(Draft)

Proceedings of the 41st Annual Conference of the Pennsylvania Association of Computer Science and Information Science Educators (PACISE 2026)

Order in Chaos

April 10th – 11th 2026

Hosted By

SHIPPENSBURG UNIVERSITY

Regular Papers (Undergraduate Students)

A COMPARATIVE ANALYSIS OF K-MEANS AND K-MEANS WITH CANNY EDGE DETECTION FOR FOREGROUND OBJECT EXTRACTION

Stella Andersen
Shippensburg University
sa5381@ship.edu

Dr. Dudley Girard
Shippensburg University
cdgira@ship.edu

ABSTRACT
Image segmentation provides quick and precise techniques for the identification and isolation of objects within images. This paper presents a comparative analysis of k-means clustering versus a hybrid approach combining k-means clustering with Canny edge detection for the task of foreground object extraction. The study details the operational methodologies of both approaches and evaluates their effectiveness in accurately segmenting single, distinct foreground objects from simple colored images. Segmentation accuracy is quantitatively assessed using the average symmetric surface distance (ASSD) metric against ground truth masks. The central hypothesis is that the integrated k-means + Canny method will demonstrate superior performance in aligning segmented boundaries with actual object edges compared to k-means clustering alone.

KEY WORDS
K-Means Clustering, Canny Edge Detection, Image Segmentation, Object Extraction, Comparative Analysis, Average Symmetric Surface Distance (ASSD)

1. Introduction

The fundamental problem addressed in this research is how to extract objects from images. Segmentation is a form of digital image processing that is used in significant real-world applications, such as medical imaging, surveillance, self-driving vehicles, etc. [1]. Given the diversity of segmentation techniques, it is important to compare their effectiveness.

This research focuses on two well-established techniques: k-means clustering, which groups items based on color similarity, and Canny edge detection, which identifies object boundaries based on gradient changes in pixel intensity [2], [3]. While both are effective, their strengths and weaknesses differ. This paper aims to provide a detailed understanding of these methods and, more significantly, to conduct a direct comparison between k-means clustering used independently
and a hybrid method that integrates k-means clustering with Canny edge detection. The specific application is the extraction of a single foreground object. The primary contribution will be the quantitative evaluation of these two approaches using ASSD to determine which method provides more accurate segmentation boundaries against ground truth data.

2. Background

2.1 K-Means Clustering

K-means Clustering is an unsupervised algorithm that organizes unlabeled data points (pixels, in this context) into a predefined number of clusters (k) based on feature similarity [1]. For this project, a spatially aware version of k-means is used, derived from the methodology used in SLIC Superpixels [2]. The k-means algorithm starts by first initializing the clusters, which defines the initial state of the process by selecting k starting points, or centroids [4]. A centroid in this context has both a color component and a spatial component. The initialization strategy is to randomly select $k$pixels from the image and use their full color and coordinate information as the initial centroids [5].

This implementation utilizes a 5-dimensional feature vector $V = [r, g, b, x, y]$, where (x, y) represents pixel coordinates normalized to the range $[0, 255]$ to match the scale of the color channels. This approach clusters pixels based on a weighted combination of color similarity and spatial proximity [2,5]. The distance D between a pixel i and a cluster centroid k is calculated using a weighted Euclidean distance formula, see equation 1.

$$D = (1 - w) \cdot \left|\left|C_i - C_k\right|\right| + w \cdot \left|\left|S_i - S_k\right|\right| \quad (1)$$

Where $||C_i - C_k||$ is the color distance, $||S_i - S_k||$ is the spatial distance, and w is the spatial weight parameter. For this experiment, the spatial weight was tuned to $w = 0.35$. This specific weighting ensures that pixels must be both similar

in color and physically adjacent to be grouped together, preventing the clusters from scattering incoherently across the image [2].

Next, the algorithm assigns each pixel to the cluster defined by the nearest centroid. After assigning pixels, the algorithm updates the centroids, which recalculates the position of each cluster centroid to be the mean of all pixels currently assigned to that cluster. This involves separately averaging the color vectors and the spatial coordinates of the member pixels to find the new centroid location [4, 5]. The new centroid is computed using equation 2 [4]:

$$c_i^{(t)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i} x_j \quad (2)$$

In this equation, $c_i^{(t)}$ is the new centroid for cluster i at the current iteration t. The term $|S_i^{(t)}|$ represents the total number of pixels assigned to cluster i, and the summation represents the vector sum of all pixel data points x_j (which includes both color and spatial information) belonging to that cluster [4, 5].

The final step is to check for convergence, where the assignment and update steps are repeated until the cluster centroids no longer change significantly between iterations, indicating that the algorithm has found a stable clustering solution. This condition is met when the squared Euclidean distance between the new centroid and the old centroid from the previous iteration is less than a small positive threshold value, ϵ, for all clusters from 1 to k [4]:

$$\left\| c_i^{(t)} - c_i^{(t-1)} \right\| < \epsilon \, \forall \, i \in \{1, \ldots, k\} \quad (3)$$

2.2 Canny Edge Detection

Canny edge detection is an algorithm designed to identify edges in an image by processing it and dynamically selecting a threshold based on local image properties. The process begins by converting the image to grayscale to simplify the image data to a single intensity channel. Following this, the algorithm applies a Gaussian blur to reduce noise and smooth the image, which helps prevent false edge detection [3, 6]. This step involves convolving the image with a Gaussian kernel G, which is a matrix of weights. The convolution operation that produces the blurred image B from an input image I is defined in equation 3 [3].

$$B(i,j) = \Sigma_{m=-a}^{a} \Sigma_{n=-a}^{a} I(i+m, j+n) \cdot G(m,n) \quad (4)$$

In this equation, $B(i,j)$ is the resulting blurred pixel values at coordinates (i,j), I is the input image, and G is the Gaussian kernel. The indices m and n iterate over the kernel's dimensions. The Gaussian kernel $G(m,n)$ is defined by a standard deviation parameter, σ as shown in equation 4 [3,6].

$$G(m,n) = \frac{1}{2\pi\sigma^2} e^{-\frac{m^2+n^2}{2\sigma^2}} \quad (5)$$

Here, $G(m,n)$ is the value of the kernel at coordinates (m,n) relative to the kernel's center, and the parameter σ, or standard deviation, controls the amount of blur [3].

The third step is to calculate the image gradient which finds areas in the image with high intensity changes that correspond to potential edges. This is often done using Sobel operators *(S_x, S_y)* to calculate the gradient magnitude (edge strength) and gradient direction for each pixel. The Sobel operators are shown in Figure 1 [3].

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 1: Sobel Operators [3]

These operators are convolved with the blurred image to find the gradient components in the x and y directions, G_x and G_y. From these components, the gradient magnitude M and direction θ are computed [3]:

$$M = \sqrt{G_x^2 + G_y^2} \quad (6)$$

$$\theta = tan^{-1}\left(\frac{G_y}{G_x}\right) \quad (7)$$

In the formulas above, the resulting gradient magnitude, M, represents the edge strength, while the gradient direction, θ, represents the orientation of the edge [3].

After gradients are calculated, non-maximum suppression is performed to thin the thick edges that result from the gradient calculation [3, 7]. The continuous gradient directions are rounded to one of four main angles (0°, 45°, 90°, and 135°). For each pixel, its gradient magnitude is compared with the magnitudes of its two neighbors along the rounded gradient direction. If the pixel's magnitude is not greater than both of its neighbors, it is suppressed (set to zero); otherwise, it is kept. This process thins the ridges of high gradient values down to sharp, single-pixel-wide lines [3].

Next, the algorithm uses double thresholding to classify the remaining pixels into "strong", "weak", and "non-relevant" categories based on their gradient magnitudes. To address the reliance on manually set thresholds, a fixed ratio technique can be used to derive thresholds from image properties [7].

Finally, the algorithm tracks edges using hysteresis. This procedure finalizes the edge map by iterating through the "weak" pixels. A weak pixel is promoted to a strong edge pixel if and only if it is involved in the same connected component as some strong edge pixel. This is typically

implemented by examining the 8-pixel neighborhood around every strong pixel. If a neighbor is a weak pixel, it is promoted to strong. This process continues recursively, so that the newly promoted strong pixel can, in turn, promote its own weak neighbors. This chain reaction allows the algorithm to connect faint but real parts of an edge to the definite parts, while discarding isolated weak pixels that are likely noise [8].

2.3 K-Means + Canny Hybrid Approach

This study investigates a hybrid method where Canny edge detection informs the k-means clustering process. Integrating spatial constraints or edge information into clustering is a recognized strategy for improving segmentation results by enforcing regional continuity [5]. First, a binary edge mask is generated using the Canny algorithm. This edge mask is used to constrain the k-means training phase. Specifically, pixels identified as edges are strictly excluded from the feature set used to calculate and update cluster centroids.

The k-means initialization strategy is updated to randomly select k centroids from the non-edge pixels. The k-means algorithm then iterates to find stable centroids using only the interior pixels of the objects. Once the centroids have converged, a final assignment step is performed where all pixels (including the previously excluded edge pixels) are assigned to the nearest cluster centroid. This approach biases the clustering to rely on the core colors of objects while preventing the initialization or mean-calculation from being influenced by the transitional colors found at object boundaries.

2.4 Evaluation Metrics

For this project, the primary metric for comparing segmentation accuracy is the average symmetric surface distance (ASSD). This metric evaluates the quality of a segmentation by comparing the boundaries of the predicted mask and the ground truth mask [9]. The selection of an appropriate evaluation metric is crucial, as many metrics can be sensitive to specific error types (e.g., size, location, or shape) but may not cover all types of errors [10]. ASSD, see equations 7 and 8, is "symmetric" because it calculates the average distance in both directions: from the segmentation to the ground truth and from the ground truth to the segmentation [9].

$$D = \left(\sum_{x \in S_A} \min_{y \in S_B} \|x - y\| + \sum_{y \in S_B} \min_{x \in S_A} \|y - x\| \right) \quad (7)$$

$$ASSD(S_A, S_B) = \frac{D}{|S_A| + |S_B|} \quad (8)$$

In this formula, S_A and S_B represent the sets of all points on the boundaries of the ground truth and the segmentation result. The terms $|S_A|$ and $|S_B|$ denote the total number of points in each of those sets. The variables x and y represent individual points within the boundary sets S_A and S_B. The expression $||x\text{-}y||$ calculates the Euclidean distance between two points. The “min” operator finds the shortest distance from a single point on one boundary to all points on the other boundary, and the summation symbol, $\sum$, adds all these shortest distances together. Lower ASSD values indicate that the two boundaries are, on average, closer to each other, signifying a more accurate segmentation [9].

3. Primary Objective

The primary research objective is to determine if the combination of k-means and Canny edge detection is more accurate at extracting objects than k-means alone. The project operates under a time limitation of 120 hours over 14 weeks.

3.1 Hypotheses

The hypothesis of this study is that the k-means + Canny hybrid algorithm will achieve significantly lower average symmetric surface distance (ASSD) values, indicating more accurate object boundary segmentation, compared to the k-means clustering algorithm applied alone. The corresponding null hypothesis is that there will be no statistically significant difference in ASSD values between the two algorithms when used for foreground object extraction.

4. Experiment Design

A dataset of at least 100 simple colored images, each containing a single, distinct foreground object, will be created. To evaluate the algorithms, a custom dataset of 100 images was created, comprising 20 distinct base objects placed against varying backgrounds. The dataset was categorized into high and low contrast to stress-test the segmentation. The high contrast images have objects with distinct boundaries and sharp color differences from the background. The low contrast images have objects against backgrounds of similar color. A corresponding ground truth binary mask was segmented with Canva’s background remover for every image to serve as the accuracy benchmark.

The experiment utilized a randomized block design. Both algorithms (k-means and the hybrid method) were executed on the full dataset. For each resulting segmentation, the ASSD was calculated against the corresponding ground truth mask. Since k-means initialization is stochastic, the process was repeated 10 times per image to obtain a reliable average. This resulted in a total of 2,000 experimental trials:

$$(100\ images \cdot 2\ algorithms \cdot 10\ runs) \quad (9)$$

The operational parameters were fixed at $k = 3$ centroids, $\sigma = 0.8$ for Gaussian blurring, and Canny thresholds of 0.01 (low) and 0.08 (high) to maximize sensitivity. A paired t-test was used to statistically compare the ASSD values obtained from the two segmentation approaches.

5. Solution Description

The dataset was created on a Windows desktop using Canva. Algorithm implementation and experimentation were developed on a Windows desktop in Python 3.11 using the PyCharm IDE. While the project utilizes the OpenCV library for image I/O and NumPy for matrix operations, the core segmentation algorithms were implemented from scratch rather than using pre-built library functions (e.g., cv2.Canny or sklearn.KMeans).

5.1 Algorithm Implementation

The Canny edge detector was coded as a manual 4-stage pipeline. An optimization that was made was the implementation of vectorized non-maximum suppression [7]. Instead of using computationally expensive Python loops to iterate through pixels one-by-one, the implementation utilizes NumPy array shifting to compare a pixel to its neighbors simultaneously across the entire matrix. The k-means algorithm was implemented using vectorized NumPy broadcasting to calculate the 5D distance matrix for the entire image in a single operation [11].

5.2 Automation and Metrics

A custom automation script was developed to iterate through the dataset, utilizing regular expressions (regex) to automatically pair input images with their corresponding ground truth masks. Segmentation accuracy was assessed using the average symmetric surface distance (ASSD) metric. Unlike pixel accuracy, ASSD measures the average Euclidean distance between the boundaries of the predicted mask and the ground truth mask, providing a rigorous assessment of shape accuracy [9, 10].

6. Results and Analysis

The quantitative results of the 2,000 experimental trials are summarized in Table 1. The primary metric for accuracy was the average ASSD, where a lower score indicates a closer match to the ground truth.

Contrary to the alternative hypothesis, the spatially aware k-means algorithm appeared to have a slightly better (lower) average ASSD of 4.82, compared to the hybrid method's average of 5.08. To determine if this difference was statistically meaningful, a paired t-test was conducted, resulting in a t-statistic of 1.15 and a p-value of 0.25. Since this p-value exceeds the standard alpha of 0.05, we fail to reject the null hypothesis; there is no statistically significant difference in accuracy between the two methods.

Table 1: Quantitative Results Summary

Average K-Means ASSD	4.82
Average Hybrid ASSD	5.08
K-Means Standard Deviation	6.55
Hybrid Standard Deviation	6.74
Paired T-Test Statistic	1.15
P-Value	0.25

To complement the quantitative metrics, a qualitative examination of specific test cases was conducted to observe the behavioral characteristics of each algorithm. Two images were selected to illustrate the algorithms' performance in opposing scenarios: one featuring low contrast, and one featuring high contrast along with complex internal texture.

In the "Denim" test case (Figure 2), where the pixel intensity difference between the object and background was minimal, the Hybrid method achieved a lower ASSD of 0.87, compared to 0.98 for k-means. The Canny edge detector successfully identified the faint boundary where color-based clustering struggled, guiding the algorithm to a more accurate shape definition.

In the "Luffy" test case (Figure 3), the object was a black poodle puppy wearing a harness with a white circle, along with white tufts of fur on the chest. The Canny detector interpreted the sharp gradients between the black fur and the white elements as structural edges. Because the hybrid algorithm treats edges as hard constraints, it prevented the k-means clustering from merging the white chest and harness areas with the rest of the body. This resulted in the hybrid method performing slightly worse (ASSD 0.75) than k-means alone (ASSD 0.71).

Finally, the randomly determined nature of k-means initialization likely contributed to the inconsistency observed across trials. Although this study mitigated variance by averaging ten runs per image, the standard k-means algorithm is prone to converging on local minima depending on where centroids are randomly placed. In the Hybrid approach, this instability is exacerbated because the "search space" for initialization is constrained to non-edge pixels. If the edge map isolates the object into small fragments, a poor random initialization can trap a centroid in a small, disconnected region, preventing the algorithm from discovering the true object shape.

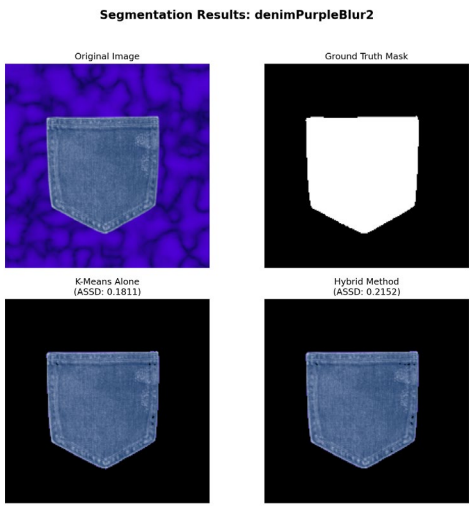


Figure 2: Low contrast between object and background.

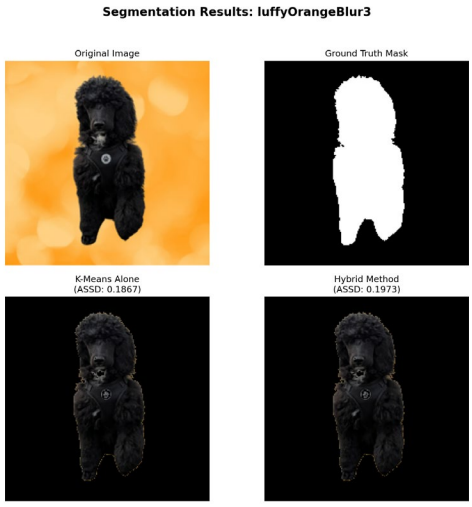


Figure 3: High contrast between the object (with internal texture) and background.

7. Conclusion

The objective of this research was to determine if integrating Canny edge detection would significantly improve the segmentation accuracy of k-means clustering. The results indicate that the hybrid method did not produce a statistically significant improvement ($p = 0.25$) over the spatially aware k-means algorithm. Therefore, the alternative hypothesis is rejected.

While both algorithms achieved comparable accuracy, the spatially aware k-means algorithm is identified as the optimal engineering solution for this task. It delivers equivalent segmentation performance while being approximately four times faster (0.08s vs 0.35s) and computationally less complex than the hybrid approach. The spatial weight parameter in the 5D feature vector successfully maintained object cohesion on its own, rendering the additional overhead of edge detection unnecessary for this specific dataset.

The limitations of the hybrid method were largely due to the use of fixed thresholds, which caused failures on objects with strong internal gradients. Future work should implement adaptive thresholding (such as Otsu's method [12]) to dynamically tune edge sensitivity for each image. Additionally, converting the dataset from RGB to the CIELAB color space [13] would allow the clustering algorithm to separate luminosity from chromaticity, potentially improving performance in images with strong shadows or highlights. Finally, the edge information could be utilized to optimize k-means initialization by restricting starting centroids to non-edge regions, ensuring the algorithm begins with “guesses” located firmly within the object's interior.

8. Acknowledgements

The conceptual basis for the hybrid k-means + Canny approach was developed through brainstorming sessions with Google's Gemini large language model.

References:

[1] E. Kavlakoglu and V. Winland, "What is k-means clustering?," IBM, Jun. 26, 2024. [Online]. Available: https://www.ibm.com/think/topics/k-means-clustering.

[2] R. Achanta *et al.*, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012.

[3] N. Kehtarnavaz and N. Kim, *Embedded Image Processing on the TMS320C6000 DSP: Examples in Code Composer Studio and MATLAB*. New York, NY: Springer, 2005.

[4] S. Jardim *et al.*, "Graphical image region extraction with k-means clustering and watershed," *J. Imag.*, vol. 8, no. 6, p. 163, 2022.

[5] D. E. Ilea and P. F. Whelan, "Color Image Segmentation Using a Spatial k-means Clustering Algorithm," in *Proc. 2011 7th Int. Symp. Appl. Comput. Intell. Inform. (SACI)*, Timisoara, Romania, 2011, pp. 275-280.

[6] A. Sharda, "Image filters — Gaussian blur," Medium, Aug. 2021. [Online]. Available: https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1.

[7] S. Sahir, "Canny edge detection: Step-by-step in Python — computer vision," Medium, Jan. 25, 2019. [Online]. Available: https://medium.com/data-science/canny-edge-

detection-step-by-step-in-python-computer-vision-b49c3a2d8123.

[8] S. Phanse, "Geometric Computer Vision Part 1: Canny Edge Detection in Python," Medium, Apr. 26, 2024. [Online]. Available: https://medium.com/@soham.phanse/canny-edge-detection-in-python-4cbc1209adbc.

[9] J. Ke *et al.*, "Deep learning-based approach for the automatic segmentation of adult and pediatric temporal bone computed tomography images," *Quant. Imaging Med. Surg.*, vol. 13, no. 3, pp. 1577–1591, Mar. 2023, doi: 10.21037/qims-22-658.

[10] V. Yeghiazaryan and I. Voiculescu, "Family of boundary overlap metrics for the evaluation of medical image segmentation," *J. Med. Imag.*, vol. 5, no. 1, p. 015006, 2018.

[11] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[12] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst. Man Cybern.*, vol. 9, no. 1, pp. 62–66, 1979.

[13] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed. Cham, Switzerland: Springer, 2022.

A QUANTUM ALGORITHM FOR RECOGNIZING THE LANGUAGE $L_= = \{ w \in \{0,1\}^* \mid |w|_0 = |w|_1 \}$

Andrew Elko
Millersville University of Pennsylvania, Computer Sciences Department
anelko@millersville.edu
Jingnan Xie
Millersville University of Pennsylvania, Computer Sciences Department
jingnan.xie@millersville.edu

ABSTRACT

Two-way finite automata with quantum and classical states (2QCFA), introduced by Ambainis and Watrous, can be viewed as two-way deterministic finite automata equipped with a constant-size quantum register. Zheng, Qiu, and Li later showed that the language $\{x\#y \mid x, y \in \{0,1\}^*, |x| = |y|\}$ is recognizable by 2QCFA. In this paper, we prove that the language $\{ w \in \{0,1\}^* \mid |w|_0 = |w|_1 \}$, that is, the set of all binary strings containing an equal number of 0s and 1s, can also be recognized by a 2QCFA. Further, we show that this quantum algorithm only needs 4 qubits, with the time complexity of $O(n^4)$.

1 Introduction

Quantum computing has produced striking breakthroughs, yet the most celebrated algorithms, such as Shor's and Grover's, rely on general quantum Turing machines. These machines require a large number of quantum bits (qubits) and long coherence times, which makes them difficult to realize with current technology. This motivates the study of simpler quantum computational models that still capture certain quantum advantages while remaining closer to what is feasible today. One such direction is the study of quantum finite automata (QFA), the quantum counterpart of classical finite automata [1; 2; 3; 4].

Among QFA, one-way models (1QFA), introduced by Kondacs and Watrous [5] and Moore and Crutchfield [6], represent the simplest case. However, their computational power is severely limited, as they recognize only a proper subset of the regular languages. Aharonov and colleagues introduced one-way quantum automata with mixed states [7], but these recognize exactly the class of regular languages and thus offer no gain over classical automata. Two-way QFA (2QFA) [5] overcome these barriers by recognizing all regular languages and even some nonregular ones. Yet, their implementation requires superpositions of head positions, demanding at least $O(\log n)$ qubits to store the head location for inputs of length n, which is technologically expensive.

To overcome these limitations, Ambainis and Watrous [8] introduced two-way finite automata with quantum and classical states (2QCFA). A 2QCFA can be viewed as a two-way deterministic finite automaton (DFA) augmented by a constant-size quantum register. This hybrid design is significantly more implementable, while still being more powerful than classical finite automata. For example, Ambainis and Watrous showed that palindromes can be recognized by a 2QCFA using only a single qubit, and subsequent work by Zheng, Qiu, and Li demonstrated that the language $\{x\#y \mid x, y \in \{0,1\}^*, |x| = |y|\}$ is also recognizable by a 2QCFA. Later extensions, such as two-way k-tape QCFA (kTQCFA) [9], further enriched the model by allowing multiple input tapes.

In this paper, we study the computational power of 2QCFA for recognizing the language $\{ w \in \{0,1\}^* \mid |w|_0 = |w|_1 \}$, that is, the set of all binary strings containing an equal number of 0s and 1s. This language is fundamental in formal language theory, as it represents a basic global counting property: membership depends on comparing the total occurrences of two symbols across the entire input rather than on local patterns. It is well known to be nonregular and therefore cannot be recognized by classical finite automata with bounded memory. Such equal-count constraints naturally arise in practical scenarios involving balanced resources, parity checks, or conservation principles. By showing that this language can be recognized by a 2QCFA with a constant-size quantum register, we demonstrate that a hybrid quantum–classical model can verify a nonregular global property without increasing classical memory, thereby highlighting the advantage of quantum interference in processing aggregate information.

2 Definitions and Notations

In this section, we give the definitions of 2QCFA. Several preliminary definitions and notations are also explained. The reader is referred to [10] for all unexplained notations and terminologies in language theory.

We use λ to denote the empty string, and $\emptyset$ to denote the empty set. For a set Q, we denote its power set by 2^Q,

i.e., the set of all subsets of Q. We use $L_=$ to denote the language$\{ w \in \{0,1\}^* \mid |w|_0 = |w|_1 \}$. For a random event E, $\Pr[E]$ denotes the probability that E occurs.

For an overview of quantum computing, the reader is referred to [11; 12; 13].

Definition 1. A *two-way finite automata with quantum and classical states* is defined by a 9-tuple

$$M = (Q, S, \Sigma, \Theta, \delta, q_0, s_0, S_{\rm acc}, S_{\rm rej}),$$

where

- Q is a finite set of quantum states;
- S is a finite set of classical states;
- Σ is a finite set of input symbols. The tape symbol set is $\Gamma = \Sigma \cup \{\mathfrak{c}, \$\}$, where $\mathfrak{c} \notin \Sigma$ is called the left endmarker and $\$ \notin \Sigma$ is called the right end marker;
- $q_0 \in Q$ is the initial quantum state;
- $s_0 \in S$ is the initial classical state;
- $S_{\rm acc} \subseteq S$ and $S_{\rm rej} \subseteq S$ are the sets of accepting and rejecting classical states, respectively;
- Θ is the quantum transition function:
  $\Theta : (S/(S_{\rm acc} \cup S_{\rm rej})) \times \Gamma \longrightarrow \mathcal{U}(\mathcal{H}(Q)) \cup \mathcal{M}(\mathcal{H}(Q))$,
  where $\mathcal{U}(\mathcal{H}(Q))$ and $\mathcal{M}(\mathcal{H}(Q))$ denote the sets of unitary operators and projective measurements, respectively, on the Hilbert space $\mathcal{H}(Q)$ whose basis is identified with Q. Thus, for $(s,\gamma) \in (S/(S_{\rm acc} \cup S_{\rm rej})) \times \Gamma$, the value $\Theta(s,\gamma)$ is either a unitary transformation or a projective measurement;
- δ is the classical transition function. If $\Theta(s,\gamma) \in \mathcal{U}(\mathcal{H}(Q))$, then
  $\delta : (S/(S_{\rm acc} \cup S_{\rm rej})) \times \Gamma \longrightarrow S \times \{-1,0,1\}$,
  where $D = \{-1,0,1\}$, representing movements of the head on tape T to the left, stationary, and right, respectively. We require that at most one of the D is non-stationary, which means that a 2QCFA can move at most one tape head and scan at most one new symbol at a time. The mapping is interpreted as follows:
  $$\delta(s,\gamma) \in (s', d)(r_i)$$
  means that if the automaton is in classical state s scanning symbol $\gamma \in \Gamma$, then it may deterministically choose to enter state s', while the head on tape T moves according to d.
  If $\Theta(s,\gamma) \in \mathcal{M}(\mathcal{H}(Q))$ and the set of possible measurement outcomes is $R = \{r_1, r_2, \ldots, r_n\}$, then
  $\delta : (S/(S_{\rm acc} \cup S_{\rm rej})) \times \Gamma \times R \longrightarrow S \times \{-1,0,1\}$,
  Here,
  $$\delta(s,\gamma)(r_i) = (s', d)$$
  means that if the automaton is in classical state $s \in S$ scanning symbol $\gamma \in \Gamma$, and the projective measurement yields outcome r_i (with probability p_i determined by the postulates of quantum mechanics), then it may deterministically choose to change to state s', while the head on the tape T moves according to d.

For a 2QCFA, a computation is assumed to halt if and only if there exists at least one computation path in which an accepting or a rejecting state can be entered. Let $L \subseteq \Sigma^*$ and $0 \leq \epsilon \leq \frac{1}{2}$. A 2QCFA M recognizes L with *one-sided error* ϵ if

- for every $w \in L$, $\Pr[M \text{ accepts } w] = 1$, and
- for every $w \notin L$, $\Pr[M \text{ rejects } w] \geq 1 - \epsilon$.

3 A 2QCFA accepting $L_=$

In this section, we prove that $L_=$ can be recognized by a 2QCFA with one-sided error. Our proof is inspired by the techniques in [8; 14], where several languages were shown to be recognizable by 2QCNFA(2). The proof for a 2QCFA accpeting $L_=$ is similar.

Theorem 3.1. Let $L_= = \{ w \in \{0,1\}^* \mid |w|_0 = |w|_1 \}$. For any $\epsilon > 0$, there exists a 2QCFA D such that for any input string w: if $w \in \{0,1\}^*$, then D accepts w with certainty; if $w \notin L_=$, then D accepts w with probability at most ϵ and rejects w otherwise.

Proof. Let U_a and U_b be defined as follows:

$$U_a = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & \cos\alpha & \sin\alpha \\ 0 & 0 & -\sin\alpha & \cos\alpha \end{pmatrix}, \qquad U_b = \begin{pmatrix} 0&0&1&0 \\ 0&0&0&1 \\ 1&0&0&0 \\ 0&1&0&0 \end{pmatrix},$$

where $\alpha = \sqrt{2}\pi$. It is straightforward to verify that both U_a and U_b are unitary. Moreover,

$$(U_a)^k = \begin{pmatrix} \cos(k\alpha) & -\sin(k\alpha) & 0 & 0 \\ \sin(k\alpha) & \cos(k\alpha) & 0 & 0 \\ 0 & 0 & \cos(k\alpha) & \sin(k\alpha) \\ 0 & 0 & -\sin(k\alpha) & \cos(k\alpha) \end{pmatrix},$$

and

$$(U_a)^m U_b (U_a)^n =$$

$$\begin{pmatrix} 0 & 0 & \cos((n-m)\alpha) & \sin((n-m)\alpha) \\ 0 & 0 & -\sin((n-m)\alpha) & \cos((n-m)\alpha) \\ \cos((n-m)\alpha) & -\sin((n-m)\alpha) & 0 & 0 \\ \sin((n-m)\alpha) & \cos((n-m)\alpha) & 0 & 0 \end{pmatrix}.$$

We construct a 2QCFA D with four quantum basis states $\{|q_0\rangle, |q_1\rangle, |q_2\rangle, |q_3\rangle\}$, where $|q_0\rangle$ is the initial state. The unitary operators U_a and U_b act as follows (here $\alpha = \sqrt{2}\pi$):

$$\begin{aligned} U_a |q_0\rangle &= \cos\alpha\ |q_0\rangle + \sin\alpha\ |q_1\rangle, & U_b |q_0\rangle &= |q_2\rangle, \\ U_a |q_1\rangle &= -\sin\alpha\ |q_0\rangle + \cos\alpha\ |q_1\rangle, & U_b |q_1\rangle &= |q_3\rangle, \\ U_a |q_2\rangle &= \cos\alpha\ |q_2\rangle - \sin\alpha\ |q_3\rangle, & U_b |q_2\rangle &= |q_0\rangle, \\ U_a |q_3\rangle &= \sin\alpha\ |q_2\rangle + \cos\alpha\ |q_3\rangle, & U_b |q_3\rangle &= |q_1\rangle. \end{aligned}$$

The behavior of the 2QCFA D is illustrated in Figure 1.

Lemma 3.1. For any input w in the form of $L_=$, if $|w|_0 = |w|_1$, then after Step 5(b) of Figure 1, the quantum state of D will reach $|q_2\rangle$ with certainty.

For any input $w \in \{0,1\}^*$, repeat the following procedure ad infinitum:

1. Let w be the input string on the tape T. Move the tape head to the leftmost marker ¢.
2. Set the quantum state to $|q_0\rangle$.
   (a) Move the tape head of T one square to the right.
3. While the currently scanned symbol of T is not \$, do:
   (a) If the currently scanned symbol of T is 0, perform U_a on the quantum state.
   (b) Move the tape head of T one square to the right.
4. When the currently scanned symbol of T is \$, perform U_b on the quantum state.
   (a) Move the tape head of T one square to the left.
5. While the currently scanned symbol of T is not ¢, do:
   (a) If the currently scanned symbol of T is 1, perform U_a on the quantum state.
   (b) Move the tape head of T one square to the left.
6. Measure the quantum state. If the quantum state is not $|q_2\rangle$, reject.
7. Repeat the following 2 times:
   (a) Move the tape head of T to the first input symbol.
   (b) Move the tape head of T one square to the right.
   (c) While the currently scanned symbol is not ¢ or \$, simulate a coin flip:
      i. If the result is "heads", move to the right.
      ii. If the result is "tails", move to the left.
8. If both times the process ends at the right end-marker \$, simulate k coin-flips:
   (a) If all results are "heads", accept.

Figure 1: A *2QCFA* for $L_=$

Lemma 3.2. For any input $w \in \{0,1\}^*$, let $n = |w|$, $u = |w|_0$, and $v = |w|_1$. If $u \neq v$, then D rejects after Step 6 of Figure 1 with probability at least

$$\frac{1}{2(u-v)^2+1}.$$

Proof. For any input w in the form of $L_=$, since we only measure the quantum state of D after step 5(b) when the classical state is s_2. So, for each measurement, the quantum state must reach the form

$$|q\rangle = (U_a)^v U_b (U_a)^u |q_0\rangle$$

$$= \cos\big(\sqrt{2}\,(u-v)\pi\big)\,|q_2\rangle + \sin\big(\sqrt{2}\,(u-v)\pi\big)\,|q_3\rangle.$$

Hence, the probability of observing $|q_3\rangle$ after a single measurement of Step 6 is

$$\sin^2\big(\sqrt{2}\,(u-v)\pi\big).$$

We know that $|v-u| > 0$. Let l be the closest integer to $\sqrt{2}\,(v-u)$. We aim to show that

$$\sin^2\big(\sqrt{2}\,(u-v)\pi\big) \;>\; \frac{1}{2(u-v)^2+1}.$$

Here, we only prove the case when $\sqrt{2}\,(v-u) > l$; the case $\sqrt{2}\,(v-u) < l$ is analogous. If $\sqrt{2}(v-u) > l$, then $2(v-u)^2 - 1 \geq l^2$, so that $l \leq \sqrt{2(v-u)^2-1}$.

Hence,

$$\begin{aligned}\sqrt{2}(v-u) - l \;&\geq\; \sqrt{2}(v-u) - \sqrt{2(v-u)^2-1} \\ &=\; \frac{1}{\sqrt{2}(v-u) + \sqrt{2(v-u)^2-1}} \\ &>\; \frac{1}{2\sqrt{2}(v-u)}.\end{aligned}$$

Because l is the closest integer to $\sqrt{2}\,(v-u)$, we have $0 < \delta < \frac{1}{2}$, where

$$\delta \;=\; \sqrt{2}\,(v-u) - l.$$

Define $f(t) = \sin(\pi t) - 2t$. On $[0, \frac{1}{2}]$,

$$f''(t) = -\pi^2 \sin(\pi t) \leq 0, \qquad f(0) = 0, \quad f\big(\tfrac{1}{2}\big) = 1-1 = 0,$$

so f is concave with zeros at the endpoints, hence $f(t) \geq 0$ on $[0, \frac{1}{2}]$. Therefore,

$$\sin(\pi t) \;\geq\; 2t \quad \text{for } t \in [0, \tfrac{1}{2}].$$

Taking $t = \delta$ and using the bound $\delta > \dfrac{1}{2\sqrt{2}(v-u)}$ obtained above, we get

$$\begin{aligned}\sin^2\big(\sqrt{2}(u-v)\pi\big) = \sin^2\big((\sqrt{2}(v-u) - l)\pi\big) &= \sin^2(\pi\delta) \\ &> \left(\frac{2}{2\sqrt{2}(u-v)}\right)^2 = \frac{1}{2(u-v)^2} \\ &> \frac{1}{2(u-v)^2+1}.\end{aligned}$$

□

Lemma 3.3. The acceptance probability after step 5(b) is

$$\frac{1}{2^k(n+1)^2}$$

where n is the input length.

Proof. Step 7 performs a random walk twice, starting from position 1 and aiming to reach position $n+1$ (the right-end marker \$). From probability theory, it is well known that the probability of reaching position $n+1$ is $\frac{1}{n+1}$. Step 8 performs k independent coin flips, each yielding "head" with probability $\frac{1}{2}$. □

The remainder of the proof follows similarly to the argument in [8], which demonstrates the existence of a 2QCFA that

accepts the set of all palindromes. By the lemmas above, we conclude that for any input w, the reject probability after step 6 is

$$P_{\text{rej}} = 0 \quad \text{if } w \in L_{=},$$

and otherwise

$$P_{\text{rej}} > \frac{1}{2(u-v)^2+1}.$$

The acceptance probability after step 8 is

$$P_{\text{acc}} = \frac{1}{2^k(n+1)^2}.$$

This sequence of steps is repeated indefinitely, causing D to eventually reject with probability

$$\Pr[D \text{ rejects } w] = \sum_{j\geq 0}(1-P_{\text{acc}})^j(1-P_{\text{rej}})^j P_{\text{rej}} \quad (1)$$

$$= \frac{P_{\text{rej}}}{P_{\text{acc}}+P_{\text{rej}}-P_{\text{acc}}P_{\text{rej}}}. \quad (2)$$

and accept with probability

$$\Pr[D \text{ accepts } w] = \sum_{j\geq 0}(1-P_{\text{acc}})^j(1-P_{\text{rej}})^{j+1} P_{\text{acc}} \quad (3)$$

$$= \frac{P_{\text{acc}}-P_{\text{acc}}P_{\text{rej}}}{P_{\text{acc}}+P_{\text{rej}}-P_{\text{acc}}P_{\text{rej}}}. \quad (4)$$

These probabilities sum to 1, and the probability of acceptance is therefore 1 when $w \in L_{=}$. If $w \in L_{=}$, the $\Pr[D \text{ rejects } w] = 0$ because if w is in the language the $P_{\text{rej}} = 0$, causing the probability to become 0, and the acceptance probability $P_{\text{acc}} = 1$. Otherwise:

Let $k = 1 + \lfloor \log_2 \frac{1}{\epsilon} \rfloor$, so that $\epsilon \geq \frac{1}{2^{k-1}}$. So,

$$P_{\text{acc}} \leq \frac{\epsilon}{2(n+1)^2}.$$

Hence,

$$\Pr[D \text{ rejects } w] = \frac{P_{\text{rej}}}{P_{\text{acc}}+P_{\text{rej}}-P_{\text{acc}}P_{\text{rej}}} \quad (5)$$

$$> \frac{P_{\text{rej}}}{P_{\text{acc}}+P_{\text{rej}}} \quad (6)$$

$$> 1-\epsilon. \quad (7)$$

□

If we assume that the input $w = L_{=}$ where $|w|_0 = |w|_1$, then step 3 and 5 take $O(n)$ time exactly, and loops 7 and 8 take $O(n^2)$ time. The expected number of repeating the algorithm is $O(n^2)$. Hence, the expected time complexity of the algorithm is $O(n^4)$.

4 Conclusion

In this paper, we investigated the computational capabilities of two–way finite automata with quantum and classical states (2QCFA). We presented a concrete construction, using 4 qubits, demonstrating that the language

$$L_{=} = \{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$$

can be recognized by a 2QCFA with one–sided error. The construction relies on carefully designed unitary transformations that encode the difference between the number of 0s and 1s in the input into the quantum state, allowing the automaton to distinguish members and non-members of the language with arbitrarily small error probability. This construction operates at a time complexity of $O(n^4)$.

This result illustrates the usefulness of hybrid quantum–classical models of computation. Even with a very small quantum component and finite classical control, a 2QCFA can recognize nontrivial languages using probabilistic and quantum interference techniques. Our proof also highlights how simple unitary operations and repeated classical control flow can be combined to amplify correctness probabilities.

References

[1] A. Gainutdinova, A. Yakaryılmaz, Unary probabilistic and quantum automata on promise problems, *Quantum Information Processing*, *17*(28), doi:https://doi.org/10.1007/s11128-017-1799-0.

[2] C. Piazza, R. Romanello, Mirrors and memory in quantum automata, in E. Ábrahám, M. Paolieri, editors, *Quantitative Evaluation of Systems. QEST 2022*, volume 13479 of *Lecture Notes in Computer Science* (Springer, Cham, 2022), doi:https://doi.org/10.1007/978-3-031-16336-4_18.

[3] A. S. Bhatia, A. Kumar, On the power of two-way multihead quantum finite automata, *RAIRO - Theoretical Informatics and Applications*, *53*(1-2):(2019), 19–35, doi:https://doi.org/10.1051/ita/2018020.

[4] D. Ganguly, K. Chatterjee, K. S. Ray, Watson–Crick quantum finite automata, *Acta Informatica*, *58*:(2021), 231–240, doi:https://doi.org/10.1007/s00236-020-00370-x.

[5] J. Watrous, A. Kondacs, On the power of quantum finite state automata, in *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, page 66 (IEEE Computer Society, Los Alamitos, CA, USA, 1997), ISBN 0272-5428, doi:https://doi.org/10.1109/SFCS.1997.646094.

[6] C. Moore, J. P. Crutchfield, Quantum automata and quantum grammars, *Theoretical Computer Science*, *237*(1):(2000), 275–306, ISSN 0304-3975, doi:https://doi.org/10.1016/S0304-3975(98)00191-1.

[7] D. Aharonov, A. Kitaev, N. Nisan, Quantum circuits with mixed states, in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 20–30 (ACM, 1998), doi:https://doi.org/10.1145/276698.276708.

[8] A. Ambainis, J. Watrous, Two-way finite automata with quantum and classical states, *Theoretical Computer Science*, *287*(1):(2002), 299–311, ISSN 0304-3975, doi: https://doi.org/10.1016/S0304-3975(02)00138-X, natural Computing.

[9] S. Zheng, L. Li, D. Qiu, Two-tape finite automata with quantum and classical states, *International Journal of Theoretical Physics*, *50*(4):(2011), 1262–1281, ISSN 1572-9575, doi:https://doi.org/10.1007/s10773-010-0582-0.

[10] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA., 1979).

[11] J. Xie, Quantum computing unveiled: A practical approach for computer science students, *Journal of Computing Sciences in Colleges*, *41*(3):, 189–203, doi:https://dl.acm.org/doi/10.5555/3801163.3801220.

[12] J. Gruska, et al., *Quantum computing*, volume 2005 (McGraw-Hill London, 1999).

[13] M. A. Nielsen, I. L. Chuang, *Quantum computation and quantum information* (Cambridge university press, 2010).

[14] S. ZHENG, D. QIU, L. LI, Some languages recognized by two-way finite automata with quantum and classical states, *International Journal of Foundations of Computer Science*, *23*(05):(2012), 1117–1129, doi: https://doi.org/10.1142/S0129054112500141.

DISTRIBUTED GRID-BASED COORDINATION FOR MULTI-ROBOT PATH PLANNING

Malachi Watson, Adrian Rublein
Lycoming College
watmala1@lycoming.edu, rublein@lycoming.edu

ABSTRACT
Robots are frequently used to tackle "find-and-retrieve" problems like warehouse packing and search-and-rescue operations. Employing multiple robots on the same problem requires coordination of both information and movement. While the most optimal solutions rely on a centralized camera view for coordination, this is not always a realistic prerequisite depending on budget and environment. We propose a grid-based coordination system that relies on distributed robot communication to search an unexplored area, thereby minimizing human intervention. Through Java simulations, we demonstrate that our multi-robot approach results in an average task completion time 18-23% faster than can be achieved by a single robot.

KEY WORDS
Robotic path-planning; multi-robot coordination; distributed systems; heuristics.

1. Introduction

Robots are frequently employed in search-and-rescue operations to aid human first responders while minimizing the risk to human life [1]. Some hardware designs, such as snakes, allow the robots to navigate terrain that a human could not. However, even when the terrain is technically traversable on foot, it can be unstable or contaminated, necessitating the use of robots to preserve the safety of the human first responders. The time of each human is valuable as well, as the less time a human spends on robot-assignable tasks, the more time they can spend administering first aid or communicating logistics with the rest of the rescue team. Therefore, any robot that is tasked with searching for victims should do so in a way that minimizes the time the robot operator must spend servicing the system.

Using robots for warehouse picking provides a similar problem with fiscal consequences instead of vital ones. Online shopping is now an expected part of society that carries with it the expectation of reliability and reasonable speed. After items are distributed to large warehouses across the country, individual orders must be fulfilled through finding various items across the warehouse. Items are located by first dividing the warehouse floor into zones, then assigning robots to particular zones (and routing their paths) in a way that minimizes travel distance and time. This picking process comprises about half of all warehouse operations [2], so if picking robots can be routed more efficiently, then the company and its customers will benefit from faster order fulfillment times.

Both of these applications depend on solving the Traveling Salesman Problem (TSP) using multiple agents. The TSP is known to be NP-Hard given that the Hamiltonian path problem is proven to be NP-Complete [3]. This necessitates the development of one or more heuristic methods to solve the TSP in various environments. The TSP also depends on a reliable method to find the shortest path in a given set of nodes, for which A* search [4] is commonly used.

When considering the assignment of targets to robots, the system can either be centralized or distributed. In a centralized system, some primary entity can access all information collected by all agents; this entity then uses that information to build a complete model of the current environment state. Developing algorithms for this model often results in a close-to-optimal solution. However, for multi-robot coordination, this model is sometimes unrealistic (consider a natural disaster in a remote location with no cell service) and contains a central point of failure; if the central computer fails, then the agents dependent on its direction are lost. Distributed systems are designed to avoid these problems by supplying each agent with an algorithm that learns locally. If one agent fails, the others can keep operating as usual; and agents can still learn from each other using local communication.

In this work, we propose a distributed coordination system for multiple robots to plan their paths in a grid environment. Each robot maintains its own map as it explores unknown sections of the grid map, which it can

opportunistically share with nearby robots using RFID technology [5]. In Section 3, we thoroughly describe this system for both the single- and multi-robot cases. We compare the results of these cases in Section 4 using Java simulations. This work was completed as part of a semester-long independent study on multi-robot systems.

2. Related Works

Search-and-rescue scenarios usually involve multiple robots searching for multiple targets or victims. Smaller robots can be modeled as swarms in particle swarm optimization, though this relies on the targets broadcasting some sort of signal (auditory, visual, or thermal) to attract the bots' attention [6]. If the group is composed of robots with differing capabilities (heterogeneity), then some areas may be reachable by one bot but not the other. Thus, developing a reachability map for each bot can be useful, as in Pereira et al. [7], where the goal is to explore/traverse the entire map instead of looking for specific targets.

The process of warehouse picking also consists of more than one subproblem. Some work focuses on distributing tasks among a robot fleet, abstracting away many details about how each of those tasks is completed [8]. Others [2] address the location and similarity of picking orders in order to increase the efficiency of the order-packing stations.

Some prior works focus on a centralized system for planning multiple robot paths; the bird's-eye view often results in a more optimal solution. Luna and Bekris [9] propose a graph-based system that models potential robot locations as graph nodes and focuses on resolving potential collisions through swapping the locations of robots. This solution performs as well as A*-related techniques, though is more suited to denser environments with a higher chance of collisions between robots. Another system optimizes the allocation of various search-and-rescue targets among a fleet of bots, reallocating tasks should one or more bots be destroyed due to hazardous conditions [10].

Due to space and resource constraints, a centralized bird's-eye view is frequently infeasible. Therefore, much work has been done towards solving these problems in a distributed fashion. Claes et al. [11] propose using Monte Carlo Tree Search for assigning warehouse picking tasks (that is, assigning goal locations) dynamically to a fleet of robots, simulating the continuous arrival of picking orders. In search-and-rescue, the searchable area is usually unknown or obscured; Chen et al. account for this by first using a variant of Lloyd's algorithm to divide the area into cells, then directing each bot to travel towards the frontier of each cell for maximum coverage [12]. This work focuses on tracking moving targets rather than establishing a fastest route to a stationary target. Other work [13] centers around distributed inter-robot communication: neighboring robots within a certain distance coordinate their planned routes to avoid collisions while traveling between a series of stations. However, the locations of borders and obstacles are assumed to be known ahead of time.

3. System Overview

Locations of robots and targets are mapped on a grid layout. Each robot completes its task by executing two guaranteed phases: exploration and base returning. There are other possible phases as well to aid in inter-robot communication. To find neighbors more quickly, a pinging system is introduced such that robots can purposefully approach each other for near-field exchange of map data.

3.1 Grid System

To model this problem in a simulation setting, the real-world environment must first be discretized. The explorable area (assumed to be finite) is first broken down into a grid of tile squares with X and Y coordinates [14]. The total dimensions of the area can be assumed to be known ahead of time, either through the dimensions of the warehouse or the human-defined boundaries of a disaster area. The size of each tile is a little larger than the size of the robot (that is, if the bot is about 2 feet in diameter, then each tile is 3 feet on each side). The advantage of choosing this dimension is that each tile can only ever be occupied by one bot, and two bots can pass next to each other without colliding. Each tile is also either traversable or non-traversable. Traversable tiles are an open path where a robot can safely move about, and non-traversable tiles represent obstacles.

3.2 Robotic States

In order for individual robots to succeed in grid traversal target finding, and cooperation, they must be able to alter their behavior for the corresponding task. We model this through different "states" the robot can exist in. Each robot can exist in one of the following states at a time: exploration, base returning, target approaching, robot approaching, or waiting. The robot approaching and waiting states are involved solely in the communication between different robots. The exploration and retrieval logic is developed in the remaining states. Each robot in the system also maintains an individual copy of the grid system they are traversing. This grid begins empty and is populated and timestamped with each node the robot moves to. This map mainly receives information during the exploration state.

3.2.1 Exploration

Each robot uses a depth-first approach to exploring unmapped areas [15]. Initially, the placement of each robot's target is unknown to it, so there is no direct path to guide a robot's basic exploration. The exploration logic is based on a visited list and a stack of adjacent nodes. An adjacent node is popped off the stack and added to the visited list, and the neighboring nodes of the most recent one are added to the adjacent stack. This has a time complexity of approximately O(n), where n is the total number of squares in the grid, since the general worst-case scenario is for a robot to search the entire grid to find its target. This is a loose approximation, as due to the physical nature of the robots, a robot must occasionally backtrack through previously explored nodes to reach unexplored nodes.

3.2.2 Base Returning

Each individual robot has a state dedicated towards the traversal back to their starting position after finding the target. This serves to model search-and-rescue scenarios where the machine must report back the position of an item, or warehouse applications where they must grab the target itself and bring it back. This logic employs the A* algorithm, where a heuristic is used to determine the shortest path between two points. Each robot finds the shortest path they can take using their personal map of known nodes.

3.2.3 Target Approaching

During the course of the grid traversal, robots are designed to merge map data with other robots. Consequently, merging map data will occasionally result in the revelation of a target's location that was previously unknown. When this is the case, the robot will enter this target approaching state, finding the shortest path to the target using its updated personal map of the grid. This logic also implements the A* algorithm as the known position of the target supplies the heuristic.

3.3 Single-Robot Traveling Algorithm

Each robot employs the A* algorithm in order to find a quick route to its target when the target's location is known. This algorithm has utility in our model as it is flexible (allowing for different heuristic goals) and works well with obstacles like the ones accounted for in our model. Although A* typically requires larger memory storage [16], this algorithm still proves most optimal in scenarios where the explored area is well-known, like in our model where map data is shared among robots and this algorithm is only used to traverse to known spots. The high modularity of this system and the reduced time required also make this algorithm more optimal for grid-based multi-robot coordination. Our system utilizes a Manhattan heuristic within the A* search, which is appropriate given the 4 directional mobility of the robots in our system.

3.4 Multi-bot Communication

A central aspect to our model is our distributed multi robot system of communication. We use a pinging system that allows robots to broadcast their location and specific ID to other robots in a specific radius. Once a viable robot is found, the broadcasting robot will stop exploring, and the receiving robot will maneuver the grid system to the broadcasted location. Once the two robots are on adjacent nodes, they can merge information. This involves the population of personal grids and the sharing of map information such as the placement of found targets. If a robot has found the target of another robot, it is stored in map memory; when a merge happens with another robot, this information is transferred.

This broadcasting logic for robots to approach each other is necessary for information to be transferred through RFID chips, which

utilize near-field communication. Although certain RFID chips exist for communication at slightly farther distances, most are only capable of reading or writing from a few centimeters away [5]. Thus, any two robots must be in very close proximity to communicate. This proximity is a requirement that does not naturally occur often enough in larger environments, as we discovered experimentally. To create these contact events between robots, then, the robots' behavior must be coordinated. The pinging system in our model allows any individual robot to orchestrate contact events naturally and without the need for a centralized system to plan them [17].

To measure the speed at which robots find their targets, we model the passage of time as discrete timesteps, with one tile of movement taking exactly one timestep to execute. This ensures accuracy when coordinating robots to approach each other.

3.4.1 Requirements for Ping Event

Robots have specified ping intervals that dictate how many movements are made before they broadcast a ping to the surrounding nodes. In Section 4, we found the length of these intervals to be very influential in the speed of goal completion. When a ping is sent, the broadcasting robot detects any other robots in the ping range. The closest one that has not exchanged maps with the broadcasting robot within several hundred timesteps, and has not yet found its target, is set to an approaching robot state. The broadcasting robot then stays on its square until the approaching robot has made contact with it.

3.4.2 Approaching and Waiting State

The waiting state is used exclusively by broadcasting robots, and it simply means that during a timestep, the robot does not move and rather waits to be reached by the approaching robot. This is necessary as both robots approaching at the same time allows for infinite loop scenarios, especially in regards to simultaneous maneuvering around obstacles. The approaching state is used by a robot when it has been pinged by another robot and is currently attempting to approach it. This logic uses the A* algorithm with the same Manhattan heuristic, although due to the unknown nature of the target, the robot must be able to traverse through nodes not already marked in their personal map. This is different than target or home approaching logic that only uses a subset of known nodes to travel. This system creates an imagined A* path from the robot to the target through unknown nodes; if the robot comes across an obstacle or a boundary, it recalculates the path with the new knowledge and continues attempting to approach. This uses the benefit of the heuristic provided by the known location while also accounting for the unknown nature of the environment. Once it reaches the broadcasting robot, the two bots merge information.

4. Evaluation

In this section, we evaluate the effectiveness of distributed multi-robot coordination against the single-robot case using simulations developed in Java.

4.1 Simulation Setup

We evaluate the effectiveness of our approach under various scenario settings (see Table 1). We chose to evaluate three distinct variables in order to see the use cases for which our system is most optimal. These variables are: the number of robots, the size of the grid, and the length of ping intervals. Modest numbers have been chosen for each to simulate a small-scale operation (of either the search-and-rescue or warehouse variety).

Variable	Settings
Number of Bots	1, 2, 5
Grid Sizes	10-by-10, 20-by-20
Ping Interval Length	10, 25, 50 timesteps

Table 1: Scenario Settings

4.1.1 Number of Robots

Varying the number of robots allows us to analyze how our multi-robot communication functionality works across different quantities. Distributed systems excel in their modularity and flexibility, especially with regards to the number of and generic nature of robots involved. Testing for varying numbers of robots ensures that this model is utilizing the benefits provided by its distributed nature.

4.1.2 Size of Grid

The size of the area being searched is a crucial variable in determining the scalability of this system. Issues such as lack of points of contact and efficiency in exploration are made more evident when running in large areas, such as warehouses or search-and-rescue landscapes.

4.1.3 Ping Interval

The interval at which a robot sends out broadcasts for contact has implications for the ratio of timesteps spent in communication states, such as "Waiting" or "Approaching Robot", as opposed to an "Exploration" state where new nodes are being discovered. This ratio is fundamental to how quickly each robot completes its goal, and therefore how fast the system as a whole runs. These pings are also set to broadcast to all robots that are within a certain radius of the broadcasting robot. This radius is set to be a quarter the size of the map (experimentally determined to provide a balanced tradeoff for communication and exploration states).

4.2 Simulation Results

The simulation results are each the result of ten separate simulation runs averaged together. For each of the two grid sizes, targets were randomized, but the boundaries were established to be the same for all 10-by-10 grid simulations. The same process was done for the 20-by-20 simulations.

Figure 1 displays an inverse relationship between the quantity of bots and the average amount of time steps needed for an individual robot to complete a task. We observed a 19% increase in speed by moving from one robot to two robots. We noticed a much smaller increase in speed by moving from 2 robots to 5, which resulted in a 3% increase. This large increase between 1 bot and 2 bots can be attributed to the lack of cooperative functionality in single robotic systems. A single robot has no neighbors to ping, so regardless of the ping interval, it never shares information with another robot. The introduction of a basic cooperative functionality into this system immediately speeds it up significantly. We also observed a slight increase in speed when the 25 timestep ping interval was used as opposed to the 10 or 50 timestep intervals. This implies that there exists a benefit of cooperation for a moderate ping interval - not so infrequent that time gains are minimal (50), but also not so frequent that too much time is spent in an exploring state (10).

