VCU College of Engineering

Project #025-345 Network Feature Extraction from Traffic Captures to Support Automation of High-Fidelity Cyber Simulation Environment

Final Design Report

Prepared for

Jae Sung Kim

Department of Defense ASPIRE

By

Christopher Castro, Ryan Collette, William Lagos, Sam Soltanian

Under the supervision of

Irfan Ahmed

05/01/2025

Executive Summary

This document outlines an OS device classification system that utilizes machine learning. Rather than focusing on real-time breach detection, this system passively collects and analyzes web traffic, classifying operating systems within the network. By employing ML-driven classification through network feature extraction, we aim to provide enhanced visibility into device behavior, addressing gaps in both research and practical implementation of consumer-level software for OS identification. With wireless enabled devices becoming increasingly prevalent across consumer and critical environments, understanding network composition and behavior is crucial for anticipating and managing potential vulnerabilities. This project focuses on creating a scalable, easily replicable software solution capable of classifying devices on a network with minimal setup, operating as a passive extension to the router to streamline device monitoring.

Table of Contents

Section A. Problem Statement

OsirisML is a machine learning model designed to identify the operating system origin of a network request. As network complexity increases and organizations around the world fail to update their systems, we are left with a wide range of possible candidates for where a request originated from. The ability to identify these operating systems using passive network monitoring can prove to be a useful ability in modern warfare. Acquisition of these details provides us with a massive advantage in the enumeration phase while remaining undetected.

By leveraging passive data collection and applying ML classification models, we can extract significant features from TCP IP packets that enable a comprehensive classification system for identifying devices based on network behavior. This non-intrusive model allows us to monitor without alerting potential malicious actors, maintaining a low profile while gathering essential data. This implementation additionally allows for a massive range of expansion in terms of deployment, usage of data acquired from this enumeration phase, and efficiency improvements to current implementations.

Current research in operating system classification implementations remains underexplored, especially in high-fidelity environments where such devices are prevalent. Implementing a low-cost, deployable model that runs on accessible consumer grade hardware allows for easy reproductivity. This model could be seamlessly introduced into a military environment with minimal setup, allowing for a broad scope of traffic capture across various sectors.

Practical challenges include ensuring proper classification of not only operating system of origin, but also release version of operating system. Larger scale machine learning models are computationally intensive and must be both fine tuned and have selected features of most importance to optimize performance.

This approach paves the way for advanced network device mapping and monitoring, enriching our ability to understand and manage the dynamic composition of multi-platform environments across conflict and civilian zones alike.

Section B. Engineering Design Requirements

B.1 Project Goals (i.e. Client Needs)

- **Enhance Existing Model Functionality:**
    - Improve the current OS fingerprinting model to accurately identify not just the operating system type but also specific builds and versions from passive network traffic.
- **Increase Model Accuracy and Robustness:**
    - Utilize underexplored machine learning techniques to enhance the model's performance.
- **Streamline Deployment and Usability:**
    - Simplify the setup, usage, and deployment of the model.

B.2 Design Objectives

- **Improve Model Accuracy:**
    - Implement k-fold cross-validation to enhance model robustness.
    - Perform extensive hyperparameter tuning using advanced optimization techniques such as GridSearch or RandomSearch.
- **Optimize Feature Selection and Model Decision Making:**
    - Utilize a bucket sorting multi-model structure with a general model to make classification decisions and submodels for fine-tuned version identification.
- **Expand OS Detection Capabilities:**
    - Extend the model's functionality to identify specific OS builds and versions.

B.3 Design Specifications and Constraints

- The model will retain at least the same level of accuracy or higher in classifying the original test set. (Functional constraint).
- The model will use K-fold cross validation to improve accuracy (Functional constraint)
- A user will be able to interact with the model using a command line tool (Functional constraint).
- The model will utilize a gradient boosting algorithm based library (Functional constraint).

B.4 Codes and Standards

- **IEEE P7001**: Transparency in AI—document the decision-making process of the AI.

Section C. Scope of Work

Project Timeline:

- **November - December 2024:**
  - **Finalize team roles, project objectives, and deliverables.**
  - **Conduct initial research and begin drafting the preliminary design report.**
  - **Receive the new dataset from the sponsor and perform initial data preprocessing.**
- **January - February 2025:**
  - **Model development with k-fold cross-validation.**
  - **Extensive hyperparameter tuning using advanced optimization techniques.**
- **March 2025:**
  - **Model feature selection and accuracy tuning.**
  - **Development of submodels.**
- **April 2025:**
  - **Documentation and user guides development.**
  - **Capstone EXPO preparation, including final reports and presentations.**

We will be following Agile methodology for development, so we can ensure each component is thoroughly tested and improved. This allows for flexibility and continuous feedback from the project sponsor and faculty advisor.

