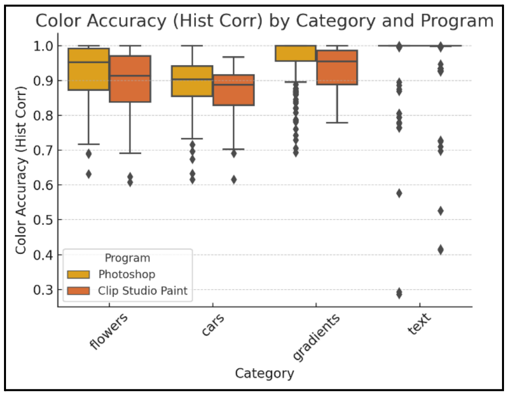


Figure 11. Ranges of Histogram scores across each category and grouped by program.

4.3 Performance Results

Performance for each category was saved as a log file to the system from Windows Performance Monitor. The scores were averaged from the grouped CPU and memory usage, having only Performance Monitor and the tested program active at once.

The average memory utilization varied greatly for Photoshop, but shows a general trend of increasing as the size deviated from 1x resolution. Clip Studio Paint had very little impact on memory, and didn’t show a significant difference between resolutions. This data is shown in Figure 12. The time spent for a single image to be upscaled to 8x took longer in Clip Studio than in Photoshop, but at lower resolutions Clip Studio was slightly faster.

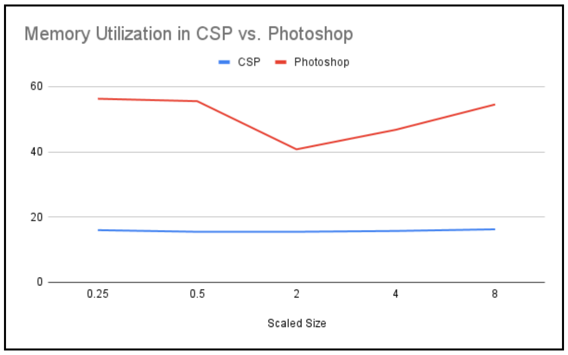


Figure 12. Average Memory usage as the tests were done across each scaled size group.

The performance of the programs resulted in Photoshop having a higher average CPU usage across all scaled resolutions. The effect on the CPU did not rise as rapidly for Photoshop at high resolutions as it did for Clip Studio. One may assume that for downscaling to very small images, the CPU would not be greatly affected compared to upscaling at sizes such as 2x, but this is not shown to be the case in Figure 13. In Photoshop’s case, 0.25x ended up in a higher average CPU% than 2x.

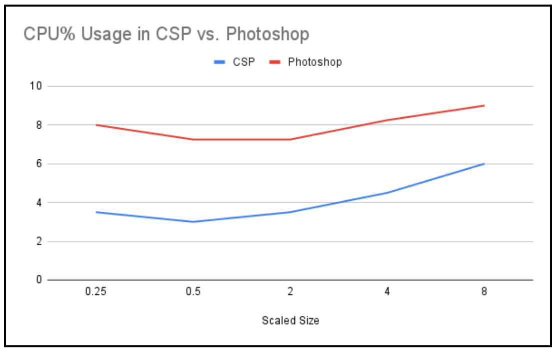


Figure 13. Average CPU usage as the tests were done across each scaled size group.

5. ANALYSIS

The results show that Photoshop’s upscaling output performed better than Clip Studio Paint. Around the 4x resolution mark, performance between the programs had similar results, but Clip Studio Paint faltered afterwards, while Photoshop remained consistently decent with quality. Each program had lower scores for images that required lots of details to be preserved. Photoshop provided a better output for the simpler image categories, gradients and text.

The lowest scores for both programs were found at the downscaling size of 0.25x. This is because much of the original images' data had to be consolidated into a small amount of pixels, so when returned to the original size for processing, much of the image had to be filled in and resulted in incorrect values.

A potential issue that could lead to some lost data involves the metric algorithms' requirements: when using PSNR, SSIM, and Histogram, the images must have the same number of pixels to be analyzed. The rescaling was done using cv2 to prevent any bias between the programs. Some values, such as small details, could be visible to a person analyzing the images at the original and scaled sizes, but could be missed by the algorithms.

6. CONCLUSION

Both programs ended up being fairly comparable in their performance. Photoshop had a greater impact on the processor and memory, but had better performance time at high resolutions. While Clip Studio had a few benefits over Photoshop, the overall quality, specifically when considering artifact reduction, will show that Photoshop has a better image scaling tool.

To expand on this project further, a comparison could be made using different algorithms for scaling provided by each of the programs, as each program allows multiple methods of exporting a scaled image. Adding more sizes to scale would be interesting, as Photoshop's quality preservation at even higher resolutions could be monitored. Additionally, more programs could be introduced to compare against the selected ones.

Both programs are competitive products. For artists looking to save money and hardware resources, Clip Studio is a great option, while Photoshop provides the added benefits that a subscription should, making it a more in-depth program.

7. REFERENCES

[1] Freedman, G., and Fattal, R. Image and Video Upscaling from Local Self-Examples. ACM Transactions on Graphics, Vol. 30, No. 2, Article 12, April 2011.

[2] Ganguly, A., et. al. ShadowMagic: Designing Human-AI Collaborative Support for Comic Professionals' Shadowing. UIST '24: Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology 105, pp. 1 - 15, Oct. 11, 2024.

[3] Adobe. How to increase resolution of an image. Retrieved from https://www.adobe.com/creativecloud/photography/discover/increase-resolution.html.

[4] Jordan, J. AI as a Tool in the Arts. Retrieved from https://amt-lab.org/blog/2020/1/ai-as-a-tool-in-the-arts, Jan. 21, 2020.

[5] Adobe. "How to upscale an image and use image enhancers with Adobe." Retrieved from https://www.adobe.com/uk/creativecloud/photography/discover/image-upscale.html.

[6] Otto, M, Fong, A. Flower Classification 104 PNG. Retrieved from https://www.kaggle.com/datasets/alenic/flower-classification-512-png. 2017.

[7] Kaggle. Cars Object Tracking. UniData. Retrieved from https://www.kaggle.com/datasets/unidatapro/cars-object-tracking. 2025.