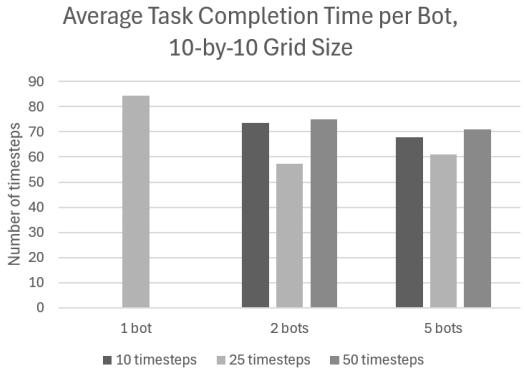


Figure 1: Average time for each robot to find its target and return to home base on a grid with 10 tiles on each side (for the 1-, 2-, and 5-bot cases).

For the 20-by-20 grid size (Figure 2), there is a similar trend to the 10-by-10 case. An increase in the number of bots increases the speed of task completion, and we see the same increase in speed for a medium-length ping interval of 25. We also see a higher increase in speed from the 2-bot simulations to the 5-bot ones as compared to Figure 1. The 20-by-20 simulations display a 23% increase in speed from 1 robot to 2. We also see an 18% increase from 2 robots to 5. This is in contrast to the previous 18% and 3% respectively. This data implies that increases in the number of cooperating robots is most effective in larger grids that have more unexplored areas.

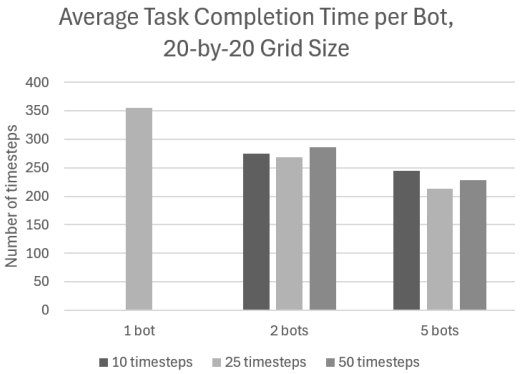


Figure 2: Average time for each robot to find its target and return to home base on a grid with 20 tiles on each side (for the 1-, 2-, and 5-bot cases).

Figure 3 displays the difference between average task completion times and final task completion times. Average task completion is calculated by taking the timestamps of when each robot finished their task and averaging them together to find the average time of task completion. Final task completion is the time the final robot finished its task, resulting in the system as a whole being completed. The average task completion times were used for Figure 1 and Figure 2 due to the effect of outliers on final task completion times (especially since the last robot can lose the help of neighbors in locating its target). In Figure 3, the difference between the average time and final time is stark: the last robot takes 50-100% more time to complete its task compared to the average case. In distributed systems where cooperation between robots is dispersed, the rate of incoming information can be randomly higher or lower for any given bot. Measuring the system by its slowest component doesn't allow us to see the effectiveness of the distributed system as a whole. The final task completion times are also not directly correlated with the number of bots; the final time for the 5-bot case was slightly lower than the 2-bot case (potentially due to more map exchanges with the greater number of bots on the field).

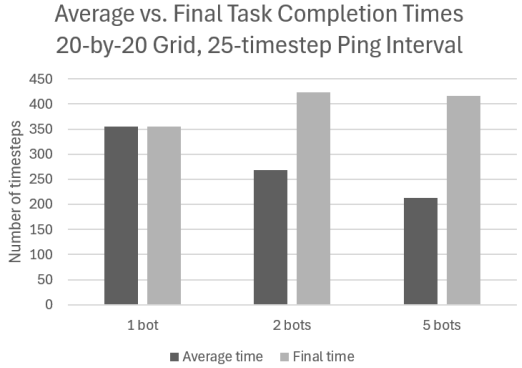


Figure 3: Comparison of average completion time per bot to the last completion time of the final bot to return to home base (on the 20-by-20 grid with a ping interval of 25 timesteps).

5. Conclusion

In this work, we propose a distributed multi-robot system to tackle search-and-retrieve problems (including warehouse packing and search-and-rescue) such that human intervention is minimized. Robots explore an unknown, finite area while periodically exchanging information with their neighbors to find their targets. Through Java simulations, distributed cooperation is shown to speed up task completion by 18-23% as compared to using a single robot for the same task. Future work will include experimenting with multiple robots working towards the same goal and implementing the system on physical robots. A deeper study will also be conducted on the effect of ping intervals on the speed of our system, especially optimizing this interval under a greater variety of grid layout scenarios.

References:

[1] V.S. Rajashekhar & G. Prabhakar, From preparedness to action: Synthesising insights on robot usage in post-earthquake search operations, *Resilient Cities and Structures, 5*(1), 2026, 60-70.

[2] S.-Y. Chou, A. N. Mauliddina, A. Dewabharata, & F. E. Zulvia, Pick order assignment and order batching strategy for robotic mobile fulfilment system warehouse, *Proc. of the Winter Simulation Conference WSC '23*, 2024, 1700–1711.

[3] R. M. Karp, *Reducibility among Combinatorial Problems* (Springer US, 1972).

[4] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (4th Edition)* (Pearson, 2020).

[5] F. Blatt and H. Szczerbicka, Realisation of navigation concepts for the multi-agent flood algorithm for search & rescue scenarios using RFID tags, *Proc. of the 20th International Symposium on Distributed Simulation and Real-Time Applications*, 2016, 112-115.

[6] A. S. Kumar, G. Manikutty, R. R. Bhavani, and M. S. Couceiro, Search and rescue operations using robotic Darwinian particle swarm optimization, *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, 1839-1843.

[7] T. Pereira, M. Veloso, and A. Moreira, Multi-robot planning using robot-dependent reachability maps, *Robot 2015: Second Iberian Robotics Conference*, 2016, 189-201.

[8] S. Hu, S. Fujimura, and Y. Feng, Task allocation optimization for warehouse autonomous mobile robot, *Proceedings of the 2024 International Conference on Innovation in Artificial Intelligence (ICIAI)*, 2024, 135-141.

[9] R. Luna and K. E. Bekris, Efficient and complete centralized multirobot path planning, *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, 3268-3275.

[10] K. Di, Y. Zhou, J. Jiang, F. Yan, S. Yang, & Y. Jiang, Risk-aware collection strategies for multirobot foraging in hazardous environments, *ACM Trans. Auton. Adapt. Syst., 16*, July 2022.

[11] D. Claes, F. Oliehoek, H. Baier, & K. Tuyls, Decentralised online planning for multi-robot warehouse commissioning, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2017, 492-500.

[12] J. Chen, J. Ma, & P. Dames, Distributed multi-robot multi-target simultaneous search and tracking in an unknown non-convex environment. www.arxiv.org/abs/2509.23308, 2025.

[13] A. Viseras, V. Karolj, and L. Merino, An asynchronous distributed constraint optimization approach to multi-robot path planning with complex constraints, *Proceedings of the Symposium on Applied Computing (SAC)*, 2017, 268-275.

[14] F. A. Blatt, *The Multi-Agent Flood Algorithm as an Autonomous System for Search and Rescue Applications* (Gottfried Wilhelm Leibniz Universitat Hannover, 2017).

[15] F. Cabrera-Mora, J. Xiao, and P. Brass, Multi-robot flooding algorithm for the exploration of unknown indoor environments, *2010 IEEE International Conference on Robotics and Automation*, 2010, 5478-5483.

[16] D. Foead, A. Ghifar, M. Kusuma, Budi, and H. Novita, A systematic literature review of A* pathfinding, *5th International Conference on Computer Science and Computational Intelligence*, 2020.

[17] M. Kegeleirs, G. Grisetti, and M. Birattari, Swarm slam: Challenges and perspectives, *National Library of Medicine*, 2017.

Economic Viability of Starlink for Developing Countries

Dierfield Frank*, William Planic*, Naresh Adhikari*
fad1003@sru.edu, wmp1002@sru.edu, naresh.adhikari@sru.edu
*Slippery Rock University

***Abstract*—The economic viability of SpaceX's Starlink broadband system when applied to developing countries is evaluated through an assessment of infrastructure requirements, long-term sustainability, and affordability. Starlink's low-Earth-orbit (LEO) satellite architecture can offer a potential alternative to mobile broadband networks and traditional fiber infrastructure, particularly in regions with geographic barriers that severely restrict connectivity. However, rising power demands, increasing hardware costs, recurring subscription costs, and accessibility challenges raise questions about the affordability of Starlink for low-income households. Cases of three countries, viz. Nepal, Venezuela, and Burkina Faso were analyzed to offer empirical insights into consumer affordability, adoption patterns, and Starlink's operational performance across a variety of socio-economic conditions.**

I. Introduction

Modern broadband connectivity is now a foundational requirement for digital inclusion and economic development. However, access to affordable and reliable internet still remains disproportionate across underdeveloped regions. The International Telecommunication Union (ITU) estimates that the total number of people who remain offline worldwide reached 2.6 billion [10]. ITU claims that the majority of this offline population resides in low- and middle-income countries, where household affordability is limited, and broadband infrastructure is underdeveloped [10]. Structural barriers also exist that impede the expansion of terrestrial and fiber networks. Some of these barriers include high deployment costs, limited national investment, and low population density in rural areas of the region. The World Bank notes that numerous regions in South Asia and Africa require fiber infrastructure that is likely to be up to 10 times more expensive than Starlink options. The return on investment is also significantly lower due to the drastically reduced purchasing power of the average consumer when compared to developed economies [22].

SpaceX's Starlink broadband system provides a modern, fundamentally different paradigm for connectivity. Instead of relying on cellular infrastructure, ground-based fiber, or microwave towers, Starlink utilizes a LEO satellite constellation [17]. This helps to deliver low-latency broadband service and high throughput. As of 2025, SpaceX has launched more than 10,000 satellites, with an estimated 8,800 still likely active in orbit. This spans the V1.0, V1.5, and V2 Mini generations included in the constellation [17]. Optical inter-satellite links are incorporated into the constellation along with orbital shells and array antennas. These features are designed to reduce latency and provide global coverage relative to traditional geostationary (GEO) systems. Public filings from SpaceX to the FCC indicate that beam reuse and capacity will be further expanded to support urban areas with a higher user density [19].

The appeal of LEO broadband in developing countries lies in its ability to navigate geographical infrastructure challenges. Countries such as South Sudan, Nigeria, Haiti, and Nepal face major geographic obstacles, including mountainous terrain, dense forests, and degradation caused by internal conflict. These obstacles slow terrestrial network deployment and significantly increase the total cost per user, thereby impacting affordability for the average consumer. ITU's Measuring Digital Development report concludes that 50% of rural Africans live beyond the scope of modern fiber infrastructure [10]. In turn, this leaves a large part of the African population dependent on congested 3G/4G networks or on costly fixed-wireless providers. To address this issue, Starlink's satellite-based model makes broadband access in regions where traditional Internet Service Providers (ISPs) cannot profitably operate theoretically possible, provided it is adopted correctly [7, 16].

Despite Starlink's technical promise, its economic viability for developing countries remains uncertain. The service requires monthly subscription fees that often exceed local telecommunications prices, expensive user terminals with one-time fees, and stable electricity, which is often not available in rural areas of developing countries. None of these conditions is widely available in low-income regions. Early Starlink deployments highlight this gap. Nigeria, the first African country to approve Starlink, had a one-time hardware kit cost of approximately $405 and a monthly subscription of $26 [20, 7]. For households living below or near the national average GNI per capita, roughly $2,200 annually, this represents an unreasonable share of disposable income. The Broadband Commission's affordability research recommends that entry-level broadband should not cost more than 2% of monthly GNI in low- and middle-income countries [3]. This threshold is exceeded by Starlink in nearly all surveyed developing countries.

Comparing Starlink to existing connectivity options further complicates this economic position. In Southeast Asia, Africa, and parts of Latin America, mobile broadband is widely available, with an average monthly cost of $5 to $15 for basic data plan bundles [18, 26]. Although mobile broadband's latency and speeds are typically inferior to Starlink's, terrestrial alternatives remain cheaper in their regions. Fixed wireless providers that operate using last-mile radio links and microwave backhauls offer monthly rates in the $10-$20 range for these regions. These prices are far below the cost threshold for average consumers to adopt satellite broadband services. In contrast,

fiber connectivity remains extremely limited to urban areas but is subsidized by private operators, creating a competitive price disadvantage for Starlink. The Global System for Mobile Communications Association (GSMA) conducted a comparative analysis indicating that the most influential factor in the adoption of broadband services is price sensitivity [26]. As a result, technical performance alone is not a strong enough driver of mass adoption when significant affordability gaps exist [1, 3].

One of the central determinants of Starlink's long-term viability, especially in developing countries, involves the cost structure. Starlink incurs substantial capital expenses annually for launch operations, satellite manufacturing, regulatory lobbying, licensing, and ground station installation. Public cost models predict SpaceX's total investment in Starlink exceeded $11 Billion in launch expenditures and manufacturing alone [17, 19]. This does not factor in orbital repositioning, satellite maintenance, fleet replacement, which is estimated every five to seven years, or other underlying factors that will further increase long-run operational costs. Starlink's economic sustainability largely depends on stable revenue generation, user adoption, and overall utilization worldwide. Profitability in low-income markets is uncertain, with most reports providing only rough estimates, particularly in areas where disposable household income limits the feasibility of Starlink's pricing window. The Organization for Economic Co-operation and Development (OECD) found that satellite operators incur higher per-user operating costs than terrestrial ISPs [18]. It is largely due to constrained bandwidth capacity in certain regions and the need to maintain expensive ground infrastructure [18].

Case studies can be evaluated to offer early insight into Starlink's feasibility. In Kenya, Starlink's deployment included institutional partnerships to support school connectivity, primarily for Information and Communication Technologies for Development (ICT4D) programs, using a shared-access model to reduce the total cost per user [7], [25]. In Haiti and South Sudan, where numerous regions face political instability and declining infrastructure, Starlink adoption is primarily limited to Non-Governmental Organizations (NGOs), businesses that require Starlink for critical operations, and households living above the economic norm. Nigeria is a perfect case study, with higher-income urban households adopting Starlink, while rural areas lag behind due to high costs and unreliable electricity. These patterns show that Starlink adoption correlates with current provider availability, income, and electricity reliability rather than technology capabilities alone [7,16].

Looking ahead, Starlink announced regulatory clearance in countries such as Venezuela, Nepal, and Burkina Faso, where Starlink is currently unavailable. These regions are characterized by challenging terrain, low GNI per capita, and political instability [5, 12, 24]. Predictive models using GNI per capita, population, and current satellite capacity assumptions estimate that adoption in these case studies would require additional constellation capacity. It is especially true in Venezuela and Nepal due to high urban density and mountainous terrain. Whether such constellation capacity expansions are economically viable depends on revenue projections, subscription levels, and long-term profit margins.

The economic viability of Starlink for developing countries rests on the affordability of the population, the long-run cost of infrastructure maintenance, current competition with fiber networks and ISPs, and the feasibility of maintaining a steady revenue stream in low-income regions. These challenges are fundamental for evaluating Starlink's projected role in global broadband expansion.

The rest of the paper is organized as follows: Section II discusses the adopted research methods and research objectives, Section III identifies key metrics such as affordability target and affordability ratio, and case studies of Burkina Faso, Nepal, and Venezuela, Section IV analyzes the projected costs, revenues, and financial performance of Starlink, Section V discusses long-term cost of Starlink deployment, Section VI discusses how Starlink differ from traditional internet infrastructure, Section VII summarizes findings and we conclude the article by conclusion in Section VII.

From hereinafter, all cost or price values in this article are expressed in U.S. dollars.

List of Acronyms

LEO	Low-Earth Orbit
GEO	Geostationary
ISP	Internet Service Providers
GNI	Gross National Income
GSMA	Global System for Mobile Communications Association
OECD	Organization for Economic Co-operation and Development
ICT4D	Information and Communication Technologies for Development
NGO	Non-Governmental Organizations
ICT	Information and Communications Technology
UNCTAD	United Nations Conference on Trade and Development
OPEX	Operational expenditure
ITU	International Telecommunication Union

II. Methodology

The research component and analysis use a desk-based, mixed-method approach that draws on current literature reviews and reputable sources, economic comparisons, and quantitative modeling to assess the viability of Starlink in developing countries and its economic impact. The methodology provides evidence-based insights into deployment feasibility, affordability, and comparative performance against current infrastructure. This avoids conducting household surveys or fieldwork to make general assumptions without sacrificing accuracy. Using publicly available technical documentation, demographic and global income data, satellite constellation statistics, and affordability standards, a baseline can be created for affordability thresholds and adoption projections. Peer-reviewed studies on Information and Communication Technology (ICT) and broadband adoption were used as key sources.

The overall analysis is guided by the following research questions:

i. What is the affordability and accessibility of Starlink for typical consumers in developing countries?
ii. What are the projected costs and revenues of deploying Starlink infrastructure in developing countries, and is it profitable for SpaceX?
iii. What is the potential financial performance of Starlink operations in developing countries?
iv. What is the cost of Starlink over the long run?
v. How does Starlink compare to current infrastructure, including ISPs and fiber options, considering future and current trends?

The overall objective of the research aligns with these questions, including the affordability conditions and summarized connectivity within developing countries. Technical, pricing, and operational characteristics are evaluated in light of individual household affordability. Long-term costs and potential revenue streams associated with deploying Starlink in developing countries are compared with those of existing wireless and terrestrial broadband infrastructure. This assesses Starlink's performance, cost, and scalability.

Understanding digital connectivity conditions and affordability thresholds for developing countries included reputable sources. Notable sources include the ITU's report on affordability metrics, mobile and fixed broadband baskets, and global broadband deployment [10, 12, 13]. Cost structures and data for regional broadband adoption were gathered from ITU, the United Nations Conference on Trade and Development (UNCTAD), the World Bank, and the Alliance for Affordable Internet [1, 2, 22]. Academic studies were also analyzed for examining rural ICT deployment challenges and satellite solutions compared to broadband [6, 15, 25]. Mobile broadband and fixed broadband baskets defined by the ITU were compared with existing service costs against the target affordability benchmark. This included the Broadband Commission for Sustainable Development Target 2, which recommends that broadband service costs should not exceed 2% of monthly GNI per capita [3]. This provides the foundation for considering whether Starlink could be a feasible option for rural and low-income area households and institutions.

Alongside the affordability review, operational and technical information related to Starlink was collected from multiple sources. This included SpaceX's technical specifications, launch tracking data, and constellation status reports [17, 19]. Together, these sources provide estimates for orbital altitudes, current satellite numbers, user terminal specifications, per-country pricing, and global coverage areas. Country-specific pricing data were estimated using publicly available information to account for taxes, currency fluctuations, and service plan tiers [5]. The national GNI per capita and population data help to highlight the affordability of Starlink relative to household income. This data was obtained from the World Bank's World Development Indicators database. The database enabled the calculation of Starlink's estimated user cost relative to the average household income [2, 24].

The quantitative analysis comprises two levels: a satellite capacity allocation model and an affordability evaluation. Affordability was assessed by comparing the one-time equipment cost and the monthly subscription fee to the average monthly household income.

A. *Affordability Ratio*

The Affordability ratio assesses whether the Starlink service can realistically be maintained by the typical household in a developing country. The metric represents the monthly income required to cover the cost of the Starlink subscription. It is calculated as:

$$Affordability\ Ratio\ (\%) = \frac{(Monthly\ Subscription\ Cost) * 100}{Monthly\ GNI\ Per\ Capita}$$

Monthly Subscription Cost is the recurring Starlink fee, which varies based on the service tier – high performance, business, or residential. Monthly GNI per Capita is calculated by dividing the country's GNI per capita by 12, representing a year. This provides a monthly income reference point [2, 24].

Utilizing this ratio standardizes how affordability across countries compares to a varying income level threshold. The metric is multiplied by 100 to convert the result from a decimal fraction to a percentage for easier interpretation. Multiple scenarios were also factored into the subscription tier variation, shared-access models in which cost was redistributed across multiple institutions or users, and subsidy considerations to offset equipment and monthly costs.

B. *Estimating Satellite Capacity*

Satellite capacity requirements were estimated using a population-based allocation methodology. This accounts for the expected adoption rates among potential users and the concurrent network usage.

$$SatellitesRequired = \frac{National\ Population * Assumed\ Adoption\ Rate}{Est.Users\ Per\ Satellite}$$

The National Population represents the total estimated population of the country. Assumed Adoption Rate factors in a fraction of the population that is expected to subscribe to the Starlink service. For this capacity estimation, 10% of the population was assumed to adopt Starlink. The Estimated Users per Satellite represents the typical capacity of a single satellite to support simultaneous users. This capacity derives from Starlink's technical specifications and operational analysis [17, 19].

The above equation establishes a baseline estimate for the total number of satellites required to serve an anticipated number of users. This formula does not consider simultaneous peak demand. To adjust for peak concurrent usage, a concurrency factor is applied.

$$AdjustedSatellites = SatellitesRequired\ x\ ConcurrencyFactor$$

The Concurrency Factor represents the proportion of Starlink subscribers likely to use the network simultaneously at

peak times. For this requirement estimation, 30% of subscribers are considered to use the network simultaneously at peak times. This factor is necessary to ensure the network maintains performance while having sufficient user capacity during periods of high demand.

C. Subscriber Statistics

Subscriber data for Starlink included the estimated number of users by country and region. Data was obtained from a publicly available dataset provided by Est Research & Advisory via the Starlink Country Data Tracker [7]. The data tracker aggregates monthly and quarterly estimated subscriber counts globally. This is further broken down by region or country. This information assisted in forming a baseline for demand analysis and for adjusting the adoption rate in satellite capacity modeling.

D. Assessing Viability of Starlink Deployment

The economic viability of Starlink deployment for SpaceX was also assessed through a combination of launch expenditures, satellite production costs, geographic coverage requirements, and capacity-based demand modeling. This helps identify the environmental and operational factors that influence Starlink's long-term financial sustainability in developing countries that predominantly have low-income markets. Cost input was obtained from publicly reported estimates, launch statistics, and technical documentation from SpaceX and aerospace sources [17, 19].

Satellite and launch costs were estimated due to the lack of SpaceX's official publications, which limits the analysis. Coverage modeling estimated the area-based satellite requirements for developing countries. Coverage footprints were approximated using the total country area, yielding a theoretical Starlink deployment requirement [17, 19]. However, this estimation is constrained by country geography, actual capacity demand, orbital plane availability, and beam reuse. Additionally, elevation, orbital geometry, and terrain provided limitations to determining the true viability of Starlink deployment. Terrain and elevation significantly impact the total number of satellites required to deliver uninterrupted service to a population. Beam reuse is also capacity-limited and must be estimated based on the maximum number of users per beam. The exact number of these limitations is known but depends on channel reuse patterns, total available bandwidth, frequency interference management, and congestion mitigation systems [17, 19]. This is a major limitation since satellite demand is proportional to economic activity and population, not just the physical coverage area.

E. Comparative Infrastructure Analysis

A comparative infrastructure analysis was also conducted alongside Starlink-specific modeling to benchmark the service against other competitive options. This analysis included fixed wireless access, mobile broadband, and fiber-optic networks. Reports on cost-per-subscriber metrics, deployment costs, and the long-term sustainability of terrestrial broadband in rural environments were reviewed [11, 12, 26]. Comparative analysis considered projected trends in LTE/5G adoption, mobile network expansion, and regulatory frameworks. These factors may influence Starlink's adoption trajectory relative to existing terrestrial networks.

F. Research Gaps and Limitations

Finally, key research limitations and gaps are considered. Publicly available information on Starlink's long-term maintenance and operational costs is limited. Most information regarding quantifications and specifications is estimated due to the lack of official public information. Additionally, peer-reviewed studies quantifying adoption patterns in urban vs. rural households are scarce. The income and affordability assessments rely solely on national averages, which may underestimate challenges faced by low-income households depending on their geographic location. Despite these limitations, combining affordability modeling, literature reviews, satellite allocation projections, and comparative infrastructure benchmarking provides a replicable framework for evaluating Starlink's economic viability in developing countries.

III. Affordability and Accessibility for the Typical Consumer

Assessing the accessibility and affordability of Starlink for consumers in developing countries requires integrating broadband affordability benchmarks, macroeconomic indicators, and country-specific socioeconomic constraints to shape average household purchasing power. Global reporting reviews from the Broadband Commission, ITU, and the World Bank show that developing countries continue to face digital inequalities even when connectivity options are available [13, 10, 22]. Several factors suppress broadband adoption, including income level, energy instability, infrastructure limitations, and total device cost. The Broadband Commission's affordability benchmark is widely accepted, stating that entry-level broadband services should cost ≤ 2% of monthly GNI per capita. This serves as the primary threshold for determining affordability in low- and middle-income countries [3, 12]. Connectivity and hardware pricing derive from publicly available global pricing aggregators and Starlink technical specifications [19, 20]. The central question is whether the average household in a developing country can afford the monthly and startup costs for Starlink's broadband subscription at the estimated levels. Three case studies will be analyzed to determine the affordability: Burkina Faso, Nepal, and Venezuela.

Following Broadband Commission and ITU standards, entry-level broadband services are considered affordable when:

$$AffordabilityTarget(Monthly) = 0.02\ x\ Monthly\ GNI\ per\ Capita$$

Factor 0.02 represents the ITU/Broadband Commission's 2% affordability ceiling [3, 12]. Monthly GNI per Capita is given by $\frac{Annual\ GNI\ Per\ Capita}{12}$

This outcome represents the absolute maximum a typical consumer can pay per month for the Starlink service while staying within an affordability range. This affordability target

serves as a benchmark ceiling: any broadband plan that exceeds the calculated limit is considered unaffordable for the average consumer in the given economy. This model is used because of the differentials in purchasing power across low- and middle-income markets. A percent-based measure makes a comparison across the entire country meaningful. Exceeding the 2% threshold indicates the Starlink service would most likely displace necessary consumption involving transportation, food, education, and healthcare. Quantifying the Starlink pricing uses the Affordability Ratio (%) to express the average person's monthly income required to maintain the Starlink subscription.

Due to limitations in price-point estimation for Starlink plans in countries without service, projected price points were based on existing locations. Two tiers were utilized for the price point estimation:

- Residential Plan (Baseline), P_{low} = *$40 per month*
- High-End Equivalent, P_{high} = *$80 per month*

These ranges are consistent with the observed international Starlink pricing structure for residential plans. Plans vary based on regulatory costs, the service region, and the service tier [3], [19].

A. Case Study: Burkina Faso

Burkina Faso is a prime example of an economically constrained country in Sub-Saharan Africa. It ranks among the world's lowest-income countries, facing challenges in broadband connectivity, reliable electricity services, and digital inclusion in rural areas. National electrification stands at approximately 22-25%, with rural electrification below 10% [1, 24]. This severely limits the operational feasibility of internet services for the average household. Mobile broadband services remain underdeveloped due to high service costs, limited 4G coverage, and a dependence on 3G networks [1], [11]. Burkina Faso's GNI per capita is estimated to be $880 per year [24]. Converting it to a monthly income value provides a baseline for the Broadband Commission and ITU affordability standards.

$$Monthly\ GNI\ Per\ Capita = \frac{880}{12} = 73.33$$

This averages to $73.33 per month, placing Burkina Faso among the lowest-income countries in the world [24]. The ITU affordability threshold then sets the monthly cost of an entry-level broadband subscription [3, 12]. Affordability Target per month is given by, AT_m,

$$AT_m = 0.02\ x\ 73.33 = 1.47$$

The broadband pricing of $1.47 per month exceeds the affordability standard set by the Broadband Commission and ITU. This reflects the country's economic constraints, in which a household will prioritize education, food, and other expenses over spending on technology. Affordability ratios (ARs) based on Starlink's price-point estimates can serve as benchmarks. The affordability ratio indicates the percentage of monthly income needed to purchase the Starlink service [20, 12].

$$AR_{40} = \left(\frac{40}{73.33}\right) x\ 100 = 56.4\%$$

$$AR_{80} = \left(\frac{80}{73.33}\right) x\ 100 = 109.2\%$$

The Affordability Ratio shows that even the lowest possible Starlink price point would require more than half of the average household income in Burkina Faso. The higher-tier pricing exceeds the average citizen's entire income [24, 1]. Furthermore, the economic profile of Burkina Faso is heavily skewed, with the bottom 40% of households earning below the national average [24, 1]. This results in an even higher affordability ratio than predicted.

B. Case Study: Nepal

Nepal has a higher income level than Burkina Faso despite some economic constraints. The government has a clear focus on ICT development, particularly when bridging the rural-urban connectivity gap [14], [21]. Over 80% of the population resides in extreme geographical regions, severely complicating terrestrial broadband deployment [2], [21]. Electricity also remains unreliable, with many urban households relying on separate power sources or microgrids [2, 14]. Nepal's GNI per capita is estimated to be $1,382 per year [24].

Again, following the ITU and Broadband Commission, the affordability threshold can be set [3, 12].

$$Monthly\ GNI\ Per\ Capita = \frac{1382}{12} = 115.17$$

The Affordability Target sets the broadband service at $2.30 per month based on an average monthly income of $115.17. This monthly cost is the maximum affordable threshold in Nepal.

$$AT_m = 0.02\ x\ 115.17 = 2.30$$

Using Starlink's estimated residential pricing, the affordability ratios can be calculated.

$$AR_{40} = \left(\frac{40}{115.17}\right) x\ 100 = 34.7\%$$

$$AR_{80} = \left(\frac{80}{115.17}\right) x\ 100 = 69.4\%$$

The Affordability Ratio indicates that the lowest Starlink subscription plan would require over one-third of the average monthly income for the average household in Nepal. Higher subscription plans would consume nearly 70% of the monthly income, exceeding the 2% affordability benchmark.

C. Case Study: Venezuela

Venezuela is a unique case study, featuring a mix of structural, political, and economic challenges that influence broadband service affordability. Venezuela has suffered from currency instability, hyperinflation, and declining household wages over the past decade, dramatically reducing overall purchasing power [23, 5]. The electricity supply is considered more reliable compared to previous case studies, but can be intermittent in smaller towns and rural areas [8]. ICT infrastructure disparities exist between urban and rural

environments, with most fixed-line broadband and fiber connections concentrated in metropolitan areas [23], [26]. Venezuela's GNI per capita is estimated to be $2,640 per year [24].

Like previous case studies, the ITU and Broadband

Commission standards are used for the affordability threshold.

$$Monthly\ GNI\ Per\ Capita = \frac{2640}{12} = 220.00$$

The Affordability Target sets the broadband service at $4.40 per month based on the average monthly income of $220.0. This represents the maximum service cost threshold in Venezuela. Using Starlink's estimated residential pricing, the affordability target ratios can be calculated [20]. Again,

$$AT_m = 0.02\ x\ 220 = 4.40$$

$$AR_{40} = \left(\frac{40}{220}\right) x\ 100 = 18.2\%$$

$$AR_{80} = \left(\frac{80}{220}\right) x\ 100 = 36.4\%$$

Once again, even the lowest Starlink residential plan would consume over 18% of the average income in Venezuela. This exceeds the 2% affordability benchmark by roughly 9 times. Higher-tier subscription pricing would exceed 36% of the monthly income, making Starlink unattainable for most households [23], [26].

IV. Projected Costs, Revenues, and Financial Performance of Starlink

Deploying Starlink infrastructure in developing countries requires substantial operational factors and capital costs. These conditions weigh against potential revenue streams and must be considered when assessing profitability. Launch and satellite manufacturing costs contribute to most of the upfront investment. Based on publicly available data estimates, Starlink satellites vary in cost by generation. Launch costs assume an average cost of $30 million per launch using Falcon 9 [17]. For this estimation, 60 satellites can be carried on Falcon 9 for v1.0 and v1.5, while 72 satellites can be carried on the v2 Mini [17]. Launch costs and total manufacturing costs for the current constellation are estimated in **Table 1**.

Variant	Manufacturing Cost	Launch Cost	Total Cost	
V1.0	$624,375,000	$832,500,000	$1,456,875,000	
V1.5	$1,120,125,000	$1,493,500,000	$2,613,625,000	
V2 Mini	$4,599,200,000	$2,395,233,000	$6,994,433,000	
Total		$6,343,700,000	$4,721,233,000	$11,064,933,000

Table 1: *Combined manufacturing and launch costs for total deployment expenditures.*

A coverage perspective based on the current 8,800-satellite estimate can be applied to the previous case studies. The current constellation provides near-global coverage with each satellite approximating a LEO footprint of 1.8 million km^2 [17], [19]. For the case studies, combined area totals can be used to calculate the additional satellites needed for coverage: Burkina Faso (272,000 km^2), Nepal (147,000 km^2), and Venezuela (916,000 km^2).

Taking the total area and dividing by the area of coverage estimation per satellite suggests that only one additional satellite would suffice to provide basic coverage [17]. Area-based count of satellites is given by:

$$SatellitesNeeded = \frac{TotalArea}{Area\ per\ Satellite} = \frac{1335000}{1800000} = 0.74 \sim 1$$

However, satellite allocation needs to account for population density and potential user demand rather than relying solely on area coverage. Starlink Satellites can support a finite number of users simultaneously per beam. V2 Minis can handle 1,000-2,000 concurrent users per satellite [16], [17]. Since the older satellite variants are obsolete, the predicted satellites will rely solely on the v2 Minis. 10% of the population is assumed to be the adoption rate for early deployment within the case studies. Estimating the total population, the potential subscribers can be predicted:

- Burkina Faso: 22M × 10% = 2.2M users
- Nepal: 30M × 10% = 3M users
- Venezuela: 28M × 10% = 2.8M users

Using a midpoint of 1,500 users per satellite, the estimated number of satellites needed to fully meet the demands of new subscribers can be calculated.

$$Satellites\ Needed = NO.ofUsers/(No.\ of\ Users\ per\ Satellite)$$

- Burkina Faso ~ 1,467
- Nepal ~ 2,000
- Venezuela ~ 1,867

Adjusting for realistic concurrent usage considers 30% of the total Starlink subscribers in these regions will be online simultaneously.

- Burkina Faso ~ 440
- Nepal ~ 600
- Venezuela ~ 560

As a result, approximately 1,600 additional satellites would be required to serve these populations based on user population-based demand assumptions. While limited, this calculation accounts for geography and urban concentration, essential for mountainous regions and urban centers, which necessitate additional capacity for consistent bandwidth.

With an estimated satellite manufacturing cost of $800,000 and a launch cost of approximately $416,667 per v2 Mini satellite, the total launch and manufacturing costs for 1,600 additional satellites can be calculated as $1600\ x\ 800000 + 1600$ x $416667 = \sim 1.95$ Billion. This estimate relies solely on launch and manufacturing costs, excluding maintenance, operating expenses, insurance, and ground infrastructure. This

provides a rough baseline for the capital SpaceX needs to invest in expanding Starlink's coverage in these developing countries.

A. *Real-world Subscriber Base Estimations*

Before calculating Starlink's profitability for deploying new satellites to developing countries, current subscriber trends can be analyzed to inform adoption rates. **Fig. 2** highlights Starlink subscriber estimates by region, with an estimated **2024** revenue of over 3.2 billion [7].

Region	Jun-24	Sep-24	Dec-24	Mar-25	Jun-25	Sep-25
Africa	116	161	222	256	376	515
Asia	122	284	488	795	1,075	1,282
Central America	84	97	118	145	184	207
Europe	296	379	507	593	633	710
North America	1,897	2,118	2,410	2,641	2,693	3,068
Oceania	262	295	315	338	390	423
South America	359	443	535	613	754	879
Total (k)	**3,136**	**3,778**	**4,595**	**5,383**	**6,105**	**7,084**

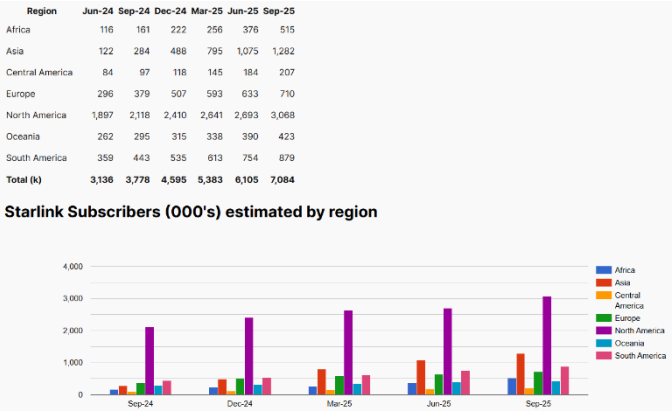


Fig.2 *Starlink subscriber estimations based on region [7]*

B. *Theoretical Annual Revenue*

From the affordability assessment, the maximum feasible adoption rate for each country was modeled based on Starlink pricing tiers and income levels. For the annual revenue estimate, the residential pricing plan at $40 per month is used as the baseline. Additionally, a concurrent adoption factor of 30% is applied, assuming the subscribers actively pay for the service.

The potential number of actively paying subscribers in each country is estimated as a function of population (P), adoption rate (R), and concurrent adoption factor (CAF),

$$Subscribers = P \; x \; R \; x \; CAF$$

Assuming an initial early-adopter rate of 10% of the population, with a 30% concurrency adjustment, the estimated subscriber counts are 660,000 for Burkina Faso, 900,000 for Nepal, and 840,000 for Venezuela. This represents a conservative estimation of active, paying households that could potentially subscribe without violating affordability thresholds. Monthly revenue is calculated by multiplying the estimated subscribers by the monthly subscription price, *MSP*.

$$Revenue = Subscribers \; x \; MSP$$

Using the $ 40-per-month baseline, the resulting monthly revenues are approximately $26.4 million for Burkina Faso, $36 million for Nepal, and $33.6 million for Venezuela. This yields an estimated total monthly revenue of $96 million across all three countries. Annual revenue calculations result in approximately $1.152 billion for SpaceX.

V. Long-Run Cost of Starlink Deployment

A comprehensive evaluation of initial capital expenditures, satellite lifecycle replacements, and recurring operational costs is required to understand the long-run cost of Starlink deployment. Using the deployment of 1,600 additional v2 Mini satellites to serve Burkina Faso, Nepal, and Venezuela as an example, the upfront manufacturing and launch costs alone are substantial. Despite these figures providing a baseline for capital expenditure, a full analysis that utilizes long-run cost must incorporate maintenance, ongoing operations, and future satellite replacement cycles [7], [16], [7].

Operational expenditure (OPEX) is a critical component of a long-run financial model. This may include network management, satellite control, customer service, ground station operations, software updates, and more [7], [16]. Industry estimates place satellite network operating costs at 5-15% of total capital expenditure annually [17], [19]. Applying a conservative 10% OPEX factor to the $1.95 billion deployment cost for the case study yields an estimated annual operational cost of $195 million.

Satellite replacement and maintenance are equally important for long-term cost projections. The v2 Mini satellites have an operational lifespan of approximately five to seven years, resulting in a large portion of the constellation being replaced in the future [19], [16]. Assuming a six-year lifespan, replacing the 1,600 satellites deployed at similar launch and manufacturing costs would require an additional $1.95 billion, doubling the initial capital investment over this timeframe. Factoring in unanticipated events such as deorbiting, collision avoidance, and orbital debris management adds complexity and increases potential cost when calculating the cost of long-term operations [16].

Long-term costs must also consider redundancy, capacity scaling, and service reliability. Factors such as increased broadband adoption, population growth, and higher concurrency may require additional satellites or beam reallocation to meet an influx in demand. Environmental conditions may also pose significant challenges, especially in areas with heavy rainfall or mountainous terrain. These environments can reduce coverage and throughput, requiring additional satellite coverage in these areas to maintain effective service [2, 16].

It is important to note that these estimations rely on several assumptions. The affordability model holds true, with consistent monthly subscription pricing, the absence of technical, environmental, and regulatory constraints, and fixed manufacturing and launch costs. Infrastructure and operational costs, along with satellite replacement cycles, will continue to influence the long-run sustainability for Starlink.

VI. Starlink vs. Traditional Infrastructure

Starlink has several core advantages that make it more practical for deployment in developing countries than traditional infrastructure. Starlink can deliver broadband to areas that terrestrial fixed wireless, cellular networks, and fiber-optic cables simply can't reach. Laying fixed-wired broadband or fiber across remote, mountainous, and rural areas is incredibly costly, not only for development but also for users. This results in a poor return on investment and long deployment times [15]. Starlink's LEO satellite constellation can reach

underserved and remote regions without extensive physical infrastructure and ground cables [15, 16]. This makes Starlink a promising tool and acceptable option for dispersed-population territories.

However, from a performance standpoint, terrestrial fiber infrastructure is held at a higher standard. Fiber-optic broadband can deliver symmetrical gigabit speeds with low latency, resulting in extremely high reliability [18, 19, 23]. In contrast, Starlink currently offers download speeds in the 50-250 Mbps range, with some premium tiers advertising higher rates. Upload speeds are also considerably lower than fiber, with latency ranging from 20 to 50 ms, depending on configuration, weather, and user load [11, 13, 19].

Long-term trends suggest a hybrid or complementary model that fuses terrestrial infrastructure with Starlink [16]. Terrestrial fiber networks continue to expand, especially in semi-urban areas where deployment costs are becoming more affordable. This helps terrestrial fiber networks retain their reliability, scalability, and superior performance over Starlink as the chosen broadband service. Meanwhile, Starlink may retain an advantage in hard-to-reach and remote regions where fiber networks remain physically or economically unviable [16].

Still, LEO systems face several recurring drawbacks compared to traditional infrastructure. Variability and instability in speeds and latency due to weather and network load remain a core challenge. The high upfront cost of hardware and service pushes most potential customers in developing countries out of the market. Long-term sustainability issues also raise questions when factoring in ground-station infrastructure requirements, orbital debris, and the need for consistent maintenance [16].

VII. Results and Findings

Several factors that could suppress broadband adoption were considered when assessing the affordability of Starlink for consumers in developing countries, including energy instability, low incomes, infrastructure limitations, and total device cost. Using the Broadband Commission and UTI's affordability benchmark for entry-level broadband service, several findings were made regarding the affordability of broadband services in Burkina Faso, Nepal, and Venezuela [3], [12].

Based on the affordability threshold, the average household in Burkina Faso, Nepal, and Venezuela would be unable to afford the upfront cost of Starlink and a monthly service subscription. Even at the lowest Starlink pricing, the total cost would exceed the total income of most citizens. This is especially true given Burkina Faso's skewed income distribution, which poses further affordability challenges for the bottom 40% of households [24, 1]. Nepal provides similar numbers to the baseline Starlink plan, consuming over one-third of the average household's monthly income. While Venezuela had lower monthly income consumption under the baseline plan at 18%, the region also faces unique political, economic, and infrastructure challenges that may make the adoption of Starlink unreliable [23], [26].

When a conservative 10% early adoption and 30% concurrency rate were assumed among households that could afford Starlink, the estimated subscriber counts were: 660,000 for Burkina Faso, 900,000 for Nepal, and 840,000 for Venezuela. At $40 per month, monthly revenues reach $26.4M, $326M, and $33.6M, respectively, totaling $96M a month or $1.15B a year. This suggests that the potential cost recovery from the estimated $1.95B investment in the additional satellite deployment could be achieved within two years, excluding all operational costs [7, 17].

Long-term costs still highlight various considerations such as satellite replacement, scaling to meet population growth, and operational expenses. These factors may double the initial capital investment over a six-year cycle. Additionally, network demand and environmental factors may necessitate additional satellites to maintain service quality [16, 19].

Discussion and Conclusion

Starlink has demonstrated the potential to significantly expand broadband access when analyzed for developing countries. This is especially true in regions where terrestrial infrastructure is limited or unaffordable. Affordability remains a key constraint, even at the lowest Starlink subscription tier. Financial modeling suggests that, under conservative adoption assumptions, annual revenue from case studies could approach $1.15 billion. This could allow SpaceX to recover the initial cost of launching additional satellites within two years. However, long-term sustainability depends on satellite replacement cycles, operational costs, and scaling to meet an increasing user demand. This may substantially raise the total operational cost over time.

Starlink offers advantages over traditional fiber and ISP infrastructure, especially in geographically challenging or remote areas. However, fiber does retain superiority in reliability, speed, and latency in semi-urban and urban regions. Altogether, Starlink presents a promising but logistically and financially complex solution for bridging the digital divide in developing countries. Its future success will depend on strategic deployment, a carefully calibrated pricing model, and ongoing investment in satellite operations and infrastructure to achieve profitability while maintaining reliable service.

Contribution

Planic assessed the affordability and accessibility for typical consumers, including modeling income-based adoption rates, estimating potential subscribers, and calculating projected revenue. He conducted estimates of deployment costs, evaluated financial performance, and projected monthly and annual revenue and profitability under different scenarios. Planic contributed to drafting the abstract, introduction, methodology, and conclusion. Frank analyzed the long-run costs of Starlink deployment, including satellite lifecycle management, replacement cycles, and operational costs. Frank also conducted a comparative evaluation of Starlink versus traditional infrastructure, highlighting differences in coverage, performance, and scalability. Adhikari provided research directions, periodic feedback, established methodologies, and proofread and reorganized the article for review and publication.

REFERENCES

[1] Alliance for Affordable Internet (A4AI), "Affordable Internet Report 2023," Web Foundation, 2023.

[2] Asian Development Bank (ADB), "Nepal: Rural Electrification and ICT Access Assessment," ADB, Manila, Philippines, 2022. [Online]. Available: https://www.adb.org/publications/nepal-rural-electrification-ict

[3] Broadband Commission for Sustainable Development, "Affordability of broadband services in developing countries — Advocacy Target 2," 2025. [Online]. Available: https://www.broadbandcommission.org/advocacy-targets/2-affordability/

[4] Broadband Commission for Sustainable Development, "2025 Broadband Advocacy Targets," Broadband Commission, 2020.

[5] DataReportal, "Digital 2025: Venezuela — Local Country Report," DataReportal, 2025. [Online]. Available: https://datareportal.com/reports/digital-2025-venezuela.

[6] E. Oughton and O. Osoro, "Policy options for digital infrastructure strategies: A simulation model for broadband universal service in Africa," *Telecommunications Policy*, vol. 45, no. 8, pp. 102–117, 2021.

[7] Idem Est Research & Advisory, "Starlink Subscribers by Country Data Tracker – 2025 (14th Edition)," Idem Est, 2025. [Online]. Available: https://idemest.com/reports/starlink-country-data-tracker/?srsltid=AfmBOoqmGneQd1OZ6Qbp6rYdY9EySf7iLGtTZdoHdzixRpm9DEovvruj

[8] Internet Society, "Country Report – Venezuela (Bolivarian Republic of)," Internet Society Pulse, Dec. 2024. [Online]. Available: https://pulse.internetsociety.org/en/reports/ve/.

[9] International Telecommunication Union, *ICT for Development Index and Methodology,* Geneva, 2021.

[10] International Telecommunication Union, *Measuring Digital Development: Facts and Figures 2023*, Geneva, Switzerland: ITU, 2023. [Online]. Available: https://www.itu.int/itu-d/reports/statistics/facts-figures-2023/

[11] International Telecommunication Union, *Measuring Digital Development: ICT Price Trends 2022*, Geneva, 2022.

[12] International Telecommunication Union, *The Affordability of ICT Services 2023*, Geneva, Switzerland: ITU, 2023. [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Documents/publications/prices2023/ICTPriceBrief2023.pdf

[13] ITU & UNESCO Broadband Commission, *The State of Broadband 2023: Digital Prosperity for All*, Geneva, 2023.

[14] Ministry of Communications and Information Technology (MoCIT), "Nepal Digital Economy Framework and ICT Infrastructure Report," Government of Nepal, Kathmandu, 2023. [Online]. Available: https://www.mocit.gov.np/digital-economy-report-2023

[15] O. Osoro, E. Oughton, and J. Karanja, "Geospatial sustainability assessment of universal Fiber-To-The-Neighborhood (FTTnb) broadband infrastructure strategies for Sub-Saharan Africa," *ICT4D Journal*, 2024.

[16] O. Osoro et al., "Sustainability assessment of Low Earth Orbit (LEO) satellite broadband megaconstellations," *Journal of Space Systems and Policy*, vol. 3, no. 2, pp. 45–63, 2023.

[17] Planet4589, "Starlink statistics," Planetar/space launch tracker. [Online]. Available: https://planet4589.org/space/con/star/stats.html

[18] R. Croshier, *Space and Development: Preparing for Affordable Space-Based Telecommunications*, Center for Global Development, Washington, DC, 2022. [Online]. Available: https://www.cgdev.org/publication/space-and-development-preparing-affordable-space-based-telecommunications

[19] SpaceX, "Starlink specifications," Starlink, 2024. [Online]. Available: https://www.starlink.com/legal/documents/DOC-1138-34130-60

[20] Starlink per-country pricing (compiled): Starlink Prices — Personal / Residential (USD). [Online]. Available: https://www.starlink-prices.com/personal/residential/usd/low

[21] United Nations Development Programme, "Nepal Human Development Report 2023: Bridging the Digital Divide," UNDP, Kathmandu, Nepal, 2023. [Online]. Available: https://www.np.undp.org/content/nepal/en/home/library/human-development.html

[22] World Bank, "Digital development," World Bank, Washington, DC, USA. [Online]. Available: https://www.worldbank.org/en/topic/digital

[23] World Bank, "Venezuela Country Overview," World Bank, Washington, DC, USA, 2024. [Online]. Available: https://www.worldbank.org/en/country/venezuela/overview

[24] World Bank, "World Development Indicators: GNI per Capita (Atlas Method)," World Bank Open Data, 2024.

[25] Y. Shaengchart and T. Kraiwanit, "Public perception of the Starlink satellite project in a developing country," *Corporate and Business Strategy Review*, vol. 4, no. 3, pp. 66–73, 2023. [Online]. Available: https://virtusinterpress.org/IMG/pdf/cbsrv4i3art7.pdf

[26] GSMA Intelligence, *The State of Mobile Internet Connectivity 2023 — Latin America & Caribbean*, GSMA, 2023. [Online]. Available: https://www.gsma.com/r/wp-content/uploads/2023/10/State-of-Mobile-Internet-Connectivity-2023-Latin-America-and-Caribbean.pdf

EFFICIENT AND ACCURATE MACHINE LEARNING ALGORITHMS FOR PREDICTING CROHN'S DISEASE

D. Tom Gehman, Kaitlyn Zeigler, Jeonghwa Lee
Department of Computer Science Shippensburg University
dg5957@ship.edu kz2818@ship.edu jlee@ship.edu

ABSTRACT
Crohn's disease is a chronic inflammatory bowel disease caused by a complex combination of environmental and genetic factors. This disease is difficult to treat, but high-risk patients can adjust their lifestyles to avoid it before it develops. Using genetic information to predict susceptibility can help people make decisions to mitigate their risk. This paper uses publicly available genetic data to compare the abilities of four machine learning algorithms to predict Crohn's disease. We compare the results of K Nearest Neighbors, Support Vector Machines, Random Forests, and XGBoost. Additionally, we experiment with feature reduction and categorical encoding to observe their impacts on model efficiency and accuracy.

KEY WORDS
Crohn's disease, single nucleotide polymorphisms, machine learning

1. Introduction

Crohn's disease is an abnormal immune reaction in the digestive tract, commonly affecting the intestines but known to affect anywhere from the mouth to the anus. An estimated 1 million people in the United States have Crohn's disease, which occurs at a higher rate than other nations for an unknown reason. The exact cause of the disease is not known, but research suggests that factors include people's environments, digestive microbiomes, and genetics. Crohn's disease is known to run in families, as having a parent or sibling with the disease increases one's chance of developing the condition; this suggests that genotype data could be useful for predicting susceptibility to the disease [1].

The vast majority of people's DNA—99.9%—is common to all humans, yet the remaining 0.1% is variable, 80% of which is composed of single nucleotide polymorphisms (SNPs), which can assist in predicting disease. They are each a single base substitution of a nucleotide, representing genetic differences between people. Though some diseases are simply caused by a single SNP, more complicated ones like Crohn's disease can be caused by an unknown combination of SNPs, where each may only contribute a small amount to the phenotype. Analyzing all SNP combinations is infeasible, but machine learning algorithms can help identify the SNPs which are associated with the disease [2].

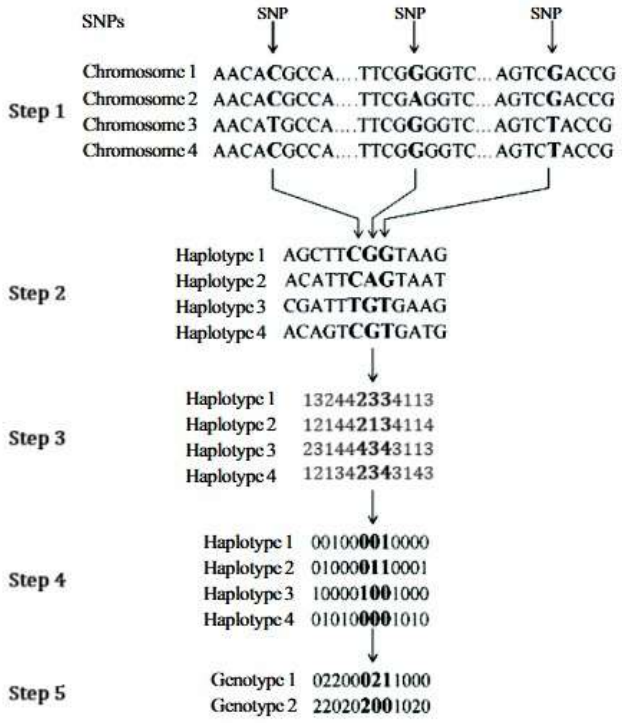


Fig. 1. Identifying SNPs, recording haplotypes, and composing genotypes [2].

We plan to use SNP data originally from Daly et al. [3]. Daly et al. examined a section of the human 5q31 chromosome, which is suspected to carry SNPs linked to Crohn's disease. The researchers studied 129 trios of European descent, each a family with two parents and a child, the child being positive for the disease and at least one of the parents without it; this allowed the researchers to compare and find SNPs [3],
[4].

Mao and Lee [2] explain how chromosomes are compared to find disparities—SNPs—and condense them into haplotypes and genotypes. As seen in Fig. 1, four chromosomes are compared (two per person), and SNPs are identified where the nucleotides in a column do not all match. The SNPs are retained in the haplotype data; other columns are ignored. The four nucleotides of the haplotypes are encoded as numbers 1-4 for our first dataset (Step 3). Then they are encoded in binary, where 0s and 1s are

assigned to the two nucleotides in each column. Each person's genotype is composed of two haplotypes, encoded in ternary for our second dataset (Step 5). In this paper, we test both the haplotype data and the simplified genotype data.

2. Algorithm Background

2.1 K-Nearest Neighbors

K-Nearest Neighrobrs (KNN) is a unique machine learning model since it is a lazy learner and does not require "training" from the data in the same way as the other following methods do. For a training dataset with *m* features, the model projects each observation as a point in an m-dimensional space [5]. Thus, the training portion of this model simply stores data for later use. When a testing observation must be classified, it is imagined in the same space, and the K-nearest points are calculated, often using Euclidean distance (though we also use Minkowski distance, and many other distance metrics exist). K is a prespecified value from the user that must be tuned. A too small value of K only considers a few neighbors and can lead to overfitting, and a large value may include too many neighbors that are much farther from the observation, underfitting the model [5]. When comparing these K points, voting classifies the testing point as the same as the majority of its nearest neighbors [6]. Fig. 2 provides a basic visualization of the model.

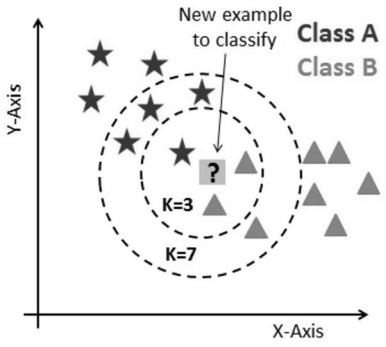


Fig. 2. Diagram of KNN [7].

2.2 Support Vector Machine

Support vector machines (SVMs) are designed to linearly separate two classes of data (viz. sick and healthy groups in this context) by imagining the observations as points in space and constructing a hyperplane with the two classes on opposite sides. Each observation k is represented by its feature vector x_k and class label $y_k = \pm 1$. The algorithm attempts to find the hyperplane that separates the classes with the largest margin between itself and the nearest points of each class, which are called support vectors. The hyperplane is represented by its perpendicular weight vector w and offsetting bias b, graphed as $w^T x + b = 0$ (see Fig. 3). A point's distance from the plane can be found by the decision function $D(x) = w \cdot x + b$, which is positive for

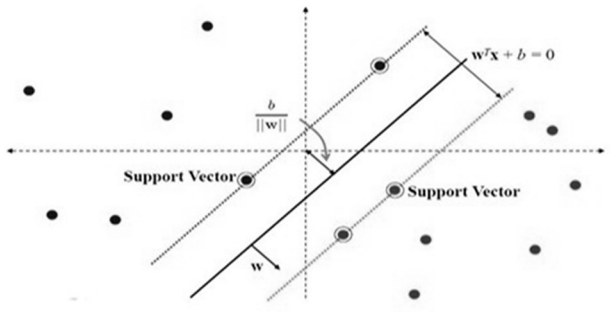


Fig. 3. An example 2-D hyperplane with margins and support vectors [9].

points above the plane and negative for those below. As the Euclidean norm of the weights, $\|w\|$, is inversely proportional to the width of the margin, $\|w\|$ is minimized, subject to the constraint that the observations are correctly classified and on/outside the margin [8].

$$\min_w \frac{\|w\|^2}{2} \text{ subject to } y_k D(x_k) \geq 1$$

By means of the Lagrangian, this optimization problem is rewritten as a dual problem, where inner products—dot products of observation vectors with each other—are calculated instead of the product of each observation with the weight vector, allowing greater flexibility with kernels. Each observation has an associated Lagrange multiplier, λ_k, which is only nonzero for the support vectors, simplifying computation. The objective is now optimized with respect to the Lagrange multipliers (and the bias) instead of the components of the weight vector [5, 8].

$$\max_{\lambda,b} \sum_{k=1}^{n} \lambda_k (1 - y_k b) - \frac{1}{2} \sum_{k=1}^{n} \sum_{l=1}^{n} \lambda_k \lambda_l y_k y_l (x_k \cdot x_l)$$

$$\text{subject to } \lambda_k \geq 0$$

Because not all data is perfectly and linearly separable, slack variables and kernels are introduced to make SVMs more versatile. Slack variables permit some training observations to be within the margins or even misclassified when complete separation is impossible, encouraging the algorithm to minimize the error instead of eliminating it completely. Kernels allow data which are not linearly separable to be interpreted as if they are transformed into higher-dimensional spaces where a linear hyperplane can be constructed to separate the classes. At a high computational cost, such a transformation $\varphi(x)$ could be computed on every observation and a weight vector could be found for the higher-dimensional data, but to save costs, the kernel trick is

used instead. A kernel $K(x_k,x_l)$ is the algebraically reduced form of the (inner) product of transformed observations, which can be inserted back into the dual problem, yielding a new decision function $D(x)$ [5,8].

$$K(x_k, x_l) = \varphi(x_k) \cdot \varphi(x_l)$$
$$\sum_{k=1}^{n} \lambda_k (1 - y_k b) - \frac{1}{2} \sum_{k=1}^{n} \sum_{l=1}^{n} \lambda_k \lambda_l y_k y_l K(x_k, x_l)$$
$$D(x_k) = \sum_{l=1}^{n} \lambda_l y_l K(x_k, x_l) + b$$

The new decision function considers a point's (x_k's) relation to the support vectors through a kernel instead of directly comparing the point's distance from the hyperplane.