C.1 Deliverables

The project will produce the following deliverables:

1. **Enhanced Machine Learning Model:**
   - **A trained model capable of detecting specific OS minor versions and builds by using a general model and submodels for in depth classification using the new dataset provided by the sponsor.**
   - **Implementation of k-fold cross-validation to improve model robustness.**
   - **Extensive hyperparameter tuning, including exploration of boost types, max_depth, gamma, learning rate, eta, and others.**
   - **Improved performant model and submodels capable of predicting the operating system of origin with a level of accuracy exceeding the original state of the project or similar.**

Potential Risks and Mitigation:

- **Data Quality and Availability:**
  - ***Risk:*** **The new dataset may have limitations or require additional preprocessing.**
  - ***Mitigation:*** **Close collaboration with the sponsor to understand the dataset and adjust preprocessing steps accordingly.**
- **Computational Resource Constraints:**
  - ***Risk:*** **Hyperparameter tuning and k-fold cross-validation may be computationally intensive.**
  - ***Mitigation:*** **Utilize university computational resources or cloud services.**
- **Time Management:**
  - ***Risk:*** **Potential delays due to unforeseen technical challenges.**
  - ***Mitigation:*** **Regular progress reviews and agile adjustments to the project plan.**

C.2 Milestones

Project Milestones Timeline:

Finalize Team Roles, Project Objectives, and Deliverables

- **Tasks:**
  - Finalize team roles, project objectives, scope, and deliverables.
  - Submit the project proposal to VCU.
  - Prepare and submit the fall design poster.
- **Expected Completion: November 15, 2024**

---

2. Resource Procurement and Research

- **Tasks:**
  - **Resource Procurement:**
    - Contact VCU IT services to procure a high-performance virtual machine (VM) with at least 128 GB of RAM, accessible by all team members.
      - If VCU IT is unable to provide, research other alternatives such as AWS EC2 or GCP.
  - **Research:**
    - Conduct in-depth research on:
      - Advanced machine learning techniques for OS version detection.
      - Feature extraction methods relevant to OS fingerprinting and distinguishing specific OS versions.

        - K-Fold Cross-Validation and its impact on improving model accuracy and robustness.
        - Hyperparameter optimization techniques, including tools like GridSearchCV, RandomizedSearchCV, and Optuna.
        - Boosting algorithms and their variants to understand how different boost types affect model performance.
    - Begin drafting the **Preliminary Design Report**.
- **Expected Completion: November 29, 2024** (2 weeks)

---

3. Getting Model to Run Locally

- **Tasks:**
    - Set up development environments on procured machines.
    - Adjust existing scripts to accommodate the new dataset structure.
    - Ensure the initial model runs successfully on local machines using a subset of the data.
    - Begin initial data preprocessing and feature extraction.
    - Continue working on the **Preliminary Design Report** and complete it.
- **Expected Completion: December 13, 2024** (2 weeks)

---

4. Data Preprocessing and Feature Extraction

- **Tasks:**
    - Perform comprehensive data preprocessing and feature extraction, including:
        - Adjusting preprocessing scripts to fully accommodate the new dataset structure.
        - Extracting features relevant to specific OS version detection.
        - Cleaning and normalizing the data to prepare it for model training.
    - Validate the processed data and ensure it's ready for model development.
- **Expected Completion: December 27, 2024** (2 weeks)

---

5. Model Development with K-Fold Cross-Validation

- **Tasks:**
    - Implement **k-fold cross-validation** in the training pipeline to enhance model robustness.
    - Train initial machine learning models using the preprocessed dataset.
    - Analyze model performance metrics to identify areas for improvement.
    - Document initial findings and update the design report accordingly.

- **Expected Completion: January 31, 2025** (5 weeks)

---

6. Extensive Hyperparameter Tuning

- **Tasks:**
  - Perform comprehensive hyperparameter tuning using advanced optimization techniques.
  - Explore and optimize parameters such as:
    - **Boost Types** (e.g., gbtree, gblinear, dart).
    - **Max Depth**.
    - **Gamma**.
    - **Learning Rate (Eta)**.
    - **Min Child Weight**.
    - **Subsample** and **Colsample_bytree**.
  - Utilize tools like **Optuna** or **Hyperopt** for efficient hyperparameter search.
  - Record the optimal hyperparameters and their impact on model performance.
  - Update documentation with tuning results and insights.
- **Expected Completion: February 28, 2025** (4 weeks)

