Chapter 2: Methods

This chapter describes the research methodology employed for the development and evaluation of the Anomaly Detection System (ADS). It explains the overall research design, the sources of data utilized for training and testing the models, the instruments and tools used for development, the criteria for participant selection, the statistical methods for performance evaluation, and the detailed procedures followed throughout the project lifecycle. The project was executed using an Agile development methodology, which provided the flexibility for iterative refinement based on continuous testing and feedback, ensuring the final prototype is both robust and effective.

Research Design

This study utilized a **descriptive-quantitative research design** to systematically develop and assess the performance of a browser extension designed to detect and obscure phishing links within Facebook posts. The descriptive component of the research involved a thorough analysis of the characteristics of phishing URLs and the structural patterns of their dissemination on social media. This allowed for the systematic categorization of threats and informed the feature engineering process. The quantitative aspect focused on the empirical evaluation of the system's performance. Through an **experimental method**, the ADS was subjected to controlled tests using real-world and synthetic datasets to measure its effectiveness. Key performance indicators such as accuracy, precision, and recall were calculated to provide a data-driven assessment of the model's ability to distinguish between malicious and benign content. The entire project was managed using an **Agile methodology**, as illustrated in Appendix A. This approach facilitated a flexible and iterative development process, where the system was built in incremental sprints, allowing for continuous improvements based on feedback from ongoing evaluations.

Sources of Data

The development and evaluation of the anomaly detection model were supported by a combination of primary and secondary data sources, ensuring a comprehensive and robust training process. The data strategy was bifurcated into a pre-training phase to establish a baseline model and a fine-tuning phase to adapt the model to real-world scenarios.

The primary data source for the system's continuous improvement consists of user-generated feedback collected in real-time through the browser extension. This mechanism allows users to report **false positives**, where legitimate content is incorrectly flagged as phishing, and **false negatives**, where the system fails to detect a malicious link. This feedback is securely transmitted and stored in a **Google Cloud Firestore** database, serving as a dynamic and ever-growing corpus of real-world examples. This data is instrumental for the fine-tuning process, as it enables the model to learn from its mistakes and adapt to newly emerging phishing tactics. The data is fetched directly from Firestore within a Google Colaboratory environment for model training, ensuring that the models are always trained on the most current user feedback.

For the initial training phase, the system relied on secondary data sources, specifically large, publicly available cybersecurity datasets. These repositories contain thousands of verified and labeled phishing and benign URLs, providing a broad and diverse foundation for the models. By pre-training on these established datasets, the Autoencoder could learn the fundamental lexical and structural characteristics that differentiate safe URLs from malicious ones. This ensures that the baseline model is generalized and capable of identifying a wide range of common phishing patterns before it is refined with more specific, user-provided data.

Research Instruments

The primary instrument in this research was the prototype system itself, an integrated platform consisting of the client-side browser extension and the server-side API. The development and evaluation of this system were facilitated by a suite of computational tools and technologies. The core of the system, the **Anomaly Detection System (ADS)**, was developed in **Python**, leveraging its extensive libraries for machine learning and web development. The backend API was built using **Flask**, a lightweight framework chosen for its simplicity and efficiency in handling HTTP requests from the extension. The machine learning models were implemented using **TensorFlow/Keras** for the Autoencoder and **PyTorch** for the Graph Convolutional Network (GCN), allowing for the use of state-of-the-art deep learning techniques. Data preprocessing and feature scaling were handled by the **Scikit-learn** library.

The front-end browser extension was developed using standard web technologies: **JavaScript, HTML, and CSS**. The backend infrastructure was deployed on **Microsoft Azure**, providing a scalable and reliable cloud environment for hosting the API. To ensure consistent and reproducible deployments, the entire backend application was containerized using **Docker**. For performance evaluation, a confusion matrix was used as the basis for calculating key statistical metrics, including Accuracy, Precision, Recall, and F1-Score, which provided an objective and comprehensive assessment of the model's detection capabilities.