2.3 Random Forest (RF)

The RF model, based on the idea of bagging or bootstrap aggregating, produces a collection (or "forest") of multiple decision trees and uses the method of voting as a means of classifying data. Bootstrapping involves taking a random sample of the observations with replacement that is equal in size to the dataset [5]. For each tree in the forest, a unique bootstrap sample of the data is taken for training. In order to increase the decorrelation between individual trees in the forest, a random sample of the input features is also taken prior to each split of the tree [5]. This subset is considerably smaller than the total number of features and maintains a constant size throughout the forest (this size is indicated by a hyperparameter of the model). A Gini index is then calculated to determine which of these features should be used to provide an optimal split in the tree. This index is a measure of the impurity of a node resulting from a split, where an index of 0 indicates true purity (the node has observations all belonging to the same class). The equation for calculating the Gini index is listed below where c represents the number of classes, t is the given node, and $p_i(t)$ is the fraction of observations of class i at that node [5]:

$$Gini = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

The feature that produces the highest magnitude gain—the difference between the impurity of a parent node and the collective sum of the impurity of the children—is chosen as the split feature [5]. Because the RF model does not include pruning, this splitting process simply continues until each terminal node is pure [10] (though hyperparameters such as max _depth can be adjusted to cause a premature stop in tree growth). After creating all trees, each test observation is classified by each tree individually, and the model finally classifies each observation as the label most frequently predicted by the trees (i.e., the label with a plurality vote).

2.4 Extreme Gradient Boost

Similar to Random Forest, the XGBoost (or XGB) model uses a series of decision trees to reach a classification conclusion. However, this model is based on boosting rather than bagging. With boosting, the trees are created sequentially rather than independently, and each is trained on the weighted errors of the preceding tree in order to gain increasingly accurate classifications and reduce the overall loss of the model [5]. For example, after the creation of the first tree, the residuals (calculated by a loss function of the predicted classifications and the actual classifications) are calculated, and a second tree, trained on those residuals as a corrective measure, is added to the first to produce more accurate predictions. This process is then repeated for several rounds, and a final classification can be made with the last tree. When creating the individual trees, XGB does not use Gini impurity like RF; rather, it takes advantage of another greedy algorithm to find the best splitting feature. This algorithm is detailed by Chen and Guestrin in the original XGB paper [11]. Essentially, the algorithm searches through the features for optimal splits by calculating the gains of potential child nodes and finding the greatest possible gain among them. The gain is a measurement of how much the new tree predicts (or improves) the residuals of the previous tree, based on the first two derivatives (i.e., the *gradient*) of the loss function, a function that evaluates the residuals.

There are several aspects of this model that distinguish it from other tree-based models, particularly in its dedication to reduce complexity. Firstly, it includes two regularization parameters, γ and λ, for the user to tune for penalizing complexity in the decision trees [11]. Also, unlike RF, XGBoost does use pruning in order to reduce overfitting and further regularize the model. Additionally, XGB manages the impacts of each tree in the model through a method called shrinkage, where the learning rate hyperparameter causes each consecutive tree to have a smaller impact than its predecessor, decreasing the chances of overfitting and encouraging the model to grow more trees [11].

3. Experiment and Results

3.1 Performance Metrics

As the basis for our performance analysis and comparison of the different machine learning algorithms, we utilized the confusion matrix tool from Scikit-Learn. A confusion matrix divides the model's predictions into four categories: true positive, false positive, true negative, and false negative (TP, FP, TN, and FN respectively). TP and TN refer to the predictions that were correctly classified while FP indicates the predictions that were incorrectly classified as positive, and FN inversely refers to observations incorrectly classified as negative [5].

From these counts, we are able to calculate accuracy, sensitivity, and specificity as our main measures of model performance, as seen in the equations below [5].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
$$Sensitivity = \frac{TP}{TP + FN}$$
$$Specificity = \frac{TN}{TN + FP}$$

Accuracy refers to the overall correctness of the model's predictions—-both positive and negative. Meanwhile, sensitivity measures the percentage of positive cases that were correctly classified, and specificity measures the percentage of negative cases that were correctly classified [5]. In the context of the dataset, sensitivity then represents the model's ability to correctly predict Crohn's disease in a sick person, and specificity is the model's ability to correctly predict the absence of the disease in a healthy person.

Finally, we visualized each model's performance with a receiver operating characteristic (ROC) curve, which plots sensitivity along the y-axis and the false positive rate (1 minus specificity) along the x-axis for various classification thresholds [5]. The models produce a probability score for each point representing the confidence that it is in the positive class, and instead of simply predicting an observation to be in the more likely class (a probability threshold of 0.5), the probability is compared to many different threshold values between 0 and 1. Sensitivity and false positive rates for the data are taken for each value of the threshold, which serves as a parameter for the ROC graph and is not plotted. A greater area under the curve (AUC), approaching 1, indicates better model performance, while an area closer to 0.5 indicates a poor classifier.

3.2 Base Experiment

We first examined the haplotype data and tested each of the four algorithms in Python, using Scikit-Learn for KNN, SVM, and RF, and the XGBoost package for XGB. Table I shows our results from tuning the parameters of each for their best accuracies. We recorded the accuracy, sensitivity, specificity, area under the ROC curve, training time taken, and total (training & prediction) time taken for each model. We tried each algorithm with and without one-hot encoding and recorded the better result. One-hot encoding replaces each feature with a binary vector, where a 1 indicates the presence of the category, and a 0 shows its absence [12]. For example when a dataset that has four categories per feature is encoded with one-hot, the number of features is quadrupled, each with two categories. Removing any zero columns helps reduce the sparsity.

For KNN, we used distance and found $\rho = 2$ (i.e., Euclidean distance) and $K = 28$ to be optimal for this dataset. We also found that using data without one-hot encoding yielded better results than with it. KNN has low training times and comparatively high prediction times as it is a lazy learner.

For SVM, we tested linear, radial-basis function (RBF), and polynomial kernels with the best overall result from RBF with $\gamma = 0.17$ and using one-hot encoding on the data.

To optimize RF performance, we experimented with several different hyperparameter combinations. We found that the most optimal combination for accuracy and sensitivity required the change of only one hyperparameter, leaving the rest as default. The "max _ features" hyperparameter, which determines how many features the model should consider per node split, takes the square root of the total number of features by default, but we changed the setting to a binary log to improve accuracy. Additionally, we used one-hot encoding to transform the unordered categorical variables and noted a slight improvement in accuracy from doing so.

Table I. Best Accuracies

	Acc.	Sens.	Spec	ROC AUC	Train Time	Total Time
KNN	.835	.657	.952	.874	.005	.013
SVM	.866	.886	.855	.927	.054	.067
RF	.928	.886	.952	.947	.133	.140
XGB	.938	.943	.936	.947	.068	.070

Like with RF, we tuned the XGBoost hyperparameters to find an optimal combination for accuracy and sensitivity. The combination we selected includes a default learning rate of 0.3, a maximum tree depth of 3, a gamma value of 0.1 for regularization, a column subsampling rate per tree of 0.5, and the minimum weight for the child nodes of 3. We kept all other values as default and applied one-hot encoding.

3.3 Feature Reduction

With the primary goal of improving time efficiency, we sought feature reduction methods for the algorithms. Rather than continuing to train the models on all 206 SNPs provided in the genetic data, we utilized a smaller subset of only the most important features. The objective was to reduce classification time without greatly sacrificing accuracy or sensitivity. We incorporated this method into the tree-based models of RF and XGB since both have built-in algorithms for ranking feature importance. For RF, the more important features are the ones that result in the better splits in the model by reducing impurity. Meanwhile, for XGB, the important features are those chosen more frequently for splitting. In both cases, these are the features contribute more impact towards the accuracy of model predictions. In the context of the dataset, these features are the SNPs that are more highly correlated with Crohn's disease.

We also experimented with an alternative method of feature ranking, permutation feature importance (PFI), to observe if it had any unique impacts on accuracy or efficiency. PFI is also included in the Scikit-Learn library but does not rely on impurity measures or better splits as the previous feature ranking method did. Instead, PFI permutes a feature's values and observes its impact on the model's

performance. If the model produces worse predictions with the permuted feature, it is clear that the feature must have been impactful toward the model's original decision, indicating that feature's importance [13]. After determining the mean importance for each feature, we were able to remove any features that had 0 or negative weights. Thus, we trained the models on a subset of only the most important (or positive-weighed) features. A benefit of this method over the previous is our ability to extend it to SVM in addition to RF and XGB as it is not a tree-based method.

1) Random Forest: Prior to one-hot encoding, we recorded the top 100 features calculated by the built-in ranking method

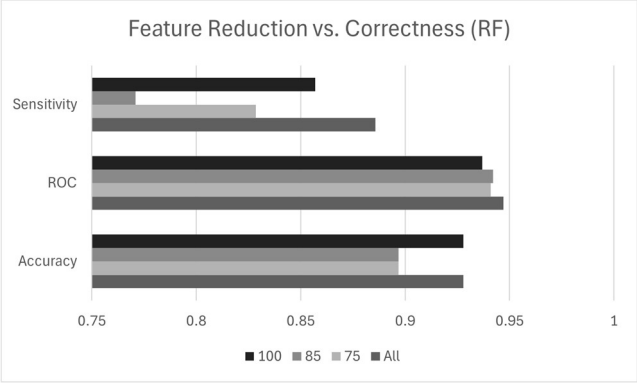


Fig. 4. Accuracy and Sensitivity comparison for RF subsets

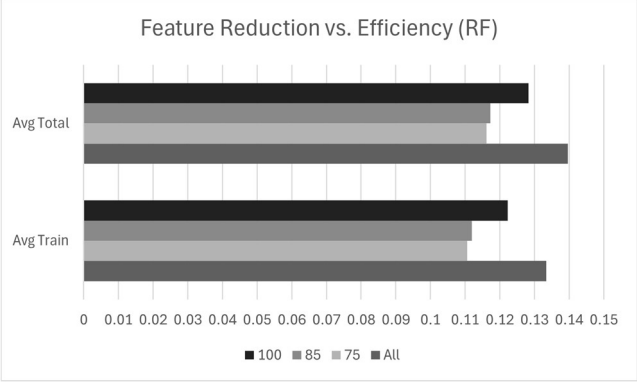


Fig. 5. Time comparison for RF subsets

and, decrementing by five features at a time, determined the ideal number of SNPs on which to train the model. When comparing the accuracy, sensitivity, and ROC values for the feature subsets, subsets of size 75, 85, and 100 were found to be the most optimal. The performance metrics of these values are visualized in Fig. 4. As seen in the chart, even the best subset sizes resulted in some sacrifice of ROC and especially sensitivity. However, the subset of 100 features was notably able to maintain an accuracy of 0.928. Focusing on the aforementioned feature subsets, we then recorded the training

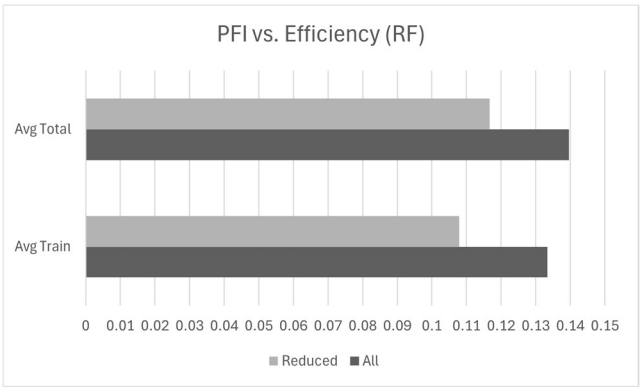


Fig. 6. Time comparison for RF with PFI

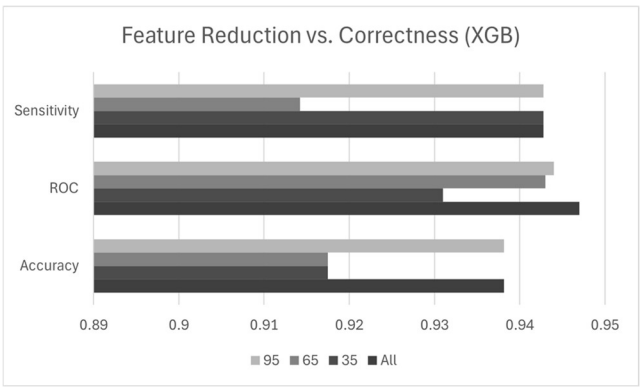


Fig. 7. Accuracy and Sensitivity comparison for XGB subsets

and total run time for each. Fig. 5 compares the different time efficiency improvements, showing about a 1-2 millisecond change. Although faster than using all 206 features, this minute time improvement may not justify the accuracy and sensitivity sacrificed by using smaller feature subsets.

Testing our other ranking method, we found that we were able to maintain an accuracy of 0.928 when using PFI (though distinctly after one-hot encoding rather than before, as the previous method had done). Although sensitivity decreased from 0.886 to 0.857, the ROC actually increased from 0.947 to 0.964. However, as seen in Fig. 6, the time efficiency still did not demonstrate a significant improvement from the reduction of features.

2) *XGBoost:* We conducted a very similar experiment for XGB, though the model's built-in feature ranking was slightly different from that of RF, so XGB identified a different set of top features to train the model. As such, the ideal subset sizes (35, 65, and 95) were also different. As seen in Fig. 7, some performance was again necessarily sacrificed with the removal of so many features. Fortunately, subset size 95 was able to maintain the same accuracy and sensitivity and experienced a decrease of only 0.003 in ROC. Additionally, the feature reduction method's efficiency was even more promising for XGB than for RF, decreasing time

by nearly half, as seen in Fig. 8. Thus, the feature reduction method proved to be more justifiable and efficient for XGB than for RF.

Similarly, when examining XGB with the PFI method after encoding, we were able to maintain constant accuracy and sensitivity despite the decrease in features. Furthermore, we observed a ROC drop of only 0.002. Also, the model's time was reduced by more than half as seen in Fig. 9. As a result, the PFI method demonstrates superiority in improving efficiency without sacrificing performance for XGB.

3) *Support Vector Machines:* For SVM, we used the builtin permutation importance tool from Scikit-Learn. The results of the tool revealed that many of the features had zero impact on the model, and some of them even had a negative impact. By removing all features that were marked as having zero or negative impact, leaving only 24 positive features, the model's accuracy improved, as seen in Fig. 10, although sensitivity

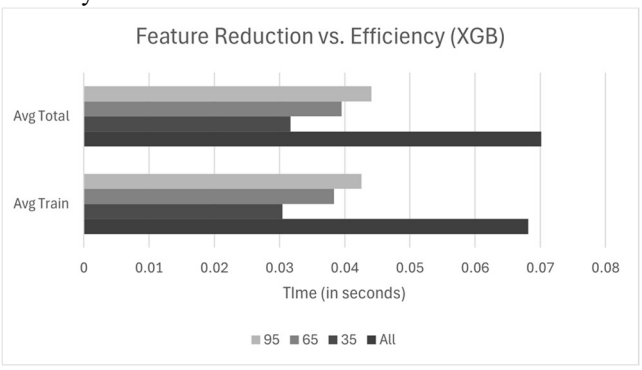


Fig. 8. Time comparison for XGB subsets

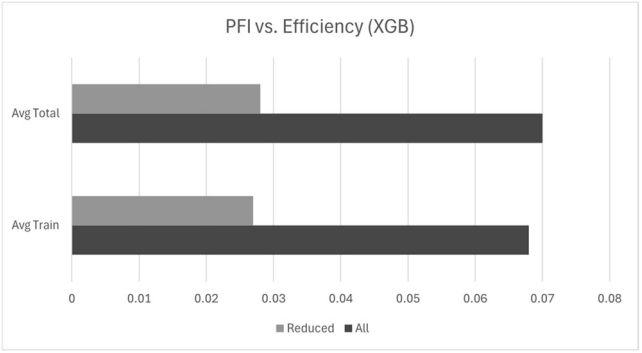


Fig. 9. Time comparison for XGB with PFI

and ROC AUC suffered. We found that leaving a few more features in, keeping the top 43, produced the optimal accuracy and ROC, with a high sensitivity as well. This suggests that the PFI tool was not completely accurate at predicting which features had no effect, but it did help. We also tried groups of 35 and 65 features; both tests performed lower than the 43-feature group. Feature reduction also cut the training and total times of the models roughly in half, seen in Fig. 11.

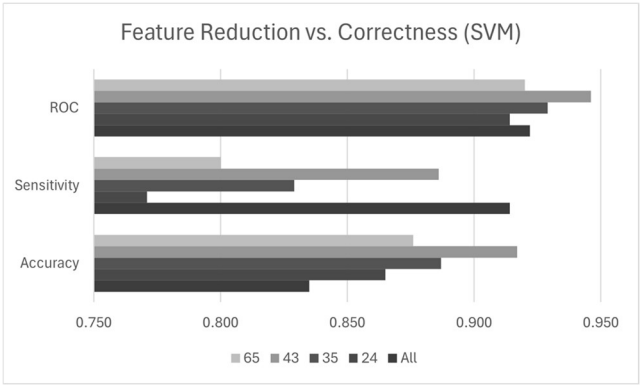


Fig. 10. Accuracy and Sensitivity comparison for SVM subsets

4) K-Nearest Neighbor: KNN is more rudimentary and does not have a clear existing metric for feature importance. Bhardwaj et al. [14] used Random Forests to find quantities for each feature's importances before using them as weights

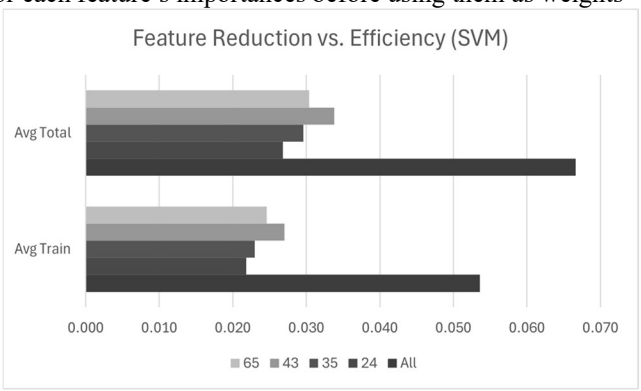


Fig. 11. Time comparison for SVM subsets

on the KNN model. They found marginal improvements in accuracy for most of the datasets they tested. However, because RF and KNN are fundamentally different, features that are most important to RF might be different than those that are important to KNN, so we considered identifying a way to find feature importance with KNN itself.

To find feature importances, we first created a simple metric to evaluate the training data with and without specific features. For each observation, we found the Euclidean distances between it and all other observations. Then, using the K nearest neighbors (for various values of K) of each observation, we classified the observation as if it was test data. Comparing these predicted labels for all observations to the actual labels yielded a sort of inner-accuracy score—the ability of the data to classify itself. For example, at K = 40, this value is 0.790 for our training data. Typically, the inner accuracy is slightly higher than the accuracy of the model on the test data. Once we found this overall inner-accuracy score, we removed each feature one at a time and

re-scored the data. If removing a feature caused the inner accuracy to decrease, we reasoned that the feature was important; if the opposite was true, the feature was likely unimportant or harmful to accuracy.

We used these differences in inner-accuracy scores to find an array of weights to apply to features when using KNN to classify the test data. Each weight was calculated as the following, where w_j is the weight for feature j, A is the overall inner accuracy, A_j is the inner accuracy calculated without feature j, and s is a scale set by the user.

$$w_j = 1 + s(A_j - A)$$

Once we found the weights, we calculated the squared distances between the test data and training data. To save time, we omitted taking the square root because the distances were only compared to each other and were not used in any other calculations. x and x' each represent a single data point.

$$D(x, x') = \sum_j w_j (x_j - x'_j)^2$$

The results were inconsistent, but many values of K showed improvement. For example, the best result we found was at $K = 40$. Without weights, our algorithm reached 75.3% accuracy, but with weights and a scale of $s = 160$, the accuracy was as much as 86.6%, a considerable improvement. However, other values of K, like $K = 50$, tended to decrease in accuracy with weighting. Some improved with a negative scale value rather than a positive one; this is often the case when the inner accuracy is lower than the test data accuracy. A summary of some of our results is shown in Table II. We will investigate this modified algorithm more in the future.

Table II. KrazyNN Results

K	Inner Acc.	s	Test Acc.	Sens.	Spec.	ROC AUC	Weight Time	Test Time
20	.790	0	.742	.686	.774	.856	0.000	0.016
20	.790	90	.783	.800	.774	.854	0.414	0.015
40	.790	0	.753	.571	.854	.861	0.000	0.015
40	.790	160	.866	.714	.952	.902	0.793	0.017
40	.790	200	.845	.657	.951	.890	0.773	0.016
50	.782	0	.794	.542	.935	.879	0.000	0.020
50	.782	10	.763	.571	.871	.872	1.020	0.010
50	.782	-10	.814	.571	.952	.894	1.006	0.013

Table II contains some of our results from the modified KNN algorithm, dubbed KrazyNN. K and s are independent variables, and where $s = 0$, weights were not calculated. The inner accuracy score and the times are not dependent on s. The weight time represents the seconds spent to calculate the weights for the data at a specific K value, and it correlates directly with the value of K. 40 was the best K value both with and without weights, and $s = 160$ scaled the weights for the best accuracy. $K = 20$ achieved an identical inneraccuracy score to $K = 40$ but lower accuracies on the test data. $s = 90$ proved to be the optimal scale for $k = 20$, and it produced a better sensitivity than any of our other KNN or KrazyNN tests.

Overall, the KrazyNN model has the potential to improve KNN accuracies, but it has some disadvantages. Despite only having two parameters—K and s—they are difficult to tune for the maximum accuracy, and occasionally no positive s value that benefits the accuracy can be found, suggesting that the weights are flawed. The weight-finding algorithm is also quite time-consuming, taking as much as a second for higher K values, although this is much faster than using PFI for other algorithms, which can take minutes on the same data.

2.4 Genotype Encoding

So far, our testing has focused on the haplotype data, but we also investigated the converted genotype data. As illustrated by Fig. 1 in the introduction, the data we obtained were interpreted and encoded from two lines of binary haplotype data into a single line of ternary genotype data for each observation. Matching 0s from both haplotypes were encoded as 0 in the genotype, a pair of 1s was encoded as 1, and a pair of both was encoded as 2. Naturally, these are regarded as nominal categories, and they suit the ternary decision trees used by Mao and Lee [2]. However, because the implementations of
RF and XGB that we chose use binary decision trees, and because KNN relies on distance metrics, we considered other ways to encode the data.

First, we tested the accuracy and ROC AUC for the genotype data without transforming or encoding it at all. Results are shown in Table III. The scores are much lower than those from the haplotype data, but we still worked on improving them slightly.

To improve the accuracies, we first tried interpreting the categories ordinally by swapping all the 1s and 2s in the genotype data. This means that the heterozygous (0-1 or 10) pairs, now encoded as 1, are between the homozygous (0-0 & 1-1) pairs, now encoded as 0 and 2. Our rationale was that the heterozygous pair was logically halfway between the homozygous pairs. With this ordering, a decision tree might more easily choose a split based on whether or not a specific nucleotide was found in either haplotype (while only looking at the genotype). KNN, relying on distance measures, would consider two different homozygous pairs to be more dissimilar than a homozygous and a heterozygous pair.

Testing each of the algorithms, we found that this is correct for some cases: As seen in Table III, the results either stayed the same or improved in accuracy with the change. Using the same parameters as our initial testing, KNN improved considerably for $\rho = 2$. KrazyNN delivered an exceptional accuracy for $k = 19$ and $s = -30$ on the swapped data
(which had an inner accuracy of only .624). With those same parameters, the original data earned an accuracy of 60.8%. Other parameters ($k = 31$, $s = 20$) earned 68.0% accuracy on the original data, which is included in the table as the best

example. SVM showed no improvement in accuracy with the translated data until γ was changed from 0.2 to 0.5, which was the optimal value for the translated data. ROC AUC decreased nonetheless. Similarly, we observed no change in accuracy with RF (.618) using the same number of estimators (125) as our initial testing. However, by decreasing the number of estimators to 75, accuracy improved for the swapped data, although ROC AUC decreased. XGB remained unchanged using the same parameters as before.

Table III. Genotype Encoding Results

Algorithm	Original Accuracy	Swapped Accuracy	1-hot Accuracy	Original ROC	Swapped ROC	1-hot ROC
KNN ρ =2	.680	.711	.639	.655	.670	.573
KNN ρ =1.7	.691	.701	.629	.639	.621	.583
KrazyNN	.680	.722	N/A	.659	.673	N/A
SVM γ =0.2	.639	.639	.649	.520	.514	.522
SVM γ =0.5	.618	.649	.629	.504	.509	.489
RF n =125	.618	.618	.629	.549	.522	.534
RF n =75	.608	.629	.649	.576	.537	.544
XGB	.608	.608	.588	.489	.489	.529

Secondly, we sought to improve results by running the same test with one-hot encoded data. This forces the three categories to be treated nominally by the classifiers. Each feature is split into three features, which are Boolean values representing which categories an observation is in and not in. This method benefited the accuracy of SVM slightly and RF considerably while harming the accuracy of KNN and XGB. No value of s could improve the KrazyNN accuracy over $s = 0$, so it is left out of the table.

4. Conclusion

Through a series of experiments, including genotype encoding and multiple methods of feature reduction, this paper sought to determine the most accurate and efficient model for classifying Crohn's disease solely based on a patient's genetic data. After using both the haplotype and genotype data, we can confidently conclude that the models trained on the haplotype data yield more accurate predictions. However, if using the genotype data instead for its fewer features, we found that the swapped dataset provides slight accuracy improvements, especially for the KrazyNN model.

Focusing on the haplotype dataset, the most accurate models are XGB, RF, and, when run on 43 features selected by PFI, SVM. Though it is the least accurate, KNN is the most timeefficient of all the models we tested and would be useful in cases where speed is prioritized over accuracy. For the best combination of efficiency and performance, we found that XGB with its features reduced by PFI retains the highest accuracy of all the models and is the fastest after KNN. The second best model is SVM trained on 43 features since it runs nearly as fast as XBG while being only around 2% less accurate. As stated previously, although the RF model performs slightly better than the SVM model, its efficiency did not improve much by reducing features.

References:

[1] Crohn's Disease. National Institute of Diabetes and Digestive and Kidney Diseases. https://www.niddk.nih.gov/health-information/digestivediseases/crohns-disease/

[2] W. Mao and J. Lee, A Combinatorial Analysis of Genetic Data for Crohn's Disease, *Journal of Biomedical Science and Engineering*, 1, 2008, 52-58, doi: 10.4236/jbise.2008.11008.

[3] M. Daly et al., High-resolution haplotype structure in the human genome, *Nature Genetics*, 29, 2001, 229–232. doi: 10.1038/ng1001-229.

[4] J.D. Rioux et al., Hierarchical linkage disequilibrium mapping of a susceptibility gene for Crohn's disease to the cytokine cluster on chromosome 5, *Nature Genetics*, 29, 2001, 223–228.

[5] P. Tan, M. Steinbach, A. Karpatne, and V. Kumar, Introduction to Data Mining, 2nd ed., New York, NY: Pearson Education, 2019.

[6] T.M. Cover, P.E. Hart. Nearest neighbor pattern classification. IEEE Trans. Inform. Theory, IT, 13(1), 1967, 21–27.

[7] A. Christopher (Dec. 29, 2021). K-Nearest Neighbor. Medium. https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4

[8] B.E. Boser, I.M. Guyon, and V.N. Vapnik, A training algorithm for optimal margin classifiers*, presented at the fifth annual workshop on Computational learning theory (COLT '92). Association for Computing Machinery*, New York, NY, USA, 1992, pp. 144–152, doi: 10.1145/130385.130401.

[9] M. Jabardi, Support Vector Machines: Theory, Algorithms, and Applications, *Infocommunications Journal*, 17(1), 2025, 66-75, doi: 10.36244/ICJ.2025.1.8.

[10] A. Cutler, D.R. Cutler, and J.R. Stevens, Random Forests, in *Ensemble Machine Learning*, C. Zhang and Y. Ma, Eds., New York, NY, USA: Springer, pp. 157-176, 2011, doi: 10.1007/978-1-4419-9326-7 _5.

[11] T. Chen and C. Guestrin, XGBoost: A Scalable Tree Boosting System, *presented at the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Association for Computing Machinery*, New York, NY, USA, 2016, pp. 785–794, doi: 10.1145/2939672.2939785

[12] F. Pargent, F. Pfisterer, J. Thomas et al., Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features, *Computational Statistics,* vol. 37, pp. 2671–2692, Mar. 2022. https://doi.org/10.1007/s00180-022-01207-6

[13] A. Fisher, C. Rudin, and F. Dominici. All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously, *Journal of Machine Learning Research*, vol. 20, no. 177, pp. 1-81, Dec. 2019.

[14] C.A. Bhardwaj, M. Mishra, and K. Desikan, Dynamic Feature Scaling for K-Nearest Neighbor Algorithm, *presented at the International Conference on Mathematical Computer Engineering*, Langkawi, Malaysia, 2017, doi: 10.48550/arXiv.1811.05062.

Examining Position Bias in LLMs through an AI Stylist

Katherine Powell and Samuel Grieggs
Indiana University of Pennsylvania
yppdc@iup.edu, sgrieggs@iup.edu

ABSTRACT
With the rise in Large Language Models (LLMs) being used for recommender systems, there has been an increased effort to examine their many biases. Position bias is one of these many biases, which is when a model can be sensitive to the order in which the candidates to be recommended are presented, which will affect the overall ranking results. After creating an LLM recommender system in the form of a fashion stylist, we have examined the stylist to determine to what extent it is affected by the position of the candidates.

KEY WORDS
LLM, Recommender, Fashion, Position Bias

1. Introduction

With the rise of large language models, they have been used increasingly in recommender systems. As these recommender systems have gained more popularity, there has been an increased effort to examine their many biases, one of which is position bias. Bito, E., Ren, Y., and He, E. define position bias as: Large Language Models' (LLMs') tendency to rely on the order of candidates in the input rather than their actual relevance to the prompt given [1], [2]. As position bias has been increasingly studied, studies have found that LLMs prioritize the candidates at the top of the list [3].
With position bias being very prevalent within LLM recommender systems, we decided to evaluate our AI fashion stylist for position bias. The AI stylist is a LLM recommender system utilizing LLMs zero shot capabilities. The LLMs are given a system prompt to act as a highly trained personal stylist. It is then asked to recommend different outfits based on event-based parameters such as the event itself, the formality, the weather, and the city in which the event is based. These outfits are made from an imported closet which is made up of descriptions of the different clothing items. These descriptions are generated based on images of the clothing items using the 27 billion parameter variant of the multimodal LLM Gemma 3 [4] and include color, season, style, fabric, cut, and occasion. The models used for the AI stylist are easily interchangeable for any Ollama LLM. Through the AI stylist, we will examine the extent to which Gemma 3 and gpt-oss [5] are affected by position bias.

2. Related Works

Position Bias is a well explored problem in the context of Large Language Models. Liu et al. [2] demonstrated that models tend to have trouble referencing information in the middle of their context windows, heavily emphasizing information that appears in the beginning or end of their context windows. This idea of position sensitivity was also explored by Wang et al. [6] who demonstrated that the bias was strong enough that LLM based evaluation systems could be hacked via altering the order in which the information appears.
Despite the fact that Position Bias is a real problem when using LLMs for recommendations, they have been widely used in the literature for various recommender systems, and have been demonstrated to work sufficiently well, even with the inherent limitations. Some examples of this can be seen in the Lin et al. [7] and Zhao et al. [8] surveys that contextualize some of the methodologies used to perform this task.
Specifically, using LLM's as fashion recommender is also a relatively well explored problem, with work by Liu et al. [9] representing one highlighted example, and Deldjoo et al. [10] offering a broader survey.

3. Experiment

The goal of our experiment was to determine if the AI Stylist's choice in recommending clothing items was affected by the position of the clothing items in the list. For this experiment, we used Ollama's large language and vision [4] [5]from OpenAI.
First, we came up with a prompt for what kind of event, weather, location, and formality the AI Stylist would be styling outfits for. The situation was: going out for coffee with friends, trying to look stylish while still being mostly casual. The location was Pittsburgh, PA, and the weather was 30 degrees Fahrenheit and windy. Then, we

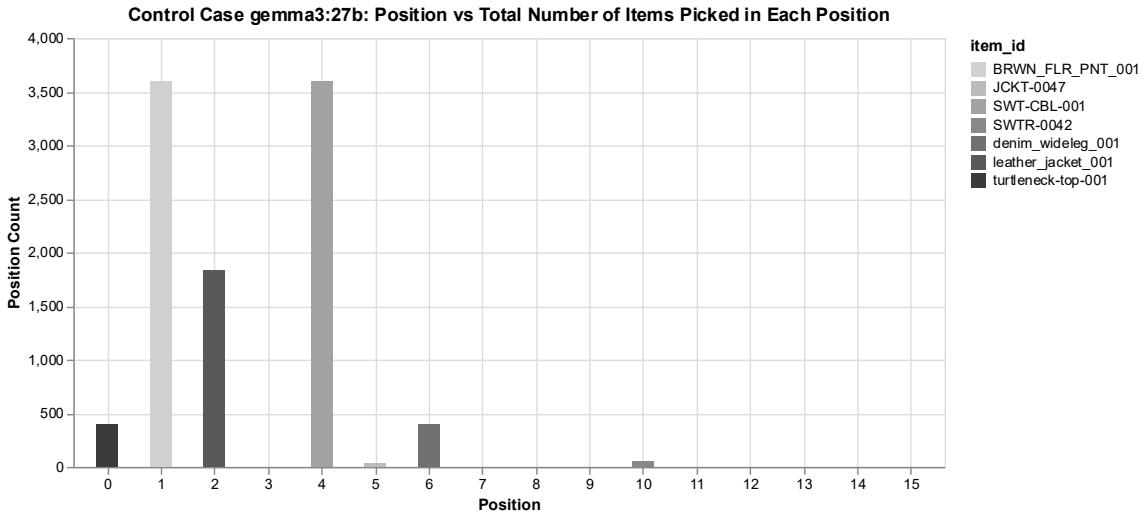


Figure 1: Gemma 3 Control Case

Results from Gemma 3 when object order was not shuffled.

generated 10 similar-sounding prompts, with the Gemma 3 model containing the same key information.

Then, based on that prompt, we compiled a closet of 16 items, which were then converted from images into AI-generated text descriptions using the Gemma 3 model. The text descriptions of the items, including cut, color, season, occasion, and a unique item id. 14 of the 16 clothing items were very suitable for the cold weather and formality expected, while the other two were clothing items that were suited for warmer weather. These two items were denim shorts, which have the item id "denim_short_001", and a tank top, "top-ribbed-squareneck-goldtrim-001".

To start examining the position bias in AI Stylist, we first had to adjust the system prompt, so that the model would produce the chosen items in a JSON file format instead of normal text. This ensured that chosen items could be analyzed more easily as data points. The AI Stylist picks 2 to 4 clothing items, enough to make a full outfit each time it is prompted. For easy of analyzing we have decided to separate these groups of clothing items into induvial data points.

From there, we created the control case for testing for position bias. First, the AI-Stylist would be fed

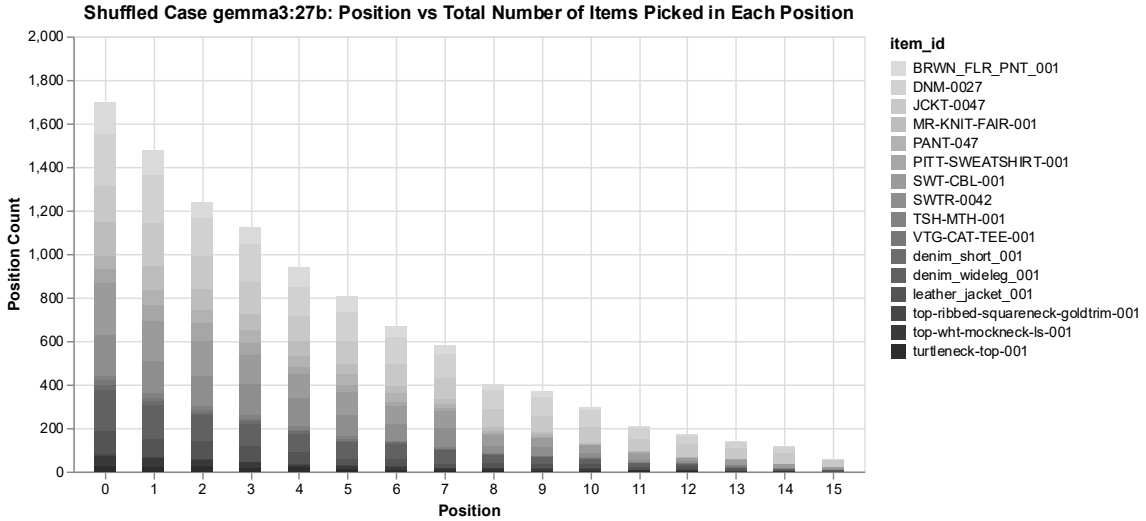


Figure 2: Gemma 3 Shuffled Case

Results from Gemma 3 when the order in which objects appear in the closet are shuffled before each run.

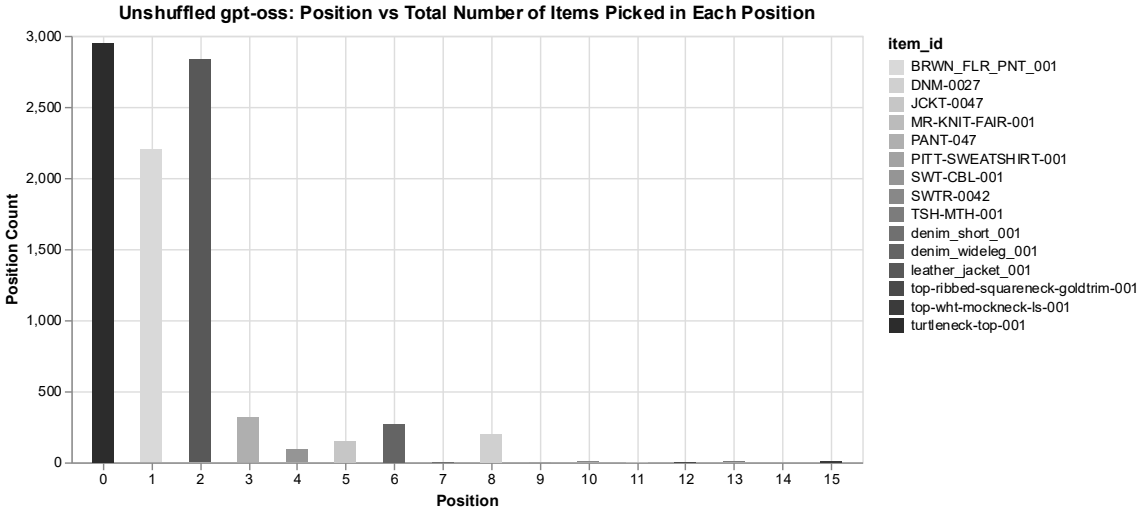


Figure 3: gpt-oss Control Case

Results on gpt-oss when the data was not shuffled.

the system prompt that describes how to be a fashion stylist and how to format the output into a JSON file. The Stylist is then fed the JSON file containing a list of all the clothing items and their descriptions, and one of the 10 different prompts. The 10 different prompts ensure that the Stylist picks clothing items based on the parameters of event, location, weather, and formality, and not based on the prompt being the same wording. Then, the Stylist outputs the items chosen in the JSON file format. , an order number is appended to the clothing item based on the position it is in the closet. Then the output is appended to a list used to store all the chosen items. This series of steps was then repeated 4000 times, and the list of the chosen items are exported into a JSON file.

Next, we created the shuffled case, which purpose was to show if the AI Stylist was affected by position bias. The beginning of the shuffled case starts similarly to the control case, with the system prompt being sent to the AI-Stylist. Then, a copy of the closet JSON file is made and is shuffled. The shuffled closet is sent to the Stylist, along with one of the 10 prompts. After receiving the prompts and closet, the Stylist outputs the items chosen in the JSON file format. Then, an order number is appended to the clothing item based on the position it is in the shuffled closet. After the output is appended to a list used to store all the

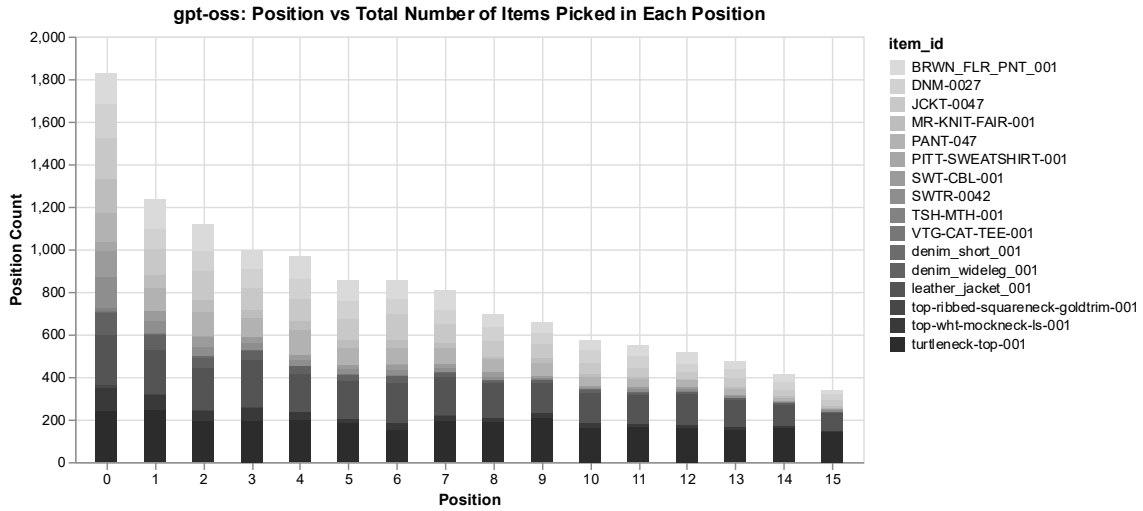


Figure 4: gpt-oss Shuffled Case

Results from gpt-oss when the order in which objects appear in the closet are shuffled before each run.

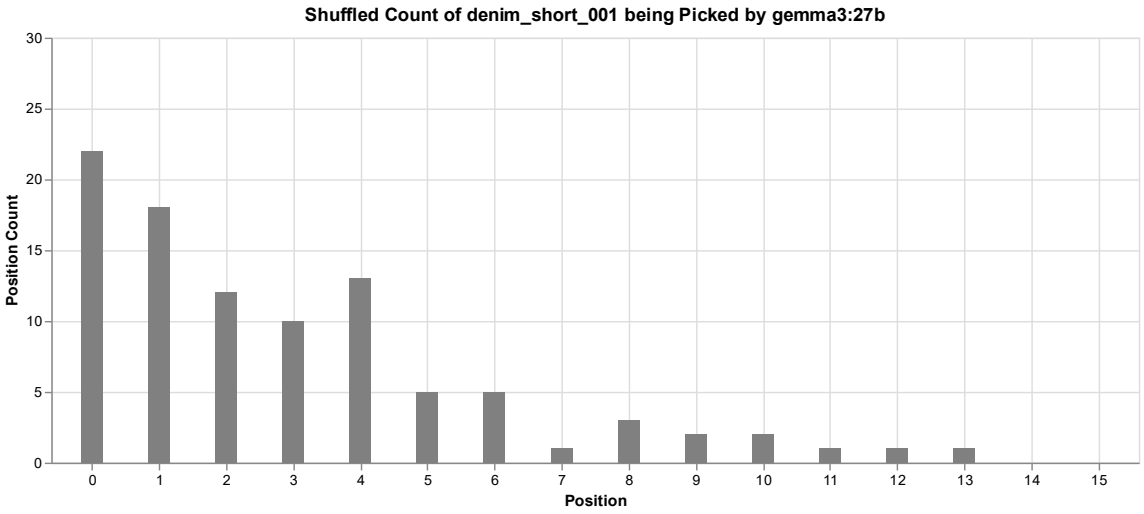


Figure 5: Gemma 3 Denim Shorts

This chart looks at the frequency at which Gemma 3, chose Denim shorts as part of the outfit, this is an inappropriate choice based on the prompt, which indicates a cold day.

chosen items. This series of steps was then repeated 4000 times, and the list of all the chosen items is exported into a JSON file.

Using the methods described above, we tested Ollama's large language models, Gemma 3 and gpt-oss. After examining the data, we found that both models had a significant position bias, favoring candidates towards the front of the list.

4. Results

When using the Gemma 3 as the AI Stylist, the model chose 9,910 items in the control case and 10,327 items in the shuffled case. In the control case, the model did not alter any of the item ids or hallucinate items that did not exist in the closet. However, in the shuffled case, the model altered item ids or hallucinated items 42 times. For the shuffled case data, we have removed the items with altered item ids or that were hallucinated from all graphs and totals. This makes the new item total for the shuffled case 10,282.

In the control case, as seen in figure 1, Gemma 3 only picked 7 of the 16 positions. The most often picked positions were position 1, “BRWN_FLR_PNT_001”, position 4, “SWTR-CBL-001", and position 2, “leather_jacket_001”, picking position 1 and position 4 3,600 times and position 2 1,829 times. These top-picked positions held items that were the first items of their class that appear in the closet list. “BRWN_FLR_PNT_001” is the first pair of pants in the closet list, and “leather_jacket_001” is the first jacket to appear in the closet list. The interesting thing is that “SWTR-CBL-001" is the first sweater to appear in the closet list, but not the first top. The first top was in position 0 and was "turtleneck-top-001". This could be the model determining that a sweater would be better to wear on a 30-degree day in Pittsburgh, PA, rather than a more lightweight turtleneck top. These top-picked positions already imply that there could be a position bias in this model towards the front, since the top-picked items are coming from the front of the list.

In the shuffled case as seen in figure 2, the top-picked position is 0, then 1, then 2, so on and so forth, with position 15, the last position, being the least-picked position, with only 56 picks. Position 0 was picked 1,698 times, position 1 was picked 1,475 times, and position 2 was picked 1,236 times. Positions 0, 1, 2, and 3 make up 53.8% of the total number of items picked, 10,282. A fourth of the positions represent 53.8% of the total number of items picked, supporting the idea that Gemma 3 has a position bias towards the front of the list. Since the closet is shuffled every time the model is called, it does not matter what item of clothing is in these lower-number positions.

It will pick these positions towards the front of the list so much so that the model will pick items that are not very suited to the situation that it was given. The 5 shows the rate at which denim shorts are picked for a 30-degree day in Pittsburgh based on the different positions the denim shorts were in. It was picked 96 times, making the jean shorts 0.9% of the total items picked. When the jean shorts were picked, 64.6% of the time they were in positions 0, 1, 2, or 3. A fourth of the positions make up 64.6% of the times that the jean shorts

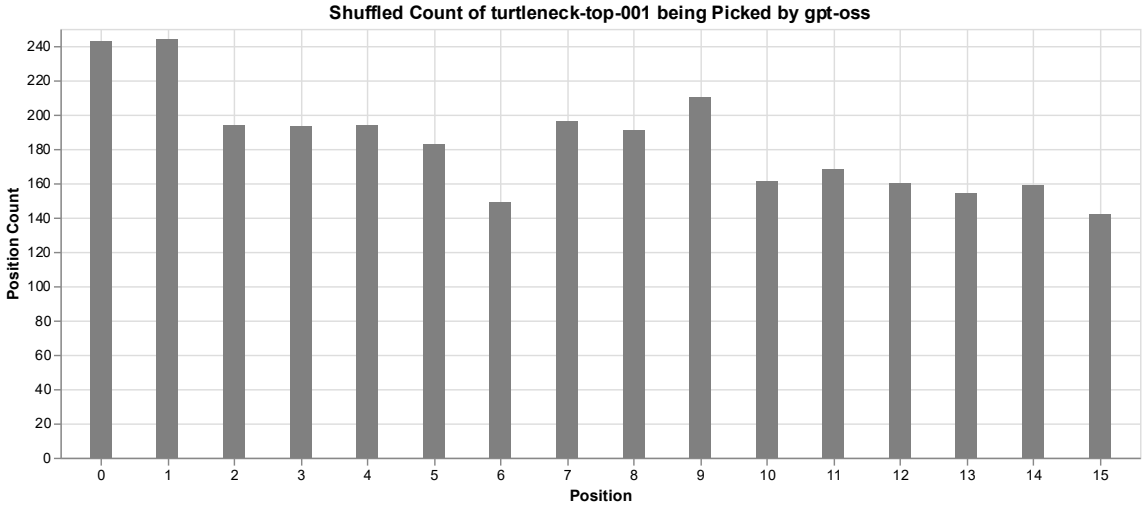


Figure 6: gpt-oss Turtleneck

The frequency and position at which gpt-oss picked a turtleneck, which is a good choice within the constraints of our prompt.

were supporting that Gemma 3 has a position bias towards the front of the list. Jean shorts are not an item that is suited for 30-degree weather, and the fact that when they were picked, they were in the front fourth of positions 64.6% of the time shows how the model Gemma 3 has a position bias towards the front of the list.

When using the gpt-oss as the AI Stylist, we ran the model 3000 times for the control case and 4000 times for the shuffled case because gpt-oss took over 12 hours to run 4000 times. For the control case, the model chose 9,070 items and 12,921 items in the shuffled case. The data required some additional cleaning in both sets to analyze, which required removing times when the mode errored in producing a JSON output and removing the key, outfit, when it appeared before the items the model picked. In both cases, the model altered some item ids or hallucinated some items. In the control case, it did this 29 times, and in the shuffled case, it altered item ids or hallucinated items 42 times. For both cases of data, we have removed the items with altered item ids or that were hallucinated from all graphs and totals. This makes the new item total for the shuffled case 12,879, and the control case's new item total 9,041.

In the control case as seen in figure 3, gpt-oss picked all the positions at least once. The most often picked positions were position 0, "turtleneck-top-001", position 2, “leather_jacket_001”, and position 1, “BRWN_FLR_PNT_001”. It picked position 0 2,952 times, position 2 2,833 times, and position 1 2,201 times. These 3 positions make up 88.3% of the positions picked in the control case. All these top-picked positions hold items that are the first items of their class that appear in the closet list. These top-picked positions already imply that there could be a position bias in the gpt-oss model towards the front, since the top-picked items are coming from the front of the list.

The gpt-oss shuffled case is very similar to the Gemma 3 shuffled case. As shown in figure 4, a noticeable difference is how sharp the initial decrease is. Position 0 is picked 1,828 times, and position 1 is picked 1,475 times, which is a sharper decrease than the difference between position 0 and position 1 in the Gemma 3 shuffled case. It's also important to note that position 6 is picked 856 times, and position 5 is picked 854 times in the gpt-oss shuffled case. Position 15 was still the least picked position, with it being picked 337 times. This is significantly more than it was picked in the Gemma 3 shuffled case. The gpt-oss still has a position bias towards the front of the list, with positions 0, 1, 2, and 3 making up 40.2% of the total items picked, 12,879.

While gpt-oss still has a position bias towards the front of the list, it is more consistent at picking an original item, which, in the control case, it preferred. As seen in figure 6, it picks "turtleneck-top-001" more consistently through the different positions than other items in the list. This model is better at picking a more situationally appropriate item even if they are later in the sequence. The gpt-oss is a newer model than Gemma 3, so its ability to pick an item more consistently could be due to that fact.

5. Conclusion and Future Work

Our work demonstrates that models Gemma 3 and gpt-oss have position bias towards candidates at the front of the list, this is consistent with the expected results from the literature, as the closet is injected into the beginning of the model's context. For Gemma 3 positions 0, 1, 2, and 3 make up 53.8% of the total number of items picked. The model also had an increased tendency to pick an item not suited for the expected weather, jean shorts, when they were in positions 0, 1, 2, or 3. The model gpt-oss still showed bias towards the front of the list, with positions 0, 1, 2, and 3 making up 40.2% of the total items picked. After testing two models for position bias, we are in the process of in testing more models or updating the test to see if different situations and closets change our results. We are also interested in coming up with and testing methods to reduce position bias within LLM recommender systems. One potential approach is to shuffle the closets, show it to the model multiple times, and then vote. Similar approaches have shown success in the work of Tang et al. [11].

References:

[1] E. Bito, Y. Ren, and E. He, "Evaluating Position Bias in Large Language Model Recommendations," Aug. 04, 2025, *arXiv*: arXiv:2508.02020. doi: 10.48550/arXiv.2508.02020.

[2] N. F. Liu *et al.*, "Lost in the Middle: How Language Models Use Long Contexts," *Trans. Assoc. Comput. Linguist.*, vol. 12, pp. 157–173, 2024, doi: 10.1162/tacl_a_00638.

[3] L. Wu *et al.*, "A survey on large language models for recommendation," *World Wide Web*, vol. 27, no. 5, p. 60, Aug. 2024, doi: 10.1007/s11280-024-01291-2.

[4] G. Team *et al.*, "Gemma 3 Technical Report," Mar. 25, 2025, *arXiv*: arXiv:2503.19786. doi: 10.48550/arXiv.2503.19786.

[5] OpenAI *et al.*, "gpt-oss-120b & gpt-oss-20b Model Card," Aug. 08, 2025, *arXiv*: arXiv:2508.10925. doi: 10.48550/arXiv.2508.10925.

[6] P. Wang *et al.*, "Large Language Models are not Fair Evaluators," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 9440–9450. doi: 10.18653/v1/2024.acl-long.511.

[7] "How Can Recommender Systems Benefit from Large Language Models: A Survey | ACM Transactions on Information Systems." Accessed: Mar. 11, 2026. [Online]. Available: https://dl.acm.org/doi/full/10.1145/3678004

[8] Z. Zhao *et al.*, "Recommender Systems in the Era of Large Language Models (LLMs)," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 11, pp. 6889–6907, Nov. 2024, doi: 10.1109/TKDE.2024.3392335.

[9] H. Liu *et al.*, "Sequential LLM Framework for Fashion Recommendation," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, F. Dernoncourt, D. Preoţiuc-Pietro, and A. Shimorina, Eds., Miami, Florida, US: Association for Computational Linguistics, Nov. 2024, pp. 1276–1285. doi: 10.18653/v1/2024.emnlp-industry.95.

[10] Y. Deldjoo *et al.*, "A Review of Modern Fashion Recommender Systems," *ACM Comput Surv*, vol. 56, no. 4, p. 87:1-87:37, Oct. 2023, doi: 10.1145/3624733.

[11] R. Tang, C. Zhang, X. Ma, J. Lin, and F. Ture, "Found in the Middle: Permutation Self-Consistency Improves Listwise Ranking in Large Language Models," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, K. Duh, H. Gomez, and S. Bethard, Eds., Mexico City, Mexico: Association for Computational Linguistics, Jun. 2024, pp. 2327–2340. doi: 10.18653/v1/2024.naacl-long.129.

EXPLORATION OF MOTION BLUR APPLIED TO RENDERING OF SIGNED DISTANCE FUNCTIONS

Evan Magill, Gary Zoppetti
Millersville University
{epmagill, gary.zoppetti}@millersville.edu

ABSTRACT
Signed Distance Functions (SDFs) are not as widely used as polygonal meshes, and as such fewer special effect techniques have been developed tailored to their use. In the context of SDFs, motion blur seems to frequently be implemented in general purpose ways that are largely agnostic of the underlying geometric representation. These techniques include post-processing screen space effects, and the compositing of renders from multiple time steps into a single image. This paper describes the early development of techniques in which SDFs are produced representing the volume occupied by an SDF over a given timespan, similar to existing techniques [1] for polygonal meshes. The approach described in this paper only enables the application of motion blur to unshaded, monochromatic circles and spheres and thereby has very limited present application. This research may hopefully give beneficial insight into future development of more general techniques.

KEY WORDS
Signed Distance Function, Raymarching, Real-time Rendering, Motion Blur

1. Introduction

As the field of rendering has advanced, numerous techniques have been developed to achieve effects that heighten realism and stylization.

Motion blur is one such effect that has been implemented through many distinct techniques. For high fidelity results, motion blur is generally achieved by rendering at many discrete timesteps or stochastically sampling at different times along a given range. While this approach can be used in real-time rendering, the high associated performance cost has encouraged the development of other techniques. Post-processing approaches can be adapted to a wide range of contexts but are prone to producing undesired artifacts. Some approaches produce new geometry to be rendered in the scene, such as by treating the positions of a polygon at two timesteps as the bases of a prism [1]. In addition to resolving some common sources of artifacts, having the effect integrated into earlier stages of the rendering pipeline may enable more creative manipulation of the render.