---

7. Documentation and User Guides

- **Tasks:**
  - Prepare comprehensive technical documentation, including:
    - Installation and setup instructions.
    - Detailed usage guides for all application features.
    - Developer documentation for future maintenance and enhancements.
  - Develop user manuals with examples and best practices.
  - Update the design report with finalized designs, methodologies, and findings.
- **Expected Completion: April 18, 2025** (1 week)

---

8. Capstone EXPO Preparation

- **Tasks:**
  - Prepare for the Capstone EXPO by:
    - Creating the poster presentation.
    - Writing the final project report.
    - Developing presentation materials and rehearsing the presentation.
  - Seek feedback from the project sponsor and faculty advisor to refine deliverables.
- **Expected Completion: April 25, 2025** (1 week)

C.3 Resources

Resources needed for project completion should be listed at the proposal stage. These resources can either be purchased within the Project Budget, or provided by the project sponsor. Some examples are: hardware such as HPCs or servers, software such as IDEs, data analysis platforms or version control systems. Access to cloud computing services may also be necessary to scale certain procedures. Additionally, databases containing operational data for testing, as well as libraries or APIs relevant to predictive analytics and machine learning may be required.

Hardware:

- **Computational Resources:**
    - **High-Performance Computing (HPC) Clusters:**
        - Access to university-provided HPC clusters or servers for computationally intensive tasks such as extensive hyperparameter tuning and k-fold cross-validation on large datasets.
        - Necessary during the **Extensive Hyperparameter Tuning** phase and when training models with k-fold cross-validation to handle increased computational load.

Software:

- **Programming Languages and Development Tools:**
    - **Python 3.x:**
        - The primary programming language for all development tasks.
        - Utilized extensively across all project phases.
    - **Integrated Development Environments (IDEs):**
        - **PyCharm**, or **Visual Studio Code** for code development and debugging.
    - **Version Control Systems:**
        - **Git** for version control.
        - **GitHub** for code hosting, collaboration, and issue tracking.

Machine Learning and Data Processing Libraries:

- **XGBoost:**
    - For model training, evaluation, and implementing advanced boosting algorithms.
    - Central to the **Model Development** and **Hyperparameter Tuning** phases.
- **Scikit-learn:**
    - For implementing k-fold cross-validation, initial model development, and basic machine learning tasks.
    - Used during the **Model Development with K-Fold Cross-Validation** phase.
- **Optuna** or **Hyperopt:**

    - Advanced hyperparameter optimization frameworks to efficiently search for optimal model parameters.
    - Employed during the **Extensive Hyperparameter Tuning** phase.
- **Pandas and NumPy:**
    - For data manipulation, preprocessing, and analysis.
    - Essential during the **Data Preprocessing** and throughout model development.
- **Scapy** or **Tshark** (if needed):
    - For advanced packet analysis and feature extraction from PCAP files.
    - May be required during **Dataset Acquisition and Data Preprocessing** if additional feature extraction is necessary.

Command-Line Interface (CLI) Development Libraries:

- **argparse**, **click**, or **typer:**
    - For building a user-friendly CLI application.
    - Integral to the **CLI Application Development** phase.

Testing Frameworks:

- **PyTest**:
    - For writing and running unit tests.
    - Employed during the **Containerization and System Integration** phase.

Data Resources:

- **Labeled Dataset Provided by the Sponsor:**
    - A new dataset containing network traffic data with specific OS version labels for training and testing the machine learning models.
    - Central to the **Dataset Acquisition and Data Preprocessing** and subsequent modeling phases.

Compute Services:

- **Cloud Computing Platforms (if required):**
    - **Amazon Web Services (AWS) EC2**, **Google Cloud Platform (GCP)**, or **Microsoft Azure:**
        - For scalable computing resources, especially during extensive hyperparameter tuning and model training.
        - Backup option if university HPC resources are insufficient or unavailable.

Section D. Concept Generation

The primary objective of these concepts is to improve the accuracy of the existing Machine Learning model used to identify operating systems through network feature extraction. By exploring different avenues for feature selection, dimensionality reduction, and model optimization, we aim to enhance the model's ability to analyze and classify network traffic.

Concept 1: Feature Importance Analysis with XGBoost

Use XGBoost to identify the most important features contributing to the model's accuracy. By focusing on high-impact features, we can streamline the model and potentially reduce noise caused by irrelevant or redundant features.

How it addresses the problem:

- Pinpoints features that significantly affect classification accuracy.
- Helps reduce overfitting by limiting the feature space.

Pros:

- XGBoost is well-documented and has built- in support for feature importance scoring.
- Can be directly integrated into the existing model workflow.