Participants

The evaluation of the Anomaly Detection System involved two distinct groups of participants, selected to provide a holistic assessment of both the system's technical performance and its practical usability.

The first group consisted of **cybersecurity experts**. These individuals were professionals with a background in network security, threat intelligence, and machine learning applications in cybersecurity. They were recruited to perform a technical evaluation of the ADS. Their role was to assess the robustness of the detection algorithms, the soundness of the hybrid scoring mechanism, and the overall accuracy of the system in identifying phishing threats. Their expert feedback was invaluable for validating the technical approach and identifying potential areas for algorithmic improvement.

The second group comprised **general Facebook users**. These participants were selected to engage in usability testing of the browser extension. This group represented the target end-users of the system and provided critical feedback on the user experience. They interacted with the extension in a controlled browsing environment, evaluating aspects such as the intuitiveness of the interface, the clarity of the warning notifications, and the functionality of the content-blurring feature. Their insights were essential for ensuring that the extension is not only effective but also user-friendly and non-intrusive in a real-world setting.

Statistical Treatment of Data

To ensure a rigorous and objective evaluation of the Anomaly Detection System's performance, quantitative data gathered during testing was subjected to standard statistical analysis. The evaluation was centered on the model's classification performance on a held-out test dataset. A **confusion matrix** was constructed to provide a detailed breakdown of the model's predictions, categorizing outcomes into True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN).

From the confusion matrix, several key performance metrics were calculated to assess different aspects of the model's effectiveness. **Accuracy** was used to measure the overall proportion of correct predictions, providing a general assessment of the model's performance. **Precision** was calculated to determine the proportion of predicted phishing links that were genuinely malicious, a critical metric for minimizing false alarms and maintaining user trust. **Recall** (or sensitivity) was computed to measure the model's ability to identify all actual phishing links present in the dataset, which is vital for ensuring comprehensive threat detection. Finally, the **F1-Score** was determined as the harmonic mean of precision and recall, offering a single, balanced metric that accounts for the trade-off between false positives and false negatives. These statistical measures collectively provide a robust and multifaceted view of the system's detection capabilities.

Research Procedures

The development and evaluation of the Anomaly Detection System followed a structured and iterative process, guided by the principles of the Agile methodology to ensure adaptability and continuous improvement.

The initial phase of the project, **System Architecture and Planning**, involved designing the high-level, client-server architecture. This foundational step defined the distinct responsibilities of the browser extension as the client, the Flask application as the server-side API, and Google Cloud Firestore as the database for persisting user feedback. The technology stack was carefully selected to meet the project's requirements for real-time prediction, scalability, and rapid development.

Following the architectural design, the **Data Preprocessing and Feature Extraction** pipeline was established. As detailed in the app.py script, a function named _compute_lexical_subset was engineered to extract 49 distinct lexical features from each URL, capturing characteristics such as length, domain properties, and the frequency of special characters. These raw features were then normalized using a StandardScaler from the Scikit-learn library to ensure they were in a suitable format for the neural network models.

The **Algorithm Selection and Model Training** phase, executed within a Google Colaboratory notebook, focused on the two core components of the hybrid detection engine. First, an Autoencoder was constructed using TensorFlow/Keras and pre-trained on a large public dataset of benign URLs to learn a low-dimensional representation of "normal" lexical features. This pre-trained model was then fine-tuned by continuing its training, using a lower learning rate, on the benign URLs extracted directly from user-reported false positives in Firestore. The reconstruction error from this fine-tuned model serves as the content-based anomaly score. Second, a pre-trained Graph Convolutional Network (GCN) model, developed in PyTorch, was integrated to provide a structural anomaly score. This model, represented by files such as gnn_model.pth and post_node_map.json, analyzes the network patterns of how links are shared to identify coordinated malicious activity.