1.1 Signed Distance Functions

Signed Distance Functions (SDFs) are representations of objects that are discussed here as an alternative to polygonal meshes. An SDF is a function which yields the distance from an input point to the surface of the represented object. If the input point is outside of the represented object, the distance returned is positive. If the input point lies within the represented object, the distance returned is negative. See Figures 1 and 2 for an SDF portraying a circle.

In 2D, rendering SDFs is relatively simple. The SDFs that compose a scene can be evaluated at each pixel, and negative distance values indicate that a pixel lies within the corresponding SDF and can be colored accordingly. The 3D case is more complex and will be performed through raymarching for this paper.

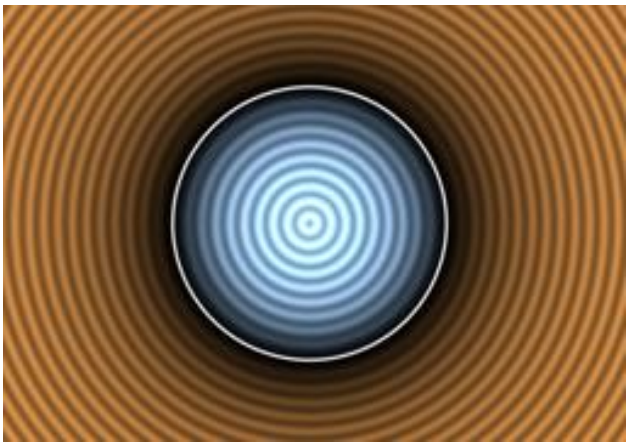


Figure 1: A 2D rendering of a circle from an SDF with visual aids indicating the surface or boundary in white, the interior with negative value in blue, and the exterior with positive value in orange [2].

```
float sdCircle( vec2 p, float r )
{
    return length(p) - r;
}
```

Figure 2: A corresponding SDF for the circle. The vector p corresponds to the point in space where the

SDF is being sampled. Here the parameter r allows for the radius of the circle to be specified. A parameter for the position of the circle's center is not present and in this case the circle is positioned at the origin [2].

1.2 Raymarching

For the purposes of this paper, SDFs in 3D will be rendered though a raymarching technique sometimes referred to as Sphere Tracing. In this approach, rays are emitted from the camera, and these rays step forward in an iterative process. At each step, the ray advances by the minimum distance evaluated from all the SDFs in the scene. By advancing in this fashion, we are ensured that the ray does not step inside or through an object.

As the ray approaches an object, the distance evaluated from that object's SDF shrinks and when this distance is below some minimum threshold, we consider the ray to have collided with the object.

For the limited scope of this paper, objects are rendered with uniform color with no lighting or shading. As such, for opaque objects, the color for a given pixel is set as the color associated with the SDF.

For rendering objects that are blurred and therefore not fully opaque, the ray marching process should be continued beyond the initial collision to determine the color underlying the object in the scene so that an accurate resulting color may be calculated.

1.3 Purpose

Significant research has been done in motion blur with polygonal meshes, but remarkably little research considers motion blur in the context of scenes composed of SDFs.

This paper will discuss rudimentary approaches to achieving motion blur through the construction of SDFs representing each blurred object.

2. Technique Development

The sphere is arguably the simplest SDF. This is simply the distance to the center of the sphere minus the radius. In addition, without surface detail, the sphere lacks a notion of orientation. We will initially develop techniques with these simplifications, then discuss the generalizability of the techniques.

If the object being moved is a sphere, reasoning about the region swept by its motion over some timespan is relatively simple. So long as the true SDF for the path traversed by the sphere's center is available, subtracting the radius results in the SDF of the region swept by the sphere. The region swept by a sphere moving linearly is thereby represented with a line segment for which a performant SDF exists as shown in Figures 3 and 4. A sphere experiencing constant acceleration moves along a parabolic arc. A section of a parabolic arc can be exactly represented by a quadratic Bézier curve. A quadratic Bézier curve is conventionally specified by its start and end points, as well as a control point which determines how the arc will curve in between. The position of the control point can be calculated at a low expense, and reasonable SDFs for Bézier curves are available making them a good choice for representing this motion. See figures 5 and 6 for the representations of these curves.

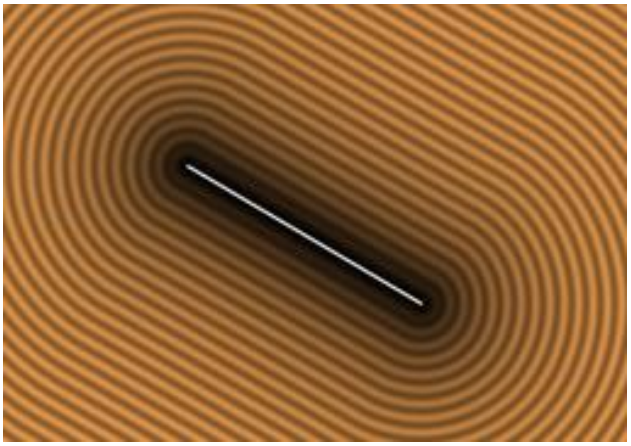


Figure 3: A 2D rendering of a line segment from an SDF [2].

```
float sdSegment( in vec2 p, in vec2 a, in vec2 b )
{
    vec2 pa = p-a, ba = b-a;
    float h = clamp( dot(pa,ba)/dot(ba,ba), 0.0, 1.0 );
    return length( pa - ba*h );
}
```

Figure 4: A corresponding SDF for the line segment with parameters for the endpoints [2].

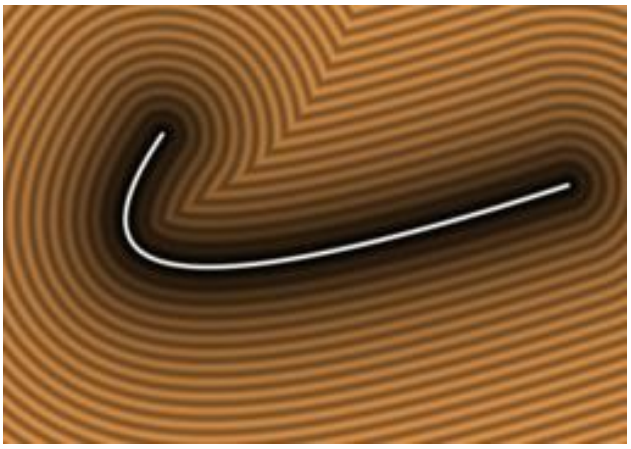


Figure 5: A 2D rendering of a quadratic Bézier curve from an SDF [2].

```
float sdBezier( in vec2 pos, in vec2 A, in vec2 B, in vec2 C )
{
    vec2 a = B - A;
    vec2 b = A - 2.0*B + C;
    vec2 c = a * 2.0;
    vec2 d = A - pos;
    float kk = 1.0/dot(b,b);
    float kx = kk * dot(a,b);
    float ky = kk * (2.0*dot(a,a)+dot(d,b)) / 3.0;
    float kz = kk * dot(d,a);
    float res = 0.0;
    float p = ky - kx*kx;
    float p3 = p*p*p;
    float q = kx*(2.0*kx*kx-3.0*ky) + kz;
    float h = q*q + 4.0*p3;
    if( h >= 0.0)
    {
        h = sqrt(h);
        vec2 x = (vec2(h,-h)-q)/2.0;
        vec2 uv = sign(x)*pow(abs(x), vec2(1.0/3.0));
        float t = clamp( uv.x+uv.y-kx, 0.0, 1.0 );
        res = dot2(d + (c + b*t)*t);
    }
    else
    {
        float z = sqrt(-p);
        float v = acos( q/(p*z*2.0) ) / 3.0;
        float m = cos(v);
        float n = sin(v)*1.732050808;
        vec3  t = clamp(vec3(m+m,-n-m,n-m)*z-kx,0.0,1.0);
        res = min( dot2(d+(c+b*t.x)*t.x),
                   dot2(d+(c+b*t.y)*t.y) );
        // the third root cannot be the closest
        // res = min(res,dot2(d+(c+b*t.z)*t.z));
    }
    return sqrt( res );
}
```

Figure 6: A corresponding SDF for the quadratic Bézier curve with parameters for end points A and B, as well as a control point C. The vector pos represents the same sample point labeled p in previous figures [2].

One key reason to generate new geometry for motion blur is that this often facilitates performant and accurate blurring of very fast motion. SDFs are well suited to taking advantage of this as primitives can be extended at little cost and the union of SDFs is easily produced as the minimum of those SDFs. Bézier curves and line segments can be made arbitrarily long while incurring very little additional performance cost. The trivial union allows for more complex paths (e.g. those involving physics interactions with other objects) to be reasonably represented.

2.1 Approximating Time Coverage

As a simplification, we consider the spheres to be unshaded. Thus, we only need to determine the opacity with which the blur should be rendered at each pixel. To determine the opacity of a given point we calculate the *time coverage*. We establish the term time coverage to refer to the duration that the given point lands within the object for the given time step. Though temporal information is not present within the produced SDF itself, the time coverage can in some cases be reasonably approximated using the SDF.

Consider the case of a circle moving with a constant velocity in a 2D space. For a given timespan the time coverage of any point that is uncovered at the beginning and end of the timespan is a function of that point's distance from the line segment between the center's starting and ending locations. In other words, in this case the time coverage can be exactly represented for most points in a scene as a function of the SDF evaluation and the velocity of the represented circle. If total accuracy is desired, an exact calculation with some additional information can be performed within the regions defined by the circle's start and end positions.

Though this approximation is not accurate for non-linear motion, it yields a reasonable looking result in most cases at a very low-performance cost.

In 3D, attempting to approximate time coverage from the SDF becomes more complicated. When raymarching, the collision with the SDF occurs near the surface (within some tolerance), where the SDF has a value of approximately 0. Instead of using this SDF value, we may base our approximation of time coverage on the thickness of the region that the ray passes through. That is, we continue marching until we reach the surface on the other side and determine the length of the vector from the entry point to the exit point. From there, marching continues so that anything behind the blurred object can be rendered as well.

2.2 Utility of Segmentation

As previously discussed, the easy union operation of SDFs enables the construction of complex paths for our objects. This means that motions that lack closed form equations can still be depicted. Physics calculations performed in discrete steps can be used to generate segments that can be joined together.

Beyond this useful path generality, segmentation can enable finer control of the appearance of our effects. Rather than combining the SDFs into a single object, we can determine which SDF segment to consider and render accordingly. This enables us to change the degree of blur for different segments according to parameters like velocity or recency. This can even allow for properties of the object to change over the time step such as size or color.

3. Implementation

We utilized the free and open-source Godot game engine [3] for our implementation of this technique in 2D. The project created in Godot consists of a 2D scene with a rect covering the rendered area. This rect has an associated .gdscript file and a .gdshader file. GDScript [4] is the preferred language to be utilized for game logic in Godot, with syntax similar to Python. The .gdshader file uses Godot's shading language which is highly similar to GLSL ES 3.0 [5].

The compiled .gdscript file is executed on the CPU before each frame and manages the circles we intend to render. For convenience, each circle is an instance of a class, with data members to represent the position, velocity, and acceleration vectors. Each frame, physics calculations are performed, updating these circles' positions and velocities with consideration of the time elapsed between frames. For this implementation, the blur is performed between the present position of the circle and an extrapolated position a constant timestep into the future, independent of frame rate. The choice of extrapolating into the future was made for simplicity though this could be easily adjusted to instead be a point in the past, extrapolated or otherwise. Considering a constant timestep for the blur makes individual frames more comparable to each other and allows for easier viewing of the effect, particularly when this timestep is very large. Once the start, end, and control points are calculated for each circle, this information is used to update the shader parameters (referred to as "uniforms" in GLSL and Godot's shading language) so that the Bézier curves may be rendered. The initial velocity of each circle is also provided as a shader parameter for use in our time coverage approximation.

The .gdshader file consists of a fragment shader which is executed at each pixel on the screen. As this implementation considers the 2D case, each SDF only needs to be evaluated once per pixel. The shader loops over the indices of our circles and evaluates each Bézier curve SDF from the corresponding entries in the arrays of start, end, and control points. Any negative SDF values are used along with the corresponding circle's velocity to approximate time coverage. This time coverage is then used to interpolate the color of the pixel between the value calculated up to this point (the background color, possibly combined with the colors of some previous objects) and the current circle's color. Circles with higher indices are therefore rendered on top of circles with lower indices. See Figures 7 and 8 for images of the running project.

Figure 7: A screenshot of the running Godot project. A high relative downward acceleration and long blur timestep are used to exhibit the curves achieved with the Bézier SDF.

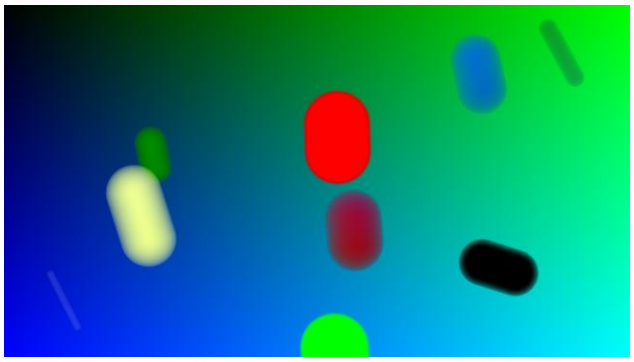


Figure 8: A screenshot of the running Godot project. A lower relative acceleration is used resulting in paths with little discernible curve. The same Bézier SDFs are being utilized.

4. Future Work

Efforts could be made to consider how this approach may be adapted to other primitives beyond n-dimensional spheres. A survey of implementations and approaches in informal online communities such as shadertoy.com could be conducted to consider how such techniques could be incorporated into or generalized beyond this approach. Some significant effort may be employed in analyzing code samples as readability of SDF code samples is often low. These shaders are often adapted directly from mathematical derivations with few comments and exceedingly short variable names. However, with experience some knowledge of community conventions can aid in this analysis.

5. Conclusion

While literature on motion blur of SDFs is very limited, some demos and code samples do perform motion blur in a fashion developed specifically for SDF based geometry, though these are often not easily generalized beyond the exact scene represented. Developing well-documented techniques and working towards generality will allow for greater flexibility and creativity in SDF rendering.

The technique described is very limited in only applying to spheres and circles without shading or lighting, however this is reasonably suited for rendering particle effects, particularly when these particles are fast-moving or under long exposure, where other techniques may produce artifacts, oversimplify motion, or be less performant.

5. Acknowledgements

The excellent work of Inigo Quilez served as a tremendous resource for knowledge and visuals relating to the rendering of SDFs.

References:

[1] M. J. L. Rønnow, U. Assarsson, & M. Fratarcangeli, Fast analytical motion blur with transparency, *Computers & Graphics, 95*, 2021, 36-46. https://doi.org/10.1016/j.cag.2021.01.006.

[2] I. Quilez, 2D distance functions. https://iquilezles.org/articles/distfunctions2d/.

[3] Godot Engine. https://godotengine.org/.

[4] GDScript reference. https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html.

[5] Godot Shading Language. https://docs.godotengine.org/en/stable/tutorials/shaders/shader_reference/shading_language.html.

Rebalancing the Ribbon: Correcting Dataset Imbalance in AI-Driven Breast Cancer Detection

Megan Seaman, Samuel Grieggs
Indiana University of Pennsylvania
zgxdc@iup.edu, sgrieggs@iup.edu

ABSTRACT
Breast cancer affects approximately 1 in 8 women over the course of her lifetime, yet early detection yields a 5-year survival rate of ~99%, making accurate AI-assisted mammography screening a critical area of research. A persistent challenge in this domain is severe class imbalance — in typical screening datasets, cancerous images represent as little as 1–2% of all samples, causing naively trained models to predict noncancerous outcomes by default and achieve 0% sensitivity on cancer cases despite high overall accuracy. This paper investigates two complementary techniques to address this imbalance: (1) a sigmoid-focal loss function with class-weighted parameters (α = 0.075, γ = 2.0) to emphasize learning from the underrepresented cancerous class, and (2) a dynamic sampling strategy that progressively shifts the training distribution from a 50-50 class split toward the true distribution across epochs.

KEY WORDS
Computer Vision, Deep Learning, Mammogram

1. Introduction

The American Cancer Society approximates that about 1 in 8 women will be diagnosed with breast cancer over the course of her lifetime, but if detected early, the 5-year relative survival rate stands around 99% [1]. Due to the effective treatment options when cancer is detected early, proactive, annual cancer screenings are recommended for all women starting at age 40. It has been demonstrated that artificial intelligence (AI) can act as a powerful tool in breast cancer detection [2] [3], promoting AI's integration into healthcare as a supplemental tool to relieve overworked radiologists and reduce the number of late-stage diagnoses. Although extensive research illustrates the use of AI to effectively classify mammograms with promising results, such studies are often hampered by having to rely on imbalanced datasets. Often, mammography datasets have an overwhelming proportion of noncancerous samples in comparison with cancerous samples due to the vast number of annual screenings. The lack of positive samples in datasets causes models to become biased towards noncancerous images, negatively affecting the model's overall performance. Therefore, this research aims to improve the performance of machine learning models training on biased datasets for biomedical imaging classification tasks.

If AI models are to be effective tools for early detection, they must be able to accurately identify both noncancerous and cancerous scans despite a lack of available positive samples for reference in training. Overworked radiologists swamped by a plethora of scans needing to be reviewed will only benefit from a reliable tool that outputs minimal false positive and completely avoids false negative diagnoses. Failing to detect cancer allows the tumor or mass of abnormal cells to grow and metastasize by the time of the patient's next screening, unnecessarily worsening her prognosis. Similarly, incorrectly identifying a scan as cancerous requires a patient to undergo further testing and screening, which increases the workload of doctors and radiologists and subjects the patient to unnecessary anxiety and invasive procedures. For that reason, this project desires to improve a model's performance in identifying cancerous images when trained on minimal positive samples by implementing dynamic sampling, weighted loss functions, and other techniques.

2. Related Works

Class imbalance in image classification, and all machine learning, is a well-known and well-explored problem. Buda et al. demonstrated that this problem extends from traditional machine learning into deep learning based image classification problems [4]. Once the problem is acknowledged, there are a couple of ways to address it. The first would be to build a balanced dataset which is easier said than done. Chawla et al. propose an alternate solution that does not require any additional data [5], namely under-sampling the majority class, and over-sampling the minority class. For that reason, we implemented a similar approach to SMOTE in this work. An alternative method would be to weight the loss function to emphasize

performance on the minority class which can be accomplished through Focal Loss [6]. Using Deep Learning for medical imaging, and for mammography in particular, is also widely adapted in the literature [3] [2].

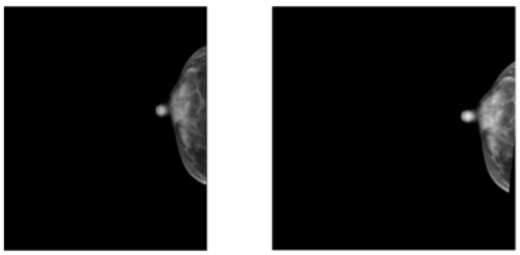


Figure 1:Example of augmented Mammogram Image

3. Methodology

This research was conducted using two datasets, namely the Radiological Society of North America's (RSNA) 2023 Breast Cancer Detection Contest dataset obtained from Kaggle [7] and The Chinese Mammography Database (CMMD) downloaded from the Cancer Imaging Archive [8]. The RSNA dataset has over 43,000 mammogram scans, and we withheld around 11,000 images for testing. The training and testing sets are highly imbalanced since noncancerous images account for roughly 98% of each dataset. Useful metadata such as the diagnosis (0 for noncancerous and 1 for cancerous), left or right breast, patient ID, etc., are listed in each set's CSV file. Contrary to the RSNA dataset, the CMMD has over 3,700 images with cancerous images accounting for roughly 70%. The CSV file has metadata such as patient ID, subtype, classification (malignant or benign), etc. Since the RSNA and CMMD datasets employ different ways of labeling cancerous and noncancerous images ("1" versus "malignant"), a separate class was written to standardize the labeling system according to the RSNA method. We combined the RSNA and CMMD datasets for training with only about 7.5% of the total images being cancerous. Lastly, the model was evaluated using only the RSNA testing dataset.

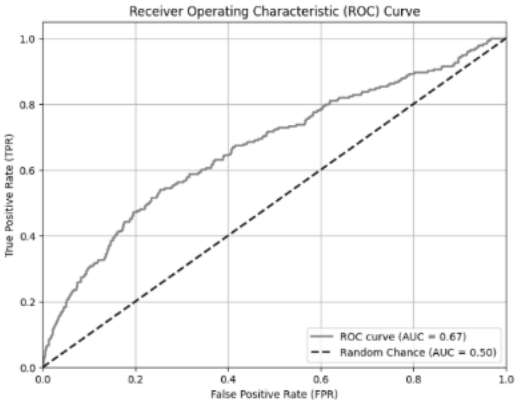


Figure 2: ROC Curve for Focal Loss

The code for this process was written in Visual Studio Code using the Python programming language and the PyTorch machine learning framework. Before training, images were preprocessed using several augmentations. For example, images were flipped horizontally ($p = 0.3$) and randomly rotated (degrees = 20). In addition, the images were slightly cropped using the random affine transform (translate = (0.05, 0)). Likewise, an elastic transform ($\alpha = 3.0$ and $\sigma = 0.5$) and gaussian noise were also applied (mean = 0.0 and $\sigma = 0.01$). To add, every image was resized to 512 by 512 and normalized according to the mean and standard deviation of the respective dataset. Figure 1 illustrates an example image before and after augmentations.

We used a ResNet-50 [9] architecture with weights pretrained on ImageNet to process the images. For reference, transfer learning from a ResNet pretrained on ImageNet is a widely accepted practice in the literature [10]. The architecture was modified to accept grayscale input by adjusting the first convolutional layer for single channel input and the fully connected layer for binary classification. To continue, stochastic gradient descent (SGD) (learning rate = 0.001 and momentum = 0.9) was selected as the optimizer, and the model processed images using a batch size of 32.

Employing a sigmoid-focal loss function [6] to address class imbalance was a main tactic of this research. Due to the overwhelming proportion of negative samples in the dataset, standard cross-entropy loss functions fail to positively impact the model's ability to detect cancer. For that reason, a sigmoid-focal loss function was implemented since the α parameter allows for weights to be applied to each class, placing greater emphasis on the cancerous class despite the low number of occurrences in the actual dataset. Furthermore, we set $\alpha = 0.075$ to match the proportion of cancerous

samples in the combined training dataset and kept the default parameter $\gamma = 2.0$.

Our dynamic sampling method focused on repeatedly updating the training dataset during training rather than using the imbalanced dataset for each epoch. Furthermore, the approach addresses imbalance by continually adjusting the proportion of cancerous and noncancerous samples within the training dataset by a predetermined amount after each epoch, subjecting the model to varying collections of images. The method began by creating a 50-50 distribution of the training dataset and adjusted the proportion of each class by 0.5% after each epoch. The model was trained employing only the focal loss function before using it with dynamic sampling. In terms of training, it lasted at least a hundred epochs. After reaching one hundred epochs, we continued until there had been no improvement in model loss within ten consecutive epochs, ensuring efficient training while preventing running the model with stagnant progress.

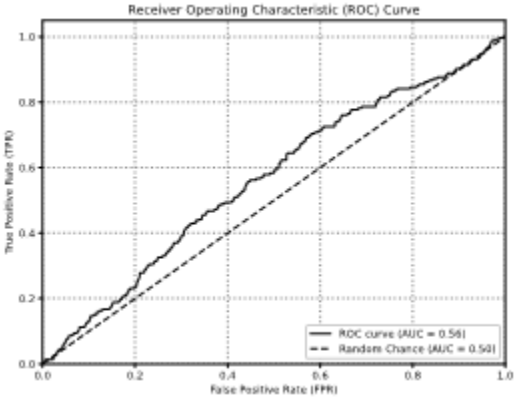


Figure 3: ROC Curve Dynamic Sampling with standard Binary Cross Entropy loss

The model's final prediction on the test images was decided using a 0.5 threshold. For probabilities greater than 0.5, the image was diagnosed as cancerous. Otherwise, the image was labeled as noncancerous. The model was evaluated after each epoch by calculating the overall accuracy and specific class accuracies on the test images, as well as the area under the curve (AUC) for that epoch's receiver operating characteristic curve (ROC) [11]. During the training and testing process, best AUC, best AUC epoch number, best loss, best loss epoch number, current loss, current AUC, and current epoch number were tracked and logged using the Weights & Biases platform which provides visualization tools for the performance and metrics of AI models

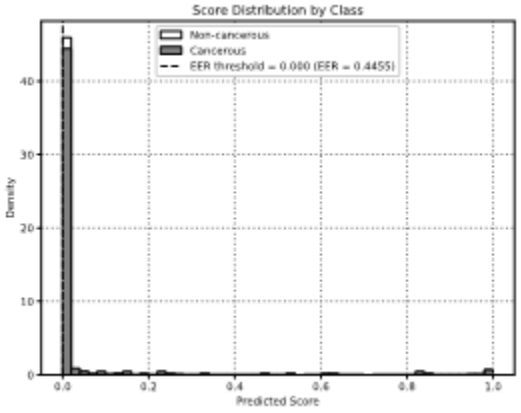


Figure 4: Score Distribution by Class for Dynamic Sampling with Binary Cross Entropy Loss

4. Results

Naively training the model with a binary cross-entropy loss function gives the best overall accuracy: 97.86%. Unfortunately, digging into this result further shows that while we get a 100% overall accuracy on noncancerous images, the accuracy on cancerous images is 0%. Despite the apparent strong performance in terms of accuracy, the model is simply learning that it should always predict noncancerous for the highest performance due to the wide class disparity in the dataset. Because of this we focused on other metrics, such as area under the ROC curve.

Implementing just the sigmoid-focal loss showed promising results. The model was trained for a total of 283 epochs, corresponding with when there had been no loss improvement after 10 epochs. Multiple metrics such as loss, AUC, and accuracy were evaluated. Despite fluctuations in the AUC, the overall score showed a visible increase over the 283 epochs, demonstrating a clear progression in learning the features of training images. The AUC scores from each ROC curve ranged from 0.48277 to 0.66986 with the lowest AUC at epoch 4 and the best at epoch 269. See Figure 2 for the ROC curve of our best model trained with focal loss. Additionally, the loss ranged from 13.13832 to 6.31197, demonstrating a steady decrease throughout the entire training session. The best loss of 6.31197 for this training session occurred at epoch 272. Figure 3 shows the ROC curve for our best experiment using

dynamic sampling with a standard Binary Cross Entropy loss. This experiment starts by under-sampling the noncancerous samples in the dataset, and periodically increasing the number of noncancerous samples in the dataset until we reach the true distribution. Our best result came after 98 epochs and only had an AUC of 0.56 as can be seen in Figure 3, but has an interesting score distribution, which could be indicative of a need to train the model for more epochs. The difference in score distributions between dynamic sampling with the Binary Cross Entropy loss and the model trained with the Focal Loss, which can be seen in Figures 4 and 5 respectively, is quite notable. Despite having worse overall performance, the BCE loss produces much more confident rejections, giving lower overall scores, but rejecting all noncancerous samples. More investigation is needed to better understand why this only seems to occur for the BCE loss. This may indicate that the model stopped too early, and we should attempt to train it for a longer period of time.

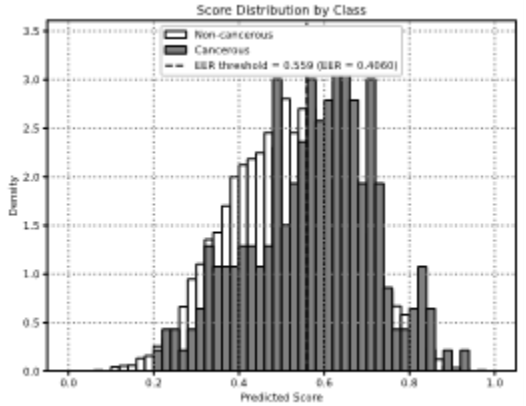


Figure 5: Score Distribution by Class with Focal Loss

5. Conclusion

In conclusion, this research focused on improving the performance of machine learning models trained on datasets that are biased towards the noncancerous class. To address class imbalance, this study implemented dynamic sampling, weighted loss functions, and other techniques with the goal of producing a model well-suited to identifying cancerous samples instead of biasedly predicting all samples as noncancerous each time. The use of a sigmoid-focal loss function showed considerable improvement in AUC and loss with AUC increasing from 0.48277 to 0.66986 and loss decreasing from 13.13832 to 6.31197. These metrics indicate that the model acted to optimize its internal parameters and extract features from cancerous images to improve its predictions. Although at present this model would not be an effective tool within a radiology lab, it demonstrates great potential for accurately diagnosing patients with the help of further research and training. Some limitations of this experiment were using only two datasets for training and a relatively small number of epochs. Therefore, future research must focus on additional training and integrating more datasets. With that, even though a groundbreaking tool was not developed from this research, it provided a fundamental educational opportunity to understand key techniques for addressing imbalanced datasets, as well as an easily replicable study for students to understand the basics of machine learning.

6. Future Work

There is still much work to be done on this problem. Rodriguez-Ruiz et al. established that the performance of human radiologists has an AUC of approximately .841 [12]. We hope to approach that number by better refining our solutions to the class imbalance problem. Some areas for future work are dynamically scaling the alpha value when dynamically sampling the dataset, generating high quality synthetic data, and other techniques.

References:

[1] A. N. Giaquinto *et al.*, “Breast cancer statistics 2024,” *CA. Cancer J. Clin.*, vol. 74, no. 6, pp. 477–495, 2024, doi: 10.3322/caac.21863.

[2] S. M. McKinney *et al.*, “International evaluation of an AI system for breast cancer screening,” *Nature*, vol. 577, no. 7788, pp. 89–94, Jan. 2020, doi: 10.1038/s41586-019-1799-6.

[3] G. Litjens *et al.*, “A survey on deep learning in medical image analysis,” *Med. Image Anal.*, vol. 42, pp. 60–88, Dec. 2017, doi: 10.1016/j.media.2017.07.005.

[4] M. Buda, A. Maki, and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural Netw.*, vol. 106, pp. 249–259, Oct. 2018, doi: 10.1016/j.neunet.2018.07.011.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic

Minority Over-sampling Technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.

[6] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," presented at the Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2980–2988. Accessed: Mar. 12, 2026. [Online]. Available: https://openaccess.thecvf.com/content_iccv_2017/html/Lin_Focal_Loss_for_ICCV_2017_paper.html

[7] H. M. Trivedi, M. Vazirabad, F. C. Kitamura, Y. Chen, and H. Frazer, "Open-Source Dataset for the RSNA Screening Mammography Cancer Detection Challenge," *Radiol. Artif. Intell.*, vol. 8, no. 2, p. e250375, Mar. 2026, doi: 10.1148/ryai.250375.

[8] H. Cai *et al.*, "An Online Mammography Database with Biopsy Confirmed Types," *Sci. Data*, vol. 10, no. 1, p. 123, Mar. 2023, doi: 10.1038/s41597-023-02025-1.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778. Accessed: Mar. 12, 2026. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[10] H. E. Kim, A. Cosa-Linan, N. Santhanam, M. Jannesari, M. E. Maros, and T. Ganslandt, "Transfer learning for medical image classification: a literature review," *BMC Med. Imaging*, vol. 22, no. 1, p. 69, Apr. 2022, doi: 10.1186/s12880-022-00793-7.

[11] "The meaning and use of the area under a receiver operating characteristic (ROC) curve.Radiology." Accessed: Mar. 12, 2026. [Online]. Available: https://pubs.rsna.org/doi/abs/10.1148/radiology.143.1.7063747

[12] A. Rodriguez-Ruiz *et al.*, "Stand-Alone Artificial Intelligence for Breast Cancer Detection in Mammography: Comparison With 101 Radiologists," *J. Natl. Cancer Inst.*, vol. 111, no. 9, pp. 916–922, Sep. 2019, doi: 10.1093/jnci/djy222.

RosettASM: Interactive Visualization of High-Level Code Execution and Machine State

Steven Counterman
Department of Math & Computer Science, DeSales University
sc4214@desales.edu
Evan Shimkanon
Department of Math & Computer Science, DeSales University
evan.shimkanon@desales.edu

ABSTRACT
In modern computer science education, students often write programs using high-level languages without being exposed to how those same programs execute at the machine level. These languages abstract away machine architecture to facilitate the development of program logic while hiding the interaction with registers, memory, and the program stack. As a result, machine behavior is often treated as a "black box," and students lack accurate mental models of machine state during runtime.

RosettASM is an educational programming environment developed by the authors that bridges the gap between high-level code and machine architecture. This system presents source code, the generated assembly, and register, memory, and stack states concurrently via step-through execution on a graphical interface. RosettASM contains its own custom, simplified, instructional programming language, the associated translation pipeline for target code generation, and a pedagogical interface to reinforce machine architecture concepts. This paper presents the design and prototype implementation of RosettASM.

KEY WORDS
Programming Education, Machine Architecture, Educational Tools

1. Introduction

Computer science education relies heavily on high-level programming languages to introduce students to programming concepts. A challenge with this approach is that these languages abstract away the underlying machine architecture, creating the appearance that they operate independently of the hardware [1]. Consequently, students lack exposure to machine behavior related to their source code and therefore lack appropriate mental models to accompany it.

Constructing accurate knowledge of machine-level execution is integral to understanding structure and control flow within high-level programs [2]. Concepts such as recursion, stack frames, and program flow become more explicit when viewed through the lens of machine-level execution. For example, recursion is often taught as repeatedly calling a function until a base case is reached. However, instruction frequently emphasizes control flow rather than the underlying runtime stack behavior that enables these calls [3].

To address this gap, we developed RosettASM, an interactive platform that allows students to work directly with their own source code. The name "RosettASM" is inspired by the Rosetta Stone, reflecting the system's goal of translating high-level programming constructs to assembly-level machine execution. The suffix "ASM" refers to assembly language. Its accessible didactic environment uncovers the black box by allowing the user to step through their assembly translation and observe runtime state changes in registers, memory, and the program stack. These components reside within the same interface, allowing explicit mapping across abstraction layers and reinforcing sequential reasoning and concrete mental models of execution. The remainder of this paper is organized as follows. Section 1.1 provides background context of work related to this topic. Section 2 describes the system design of RosettASM, including the instructional language, translation pipeline, and visualization interface. Section 3 discusses the advantages, limitations, practical applications, and future work associated with the system.

1.1 Previous Work

Prior research on visual program execution has explored the use of notional machines to help students understand program behavior during execution [4]. Python Tutor, for example, is a widely used web-based visualization system designed to help students step through code and observe program state [5]. While these tools make program execution more visible to beginners, they typically operate at the level of the source language and abstract away the underlying architecture.

Figure 1. Python Tutor visualization of variable state after expression evaluation.

Alternatively, debuggers allow developers to step through program execution, set breakpoints, and inspect program state during runtime. While immensely useful, debuggers such as GNU's GDB [6] are tailored to professional developers with prior knowledge of the underlying architecture and familiarity with development tools. As a result, they tend to have complex interfaces and large feature sets that are not paired with instructional guidance. While they are powerful, debuggers are primarily designed for professional software development rather than programming pedagogy.

RosettASM aims to consolidate program visualization environments and professional debuggers by providing an instructional sandbox environment. By connecting high-level code, generated assembly, and machine state visualization within a single interface, RosettASM provides a learner-friendly setting conducive to reasoning about program execution.

2. System Design

The RosettASM environment includes three main components that support code translation and visualization. The first is a custom instructional programming language that allows students to create their own source code. The second is a translation pipeline responsible for converting the source program into assembly code. The final component is an interactive visualization interface that represents the resulting assembly and machine state. The following subsections describe each component in greater detail.

2.1 Language Design

The primary motivation for a custom programming language is to reduce the cognitive load associated with programming. Many programming languages, such as Java or C-based systems, use similar syntax, making it familiar to students. To avoid unnecessary complexity, it does not utilize a full feature set or libraries. The simplicity allows students to focus on the machine execution represented in the interface rather than the code itself.

The language currently supports the following constructs:

- Variable declaration and assignment
- Arithmetic expressions
- Relational and equality comparisons
- Conditional control flow (if/elif/else)
- Iterative control flow (for, while)
- Flow control (break, continue)
- Commenting

Included is a small statically typed system that supports integer, float, Boolean, and character values to enable basic semantic checking while maintaining simplicity. The syntax is similar to many widely used high-level programming languages. It is semicolon-delimited and block structured, reminiscent of Java. Functionality is intentionally limited, as the language is designed for instructional purposes rather than production development. This allows assignments to focus on program behavior rather than full-scale software projects.

2.2 Translation Pipeline

When a student runs their program in the text editor, the source code is sent to the lexical analyzer. During lexical analysis, the source code is converted into token objects representing keywords, symbols, identifiers, literals, operators, and punctuation. Whitespace and line comments are ignored at this stage, while delimiters such as semicolons, parentheses, and braces are preserved as tokens for parsing.

Using these tokens, the parser evaluates the structure of the program according to the language's syntax rules. It recursively verifies that the code follows these rules and produces Node objects containing relevant metadata. These nodes are nested according to their syntactic relationships to construct an Abstract Syntax Tree (AST), which represents the program's structure and order of evaluation. The AST is then passed to semantic analysis for verification against the language's semantic rules.

During semantic analysis, a symbol table is maintained to track variable declarations and ensure variables are instantiated before use. This stage also verifies type correctness for assignments and expressions. In addition, scope rules are introduced for code blocks, including nested scopes within conditionals and loops. Once these checks are completed, the verified AST is passed to the Three Address Code (TAC) generator.

The TAC generator traverses the AST to produce an intermediate representation resembling assembly instructions. These statements follow the three-address format and use temporary variables to represent intermediate values while remaining independent of specific hardware registers. In contrast to typical compilers, optimization is intentionally avoided so that intermediate steps remain explicit, allowing students to

observe how higher-level expressions translate into lower-level operations.

The TAC is then passed to the target code generator. At this stage, variable lifetimes and next-use information are analyzed to determine register allocation and stack usage. Values that cannot be assigned to registers are temporarily stored on the program stack. The resulting assembly instructions are then displayed in the appropriate panel of the user interface.

2.3 Visualization Interface

The RosettASM environment presents the results of the translation pipeline through an interactive user interface designed to expose machine behavior alongside the source program.

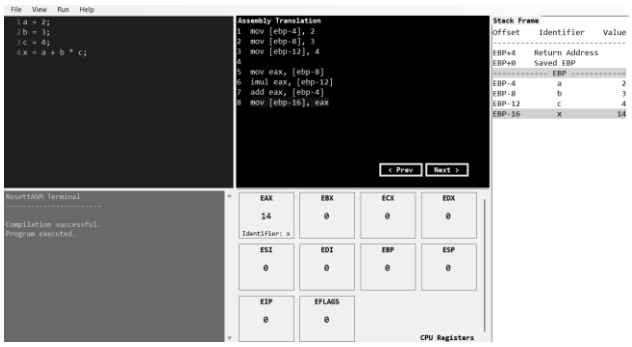


Figure 2. RosettASM interface displaying example program from Figure 1.

Students write code directly in an integrated editor and can select individual lines to observe how they are translated into assembly instructions. Selecting a line of assembly reveals the associated machine state, allowing students to observe how variables move between registers and memory as execution progresses. The interface displays multiple panels simultaneously to provide a complete synchronous view of program execution. The editor panel allows students to write and select source code, while a terminal panel reports compilation and runtime errors. A central panel displays the generated assembly instructions, and additional panels visualize register contents and the program stack.

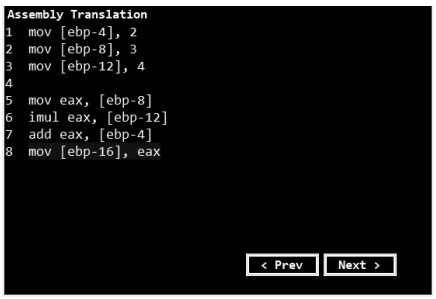


Figure 3. Assembly instruction translation panel with sample data.

The stack visualization shows variables relative to the base pointer, allowing students to observe how values are stored and updated during execution.

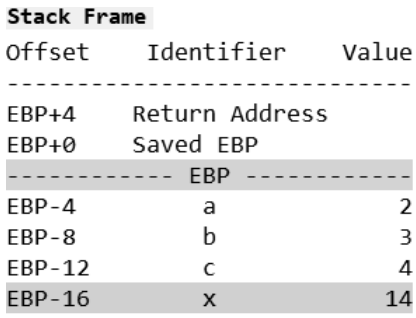


Figure 4. Stack visualization panel with sample data.

As students step through the program, the interface dynamically updates to reflect changes in machine state. Variables that move into registers are highlighted, while values written back to the stack are visually indicated.

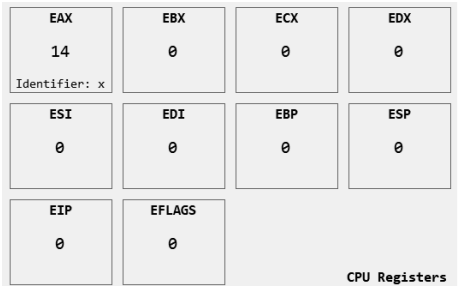


Figure 5. CPU register panel with sample data.

Register values are displayed in decimal form for readability, and stack entries show the variable name, stored value, and offset relative to the base pointer. These visual cues help students track how program execution modifies machine state through each step. This simultaneous presentation of source code, assembly, and machine state helps students build an intuitive understanding of how high-level programs execute at the hardware level.

3. Conclusion

RosettASM was developed as an educational environment meant to expose the relationship between high-level code and the machine architecture that executes it. By combining a simplified instructional language, a translation pipeline, and an interactive visualization interface, this system allows students to observe the transformation of their source code into machine state changes during runtime. This approach supports constructivist learning by making the normally hidden processes of program execution visible to learners. The following subsections discuss the advantages, limitations, potential educational applications, and future work associated with RosettASM.

3.1 Advantages

The primary advantage RosettASM offers is the bridging of abstraction layers. Concepts like stack frames, registers, data movement, and control flow are taught using architecture visualizations. The separation of computer science courses into those focused on high-level programming and those focused on computer architecture can create the false impression that these concepts are independent. This system consolidates these principles and presents computing constructs as a unified pipeline. Typical development environments do not present source code and execution state simultaneously for evaluating machine behavior. While these environments hide the interaction, RosettASM focuses on constructing the mental models associated with removing abstraction.

Another advantage is that seeing machine state reinforces constructivist learning concepts. This allows foundational machine components to be learned first and high-level constructs to be built with concrete understanding of these components. Stepping through execution allows students to directly see their own source code in action instead of interacting with assembly in isolation. This helps capture explicit machine state the moment each instruction runs. As a result, learners can observe what operations occur for each line of source code.

RosettASM provides an avenue to lower the barrier to machine-level concepts. Allowing students to write high-level syntax instead of assembly reduces the prerequisite knowledge required to begin exploring program execution. Assembly languages express computation in terms of registers and instructions rather than variables and human-adjacent linguistic systems. Without requiring immediate familiarity with assembly language, students can observe how common programming constructs translate into machine-level behavior. This simplification promotes independent exploration, as students can utilize concepts they already understand from prior programming experience.

Explicit visualization of execution is paramount to constructing accurate knowledge. To accomplish this, non-optimized translation from intermediate representation to assembly keeps intermediate steps explicit. Typically, these steps are merged for compilers for efficiency, but RosettASM preserves them so students can observe atomic operations. In showing register and stack information simultaneously, data movement and stack storage is explicit for student observation. This helps learners understand how variables are stored in memory, their usage, and stack changes during execution.

3.2 Limitations

The custom programming language has explicit limitations in functionality. It does not currently support advanced language constructs typically provided by high-level counterparts used in development. For example, user-defined functions are not supported, so recursion cannot yet be exemplified. Object-oriented design and data structures are also not supported at this time. At minimum, user-defined functions would be implemented before broader classroom deployment. The current version represents the initial prototype and serves as a framework for future expansion.

Secondly, the system supports a single architecture. The current implementation focuses solely on x86 32-bit assembly. This restriction contrasts the common usage of x86-64 in modern systems. Additional modules for x86-64, ARM, and RISC-V have been considered for further capability extension.

The visualizations provided represent a simplified machine model. Certain machine behaviors such as cache interactions and heap memory are not currently represented. Additionally, nuanced aspects of execution are simplified, such as character variables utilizing four-byte allocations to maintain consistent stack alignment. The system is designed as an instructional tool rather than a production environment.

Ultimately, RosettASM is intended to ease students into low-level concepts, rather than replacing deeper study of these topics. Advanced students may eventually outgrow the environment as they move toward full systems programming. There exists no replacement for learning real assembly programming when deeper comprehension of architecture is required. The system is intended to be an introductory bridge to machine-level concepts rather than a replacement.

3.3 Practical Applications

The RosettASM environment may support instruction in several areas of computer science education. In computer architecture or organization courses, students are often introduced to registers, memory, and other low-level components for the first time. Novice programmers frequently struggle to understand the hidden runtime behavior of programs and the machine state that underlies program execution [7]. RosettASM provides a way to expose students to assembly instructions and machine state without requiring them to write assembly code directly. This reduces the friction associated with learning low-level concepts, while still allowing students to observe how programs interact with underlying hardware. Initial classroom deployment of RosettASM is planned at a local university, with a potential trial run at a local high school, to evaluate its effectiveness in introducing machine architecture components to both university and secondary-level students.

This system may also benefit introductory programming courses once students are familiar with fundamental

concepts such as variables, conditionals, and loops. Since the custom programming language uses familiar high-level syntax, students can write programs without extensive prior experience while still observing the assembly translation process and machine-level execution. This lower barrier to entry encourages independent exploration of machine-level behavior without requiring students to first learn the complexities of assembly programming. Early exposure to these relationships can help students construct mental models of program execution before encountering more advanced programming language concepts. RosettASM also allows students to explore concepts that are often difficult to internalize, including stack frames, data movement, and control flow. By presenting the source code, generated assembly, and machine state together, students can observe how program execution affects registers and memory at each step.

3.4 Future Work

RosettASM's compiler was developed to promote modular expansion. The next planned addition is support for user-defined functions. The addition of functions would enable the visualization of function calls and recursion. This would allow students to observe parameters pushed to the stack and the resulting changes to program state. Recursion functionality would allow students to observe how stack frames accumulate and unwind during program execution.

The next logical expansion for architecture would be development of an x86-64 module to align with modern systems. Additional architectures such as ARM or RISC-V may also be considered to broaden the system's applicability. Utilization of different target assembly code reinforces understanding of foundational assembly constructs.

Future visualization development includes representing heap memory in addition to the program stack. Additional alternative visualizations could include explicit function stack frames and recursive execution behavior. These additions would allow students to observe a wider range of runtime behaviors beyond simple stack-based variable storage.

Lastly, RosettASM may be expanded for use in additional computer science courses. For example, extensions could support instruction in operating systems by visualizing concepts such as scheduling or synchronization. The system could also expose portions of the compiler pipeline itself to help students understand how source code is transformed into lower-level representations. Continued development of RosettASM aims to further strengthen the connection between student and machine.

References:

[1] R. Bryant and D. O'Hallaron, Computer systems: a programmer's perspective 2nd edition (Boston, MA: Pearson, 2011).

[2] M. Ben-Ari, Constructivism in computer science education, *Proc. 29th SIGCSE Technical Symposium on Computer Science Education*, Atlanta, GA, 1998, 257-261.

[3] C. Rinderknecht, A survey on the teaching and learning of recursive programming, *Informatics in Education*, *13*(1), 2014, 87-119.

[4] J. Sorva, *Visual program simulation in introductory programming education* (Aalto University, FI, 2012).

[5] P.J. Guo, Online Python tutor: embeddable web-based program visualization for cs education, *Proc. ACM Technical Symposium on Computer Science Education*, New York, NY, 2013, 579-584.

[6] Free Software Foundation, GDB: the GNU project debugger, www.gnu.org/software/gdb, December 12th 2025.

[7] J. Sorva, V. Karavirta, & L. Malmi, A review of generic program visualization systems for introductory programming education, *ACM Transactions on Computing Education*, *13*(4), 2013, 1-64.

RHYTHMICAI

Connor Skiles
Shippensburg University
cs8559@ship.edu
Skiles8703@gmail.com

ABSTRACT

This paper presents RhythmicAI; a Java based AI system that is designed to independently generate song lyrics and musical notation based on user specified parameters which include genre, mood, tempo, musical key, song duration, and prompts. The system uses an N-gram statistical model that is trained on a music dataset from Kaggle. To evaluate the system's performance, comprehensive testing was conducted across component functionality, integration scenarios, edge cases, and performance metrics, along with a user study involving 30 participants who generated tracks and provided feedback through a structured google survey. RhythmicAI utilizes a web interface that allows users to generate their own unique music tracks, view any previously generated content, and convert generated lyrics into audio files using Suno AI's free API. User feedback validated the interface's usability while identifying areas of improvement in lyrical depth and musical complexity. Thorough testing validates the system's functionality across all components, integrations, and performances. Though the project demonstrates the possibilities of AI-generated musical composition, further research reveals its limitations, including a lack of deeper theme, and musical notations that remain relatively simplistic compared to human compositions.

KEY WORDS

Artificial Intelligence, Music Generation, N-Gram Model, Tokenization, Natural Language Processing (NLP)

Introduction

The RhythmicAI project was sparked from an interest in furthering my brief knowledge of the capabilities of artificial intelligence. The idea of this project is centered around developing a system that could autonomously generate song lyrics, musical notation, and an audible version of those lyrics or notation. These lyrics or musical notations are based off a multitude of parameters such as genre, mood, prompts, song duration, tempo, and musical key. This goal was a hefty task as it required bridging natural language processing techniques with musical composition principles, which presented both technical challenges and creative opportunities. This program offers four main options: generate user-input-based music, generate random samples, create audio, and list tracks which are explained in further detail in the following paragraphs.

Related Work

Prior research in AI assisted music generation has taken numerous different approaches from rule based systems to deep learning models. Recent advances in AI generated music have shifted to neural network architectures such as OpenAI's MuseNet which uses a transformer based model trained on data that generates muti instrumental musical pieces across a wide variety of genres [1]. Similarly, Google's Magenta uses neural networks for both melody generation and style change in music [2].

Researchers have used the N-gram models, neural networks, and large language models to generated musical tracks. RhythmicAI distinguishes itself from the other approaches by using a computational inexpensive approach through an N-gram statistical model while still ensuring to integrate a rhyme scheme and a template based generation to produce structured tracks for wide varieties of user inputs.

System Architecture / Implementation

Core Components

RhythmicAI employs a modular, Java-based architecture consisting of several interconnected components that handle data processing, model training, content generation, and user interaction. The system's design allows for independent development and testing of each module while also maintaining a cohesive integration through a developed interface.

The central component to the RhythmicAI is the MusicTransformer class, which implements an

N-gram statistical model rather than employing the architecture from neural networks. Since the N-gram model employs simple counting and probability lookups instead of intricate mathematical operations like matrix multiplications it is computationally simpler than neural networks. In contrast to neural networks, which must learn millions of weighted parameters through iterative optimization, it only stores frequency counts of token sequences, requiring substantially less memory and training time. This eliminates the need for specialized GPU resources or lengthy training times, making the N-gram approach easier to implement and execute on common hardware.

The system utilizes an N-gram approach where N equals 5, meaning the model predicts the next token based on the previous five tokens in a sequence [3]. This choice balances efficiency within the computer with a sufficient context for generating reasonable short-term patterns. The model keeps track of a vocabulary mapping that transforms words into integer identifiers for effective processing and storage, enabling the system to compute with numbers rather than strings. Furthermore, it creates probability distributions that direct the model's predictions using weighted random sampling based on observed transition frequencies by storing frequency distributions that document how frequently each token follows a specific 5 token context in the training data.

Data preprocessing is handled by the MusicPreprocessor class, which reads CSV files containing track information from Kaggle datasets [4]. The preprocessor performs several critical functions including identifying relevant track information within columns of various CSV formats, cleaning text data by removing unnecessary whitespace and special characters, segmenting any long texts into appropriate training sequence lengths, and tracking metadata pertaining to the genre and mood distribution within the dataset. The preprocessing pipeline creates structured training sequences that flow directly into the model training phase.

Tokenization is managed through two specific classes. The LyricsTokenizer handles natural text, implementing word-based tokenization with special handling for contractions, and punctuation. It sustains a vocabulary that is solely built from the training data, filtering out words that appear a very few times to reduce vocabulary size while also providing a proper coverage of the dataset vocabulary. The NotationTokenizer is built to properly handle musical notation symbols including note names, octaves, and duration markers. Both tokenization classes implement special tokens used for padding, unknown words, the sequence start, and the sequence end, which allows for proper handling of differing lengths between sequences.

Training Pipeline

The training process begins with construction of the system vocabulary across all training texts. The system builds a vocabulary collection from both lyrics and notation, establishing a consistent token to integer connection that is used throughout the generation process. Once the system vocabulary is established, the preprocessor segments training texts into sequences, ensuring each sequence falls within minimum and maximum sequence constraints of 10 and 512 tokens.

To train its N-gram model, it analyzes all training instances using a sliding window approach to extract overlapping 6 token segments. At each position in the sequence, it uses the previous 5 tokens as context and the current token as the target. For every unique 5 token context encountered, it maintains frequency counts of all observed target tokens that followed that context. This creates a probability distribution mapping each context to its possible next tokens. During training, it also tracks overall token frequencies to identify the most common tokens, which serve as fallback options when generating text with unseen contexts.

To enhance quality of lyrics, the system incorporates a rhyme dictionary which chooses lyric endings that phonetically match earlier lines. This lets the system maintain consistent rhyme schemes while still sampling new word sequences from the N-gram probabilities during generation. This creates a produce that has a more natural sounding verse structure with consistent rhyme schemes instead of producing inconsistent rhyming patterns which would occur with just the use of the N-gram structure.

Generation Process

There are various steps involved in music generation based on whether there is training of

the model and what kind of content it needs to generate. For the generation of lyrics through a trained model, it starts with tokenization for forming the initial context from the input prompt from the user. It then starts to predict the next tokens based on an N-gram model and chooses these tokens with weighted random sampling based on its frequency. This method includes some variation to the generated content with the same input.

In situations where the context being processed by the model has not been seen during training, the mechanism resorts to smaller context window sizes, gradually lowering the context size from a 5-word window to a smaller 2-word context. In situations where there isn't an exact match at all context sizes, the system chooses between the most frequent words in the vocabulary.

In cases where the model is untrained or the training data is not enough, RhythmicAI resorts to template-based generation. This means that pre-defined song structures such as verses, choruses, and bridges that get populated with thematically relevant words come from the user's prompt. The template systems also use the rhythm dictionary for the purpose of maintaining rhythm schemes across sections to create structured lyrics that, although less varied than model generated content may still offer better readability, and understanding.

The generation of musical notation follows a set of similar style but in a much more limited range. It produces a sequence of note duration combinations using the melody contour pattern, while occasionally changing it up. It makes sure to account for music octaves that may become too high or low.

Database and Persistence

RhythmicAI uses JPA and Hibernate options for database management, using the H2 embedded database system to store the generated songs. The MusicDB entity class makes up the database structure, making sure to store key information such as the filename, genre, mood, song duration, tempo, musical key, generation time, and generation time in milliseconds. The database also includes the complete generated text, formula, and optional audio URL values when generated songs are converted into audio format by the integrating Suno AI.

Transactions enable management of the database by the EntityManager, providing a safe system for continuous usage of RhythmicAI, even while functioning without persistence capabilities. This allows users to create and display content despite any database failures that may arise.

Web Interface and Server

Interaction with the user happens through a dynamic website interface developed using HTML, CSS, and JavaScript. The website interface incorporates a layout with separated regions of selection for various functionalities such as the generation of music based on custom parameters entered by the user, the generation of random samples, the generation of audio files, and a list of all previously generated tracks.

The MusicGeneratorWebServer class provides an implementation of the HTTP server based on Java's HTTP Server API. The server provides small section that displays whether the model has been trained, the total number of system vocabulary tokens, whether the model has been preprocessed, whether the system server is running, where the output is going, and the name of the dataset that is being used for the model to be trained on. And a section that displays all possible options for genre, and moods for the music generation. Each of the above services uses proper error handling techniques based on the corresponding HTTP statuses, as well as the return of JSON formatted results to be processed in JavaScript.

Communication between the frontend and backend involves a request-response procedure. When users make generation requests, the frontend sends JSON data with all the parameters asked for in the request and receives notifications in the form of loading indications until the responses are received. The server executes all the requests in a simultaneous manner and produces tracks on demand.

Suno AI Integration

Audio generation uses the Suno AI API based on the SunoAIClient and SunoAudioService classes. When users submit a request for audio conversion of a track they have generated, the API request is created based on the lyrics of the track, its genre, mood, and other parameters. It supports both lyrical and notational modes with

proper care while handling JSON escapes and boolean flags to get the API request in the correct form [5].

The audio generation flows in an asynchronous process where the API returns a task handle, after that the application polls this task handle in intervals until the whole audio file is generated. The SunoAudioService contains proper logic to poll the task handle. Once the audio is successfully generated, it fetches the URL of the audio from the task handle response, updates the records in the database with the URL, and notifies the user interface to display the audio file.

The correct processing of the formatting for the lyrics has an important role in the process for effective use of the Suno AI that is integrated. The cleanLyricsForSuno function removes markers such as verse and chorus but keeps the generated lyrics together. This allows the API to get the formatted text it needs; while ensuring it still receives the same lyrics that were generated.