Cons:

- Requires fine-tuning of hyperparameters for optimal performance.

Potential Risks:

- Important features may still interact in ways that XGBoost alone can't capture.

Concept 2: K-Fold Cross Validation with PCA Variable Selection

Integrate Principal Component Analysis to reduce dimensionality, followed by k-fold cross validation to validate the model on multiple splits of the data. This ensures robust performance and helps evaluate how well the model generalizes.

How it addresses the problem:

- Reduces dimensionality, simplifying the model and reducing computational overhead.
- Provides more robust and less variance-prone accuracy estimates.
- Efficiently uses all available data by training and evaluating on the entire set.

Pros:

- PCA highlights key latent variables that may improve model interpretability.

Cons:

- PCA can sometimes remove variables that have nonlinear contributions to the output.

Potential Risks:

- Risk of underfitting if too much information is lost during dimensionality reduction.

Concept 3: Model Training with CatBoost

Implement CatBoost as an alternative to the existing model, taking advantage of its ability to handle categorical features natively and reduce overfitting through advanced regularization techniques.

How it addresses the problem:

- CatBoost can capture complex interactions in data that other algorithms might miss.
- CatBoost is especially effective with noisy datasets, common in network traffic analysis.

Pros:

- Faster training time compared to some other ensemble methods.

Cons:

- Original project was built with XGBoost rather than CatBoost, so there is a learning curve for team members and sponsors unfamiliar with CatBoost.

Potential Risks:

- CatBoost may require significant computational resources for tuning.

Section E. Concept Evaluation and Selection

Feature Importance Analysis with XGBoost (Deprecated)

- By optimizing model features using Feature Importance Analysis we are able to pinpoint issues which prevent the model from obtaining a high accuracy by filtering features which may cause excess bloat and memory usage.

K-Fold Cross Validation with PCA Variable Selection

- PCA Variable Selection reduces computational complexity and mitigates overfitting by focusing on the most informative features.
- K-Fold Cross Validation ensures more reliable performance estimates and stronger generalization by testing the model on multiple data splits.

Model Tuning & Feature Importance with CatBoost

- Projected to improve performance post training which can be measured by both testing the model against datasets or referencing the confusion matrix.
- Reduces overfitting and improves model generalization, leading to more reliable performance.
- CatBoost has highly optimized default parameters, often providing strong results with minimal tuning.
- CatBoost provides feature selection as a pre-built feature of the library.
- CatBoost is not often used as it is found to be slower than LightGBM, but with its implementation of gradient boosting can improve performance accuracy with a large amount of features

Weight	Feature Importance Analysis with XGBoost (Deprecated)	K-Fold Cross Validation with PCA Variable Selection	Model Tuning with CatBoost
Accuracy Improvement	**0.0**	**0.2**	**0.8**
Compute Usage Improvement	**0.0**	**0.0**	**1.0**
Overfitting Reduction	**0.0**	**0.9**	**0.1**
Total	**0.0**	**1.1**	**1.9**

Section F. Design Methodology

F.1 System Architecture

The system consists of three main components:

Data Preprocessing Pipeline

- Extracts data from packet-capture files using bash
- Divides data into data equivalent to that required from nPrint by operating system and labels the data.
- Converts labeled data to csv format for training the model.
- Removes columns identified as noise and compresses single bit columns to binary strings.
- Performs k-fold cross validation on data for proper randomization.