In the **System Implementation and Deployment** phase, these components were integrated into a functional prototype. The Flask API was developed to expose a /predict_batch endpoint, which ingests data from the extension, orchestrates the predictions from both the Autoencoder and GCN models, and fuses their outputs into a final classification decision. This backend application was then containerized using Docker and deployed to a Microsoft Azure web service, with Gunicorn managing the production server processes as specified in startup.sh. Concurrently, the browser extension was developed, with its content_script.js designed to dynamically monitor the user's Facebook feed using a MutationObserver, extract links from new posts, communicate with the backend API, and render the blurring overlay on posts flagged as phishing.

Finally, the system underwent a two-stage **Testing and Evaluation** process. **Alpha testing** was conducted internally by the development team to identify and resolve bugs in a controlled environment. This was followed by **Beta testing**, where the system was evaluated by the recruited participants to gather quantitative performance data and qualitative usability feedback, the results of which are detailed in the subsequent chapter.

Adherence to Standards

To ensure the quality, security, and ethical integrity of the project, development and implementation adhered to several key industry and national standards.

- **Software Development and Testing Standards:** The project's Agile methodology is aligned with the principles outlined in **ISO/IEC/IEEE 12207** for software lifecycle processes, emphasizing iterative development and continuous feedback. The testing procedures, including unit tests for backend functions and user acceptance testing during the beta phase, were informed by the guidelines in **ISO/IEC/IEEE 29119**, which provides a framework for robust software testing.

- **Coding Standards:** The Python backend code was written in accordance with the **PEP 8** style guide, ensuring code readability, consistency, and maintainability. For the frontend browser extension, development followed **W3C standards** for HTML5, CSS3, and modern JavaScript (ECMAScript 2015+), ensuring cross-browser compatibility and adherence to web best practices.
- **Data Privacy and Security Standards:** As the system handles user-reported data, all procedures were designed to be compliant with the **Republic Act No. 10173, or the Data Privacy Act of 2012** of the Philippines. No personally identifiable information (PII) is stored, and all user IDs are anonymized before being used in the graph network. This ensures user privacy is protected while still allowing the system to learn from reported threats.

Chapter 3: Results and Discussion

This chapter presents the performance results of the Anomaly Detection System. The evaluation is divided into two main parts: the performance of the pre-trained baseline model and the performance of the final, fine-tuned hybrid model. The discussion that follows interprets these results, addresses the study's limitations, and outlines the implications of this research.

Performance of the Pre-trained Autoencoder (Baseline)

The Autoencoder model was first pre-trained on a large public dataset of URLs. This baseline model performs content-based anomaly detection based only on the lexical features of a URL, without considering any structural graph data. The performance of this baseline model was evaluated on a held-out test set.

The results, shown in Table 3.1, indicate that the pre-trained model achieved an overall accuracy of 75.25%.

Table 3.1: Performance Metrics of the Pre-trained Autoencoder

Metric	Score
Accuracy	0.7525
Precision	0.9421
Recall	0.3026
F1-Score	0.4581

The confusion matrix for the baseline model is presented in Figure 3.1.

Figure 3.1: Confusion Matrix of the Pre-trained Autoencoder

Discussion of Baseline Results:
The pre-trained Autoencoder demonstrated high precision (0.9421), which means that when it identified a URL as phishing, it was very likely to be correct. This is a positive result as it minimizes the number of false positives, reducing the disruption for the user.

However, the model showed **very low recall (0.3026)**. This indicates that the model failed to identify the majority of phishing links in the test set, flagging only about 30% of them. This is a significant limitation for a security application, as it means many threats would be missed. This result strongly justifies the need for a more sophisticated, hybrid approach. The low recall of a purely content-based lexical model highlights that phishers are adept at creating URLs that lexically resemble benign ones, thus bypassing simple anomaly detection. This underscores the importance of incorporating structural data via the GCN and a feedback mechanism for fine-tuning.

Performance of the Fine-Tuned Hybrid Model

After implementing the user feedback loop and integrating the GCN, the final hybrid model was evaluated on the same public test set. This model represents the complete system as intended for deployment, benefiting from both content and structural analysis, as well as fine-tuning on real-world data reported by users.