Testing and Validation

Testing Strategy

Comprehensive testing was done using the ApplicationTests class. Several aspects of the functionality of the system are covered. The testing techniques used in the program include unit testing of various parts of the program, integration testing of various functionalities within the program, testing of edge cases, as week as performance testing.

Class component tests assert the basic functionality of individual classes. Tokenization tests demonstrate the ability to create nonempty tokenization's for both the LyricsTokenizer and NotationTokenizer, including special tokens. Validator tests verify the functionality of the DataValidator by ensuring it correctly identifies a song as valid or not, as well as its functionality to clean the text by removing whitespace. Data class functionality is verified by ensuring the right values are calculated.

Integration tests are used to test the flows involving more than one component at a time. The preprocessor integration test will make sure that the MusicPreprocessor can read CSV files and correct data for the ProcessingResults. The model integration tests will verify that MusicTransformer can create lyrics and music successfully and return nonempty results for input strings. This ensures that each component functionals well together, even in scenarios where each component functions properly individually.

Test Results and Analysis

It was also tested that the system operates correctly on standard scenarios while being robust on edge case scenarios. The component tests results in successful testing for tokenization, validation, and data structure manipulation tests. The tokenizer operated properly on blank inputs, long texts, and special symbols. The validator worked correctly on null inputs as well as on songs with less content than others.

Integration testing showed that the preprocessing pipeline takes in CSV data from multiple file formats well, correctly identifying lyric columns despite various header names. Over 1,000 songs in test datasets processed, building vocabularies of appropriate sizes and generating training sequences within expected lengths. Model integration tests confirm generation capabilities, output quality depending on the quantity and variability of the training data.

Edge cases also revealed some important functionalities. The program handles invalid genre and mood pairs correctly by rejecting them and showing corresponding error messages. Null inputs in validation results in a series of tokens consisting of only special tokens. The absence of a null validation input also prevents an exception being raised.

Performance testing results in a time of less than 1 millisecond for every tokenize operation on average, while the entire song generation process takes roughly 200 to 500 milliseconds based on the song length.

Limitations Identified Through Testing

Testing brought out the fact that the N-gram technique has some limitations. The generated songs tended to have a very basic theme. The individual lines of each song may be grammatically correct with a consistent them but lacks a deeper thematic result. This shows that

the N-gram approach imposes its own constraints.

The process of music notation results in rhythmically correct sequences, though they lack complexity in terms of harmony. The music algorithm merges melodies together, even though music theories such as changes in music key have not been applied. The music synthesizes sequences that resemble random melodies, rather than a composed musical expression.

Although the template generation produces more organized text than the untrained statistical model, it depends on certain patterns that may repeat on several generations. The rhyme dictionary, although it is helpful in rhyming schemes, it doesn't provide completely perfect rhyming.

Critical Evaluation

Strength of this Approach

RhythmicAI provides an example of how natural language processing methods can be used to generate creative materials with manageable computation complexity. The N-gram model is more efficient when dealing with computation complexity compared to other methods such as neural networks. It requires less computation time, and memory.

The combination various strategies revolving around generation such as trained models, template fallback generation, and rhyme-based compositions allow for a more well-rounded application that can generate outputs based on a variety of conditions. It provides outputs that have a rhyme-based structure for lyrics to the users even in circumstances where there is no training data.

The web interface works well to provide users of all backgrounds regardless of their knowledge of programming an easy to navigate application for an AI music generator. The immediate outputs and interactive user inputs allow the user to understand how different inputs affect the generated outputs. Integration with the Suno AI expands on the applications ability beyond text generation to producing audio, which allows for a complete process of creativity for song generation.

Limitations, Flaws, and Challenges

The first limitation that comes with the N-gram approach is that it cannot maintain long dependencies or an overall continued theme with large sequences. The model tends to thrive in capturing short, local language patterns, it often lacks an understanding for a broader structure such as organization, and emotional development. The generated text can jump between various subjects and repeat patterns.

Musical notation generation remains very simplistic in comparison to human compositions. The system doesn't properly incorporate analysis of harmonies, changes in chords, connections between keys, or any rhythmic complexity. Generated melodies, while they do hold to basic music principles, they lack the any memorable musical phrases that pop out. The notation tokenizers vocabulary covers basic notation only; it does not extend further for variety in dynamics, tones, or emotions to help contribute to the overall creativity of the generated music.

The quality and representation of the training data play a vital role in the generative quality. Since the system depends on the use of CSV format from the Kaggle repository, the quality of the data is not consistent. The datasets contain invalid data, missing information, blank spaces, or songs that are sung in languages that are not English. The system does try to filter out these inconsistencies from the data, but quality is quality regardless of what occurs during preprocessing. The system may not generate the expected variations of genres that are less common within the training data.

While functional, the use of the Suno AI functionality comes with various dependencies on another service that comes with a cost and availability implications. The advantage is that the Suno AI offers a free version that allows users with google accounts to create a limited number of audio files using the 50 tokens provided to new users; however, the downside is that these 50 tokens are quickly used up. There are also implications with possible API downtimes. The use of asynchronous polling that comes with the process of turning generated tracks into audio with the Suno AI application results in slow response times, causing waits of several minutes to generate audio. Other techniques such as local audio generation may

avoid these dependencies but contain a much more complex implementation for audio generation.

Comparison to Alternative Approaches

More advanced models, such as GPT, which are transformer models, perform much better with text writing tasks, such as music composition, theme, and song creativity. Models such as these pay close attention to entire sequences of data, which means there is an ability to model dependencies between long range elements, something the N-gram models cannot do, but these models require much more computing power.

More specialized music generation models, such as OpenAI's MuseNet or Google's Magenta, are much more direct with music composition and use various music constraints to generate actual audio. These music generation models are much more complex.

User Feedback and Criticism

To evaluate RhythmicAI from an end-user perspective, a google survey was conducted with 30 participants who interacted with the web interface and generated music tracks. The system's design and output quality have both strengths and room for improvement, according to the feedback. Users consistently praised the interface's simplicity and ease of use, with multiple respondents noting it was "clean, simple, and easy to understand" and "simple, fun, and easy to use." The "minimalistic" look and "appealing colors" of the visual design garnered lots of praise. However, several issues with the quality of the generated content arose. Lyrics frequently lacked depth and coherence, according to users one respondent commented, "it seems like one of those newer applications, and the color is annoying, but the music beats you pick are good." Although some users recommended enhancements like the ability to choose instruments and adding more variety to moods, the Suno AI audio generation feature was well received. Participants expressed interest in enhanced customization options, including the ability to generate music in different languages, create audio directly from the notation, and have more control over the structural elements of the composition. This feedback validates the system's usability while highlighting the need for deeper musical and lyrical sophistication, which aligns with the technical limitations identified through internal testing.

Future Enhancements

Model Architecture Improvements

The introduction of more advanced models, such as neural networks or transformers, would provide a substantial improvement. These more advanced models would have been able to focus on larger contexts or would have allowed the focus to be on relationships between song components that are far apart.

Examples of fine tuning the model with genre specific datasets could result in better styling for certain genres of music. Instead of fine tuning the model to learn every genre of music together in one model, having different models for each genre of music could ensure that each model learns the expressions and themes of each genre type.

Music Theory

Incorporating music theory constraints into the generation of notation would improve musical quality substantially.

Rhythm and handling of time signatures currently are very simplistic. Future versions could use quantization to properly align notes to the corresponding positions and generate patterns for specific genres.
Integrating music libraries could allow for more sophisticated notation management, analysis, and export to standardized formats. This allows generated notation to be imported to professional notation software for further editing, management, and critiquing to increase the systems practicality for musicians and composers to use.

User Interface Enhancements

The graphical user interface would benefit from the addition of indicators that show the current stage of the generation process such as training, or preprocessing. Graphical representations of the processing of the generation, such as representations of the tokens or music notations would provide further clarity and engagement to the overall AI music generation process. The

addition of an editing feature that allows users to modify generated lyrics or musical notation before finalizing a track would improve the human and AI collaboration in the composition process.

The incorporation of user accounts with the ability to save generated tracks to designated accounts would allow users to look at and compare their previously generated tracks. Sharing and being able to rate other accounts generated tracks could also provide community engagement with AI-assisted music generation.

Dataset Expansion and Management

The training datasets variability and quantity for genres and moods could be improved dramatically with a larger set of tracks. Data could be managed to focus on quality over quantity. The data could include lyrics with manual intervention to ensure they contain the correct metadata information. Adding other data sources, such as Spotify or Apple music, would provide a more diverse and expanded collection of licensed lyrical data.

The inclusion of a multilingual feature would make the system useful for non-English musical content as well. The model would have to be trained in each language such as Spanish, French, German, Japanese, or any other desired language. It would greatly improve the systems usability, as it currently is only properly trained on English. Bilingual language incorporation would allow users to convert any generated tracks into other languages. This would be a useful tool that allows for greater creativity while also providing users with an educational resource to help expand their language skills.

The release dates of the songs within the dataset would allow users to generate lyrics from specific eras, like the 1960's, 1980's, early 2000's, and so one, depending on their desired time. This would improve the applications applicability to the overall style of the generated music.

Conclusions

RhythmicAI shows the possibility to apply natural language processing (NLP) processing to the task of creating music in a functional system. Even though the N-gram approach is not as advanced as the current deep learning models, this approach is very practical when considering the need to generate various outputs.

The system successfully integrates complex system components such as data processing, modeling, content creation, web interface, and API integration all into one application. The system has been extensively tested for functionality amongst several aspects, such as components, integration, as it works properly for actual applications as well as various special conditions.

In-depth analysis points out numerous limitations in the N-gram approach regarding coherence, and analysis complexity. The generated lyrics from the model have a local coherence and consistent theme that lacks an overall message, and the musical conversion is very simplistic in comparison to human compositions.

Despite its limitations, RhythmicAI provides a functional demonstration of how natural language processing techniques can be adapted to creative content generation. This program proves that the capabilities of the NLP process can be reused in new and creative ways. It offers insight into both the potential and challenges that come with AI-assisted musical composition.

RhythmicAI makes it clear that significantly creative output can result from simplistic statistical models, while also highlighting the large gap there is between statistical pattern matching and creative comprehension.

Acknowledgements

This work was supported by the ENGR 390 program at Shippensburg University. I would like to thank Kaggle for providing the datasets of song lyrics and a thanks to Suno AI for giving the chance to generate audio using their API. I would also like to thank the open-source community that has developed libraries and tools that were used in the creation of this application.

References:

[1] C. Payne, MuseNet. https://openai.com/blog/musenet, 2019.

[2] D. Eck et al., Magenta: Music and Art Generation with Machine Intelligence. https://magenta.tensorflow.org, 2016.

[3] D. Jurafsky and J. H. Martin, *N-grams*. https://web.stanford.edu/~jurafsky/slp3/3.pdf, 2025.

[4] B. Wandowando, *Spotify Songs with Attributes and Lyrics Dataset*. https://www.kaggle.com/datasets/bwandowando/spotify-songs-with-attributes-and-lyrics/data, 2023.

[5] Suno API, *Suno API Documentation*. https://sunoapi.org, 2025.

SENTIMENT ANALYSIS IN MUSIC EVALUATION AND RECOMMENDATION

Zachary Yourkavitch
Shippensburg University
vakruoy@gmail.com

ABSTRACT

Many current services for music recommendation utilize factors such as genre or recommend other songs that are listened to by users with similar listening patterns. In an era where artificial intelligence is making its way into many services, it seems sound to reason that utilizing sentiment analysis to recommend songs based on lyric content would be a potential advance. The usefulness of this approach was tested by having users interact with two versions of a program. One did not utilize its sentiment analysis model for user recommendations, while the other used a finetuned Longformer model and factored its output into recommendations. After subjects used both versions of the program, there was enough evidence to conclude that factoring sentiment analysis output into the recommendation feature improved user satisfaction.

KEY WORDS

Sentiment Analysis, Artificial Intelligence, Music

1. Introduction

The integration of artificial intelligence into multiple disciplines and areas of everyday life has been developing at a rapid rate. One such implementation of artificial intelligence is known as sentiment analysis, which involves using natural language processing (NLP) tools to attempt to identify the general emotional tones within texts. Previous works have attempted to compare and contrast various methods of performing sentiment analysis [1]. One of the discussed methods was the machine learning approach, wherein an algorithm is trained on a dataset featuring inputs with known outputs. This approach was utilized in the research being documented in this paper. According to previous work, sentiment analysis using machine learning often demonstrates the highest accuracy in classifying texts. On the other hand, it often sacrifices versatility and only boasts that accuracy in the same domain as the dataset in which the classifier was trained on. Other research [2] has yielded a proposed framework for music recommendations that utilizes sentiment analysis titled SentiSpotMusic. This framework aimed to solve the issue of modern music services supplying users with predefined playlists rather than adapting to the mood of the user.

This research aims to further previous work by building a functioning program that will use sentiment analysis to refine music recommendations. Its effectiveness will be judged through comparison to a previously built program that offered a recommendation feature that did not factor in sentiment analysis. This program will utilize sentiment analysis models trained on Google's GoEmotions dataset [3], a manually annotated dataset of short pieces of text labeled with associated emotional states. These models will identify emotional nuance within song lyrics, and apply that information to provide users with song recommendations that are more refined than recommendation features lacking sentiment awareness.

2. Methods

In order to test the effectiveness of sentiment analysis in the domain of music, we performed an experiment that involved having users interact with two versions of a sentiment analysis and music recommendation program.

2.1 Program Development

Two programs were developed in this research. Both displayed a recommendation from the same artist as the user-given song and a different one, though the second version's recommendations are refined based on scores. Version 1 of the program was built prior to this semester of research as a personal project. The sentiment analysis in version 1 was performed by a preexisting model created for sentiment analysis [4]. In version 1, though, this analysis was only presented as output for the user and not utilized for any recommendation. The model performing the analysis is fed lyrics that are retrieved by the LyricsGenius Python library [5]. The recommendation feature simply used the LastFM API [6], a tool for retrieving data from the online music service "Last.fm", and returned the first recommendations that service returned. Version 2 is a more in depth take on the same functionality. It uses a larger scale Longformer model [7], which was finetuned on the GoEmotions dataset to become a sentiment classification model, for sentiment analysis. For context, a Longformer model is a type of model that aims to understand the relationships

between all parts of a large input, rather than simple sequential processing. The GoEmotions dataset contains 27 emotions and a neutral as opposed to the standard 1, 0, and -1 values featured in traditional sentiment analysis. Multiple languages are also supported within this version thanks to the Googletrans Python library [8]. This version also utilizes the LastFM API to fetch candidate recommendations, while also implementing the ReccoBeats API [9], which is a service that provides users access to a large-scale music database, to fetch audio features for the user-given song. Although the candidate recommendations are retrieved via the same external API, this time they are refined via scoring. Scores are based on the cosine similarity of audio features combined with an average sentiment alignment value derived from the summation of the difference in confidence levels for each emotion shared by both tracks. This method of scoring was chosen to balance both audio features and the emotional tones detected in the lyrical content. In the end, both versions displayed a recommendation from the same artist as the user-given song and a different one, with version 2's recommendations being refined based on scores.

2.2 Empirical Study

To test whether sentiment analysis was able to improve user experience as utilized in version 2, an empirical study was conducted. This experiment was carried out in two phases. The first phase took place on Shippensburg University's campus and saw students, staff, and faculty interact with both programs and rate each facet of them on a scale of 1 through 10 via a Google Form. These aspects were the quality of the sentiment analysis, the song recommendation from the same artist, and the song recommendation from a different artist for each version. The second phase saw the same experiment open up to those outside of the university who were willing to participate as well.

A total of 42 participants took part in the experiment across both phases. One subject's data had to be neglected due to a bug encountered during the experiment, meaning there ended up being 41 total records that could be analyzed to produce results. To analyze the results, the data was imported into Minitab, a statistical analysis software. Paired t-tests were run on each of the surveyed facets of version 1 and 2 to test for improvement from the latter to the former. Afterwards, Cohen's d_z was calculated from the paired t-test outputs in order to determine effect size. Finally, frequency histograms and boxplots were created to visualize differences in reported scores between the two versions.

3. Results

Based on the information garnered from Table 1, it is evident that there was a significant improvement all around from version 1 to version 2.

Table 1: Paired t-test and effect size results

Facet	v1 Mean	v2 Mean	t	p	Cohen's d_z
Sentiment Quality	5.61	7.10	-3.14	0.002	-0.49
Same-Artist Rec	7.81	8.61	-2.45	0.009	-0.38
Different-Artist Rec	6.98	8.05	-3.39	0.001	-0.53

Note. Negative values indicate that version 2 scored higher than version 1 because differences were computed as (Version 1 – Version 2).

Within the results featured in Table 1, a negative effect size (Cohen's d_z) is observed. The table also displays heightened means for version 2 compared to its earlier counterpart, as well as justification for statistical significance in the form of p-values.

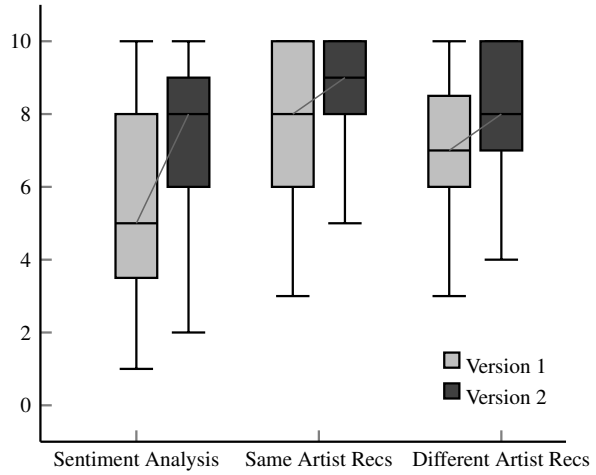


Figure 1: Box plots demonstrating the difference between scores received by versions 1 and 2 on all facets.

The box plots shown in Figure 1 serve to visualize the differences between version 1 and version 2's performances in each measured area. Overall, the general behavior of version 2 performing better persists in this visualization.

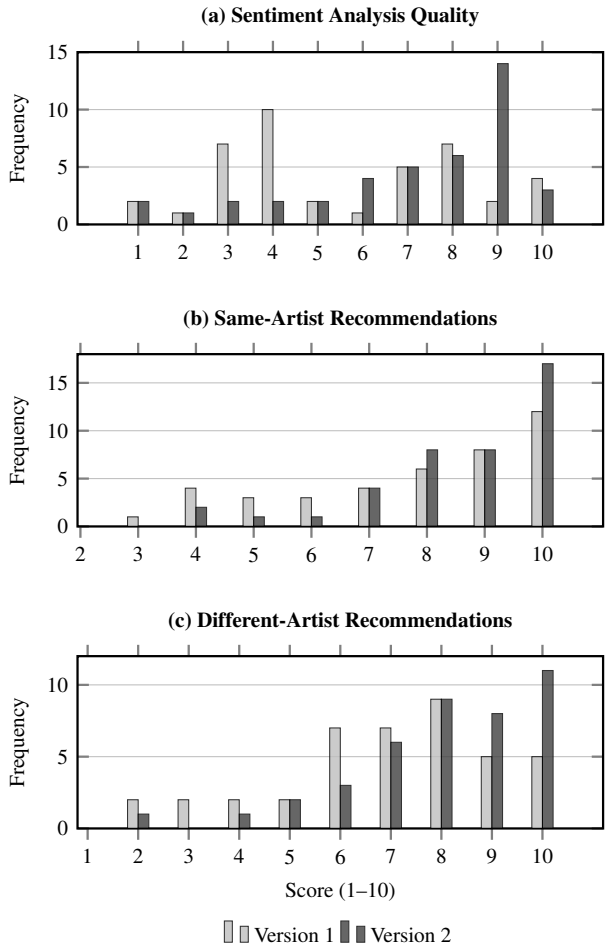


Figure 2: Frequency histograms displaying how often each version received each score for each facet.

Figure 2 conveys that in most cases version 2 received a greater number of the highest scores. In sentiment analysis, version 1 actually slightly outperformed version 2 when it came to 8's and the highest possible score of 10. When considering scores of 9, though, version 2 drastically outnumbers version 1.

All visualizations and metrics derived from the empirical data point to version 2 outperforming version 1. The overall negative effect sizes, greater mean scores, and a mostly greater frequency of higher scoring responses all convey that version 2 was generally better received than version 1 was.

4. Future Work

Although the overall goal of the project was reached and supported the idea that sentiment analysis can be used to improve user experience with music recommendations, it should be noted that there are clear areas of limitation. To begin with, there are a number of cases within both programs that can lead to errors. Some of these cases include timing out due to rate-limited APIs and certain languages causing errors within the translator. There are also limitations outside of program bugs. For example, the API for fetching candidate recommendations always returns a smaller pool of same-artist recommendations, meaning the likelihood of returning the same song across both versions is higher. Having a larger pool would give the versions a better chance to display their different methods of recommending tracks. There is also the inherent issue of translation removing some nuance or meaning that is only present in the track's native language. Future work could make improvements in many of these areas of weakness. An easy fix would be to simply increase the number of candidate same-artist recommendations in order to increase the chances of a differing experience between versions 1 and 2. Additionally, a next step could be to train a sentiment analysis model on multiple languages, rather than having to rely on translation to English. This would obviously require a much larger dataset and would increase the time it would take to finetune or train a model, but it is possible and could improve user experience when inputting foreign songs.

5. Conclusion

In spite of the limitations mentioned above, this study still served as a strong proof of concept that an effective enough sentiment analysis can be used to improve a user's experience with a music recommendation app. Many current recommendation features within existing apps base most of their recommendations on a mixture of songs in the same genre and songs that are popular with other listeners who listen to the same source song. To an extent, audio features may be factored in, but that goes alongside genre as songs in the same genre often have some similar characteristics. To improve recommendations to their users, applications could utilize an effective sentiment analysis model to also recommend songs based on lyrical content and song meaning. This would be especially useful for the many platforms that already attempt to create playlists for users based on emotions or general feelings. Further advances in this concept could refine the ability to create a fitting mood-based playlist and recommend a better song for any given emotion.

References

[1] M. D. Devika, C. Sunitha, & A. Ganesh, Sentiment analysis: A comparative study on different approaches, *Procedia Computer Science, 87*, 2016, 44-49.

[2] E. Sarin, S. Vashishtha, Megha, & S. Kaur, SentiSpot-Music: a music recommendation system based on sentiment analysis, *2021 4th International Conference on Recent Trends in Computer Science and Technology (ICRTCST)*, Jamshedpur, India, 2022, 373-378.

[3] D. Demszky, D. Movshovitz-Attias, J. Ko, A. Cowen, G. Nam, & S. Ravi, GoEmotions: A dataset of fine-grained emotions, *Proc. 58th Annual Meeting of the Association for Computational Linguistics*, Online, 2020, 4040–4054.

[4] S. Lowe, "roberta-base-go_emotions," Hugging Face, 2024. Online. Available: https://huggingface.co/SamLowe/roberta-base-go_emotions. Accessed: Jan. 9,. 2026.

[5] J. W. Miller, "LyricsGenius," Github, Oct. 26, 2025. Online. Available: https://github.com/johnwmillr/LyricsGenius. Accessed: Jan. 9, 2026

[6] Last.fm Ltd., "Last.fm Music Discovery API," Last.fm. Online. Accessed: Jan. 9, 2026.

[7] I. Beltagy, M. E. Peters, & A. Cohan, Longformer: The long-document transformer, *arXiv:2004.05150*, 2020.

[8] S. Han, "Googletrans: Free and Unlimited Google Translate API for Python," Read the Docs. Online. Available: https://py-googletrans.readthedocs.io/en/latest/. Accessed: Jan. 9, 2026.

[9] Reccobeats, "Reccobeats Music Recommendation and Database API Service," Reccobeats. Online. Available: https://reccobeats.com/. Accessed: Jan. 9, 2026.

Teaching Compiler Construction with LLVM: Bridging Theory and Practice

Tristan Rush
Department of Computer Science, Millersville University
Millersville, PA, USA
tjrush@millersville.edu

Abstract

Offering compiler construction in undergraduate curricula is commonly debated topic. Compiler engineering is a niche area of work in today's industry. However, students taking a compiler construction course will learn valuable skills not typically associated with compiler construction. It is important to use a course such as compiler construction as a way to provide students exposure to the reality of programming. Using frameworks and tools to support the development process not only eases the technical demand, but also allows a better balance of theory and implementation.

Code generation is a significant challenge for students implementing compilers in undergraduate compiler construction courses. Traditional course projects often require students to target architecture-specific assembly, which increases implementation complexity and limits portability. Modern compiler infrastructures such as LLVM instead provide a portable intermediate representation and a mature optimization framework widely used in industry.

This paper explores the integration of LLVM into compiler construction courses. We discuss traditional approaches to the code-generation phase of compiler construction against LLVM-based code generation. Identifying common challenges students face when working with LLVM within a course setting.

Keywords

Compiler Construction, Computer Science Education, LLVM, Educational Compilers

ACM Reference Format:
Tristan Rush. 2026. Teaching Compiler Construction with LLVM: Bridging Theory and Practice. In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Compiler Construction is an elective course which is not offered often at our university. In its place, many students aim to take electives more directly applicable to industry trends. While ABET accredited computer science programs require courses like Computer Architecture and Operating System [1], theoretical courses like Computation Models are not required. Under the same guidelines, this also means that many undergraduate students may lack exposure to formal language theory such as regular expressions and automata. Compiler construction courses offer a chance for instructors to cover these topics and bridge them to industrial programming practices.

Compiler frameworks, beyond providing an industrial tool that powers many of our programming languages today, aid instructors in achieving the balance of theory and implementation throughout the course [3]. In stages of compiler development, instructors often use tools like FLEX (Fast Lexical Analyzer) [3, 11], GNU Bison [3, 4], and ANTLR [3] to automate the production of language recognizers and parsers. However, these tools fall short in features surrounding code generation.

In pursuit of making compiler construction more approachable for core curriculum, instructors have waited for a code generation tool to be developed [4]. In its place we have seen instructors opt for a smaller input language [2] and intermediate representations [3] to help simplify the process. Compiler courses that have previously had students produce an intermediate representation have demonstrated equal if not increased success among students [2, 4].

In this work, I propose the use of LLVM in compiler construction courses to give professors more time to convey the theoretical concepts behind compiler design, building on existing efforts to make compiler construction more approachable for the core curriculum and highlighting how exposure to an industrial-grade framework helps students develop as computer scientists.

2 BACKGROUND

As a student at Millersville University, I was able to study compiler construction individually with a professor. Through these means, I was encouraged to conduct independent research on tools and frameworks to integrate into my compiler. In the case of code generation, my professor had provided the option of using LLVM instead of the standard IA-32 assembly mandated for code generation within the normal course.

For each section of the course, I followed a schedule that reflected the course's normal progression, meeting equivalent deadlines for each lab and assignment. There were no independent lectures, instead a weekly meeting time for me to ask any questions surrounding provided lecture notes, research, and assignments.

Both as an (unintentional) challenge and a demonstration of the infrequence of the course being offered, there were no other students currently enrolled within the program that had taken the compiler construction course. This provided me with limited outside influence from other students on the shape and structure of my compiler.

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

```
/****************************************************/
/*   File: gcd.cm                                   */
/*   A C- program to perform Euclid's Algorithm     */
/*     to compute gcd                               */
/****************************************************/

int gcd (int u, int v)
{
  if (v == 0)
    return u;
  else
    return gcd (v, u - u/v*v);
  /* u-u/v*v == u mod v */
}

int main(void)
{ int x;
  int y;
  x = input();
  y = input();
  /* printf ("gcd(%d, %d) = %d.\n", x, y, gcd (x, y)); */
  output (gcd (x, y));
  return 0;
}
```

Figure 1: An example input program for student compilers written in "C Minus"

3 CODE GENERATION

Undergraduate compiler construction courses typically guide students through the first stages of developing a compiler, including lexical analysis, parsing, semantic analysis, and code generation. Labs for the course start off by enforcing knowledge of theoretical concepts before requiring students to write their own compiler for a small teaching language for the remainder of the semester. While the first stages of compilation are supported through the use of tools such as FLEX and Bison, code generation remains one of the most challenging components for students.

In many traditional compiler projects, students would generate architecture-specific assembly instructions. This approach strengthens concepts taught in computer architecture courses. However, targeting assembly directly introduces unnecessary complexity as students already experienced much of the challenges of register allocation and assembly instructions through computer architecture courses.

At our university, the compiler construction course follows this approach. Students implement a compiler for Kenneth Louden's "C Minus" [10] An example of which can be seen in **Figure 1**. While they are given the opportunity to implement the stages through any means, they are required to use C++, and it is recommended that students use the visitor design pattern with double dispatch in order to traverse the abstract syntax tree (AST) conveniently. Once the students reached the code generation stage, they were instructed to produce IA-32 assembly as an output for the compiler, which would then be linked using GNU's binutils. While this approach provides valuable knowledge into low-level code execution, students often encounter challenges due to the complexity of having to dynamically produce assembly.

Modern compiler infrastructures such as LLVM provide an alternative approach to code generation. Rather than directly producing assembly, compilers are able to use LLVM's intermediate representation (IR), a portable low-level language designed for multi-stage optimization and code generation for multiple architectures [7]. This abstraction layer has the potential to simplify the backend implementation of compiler construction while still exposing students to important compiler concepts.

4 INTEGRATING LLVM

Despite the standard output of code generation for student's compilers being IA-32 assembly. I was given the option to use LLVM as an alternative. For requirements of the code generation stage, my compiler had to successfully executed a minimum of 3 programs out of a specified list of example programs in 3 weeks time. During which, I was able to learn the syntax around LLVM's intermediate representation, its IR Builder library, and its optimization passes.

In the process of developing my compiler, I had found myself conducting more research than I might have needed when compared to producing IA-32 due to the comparitively little amount of documentation surround LLVM's IR Builder library. Not only did the prerequisite Computer Architecture class provide me with a background in IA-32, but there are a number of resources available as well [5, 6]. However, the doxygen documentation and tutorials provided for LLVM [8, 9] still provided enough context to implement Louden's "C Minus" without additional resources.

Static single assignment also proved to be a challenge when implementing branching logic into my compiler. As a workaround, I found myself implementing redundant load instructions within certain nodes in order to prevent accessing registers/value pointers outside of the current scope.

I found that I took advantage of many higher-level language features when producing LLVM IR that would not be possible when generating IA-32. An example of this would be using a hashmap to map lexemes to their respective llvm value pointers. This eliminated the need to keep track of what values are in registers, eliminating much of the complexity of architecture-specific code generation.

Portability was also a big advantage of LLVM, as it was possible to produce a program executable for any architecture supported by LLVM through specifying the target triple, which could be determined through existing c++ functionality.

5 PRELIMINARY RESULTS AND EVALUATION

Figure 1 provides an example of an input program that student compilers were required to support. There were a minimum of 3 example programs that did not required an input. However, I chose programs with input and output as they demonstrated more functionality of my compiler. **Figure 2** shows a segment intermediate representation generated by my compiler when provided the input program shown in Figure 1. **Figure 3** shows the result of running the executable outputted by the compiler I developed. The function will return the greatest common denominator of 2 user specified numbers.

In the process of researching compiler construction, I had spent about a week with my instructor going over the various techniques compilers engineers take advantage of to optimize an input program. Unfortunately, I was unable to implement any of these techniques, or any of the optimization passes provided by LLVM into my compiler. Between the time spent researching LLVM to discover LLVM's IR Builder Library, and other time used facing burnout during the

```
define i32 @main(i32 %u, i32 %v) {
entry:
  %u1 = alloca i32, align 4
  store i32 %u, ptr %u1, align 4
  %v2 = alloca i32, align 4
  store i32 %v, ptr %v2, align 4
  %x = alloca i32, align 4
  %y = alloca i32, align 4
  %0 = load i32, ptr %x, align 4
  %1 = call i32 @input()
  store i32 %1, ptr %x, align 4
  %2 = load i32, ptr %y, align 4
  %3 = call i32 @input()
  store i32 %3, ptr %y, align 4
  %4 = load i32, ptr %x, align 4
  %5 = load i32, ptr %y, align 4
  %6 = call i32 @gcd(i32 %4, i32 %5)
  call void @output(i32 %6)
  ret i32 0
}
```

Figure 2: A segment of LLVM IR for the input program

```
tjrush@csciarch05 ~/4/cminus_compiler> ./gcd
14239854
21128
38
tjrush@csciarch05 ~/4/cminus_compiler>
```

Figure 3: Running the executable provided after using clang on the LLVM IR

parsing stage, I had wasted about 2 weeks. After the semester concluded, I had spent two weeks introducing LLVM's optimization passes into my compiler, and was left with additional time.

6 CONCLUSION

The preliminary results demonstrate that using LLVM for code generation in a compiler construction class is a feasible practice. The main challenges faced when using LLVM for code generation surrounded documentation, with static single assignment proving to be mainly an exercise in logical rationale. It is possible for compiler construction courses to spend more time addressing optimization, which is increasingly relevant as software engineering practices promote more abstraction layers. Additional documentation for LLVM's IR Builder library may provide instructors with more time to address theoretical aspects of compiler design. Further research may prove an ideal balance of theoretical instruction and practical implementation, which may support an argument that less documentation is a better reflection of large-industry repositories, and an aspect of programming the students may benefit from exposure to.

References

[1] ABET. 2025. Criteria for Accrediting Computing Programs, 2025–2026. https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2025-2026/#4 Accessed: 2026-03-08.

[2] Doug Baldwin. 2003. A compiler for teaching about compilers. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Navada, USA) *(SIGCSE '03)*. Association for Computing Machinery, New York, NY, USA, 220–223. doi:10.1145/611892.611974

[3] Rafael del Vado Vírseda. 2021. Learning Compiler Design: From the Implementation to Theory. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2* (Virtual Event, Germany) *(ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 609–610. doi:10.1145/3456565.3460041

[4] Akim Demaille. 2005. Making compiler construction projects relevant to core curriculums. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE '05)*. Association for Computing Machinery, New York, NY, USA, 266–270. doi:10.1145/1067445.1067518

[5] Intel Corporation. 2002. IA-32 Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference. https://pdos.csail.mit.edu/archive/6.097/readings/intelv2.pdf. MIT 6.097 course reading.

[6] Intel Corporation. 2026. Intel® 64 and IA-32 Architectures Software Developer's Manual. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html. Accessed: 2026-03-08.

[7] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization* (Palo Alto, California) *(CGO '04)*. IEEE Computer Society, USA, 75.

[8] LLVM Project. 2026. Getting Started / Tutorials. https://llvm.org/docs/GettingStartedTutorials.html. LLVM documentation. Accessed: 2026-03-08.

[9] LLVM Project. 2026. llvm::IRBuilder Class Template Reference. https://llvm.org/doxygen/classllvm_1_1IRBuilder.html. LLVM Doxygen documentation. Accessed: 2026-03-08.

[10] Kenneth C. Louden. 1997. *Compiler Construction: Principles and Practice* (1st ed.). Cengage Learning, Independence, KY.

[11] Julie Zelenski and Keith Schwarz. 2012. flex In A Nutshell. CS143 Handout 05, Stanford University. https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/050%20Flex%20In%20A%20Nutshell.pdf Course: CS143 Compilers.

THE EFFECT OF LEARNING RATE AND HIDDEN LAYER SIZE ON A NEURAL NETWORK LEARNING TO PLAY *PUYO PUYO*

Christian Honicker
ch8580@engr.ship.edu
C. Dudley Girard
cdgira@ship.edu

ABSTRACT
The development of artificial intelligence, especially neural networks, to play video games is a common topic amongst researchers and developers alike. One game that has received very little attention in this regard though is the falling block puzzle game *Puyo Puyo*. Despite it having high potential for AI development, there's relatively little literature on *Puyo Puyo* AI. As such, little is known on what factors might affect its performance. This paper provides results of an experiment to determine how learning rate and number of hidden nodes impact the performance of a *Puyo Puyo*-playing neural network.

KEY WORDS
Puyo Puyo, Games, Artificial Intelligence, Neural Networks, Hyperparameters

1. Introduction and Problem Description

Ever since the conception of Artificial Intelligence (AI for short), developers and researchers alike have been making efforts to coerce AI to play their favorite games. Whether it's a strategic, turn-based board game like Chess or Go, or something more action-based like arcade games and first-person shooters, many projects have been dedicated towards developing AI's to perform well in their respective games. Different approaches have been used in the development of these AIs, but one popular approach amongst developers and researchers is the neural network. Neural networks have been developed and tested to play a wide variety of games, including some of the ones mentioned previously. And one game that shows a lot of potential for AI success using a neural network is *Puyo Puyo*, a falling block puzzle game that bears resemblance to games like *Tetris* and *Dr. Mario*.

This paper will explore how a neural network can be used to play a puzzle game like *Puyo Puyo* and what factors of the neural network would impact its ability.

2. Literature Background

This literature background will cover what neural networks are and how they work, how the game of *Puyo Puyo* is played and the previous attempts to build AIs that play it, and a case study of neural networks playing the similar game of *Tetris* and how the techniques they used could apply to a similar *Puyo Puyo* network.

2.1 Neural Networks

Neural networks are a type of AI that have grown increasingly popular over the past decade or so. Neural networks have been implemented in a very wide range of applications, such as statistical prediction models, image recognition, natural language processing, and playing games [8].

Generally speaking, a neural network consists of three components: an input layer, some number of hidden layers, and an output layer. Each layer consists of some number of nodes, and each node is designed to produce a value, usually a decimal ranging from 0 to 1. There may also be some number of bias nodes, nodes whose values remain constant at each layer. There's then weighted edges usually connecting from every node at a given layer to every node in the next layer. These weighted edges help compute the values of the next layer iteratively, starting at the input layer until it eventually reaches the output layer, where the values of the output nodes determine what action to take [8]. A diagram showcasing a simplified neural network structure is shown in Figure 1.

The process by which values are carried and processed from the input nodes to the output nodes is called forward propagation. Forward propagation generally consists of two steps: preactivation and activation. In preactivation, the weighted values of all the edges going into a given node are summed together. Then in activation, this weighted sum is then passed through an activation function [7]. There are many possible activation functions a network could use at this stage. Some of the most common activation functions are the sigmoid function, which

normalizes the input value to be between 0 and 1, and the rectified linear unit—ReLU for short—which is typically defined as ReLU(x) = max(0, x), bringing the input up to 0 if it's negative. ReLU is the most common activation function used in neural networks nowadays, but many other functions are still in common use [13].

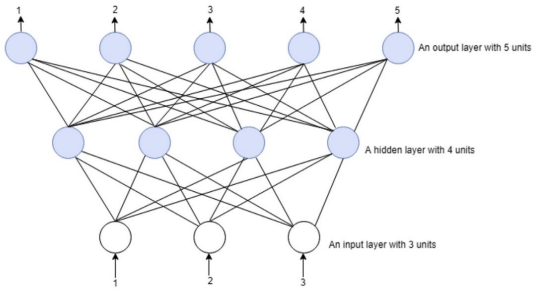


Figure 1: A simple neural network structure. Adapted from [7].

Once the neural network is set up, it then needs to be trained to determine the optimal configuration of the edge weights. There are many algorithms that exist to train these networks, and networks are often classified by how they're trained. Such classifications include feed-forward networks, feedback networks, convolutional networks, neuro-evolutionary networks, and many more [8]. In feed-forward networks, edge weights are adjusted via a process called backpropagation. In backpropagation, each edge changes by an amount relative to the error between the expected output and the actual output, see equation 1. For a given weight w, learning rate α, error function E, and output vector X, the change in weight Δw is defined as the partial derivative of error with respective to the weight multiplied by the learning rate [4].

$$\Delta w = \alpha \frac{\partial}{\partial w} E(X) \tag{1}$$

Equation 1 depends on the error function, the activation function, and where in the network the edge is. For example, in the instance of a weight w that connects from the hidden layer to the output layer, using a sum of squares error function $E(X)$:

$$E(X) = \frac{1}{2}(target_o - out_o)^2 \tag{2}$$

and a sigmoid activation function $g(x)$:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

would use equation 4. Where target output is $target_o$, actual output is out_o, and actual output before activation is out_h [4]:

$$\frac{\partial}{\partial w} E(X) = (-(target_o - out_o) * out_o(1 - out_o)) * out_h \tag{4}$$

This derivative of error is then multiplied by the learning rate, and then the weight changes by that amount. This process is repeated for every edge in the network. Backpropagation typically occurs after every produced output during training, but may be applied at more irregular intervals.

Neural networks are highly adjustable, in that there's a large number of factors to consider when structuring the network. These factors that influence the network's structure and how it learns are called hyperparameters [11]. Hyperparameters that influence the structure include the number of hidden layers, the number of nodes per hidden layer, and how edge weights are initialized. A hyperparameter that influences learning is the learning rate, a real number greater than 0 that adjusts by how much the network changes its edge weights [11]. Learning rate is expressed as α in Equation 1.

Neural networks are a very versatile type of artificial intelligence. They're highly configurable and have lots of existing tools to enable their creation. These reasons have allowed neural networks to succeed in a wide range of applications, which has led many researchers and developers to use them for the purpose of playing games.

2.2 *Puyo Puyo*

Puyo Puyo is a falling block puzzle game developed by SEGA, traditionally played as a two-player versus game. The game is played by dropping pairs of colored puyos onto a 6 by 13 grid. When four or more puyos of the same color are matched together, they pop, clearing space and allowing the puyos above to fall. Puyos popped this way are converted to points for the player's score, as well as garbage puyos that are sent to the opponent's board that can only be cleared by popping a puyo near it. The goal of the game is to make the opponent "top out" by filling up their board—or more specifically, filling up their entire third column, the top of which is usually denoted with an X.

This is typically done by creating chain reactions of popping puyos, known simply as chains, as higher chains produce more garbage puyos for the opponent and a higher score [10]. An example of a chain can be seen in Figure 2. First a group of four blue puyos pop, then five greens, then four more blues, then finally four reds. The game can also be played with more than two players, in which the goal is to be the last player standing, or with only one player, in which the goal is to obtain as high of a score as possible.

Puyo Puyo is believed to be an appropriate game to train an AI, and particularly a neural network, to play for a number of reasons. First, the game rules are simple to model, with a fixed number of possible moves per board state and ways to identify "good" and "bad" moves for any given board state [10]. These reasons have led to a small

number of efforts to create & test an AI that plays *Puyo Puyo* [1,2,3,10].

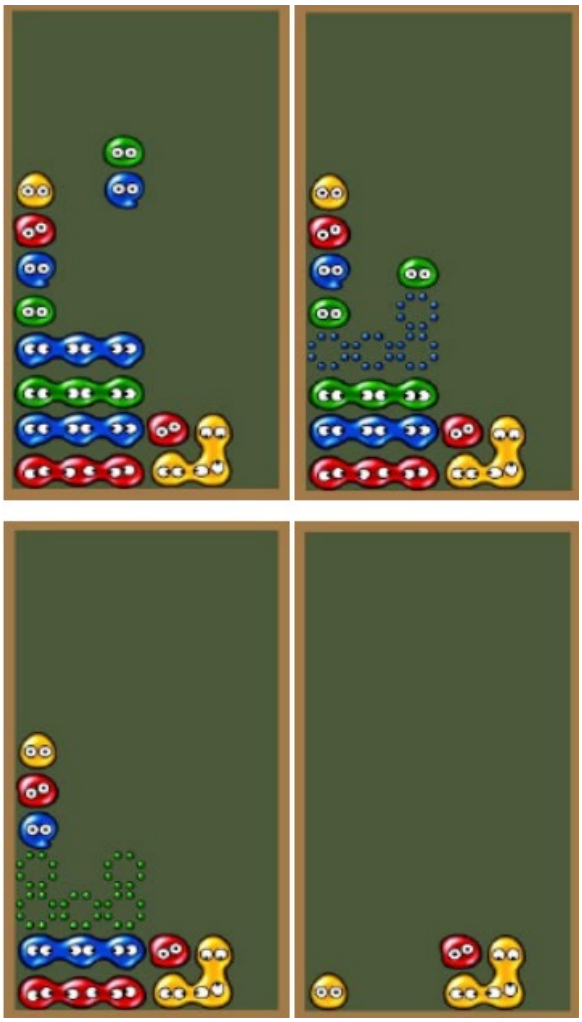


Figure 2: Chaining Example [10]

One of the few scholarly published efforts to create an AI that plays *Puyo Puyo* is that of Paul Hanson and David Moffat, who implemented a Monte Carlo Search (MCS) algorithm to play the game [10]. The AI was tested in trials that lasted either until the AI topped out, or until 150 pieces were placed, whichever came first. Then, the AI's average score at each interval of 10 pieces was computed, then compared to other search implementations. The results were somewhat mixed; the MCS performed similarly well to implementations of Random Search and Breadth-First Search, but performed significantly worse than an Exhaustive Search that wasn't time-limited. Different depth limits of the MCS algorithm were also compared against one another, to which the results found a depth of 4 consistently produced the best results [10].

Another noteworthy attempt at a *Puyo Puyo* AI, even though it's not published, is that of Dylan Leclair's evolutionary approach [3]. Leclair created networks of various sizes and complexities, then trained each network over the course of several hours at a time via neuroevolution. His best results were achieved by a network with two hidden layers at 150 nodes each, as well as a modified fitness score that valued chains more highly, which notably was able to form some basic color matching and chaining strategies. Leclair noted towards the end of his report that if he were to revisit the project, he would change the way the output was formatted from raw button outputs to piece locations, the output method that was used in in Hanson and Moffat's approach, and was likely made better for it [3,10].

Further attempts at *Puyo Puyo* AIs exist, but with lesser documentation such that it's difficult to work out exactly what methods they're using. For example, one implementation developed by a group simply known as "puyoai" was showcased to be capable of some complex chaining strategies [2]. However, the documentation is written entirely in Japanese, which has proven difficult to understand. These efforts have even extended to SEGA, the game's developers; more recent *Puyo Puyo* games have added a feature called Core AI, which is also capable of some complex chaining strategies and is even able to keep up with the best human players [1]. However, this AI is closed source, and it's not known exactly what methods were used to develop it.

While the amount of literature regarding the development of *Puyo Puyo*-playing AIs is relatively small, the literature that is out there showcased relative success on the matter, as well as room for improvement. The game has proven to be well suited for AI development, particularly using a neural network. Additionally, these attempts have showcased AIs that have been able to develop strategies to play the game at a competent, if not a proficient level. In addition to these approaches, it's worth investigating whether AI in similar games could provide additional insight.

2.3 *Tetris* Case Study

Puyo Puyo belongs to a specific genre of game called falling block puzzle games [5], a genre which contains a number of games that have an equal or greater amount of AI-related literature. Perhaps the single most popular game in this genre, both in terms of research and in general, is *Tetris*. Alongside being one of the most popular and highest-selling games in the world, *Tetris* has a very large amount of existing literature in regards to the development of AI using neural networks. Given that *Puyo Puyo* and *Tetris* are very similar games, it can be reasoned that some of the methods used to build functional *Tetris* AIs can also be applied to *Puyo Puyo*. This section will cover two such attempts at building a *Tetris*-playing AI.

The first of these is an attempt by Nicholas Lundgaard and Brian McKee at a *Tetris* AI using a neural network and reinforcement learning. Their approach implements a large number of high-level states and actions, seemingly

designed to instill specific strategies in the network from the start. The work also compares a number of different heuristic agents—agents that differ in the fitness estimation functions that determine what the network should be trying to achieve—against one another; interestingly, the "Maximize Lines" agent performed the worst, even worse than a random agent, meanwhile the "Clear Board" agent performed the best. They also compared how the network performed in single player and competitive contexts, in which they found the network performed poorly when playing against a similar *Tetris*-playing AI made by Pierre Dellacherie [9].

The other instance of interest is that of Matt Stevens and Sabeek Pradhan, who built their *Tetris*-playing AI using a convolutional neural network. Notably, they referenced the efforts of Lundgaard and McKee as a preexisting example of a functional *Tetris* neural network, but pointed out how it makes use of "explicit featurization"—that is, it's able to interface with the game and get game variables directly. Instead, Stevens and Pradhan were more interested in building an agent that processed raw pixel data—by effectively taking screenshots of the game and processing those screenshots in a similar manner to how a human player would. To do this, they made use of a convolutional neural network, a special type of neural network that's seen significant usage in image processing. They also implemented an epsilon-greedy policy, in which the agent has a small probability to make a random move as opposed to the optimal move to allow it to search more of the game space [12]. Lundgaard and McKee's AI made use of an epsilon-greedy policy as well [9], but it's more clearly emphasized and explained in Stevens and Pradhan's paper. While they did have trouble getting the network to achieve proficiency, they were able to at least train a network that could play the game on a basic level, and found that the AI performed best when there was minimal delay between actions and rewards [12].

This is far from an exhaustive list of efforts to create a *Tetris*-playing neural network, but they are notable examples. They contain a lot of ideas that would be worth implementing in a similar *Puyo Puyo* playing network, such as the epsilon-greedy policy. They also contain some ideas that don't seem as appealing—as interesting as Stevens and Pradhan's convolution-based approach is, it's easier to just allow the network to be able to interface with the game directly. But they've both shown moderate success in constructing a competent *Tetris*-playing agent, suggesting that the techniques they used could be adapted to create a similarly performing *Puyo Puyo*-playing agent.

3. Primary Objective

The primary objective for this project: To determine the impact of learning rate and number of hidden neurons on a feed-forward neural network's ability to play the falling block puzzle game *Puyo Puyo*, subject to the following governing proposition:

- The network plays a simplified version of the game that is single player and in which pieces are placed at the desired position at the top of the board.

4. Solution Description

To achieve this objective, a highly configurable feed-forward neural network was created to play a simplified version of *Puyo Puyo*. Specifics on how the game and neural network were implemented are as follows.

4.1 *Puyo Puyo* Implementation

Leclair's Puyoai program served as the basis for the project, as the codebase was fully open-source, relatively small, and had its own neural network easily separable from the main game [3]. The game was then modified to remove Leclair's network and allow for the placement of pieces at the top of the board at the player's discretion, as opposed to pressing buttons to move the current piece. This was done by assigning each possible placement at the top of the screen to a number from 0 to 21. There are 6 possible positions that a piece can be placed vertically, and 5 possible positions it can be placed horizontally. Each of these positions also has 2 rotations the piece can be in. Combining these gives 2 * (6 + 5) = 22 possible placements.

4.2 Neural Network Structure

Leclair's Puyoai program was written in Python using the PyGame engine, so a basic feed forward neural network was constructed in the Python language. In order to allow the network to train in real-time as it played the game, a custom training loop was created.

The game board consists of a 6 by 13 grid, each of which can contain one of four colored puyos—red, blue, green or yellow—or be empty. The player can also see their current piece, consisting of a pair of puyos, as well as their next piece, also a pair of puyos. A single puyo was encoded as a three-dimensional vector [A, B, C]. The A dimension represented the color of the first puyo in the pair of puyos to be played next. The B dimension represented the color of the second puyo in that same pair if it was different from the first puyo in the pair. The C dimension represented any colors that did not match the colors for the A or B dimensions. All three dimensions were set to 0 if the location did not contain a puyo.

Instead of representing the full board, only the puyos around which the next puyos would land given their drop point were represented. Counting the two puyos being dropped and the max possible puyos around them meant 13 total puyos had to be represented by the input layer for a total of 39 input nodes. All 39 input nodes would get utilized if the puyos landed horizontally and on different rows in the center. The fewest input nodes would get used

if the puyos landed on the bottom row next to an edge. An example, where the puyos fell vertically is shown in Figure 3. The red box represents the puyos possible landing spot and the blue boxes the additional puyos to be represented in the input layer of the neural network.

Figure 3

The network then consists of a single hidden layer of variable size—the hidden layer sizes selected for study were 10, 20, and 30. This was then followed by an output layer of 1 node. This node would provide a score based on where the dropped puyos would land, with 22 different landing possibilities, see Section 4.1. During training each starting location was given a weighted chance of being chosen based on its score and the sum of all the other scores. To ensure all options had a chance the lowest score was set to 0.1 and all scores linearly adjusted relative to it. A weighted random selection was then made. During evaluation, whichever starting location's output had the highest value was the selected move.

Along with the hidden layer size, the learning rate was also manipulated to determine its impact on performance. The chosen learning rate values were 0.005, 0.01, and 0.02. For the activation function a modified ReLu function was used. It had a slope of 1 from 0 to 1, but values less than 0 had a slope of 0.01 and the same for values greater than 1. Initial weights were set to random values between -0.5 to 0.5.

4.3 Move Evaluation

In order for the network to train, it needed some expected output for it to compare against and train towards. Since there didn't exist a readily available *Puyo Puyo* neural network training set, and to create one would likely take considerably more time than was allotted, a live move evaluation system was implemented instead. This move evaluation system was based on five simple principles:

- Moves that increase all matched colors are very good.
- Moves that score points are very good.
- Moves that reduced the matched colors are bad.
- Moves that both increase matched colors and decrease matched colors are ok.
- Moves with an empty board are ok.

Several different iterations of this system were attempted, with these rules being what was used in the final version.

Moves that were labelled as very good were given a score of one. Moves that were labelled as good were given a score of 0.75. Moves that were bad were given a score of 0. Moves that were ok were given a score of 0.5 to 0.65. After scoring the move the neural network would then update its weights using backpropagation. Since we only focused on one possible output at a time, we didn't have to worry about trying to figure what move the neural network should have made. The random weighted select also ensured many different moves would be tried out during training. Both to help it find the good moves and find the bad moves.

4.4 Training and Evaluation

Each network was given 2000 moves to train. During test runs it was found that the neural network seemed to start mastering the game by 2000 moves. Once the game was "mastered" the neural network could play a single game forever. During the training period, each network would play games of *Puyo Puyo*, backpropagating after each move based on the board state it saw and the results of the move it played on it. Because a network could make several bad moves in a row, the more bad moves it made the less likely it would be to train off that move. If the network gets a game over before its training period runs out, it starts a new game, playing continuously until its total moves run out.

For evaluation, a network would start a new game and make a total of 200 moves. If the game ended early due to bad moves a new game was started until all 200 moves were used. After 200 moves the game was stopped and the score recorded.

5. Experiment

A total of 135 neural networks were trained for the experiment—15 for each combination of hidden layer sizes (10, 20, 30) and learning rates (0.005, 0.01, 0.02). The starting game for each of the 15 networks in a trial were seeded, meaning pairs of Puyos would arrive in the same order for each instance of that game, to eliminate any variance caused by random piece generation. However, if a game was lost then the seed was increased by 1 for the next game.

5.1 Analysis of Training

Before running the experiments, the following hypotheses were proposed:

- $H_{A\text{-}}$: The lower the learning rate the better the network will do.

- H_B: The greater the number of hidden nodes the better the network will do.

The lower learning rate was hypothesized as those neural networks would be less likely to over adjust on one bad or good move. The number of hidden nodes was hypothesized because more hidden nodes should allow the network to match a larger number of patterns more easily.

The randomness of what weights a neural network might start out with can greatly impact performance. As such the 5 networks that were the greatest outliers were dropped from each group. In some cases, networks completely failed to learn, scoring 0 points during evaluation. In other cases, networks randomly learned how to chain even though that was not built into the training feedback. These also got removed as they overperformed.

To evaluate these hypotheses, a Student's t-test will be used to compare the averages of each trial's 10 remaining networks. In Table 1 we can see the results organized to see the effect learning rate has on network performance.

Hidden Layers	Learning Rate	Average	Standard Dev
10	0.005	16244	3849
	0.01	17643	1977
	0.02	15648	1638
20	0.005	19050	5270
	0.01	17177	2373
	0.02	14481	2161
30	0.005	18548	4755
	0.01	15353	1356
	0.02	16799	2110

Table 1: Effect of Learning Rate

Based on the results of the t-tests for the effect of learning rate it was found that when there are 10 hidden nodes a learning rate of 0.01 is better than 0.02. For 20 hidden nodes a learning rate of 0.01 is better than 0.02 and a learning rate of 0.02 is worse than 0.005. For 30 hidden nodes a learning rate of 0.005 is better than a learning rate of 0.01. Unfortunately, all other comparisons came out as not statistically different. So, there is some support for the H_A hypothesis, but more work is needed to what if any effect learning has.

In Table 2 we can see the results organized to see the effect the size of the hidden layer has on performance. Looking at the effect of number of hidden nodes, the t-test found no statistical differences. This result creates the new question then of is 10 the minimum hidden nodes needed or would going beyond 30 nodes improve performance?

Learning Rate	Hidden Layers	Average	Standard Dev
0.005	10	16244	3849
	20	19050	5270
	30	18548	4755
0.01	10	17643	1977
	20	17177	2373
	30	16799	2110
0.02	10	15468	1638
	20	14481	2161
	30	15353	1356

Table 2

For hypothesis H_A there is some support for it, but not enough to declare it true. Additional experiments would be needed to see under what conditions is H_A true and when is it not true. Based on the results hypothesis H_B proved untrue and better support the statement that the number of hidden nodes will have no effect on the performance of the neural network.

6. Suggestions for Further Research

Some possible areas of focus that go beyond just the basic findings of this work:

- **More sophisticated move evaluation.** Is it possible to help the neural network understand how to create chain events for a higher score per move. One or two of the networks that trained appeared to achieve this, but purely by accident given their limited view of the board.
- **Neuroevolution.** Neuroevolution—in particular the Neuroevolution of Augmenting Topologies (NEAT) algorithm—was considered in the early proposal stages of this project, but was rejected under the assumption that to train a network via neuroevolution would take more computer-hours than were available. Even so, Leclair's AI showed the relative success of neuroevolution when applied to *Puyo Puyo* as well as room for improvement [3].