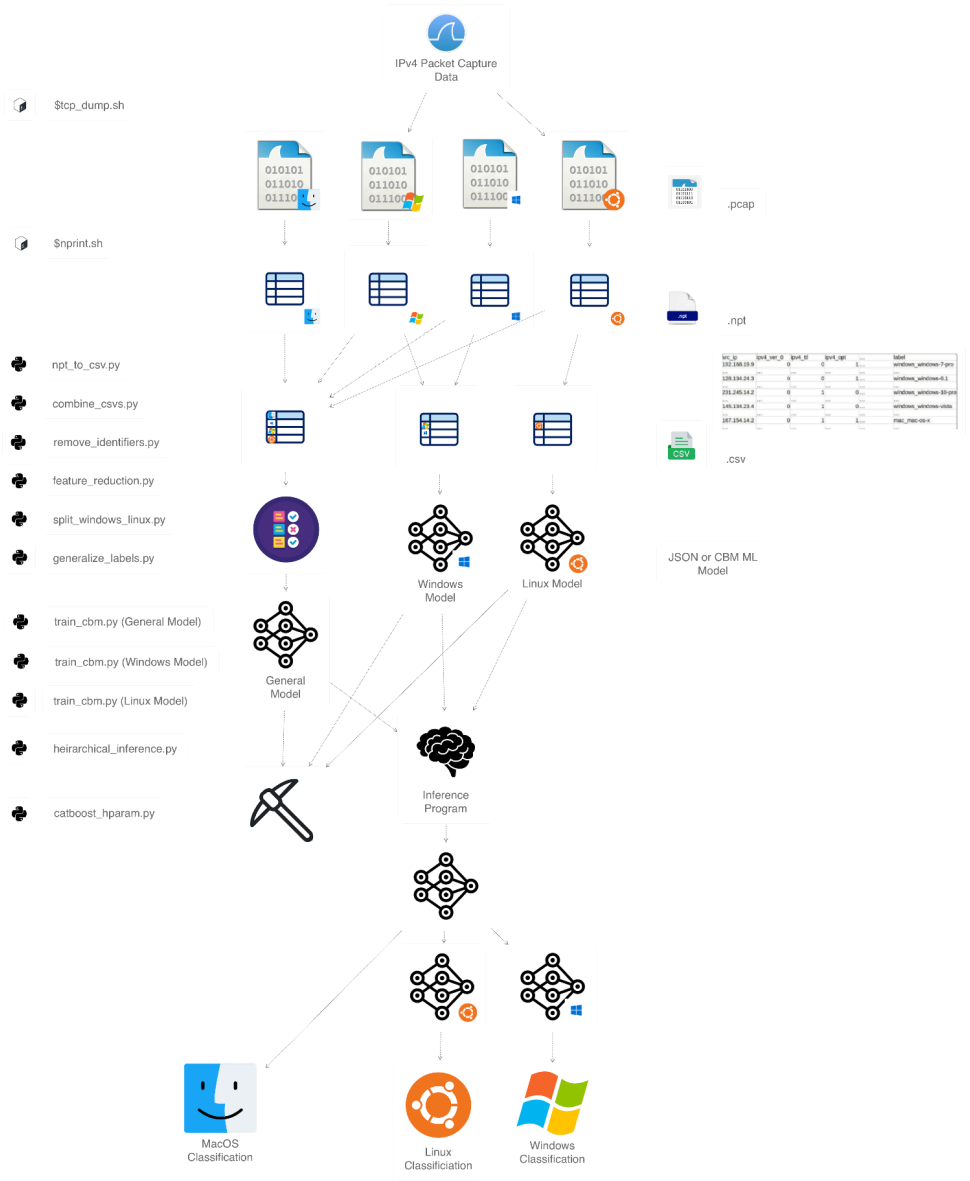
General Classification Model

- Trains the general classification model and extracts features from the data by extracting csv data from preprocessing pipeline.
- Produces performance metrics and confusion matrix for training with Catboost and testing data.
- Performs initial classification of network traffic to operating system identification.

Fine Tuned OS Models

- Trains the Windows and Linux models using CatBoost to make inferences on the origin system's operating system version.
- Produces performance metrics and confusion matrix for training and testing data.
- Performs targeted classification of network traffic to operating system version from previous classification.

F.2 Data Flow


IPv4 Packet Capture Data
$tcp_dump.sh
.pcap
$nprint.sh
.npt
npt_to_csv.py
combine_csvs.py
remove_identifiers.py
feature_reduction.py
split_windows_linux.py
generalize_labels.py
.csv
JSON or CBM ML Model
Windows Model
Linux Model
train_cbm.py (General Model)
train_cbm.py (Windows Model)
train_cbm.py (Linux Model)
General Model
heirarchical_inference.py
Inference Program
catboost_hparam.py
MacOS Classification
Linux Classificiation
Windows Classification

F.3 Validation Procedure

Testing Dataset

- Create or obtain testing data with diverse operating systems and traffic patterns. Testing on both CIC-IDS2017 and fresh datasets provided by the ASPIRE team.
- Test the model against randomized data splits to gauge true effectiveness of its classification capabilities and capabilities in identifying versioned operating system builds.
- Validate against known OS configurations which are prone to appearing in a wide variety of system configurations.

Performance Metrics

- Confusion Matrix Accuracy Output
- Utilization of System Resource Improvement
- Training & Model Response Time

Deployment Testing

- Testing containerized lightweight models on different operating systems.
- Testing model against system specification constraints to find true compute requirements.
- Testing limits the model's ability to handle larger datasets in an efficient manner.

Section G. Results and Design Details

G.1 Feature Engineering & Reduction

By eliminating noisy or redundant bit-fields, we achieved a lean 200-feature representation that accelerated all downstream steps without sacrificing predictive power.

Original design: Original OsirisML ingested 784 raw bit-features per packet, derived from IPv4 and TCP header fields (TTL bits, window-size bits, flags, checksums, options, etc.) .