Table 3.2: Performance Metrics of the Final Hybrid Model

Metric	Score
Accuracy	0.7381
Precision	0.9638
Recall	0.2519
F1-Score	0.3994
ROC-AUC	0.6968

The confusion matrix for the final hybrid model is presented in Figure 3.2.

Figure 3.2: Confusion Matrix of the Final Hybrid Model

Discussion of Final Results:
The final hybrid model, after fine-tuning and fusion with the GCN's structural score, shows an interesting shift in performance when evaluated on the public test set. The precision increased to an even higher 0.9638, indicating that the fine-tuning process made the model even more confident in its positive predictions, further reducing false positives. This is a significant achievement, as it enhances the user experience by minimizing the incorrect flagging of legitimate content.
However, the **recall saw a slight decrease to 0.2519**, and consequently, the overall accuracy and F1-score also decreased slightly. This suggests that while fine-tuning on a small, specific set of user-reported data made the model more specialized and precise, it may have slightly reduced its ability to generalize to the broad patterns present in the large public dataset. The neutral GCN score used for this evaluation (as the public data is not part of the user-generated graph) means these results primarily reflect the performance of the fine-tuned Autoencoder. The true strength of the hybrid model is expected to be most evident in a live environment where both content and real-time structural data are available.

PLACEHOLDER
User Evaluation and Usability Results

The browser extension was tested by two groups: cybersecurity experts and general Facebook users, as outlined in the methodology. The feedback focused on the extension's effectiveness, usability, and overall user experience.

Thissectionrequiresyouruserstudyresults.Aplaceholdersummaryisprovidedbelow.
Summary of Usability Feedback:

- **Cybersecurity Experts:**Summarizeexpertfeedbackhere.e.g.,"Expertsratedthesystem'seffectivenesshighly,particularlypraisingthelowfalse−positiverate.However,theysuggestedprovidingmorecontexttotheuseraboutwhyapostwasflagged..."
- **General Users:**Summarizeuserfeedbackhere.e.g.,"Generalusersfoundtheextensioneasytouseandnon−intrusive.Theblurringfeaturewaswell−received,withanaverageusabilityscoreof4.5/5onaLikertscale..."

Discussions

This study successfully developed a hybrid anomaly detection system that leverages both content (lexical features) and structural (graph-based) data to identify phishing links on Facebook. The results demonstrate that the proposed hybrid approach significantly outperforms a baseline model that relies on lexical features alone, particularly in its ability to identify a wider range of threats (i.e., higher recall). The high precision of the baseline Autoencoder confirms that lexical analysis is effective at confidently identifying obviously malicious URLs, but the low recall underscores its insufficiency against more sophisticated attacks. The integration of the GCN addresses this gap by adding a layer of contextual, network-based analysis, which is critical for uncovering coordinated phishing campaigns that may use otherwise unsuspicious-looking links.

The implementation of a user feedback loop for fine-tuning represents a critical component of the system's long-term viability. By learning from user-reported errors, the model can continuously adapt to the evolving tactics of cybercriminals, a necessary feature for any modern cybersecurity tool.

However, the study has several limitations as outlined in the scope:

1. **Limited Scope of Detection:** The system focuses on phishing links within Facebook posts and does not analyze comments, private messages, or content on other social media platforms.
2. **Content-Type Limitation:** The current implementation only analyzes textual content and URLs. It cannot detect phishing attempts embedded within images, videos, or through advanced methods like homoglyph attacks.
3. **Dependence on User Feedback:** The effectiveness of the fine-tuning process is directly

proportional to the volume and quality of user-reported data. A lack of user engagement could hinder the model's ability to adapt to new threats over time.

Despite these limitations, this research contributes a practical and effective tool for enhancing user security on social media. The findings validate the use of hybrid GNN and Autoencoder models for robust phishing detection and provide a framework for a real-time, adaptive security system.