8. Conclusion

This was an attempt at the creation of a simple feed-forward neural network that plays the falling block puzzle game *Puyo Puyo* and an experiment to determine what hyperparameters of the neural network would impact its performance. The results showed that for learning rate it can have an effect on the learning performance. We hope that this project can serve as a starting point for further research on the development and evaluation of neural networks playing arcade-style puzzle games such as *Puyo Puyo*.

References:

[1] "Anyone figure out how to activate Core AI yet?," Steam Community, http://steamcommunity.com/app/1259790/discussions/0/3073117690253301376/.

[2] Puyoai, "Puyoai," GitHub, https://github.com/puyoai/puyoai.

[3] D. Leclair, "Puyoai," GitHub, https://github.com/dylanleclair/puyoai?tab=readme-ov-file.

[4] M. Mazur, "A Step by Step Backpropagation Example," Matt Mazur, https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/.

[5] "Category:falling block puzzle games," Wikipedia, https://en.wikipedia.org/wiki/Category:Falling_block_puzzle_games.

[6] Neuroph, https://neuroph.sourceforge.net/index.html.

[7] V. Luhaniwal, "Forward propagation in Neural Networks-simplified math and code version," Medium, https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250.

[16] "Welch's t-test," Wikipedia, https://en.wikipedia.org/wiki/Welch%27s_t-test.

[8] O. I. Abiodun et al., "State-of-the-art in Artificial Neural Network Applications: A survey," *Heliyon*, vol. 4, no. 11, Nov. 2018. doi:10.1016/j.heliyon.2018.e00938

[9] N. Lundgaard and B. McKee, "Reinforcement Learning and Neural Networks for Tetris," Technical Report, University of Oklahoma, 2006. [Online]. Available: https://www.idi.ntnu.no/emner/it3105/materials/neural/lundgaard-tetris-ann.pdf

[10] P. Hanson and D. C. Moffat, "Monte Carlo Search for a real-time arcade game, Puyo-Puyo," *Proceedings of the Symposium on AI Games, AISB*, 2014. [Online]. Available: https://doc.gold.ac.uk/aisb50/AISB50-S02/AISB50-S2-Hanson-paper.pdf

[11] P. Radhakrishnan, "What are hyperparameters? and how to tune the hyperparameters in a deep neural network?," Medium, https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a.

[12] M. Stevens and S. Pradhan, "Playing Tetris with Deep Reinforcement Learning," Convolutional Neural Networks for Visual Recognition CS23, 2016. [Online]. Available: https://cs231n.stanford.edu/reports/2016/pdfs/121_Report.pdf

[13] S. Sharma, "Activation functions in neural networks," Medium, https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

THE FUNDAMENTALS OF FUNCTIONAL PROGRAMMING IMPLEMENTATION AND OPTIMIZATION

David Good, Dr. Gary Zoppetti
Millersville University of Pennsylvania
dwgood@millersville.edu, Gary.Zoppetti@millersville.edu

ABSTRACT
This paper explores the implementation details needed when developing a compiler for a functional programming language. To implement a functional compiler, a full compiler application was written in C++ with next to no use of third-party libraries. With this compiler, we can inspect each stage of a functional program's compilation and can observe the techniques used to optimize the generated code at varying levels of compilation.

KEY WORDS
Functional Programming, Compilers, Optimization, Code Generation, C++

1. Introduction

Historically, the imperative paradigm that encompasses many popular programming languages, like C++ or Java, closely models the imperative architecture that these languages usually target. Yet, the imperative paradigm is only one way of expressing computation. There has been another paradigm which encompasses languages such as Lisp, Haskell, ML, OCaml, and more. That would be the functional programming paradigm. The core idea of functional programming is that, instead of juggling state within a sequential, imperative programming language, the programmer must simply tell the compiler what computations it would like to perform declaratively by composing functions. Placing such a restriction on the programmer could potentially limit the ability for them to make mistakes by forcing them to avoid state mutation and thus avoid logical errors that arise when code has side effects. The other important facet of functional programming is the basis on which it is named – functions are first class citizens of the language. What this entails is that functions themselves can be passed and used as values to express computation. This differs from most imperative languages, which favor the concept of mutable state and execution order as the basis of computation. Having functions as values gives functional programming languages the ability to implement useful features not seen in imperative languages, such as partially applied functions.

With a new programming paradigm comes the question, how do we implement such a language? From a compiler standpoint, imperative languages are fairly straightforward to implement – sequential instructions translate quite seamlessly to the average imperative machine. The advantages that functional programming provides (e.g. immutability, functions as values, expressivity, etc.) must come at some implementation cost, or otherwise they would be widely used in imperative languages already. One such cost is that of memory management, as these languages typically have a garbage collector for the amount of allocations needed to execute even simple functional programs [1]. Much research has been done into fast implementations of functional compilers, notably [1, 3, 5]. Further research has been done, but these texts are the basis for the implementation of functional compilers today, such as Haskell.

The purpose of this research is not to revolutionize the functional programming space. Rather, it is to serve as a basis of understanding as to how functional programming languages can be compiled into sequential code for most modern machines. Importantly, certain tactics to optimize the resulting programs will be identified, as to observe how functional programming languages can still compete with compiled imperative languages today.

2. Background

Before we can discuss the compiler implementation details, we must first understand the execution model that functional languages employ. From there, we can understand how we can achieve the model of execution with our compilation strategy. First, we must discuss the basis of all functional programming languages: the lambda calculus [1].

2.1. The Lambda Calculus

Since the inception of computing, the simplest and most powerful computational model is that of the Turing machine [2]. Turing machines are analogous to modern day computers, effectively being finite state machines with the ability to read an infinite tape and modify it according to a transition model, which effectively programs the machine [2]. While Turing machines serve as the basis for imperative languages today, the functional programming paradigm is built upon Church's research into the lambda calculus [1].

The lambda calculus is fundamental to functional computing, as it is sufficiently expressive to allow the transformation of high-level functional programs into the lambda calculus [1]. Importantly, the lambda calculus is

both simple, with only a limited few constructs to implement, and expressive, in that lambda calculus can represent any computable function, and any higher-level functional programming language can be built upon a working implementation of the lambda calculus [1].

```
(λ x . x + 1) 2
```

Figure 1: Example lambda calculus code. The result of the function application is 3.

Only a few features are supported by the base lambda calculus: function applications, lambda expressions, and atomics such as variables or built-in constants. Function application, the method of computation in the lambda calculus, is represented by juxtaposition of expressions. Applying functions can be more powerful than allowed in imperative languages; currying, which is the practice of functions only accepting one argument, can be leveraged to create interesting partially applied functions, such as a function that adds a constant to the given argument [1]. This is possible because the lambda calculus allows one to return functions as the result of applying a function. This paves the way for the developer to compose functions in meaningful and useful ways. Lambda expressions allow syntax to define functions not at the global scope, but as a value used in code. Efficient implementations of functional compilers would provide essential functions, such as addition or reading the head of a list, as built-in functions [1]. Yet, one may define their own functions through the use of lambda expressions, and indeed global-level function definitions in a functional program are expressed as lambdas during compilation. Last but not least, atomic values such as variables or constants are manipulated and passed around during function evaluation to produce a final computed result.

2.2. Basic Graph Reduction

The entirety of a lambda calculus program can be regarded as an expression which evaluates to some value, and computing such a value can be done through reducing the expression to a simpler value until it cannot be done anymore. The act of reducing an expression until it cannot be reduced anymore, in terms of functional programming, is called graph reduction [1]. After compilation to a lambda calculus form, the structure of the program can be observed as being a graph [1]. Computing the graph into a final result involves transforming parts of the graph into simplified results; hence, we say we perform reductions on the graph. Altogether, the execution of a functional program can be viewed as follows: the tree is walked until an expression is found that is reducible (i.e. a function object is applied to another expression), the reduction is performed on that node, and then the process is repeated until the tree cannot be reduced anymore.

A naive implementation of this algorithm would be a literal translation: the source code gets parsed into the lambda calculus, and then the tree structure is walked upon. In a particularly frequent case, the evaluator may encounter a lambda node being applied to some expression. By definition, a lambda body applied to an argument results in a copy of the lambda body, with all instances of the formal parameter of the function being replaced by the argument [1]. The copying is necessary, as the lambda body serves as a template that could be used many times within a program. Whenever a copy is requested, the entire graph of the internal expression must be walked, following all indirections and jumping around in memory.

Performing computations this way could be particularly expensive [1]. The bodies of lambda expressions could be unboundedly large, requiring a walk across each node every time a lambda is to be used (which could be quite frequent for many standard functions). As such, the basic forms of graph reduction are not a focus of this paper nor the project. Instead, there are several tactics to improve the performance of a functional program. One tactic in particular is half of the focus of [1], being supercombinator compilation; the book also discusses lazy evaluation, which will be briefly mentioned. Other strategies of optimization exist, and will be discussed briefly.

2.3. Efficient Graph Reduction

A fault of the graph reduction evaluator discussed is the overhead of copying trees every time a lambda is applied as a function. However, at compile time, Peyton Jones notes that there is enough information about the lambdas known at compile time such that we could describe a lambda body with a set of instructions that would reconstruct the body [1]. This completely eliminates the need to walk over every lambda body for copying, as that time is now spent at compilation time rather than runtime.

One issue with this is the presence of lambdas throughout the initial stage of compilation. The compiled main expression is littered with lambda definitions, some of which contain variables not bound by the lambdas (namely free variables). As Peyton Jones observes, lambdas containing free variables, when transcribed into a sequence of instructions, are not fit for compilation; they create a unique body for every value that the free variable could be assigned [1]. Thus, Peyton Jones proposes an important step beforehand: lambda-lifting with supercombinators. A supercombinator is simply a lambda definition without any references to free variables.

We perform lambda lifting through a visitor that tackles each unique form of AST node properly. To do so, the algorithm goes as follows: we search until we find a lambda with no other lambdas inside of it, then we represent any free variables as extra parameters to the function, extract the modified lambda and assign it some unique global name, and then in place of the lambda will

sit the supercombinator applied to any free variables referenced. This will repeat until all lambdas are retrieved from the main expression and separated into some list of supercombinator definitions.

With a list of supercombinator definitions, many opportunities for efficient compilation become available. First, the algorithm, though correct, may produce supercombinator definitions that are redundant (e.g. a supercombinator that accepts an argument, and simply applies another supercombinator on that argument). Luckily, given that we now have a list of all supercombinators used within the resulting program, it is trivial to iterate through them and eliminate redundant definitions. Another strategy now available to us is dead-code elimination. Since supercombinators are named, it is easy to walk through all supercombinator bodies and the main expression to determine which supercombinators are actually used and which are not. Any supercombinators that are not referenced can be safely omitted from the final compiled program, reducing program size. More importantly, however, is the fact that these supercombinator definitions can be compiled into instructions that reconstruct the body of the expression. This gives the major advantage of avoiding dynamically copying graphs at runtime, and it also allows one to avoid dynamically searching for the formal parameters, which would have been necessary in the naive implementation of graph reduction. Overall, Peyton Jones concludes his book with a virtual machine specialized for graph reduction [1]. The code for the machine, named G-Code by Peyton Jones, houses both the main runtime logic as well as the compiled supercombinator definitions in sequential form. We have aimed to use this as the basis for our final compilation in the implementation. Most tactics described here, including lambda lifting and optimization, are also used in our implementation.

2.4. Functional Optimizations

Functional programming languages have unique properties that allow opportunities for optimizations. A fundamental property of such languages is that of referential transparency – that an assigned variable can be replaced with its assigned value all throughout an expression without changing the final computed value [3]. In other words, assigned variables are effectively immutable, as the reassigning of values would break the referential transparency observed. This is in line with the values of functional programming; functions are composed together and pass along immutable data, instead of imperative programming where sequences of instructions modify some state. Compilers for functional programming languages can leverage this property by taking liberties in the behavior of the compiled code hidden from the user [3].

Memoization – the process of storing outputs of a function per given input – is one such strategy that can be applied to functional programming languages. If a given function is computationally expensive or is frequently invoked on recurring inputs, then memoization can be used as a tool to avoid excessive reinvocations through caching results [4]. A common example function is the Fibonacci function, which increasingly reuses previous calls to itself with recurring inputs. At what level of execution to perform memoization is not concrete; compiler implementers may choose to do so at the function level, instruction level, or even hardware level [4]. Memoization has been used in languages that support functional programming, such as Lisp and Haskell [4], but care must be taken to balance the overhead of looking up past computations with the potential speedup gained.

Parallelism is another technique that shows promise in the field of functional computing. In imperative languages, parallelism is usually not trivially achieved [5]. To achieve it, one must manually spawn threads and allocate processors to some computation. Parallel code also opens up a can of worms in that mutations and accesses of a variable cannot be reliably sequenced without specialized use of atomics to avoid race conditions [5]. With functional programming, all data is immutable, and functions can be computed independently of each other. This can enable compiler developers to leverage parallelism in functional code, with varying levels of success [5]. Some research has been done into performant automatic functional parallelization, but the benefits and drawbacks are not yet clear.

3. Implementation

To observe the techniques for efficient functional compilation and optimization, a new compiler was built for this project. We will discuss the design for the source language of the compiler, the implementation choices taken when crafting the compiler, and the results of the work.

3.1. Design

```
let foo: int [x: int, y: int] :=
    let a: int := 1,
    let b: int := 2,
    (a * x) + (b * y)
end

let main: int [] :=
    foo 3 7
end
```

Figure 2: The basic source language for the compiler.

When building a compiler, one needs a source language to parse and translate into some other useful code, which is often assembly or some machine code. The source code for this compiler is not of any existing functional language; it was drafted from scratch to support function definitions, variable bindings, and basic arithmetic. This decision was made to emphasize the implementation being standalone

and to allow for an understanding of why certain language choices are made. For example, the use of keywords, such as “let” and “end”, were embraced to simplify parsing in the implementation. For familiarity with popular languages such as C++ and Java, the top-level binding with the name “main” was designated as the entry point of the program. All types are explicit for the sake of clarity and for the sake of an easier type checking implementation. One particular goal of this source language was to preserve juxtaposition as a signal of function application, which is quite prevalent in popular functional languages. Fortunately, parsing such expressions did not prove to be too difficult.

3.2. Compiler Internals

The language of choice for the compiler implementation was C++, as it is a ubiquitous systems programming language that allows for concise yet descriptive compilation code. We chose to use C++17 as the standard of choice to allow for the usage of relatively modern C++ features for a more clean codebase. To fully observe all steps of the compilation process, absolutely no large compiler-development libraries were used. This meant that every step of the compilation process had to be spelled out and implemented sufficiently, which was not a trivial task. General libraries that allow for better UX or DX, such as libfmt and CLI11, were used to keep the compiler well rounded. Specifically, CLI11 was used to allow the user to specify command line arguments that help interrupt or spell out different phases of compilation. The library “libfmt” was used for robust formatted output for the program; notably, this was necessary as the equivalent option for formatted output in standard C++ was not available until C++23.

Any implementation of a novel programming language is met with the hurdles of lexical analysis and parsing. Lexical analysis involves reading text from a string and transforming it into a list of tokens corresponding to language syntactical structures, such as an identifier or an equals sign. Lexical analysis was implemented in the compiler through a state-machine scanner as demonstrated in [6]. Whilst not particularly noteworthy, the approach of making a finite-state machine for lexical analysis did prove to be a useful approach, and appeared to be more readable than a large chain of if-else blocks. Parsing a language involves reading a list of tokens and generating an equivalent tree representation of the entire program, which is usually referred to as an abstract syntax tree (AST). To parse such a simple language, a recursive descent parser was written to translate the sequence of tokens into a hierarchical abstract syntax tree. While parsing expressions, precedence climbing proved to be a useful algorithm. Normally, to implement associativity and precedence rules within expressions, one would have to specify almost arbitrary recursive functions which model grammar production rules that would allow for correct parsing. However, precedence climbing involves keeping track of the current expression precedence and cutting off parsing a subexpression when the precedence has been lowered. This results in correct parsing of binary expressions with varying precedence while not having to manage a large amount of recursive functions only for expression parsing; parsing of an expression can be achieved with a loop instead of many recursive functions.

After parsing the source language, we receive a syntax tree. As it is in a tree structure, we can get away with using smart pointers. Specifically, we can allocate each tree node within a unique_ptr as each node will naturally own their own children. This allows for effective memory management without intervention by the developer. Before proceeding further, we must perform semantic analysis to determine if the code provided was described correctly. We do this in two passes: one to build the symbol table, and the other to check for program correctness, which mainly entails type checking. The choice of a multi-pass compiler was intentional, as this allows for avoiding forward declarations. The symbol table pass throws away several symbol tables as it is initially only left with the top-level declarations, but this was acceptable for a small compiler implementation. After building the symbol table, basic type checking is performed on binary expressions and function applications.

After parsing the source language into a syntax tree, the compilation strategy typically entails numerous transformations through one or many intermediate representations [1]. First, we must translate the high level source code into an extended form of the lambda calculus. Then, we must perform transformations upon the lambda calculus until we are ready for lower-level intermediate representations. These transformations are upon trees mainly, and thus the visitor pattern proves to be quite useful in the final implementation. The visitor pattern encompasses a unique object which implements a visitor interface on top of the AST, and can recursively travel within the tree to perform a computation. This technique proved to be so useful that it is widely used within the backend of the implementation. When employing visitors, one must be careful when managing visitor state and intermediate values. Care was taken in implementing a “VisitValue” class that wraps a value computed within recursive visitation with a validity boolean. The purpose of this class was to keep track of values computed recursively, as the visit interfaces typically returned void. Using asserts with the custom class helped find bugs quickly and faster when using the visitor pattern, which helped when it ended up being quite prevalent across the codebase.

Using the visitor pattern above, we transform the AST into the enriched lambda calculus (ELC) intermediate representation. This representation extends the lambda calculus with a “let-rec” construct, which is a recursive list of bindings that are valid within a specified expression. This is the target model for our program, as each global definition can be represented as a binding within the let-rec, and the expression to be evaluated within main can be

slotted in as the expression where the bindings are valid. From here, we can perform transformations upon the lambda calculus. An especially important one for Peyton Jones' guide is lambda lifting, where lambdas that are fit to be supercombinators are lifted and extracted into a similar let-rec construct [1]. This allows for making lambdas compatible with the compilation strategy Peyton Jones aims for, so that each function can be compiled into a sequence of statements that builds the correct graph [1]. As mentioned prior, several optimizations can be performed here, such as eta-reduction and dead-code elimination [1]. The implementation performs eta-reduction as it will avoid creating supercombinators that are simply aliases of one another. Dead code elimination is not yet realized within the implementation, but it would be relatively trivial to do so.

After we compile the program into a list of supercombinators, we are ready to transform into our last intermediate representation: G-Code. The G-Machine was spelled out by Peyton Jones in [1], and it abstracts away the manipulation of graphs to simplify the final stage of code generation and to avoid jumping from the lambda calculus straight to assembly-level code. The G-Machine is a stack based machine that has many specialized instructions for handling the execution of a graph on the top of the stack. The translations of lambda calculus constructs to G-Machine instructions were followed through, and built-in functions, such as addition and subtraction, were specified through G-Code.

After getting G-Code instructions, we must find some way to translate these instructions into code that can run on modern machines. While assembly would have been a reasonable choice, the decision was settled on C, as it allowed for enough abstraction to easily convey the operational semantics of the G-Machine without being bothered by lower-level details such as register management. C is also a pretty handy choice as it gives quite a fine grain of control over the behavior of the resulting code, meaning our generated code will not perform any unnecessary allocations that were not explicitly specified. It is at this stage that implementation fell short, as Peyton Jones' G-Machine instructions involved saving entire G-Code sequences to runtime memory, which was not an intended feature of my C code generation. Figuring out how to preserve this functionality would result in a complete functional compiler pipeline, implemented from frontend to backend with an executable produced.

3.3. Results

The results of the compiler ended up in a nearly-full implementation of a functional compiler as guided by Peyton Jones in [1]. Every stage of the compilation process was implemented concretely in C++. With no libraries to assist in any compilation strategies past parsing, implementing the compiler from top to bottom was not an easy task. The compiler can accept a source file and, through various flags, can output the result of each phase of compilation from tokenization to G-Machine instructions.

4. Conclusion

Overall, functional programming languages represent an entirely different ideology with respect to computation. Originating from Church's lambda calculus, functional languages provide an alternative method for specifying computations in a declarative manner by composing functions. Being able to express high-level code elegantly and in a way that is expressly readable is a clear benefit to functional languages.

This study aimed to spell out key techniques that can optimize the execution of compiled functional programs. We have observed strategies in which high-level functional code can be executed on imperative machines. We note that the naive graph reduction approach is hopeless, and mention several different strategies for optimizing the compiled code. We have built a compiler from scratch to compile a self-designed subset of a functional programming language, using techniques and principles that have been studied and referenced.

One of the main challenges when completing this work was the lack of good references for functional programming language implementations. Peyton Jones' work, while massively helpful in understanding functional compilation, is dated and focused on lazy compilation which was not a requirement of the project. More resources in this area would have been greatly helpful, and could have assisted in exploring deeper optimizations. This research could be expanded by implementing further optimization techniques and analyzing how those affect more sophisticated programs.

4. References

[1] S.P. Jones, *The Implementation of Functional Programming Languages* (University College London: Prentice-Hall, 1987).

[2] J. Hopcroft, & J. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Reading, MA: Addison-Wesley, 1979).

[3] P. Hudak, Conception, evolution, and application of functional programming languages, *ACM Comput. Surv.*, 21(3), 1989, 359-411. https://doi.org/10.1145/72551.72554.

[4] A. Suresh, E. Rohou, & A. Seznec, Compile-time function memoization. Proc. 26th International Conf. on Compiler Construction, New York, NY, 2017, 45-54. https://doi.org/10.1145/3033019.3033024.

[5] J. Arora, S. Westrick, & U.A. Acar, Provably space-efficient parallel functional programming. Proc. ACM Program. Lang., New York, NY, 2021, 1-33. https://doi.org/10.1145/3434299.

[6] K.C. Louden, *Compiler Construction: Principles and Practice* (20 Park Plaza Boston, MA: PWS Publishing Co., 1997).

Regular Papers (Graduate Students)

EMERGING TRENDS IN LARGE LANGUAGE MODEL USE FOR UNDERGRADUATE INTRODUCTORY PROGRAMMING EDUCATION

Brian Kominick, Md Amiruzzaman, Linh B. Ngo
Computer Science Department, West Chester University
{bk105998,amiruzzaman,lngo}@wcupa.edu

ABSTRACT

Since the breakthrough release of ChatGPT in 2022, generative AI has undergone rapid investment and development, spanning many domains of consumer technology and introducing new concerns about the nature of human-computer interaction, particularly ways in which users ingest information. Notably, generative AI's potential for information retrieval and processing has already driven its adoption in education, particularly in the field of computer science, along with a surge of research on its pedagogical use, impacts, and challenges. With the pace at which this expansive field of research is growing, survey and review papers will provide a much-needed foundation for facilitating further cohesive, grounded analysis. Focusing on undergraduate computer science education, this study seeks to add new facets to the existing corpus of such works. Positioning itself among contemporary surveys and literature reviews, this paper engages in a survey of recent empirical studies on the use of generative AI in the classroom. By identifying patterns in methodological approaches, research questions, and student experiences, this paper surfaces underlying dynamics in student-LLM interactions. The result provides not only a new lens for understanding ongoing educational trends, their practical implementations, and their outcomes, but also actionable insights to support new empirical and conceptual developments.

KEY WORDS
AI, Education, Introductory Programming, HCI, LLM

1. Introduction

In just the past few years, large language models (LLMs) have rapidly become accessible tools for generating, modifying, and explaining code, and their use by students in introductory programming courses is now widespread. In response to the increasing call for the adoption of LLMs in professional and learning environments, educators and researchers have conducted studies on methods to effectively integrate LLMs into programming and computer science education, testing a range of approaches, including utilizing these tools as tutors, code generators, collaborative partners, and more. With this movement, a continually growing body of empirical work has emerged that examines the potential benefits and challenges of LLM use in early programming education. However, this pace of expansion carries with it fragmentation in study design, outcome measures, and assumptions on the role of LLMs in learning, making it difficult to draw coherent conclusions on their impacts and use cases.

While the demand for experience with AI grows increasingly common in post-graduate careers, the use of these technologies is also reshaping how students form their foundational knowledge of programming skills and concepts. Thus, educators face a new challenge in their responsibility to students in the shape of generative AI, of which the full extent of effects are still being studied. In an effort to provide a grounding for future research directions and pedagogical development, this survey paper examines recent literature with the following guiding questions.

- RQ1: What methods are being used to evaluate students' experiences with LLMs?
- RQ2: What trends in interactions and learning outcomes emerge from novice programmers' usage of LLMs?

Our contribution in this paper is a comprehensive review of empirical studies on the use of LLMs in introductory programming courses published between 2024 and 2025, encompassing the experiences of roughly 1539 students. Within the body of literature, it identifies cross-study trends and recurring challenges in addition to limitations of current research and questions raised or left unanswered. With consideration to the timeline of research and the level of experience with LLMs that introductory students possess, this paper characterizes the disparities in preexisting knowledge among novice programmers and attitudes they hold toward generative AI, along with reported learning experiences. Additionally, it highlights gaps and limitations felt by both students and instructors. Together, these analyses guide future work with direction for the scoping and design of subsequent empirical studies and targeted literature reviews.

2. Related Works

Existing literature reviews have examined generative AI in programming instruction at a high level, including a broad range of experience and education levels, from elementary students to experts. In 2025, a review surveying generative AI in computer science education from Reihanian et al. acknowledges the importance of studying impacts on introductory-level students [1]. Introductory courses represent a particularly sensitive context for such a change, since students are developing foundations for mental models and learning which tools and resources to turn to for help. In the same year, a systematic literature review from Agbo et al. states that a high concentration of published research occurs at the introductory level and that most papers focus on undergraduate students [2]. Similarly, Harrington et al. (2025) show in a systematic literature mapping that undergraduate education sees the highest output of research in this domain, although data from this paper does not specify the concentration of research at the introductory level in this context [3]. With respect to the rapid changes toward generative AI, the pace at which research is published, and the unknown long-term effects of its use in learning, an imperative arises to ensure proper grounding for new studies and maintain a dialogue with contemporary work through continued reviews.

As these reviews cover a large scale of papers, their analyses naturally highlight valuable high-level trends and research gaps in the broader field in relation to attributes like setting, research focus, intervention type, and evaluation methods. These studies and others, like a systematic review from Chang et al. in 2024, inform the scoping of this review with previously identified challenges and advantages in existing research, such as concerns around the accuracy of information generated and how to best assess outcomes, in addition to the potential for increased personalization for students and decreased workloads for instructors [4]. Operating with a smaller, focused corpus, this paper complements prior large-scale reviews by synthesizing how students' interactions with LLMs and their affective responses intersect within introductory programming contexts, surfacing commonalities and tensions lost in broader mappings.

3. Methodology

This paper adopts a structured approach to examine recent empirical studies on the use of LLMs in undergraduate introductory programming education. The goal of the review is not exhaustive coverage but rather scoping and synthesis around how LLMs are currently being deployed and studied in this introductory learning context and how students perceive and interact with them, identifying actionable insights that can inform future empirical work and reviews.

3.1 Search Strategy and Scope

An initial potential corpus of work was identified through a keyword-based search of the ACM Digital Library. The search was limited to ACM publications in order to maintain a focused scope, consistency in publication standards, and identify venues most commonly associated with computing education research. To complement this search, backward and forward snowballing were done on selected papers to identify additional relevant studies cited by or citing the initial results. The search process was terminated after ten iterations, at which point new papers no longer introduced substantively different study contexts or methodological approaches.

3.2 Inclusion and Exclusion

Studies were included if they reported on actual deployments of an LLM in introductory programming contexts, involved direct interactions between learners and generative-AI systems, and provided data and analyses related to student experience, learning outcomes, or instructional impact. Papers were excluded if they focused solely on tool design without deployment, speculative discussion without empirical evaluation, or contexts not classified by their authors as introductory.

3.3 Data Extraction

The final set of studies included in this review is summarized in Table 1, which outlines publication venues, participant counts, assessment types, and study durations. For each included study, a standardized data extraction process was applied to capture key characteristics for comparison, including research focus, study context and design, participant number and level of experience, assessment methods, reported outcomes, and challenges or limitations in experience or overall design. This process supported cross-study comparisons and the identification of recurring themes across heterogeneous study designs. The resulting trends were ascertained through the iterative comparison of these attributes. Attention was given to convergent conclusions and common assessment patterns, as well as points of divergence across studies and contexts. Observations included both supporting and complicating evidence to avoid overgeneralization of experience and outcomes.

4. Findings

Across the reviewed studies, several recurring themes emerged regarding how LLMs functioned in introductory programming contexts and how students experienced their use. While study designs and instructional goals varied, consistent patterns were observed in LLM performance and reliability, students' affective responses and perceptions, and the strategies students employed when interacting with LLM-based tools. Figure 1 provides a visual overview of

the findings, grouping them by study duration to highlight how this dimension can affect outcomes. The following subsections synthesize these trends across the surveyed literature.

4.1 LLM Performance as Tools in Introductory Programming Contexts

Across the reviewed studies, LLMs demonstrated a mixed and highly context-dependent level of performance when deployed as tools in introductory programming settings. While students frequently perceived LLMs as helpful for generating code, explanations, or practice materials, empirical evidence consistently highlighted limitations in reliability, correctness, and alignment with novice learners' expectations. These limitations were particularly salient given students' limited ability to detect errors, omissions, or misleading guidance in generated outputs [5, 6, 7].

Several studies reported that LLMs were able to produce syntactically correct and superficially plausible code, yet struggled with deeper semantic correctness, edge cases, or adherence to unstated instructional constraints. In studies where students relied on LLMs to generate solutions to programming tasks, incorrect or partially correct outputs were common and often required additional student intervention or instructor oversight to resolve. Non-deterministic behavior further complicated student use, as repeated prompts could yield substantially different solutions, leading to confusion and frustration among novice users who expected consistent behavior from computational tools [5, 7].

These challenges were particularly evident in studies examining prompt-based code generation. Nguyen et al. [7] found that beginning programmers frequently experienced difficulty interpreting why generated code failed, often attributing errors to misunderstanding by the model rather than deficiencies in their own prompts or task descriptions. Similarly, Prather et al. [5] and Rahe and Maalej [8] observed that students struggled to provide sufficient context or constraints in prompts, resulting in outputs that deviated from assignment requirements or instructional intent. In such cases, LLM performance limitations were closely intertwined with students' emerging mental models of programming and problem specification.

Studies that embedded LLMs within more structured instructional systems reported somewhat improved performance, though not without trade-offs. In CS50's AI-integrated environment, Liu et al. [9] reported that retrieval-augmented generation and persistent conversational context improved response quality relative to baseline models. However, students still encountered hallucinations, outdated information, and responses that exceeded the scope of course material, underscoring the need for careful system-level constraints. Similarly, systems designed to generate personalized programming exercises [10, 11] were generally effective in producing functional tasks, yet required expert validation to ensure alignment with learning objectives and to mitigate hidden constraints or unintended difficulty.

Throughout the reviewed literature, instructor involvement emerged as a critical factor in LLM performance and is requisite to their deployment, drawing into question exactly how much setup and maintenance is required by instructors. Tools that were explicitly framed as supplementary resources and paired with guidance about their limitations were less likely to be perceived as authoritative or error-free [12, 13]. In contrast, unrestricted access or interactions with weak scaffolding increased the risk that students would over-trust generated outputs, particularly when they lacked the conceptual grounding to independently verify results [8]. These findings suggest that LLM performance in introductory programming contexts cannot be evaluated solely in terms of output quality, but must be understood in relation to instructional design, scaffolding, and students' evolving technical literacy.

Taken together, the literature indicates that while LLMs are capable of solving traditional introductory-level coding tasks and can function as powerful supports for exploration, practice, and rapid feedback, their performance remains insufficiently reliable to serve as standalone instructional agents in introductory programming education. The effectiveness of LLMs as learning tools appears to depend less on raw model capability and more on how their limitations are surfaced, constrained, and contextualized for novice learners.

4.2 Student Interaction Strategies with LLMs

Across the reviewed studies, students adopted a range of strategies when interacting with LLMs in introductory programming contexts. These strategies varied in sophistication, intent, and alignment with course learning objectives, and were often shaped by prior programming experience, task structure, and instructor guidance or constraints [7, 5, 14, 8].

A commonly observed pattern involved direct answer-seeking, where students prompted LLMs to generate complete solutions or code fragments with minimal contextual framing. This strategy was more prevalent among novice students and in assignments emphasizing correctness over process. Several studies noted that while this approach could accelerate task completion, it frequently resulted in shallow engagement with underlying concepts and limited opportunities for debugging or conceptual transfer [7, 5, 14].

In contrast, when studies employed LLMs as interactive tutors or debugging aids, some students used iterative prompts to request explanations, step-by-step reasoning, or clarification of compiler errors. These interaction patterns more closely resembled traditional help-seeking behaviors, such as consulting documentation or attending office hours. Studies reporting this strategy often associated it with higher-quality learning outcomes or success rates, particularly when students actively evaluated and modified LLM-generated suggestions rather than adopting them ver-

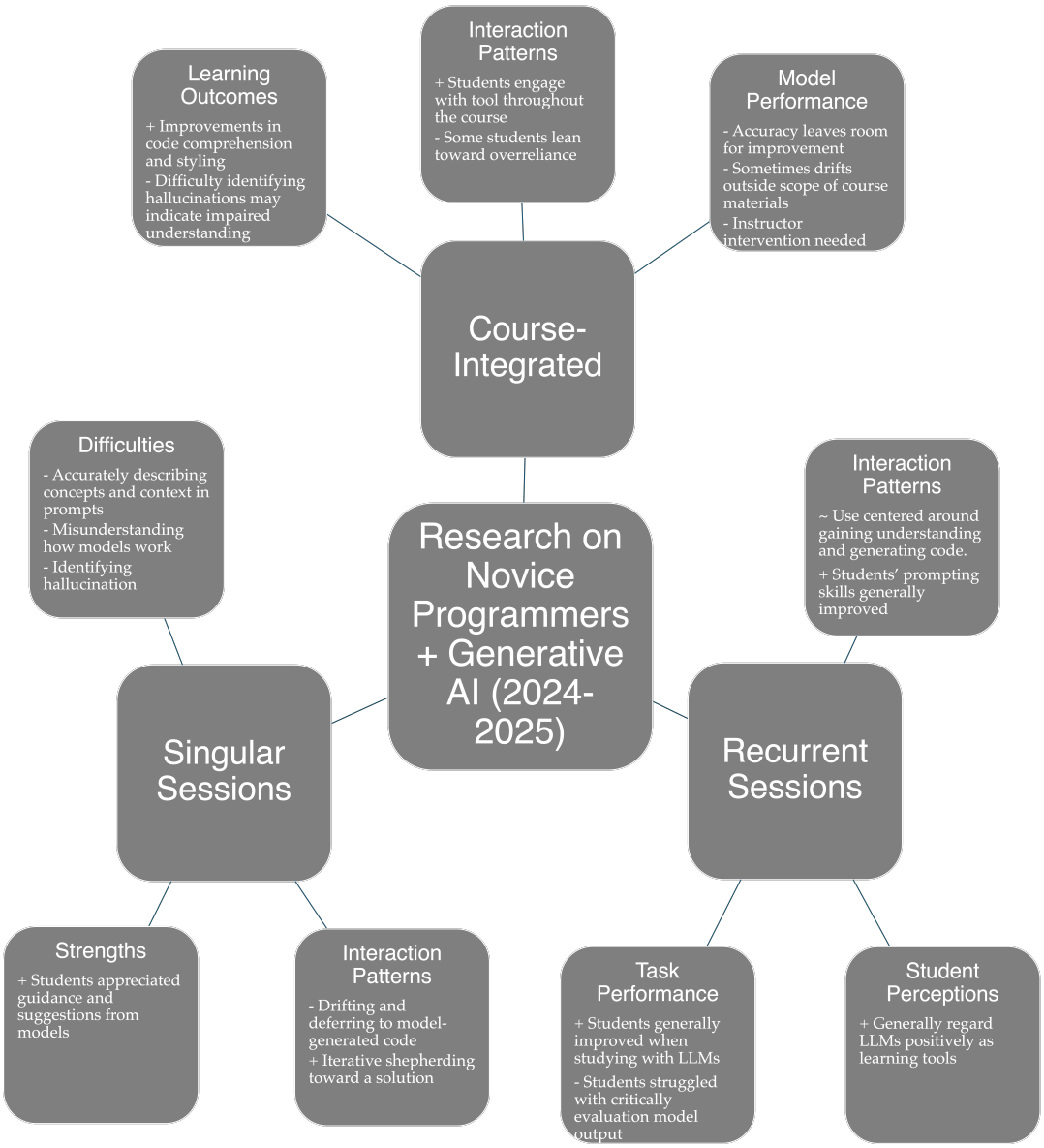


Figure 1: Findings from Survey

batim [12, 10, 9].

Another class of interaction strategies involved prompt refinement and scaffolding, where students experimented with varying levels of specificity, constraints, or examples in their prompts to influence model output. While this behavior was less frequently documented among beginners with no experience preceding an introductory level class, it appeared more often in cohorts with prior programming experience or progress in a course. Some studies suggest that prompting itself became a metacognitive activity, leading students to articulate their understanding of a problem more explicitly [5, 8, 13].

Conversely, a split emerged between students taking active and passive roles with LLMs during exercises. While some students continued down their own thought processes with assistance from AI-generated responses, others more heavily relied on suggestions at the end of the responses when deciding what to do next. This pattern bears some relationship with the behavior of copying complete solutions from an LLM [7, 14, 8].

Several papers also identified verification and comparison strategies, in which students cross-checked LLM outputs against their own solutions, instructor-provided examples, or other external resources. These strategies were more likely to emerge in courses that explicitly discussed limitations and potential failure modes of LLMs. When present, verification behaviors were associated with reduced dependence on AI-generated code and increased critical evaluation of correctness and style [10, 9, 13, 12].

Collectively, these observations suggest that student interaction strategies with LLMs are not monolithic but instead reflect a spectrum of behaviors influenced by expe-

rience level, instructional framing, and assessment design. However, across the surveyed work, interaction strategies were often inferred indirectly, through surveys or self-reported results, highlighting a need for more fine-grained observational and log-based analyses in future work.

4.3 Affective Impact and Student Perceptions

Students' emotional and perceptual responses to LLMs were a recurring theme across the reviewed studies, highlighting both opportunities and limitations in integrating generative AI into introductory programming education. Across contexts, student perceptions encompassed dimensions of comfort, motivation, engagement, trust, and perceived utility, with variation driven by the role of the LLM, interface design, and individual experience levels [12, 13, 6, 14, 8].

One consistent finding was that LLMs generally lowered barriers to asking questions and seeking assistance. In studies of AI-driven tutors such as Iris [12], 92% of students reported feeling comfortable asking questions without fear of judgment, and 62% indicated they felt safe asking questions they might avoid with a human tutor. Similar trends were observed in studies examining ChatGPT and Copilot, where students appreciated the low-stakes environment for trial-and-error experimentation, especially when encountering difficulty or uncertainty [6, 14, 8].

Engagement and motivation were moderately impacted by LLM use. For instance, the Iris study reported that 60% of students found interactions engaging, though motivation effects were more neutral, suggesting that while LLMs may enhance the interactive experience, they do not uniformly increase drive to complete exercises [12]. When directly applied to personalization, however, a majority of students indicated that they enjoyed the LLM-generated programming problems and considered them valuable for learning [11, 10]. In line with this, students often perceived LLMs as a complementary resource rather than a replacement for instructors, valuing accessibility and immediate feedback while still relying on traditional human support for conceptual understanding or high-stakes assessments [9, 13].

Students' perceptions of reliance and usefulness were nuanced. In several studies, a minority of students indicated that LLMs made exercises easier, with 43% of participants in the Iris study agreeing that tasks would be more challenging without AI assistance. Conversely, many students retained confidence in completing assessments without LLM support, suggesting that perceived reliance and utility may be task-dependent. When AI-generated outputs conflicted with students' understanding or were perceived as inconsistent, frustration or decreased trust emerged, emphasizing the importance of guided usage and explicit scaffolding [12, 6, 8, 15, 13].

A subset of studies highlighted concerns regarding over-reliance, equitable access, and risk awareness. Students occasionally depended on LLMs for solution generation without sufficient conceptual reflection, particularly when engagement was unrestricted [8, 7, 14]. Survey and observation data also suggested potential inequities in benefit, as students with minimal prior experience or lower engagement reported lower perceived value from LLM interactions. In addition, students demonstrated varying awareness of AI limitations, underscoring the need for pedagogical guidance on verification and critical evaluation of outputs [13, 10, 9].

Overall, the affective impact of LLMs in introductory programming education appears largely positive, especially in fostering comfort, low-stakes help-seeking, and engagement with exercises. However, benefits are tempered by variability in student experience, risk of developing superficial understandings and dependence, and the need for instructor-mediated guidance. Future work could explore strategies to maximize engagement and motivation while mitigating over-reliance and promoting metacognitive awareness of AI limitations.

5. Discussion

The findings of this survey suggest that the integration of LLMs into introductory programming education produces a complex dynamic between tool performance, students' interaction strategies with those tools, and their affective responses from engaging with them. Consistently, LLMs demonstrated variable reliability and correctness, which shaped the ways in which students approached problem-solving and engagement with exercises. Notably, inconsistencies in output quality in conjunction with diversity among students' prior experiences and mental models of both programming and LLMs often resulted in students adopting a spectrum of strategies, ranging from passive acceptance of AI-generated code to active iterative prompting, verification, and refinement.

These patterns have clear implications for instructional design. First, LLMs are most effective when embedded within structured learning environments that provide explicit scaffolding, guidance on prompt design, and discussions of model limitations. Second, the spectrum of observed interactions underscores the importance of metacognitive support and analysis: encouraging students to reflect on AI outputs, compare them with personal reasoning, and validate with other resources can reduce over-reliance and improve depth of understanding. Third, affective responses indicate that AI can lower barriers to help-seeking and promote engagement, but these benefits are not uniformly distributed across all learners.

Extending previous literature, this survey uniquely highlights the presence of model performance, students' behavior, and their resulting affect as key dimensions within the context of LLMs in introductory computer science. Whereas earlier reviews primarily examined adoption patterns or high-level benefits of generative AI [2, 1, 3, 4], this work explicitly links observed student strategies to both tool limitations and emotional responses, providing a more nuanced understanding of how LLMs function as educa-

tional resources. Overall, the evidence reinforces the position that LLMs hold significant potential as supplementary tools, but their effectiveness is contingent on careful integration, instructor oversight, and support for critical engagement.

6. Limitations and Future Work

The rapid development of LLMs and related technology imposes several limitations. Due to the number of studies in this field that have reached publication, in-depth surveys may not capture all relevant studies or recent developments in LLM technology. Additionally, some of the included studies focused on short-term, limited interventions and featured small cohorts, limiting the ability to generalize findings across institutions or student populations. Moreover, most studies relied on surveys and self-reported data rather than detailed log analysis or longitudinal tracking. In combination with short durations and limited participant pools, this reliance restricts insights into long-term learning outcomes and strategy evolution. Finally, given the rapid advancement of LLMs, findings may be specific to the capabilities of the models studied (ChatGPT/GPT-3.5-4) and may not be fully applicable to future iterations.

Future research should address these gaps through longitudinal and multi-institutional studies that examine how student strategies, performance, and affective responses evolve over time. Controlled experiments comparing different interaction paradigms—such as AI tutors, code generators, or debugging assistants, can clarify which applications produce the most effective learning outcomes. Additionally, interface and instructional designs that explicitly incorporate verification and critical engagement should be explored to maximize learning while minimizing overreliance. Equity-focused investigations are also needed to understand how prior programming and LLM experience, access, and engagement shape students' benefits from LLMs. Finally, continuous evaluation of newer models and deployment contexts will ensure that insights remain relevant as generative AI continues to evolve.

7. Conclusion

This survey provides a focused synthesis of empirical studies on the use of LLMs in introductory programming education, examining tool performance, student interaction strategies, and affective responses. Throughout the surveyed literature, LLMs were found to support experimentation, rapid feedback, and low-stakes help-seeking, but exhibited inconsistent reliability, leading students to develop a spectrum of strategies from passive acceptance to active iterative prompting and verification. Affective outcomes were generally positive or neutral along the lines of comfort and engagement, though trust and perceived utility varied depending on context.

The unique contribution of this work lies in linking performance, behavioral, and affective dimensions within the introductory programming education environment, providing actionable insights for educators, instructional designers, and HCI researchers. LLMs hold significant potential as supplementary educational tools, but their effective integration requires thoughtful design, guidance on AI limitations, and support for metacognitive reflection and development. By clarifying challenges and opportunities, this review lays a foundation for future empirical work that can guide the responsible adoption of generative AI in programming education.

References

[1] Iman Reihanian, Yunfei Hou, Yu Chen, and Yifei Zheng. A Review of Generative AI in Computer Science Education: Challenges and Opportunities in Accuracy, Authenticity, and Assessment, June 2025.

[2] Friday Joseph Agbo, Chris Olivia, Godsalvation Oguibe, Ismaila Temitayo Sanusi, and Godwin Sani. Computing education using generative artificial intelligence tools: A systematic literature review. *Computers and Education Open*, 9:100266, December 2025.

[3] Brian Harrington. A Systematic Literature Mapping of Early Generative AI Research in CS Education.

[4] Chi In Chang, Wan Chong Choi, and Iek Chong Choi. A Systematic Literature Review of the Opportunities and Advantages for AIGC (OpenAI ChatGPT, Copilot, Codex) in Programming Course. In *Proceedings of the 2024 7th International Conference on Big Data and Education*, pages 29–35, Oxford United Kingdom, September 2024. ACM.

[5] James Prather, Paul Denny, Juho Leinonen, David H. Smith IV, Brent N. Reeves, Stephen MacNeil, Brett A. Becker, Andrew Luxton-Reilly, Thezyrie Amarouche, and Bailey Kimmel. Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models.

[6] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. 31(1):1–31.

[7] Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. How Beginning Programmers and Code LLMs (Mis)read Each Other. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–26. ACM.

[8] Christian Rahe and Walid Maalej. How Do Programming Students Use Generative AI? 2:978–1000.

[9] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 750–756. ACM.

[10] Sven Jacobs, Henning Peters, Steffen Jaschke, and Natalie Kiesler. Unlimited Practice Opportunities: Automated Generation of Comprehensive, Personalized Programming Tasks. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 319–325. ACM.

[11] Evanfiya Logacheva, Arto Hellas, James Prather, Sami Sarsa, and Juho Leinonen. Evaluating Contextually Personalized Programming Exercises Created with Generative AI. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, pages 95–113. ACM.

[12] Patrick Bassner, Eduard Frankford, and Stephan Krusche. Iris: An AI-Driven Virtual Tutor for Computer Science Education. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pages 394–400. ACM.

[13] Jacob Penney, Pawan Acharya, Peter Hilbert, Priyanka Parekh, Anita Sarma, Igor Steinmacher, and Marco Gerosa. Understanding Programming Students' Help-Seeking Preferences in the Era of Generative AI. In *Proceedings of the ACM Global on Computing Education Conference 2025 Vol 1*, pages 15–21. ACM.

[14] Andreas Scholl and Natalie Kiesler. How Novice Programmers Use and Experience ChatGPT when Solving Programming Exercises in an Introductory Course.

[15] Andreas Scholl, Daniel Schiffner, and Natalie Kiesler. Analyzing Chat Protocols of Novice Programmers Solving Introductory Programming Tasks with ChatGPT.

Paper	Source	Students	Assessment Type	Duration
Nguyen et al. (2024) – How Novices and LLMs (Mis)read Each Other [7]	ACM Conference on Human Factors in Computing Systems	120	Code generation	Single session
Prather et al. (2024) – Interactions with Prompt Problems [5]	ACM Conference on Human Factors in Computing Systems	58 (Pilot) 202 (Follow-up)	Code generation	Two lab sessions
Prather et al. (2024) – Usability of GitHub Copilot for Novices [6]	ACM Transactions on Computer-Human Interaction	19	Programming exercise	30 minutes
Jacobs et al. (2025) – Unlimited Practice Opportunities [10]	ACM Innovation and Technology in Computer Science Education	26	Solving personalized tasks	Tutorial session
Scholl et al. (2024) – Novice Use of ChatGPT [15]	IEEE ASEE Frontiers in Education Conference	298	Programming exercises	Two weeks
Logacheva et al. (2024) - Evaluating Personalized Programming Exercises [11]	ACM Conference on International Computing Education Research	68	Programming Exercises	Embedded in First 3 Chapters of Course
Bassner et al. (2024) - AI-driven Virtual Tutor [12]	ACM Innovation and Technology in Computer Science Education	121	Programming Exercises	Embedded Throughout Course
Liu et al. (2024) - Teaching CS50 with AI [9]	ACM Technical Symposium on Computer Science Education	70 (Summer) 500 (Fall)	Course with AI Assistance	Embedded Throughout Course
Penney et al. (2025) - Understanding Students' Help Seeking Preferences [13]	ACM Global Computing Education Conference	20	Two 5-Minute Assessments	20 Minutes (Including 10-Minute Tutoring Session)
Rahe and Maalej (2025) - How Do Programming Students Use Generative AI? [8]	ACM Software Engineering	37	Programming Exercise	Task Completion or $\geq$ 20 Minutes

Table 1: Overview of empirical studies included in this survey (2024-2025)

Regular Papers (Faculty)

ENSURING PERSISTENCE OF TIMELESS FREEMASON VIRTUES IN THE AI AGE

Reiley Walther
Kutztown University of Pennsylvania
rwalther@kutztown.edu

ABSTRACT
Ethics and virtues in many subjects can be difficult to exhaustively define. Freemasonry and Artificial Intelligence (AI) are both built on relevant ethical frameworks that aim to govern each of these subjects accordingly. Both Freemasonry and AI have well-documented standards, constitutions, rules, regulations, and edicts that define these frameworks. An investigation was therefore conducted into how these two seemingly disparate subjects' frameworks intersect, and how those items within the framework of the timeless and ancient Freemasonry craft can be identified in the advanced and contemporary world of AI. A thorough analysis and comparison of the principles of both Freemasonry and AI took place through samples of work. Results revealed that there is a lot more of an intersection between Freemasonry principles in modern-day life than just that of AI. Since Freemasonry was not established during the Industrial Age, let alone the Digital Revolution, the transition into modern AI contexts require meticulous, hermeneutical care. These findings suggest that their principles can be viewed as interchangeable. This ethical continuity between the 'ancient craft' and modern AI systems further suggests that Freemason principles are not historical artifacts, but rather persistent virtues that remain applicable across this evolving sociotechnical landscape of the past, present, and future.

KEY WORDS
Freemasonry, Artificial Intelligence, Ethics, Values, Principles, Intersection

1. Introduction

Artificial Intelligence systems are evolving rapidly and are being integrated largely into as many aspects of modern life as possible, including sectors such as healthcare, transportation, finance, energy, insurance, manufacturing, pharmaceuticals, and retail according to [1]. As this evolution and autonomy continues, there are great concerns surrounding the ethical use of AI systems. Important questions have been raised about bias and data privacy in AI systems [2] regarding how they should be designed, deployed, and governed. These questions and concerns uncover a larger challenge of ensuring that these AI systems are operating in ways that align with the well-being of society and its expectations, and arguably certain human values.

Many frameworks have been proposed, developed, modified, and implemented that aim to guide the responsible development of AI systems. Some of these frameworks incorporate the basic principles of fairness, transparency, and trustworthiness, but these focuses are on contemporary technological concerns without the consideration of much older, more traditional ethical questions. Including these questions may provide further insight into the responsibilities associated with artificially intelligent systems.

Freemasonry represents an organization with such traditional questions that emphasize the potential missing piece for these guiding frameworks. Through allegory and symbolic instruction drawn from the skills of the operative stonemasons, Freemasonry promotes moral principles seeking to guide individuals toward ethically acceptable behavior. Freemasonry is built upon three tenets: Brotherly Love, Relief, and Truth. These emphasize equality, honesty, discipline, and the pursuit of knowledge. These principles are further supported by the tools in Freemasonry, namely, the square, the compasses, the level, and the plumb. These tools reinforce the lessons concerned with balance, integrity, and fairness.

The purpose of this paper is to examine how these tenets, principles, and ancient Landmarks provide a significant and meaningful perspective for interpreting modern AI system ethics. Through the examination of the parallels between the tenets of Freemasonry and the concerns in AI ethics, this paper proposes a conceptual link between traditional philosophy and modern AI ethical frameworks. The contribution of this paper is, first, to demonstrate that many ethical challenges in AI do reflect the moral questions that continue to be addressed by philosophers for centuries, and second, to provide a structured comparison of the principles of Freemasonry and AI ethics. The latter contribution highlights how the symbolic teachings may reinforce the efforts to improve upon the development of AI systems responsibly.

2. Body of Paper

2.1 Artificial Intelligence

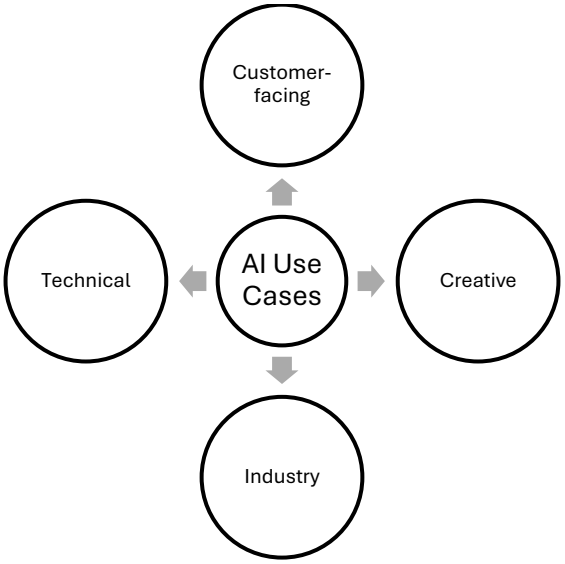
Artificial Intelligence, otherwise more commonly known as AI, has gained an overwhelming amount of momentum since the dawn of deep learning in 2011 according to [2], the presentation of generative adversarial networks (GANs) in 2014 mentioned in [3], and the introduction of transformer architecture in 2017 from [4].

[2] and [5] define AI as systems that perform tasks humans normally associate with human intelligence by thinking and acting rationally. [2] believes that this rationality can be combined with human behavior and divided into four arrangements: thinking humanly, acting humanly, thinking rationally, and acting rationally. "It is impossible to anticipate all the ways in which a machine pursuing a fixed objective might misbehave." With this quote by [2], it is clear that achieving all four combinations is a challenging task, as each one will have specific goals and aims.

AI is a blanketing term for the many different subfields associated with it, including, but not limited to, natural language processing (NLP), robotics, computer vision, and generative AI. With such a wide range of subfields pertaining to AI, it is no surprise that a wide range of tasks can be performed with tools that incorporate AI. IBM partitions AI use cases into four categories in [1] (see figure 1): customer-facing AI use cases, creative AI use cases, industry AI use cases, and technical AI use cases.

Customer-facing AI use cases show the interactions between customers/clients, and AI-integrated products including customer service, customer experience, cross-selling and up-selling, personal assistance, and streamlining human resources tasks. Creative AI use cases go beyond the text-AI crossover, and include recognition, creation and generations of text, images, videos, sounds, code, and other visual inputs through generative AI and computer vision. Industry AI use cases show the different sectors that AI has reached including automotive, education, finance, healthcare, insurance, manufacturing, pharmaceuticals, transportation, and retail. Finally, technical AI use cases highlight the streamlining of operations and the under-the-hood tasks aiming to boost performance, automate tasks, strengthen security, effectively equip internal departments, and predictive analytics.

Figure 1: IBM AI use cases



With all these sectors and all these companies adopting AI tools for the aforementioned purposes, innovation and evolution, with the use of these tools, has the potential to shift into unethical and unlawful regions. Certain potential dilemmas may call for experts who are not only well-versed in ethical theories, but who are also capable of applying them to the relevant fields, thereby establishing each of these specific, professional fields as a now interdisciplinary field.

2.2 AI Ethics

Society now calls for stronger standards from an ethical perspective to guide the overt technological revolution driven by AI technologies and tools according to [6]. The challenges of developing, deploying, and maintaining AI tools can be addressed by beginning this process immediately with AI ethics in mind. Before deciding upon what basis AI ethics should be established, it is important to define ethics itself.

In [7], George Reynolds defines ethics as a group-defined code of behavior for what is acceptable and unacceptable to which an individual belongs. It is also associated with moral philosophy, in other words, concerned with what people should and should not do. Ethics extend beyond the morals of a single person, since morals can be defined as *personal* principles one uses to determine what is right and what is wrong [7]. Standford University conducted a study on AI called the One Hundred Year Study on AI (also known as AI100). In their 2018 and 2019 reports, under 'Sentiment' mentioned in [2], it was found the most common issues were ethical issues, particularly algorithm bias and data privacy. Thus, by deduction, the aim of AI ethics should then be to improve data privacy, mitigate algorithm bias, and ensure transparency throughout the process.

In [8], bias is defined as the unfair favoring of, or opposition to, a person or group over others, often resulting in disadvantages to those unfavored people or groups. If the aim is to mitigate bias, then it is to mitigate this unfair favoring. In other words, the aim is to enhance unbiasedness in AI. Unbiasedness is synonymous with fairness, and according to [2], fairness can be partitioned into six common domains: individual fairness, group fairness, fairness through unawareness, equal outcome, equal opportunity, and equal impact.