Our improvement: We measured per-feature importance via Gradient-Boosted Trees, identifying the top 200 predictors (e.g. TTL, TCP window size, IPv4 checksum patterns).

Results:

- **Memory**: Reduced peak RAM from 424 GB ➔ 123 GB (–71%).
- **Training time**: Hyperparameter tuning (80 trials) shrank from 116 hr ➔ 2 hr.
- **Accuracy**: Held constant or slightly improved—no degradation from feature removal.

G.2 Switch from XGBoost to CatBoost

Adopting CatBoost yielded consistently higher OS-classification accuracy and simplified our tuning pipeline.

Original Design: XGBoost required extensive feature-wise parameter tuning and still under-performed on highly categorical/bit-pattern data.

Our Improvement: We re-implemented all classifiers in CatBoost, leveraging its ordered boosting and built-in handling of categorical/imbalanced features.

Results:

- **Accuracy uplift**: +5–10% absolute across test sets (e.g. 72% ➔ 91% on cleaned data).
- **Robustness**: Reduced sensitivity to hyperparameter settings ("out-of-the-box" performance).
- **Compute**: Comparable CPU usage; no increase in resource demands.

G.3 Hierarchical Modeling Architecture

Our hierarchical architecture yields both improved accuracy and more actionable partial predictions, overcoming limitations of a single model.

Original design: Single monolithic model predicting all OS versions; low-confidence misclassifications provided no partial insight

Our Improvement: Coarse classifier ("big model") distinguishes between Windows, Linux, and Mac, and specialized sub-models to classify the version within each OS family.

Results:

- Higher end-to-end accuracy (up to 91% on clean data).
- Interpretability: Even when sub-model errs, the coarse model gives high-confidence family label (e.g. "99% Linux, then 50% Ubuntu-16.04").

Section H. Societal Impacts of Design

H.1 Public Health, Safety, and Welfare

- Enhances military capability by automating the enumeration phase of offensive or defensive cybersecurity and network mapping.
- Easily identifies systems in these environments which may be susceptible to cyber attacks.
- Allows military forces to obtain a mapping of the network around them to perform action as needed.

H.2 Societal Impacts

- Risks misuse for surveillance or malicious purposes.

H.3 Political/Regulatory Impacts

- Does not notify any group which the model could be used on that collection of data and classification of their systems is taking place. This could eventually lead to a regulatory issue if used by anyone outside of the intended environment.
- Implications for network security standards and passive identification of network traffic.

H.4 Economic Impacts

- Reduces costs associated with alternative methods of in-field enumeration phases.
- Enables efficient resource allocation in military environments.
- Significantly reduces required time to map and understand environments with network traffic and could possibly decrease required payroll.

Section I. Cost Analysis

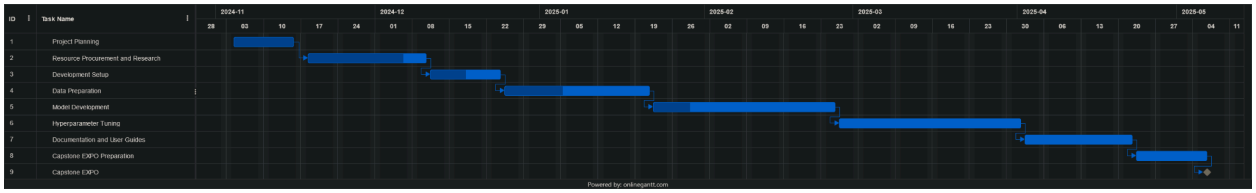
This project was able to be completed to term without any use of sponsor funds. The Virginia Commonwealth University Sycamore High Performance Computing Cluster was provided and proved sufficient to carry out the required work.

Section J. Conclusions and Recommendations

The original goals of the project were to improve the model of the previous researchers by addressing the low accuracy on the entire dataset, long training times, and significant ram usage. Through discussions with our project sponsor, we decided our first area to approach was swapping the underlying XGBoost implementation with CatBoost. We benchmarked CatBoost vs. XGBoost on the same sample of PCAPs, and found that CatBoost delivered better accuracy and faster training out of the box. To tackle ram usage and training times, our project sponsor encouraged us to try ranking the most important features for classification and keeping only those. We found that this feature selection process could heavily cut dimensionality of the features with a low accuracy loss and significantly improved training times. To further cut dimensionality, we tested feature aggregation by compressing multi bit features into single feature integers, which showed some promise, but also sometimes reduced accuracy, likely due to feature synergies being lost in the aggregation. During experimentation, we noticed that Windows systems typically used a TTL (Time to Live) of 128, while Linux systems used 64. Through brainstorming as a team, we realized we can use this fact to our advantage, and came up with a hierarchical approach where packets are passed down from a base model that decides whether the packet is from a Windows or Linux system, to a smaller sub-model fine-tuned for detecting differences between Windows and Linux versions. We found this significantly improved accuracy.