Individual fairness narrows the idea of fairness down to a single unit, being that fairness applies to individuals. This is the idea that, regardless of class, all individuals are treated in an analogous way. Inflating the population size, group fairness encompasses more than one class and how they are each treated. Fairness through unawareness is an implied fair treatment based on the removal of certain identifiable attributes. Equal outcomes, as the name suggests, looks to produce the same results regardless of class. Equal opportunity looks at the balance of this idea of fairness, potentially leading to unequal outcomes even though bias is being ignored due to the way training data may be processed. Finally, equal impact considers both the benefits and the costs of true and false predictions, respectively. This goes beyond equal opportunities since the consideration of both results is a realistic acknowledgment of many processes from the same training data.

The fourth Amendment of the United States Constitution protects American citizens when there is a "reasonable expectation of privacy" according to [7]. Privacy in AI systems is a major concern since AI systems use enormous amounts of data in training and improving models. With the versatility of AI systems, particularly that of user input seen mostly in AI chatbots, users are not limited to the ways in which information can be placed into these systems. If the information that is entered into AI systems is not handled responsibly, the potential for misusing the information increases, and this could cause severe consequences if the information is sensitive. This can be confidential or private information, for example, trade secrets of a company, or personally identifiable information (PII) respectively.

Correspondingly, companies and AI systems gathering information must handle all incoming information responsibly. If this fails, the integrity of the information could be compromised or leaked, exposing information, including any sensitive information, to unauthorized parties. This introduces a large element of trust into our ethical concerns. Two ways to ensure AI systems are, in fact, safe, fair, and private, is to go through a verification and validation process [2]. To verify an item means to have an AI system meeting specifications, while validation means to ensure that the specifications meet the needs of the users. Another aspect of trust is transparency.

It is far easier for users to place their trust in things when said things are transparent, or extremely convenient. AI systems have the potential to be both. The former is seen through privacy policies, how certain systems assist users, and how they work with a user to achieve their goals. The latter can be seen when intellectual property is concerned since many aspects pertaining to the success of these revolutionary AI tools for a company may be kept hidden from consumers. With both views present, AI system developers must balance these items so that users are seen wanting to use the system, while the company simultaneously does not compromise the functionality of the system that contributes to its success.

Ideally, each of the fairness domains, as well as privacy and trust, would all be accounted for which would alleviate the bias in an AI system, however, this is not possible, particularly concerning equal opportunity fairness and having a well-calibrated system [2]. A well-calibrated system is one in which all individuals are given similar results regardless of external classes. This then raises the question on how systems can be built to include fairness. The answer to this question would be to implement best practice techniques while designing and implementing the systems. Some best practices may include considering fairness (and other essential AI ethical items) from the beginning, as mentioned before, as well as, including a wide range of testing methods of the system to accommodate smaller minority groups, explicitly defining the classes and groups that the system will accommodate, and tracking the metrics of the system in general, and that of the different groups in particular. Regardless of the number of best practices implemented into these systems, all training data in AI systems originate from some source, and since all sources cannot and do not contain exclusively objective data, no AI system is completely impartial and unbiased.

2.3 Freemasonry

Freemasonry, or more simply, Masonry, is the oldest and most popular fraternal organization in the world. This fraternity of gentlemen is bound together by obligations and their words of integrity [9]. Freemasonry beliefs can be narrowed down to three tenets, namely, brotherly love, relief, and truth. Brotherly love is the respect, camaraderie and love for each other as well as all of mankind. Relief is providing charity to others, especially those in need, as well as the mutual aid and assistance for fellow Freemasons, or Masons. Truth is the continuous search for knowledge, often for the answers to many philosophical and universal questions of morality, which can include salvation of one's soul that only one's faith and relationship with their belief in their Supreme Being can provide. An extremely popular saying and basic idea of this fraternity is to simply "make good men better."

In addition to the three tenets of Freemasonry, the fraternity also strives to improve on education, social responsibility, political neutrality, and equality among all its members.

The education associated with Masonry focuses on a system of morality, and brotherhood of its members under the fatherhood of a Supreme Being. The teachings use symbols and presentations through allegory and interpretation. With these teachings, it falls upon each member of the fraternity to become well-versed in the knowledge presented to them as far as they see fit to learn and understand. Unlike grade schools and university programs, there is no definitive milestone every Freemason must reach in order to be able to call themselves a 'Freemason' besides the basic knowledge needed for a new member to advance through the three rituals, or degrees, when they first become a Freemason. Simply put, these three degrees each play an important role in the allegory of a man becoming a Mason further skilled in his work, which relate to that of the stonemason guilds, where a stonemason took time to learn the skills of the craft and then advanced to a new skill level.

Freemasonry charges its members to be true, loyal, and law-abiding citizens in the country that they live in. Members owe their allegiance to their country and are to be obedient to the laws of the state that they may reside in at any time. With that being said, Masonry also rejects any suppression of basic human rights, any suppression of human dignity and any suppression of free exercise of religion, as well as rejecting dictatorship and tyranny [9].

Politics and religion are two topics that are forbidden to be discussed in Masonic meetings. These topics are, and have often been, the root of personal conflicts and disagreements in a Masonic meeting place, or lodge. Any conflicts or disagreements suffered between members defiles brotherly love, which is one of the three tenets of the fraternity.

To ensure equality of its members, Masonry disregards all monies and valuables that could be seen as superior wealth, status, or outward appearance compared to others. For centuries, Freemasons have come from many different walks of life, through many career paths, and different socioeconomic backgrounds, including, but not limited to, kings, princes, factory workers, lawyers, gardeners, doctors, and fast-food employees. A member's background, wealth and line of work is not what makes them a Freemason, rather their individual judgement and uprightness of character.

There are certain requirements that need to be met in order for a person to become a Freemason [10]. A person must be a man, seeking to join the fraternity of his own free will and accord, be at least 18 years of age, highly vouched for by at least two existing Freemasons, and have a belief in a Supreme Being. This term "Supreme Being," having been mentioned already, has been phrased this way for a significant purpose. Freemasonry does not limit membership to a single religion, but it does require a religion that does believe that there is a singular higher power. This is where this requirement ends. After this requirement, a man's personal beliefs remain as the name suggests: personal, and no religious precept is forced upon anyone hereafter.

Masonry has concrete, written documentation dating back to the establishment of the first Grand Lodge of England in 1717, and other records suggest that they date back even further [9]. A Grand Lodge is the governing body of a particular jurisdiction under whose authority all other subordinate lodges in that jurisdiction preside. Many countries around the world have a Grand Lodge governing the jurisdiction of the entire country, but in the United States, each state is governed by its own Grand Lodge.

The narratological origins of Freemasonry trace back to the building of the Temple of King Solomon in Jerusalem around 1000 B.C. The more accepted and historic origins date back to the stonemason guilds that formed in Europe during the Middle Ages [9]. The earliest English records claim that a guild of Masons was organized in York, England in 926 A.D. by Athelstan, the first King of Saxons and Angles at Kingston-upon-Thames [11]. As this fraternity was originally a craft guild, Freemasonry is often referred to as the *Craft.* During the 18th century, Masonry underwent a progressive change regarding admittance into the fraternity. Before this time, the members were referred to as operative Masons [12], since they truly did work with the tools to erect structures. As the 18th century progressed, more "gentlemen" members were being accepted into the fraternity. Since they had no prior operative knowledge of the Craft, but had been admitted to the fraternity, they became known as speculative Masons, and they adopted many of the working tools of the Craft, including the square and the compasses, which remain the identifying symbol of Freemasons today (see figure 2).

Figure 2: Basic Freemasonry Logo

2.4 Freemasonry Ethics

Unlike the ethics associated with AI, society is not actively calling for stronger standards where Freemasonry ethics is concerned. Instead, [12] says Freemasonry ethics have existed from time immemorial and have been compared to "natural law" in that they have never been legislated into existence but rather discovered and then adopted. These ethics also cannot be created nor annihilated. Almost all Grand Lodges around the world aim to observe and follow what are known as "Ancient Landmarks." Unlike the modern meaning, a Landmark in Freemasonry, defined by [12], is said to be "a distinctive characteristic of the fraternity established at a very early date in its history and, in a measure, set the Order apart and separates it from all other societies." A sum of these characteristics constitutes what is referred to as the "Body of Masonry."

In addition to the three tenets of Freemasonry, Dr. Albert G. Mackey, a 17th century doctor, an illustrious Masonic authority, and an author best known for his books and articles about Freemasonry, compiled a list of twenty-five distinct characteristics which, in his opinion, constitute the "Body of Masonry." The characteristics he defined are as follows:

I. The modes of recognition, for instance, the signs, grips and words
II. The division of Symbolic Masonry into three degrees.
III. The legend of the third degree.
IV. The government of the fraternity by a presiding officer, called a Grand Master, elected from the body of the Craft.
V. The prerogative of the Grand Master to preside over every assembly of the Craft wheresoever and whensoever held.
VI. The prerogative of the Grand Master to grant dispensations for conferring degrees at irregular times.
VII. The prerogative of the Grand Master to grant dispensations for opening and holding lodges.
VIII. The prerogative of the Grand Master to make Masons at sight.
IX. The necessity for Masons to congregate in lodges.
X. The government of every lodge by a Master and two Wardens.
XI. The necessity for every lodge, when congregated, to be duly tiled.
XII. The right of every Mason to be represented in all general meetings of the Craft and to instruct his representative.
XIII. The right of every Mason to appeal from the decision of his brethren in lodge convened to the Grand Lodge or general assembly of Masons.
XIV. The right of every Mason to visit and sit in any regular lodge.
XV. That no Mason, not known to some brother present as such, can enter a lodge without undergoing an examination.
XVI. That no lodge can interfere with the business or labors of any other lodge.
XVII. That every Freemason is amenable to the laws and regulations of the Masonic jurisdiction in which he resides.
XVIII. That every candidate for initiation must be a man, freeborn, and of lawful age.
XIX. That every Mason must believe in the existence of God, or the Supreme Architect of the Universe.
XX. That every Mason must believe in a resurrection to a future life.
XXI. That a Book of the Law of God must form an indispensable part of the furniture of every lodge.
XXII. That all men in the sight of God are equal and meet in lodge on one common level.
XXIII. That Freemasonry is a secret society, in possession of secrets that cannot be divulged.
XXIV. That Freemasonry consists of a speculative society founded upon an operative art.
XXV. That these Landmarks can never be changed.

Upon analyzing the "Body of Masonry," many of the Landmarks of the fraternity originate from, and show similarity to, the operative stonemason guilds. Additionally, many of the Landmarks can be seen in many places in everyday life today through interpretation and symbolism. [9] defines a symbol as "an object or design or other material object that stands in for something abstract or even invisible." Freemasonry, its tenets, Landmarks, and teachings are abundant with symbolism, with some symbols having more relevance to modern life in general, and to AI in particular. Some of these relevant symbols in [9] and [12] that seem disparate but are applicable to AI today would be The Number Three, King Solomon's temple, Square and Compasses, The 47th Problem of Euclid (the Pythagorean Theorem), the Level, and the Plumb.

The number three has been a significant number for thousands of years both within and outside of Freemasonry. Often, this number is associated with completeness. In Freemasonry, which is also applicable to life in general, the number 'three' symbolizes the ongoing search for perfection. There are also three degrees, three elected line officers in a lodge, and three principal tenets in Freemasonry. Aristotle believed that the number three symbolized the Supreme Being since the number three contains the first two numbers (through addition), and it implies a beginning, a middle, and an end. Other interpretations include the three stages of life – youth, adulthood, maturity, in some cultures it represents the father, mother, and child, and in Christianity, it represents the Father, the Son, and the Holy Spirit.

King Solomon's Temple symbolizes the individual Mason and the great journey he travels to build a suitable temple (being his own body) worthy for God to inhabit. This is similar to how the ancient operative Masons erected King Solomon's Temple, as such a place for God to inhabit took time to plan, prepare and complete (see figure 3). This was

also the resting place of the Ark of the Covenant. The goal of a builder is perfecting the workmanship, often overtime. At the time of the Temple's completion, many viewers came from far places to gaze upon the magnificent structure. When it was destroyed, the memory of the Temple in all its awe and glory remained. Like the Temple, as speculative Masons construct their personal temple over time, Masons strive for perfection in building themselves too. After a Mason has passed on, the memory of them lives on through their loved ones, just as the memory of the Temple lives on.

Figure 3: King Solomon's Temple Depiction

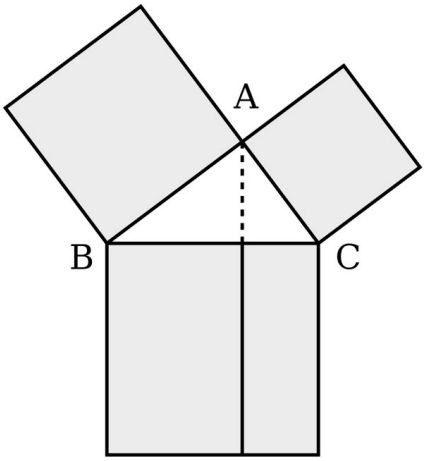
The Square and Compasses, which represent the Masonic symbol to this day, accompanied by the letter 'G,' are two working tools in operative Masonry. The square (see figure 4) is known for builders to square their work correctly, but Masonic teachings reveal that it is used symbolically to uphold the ideas of equality by governing all actions correctly towards others. It is also the jewel worn by the leader, or Worshipful Master, of a lodge. This elected officer presides over the lodge, like a chairperson, or president. They are elected to serve for a full year beginning on December 27th each year at noon, otherwise known as St. John the Evangelist's Day. The compasses (see figure 5), often used in its plural version in Masonry, allows operative Masons to draw parallels and perfect circles. Masonic teachings show that a man represents one point and those within his life are the other point which is drawn around him. The Freemason is to apply the tenets of brotherly love, relief and truth to those with whom he comes into contact throughout his life.

Figure 4: A Simple Square Tool in Masonry

Figure 5: Simple Compasses in Masonry

The 47th Problem of Euclid (see figure 6), or Pythagorean Theorem, is a popular equation today in mathematics, and the most useful equation in the building trade. It can be used for navigation, astronomy, and determining if rooms and foundations were square. It also allows the construction of a perfect right-angled triangle. In other words, it "squares your square when it loses its square." In Freemasonry, it teaches realignment to the virtues taught within the fraternity and it is often associated with a Mason who has served as the Worshipful Master of a lodge. Once the Worshipful Master's one-year term ends, this Mason is then a Past Master. The term "Worshipful" has no relation to being worshipped. It is an old English honorific that means "respected." These Past Masters aim to realign their way within the lodge and amongst other members after serving as the Worshipful Master and impart their knowledge to other Masons.

Figure 6: Euclid's 47th Problem Diagram



The Level (see figure 7) is a tool used by operative Masons similar to the plumb, but this tool measures horizontals instead of perpendiculars. Speculative Masons are taught that they are to meet all fellow Masons, and others in life, on the same level, reenforcing the idea of equality. The level is the jewel worn by one of the three elected officers in the lodge. It is worn by the Senior Warden. In the absence of the Worshipful Master, the care of the lodge falls to this elected officer's station.

Figure 7: The Level Tool in Masonry

The plumb (see figure 8) is a tool used by operative Masons to test if a wall or surface is level, often by measuring perpendiculars. Speculative Masons are taught to use the plumb to symbolize an uprightness of behavior when collaborating with others in life, as well as while fulfilling their duty to God. The plumb is also a jewel worn by one of the three elected officers in the lodge. It is worn by the Junior Warden. In the absence of the Worshipful Master and the Senior Warden, the care of the lodge falls to this elected officer's station. This symbol of rectitude within the plumb is often associated with the scales of justice since this also ensures balance.

Figure 8: The Plumb Tool in Masonry

As speculative Masonry continues to use these tools of the operative Masons that came before them, the lessons and teachings of said tools in Freemasonry remain as the firm foundation for all Masons to revert to. Through a hermeneutical approach of these tools and tenets, combined with a translation of the ancient Landmarks of Freemasonry, it is possible to see how these items charged to Masons can be applied to AI systems today.

2.5 Mapping Masonic Virtues to AI Ethics

The disparity between Freemasonry and its associated ethics and values, and Artificial Intelligence systems and their own ethical frameworks is significant since the former has writings tracing back to the early 18th century, while the latter only surfaced less than 90 years ago [2]. Regardless of the timelines, each resemble and require long-standing ethical traditions that emphasize justice, responsibility, and truth. Not all characteristics defined in the "Body of Masonry" are relevant to AI systems, or technology at all, but for the few ancient Landmarks that are relevant (take note of Landmarks II, IV, IX, X, XIV, XXIV, and especially XI regarding security), the connections of these Landmarks to items in AI systems become clear and, to an extent, obvious when examined with a clear understanding of the underlying principles.

Freemasonry uses its teachings and virtues to guide Masons, both operative and speculative, towards the correct conduct found within the Craft. These teachings and virtues are not exclusively intended for Freemasons, since any rational person may adopt a variation of the teachings of Freemasonry and act in accordance with their expectations of all mankind. These teachings and virtues extend even further beyond non-Masons. When examined in the context of more modern and technology-driven systems, these same teachings and virtues provide a valuable foundation for interpreting how the ethics, values and guiding principles of AI systems should be structured.

The goal is to not suggest that Artificial Intelligence is a direct byproduct of Freemasonry, but to instead

demonstrate that these timeless traditions and ancient Landmarks of the fraternity are still relevant to contemporary technological challenges, like defining AI systems' guiding principles. Using the tenets, ethics, teachings and virtues of Masonry, it is possible to link these to items within an AI framework and identify parallels between them.

Brotherly love is the first tenet of Freemasonry. This principle aligns closely to a central concern of AI ethics seen in [2], being fairness and bias mitigation. In this new context, 'fairness' can be understood as the technological equivalent of Brotherly Love. Through the symbolism of the working tool, the level, Masonic teachings encourage the good treatment of others and that all individuals are on the same moral level. Similarly, AI system developers can ensure that the systems that they create operate with this same level principle resulting in the equal treatment of its users operating on moral grounds, while simultaneously avoiding unintentional and unjust bias as much as possible. Many techniques already exist that aim to promote this equal treatment, including the presence of bias detection subsystems, diverse metrics to accommodate fairness, as well as diverse training data reflecting a larger population size, thus mitigating bias and improving fairness.

Relief is the second tenet of Freemasonry. This principle stresses the importance of one using their knowledge and resources to improve the general well-being of society, if this is within their power to do so, and it not being financially, physically, mentally, etc. detrimental to themselves. This idea of relief, or charity, is not limited to charitable acts or donations. This extends beyond this to include social responsibility and engagement within the community. This should reflect exactly in AI ethics since AI systems should promote the well-being of society as well. There have been many ground-breaking discoveries and uses for AI seen in [1]. These key discoveries and uses were realized with the idea of human welfare in mind, and for this to continue, this principle must remain at the forefront of AI developers' minds to better serve society, rather than just that of maximizing profits. Developers must also consider the impacts of these technologies on communities and if they really are contributing positively to them. Potential pitfalls arise when responsibility and protection are neglected. If data and personally identifiable information are not handled correctly, these tools become harmful to society, thus a balance must be struck between what AI tools offer and what they use throughout their processes. This balance can be obtained from the teachings using the plumb in Freemasonry, as this represents uprightness and moral rectitude. These teachings speak directly to AI system implementation by reminding developers that innovation must still align with ethical standards and prioritize the well-being of society.

Truth is the third tenet of Freemasonry. This principle, naturally, upholds honesty, the pursuit of new and further knowledge on one's journey through life, and intellectual integrity. Truth for a Freemason is not only a moral virtue, but also the guide to how one seeks further understanding and their personal development. A comparable ethical item concerning AI systems would be the need for transparency of the system for the user, and trust in the system by said user, again, as far as it is not financially, physically, mentally, etc. detrimental to the developers or company of the AI systems. Evidently, this can be quite difficult if certain computational processes, algorithms, and other trade secrets essential to the success of a company are not known. The lack of transparency, or lack of complete truth, from a company can undermine users' confidence in the system and raise concerns about the aspect of accountability for the system regarding input and data kept, as well as results and potential biases that may appear. It is for these reasons that AI ethics prioritizes transparency as far as companies can provide to maximize user trust in their systems.

Other notable mentions of trust can be seen in AI systems through detailed documentation and well-explained procedures of reaching results and decisions. Additionally, validation and verification processes aid in the reassurance of models operating as they should to users. Similar to the way the knowledge a Freemason seeks can be validated and verified by more experienced members of the fraternity, so can the behaviors of AI systems, ultimately contributing to increased system credibility. The symbolism linked here can be found in the logo, or two most recognizable symbols of Freemasonry, the square and the compasses. Respectively, these working tools teach alignment or the regulation of your behavior in all situations, and acting with integrity, or being disciplined and honest in what you say and do. From a technological perspective, the symbolism teaches constraints within AI systems so that they remain aligned with the well-being of society and human values.

Beyond the three tenets of Freemasonry, other symbols demonstrate equally important parallels required for the success of AI system development. These additional symbols are found in The Number Three, King Solomon's Temple, and Euclid's 47th Problem.

As mentioned, the number three represents completeness. The three tenets of Freemasonry are included in this with each of its symbolic links to AI systems. The completeness of an AI system operating flawlessly from start to finish (gathering data to producing results) with all of the appropriate and necessary guidelines for an ethically profound system can also be linked to this symbol.

The development of these AI systems, including the design and implementation of responsibility can be viewed as impressive intellectual architecture, similar to that of the construction of King Solomon's Temple. The construction of both structures requires precise planning, careful and purposeful designs, strong reasonings and strong foundations, ethically and physically.

Finally, it would be remiss to not mention some kind of mathematical symbolism between Freemasonry and AI systems. Symbolically, the 47th Problem of Euclid represents the importance of Geometry in Freemasonry, in life in general, and in AI systems in particular. This Problem also emphasizes the importance of rational understanding. The obvious link here is that AI systems rely heavily on mathematical models and algorithms to handle the large amounts of data, through analysis and standardization, and to produce results and decisions based on these logical and analytical instructions.

The teachings and symbolism studied in Freemasonry give an unexpected, but valuable framework for understanding ethical principles that are required in AI systems. The timeless tenets align closely with contemporary issues regarding fairness and bias, and transparency and trust in these AI systems. Although the inception of these tenets and ancient Landmarks kept tradition in mind and not technology, they still remain remarkably relevant in the AI digital age. It is important to note that developers and AI researchers can gain important insight on the direction of their technologies and associated ethical frameworks by recognizing certain parallels between them and timeless values.

3. Conclusion

The ethical issues associated with Artificial Intelligence continue to grow and surface with the growth and arrival of newer, more complex systems. This paper explores the foundations of responsible AI ethics through an unconventional, but valuable means of examining certain parallels between Freemason tenets and contemporary AI ethical concerns. The tenets of Freemasonry being brotherly love, relief, and truth, and the ethical concerns of AI being fairness and bias, transparency, and trust.

Advantages of this paper include a thorough demonstration of how AI ethical concerns are not new but include a plethora of moral questions that have continued to be asked for centuries by Masons. The emphasis on equality and social responsibility by Freemasons show the useful structure that can be used to interpret current ethical issues, such as bias, and irresponsible data usage. Symbolically, the working tools used in Freemasonry, along with the teachings and allegory, offer metaphorical representations of discipline and honesty through alignment, fairness and equality, and balance. All of these items are reflections in many of the modern ethical principles highly recommended in AI frameworks. Seeing our evolving AI systems through the eyes of Freemasons reinforces the importance of ethics and responsibility in technological development.

Even with these advantages, some limitations must be acknowledged. The comparison between a timeless fraternity's tenets and modern, innovative systems is abstract, rather than being pragmatic. This does not, therefore, provide immediate quantitative evidence that Freemason teachings and virtues can be easily and systematically implemented into AI system frameworks. In addition to this, Freemasonry does not have a monopoly on philosophical traditions emphasizing moral conduct, so the study of their tenets is not exclusive nor exhaustive. Future

Table 1: the Comparison between the Principles of Freemasonry, the Ethical Equivalent, and the Role it plays in the AI System

Masonic Tenet/Principle	AI Ethics Equivalent	AI System
Brotherly Love	Equal, unbiased treatment of users	• Including diverse metric sources • Ensuring neutrality with decisions and responses
Relief	• Promote the well-being of society • Consider impacts on society	• Handling data and personally identifiable information correctly. • Ensuring innovation and design still align with the well-being of society
Truth	• Honesty • Transparency • Trust	• Systems are truthful about what it does with data • Ensuring model functionality are obvious as much as possible • Ensuring users trust the systems through validation and verification methods.
The Number 3	A Complete and detailed framework for ethical and responsible use	A complete system from start to finish (accepting input/data to producing results)
King Solomon's Temple	High Importance placed on detailed planning and design	Architecture and Structure of models and algorithms
Euclid's 47th Problem	Rational understanding	Mathematical models & algorithms used

work could expand these comparisons by introducing other traditions, as well as developing a more formal framework that can quantitatively translate these tenets and principles into designs for ethical guidelines.

Nevertheless, the mapping in this paper suggests that there are many potential applications for this translation already. The derivatives of Freemason ethics can provide many different perspectives for guiding AI framework development, as well as responsible system design. By recognizing these older, moral questions, new insights for structuring ethical frameworks can support responsible development of artificially intelligent systems.

Ultimately, as Artificial Intelligence continues to evolve, one of the highest priorities is to ensure that these systems operate in fair, unbiased ways that are transparent, trustworthy, and beneficial to society. Linking and integrating insights from timeless and ancient Landmarks may bode well in strengthening the ethical foundations upon which AI is, and will continue to be, built.

References:

[1] J. Holdsworth, & M. Finio, The most valuable AI use cases for business, https://www.ibm.com/think/topics/artificial-intelligence-business-use-cases, 2026.

[2] S. Russell, & P. Norvig, *Artificial intelligence a modern approach* (Harlow, United Kingdom: Pearson Education, 2022).

[3] D. K. Foote, A Brief History of Generative AI, https://www.dataversity.net/articles/a-brief-history-of-generative-ai/#the-2010s-and-virtual-assistants-and-chatbots, 10 September 2025.

[4] T. J. W, Transformers in AI: The Attention Timeline, From the 1990s to Present, https://towardsai.net/p/data-science/transformers-in-ai-the-attention-timeline-from-the-1990s-to-present, 3 June 2024.

[5] A. Karimov, History of Artificial Intelligence, https://swisscyberinstitute.com/blog/history-artificial-intelligence/, 21 January 2026.

[6] G. Arriagada-Bruneau, C. López & M. Mendoza, *Ethics in artificial intelligence*, (Boca Raton, Florida: CRC Press, 2026).

[7] G. W. Reynolds, *Ethics in information technology 6th ed*. (Boston, MA: Cengage, 2019).

[8] M. Ciampa & C. Hoisington, *Introduction to artificial intelligence (*Mason, Ohio: Cengage Learning, 2026).

[9] C. L. Hodapp, *Freemasons for dummies* (Hoboken, New Jersey: John Wiley & Sons, Inc., 2022).

[10] Pennsylvania Masons, Become A Freemason, https://pagrandlodge.org/join/, 2026.

[11] Athelstan Museum Malmsbury, King Athelstan, https://www.athelstanmuseum.org.uk/malmesbury-history/people/king-athelstan/, 2026.

[12] R. C. Blackmer, *The lodge and the craft* (Richmond, Virginia: Macoy Publishing & Masonic Supply Co., Inc., 1976).

[13] M. R. Poll, *Masonic enlightenment the philosophy, histroy and wisdom of freemasonry (*Lafayette, Louisianna: Cornerstone Book Publishers, 2006).

[14] OpenAI, Defining and evaluating political bias in LLMs, https://openai.com/index/defining-and-evaluating-political-bias-in-llms/, 9 October 2025.

[15] Unknown, Freemasonry Logo, Word Creative Commons, Online, 2026.

[16] Unknown, King Solomon's Temple, Word Creative Commons, Online, 2026.

[17] Unknown, Masonry Square tool, Word Creative Commons, Online, 2026.

[18] Unknown, Compasses Tool, Word Creative Commons, Online, 2026.

[19] Unknown, Euclid's 47th Problem, Word Creative Commons, Online, 2026.

[20] The Masonic Collection, Standard Working Tool Plumb Rule, The Masonic Collection, Coventry, 2026.

[21] The Masonic Collection, Standard Working Tool Level, The Masonic Collection, Coverntry, 2026.

Examining Position Bias in LLMs through an AI Stylist

Katherine Powell and Samuel Grieggs
Indiana University of Pennsylvania
yppdc@iup.edu, sgrieggs@iup.edu

ABSTRACT
With the rise in Large Language Models (LLMs) being used for recommender systems, there has been an increased effort to examine their many biases. Position bias is one of these many biases, which is when a model can be sensitive to the order in which the candidates to be recommended are presented, which will affect the overall ranking results. After creating an LLM recommender system in the form of a fashion stylist, we have examined the stylist to determine to what extent it is affected by the position of the candidates.

KEY WORDS
LLM, Recommender, Fashion, Position Bias

1. Introduction

With the rise of large language models, they have been used increasingly in recommender systems. As these recommender systems have gained more popularity, there has been an increased effort to examine their many biases, one of which is position bias. Bito, E., Ren, Y., and He, E. define position bias as: Large Language Models' (LLMs') tendency to rely on the order of candidates in the input rather than their actual relevance to the prompt given [1], [2]. As position bias has been increasingly studied, studies have found that LLMs prioritize the candidates at the top of the list [3].
With position bias being very prevalent within LLM recommender systems, we decided to evaluate our AI fashion stylist for position bias. The AI stylist is a LLM recommender system utilizing LLMs zero shot capabilities. The LLMs are given a system prompt to act as a highly trained personal stylist. It is then asked to recommend different outfits based on event-based parameters such as the event itself, the formality, the weather, and the city in which the event is based. These outfits are made from an imported closet which is made up of descriptions of the different clothing items. These descriptions are generated based on images of the clothing items using the 27 billion parameter variant of the multimodal LLM Gemma 3 [4] and include color, season, style, fabric, cut, and occasion. The models used for the AI stylist are easily interchangeable for any Ollama LLM. Through the AI stylist, we will examine the extent to which Gemma 3 and gpt-oss [5] are affected by position bias.

2. Related Works

Position Bias is a well explored problem in the context of Large Language Models. Liu et al. [2] demonstrated that models tend to have trouble referencing information in the middle of their context windows, heavily emphasizing information that appears in the beginning or end of their context windows. This idea of position sensitivity was also explored by Wang et al. [6] who demonstrated that the bias was strong enough that LLM based evaluation systems could be hacked via altering the order in which the information appears.
Despite the fact that Position Bias is a real problem when using LLMs for recommendations, they have been widely used in the literature for various recommender systems, and have been demonstrated to work sufficiently well, even with the inherent limitations. Some examples of this can be seen in the Lin et al. [7] and Zhao et al. [8] surveys that contextualize some of the methodologies used to perform this task.
Specifically, using LLM's as fashion recommender is also a relatively well explored problem, with work by Liu et al. [9] representing one highlighted example, and Deldjoo et al. [10] offering a broader survey.

3. Experiment

The goal of our experiment was to determine if the AI Stylist's choice in recommending clothing items was affected by the position of the clothing items in the list. For this experiment, we used Ollama's large language and vision [4] [5]from OpenAI.
First, we came up with a prompt for what kind of event, weather, location, and formality the AI Stylist would be styling outfits for. The situation was: going out for coffee with friends, trying to look stylish while still being mostly casual. The location was Pittsburgh, PA, and the weather was 30 degrees Fahrenheit and windy. Then, we

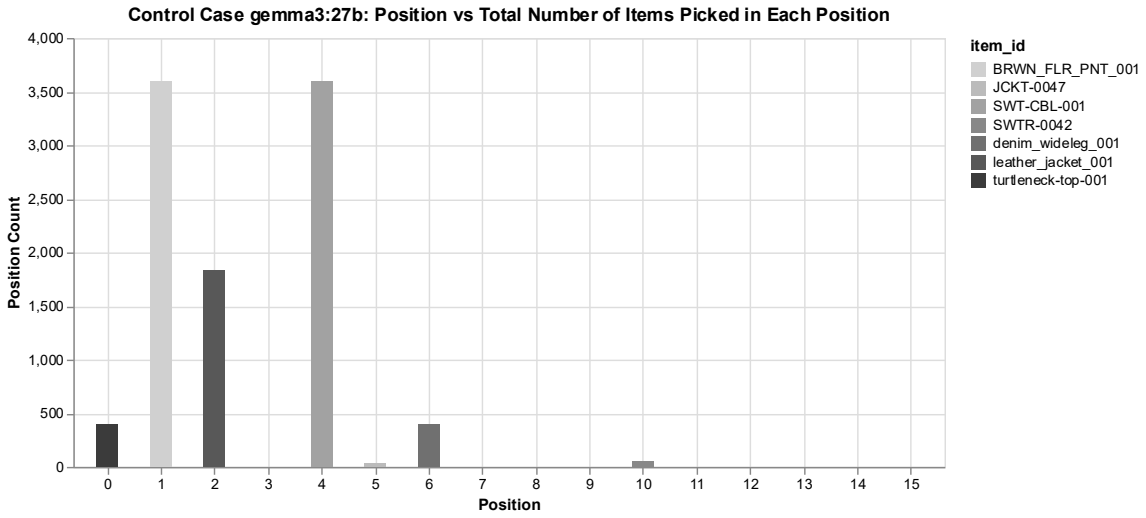


Figure 1: Gemma 3 Control Case

Results from Gemma 3 when object order was not shuffled.

generated 10 similar-sounding prompts, with the Gemma 3 model containing the same key information.

Then, based on that prompt, we compiled a closet of 16 items, which were then converted from images into AI-generated text descriptions using the Gemma 3 model. The text descriptions of the items, including cut, color, season, occasion, and a unique item id. 14 of the 16 clothing items were very suitable for the cold weather and formality expected, while the other two were clothing items that were suited for warmer weather. These two items were denim shorts, which have the item id "denim_short_001", and a tank top, "top-ribbed-squareneck-goldtrim-001".

To start examining the position bias in AI Stylist, we first had to adjust the system prompt, so that the model would produce the chosen items in a JSON file format instead of normal text. This ensured that chosen items could be analyzed more easily as data points. The AI Stylist picks 2 to 4 clothing items, enough to make a full outfit each time it is prompted. For easy of analyzing we have decided to separate these groups of clothing items into induvial data points.

From there, we created the control case for testing for position bias. First, the AI-Stylist would be fed

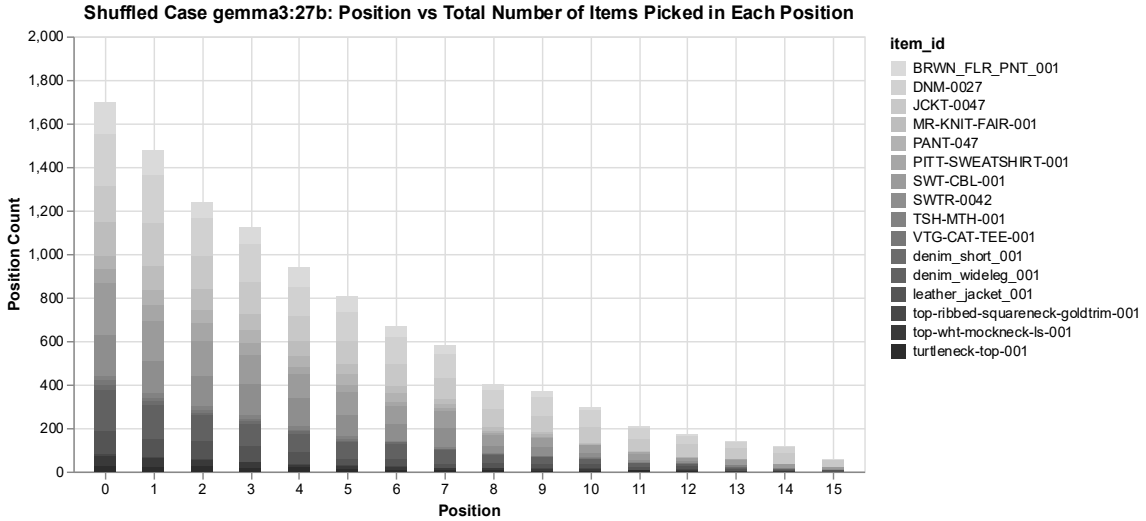


Figure 2: Gemma 3 Shuffled Case

Results from Gemma 3 when the order in which objects appear in the closet are shuffled before each run.

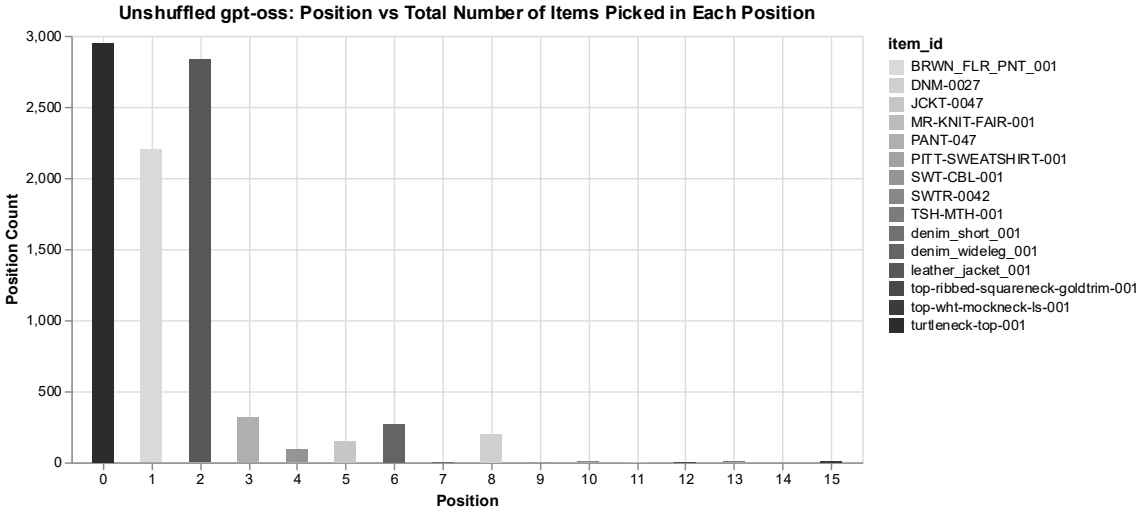


Figure 3: gpt-oss Control Case

Results on gpt-oss when the data was not shuffled.

the system prompt that describes how to be a fashion stylist and how to format the output into a JSON file. The Stylist is then fed the JSON file containing a list of all the clothing items and their descriptions, and one of the 10 different prompts. The 10 different prompts ensure that the Stylist picks clothing items based on the parameters of event, location, weather, and formality, and not based on the prompt being the same wording. Then, the Stylist outputs the items chosen in the JSON file format. , an order number is appended to the clothing item based on the position it is in the closet. Then the output is appended to a list used to store all the chosen items. This series of steps was then repeated 4000 times, and the list of the chosen items are exported into a JSON file.

Next, we created the shuffled case, which purpose was to show if the AI Stylist was affected by position bias. The beginning of the shuffled case starts similarly to the control case, with the system prompt being sent to the AI-Stylist. Then, a copy of the closet JSON file is made and is shuffled. The shuffled closet is sent to the Stylist, along with one of the 10 prompts. After receiving the prompts and closet, the Stylist outputs the items chosen in the JSON file format. Then, an order number is appended to the clothing item based on the position it is in the shuffled closet. After the output is appended to a list used to store all the

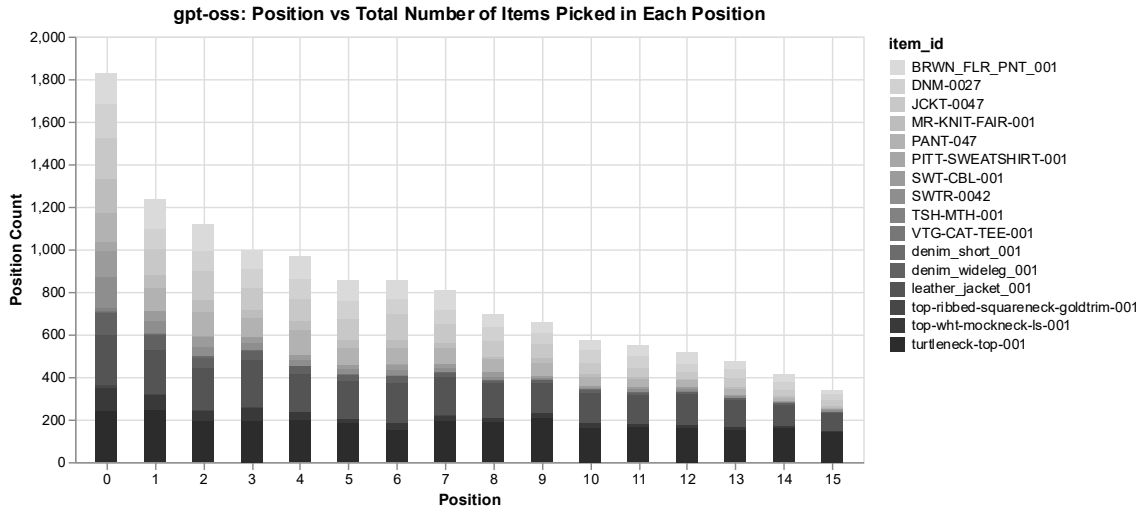


Figure 4: gpt-oss Shuffled Case

Results from gpt-oss when the order in which objects appear in the closet are shuffled before each run.

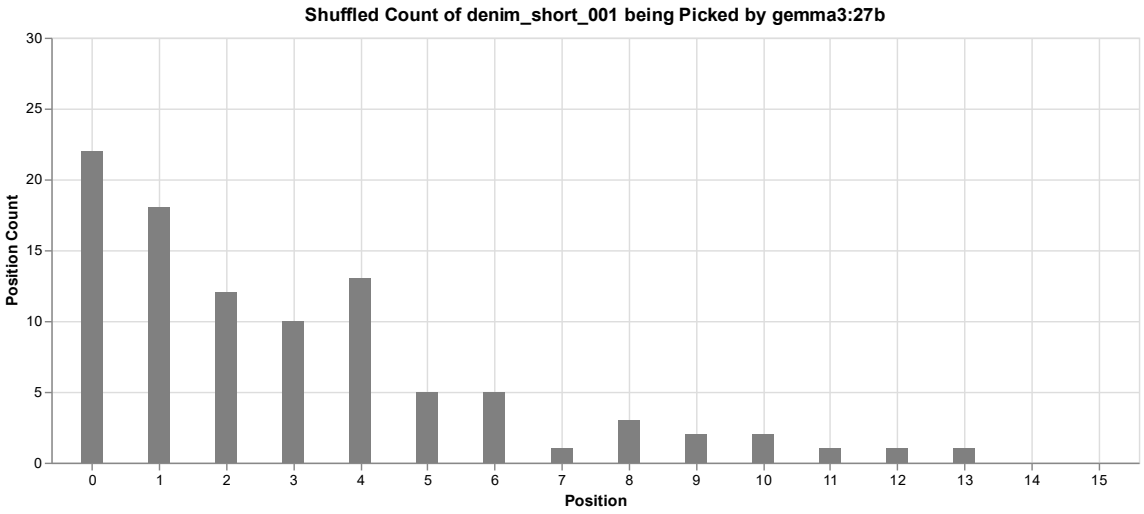


Figure 5: Gemma 3 Denim Shorts

This chart looks at the frequency at which Gemma 3, chose Denim shorts as part of the outfit, this is an inappropriate choice based on the prompt, which indicates a cold day.

chosen items. This series of steps was then repeated 4000 times, and the list of all the chosen items is exported into a JSON file.

Using the methods described above, we tested Ollama's large language models, Gemma 3 and gpt-oss. After examining the data, we found that both models had a significant position bias, favoring candidates towards the front of the list.

4. Results

When using the Gemma 3 as the AI Stylist, the model chose 9,910 items in the control case and 10,327 items in the shuffled case. In the control case, the model did not alter any of the item ids or hallucinate items that did not exist in the closet. However, in the shuffled case, the model altered item ids or hallucinated items 42 times. For the shuffled case data, we have removed the items with altered item ids or that were hallucinated from all graphs and totals. This makes the new item total for the shuffled case 10,282.

In the control case, as seen in figure 1, Gemma 3 only picked 7 of the 16 positions. The most often picked positions were position 1, “BRWN_FLR_PNT_001”, position 4, “SWTR-CBL-001", and position 2, “leather_jacket_001”, picking position 1 and position 4 3,600 times and position 2 1,829 times. These top-picked positions held items that were the first items of their class that appear in the closet list. “BRWN_FLR_PNT_001” is the first pair of pants in the closet list, and “leather_jacket_001” is the first jacket to appear in the closet list. The interesting thing is that “SWTR-CBL-001" is the first sweater to appear in the closet list, but not the first top. The first top was in position 0 and was "turtleneck-top-001". This could be the model determining that a sweater would be better to wear on a 30-degree day in Pittsburgh, PA, rather than a more lightweight turtleneck top. These top-picked positions already imply that there could be a position bias in this model towards the front, since the top-picked items are coming from the front of the list.

In the shuffled case as seen in figure 2, the top-picked position is 0, then 1, then 2, so on and so forth, with position 15, the last position, being the least-picked position, with only 56 picks. Position 0 was picked 1,698 times, position 1 was picked 1,475 times, and position 2 was picked 1,236 times. Positions 0, 1, 2, and 3 make up 53.8% of the total number of items picked, 10,282. A fourth of the positions represent 53.8% of the total number of items picked, supporting the idea that Gemma 3 has a position bias towards the front of the list. Since the closet is shuffled every time the model is called, it does not matter what item of clothing is in these lower-number positions.

It will pick these positions towards the front of the list so much so that the model will pick items that are not very suited to the situation that it was given. The 5 shows the rate at which denim shorts are picked for a 30-degree day in Pittsburgh based on the different positions the denim shorts were in. It was picked 96 times, making the jean shorts 0.9% of the total items picked. When the jean shorts were picked, 64.6% of the time they were in positions 0, 1, 2, or 3. A fourth of the positions make up 64.6% of the times that the jean shorts

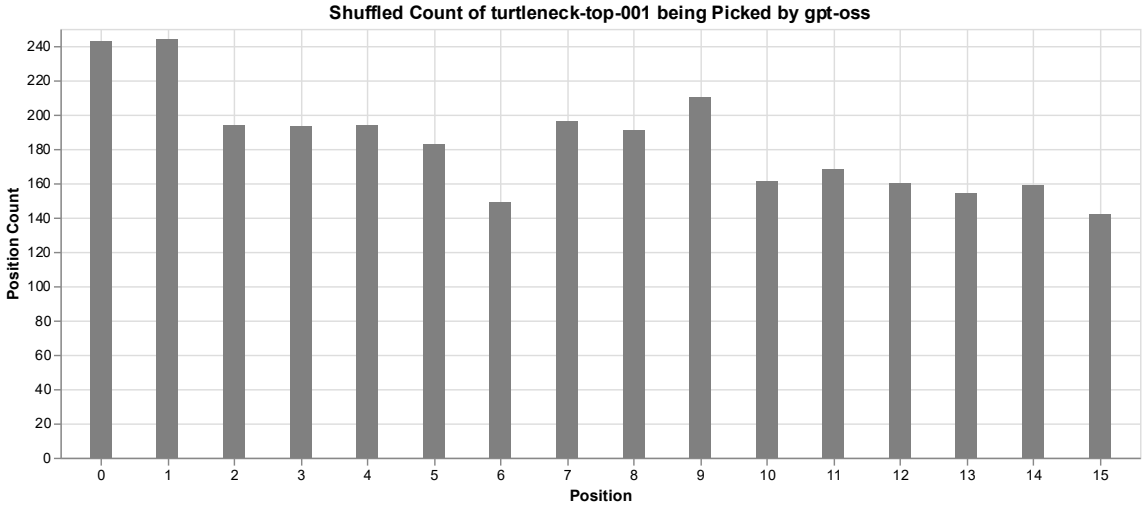


Figure 6: gpt-oss Turtleneck

The frequency and position at which gpt-oss picked a turtleneck, which is a good choice within the constraints of our prompt.

were supporting that Gemma 3 has a position bias towards the front of the list. Jean shorts are not an item that is suited for 30-degree weather, and the fact that when they were picked, they were in the front fourth of positions 64.6% of the time shows how the model Gemma 3 has a position bias towards the front of the list.

When using the gpt-oss as the AI Stylist, we ran the model 3000 times for the control case and 4000 times for the shuffled case because gpt-oss took over 12 hours to run 4000 times. For the control case, the model chose 9,070 items and 12,921 items in the shuffled case. The data required some additional cleaning in both sets to analyze, which required removing times when the mode errored in producing a JSON output and removing the key, outfit, when it appeared before the items the model picked. In both cases, the model altered some item ids or hallucinated some items. In the control case, it did this 29 times, and in the shuffled case, it altered item ids or hallucinated items 42 times. For both cases of data, we have removed the items with altered item ids or that were hallucinated from all graphs and totals. This makes the new item total for the shuffled case 12,879, and the control case's new item total 9,041.

In the control case as seen in figure 3, gpt-oss picked all the positions at least once. The most often picked positions were position 0, "turtleneck-top-001", position 2, “leather_jacket_001”, and position 1, “BRWN_FLR_PNT_001”. It picked position 0 2,952 times, position 2 2,833 times, and position 1 2,201 times. These 3 positions make up 88.3% of the positions picked in the control case. All these top-picked positions hold items that are the first items of their class that appear in the closet list. These top-picked positions already imply that there could be a position bias in the gpt-oss model towards the front, since the top-picked items are coming from the front of the list.

The gpt-oss shuffled case is very similar to the Gemma 3 shuffled case. As shown in figure 4, a noticeable difference is how sharp the initial decrease is. Position 0 is picked 1,828 times, and position 1 is picked 1,475 times, which is a sharper decrease than the difference between position 0 and position 1 in the Gemma 3 shuffled case. It's also important to note that position 6 is picked 856 times, and position 5 is picked 854 times in the gpt-oss shuffled case. Position 15 was still the least picked position, with it being picked 337 times. This is significantly more than it was picked in the Gemma 3 shuffled case. The gpt-oss still has a position bias towards the front of the list, with positions 0, 1, 2, and 3 making up 40.2% of the total items picked, 12,879.

While gpt-oss still has a position bias towards the front of the list, it is more consistent at picking an original item, which, in the control case, it preferred. As seen in figure 6, it picks "turtleneck-top-001" more consistently through the different positions than other items in the list. This model is better at picking a more situationally appropriate item even if they are later in the sequence. The gpt-oss is a newer model than Gemma 3, so its ability to pick an item more consistently could be due to that fact.

5. Conclusion and Future Work

Our work demonstrates that models Gemma 3 and gpt-oss have position bias towards candidates at the front of the list, this is consistent with the expected results from the literature, as the closet is injected into the beginning of the model's context. For Gemma 3 positions 0, 1, 2, and 3 make up 53.8% of the total number of items picked. The model also had an increased tendency to pick an item not suited for the expected weather, jean shorts, when they were in positions 0, 1, 2, or 3. The model gpt-oss still showed bias towards the front of the list, with positions 0, 1, 2, and 3 making up 40.2% of the total items picked. After testing two models for position bias, we are in the process of in testing more models or updating the test to see if different situations and closets change our results. We are also interested in coming up with and testing methods to reduce position bias within LLM recommender systems. One potential approach is to shuffle the closets, show it to the model multiple times, and then vote. Similar approaches have shown success in the work of Tang et al. [11].

References:

[1] E. Bito, Y. Ren, and E. He, "Evaluating Position Bias in Large Language Model Recommendations," Aug. 04, 2025, *arXiv*: arXiv:2508.02020. doi: 10.48550/arXiv.2508.02020.

[2] N. F. Liu *et al.*, "Lost in the Middle: How Language Models Use Long Contexts," *Trans. Assoc. Comput. Linguist.*, vol. 12, pp. 157–173, 2024, doi: 10.1162/tacl_a_00638.

[3] L. Wu *et al.*, "A survey on large language models for recommendation," *World Wide Web*, vol. 27, no. 5, p. 60, Aug. 2024, doi: 10.1007/s11280-024-01291-2.

[4] G. Team *et al.*, "Gemma 3 Technical Report," Mar. 25, 2025, *arXiv*: arXiv:2503.19786. doi: 10.48550/arXiv.2503.19786.

[5] OpenAI *et al.*, "gpt-oss-120b & gpt-oss-20b Model Card," Aug. 08, 2025, *arXiv*: arXiv:2508.10925. doi: 10.48550/arXiv.2508.10925.

[6] P. Wang *et al.*, "Large Language Models are not Fair Evaluators," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 9440–9450. doi: 10.18653/v1/2024.acl-long.511.

[7] "How Can Recommender Systems Benefit from Large Language Models: A Survey | ACM Transactions on Information Systems." Accessed: Mar. 11, 2026. [Online]. Available: https://dl.acm.org/doi/full/10.1145/3678004

[8] Z. Zhao *et al.*, "Recommender Systems in the Era of Large Language Models (LLMs)," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 11, pp. 6889–6907, Nov. 2024, doi: 10.1109/TKDE.2024.3392335.

[9] H. Liu *et al.*, "Sequential LLM Framework for Fashion Recommendation," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, F. Dernoncourt, D. Preoţiuc-Pietro, and A. Shimorina, Eds., Miami, Florida, US: Association for Computational Linguistics, Nov. 2024, pp. 1276–1285. doi: 10.18653/v1/2024.emnlp-industry.95.

[10] Y. Deldjoo *et al.*, "A Review of Modern Fashion Recommender Systems," *ACM Comput Surv*, vol. 56, no. 4, p. 87:1-87:37, Oct. 2023, doi: 10.1145/3624733.

[11] R. Tang, C. Zhang, X. Ma, J. Lin, and F. Ture, "Found in the Middle: Permutation Self-Consistency Improves Listwise Ranking in Large Language Models," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, K. Duh, H. Gomez, and S. Bethard, Eds., Mexico City, Mexico: Association for Computational Linguistics, Jun. 2024, pp. 2327–2340. doi: 10.18653/v1/2024.naacl-long.129.

INDUSTRY ADOPTION OF GENERATIVE AI TOOLS AND SOME IMPLICATIONS FOR SOFTWARE ENGINEERING PEDAGOGY

Stephanie Schwartz and Chad Hogg
Millersville University
{stephanie.schwartz, chad.hogg}@millersville.edu

ABSTRACT
The software engineering industry is rapidly changing, as tools based on large language models allow professionals to automatically generate text such as documentation, application code, tests, and explanations. As educators consider how computing curricula should respond to this changed and continuously evolving environment, many discussions are centered on anecdotal data. Before modifying our own curriculum, we wanted to gain a better understanding of how our alumni and local developers are utilizing these tools in their professional work. To this end, we surveyed 75 software engineering professionals about their adoption of generative AI tools, their patterns of usage, levels of trust in AI-generated code, and perspectives on shifts in educational priorities. Results indicate near-universal usage of generative AI tools (96%) with only measured trust in their output. These findings provide an empirical snapshot of professional practices as pertaining to generative AI and offer a foundation for more evidence-informed discussions about potential changes to computer science curricula.

KEY WORDS
Generative AI, CS Education

1. Introduction

Artificial Intelligence tools capable of generating code based on large language models, such as chatbots or embedded assistants in development environments, have been rapidly adopted by professional software engineers. This has been widely reported [1] [2] and is further validated by the survey results presented in this paper. The broad industry use of these tools, which can generate code from scratch, refactor modules, write test code, and explain existing code, necessitates the re-examination of longstanding assumptions about computer science education and the skills that computer science students should be taught.

2. Survey Details

In order to better ground our own thinking in considering changes to our curriculum, we decided to conduct a survey of software engineering (SE) professionals[1] regarding the use of generative AI tools in their jobs. Our purpose was to better understand several overall questions:

- How extensively, and in what ways, are generative AI tools used in industry?
- Do SE professionals feel that these tools are having an overall positive or negative impact on their work?
- How much do professionals trust AI-generated code?
- Which foundational skills are perceived to remain essential and which might have diminished importance given the adoption of AI tools?
- How do SE professionals feel that computer science programs should respond to the industry use of generative AI coding tools?

2.1. Respondents

The survey was distributed through the authors' professional networks, a departmental Discord server including alumni participants, a departmental alumni mailing list, a Slack channel associated with our local technology community, and a mailing list for a local technical conference. In total, 75 responses were collected from SE professionals. The aim of the varied distribution channels was to reach individuals with varying experience levels and educational backgrounds, while targeting professionals who are closely connected to the organizations where our graduates work and who may be impacted by our curriculum. Responses were gathered between September 2025 and February 2026.

When filling out the survey, we allowed respondents to select from multiple listed job roles or enter roles that were not listed. We did this because many SE professionals wear multiple hats in the development process. The majority of respondents identified themselves as software engineers in

[1] We adopt this term rather than "engineers" or "developers" because our participants include technical project managers and other related roles.

core development roles. Excluding job titles with only a single response, the most commonly reported roles were:

- Software Engr (Full-Stack): 35 (47%)
- Software Engr (Backend): 17 (23%)
- DevOps Engr: 8 (11%)
- Engineering Mgr/Director: 7 (9%)
- Data Science/Machine Learning Engr: 4 (5%)
- Technical Project Mgr: 4 (5%)

Respondents represented a broad range of professional experience as follows:

- $<$ 2 years: 6 (8%)
- 2-5 years: 19 (25%)
- 6-10 years: 18 (24%)
- 11-15 years: 9 (12%)
- $>$ 15 years: 23 (31%)

The majority of respondents (75%) hold at least a bachelor's degree in computer science or a closely related discipline. However, a number of respondents also hold higher degrees (20%) or represent alternative pathways (20%) such as bootcamps or self-study. The variety of experience, background, and roles represented in our respondent pool should help to provide a broad view of generative AI usage and perceptions.

3. Survey Results

The following sections present the data gathered through our survey. We first examine how SE professionals are currently using generative AI tools in their work. We then describe respondents' perceptions of AI-generated code and its impact on development workflows. Finally, we report participants' perspectives on the need for foundational computer science skills and the potential impact of generative AI tools on computer science curricula.

3.1. Usage of generative AI Tools

The usage of generative AI tools is nearly ubiquitous among the SE professionals surveyed. Of the 75 participants, 96% (72 respondents) report using these tools in their work, while only 4% (3 respondents) report never having used generative AI tools for work purposes. It is also important to note that 71% of respondents report frequent or constant use of generative AI tools. These results indicate that the use of generative AI tools is widespread, if not pervasive, in professional SE practices. Note that there may be some self-selection bias in terms of who chose to respond to the survey. Since it concerned generative AI tools, SE professionals using these tools may have been more likely to respond. This would be consistent with wider industry results such as GitHub's 2024 survey on AI in software Development reporting 97% usage [2] while Stack Overflow's more general 2025 Developer Survey reported 84% of developers using AI tools in their development process [1].

We also asked participants how they are using generative AI tools while coding, including which modes of interaction and for what purposes. In terms of mode of use, we asked participants whether they use generative AI tools primarily 1) for autocompletion or for chatting as they code, 2) for having AI directly write or modify/refactor code (agentic mode), or 3) as a balanced mixed of both types. There was also a fourth option for those who do not use any of these tools while coding. The largest group of respondents, 43% (32 respondents), reported using a balanced mix of agentic and non-agentic interactions. This indicates a workflow in which SE professionals alternate between asking generative AI systems for suggestions, explanations, or debugging assistance and having generative AI working as a collaborator in code implementation. However, 25% of respondents reported using these tools primarily in agentic mode where the tool is acting as a semi-autonomous collaborator and 20% reported primarily using these tools for autocompletion or chat-based assistance. Finally, 9% of respondents reported not using generative AI while coding. It should be noted that this figure is higher than the previously reported 4% of respondents not using these tools at all for work, which implies that some respondents are using generative AI in other areas of their work, but not for coding. These responses, when combined, suggest that generative AI tool integration into SE professionals' workflow is not uniform and that developers are finding their own rhythms and preferences. For educational purposes, this makes the teaching of these tools and potential workflows much more complex.

We also asked respondents which generative AI tools they use for professional purposes and for what tasks they use them. The tool usage reflects both diversity and some consolidation around a small number of widely used tools. The most frequently used tools, where respondents could choose multiple tools, were:

- ChatGPT: 40 (53%)
- GitHub Copilot: 31 (41%)
- Google Gemini/Bard: 20 (27%)
- Claude Code: 20 (27%)
- Cursor: 10 (13%)

This list of tools corroborates the answers to the previous question in terms of the different modes of use, agentic and non-agentic, since some are embedded in IDEs and some are chatbot-based tools.

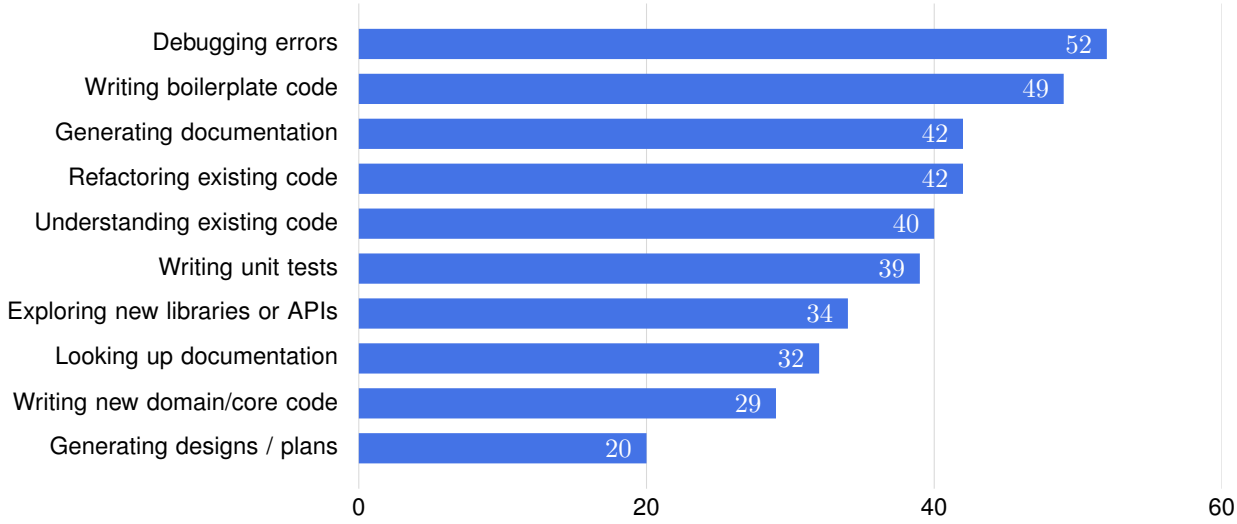


Figure 1: Number of respondents who regularly use generative AI for various software development tasks (N=75).

In terms of the tasks for which SE professionals are utilizing generative AI tools, the counts for the tasks we included in our question are shown in Figure 1. These tasks span much of the software lifecycle and vary from specific, low-level tasks to higher-level design tasks. We confirmed a hypothesis that we had formed while composing the survey that generative AI tools are currently still being leveraged more frequently for procedural, repetitive, and support-oriented activities, while being used less frequently for core conceptual design and higher-stakes engineering judgments. We also allowed respondents to enter other tasks for which they use generative AI in their work. We received only a handful of additional uses, which gives us some additional overall confidence in the breadth of the original list. Our favorite free-form response was "Helping me to have difficult conversations with various types of coworkers" since it was a use-case that we we had not anticipated but can appreciate.

3.2. Perceptions of Generative AI Tools

While adoption rates provide a measure of the integration of generative AI tools into professional SE workflows, we also sought to better understand how SE professionals view the output from these tools. Therefore, we included in the survey questions aimed at assessing perceived code quality, trust in AI-generated code, overall experience with generative AI tools, experience with coworkers use of these tools, and perceived effects on overall productivity. Open-ended prompts provided respondents with opportunities to provide both positive and negative examples and comments.

3.2.1. Code Quality

We will begin our discussion with the question of the quality of code generated by AI tools (Figure 2, far-left), since if the code is simply not reliable or usable, the broader impacts of generative AI on the industry would be negligible. Participants were asked to rate the overall quality of code produced by these tools on a scale from 1 to 5 with 1 representing "Poor quality, often incorrect" and 5 indicating "Excellent quality, rarely needs changes". Responses clustered around the midpoint of the scale with the most common response being a 3 ("Decent quality, often needs minor tweaks") and there were very few responses at either end of the scale. These responses suggest that, in the perceptions of participants, generative AI typically produces code that is useful but imperfect. Some of the open-ended responses support this characterization. For example:

- "Generative AI autocomplete is good at looking at the first instance of a typing-intensive change and guessing how a similar change might be shaped. Typically it will be slightly wrong but still a useful time saver."
- "I asked AI to convert a library written in Groovy into Python and it did it all correctly except for one function and even made correct unit tests for all functions except for one. I simply fixed those myself. It saved about two hours of time."
- "We were having trouble unintentionally closing postgres cursors while streaming large database results. Chatting with ChatGPT gave us some ideas for how to resolve the issue (additional flags on the JDBC statement), but the code generated was incorrect, forcing us to revisit documentation and see how things were really being used. So it helped lead

Figure 2: Survey participants' perceptions of the results and impact of generative AI tools on their work (N=75).

us to a solution even though the solution it provided was incorrect."

3.2.2. Overall Experience

These results and observations tend to note small glitches within an overall helpful direction, so it makes sense that when asked to rate the overall experience of using AI tools while coding, participants responded slightly more favorably than on the perceived quality of code (Figure 2, second from left). While the data still clusters around the middle of the scale, 80% rated the overall experience 3 ("Decent, helpful overall, but it is not always exactly what I am looking for or asked for"), 4 ("Good, it is pretty easy to get it to generate what I need, and I am more efficient"), or 5 ("Excellent, easy to use, I can focus on larger issues and am far more efficient"). We interpret these results as indicating that even when generated code is flawed, the overall interaction process is still valuable.

3.2.3. Trust

Trust ratings (Figure 2, center) show a slightly more cautious pattern. Nearly half of the respondents report "moderately" trusting generated code (a 3 on the scale), with 42.6% reporting "slightly" or "not at all" trusting the code. All three of these responses indicated that checking/testing/validation was required. Only 8% of participants reported being very confident in the generated code and only performing a quick, high-level review of the output. No respondents indicated that they "completely" trust the generated code.

3.2.4. Coworkers

When participants were asked whether they have been positively or negatively impacted by colleagues' use of generative AI for coding tasks, the results span the scale, again clustering in the middle with 42.7% responding that they do not think that they have really been impacted (Figure 2, second from right). However, 23% feel that they have been positively or somewhat positively impacted and 33.3% feel that they have been negatively or somewhat negatively impacted. In the open-ended follow-up for this question, some of the uncertainty in the responses seems to come from a lack of transparency in terms of who is and is not using these tools. However, several who indicated that they have not been impacted noted a workplace culture that heavily emphasized accountability. One participant said, "We all take responsibility for our code, AI or not, so you can not blame AI here" and another said, "My workplace has a culture of careful review; I have not been impacted by AI-introduced bugs in colleagues' code." Much of the impact of colleagues' habits seems to come during peer reviews of code. Some participants note being able to produce higher-quality code reviews due to generative AI tools, while others complain about the sheer amount of code that has to be reviewed having been dramatically increased by the use of these tools. Three responses in particular stood out:

- "The amount of work I need to review from my peers heavily using AI is much larger (they are writing more, committing more code faster), but the answers I get in response to questions about their code are shallow. It is frustrating and the maintainability of the code by humans is suffering."

- "Irresponsible use of AI from junior developers has introduced a lot of pain into our code review process."
- "Some colleagues have embraced AI use to help them become more productive. This means that they generate more code and more solutions faster, but as a senior developer who needs to approve pull requests, check for cross-module incompatibilities, and oversee designs across our codebase, it has created much more work for me. I often see AI generated 'solutions' that could have been avoided with a little more communication between peers."

Taken together, these responses highlight a social dimension to the adoption of generative AI tools in software engineering. Even though these tools may be perceived to improve the efficiency of individuals (see the next section), the workload may be shifted elsewhere in the process such as code review and integration or architectural oversight.

3.2.5. Productivity

The final set of results to be discussed in this section is perceived productivity changes (Figure 2, far right). When asked "How has generative AI impacted your overall productivity as a software engineer?", the results skewed positive. On a scale of 1 to 5 with 1 indicating "Significantly decreased it" and 5 indicating "Significantly increased it", 65.3% of participants indicated that their productivity has increased (ratings of 4 or 5). Only 4% feel their productivity has decreased (rating of 2), with none feeling that it has significantly decreased (rating of 1). This pattern of perceived increases in productivity is notable when considered together with the earlier results. Although respondents generally only rated generated code output as "decent" and expressed only cautious trust in it, most still reported gains in productivity. Qualitative responses help to contextualize this observation. Respondents frequently describe time gains in tasks such as debugging, writing boilerplate code, and especially in understanding unfamiliar code bases or libraries.

3.3. Perceptions of Computer Science Curricular Impact

The positive productivity gains and overall experience using generative AI tools combined with imperfect code quality and conditional trust suggests to us that there may be a shift in the skills and foundational knowledge needed by new SE professionals. To this end, the survey asked participants to reflect on how the increasing use of generative AI tools in software engineering should (or should not) impact CS education. Several questions addressed whether the ability to effectively use AI tools is itself a learned skill (particularly "prompt engineering"), as well as respondents' views on whether the skills being taught in CS curricula should be different (or have different emphasis). Additional questions asked for respondents' views on the specific role of generative AI in CS curricula and issues such as academic integrity.

3.3.1. Learning to Use Generative AI Tools

Participants were asked whether their ability to use generative AI tools while coding was a skill that came naturally or had to be deliberately developed. Responses suggest a wide range of experiences (Figure 3) with a nearly even split between perceptions of learning the skill naturally vs actively learning or developing the skill.

When asked to rate the importance of prompt engineering (defined as the ability to write effective prompts for generative AI systems), responses indicate a moderate but not overwhelming perceived importance, with responses clustering around the middle of the scale from 1 to 5.

Taken together, these results suggest that while there may be a learning process involved in designing effective prompts and learning to use AI coding tools, it is not yet clear to us that these pragmatic skills represent a primary foundational skill competency.

3.3.2. Perceived Foundational Skill Impact

Participants were asked to reflect on which foundational skills from their computer science education remain critical for their current work. Figure 4 shows the skills listed in the original question, although some labels have been shortened for display. For example, "Computer Systems Knowledge" included "(computer architecture, operating systems)" and Low-level Programming included "(e.g. memory management and pointers)".

Several foundational skills were consistently identified as still essential, including debugging and problem solving, software design and architecture, communication and teamwork, programming language proficiency, data structures, and computer systems knowledge. It is notable that no respondents listed communication and teamwork as being less important in light of generative AI tool usage.

Some of the skills considered less important given generative AI coding capabilities were manual documentation and commenting, low-level debugging of syntax errors, writing code from scratch, and algorithms. However, even these skills were still selected as "still critical" by many respondents. We believe that the tension observed in these responses may correspond to a shift in emphasis on some of these specific skills where AI can be most useful but not a total replacement of these skills. This

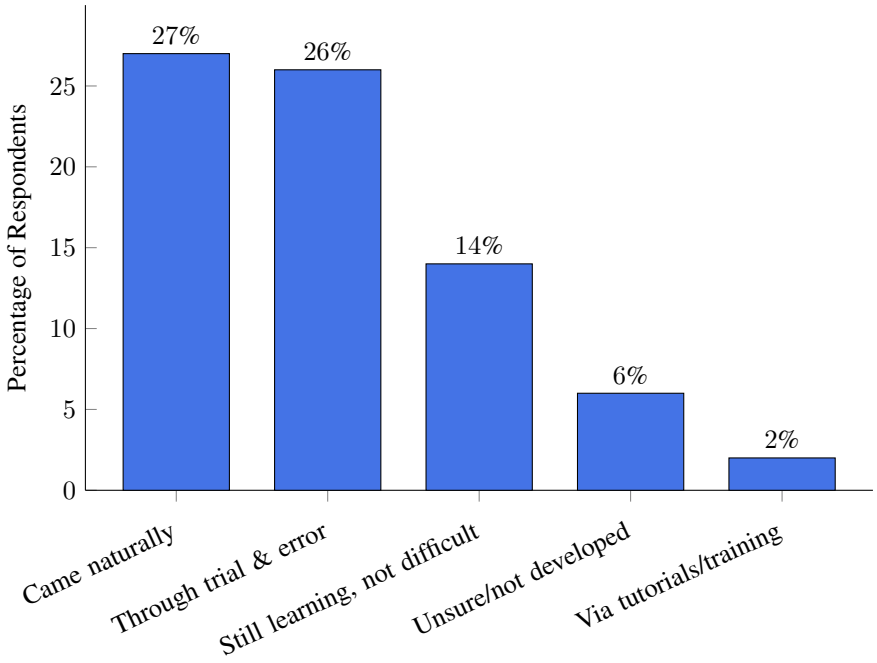


Figure 3: How current professionals learned to use generative AI tools.

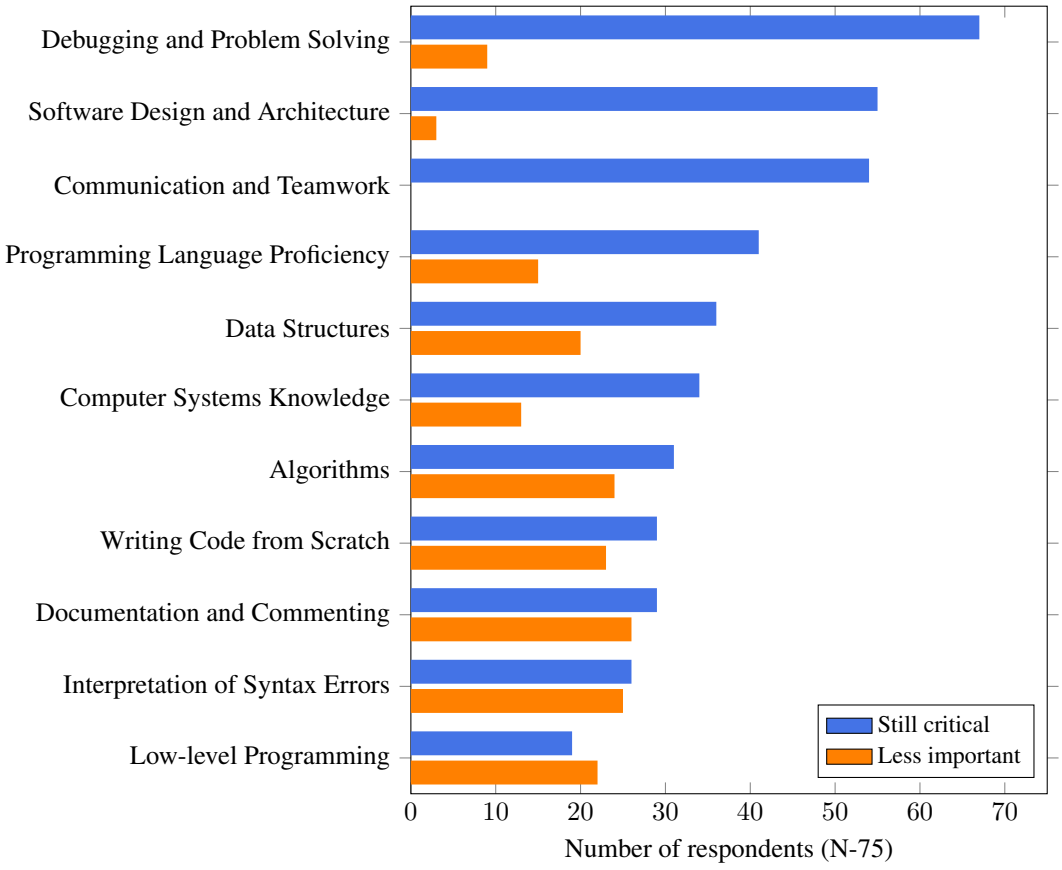


Figure 4: Perceptions of the importance of foundational skills in generative AI era.

hypothesis is backed up by an abundance of qualitative comments which leaves us with little doubt that current professionals still deem a computer science education containing the foundational skills to be key to successfully using generative AI tools to increase productivity.

- "AI can help write the code, but you still need to understand the algorithms and structures behind it."
- "It is difficult to fix the broken parts of generated code if you do not understand what the code is doing."
- "I think that learning the above topics is important even with access to generative AI. It often needs to be guided, which can require having a deep understanding of topics to get AI unstuck, even if a Computer Scientist is not writing everything from scratch."
- "The fundamentals remain important. You can ask an AI to code for you, but you need to be able to reason about what is computationally practical/achievable. Practical skills involve understanding the engineering behind the code."

3.3.3. Perspectives on Generative AI Tools in CS Curriculum

Participants were asked to share their perspective on whether, when, and how generative AI tools should be incorporated into the CS curriculum. While the overwhelming majority (81.3%) supported the inclusion of these tools in the curriculum, opinions on how it should be done varied widely. The most common response, with 37 participants (49.3%), was that AI tools should be integrated into existing courses. However, 24 respondents (32%) indicated that it should be a standalone course or an optional elective/workshop. 10 participants (13%) indicated that they were unsure how AI tools should be included in the curriculum. Only 4 respondents (5%) felt that the topic should not be formally taught. The qualitative responses to the open-ended question of "In your opinion, what changes should computer science programs make to adapt to the prevalence of generative AI?" provide a wealth of ideas and opinions. but no dominant themes other than the fact that generative AI is here to stay and students should be exposed to it due to its heavy presence in industry. Several respondents highlighted the importance of teaching the ethics of using generative AI and also the value in thinking critically about when and how to use it.

4. Pedagogical Recommendations

Due to the fast-paced rollout and adoption of generative AI tools for coding, computer science educators are facing intense pressure to adapt curricula to this new reality. Opinions on how best to respond vary widely. Some argue that instruction should shift toward prompt engineering strategies and early and effective AI tool use (for example [3] [4] [5]) while others staunchly contend that foundational competencies (such as writing code from scratch, comprehensive knowledge of data structures, etc.) remain as important as ever [6] [7] [8]. Within only a few years, systematic literature reviews have already identified dozens of studies investigating how generative AI tools impact programming instruction, assessment practices, and student learning outcomes [9] [10] [11] as educators grapple with whether, when, and how to incorporate these tools into their courses.

Rather than taking a stance on what we are still struggling to understand, the results of our survey suggest a middle ground, at least for the short term. Across multiple questions, respondents emphasized the continued importance of foundational knowledge while also describing software engineering workflows where generative AI tools assist with a wide variety of tasks. A consistent theme emerging from the survey results is that these tools appear to shift effort away from writing code and toward debugging, evaluating, and validating generated solutions. This potential shift has been noted by other researchers as well [12]. While most of our survey respondents reported productivity gains from generative AI tools, in the answers to open-ended questions, it was often noted that the ability to generate a lot of code very quickly also caused an increased burden in terms of reviewing, fixing, testing, and integrating code.

From an educational perspective, this shift and the areas of increased work imply that students will benefit from increased emphasis on skills related to analyzing and evaluating code produced by others (whether peers or AI). This could generally be viewed as an increased emphasis on program comprehension, which has long been an integral skill in computer science education [13]. Two instructional practices that directly develop these evaluation-oriented skills are structured code review and systematic testing practices. It is important to note that we are advocating for increased emphasis on these skills while still teaching the foundational skills, an approach supported by our survey data.

4.1. Code Reviews

It is extremely common for professional software engineers to require code reviews before any changes to products are made permanent. In fact, many workflows require a peer code review before code can even be merged onto a development branch in a version control system [14] [15]. In a code review, one SE (often, but not always, a more senior one) reads through and critiques changes proposed by another SE. They might do so only by looking at the changes themselves, or by both reading the changes and interviewing the proposer about them. A code review can

detect code that is incorrect, inefficient, unmaintainable, unnecessary, or in other ways deficient while it is still fresh in peoples' minds and ensures that at least two people are familiar with each component of a project.

The same techniques can be used when a developer (even a junior one) requests that an AI tool generate code to solve some problem. Our survey results say that developers typically find solutions proposed by AI tools to be directionally correct but wrong in several important details. They are forming these assessments through code reviews.

Being able to read proposed code that you did not write and attempt to understand what it is doing and make judgments about whether it is achieving its goals is thus now a skill that all SEs need and cannot learn through the process of having their own code reviewed on the job. Participating in a code review as the change proposer is also a skill, as it requires being able to clearly explain the decisions you have made and the alternatives that you considered. As junior software engineers may be proposing solutions that are partially or wholly generated by AI tools, they need to know how to build this thorough understanding even about code that they did not personally write. Thus, computer science programs should be teaching students to effectively participate in code reviews both as the reviewer and the reviewee.

4.2. Testing

Thinking about and writing tests that verify that code works correctly has long been a foundational skill for software engineers [16] and Edwards advocated teaching testing as a way to move students toward a more reflective mode of problem solving [17]. Many developers work (or at least strive to) in a "test-driven development" paradigm, which centers tests as the most important part of a code base [18]. As such, most computer science programs already teach their students how to effectively write and maintain tests.

Several of our respondents indicated that one of the tasks they frequently assign to generative AI tools is writing unit tests. This might suggest that testing will be a less important skill for new SEs, but we believe that exactly the opposite is true.

Tests are how we confirm that code works correctly, but a fully passing suite of tests with perfect coverage of our code can be dangerously comforting. If a generative AI tool (or a human) misunderstands a problem, they will not only write incorrect code but also tests that testify to the correctness of the incorrect code. The tedium of converting the idea of a test into code that can be automatically run may be easily automatable, but deciding which cases are interesting enough to test for (e.g. finding edge and corner cases) and what the correct results are for those cases requires deep understanding of the desired code and its context.

Good, human-directed testing can provide confidence that AI tools have generated code that matches human-created specifications. It is thus more important than ever that computer science students be taught how to create and use tests, and for doing so to be modeled to them as an essential, standard part of the development process.

5. Conclusion

Generative AI tools are rapidly being integrated into professional software engineering workflows, raising important questions about how computer science education should respond. This paper presents the results of a survey of software engineering professionals exploring how generative AI tools are being used in industry, perceptions of the quality and reliability of AI-generated code, and how professionals believe CS curricula should evolve.

This work makes three primary contributions. First, it provides empirical insight into how SE professionals are currently using generative AI tools and how those tools are impacting workflows. Second, it documents professionals' perspectives on the continued importance of foundational computing skills. Third, it leverages these findings along with existing research to propose pedagogical recommendations.

Our survey results indicate that generative AI tools are widely used by SE professionals for a variety of development tasks. Respondents frequently reported productivity gains from these tools while also expressing only moderate levels of trust in AI-generated code. Qualitative responses suggest that while generative AI can accelerate code generation, it may also shift effort toward activities such as reviewing, verifying, and integrating generated solutions. Participants also emphasized that foundational computer science knowledge remains essential despite the growing use of AI tools. At the same time, many respondents noted that the ability to critically evaluate code is becoming increasingly important in AI-assisted development. Together, these findings suggest that computer science curricula should continue to emphasize foundational computing knowledge while adding emphasis to skills in evaluating, analyzing, and verifying code, particularly through practices such as code review and testing.

There are also several limitations inherent in this work. The survey responses represent a convenience sample drawn largely from professional networks and community channels (though this was strategic for us because we wanted to focus on stakeholders who might be impacted by our curriculum). It is possible that the whole population

of software engineers is different in important ways from those in our local area. It is also possible that there is a nonresponse bias, as people who are using generative AI are more likely to find an invitation to be surveyed about it interesting than those who are not. Also, the survey results reflect self-reported perceptions rather than direct measurement of outcomes. Finally, the ecosystem of generative AI tools is evolving rapidly, meaning that developer practices and attitudes will certainly continue to change as tools mature.

Despite these limitations, the results provide insight into how professionals are currently integrating AI tools into their workflows and how they perceive the implications for computer science education. Preparing students for this evolving landscape will require not only teaching them how to write code, but also how to critically evaluate and validate code, regardless of whether it was written by a peer or generated by an AI system.

References

[1] Stack Overflow, 2025 Developer Survey, https://survey.stackoverflow.co/2025, Accessed March 6, 2026.

[2] Kyle Diagle and GitHub Staff, Survey: The AI wave continues to grow on software development teams. https://github.blog/news-insights/research/survey-ai-wave-grows/, August 20, 2024 (Updated April 15, 2025).

[3] Wenhan Lyu, Yimeng Wang, Tingting (Rachel) Chung, Yifan Sun, and Yixuan Zhang, Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study. In Proceedings of the Eleventh ACM Conference on Learning @ Scale (L@S '24), Association for Computing Machinery, New York, NY, USA, 2024, pages 63–74.

[4] Orit Hazzan and Yael Erez, Rethinking Computer Science Education in the Age of GenAI, ACM Transactions on Computing Education, 25(3), Article 26, September 2025, 9 pages.

[5] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman, Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), Association for Computing Machinery, New York, NY, USA, Article 455, 2023, pages 1–23.

[6] Valerie Barr, Feeling cranky about AI and CS education. https://cacm.acm.org/blogcacm/feeling-cranky-about-ai-and-cs-education/, August 29, 2025.

[7] Bruno Pereira Cipriano and Lucio Studer Ferreira, Programmers Aren't Obsolete Yet: A Syllabus for Teaching CS Students to Responsibly Use Large Language Models for Code Generation, https://arxiv.org/abs/2502.15493, February 21, 2025.

[8] Marcus Birkenkrahe, The Role of AI Coding Assistants: Revising the Need for Literate Programming in Computer and Data Science Education, In Proceedings of the 18th International Technology, Education and Development Conference, Valencia, Spain, 2024, pages 127-132.

[9] Said Elnaffar, Farzad Rashidi, and Abedallah Abualkishik, Teaching with AI: A Systematic Review of Chatbots, Generative Tools, and Tutoring Systems in Programming Education, International Journal of Learning, Teaching and Educational Research, 25(1), 2026, pages 1-28.

[10] Juan Garzon, Eddy Patino, and Camilo Marulanda, Systematic Review of Artificial Intelligence in Education: Trends, Benefits, and Challenges, Multimodal Technologies and Interaction", 9(8):84, 2025.

[11] Maria Ijaz Baig and Elaheh Yadegaridehkordi, ChatGPT in the higher education: A systematic literature review and research challenges, International Journal of Educational Research, Volume 127, 2024.

[12] Yael Erez and Orit Hazzan, Program Comprehension as a Central Skill in CS Education in the Era of Generative AI, https://cacm.acm.org/blogcacm/program-comprehension-as-a-central-skill-in-cs-education-in-the-era-of-generative-ai/, December 4, 2025.

[13] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, and Renske Weeda, Fostering Program Comprehension in Novice Programmers - Learning Activities and Learning Trajectories, In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '19), Association for Computing Machinery, New York, NY, USA, 2019, 27–52.

[14] Oleksii Kononenko, Olga Baysal, and Michael W. Godfrey, Code review quality: how developers see it. Proc. 38th Intl. Conf. on Software Engineering, Austin, TX, 2016, 1028-1038.

[15] Alberto Bacchelli and Christian Bird, Expectations, outcomes, and challenges of modern code review, In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13), IEEE Press, 2013, pages 712–721.

[16] Otávio Augusto Lazzarini Lemos, Fábio Fagundes Silviera, Fabiano Cutigi Ferreri, and Alessandro Garcia, The impact of software testing education on code reliability: an empirical assessment, Journal of Systems and Software, 137, 2018, 497-511.

[17] Stephen H. Edwards, Using software testing to move students from trial-and-error to reflection-in-action. In Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE '04), Association for Computing Machinery, New York, NY, USA, 2004, pages 26–30.

[18] Kent Beck, Test-Driven Development: By Example, Addison-Wesley Professional, 2002.

INSTITUTIONALIZING CLIENT-BASED CAPSTONES THROUGH CAREER SERVICES: A PASSHE CASE STUDY

Linh B. Ngo, Md Amiruzzaman, Richard Burns
Computer Science Department, West Chester University
{lngo,amiruzzaman,rburns}@wcupa.edu

ABSTRACT

Regional public universities are increasingly expected to provide authentic experiential learning opportunities that align with workforce needs while operating under constrained resources. This paper presents a practice-based case study of a partnership model between an institution's career development center and the Computer Science department at a regional state university to provide students with local and regional businesses exposure through departmental-specific events and real-world projects for the senior-level capstone course. Rather than relying solely on informal faculty outreach, the model formalizes employer engagement through existing career services infrastructure, enabling systematic client identification, vetting, and onboarding. This approach reduces faculty workload, improves project feasibility, and increases employer preparedness for student collaboration. Observations across multiple course offerings and events suggest that the model strengthens student professionalism, enhances employer engagement, and reinforces the university's role as a regional talent pipeline. The paper details roles in the model, outreach workflows, and implementation lessons, and concludes with practical guidance for replication at other teaching-focused and regional institutions.

KEY WORDS
capstone, experiential learning, career development center

1. Introduction

Experiential learning has long been recognized as a high-impact educational practice, particularly in disciplines where professional readiness and applied skills are central learning outcomes. Foundational work in experiential learning theory emphasizes the importance of authentic tasks, reflection, and iterative engagement with real-world problems [1]. More recent studies have demonstrated that applied and work-integrated learning experiences contribute positively to student engagement, skill development, and post-graduation employability [2]. One well-known and successful example of integrated experiential learning in the region is Drexel University's co-op program model [3], where students are aggressively placed on multiple rotations at companies in the Philadelphia area.

There are very few institutions in the U.S. that could replicate Drexel University's setup, and implementing sustainable experiential learning models remains a significant challenge. At teaching-focused institutions, faculty often rely on informal industry contacts to secure external projects. This results in fragile arrangements that are difficult to scale or sustain. When such partnerships falter, the burden of recovery frequently falls on individual instructors, increasing workload and limiting long-term viability. These challenges are particularly acute in capstone courses, where project authenticity and external accountability are central to the learning experience.

The career development centers at higher-education institutions represent a largely underutilized institutional resource in addressing these challenges. While traditionally positioned as post-curricular support units, career offices increasingly possess strong regional employer networks and expertise in employer engagement, professional preparation, and workforce alignment. Prior work on university-community partnerships suggests that institutionalizing external engagement through existing organizational structures improves sustainability and stakeholder alignment [4, 5].

This paper presents a case study of a structured outreach model that integrates our institution's Career Development Center into the design and execution of a computer science senior capstone course at a PASSHE institution. By proactively collaborating with career services rather than considering them as a downstream service provider for students, the model establishes a repeatable pipeline connecting local businesses, students, and faculty. The goals of this work are to (1) describe the design and implementation of the outreach pipeline, (2) examine its impact on student and employer engagement, and (3) provide practical guidance for replication at similar institutions. The remainder of the paper is organized as follows. Section 2 provides the motivation and institutional/departmental context. Section 3 describes how the new collaborative relationship is designed. The modifications to the capstone course's implementation are discussed in Section 4. Section 5 examines preliminary student and external client feedback. Section 6 discusses lessons learned, and Section 7 concludes the paper and provides a general framework for community adoption.

2. Historical Context and Motivation

Our institution and others in PASSHE occupy a distinctive position within the higher education landscape. As regional public universities, they are tasked with providing accessible, high-quality education while maintaining strong alignment with workforce development and regional economic needs. We typically serve a high proportion of first-generation and commuting students and operate under resource constraints that limit the feasibility of labor-intensive instructional models. Within this context, experiential learning is both a strategic priority and an implementation challenge. Our senior capstone course plays a central role in addressing this tension. As culminating academic experiences, capstones are often expected to integrate disciplinary knowledge with professional competencies such as communication, teamwork, and problem solving. Prior research has identified such applied learning experiences as high-impact practices, particularly when they involve authentic tasks and external stakeholders [2]. However, as we develop our capstones, the responsibility for sourcing and sustaining external projects frequently falls on faculty members assigned to the course. This reliance on informal networks introduces fragility into course design, making experiential learning vulnerable to personnel changes, shifting workloads, and inconsistent partner engagement.

At the same time, our institution maintains a Career Development Center (CDC), a centralized unit that supports student career readiness through employer engagement, internship coordination, and workforce-aligned programming, with established relationships across regional employer networks. The CDC is already tasked with employer outreach, student professional preparation, and workforce alignment; however, its involvement in curricular design remains limited. This separation reinforces a model in which experiential learning is treated primarily as an instructor-driven activity rather than an institutional function. Furthermore, because typical CDCs serve all disciplines within an institution, they are often unable to provide effective, discipline-specific support for CS/IT students [6].

The motivation for this work emerged from the need to reconcile these structural realities. Rather than expanding faculty responsibilities or introducing parallel outreach mechanisms, the partnership model described in this paper leverages existing institutional capacity to support experiential learning. By integrating career services into the capstone structure and other domain-specific outreach approaches, the model helps the department and institution to align educational practice with PASSHE's workforce mission while addressing sustainability concerns inherent in faculty-centric outreach approaches.

3. Collaboration Model Design

The collaborative model is designed around a clear division of responsibility among the CS department and the CDC, such that aggregated capabilities are lined up along existing organizational structures and resources rather than relying on individual faculty initiative.

3.1 Career Development Center

The Career Development Center (CDC) functions as the primary liaison between the university and potential external partners. As a regional institution and for Computer Science specifically, our institution does not yet have the same brand power as other higher ranking institutions in the region. To this end, CDC provides the department with the following services:

- **Customized employer meetup**: Labeled as Lunch-and-Learn, CDC helps the department to invite targeted employers to visit and give an informal talk to students. The talk is a mixture of technical presentation and potential internship/job openings. Criteria for invitation includes number of current employees who are CS alumni, historical institutional collaboration, and regional distances. CDC provides lunches and other logistical support for the event.

- **Outreach to potential clients**: Through their extensive network, CDC helps advertise the department's capstone course to other local and regional businesses. Unlike the meetup participants, the targets of the outreach do not necessarily need to have an interest in hiring our CS students. Rather, CDC helps the department to also connect to those with a need for IT/CS development but without an internal IT team or IT awareness.

3.2 Faculty

Capstone faculty serve as academic designers and technical mentors. Their responsibilities include defining learning outcomes, aligning projects with curricular goals, and supporting students in the technical execution of their work. While faculty do not typically initiate employer outreach, they get in touch with interested clients prior to the semester starts to finalize the scope and scale of the projects. Once the semester starts, the faculty's participation in individual projects can differ depending on the team's capabilities, project's requirements, and clients' interests. Over the semester, we have had cases where clients with a strong internal IT team become more proactive in working with or managing students directly. There are also cases with non-profit organizations where students have taken the lead and become more in-control of the project outcomes.

3.3 Institutional Alignment

By locating employer engagement within an institutional unit, the model aligns with scholarship emphasizing the sustainability of community partnerships when they are

embedded within organizational structures rather than individual relationships [5]. This approach also supports continuity across semesters and reduces dependency on specific faculty members. Although developed within a computer science capstone, the model could potentially become discipline-agnostic. Its core components are transferable to other applied programs and institutions seeking to expand experiential learning opportunities without increasing faculty workload.

4. Capstone Implementation

While Lunch-and-Learn remains students' favorite, the core product of our collaboration model is implemented within a senior-level computer science capstone course structured around client-based projects with external stakeholders. The capstone is positioned as a culminating academic experience in which students are expected to integrate technical knowledge with professional communication, project planning, and client interaction. Rather than treating employer engagement as an external add-on, the outreach pipeline was embedded directly into the course timeline and expectations.

4.1 Before the Semester: Planning and Prep

Prior to the start of each semester, the Career Development Center coordinated with faculty to identify potential local business partners and assess their suitability for student collaboration. When this version of our capstone was first offered in Fall 2024, this outreach and vetting activity was done throughout Summer 2024, with CDC handling the initial introduction and connection, while the faculty followed with project vetting discussion. In the subsequent semesters, both the outreach and vetting activities have been carried out at the end of the semester, where CDC can include previous semesters' successful project showcases in their outreach efforts. We will discuss clients' satisfaction in detail in Section 5. Generally, vetting activities have lessened over time given the amount of returning clients and more structured client intake process.

The vetting process emphasized project feasibility, clarity of expectations, and willingness to engage consistently throughout the semester. Only projects that could be reasonably scoped for a student team with 10 to 12 weeks were introduced into the course. This step proved critical in preventing common capstone failure modes such as overscoped deliverables, ambiguous success criteria, or client disengagement.

4.2 During the Semester: Action and Execution

Our capstone course is typically capped at 40 students. Mirroring the capstone course at Cornell [7], where students are grouped into teams and work on projects curated by the instructor of record, the team size is set to be between six and eight students. This is to encourage larger client projects and to provide optimal team interaction as recommended in Scrum [8]. The students spend the first week learning about **their peers** technical skills and **history on collaborative** projects. Afterward, they will form their own team based on which projects they are interested in. In the case multiple teams are interested in the same project, a decision is made through a first-come-first-serve process, where the first team with full members and an official email statement to the faculty is selected. Students who do not have a team will be assigned to one by the faculty.

Once all teams are matched to projects that were selected, the faculty will make email instruction to the respective clients. This is completed, at the latest, during the second week of the semester. After the introduction, teams will then take the initiative to reach out to the clients to setup the kickoff meetings and subsequent communications. It is emphasized early in the course that students are responsible for ongoing communication with clients, including progress updates and clarification of requirements. At the same time, faculty can be CCed and will serve as an informal external advisor to the students, reinforcing accountability without removing institutional support.

Throughout the semester, the instructional emphasis remained on student ownership of both technical and professional dimensions of the project. Faculty intervention in client relations was intentionally limited, occurring primarily when misalignment threatened learning outcomes or project viability. This balance allowed students to experience authentic external accountability while preserving a safety net consistent with experiential learning principles [1]. Critical checkpoints including a half semester progress report to the class (with the clients are invited to attend) and the four individual progress reflections throughout the semester.

4.3 After the Semester: Reflect and Succeed

The scope of the project is designed to fit within 10-12 weeks of instructions. The last two weeks of the semester are reserved for in-class presentations and for teams to wrap up all deliverables and documentations to their clients. Starting in Fall 2025, we have organized an end-of-semester showcase outside of class, where existing clients, prospective clients and other local/regional partners, and institutional leadership are invited to see a formal showcase events. In this event, the presentations are more business-oriented and less technical, providing students with an opportunity to showcase their work outside of the class. As the semester finishes up, recorded presentations (approved to be released by the clients) and other relevant documentations are posted on a public **-facing** website. Using this information, CDC and the faculty can then prepare for the next round of clients and projects.

5. Observations

Lunch-and-Learn's quantitative measures of success include students' attendance and number of company speakers. As shown in Table 1, we have had increasing number of companies going to the individual Lunch-and-Learn events. Some of these companies attend the university-wide career fair, but some do not, especially those that are pure IT-oriented.

Table 1: Number of companies and students attending each company event per semester

Semester	Students	Company
Spring 2024	18	Deloitte
Spring 2024	9	Pariveda
Fall 2024	26	Deloitte
Fall 2024	20	Blackrock
Fall 2024	19	Pariveda
Spring 2025	15	Deloitte
Spring 2025	7	Blackrock
Fall 2025	13	iPipeline
Fall 2025	16	Blackrock
Fall 2025	14	SAP
Fall 2025	16	Lockhead Martin

For the senior capstone course, the quantitative measures of success include (1) the number of internal/external projects available, (2) average client satisfactory score, and (3) student feedback survey. Measures (1) and (2) and the number of student enrolled in the course over the three consecutive semesters are shown in Table 2. Since the first course offering, the course has been always been full and override requests has been granted with the except of Fall 2025, where override was not possible due to the classroom's physical constraints. Early junior students are encouraged to drop the class and to retake at a later date so that they are more technically prepared for the projects' demands. The full list of project showcases since the beginning of the course can be found at [REDACTED].

5.1 Fall 2024: Pilot

The kickoff of the course consisted of three external client projects. CDC had assisted with the initial outreach, but many clients responded asking for a later follow-up, as our proposed capstone model was new to them. One of the clients was from a company developing research software, and the other two were local non-profits. There were two internal projects: one from the course faculty and another from a faculty in a different department. The range in clients and projects led to very interesting technical and social dynamics among the groups. Different clients exhibit different internal culture, and students adapted to them in different ways. Different projects also force the teams to learn to use different technologies, which can range from cloud computing and AI to web development, hosting, and content management systems.

The Fall 2024 end-of-semester client feedback was generally strong, with high satisfaction on both project outcomes and work quality (average 5.25/6), driven by three clients reporting top-tier satisfaction, and one client reporting a markedly lower experience (3/6). Most clients emphasized professional conduct and reliable communication, noting frequent check-ins, strong responsiveness, and teams that operated independently while still incorporating client input; one client explicitly described the experience as entirely positive and another indicated they would readily hire a student from the team. At the same time, the outlier response surfaced a serious quality-assurance gap, reporting that multiple delivered products were not validated for basic functionality and that the team struggled to adapt as issues emerged. Taken together, the feedback supports the model's strengths in engagement and professionalism while reinforcing the need for tighter readiness checks and explicit testing expectations to reduce the risk of non-functional final deliverables.

Student feedback from the Fall 2024 offering reflects strong perceived value in the authenticity of the capstone experience, alongside predictable friction points in pacing and structure. We requested an unofficial PASSHE Student Feedback on Instructor (SFI) instrument for this semester and received a 74% response rate. From the SFI, overall teaching quality was rated very highly (approximately 5.45/6 based on the response distribution), with the strongest ratings clustering around instructor-student interaction, encouragement of learning, meeting published objectives, and fairness in evaluation. Most individual SFI items showing 90%+ responses in the top two categories. Qualitatively, students repeatedly emphasized that working with real clients and executing a single long-term project felt "industry-focused," practical, and notably free of "busy work," with many framing the course as a meaningful transition into professional expectations. In contrast, critical comments concentrated on course logistics rather than the client-based premise: requests for clearer assignment guidelines, a semester-level timeline to avoid clustered deliverables, and more advance notice for major presentations. Some students also preferred reduced lecture time in favor of project work and client interaction. Several respondents asked for a broader and more technical project pool (and more variety beyond web-centric work), plus more explicit examples of milestones and stronger support on engineering artifacts (e.g., diagrams and best practices).

5.2 Spring 2025: the Second Semester

The short break time between the fall and spring sessions gives little turn around time. Preparations for the spring semester, and potentially inviting fall clients to stay on board, requires a commitment from clients before their fall deliverable is received. For this semester, follow-ups with clients who engaged in the previous semester resulted in three external projects, all from for-profit businesses: one was from a Fall-2024 follow up, one was a returning client,

Table 2: Enrollments, number of projects, and average external client satisfactory scores (1 is lowest and 6 is highest) over three semesters.

Semester	Students	External Projects	Internal Projects	Avg External Client Satisfactory
Fall 2024	27	3	2	5.25
Spring 2025	39	3	4	6
Fall 2025	35	6 (-2)	0 (+2)	5.75

and one was a new client acquired through CDC outreach. There were four internal projects, two of which were from institutional student organizations, one from the capstone faculty, and one self-selected project, where the students picked their own topic.

The end-of-semester client feedback was uniformly positive, albeit from a small pool of respondents (3 clients). All clients reported the highest possible satisfaction ratings for both project outcomes and quality of work (average 6.0/6 on each item). Comments consistently praised the teams' professionalism and reliability. Clients highlighted strong responsiveness, dependable meeting cadence, and a "take-ownership" attitude where students pushed work forward without needing constant supervision. Adaptability also came through as a theme: clients noted that teams adjusted well to midstream changes and technical hurdles, and the delivered products largely matched expectations (with one client still in the process of fully evaluating the final build, but already expressing confidence based on what was delivered). Concerns were minimal, and when mentioned, were framed more as inherent constraints of time and project challenge than as issues with student performance.

For student feedback, we collected informal anonymous responses (a 40% response rate) in lieu of an official SFI. The pattern largely reinforces what students articulated in Semester 1 with a strong endorsement of authenticity, paired with pointed requests for a clearer structure. Students most frequently emphasized that the course was "unlike any other" in the curriculum because it forced real client interaction, team-based delivery, and the shift from "doing homework" to building a product with professional stakes. At the same time, the dominant improvement requests centered on old-fashioned fundamentals: clearer expectations for deliverables and presentations (rubrics/templates/examples), more predictable pacing and early guidance, and a lecture format that feels less disconnected from day-to-day project realities. Compared with Semester 1's open-ended comments, several issues appear to have softened in salience, most notably the earlier anxiety around slow grading/grade visibility and the frustration with cross-institution benchmarking. At the same time, the enduring friction points remain remarkably consistent across cohorts: students want more project options up front (and more technical variety), and they want the capstone's freedom to come with just enough scaffolding to keep teams from reverse-engineering requirements from feedback. A new theme in the Spring 2025 feedback was team accountability (including a detailed complaint about a non-contributing member and requests for more frequent peer reviews), suggesting that as the client-based model stabilizes, students are increasingly focused on collaboration mechanics and professional norms.

5.3 Fall 2025

The strong success of Fall 2024/Spring 2025 allowed for a repackaging of the marketing pitch for CDC to deliver to external clients. The semester started with a full five external clients and one internal non-CS faculty client. The popularity of course had become well known among the student body, with both the roster and the wait list filled up completely on the first day of Spring priority registration. The instructor had to send out an email encouraging early junior students to drop and wait to retake the course at a later semester.

Fall 2025 end-of-semester client feedback remained very strong (4 client responses), with both project outcomes and quality of work averaging 5.75/6 (three clients at 6/6, one at 5/6 on each item). Clients consistently highlighted professionalism and reliability. They explicitly mentioned that teams showed up prepared, maintained steady weekly touch-points, responded promptly, and demonstrated clear ownership across technical areas. One client also noted solid adaptability when requirements evolved (including accommodating new constraints on short notice). Concerns were limited and practical rather than fundamental: a request for more testing, and more standard engineering hygiene around repository organization and build/packaging (e.g., cleaner branching/merging, clearer build configuration). However, we also encountered our first client challenges: two clients had to be dropped from the project list. One client did not account for internal IT resistance, and the student team ended up not getting access to any relevant internal data or tools. Another client did not account for the lackluster responses of the potential users. As a result, this specific project could not rise to the originally anticipated level. In both cases, the student teams learned to courteously communicate the desire to withdraw from the projects, and both teams came up with their own self-motivated project ideas.

Fall 2025 feedback indicates that the capstone model has matured in course execution. On the formal SFI (71.4% response rate), students rated overall teaching quality exceptionally high (92% "Outstanding," yielding an approximate mean of 5.92/6), with similarly strong distributions for interaction, fairness, preparedness, and usefulness of feedback. Qualitative comments in Fall 2025 strongly reinforce the core value proposition seen in Fall 2024 and Spring 2025 informal feedback: students repeatedly de-

scribe the course as the most useful and professionally relevant experience in the program, highlighting autonomy, authentic client accountability, and candid and constructive critique. At the same time, the persistent friction points have shifted from primarily instructor logistics toward capstone system design issues that are harder to fully control, which are client volatility (two projects falling through mid-semester) and uneven responsiveness due to clients' internal work culture, perceived mismatch between lecture topics and project-specific needs, and assessment mechanics that students experience as low-signal (especially quizzes and unclear grade composition). These themes closely track Spring 2025's anonymous feedback calling for clearer rubrics/templates, stronger guidance on client communication/professionalism, more projects earlier, and more frequent peer/accountability mechanisms.

5.4 Overall

Overall, the longitudinal quantitative and qualitative data suggest positive impacts from the CDC-department collaborations. While results from Lunch-and-Learn are harder to track, the external client pipelines made available to the capstone course demonstrated clear improvement in execution and course operations (especially turnaround/feedback and perceived organization). One additional anecdotal piece of evidence from the capstone course is the post-semester student-client interaction. In Fall 2024, one student was hired full-time by one external client. A similar scenario happened in Spring 2025. In Fall 2025, one client offered two students on their team with fellowships to continue expanding the project beyond the scope of the class. Course-specific complaints reflect the inherent trade-off of authentic client-based capstones between maximum realism and autonomy and reliable guardrails.

6. Lessons Learned and Replication Guidelines

Several lessons emerged from implementing this partnership model that may inform adoption at other PASSHE and teaching-focused institutions. First, successful replication depends less on the specific discipline and more on the presence of an institutional unit capable of managing employer relationships. Career Development Centers are particularly well-suited for this role due to their existing networks, professional expertise, and alignment with workforce preparation goals.

Second, client vetting is essential. Not all well-intentioned employers are prepared to support student projects, and early screening prevents downstream challenges that can undermine learning outcomes. Clear documentation of expectations, timelines, and communication norms proved more effective than informal agreements.

Third, faculty roles must be carefully scoped. When instructors function simultaneously as technical mentors, project managers, and client liaisons, sustainability suffers. Deliberately limiting faculty responsibility for employer outreach allowed instructional effort to focus on mentoring student learning rather than maintaining external relationships.

Finally, institutions should recognize that client-based capstones are not substitutes for internships but complementary experiences. Capstone projects offer structured, faculty-guided engagement with external stakeholders while lowering barriers for small and mid-sized employers. This distinction broadens access to experiential learning opportunities and aligns with high-impact practice frameworks that emphasize equity and participation [2].

7. Conclusion

This paper presented a practice-based case study of an institutional partnership model that integrates Career Development Center and the Computer Science department at a PASSHE institution. By facilitating department-specific events and formalizing employer outreach and client preparation through existing campus infrastructure, the model reduces faculty workload, improves project feasibility, and strengthens connections between academic programs and regional workforce needs. The findings suggest that institutionalizing external engagement supports sustainable experiential learning in resource-constrained environments. While the model was implemented within a computer science department, its core principles are adaptable across disciplines that emphasize applied learning and professional readiness. Future work may examine longitudinal outcomes related to student placement, employer retention, and cross-departmental adoption. A survey of potentially related models at other similar schools could would also be beneficial to the research community. Nevertheless, this study demonstrates that meaningful experiential learning can be achieved at regional public universities through intentional collaboration, clear role definition, and alignment with institutional mission.

References

[1] David A. Kolb. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice Hall, Englewood Cliffs, NJ, 1984.

[2] George D. Kuh. *High-Impact Educational Practices: What They Are, Who Has Access to Them, and Why They Matter*. Association of American Colleges and Universities, Washington, DC, 2008.

[3] Megan Elrath, Joseph Hawk, and Nancy LeClair. Professional development for college students in tough economic times: The drexel university co-op program model. *Academic Leadership: The Online Journal (2003-2012)*, 8(4):51, 2010.

[4] Robert G. Bringle and Julie A. Hatcher. Implementing service learning in higher education. *The Journal of Higher Education*, 67(2):221–239, 1996.

[5] Lorilee R. Sandmann, Charlotte H. Thornton, and Audrey J. Jaeger. Institutionalizing community engagement in higher education. *Journal of Higher Education Outreach and Engagement*, 13(2):3–32, 2009.

[6] Joseph Packy Laverty, Christine Laverty, and David Wood. Improving the internship and career search process for is, cs, and it students. In *Proceedings of the Information Systems Educators Conference ISSN*, volume 2167, page 1435. Citeseer, 2012.

[7] Cornell University. CS 5150: Software Engineering (Spring 2025). `https://www.cs.cornell.edu/courses/cs5150/2025sp/`, 2025. Accessed: 2025-02-05.

[8] Ken Schwaber and Jeff Sutherland. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Scrum.org, Nov 2020.

Abstracts of the Other Presentations

Driving Efficiency & Innovation: Strategic AI Implementation for Professional Growth

Naresh Adhikari

naresh.adhikari@sru.edu

Slippery Rock University

Artificial Intelligence (AI) is rapidly reshaping the modern workplace, redefining how organizations operate and how professionals create value. *Driving Efficiency & Innovation: Strategic AI Implementation for Professional Growth* explores how purposeful integration of AI technologies can enhance operational efficiency, accelerate innovation, and elevate individual career paths. Rather than viewing AI as a standalone tool, this *interaction* emphasizes a strategic framework that aligns AI adoption with organizational goals, governance standards, and measurable return on investment (ROI). By advancing AI literacy, redesigning workflows, and embedding responsible oversight, leaders and employees can transition from task-based execution to higher-order decision-making and strategic contribution. The expected result is a transformed workplace where human expertise and intelligent systems collaborate to improve productivity, strengthen competitiveness, and foster sustainable professional growth

REVEALING HIDDEN PATTERNS: A DATA-DRIVEN ANALYSIS OF DECEPTIVE ODDS IN WORDSCAPES

Brandon Packard (Presenting), Daniel Shifflet
PennWest - Clarion
Bpackard@pennwest.edu, dshifflet@pennwest.edu

ABSTRACT
Video game mechanics range from incredibly simple to incredibly complex. One word game that is rather simple, Wordscapes, is also incredibly addictive. In addition to the main gameplay of creating words from a set of letters, there are also multiple minigames to play. One of these, Mt. Fortune, allows the player to pick a random card from a set of cards. If that card is a reward, the player can take their rewards and go or move on. However, there are also penalty cards that take away all of the player's rewards unless they pay in-game currency, making the game a balance of risk versus reward. As there are 4 cards per level and the game always reveals one of them to be bad after the player chooses, the player is lead to believe that there's always a 25% chance of hitting a penalty card. However, after collecting data from over 200 runs of the minigame, we purport to show that not only are the odds of hitting a penalty card not 25%, but that the odds quietly shifted on the back end on July 25th, 2024, leading to the player hitting more mines and spending more in-game currency.

KEY WORDS
Data Analysis, Statistics, Wordscapes, Probability

VIDEO GAMES AS LITERATURE

Brandon Packard
PennWest - Clarion
bpackard@pennwest.edu

ABSTRACT
In many ways, video games have been a topic of hot debate in today's culture. One of the most often discussed topics is whether video games should be considered art. In this talk, we will instead be focusing on video games as literature. Our aim with this talk is to show that games can hold their own in the world of literature. To this end, we discuss in-depth some of the games which have the story-telling prowess to be considered works of literature, as well as some of the elements video games use to tell stories that would be impossible for more classic literature such as books or essays.

KEY WORDS
Video games, literature, storytelling

WHODIS? DECENTRALIZED IDENTITY SYSTEMS

Peter Schaefer, Dr. Yong Zhang
Kutztown University
pscha710@live.kutztown.edu, zhang@kutztown.edu

ABSTRACT

In the world of massive digital storage silos, breaches expose millions of personal identities and documents. In such centralized systems, individuals have little to no control over how their data is collected, stored, or shared. Central authorities act as gatekeepers and single sources of truth, concentrating power and risk. Decentralized Identity Systems provide a unique opportunity of user-owned digital identities through digital technologies such as public key cryptography, hashes, digital signatures, blockchains, and zero knowledge proofs. Such systems solve the risks of centralized identity systems through the distribution of trust, the elimination of centralized data silos, and allow individuals to own and control their identity credentials directly.

This presentation covers how decentralized identity systems are able to securely distribute trust, minimize data centralization, and put control into individuals about their own credentials by covering how fundamental concepts of public key cryptography, hashes, digital signatures, blockchains, and zero knowledge proofs are able to work together to provide a secure decentralized system for managing identification credentials. Useful analogies, combined with programming examples enable those from both technical and non-technical backgrounds to gain useful insight into the operation of a decentralized identity system. Finally, an existing standard is covered, demonstrating how these technologies are already being applied in the real world.

KEY WORDS

public key cryptography, digital signatures, blockchain technologies, zero knowledge proofs, data security

Abstracts of Birds of a Feather