A question we were unable to answer with our resources is whether or not the model could be improved by cleaning the training data further. The IDS-2017 dataset is not originally meant to be used for operating system classification, and is instead for intrusion detection. If we were able to acquire a dataset intended for the purpose of our project, we predict a significantly higher accuracy on our models. We also were unable to adequately test Bayesian / evolutionary hyper-parameter search, which could lead to higher model accuracy. Another area that would be interesting to pursue is integrating SHAP values to give us a human-readable reason for each prediction from the models.

Appendix 1: Project Timeline



Appendix 2: Team Contract (i.e. Team Organization)

Step 1: Get to Know One Another. Gather Basic Information.

Task: This initial time together is important to form a strong team dynamic and get to know each other more as people outside of class time. Consider ways to develop positive working relationships with others, while remaining open and personal. Learn each other's strengths and discuss good/bad team experiences. This is also a good opportunity to start to better understand each other's communication and working styles.

Team Member Name	***Strengths each member bring to the group***	***Other Info***	***Contact Info***
William Lagos	*Technical skills, flexibility, problem solving, teamwork*		*lagoswj@vcu.edu*
Sam Soltanian	*Always trying to learn, problem-solving flexible*	*I enjoy learning from other people and in turn offering new knowledge*	soltanians@vcu.edu
Christopher Castro	Friendly, curious, bias for action, hard worker	Unfamiliar with cybersecurity topics / PCAP data, but eager to fill in knowledge gaps.	castrocm5@vcu.edu
Ryan Collette	*Always want to learn, willing to adapt to complete goals, enjoy working with a team.*	*Lack of in depth cyber-security concepts, but surface level knowledge and very interested in learning.*	colletterc@vcu.edu

Other Stakeholders	***Notes***	***Contact Info***
Project Sponsor	*Pending*	*Nibir Dhar dharnk@vcu.edu*
Staff Mentor	*Initial Meeting Wed, Sep. 4 @ Zoom*	*Irfan Ahmed iahmed3@vcu.edu*

Step 2: Team Culture. Clarify the Group's Purpose and Culture Goals.

Task: Discuss how each team member wants to be treated to encourage them to make valuable contributions to the group and how each team member would like to feel recognized for their efforts. Discuss how the team will foster an environment where each team member feels they are accountable for their actions and the way they contribute to the project. These are your Culture Goals (left column). How do the students demonstrate these culture goals? These are your Actions (middle column). Finally, how do students deviate from the team's culture goals? What are ways that other team members can notice when that culture goal is no longer being honored in team dynamics? These are your Warning Signs (right column).

Resources: More information and an example Team Culture can be found in the Biodesign Student Guide "Intentional Teamwork" page (webpage | PDF)

Culture Goals	***Actions***	***Warning Signs***
1: Punctuality	- *Students will arrive to meetings on time* - *Students need to communicate any needs to cancel or reschedule meetings*	- *Student misses with no explanation first meeting, warning is granted* - *Student misses meetings with no explanation afterwards – issue is brought up with faculty advisor*
2: Informing the group of any delays in completing assignments	- *Stay up to date with each other's project responsibilities* - *Set reasonable deadlines and note when an extension is needed*	- *Student shows up for weekly meeting with no considerable work done*
3: Be open to critical feedback and receptive	*- Be receptive to ideas that potentially differ from your own.* *- Be open to alternative design options that you may be unfamiliar with.* *- Respond to feedback in a mature and normal manner.*	*- Students are unreceptive/immature about feedback.*

Step 3: Time Commitments, Meeting Structure, and Communication

Task: Discuss the anticipated time commitments for the group project. Consider the following questions (don't answer these questions in the box below):

- What are reasonable time commitments for everyone to invest in this project?
- What other activities and commitments do group members have in their lives?
- How will we communicate with each other?
- When will we meet as a team? Where will we meet? How Often?
- Who will run the meetings? Will there be an assigned team leader or scribe? Does that position rotate or will the same person take on that role for the duration of the project?

Required: How often you will meet with your faculty advisor advisor, where you will meet, and how the meetings will be conducted. Who arranges these meetings?
See examples below.

Meeting Participants	***Frequency Dates and Times / Locations***	***Meeting Goals Responsible Party***
Students Only	*As Needed, in-person or on Discord based on requirements per week. Scheduled based on discussion every Thursday.*	*Actively work on project (Group will document these meetings by taking photos of whiteboards, physical prototypes, etc, then post on Discord and update Capstone Report)*
Students Only	*Every Thursday during the seminar time block*	*Update group on weekly challenges and accomplishments, scheduling as needed meetings for other time slots during the week.*
Students + Faculty advisor	*Every Every Monday at 4:15 on Zoom*	*Update faculty advisor and get answers to our questions, updates following agile methodology (what we've accomplished, what we're blocked on, what we're doing next)*
Project Sponsor	*Pending*	*Update project sponsor and make sure we are on the right track*

Step 4: Determine Individual Roles and Responsibilities

Task: As part of the Capstone Team experience, each member will take on a leadership role, ***in addition to*** contributing to the overall weekly action items for the project. Some common leadership roles for Capstone projects are listed below. Other roles may be assigned with approval of your faculty advisor as deemed fit for the project. For the entirety of the project, you should communicate progress to your advisor specifically with regard to your role.

- **Before meeting with your team**, take some time to ask yourself: what is my "natural" role in this group (strengths)? How can I use this experience to help me grow and develop more?
- **As a group,** discuss the various tasks needed for the project and role preferences. Then assign roles in the table on the next page. Try to create a team dynamic that is fair and equitable, while promoting the strengths of each member.

Communication Leaders

Suggested: Assign a team member to be the primary contact for the client/sponsor. This person will schedule meetings, send updates, and ensure deliverables are met.

Suggested: Assign a team member to be the primary contact for faculty advisor. This person will schedule meetings, send updates, and ensure deliverables are met.

Common Leadership Roles for Capstone

1. **Project Manager:** Manages all tasks; develops overall schedule for project; writes agendas and runs meetings; reviews and monitors individual action items; creates an environment where team members are respected, take risks and feel safe expressing their ideas.
   Required: On Edusourced, under the Team tab, make sure that this student is assigned the Project Manager role. This is required so that Capstone program staff can easily identify a single contact person, especially for items like Purchasing and Receiving project supplies.
2. **Logistics Manager:** coordinates all internal and external interactions; lead in establishing contact within and outside of organization, following up on communication of commitments, obtaining information for the team; documents meeting minutes; manages facility and resource usage.
3. **Financial Manager:** researches/benchmarks technical purchases and acquisitions; conducts pricing analysis and budget justifications on proposed purchases; carries out team purchase requests; monitors team budget.
4. **Systems Engineer:** analyzes Client initial design specification and leads establishment of product specifications; monitors, coordinates and manages integration of sub-systems in the prototype; develops and recommends system architecture and manages product interfaces.
5. **Test Engineer:** oversees experimental design, test plan, procedures and data analysis; acquires data acquisition equipment and any necessary software; establishes test protocols and schedules; oversees statistical analysis of results; leads presentation of experimental finding and resulting recommendations.

Team Member	*Role(s)*	*Responsibilities*
William Lagos	Systems Engineer	Establish project specifications, monitor, coordinate and manages integration of subsystems in the prototype, and develop/recommend systems for use in product
Christopher Castro	Project Manager	Write agendas and run meetings, review and monitor individual action items. Come up with major stories and set up agile board, organize action items.
Sam Soltanian	Logistics Manager	Communicate with Faculty Advisor & Advisor and communicate between team and faculty/sponsor. Make sure everyone is on the same page.
Ryan Collette	Test Engineer	Collaborate and plan experimental design, plan testing frameworks, scheduling testing, statistical analysis of training results or confidence weights.

Step 5: Agree to the above team contract

Team Member: Sam S *Signature:* ***Sam Soltanian***

Team Member: Ryan C *Signature: Ryan Collette*

Team Member: Chris C *Signature: Christopher Castro*

Team Member: William L *Signature:* *William Lagos*

References

[1] Canadian Institute for Cybersecurity. (2017). *Intrusion detection evaluation dataset (CIC-IDS2017)* [Data set]. University of New Brunswick. https://www.unb.ca/cic/datasets/ids-2017.html

[2] scikit-learn Developers. (2025). *scikit-learn: Machine learning in Python* (Version 1.6.1) [Computer software]. https://scikit-learn.org/stable/

[3] XGBoost Developers. (2025, February 27). *XGBoost documentation* (Version 3.0.0) [Computer software]. https://xgboost.readthedocs.io/en/release_3.0.0/ XGBoost Documentation

[4] CatBoost Developers. (n.d.). *CatBoost documentation* [Computer software]. Yandex. https://catboost.ai/docs/en/

[5] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*, 281–305. https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf