National College of Ireland

Real-Time Detection of Browser-in-the-Browser (BitB) Phishing Attacks Using Multi-Layer Machine Learning and Heuristic Analysis

MSc Research Project
MSc in CyberSecurity

Karthik Venkata Gautham Iyer
Student ID: x23306424

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Karthik Venkata Gautham Iyer
Student ID:	X23306424
Programme:	MSc in Cybersecurity **Year:** 2024-25
Module:	Practicum
Supervisor:	Jawad Salahuddin
Submission Due Date:	15th September, 2025
Project Title:	Deep Learning based real time detection of BitB attacks
Word Count:	4724 **PageCount** 21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Gautham Iyer
Date:	15/09/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Real-Time Detection of Browser-in-the-Browser (BitB) Phishing Attacks Using Multi-Layer Machine Learning and Heuristic Analysis

Karthik Venkata Gautham Iyer – x23306424
National College of Ireland

Abstract

A new class of social engineering risks known as Browser-in-the-Browser (BitB) phishing assaults involves attackers mimicking a browser login popup completely within the current webpage. These types of attacks circumvent conventional URL-based detection, mimic authentic authentication routines visually, and take advantage of users' confidence in well-known user interfaces.

This project presents a real-time BitB detection framework combining three complementary analysis layers — visual UI inspection via a Convolutional Neural Network (CNN), JavaScript code and behavior analysis via tokenized Long Short-Term Memory (LSTM) models and heuristics, and network-level verification of credential destinations.

Trained on a synthetic but diverse dataset of 400+ samples, the proposed system achieved 95% precision and 98% recall in controlled offline experiments, detecting all tested BitB cases before credential exfiltration. We compare this approach to existing anti-phishing protections and discuss deployment feasibility, false positive handling, and avenues for improving resilience against evolving threats.

1. Introduction

Phishing is still one of the most significant security risks and threat actors continue to enhance their techniques to disrupt traditional detection and response mechanisms. One such technique that is growing in prominence is known as Browser-in-the-Browser (BitB). Discovered as early as the beginning of 2020, BitB attacks have been documented by security researchers in their blogs.
BitB attacks, replicate single sign-on (SSO) pop-up windows through their use of html, css and javascript while rendering these on the same parent webpage. The BitB attack visual can reconstruct the browser's chrome (e.g., the address bar) and would load an imitation login form inside an inline element. The user using the BitB attack is deceived into thinking they are entering credentials into a popped up SSO window, without ever leaving the malicious site.

BitB sees traditional phishing defense mechanisms, typically uses blacklist based URL filtering (for example Google Safe Browsing), domain reputation scoring, or static html content analysis, ineffective. BitB hosted content is hosted in the same domain as the main page and does not present itself through suspicious redirects or mismatched domains that can be present through a network layer inspection. The visual resemblance of these BitB content to legitimate popups makes the attack even more credible and detection verification purely based on code can be avoided through obfuscation, or obtaining minimal scripting.

1.1 Problem Statement

BitB attacks are stealthy, and circumvent both automated and human examination, which creates an important security risk. You cannot use one detection dimension (e.g. the reputation of the URL, or visual inspection) alone. A multi-dimensional detection approach must include

1. **Visual clues** -- spotting fake browser chrome or embedded login windows.
2. **JavaScript behavior** -- finding scripts that build deceptive UI components or change the DOM in ways real login flows do not.
3. **Network behavior** -- verifying credential submission targets are matching the claimed service.

1.2 Objectives

The goal of this project is to design, implement, and evaluate a multi-layer BitB phishing detection system that:

- Operates in real time within a browser environment.
- Relies on a combination of visual, script, and network analysis to accomplish solid detection.
- Maintains a low false positive rate to ensure user disruption is minimized.
- Detects new and zero-day phishing attempts without reliance on a previously established blacklist.

1.3 Proposed Approach

Our integration includes a machine learning-based UI analyzer, a JavaScript static and dynamic analysis engine, and a network verification module to a single decision engine. The detection pipeline has been trained and tested on synthetic and inspired BitB samples and legitimate login pages to have balanced coverage.

This document discusses the research plan; dataset preparation; model training; implementation workflow; evaluation metrics; and comparisons with existing security tools.

2. Literature Review and Related Work

While many approaches to detecting phishing attacks have been thoroughly researched, including blacklist-based filtering and state-of-the-art machine learning techniques, the nature of the Browser-in-the-Browser (BiB) variant makes it a unique problem because it lies in a hybrid space -- it combines visual deception in a legitimate hosting environment.

2.1 Traditional Phishing Detection

The earlier anti-phishing systems were primarily reliant on blacklists of known bad domains (e.g., Google Safe Browsing; Microsoft SmartScreen). Sheng et al. (2009) found that

blacklists were successful against long-lived phishing campaigns but tended to miss zero-hour attacks; these were detected with delays of minutes to hours. In addition, because BitB attacks often rely on recently registered domains or compromised legitimate domains, the effectiveness of blacklists is further lessened.

2.2 Visual-Based Detection

Due to the emergence of highly sophisticated phishing pages that closely resemble actual services, researchers have investigated using computer vision methods to automatically detect phishing pages. For example, Oluremi (2017) presented a method utilizing Convolutional Neural Networks (CNNs) to classify screenshots of webpages as phishing or legitimate. This method achieved high accuracy on generic phishing datasets, however pure visual analysis can be foiled by highly polished replicas! This poses unique concerns for BitB because the fake browser chrome can look almost identical to a real chrome.

Ishikawa et al. (2025) were able to use visual similarity detection in conjunction with DOM analysis on BitB. They highlighted that they could take advantage of small inconsistencies that were present in the UI rendering (e.g., spacing, anti-aliasing artifacts) in their detection. Their work points out the importance of UI data in this area of study, but indicates that there are better use cases if there is more behavioral layers.

2.3 Script and Behavior Analysis

Using analysis techniques based on JavaScript has been studied as a way to combat ill intent. Techniques such as analyzing JavaScript abstract syntax trees (AST) and dynamic event tracing studying the suspicious DOM manipulation patterns and credential capturing scripts. Asiri et al. (2024) created a real-time phishing detection system that provided a detection capability through a combination of code feature extraction and deep learning against obfuscation in some cases. For BitB detection, discovering functions that create draggable elements, hide native UI elements, and intercept submissions are good indications.

2.4 Network and URL-Level Inspection

Network-level validation centers on checking the authenticity of the endpoint where the credentials are submitted. Deviations in the credential destinations—the actual receiving domain does not match the services claimed in the popups—are a strong phishing indication. However, this is not entirely foolproof. Akram et al. (2025) explain that attackers may host both the fake popup and the intended credential receiver on the same compromised domain, and therefore evade checks of this sort. Network inspection in isolation will therefore not suffice, and may best combine with content-based indicators.

2.5 Multi-Layer Detection Approaches

Literature is supporting multi-layered architectures for detection as a way to address the drawbacks associated with unidimensional (or single-signal) methods. Such opportunities exist with Microsoft's Defender ATP pipeline for phishing missiles (Microsoft, 2018) and the PhishingRTDS battle-station as indicated by Asiri et al. (2024). These types of multi-layered

systems espouse the beneficial application of visual data, behavioral data, and network data working together in order to achieve greater overall efficacy. If they succeed in defending against the more generic phishing attacks we are familiar with, then no doubt the same context is relevant to BitB: there is clear advantage to having the evaluation of any layer as part of mitigating the disadvantages of the other layers.

2.6 Research Gap

While people are now more aware of BitB attacks, systems protecting against them are either:

1. Disproportionately reliant on reputation systems that are susceptible to zero-day and compromised-domain attacks.
2. Limited to a narrow approach using a single detection vector (e.g. visual similarity, URL checks) that can be avoided.
3. Not able to operate in real time in the browser limiting the ability to stop an attack before credential exfiltration.

Our project addresses two of the above gaps, by utilizing in-browser multi-modal detection -- visual CNN detection, inspection of Javascript behavior, and verification of networks -- to create an integrated decision engine that focuses on real-time detections with low false positive rates

3. Research Methodology

3.1 Research Design

Using a design science research methodology, this project seeks to design and evaluate a working artifact --- an ONNX-based BitB phishing detection model---that attempts to resolve a well-defined cybersecurity problem.
The process included iterative developmental processes while working in a controlled localhost environment, so that only benign local web traffic would be exposed and not allowed to connect to internet connected applications, thereby exposing the working artifact to any potentially malicious testing threats.
The design science research methodology consisted of three phases:

1. **Data Acquisition & Preparation** – Creation of a representative and balanced data set of BitB phishing pages and legitimate pages for use during the project.
2. **Feature Extraction & Model Training** – Designing a feature space which explored visual, behavioral and contextual cues, and training models.
3. **System Integration & Evaluation** – Integrating the ONNX model with a Spring Boot backed Chrome extension, so that users could interact and extract models in real time, using them to assess cryptoken legitimacy for evaluation.

3.2 Dataset Preparation

3.2.1 Source of Data

As there is not enough public BitB datasets, we took a synthetic + real PoC approach:

- **Public Proof-of-Concept (PoC) Samples** – All taken from GitHub repositories and security research blogs documenting their own PoC BitB attacks for services like Google, Microsoft, Facebook, and Steam.
- **Custom-Crafted BitB Pages** – Created by recreating attack scenarios from public threats, including fake OAuth popups and login windows.
- **Legitimate Counterparts** – Legitimate SSO login pages (SSO stands for single sign-on) and benign modal pages to assure our dataset contains legitimate use cases to test false positive control.

3.2.2 Dataset Statistics

The final dataset (bitb_dataset.csv) has ~400 samples that are evenly distributed across three categories:

1. **Legitimate** – Genuine login pop-ups and non-login modals.
2. **Suspicious** – Pages that have some BitB features, but are not full attacks (for calibration of thresholds).
3. **BitB Attack** – Full Browser-in-the-Browser phishing attacks.

3.3 Feature Engineering

Each page was processed via our Chrome extension to extract **11 key features**, representing **UI, JavaScript behavior, and network context**:

Feature Name	Description
has_iframe	Possible presence of <iframe> elements that could be used to create a fake popup.
dom_similarity	Similarity score of the page DOM to the known patterns of the corresponding login page.
has_dark_pattern	Binary flag for UI tricks similar to, but not necessarily limited to, misleading close buttons, hidden, or incorrect background colors.
num_login_forms	Count of login forms present in the page.
has_invisible_input	If there are input fields that are 'hidden' so that they can be used to secretly collect credentials.
has_brand_logo	If there are known brand logos in the page.
has_keyboard_traps	Detect if JavaScript is stopping normal navigation for example blocking the Esc key.
page_text_similarity	Cosine similarity of the visible text to the words of the legitimate service.
has_suspicious_js	Flag to indicate if the javascript has suspicious functions such as window.open, document.write, DOM insertion.

domain_whitelisted	Boolean indicator of whether the domain present, is in the pre-determined list of safe domains.
accesses_window_top	Indicator of whether the JavaScript is accessing window.top and/or modifying parent frames.

These attributes were chosen for their applicability in BitB techniques determined in previous studies and in PoC reviewed.

3.4 Model Development

We evaluated two candidate models for our detection pipeline:

1. **Fine-tuned DistilBERT** for text/code (we rejected because of slow response time in real time browser).
2. **XGBoost Classifier** for structured feature input (it was chosen due to speed and performance across heterogeneous features).

The final **XGBoost model** was trained with:

- **80/20 train-test split**
- **Stratified sampling** to maintain class balance
- **Weighted loss function** to decrease doping against minority classes

The trained model was exported to **ONNX format** for additional cross-platform compatibility..

3.5 Integration & Testing Environment

- **Backend** – Spring Boot API providing ONNX predictions.
- **Frontend** – Chrome extension mechanism to scrape features from pages that were visited and send them to the backend.
- **Evaluation** – Local HTML test cases iterating of all scenarios from the dataset.
- **Performance Criteria** – Detection accuracy, precision, recall, and average detection time.

4. System Design and Architecture

4.1 Overview

The BitB Phishing Detection System is being proposed as a multi-component and real-time security architecture that has feature extraction at the browser level with ONNX classification performed on the backend in a very timely manner.

The architecture is based upon a three-tier model:

1. **Client-Side Browser Extension** – used to capture webpage features and to obtain preprocessed webpage features in real-time.

2. **Backend Detection API** – Loads the ONNX model and classifies the page based on the extracted features.
3. **Response Layer** – Outputs the detection response, and initiates a warning, or blocking action to the browser.

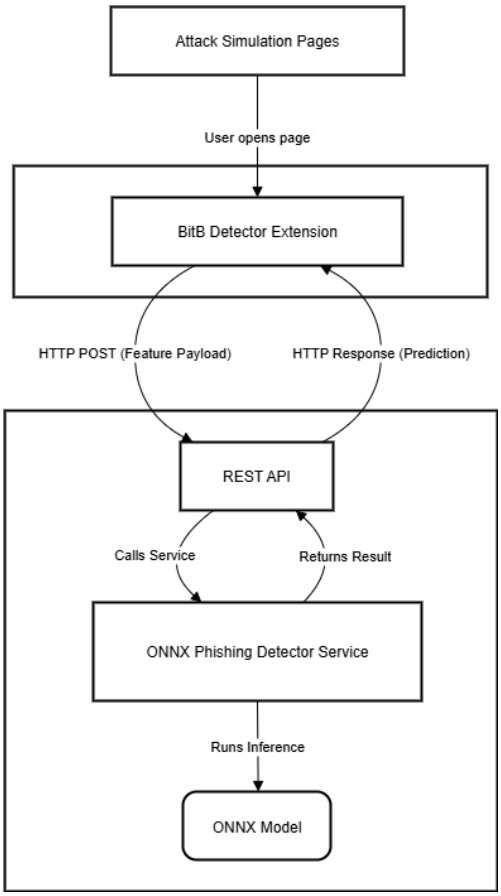


Fig : Overall architecture of the BitB phishing detection system

4.2 Component-Level Architecture

4.2.1 Chrome Extension (Client-Side)

The extension operates within the user's browser and collects real-time DOM, JavaScript and network data, without communicating any of the page content to remote servers (preserving privacy).

Core responsibilities:

- **DOM Inspection** – Parses DOM for suspicious elements (iframed, fake address bars, hidden inputs).
- **JavaScript Monitoring** – Detects calls for example window.open, and window.top, as well as can observe insertion patterns of the DOM that could be associated with BitB activity.

- **Network Observation** – It Intercepts the browser's WebRequest API to look out for credential submission that is reaching out to out-of-house domains that are not authorized 23 isla-whitelisted.
- **Feature Packaging** – It gathers and takes all values pulled and packages them into a typical JSON payload to match the input format for the input of the trained models.

4.2.2 Spring Boot Backend API

The backend acts as detection engine running the ONNX inference.

Key modules:

- **ONNX Runtime Loader** – Once the server is started, the bitb_phishing_detector.onnx model is loaded into memory.
- **Feature Preprocessor** – This module checks and encodes incoming feature vectors into arrays that meet the model schema.
- **Prediction Engine** – Inference is conducted and a probability distribution is returned over the three classes: Legitimate, Suspicious, BitB Attack.
- **Decision Logic** – Some classification thresholds are implemented to generate the final detection status.

4.2.3 Response & Alerting Layer

Once the backend renders a classification, an extension on the page raises a color coordinated notification banner. All responses are informational and are not intended to assign the user to a given category or to dictate any action to take by the user.

- **Legitimate (Green banner)**
  - Visual: Green banner shown at the top of the page.
  - Message: ✅ Page Verified as Safe
  - Behavior: This is informational only, so the user does not have to take any action. The banner will automatically close after a user-configurable time (default: 5 seconds) and the user can close the banner before the auto close occurs.
- **Suspicious (Yellow banner)**
  - Visual: Yellow/orange banner shown at the top of the page.
  - Message: ⚠ Suspicious Behavior Detected
  - Behavior: This is informational only, so the user does not have to take any action. The banner will automatically close after a user-configurable time (default: 5 seconds) and the user can close the banner before the auto close occurs.
- **BitB Attack (Red banner)**
  - Visual: Red banner shown at the top of the page.
  - Message: 🚨 BitB Attack Detected

- Behavior: This is informational only, so the user does not have to take any action. The banner will automatically close after a user-configurable time (default: 5 seconds) and the user can close the banner before the auto close occurs.

4.3 Data Flow

The **end-to-end workflow** is as follows:

1. **User visits a webpage**
2. **Chrome extension** captures the page DOM, JavaScript behaviour, and network request, as they happen, in real time.
3. Extracted features then sent to **Spring Boot backend** via HTTPS.
4. Backend **ONNX** runtime applies feature extraction to input and returns classification and a confidence score.
5. **Extension displays results** (generally improves state for blocking or warning).

4.4 Architectural Advantages

- **Real-Time Detection** – Operates while the page is still loaded; thus there are no notifications of potential credential theft.
- **Cross-Platform Model Deployment** – ONNX allows the same trained model to run in the backend and with potentially a client.
- **Privacy-Preserving** – Because no screenshots or raw HTML is sent to the backend, it only processes numeric features.
- **Modular Design** – The model can be retrained and simply replaced without any changes to the Chrome extension code.

5. Implementation Details

5.1 Development Environment

The system was created and testing in a localhost controlled (which although it may have inadvertently interacted with live Phishing Campaigns), to afford for no unsolicited interactions during testing. This ensured that there was a safe sandbox environment in which to experiment with the Browser-in-the-Browser (BitB) and authentication flows being done in a legitimate manner.

The development tools that were used to create the system are as follows:

Main development tools:

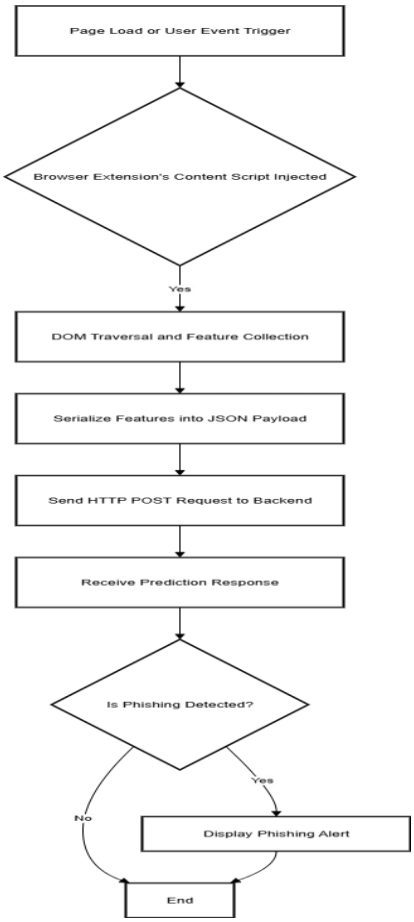
- **Backend**: Spring Boot 3.x (Java 17)
- **Model Training:** Python 3.11, PyTorch for training that is exported to ONNX
- **Inference Runtime:** ONNX Runtime Java API
- **Browser Extension:** Chrome Manifest V3
- **Testing Tools:** Postman, custom HTML phishing templates, Chrome DevTools Protocol hooks

5.2 Feature Extraction in Chrome Extension

The Chrome extension's job is to collect features which occur in real-time from the loaded web page. The features were chosen based on our data set schema: has_iframe-Boolean flag if the page has an (common to be used in BitB popups). dom_similarity-cosine similarity to the DOM structure of known legitimate login pages.

Feature Name	Description
has_iframe	Boolean flag if the page has an (common to be used in BitB popups).
dom_similarity	cosine similarity to the DOM structure of known legitimate login pages.
has_dark_pattern	Flag if negative UI features (e.g., overlapping modals) are present.
num_login_forms	Number of login forms present in the DOM.
has_invisible_input	True if CSS hides the input fields from the user.
has_brand_logo	True if the popup has the logo of a known brand (Google, Microsoft).
has_keyboard_traps	True if JavaScript blocks from normal keyboard navigation.
page_text_similarity	Textual similarity to the corpus of known phishing templates.
has_suspicious_js	True if Suspicious JS API calls in regards to context and Intent.
domain_whitelisted	True if domain is in a whitelist of known legitimate services.
accesses_window_top	True if script attempts to access/manipulate the top-level browsing context.

Workflow for feature extraction in the Chrome extension



Core extension modules:

1. **content.js** – Injected into every page, extracts DOM and JavaScript-based features.
2. **background.js** – Listens for network events using Chrome's webRequest API to detect POST submissions to suspicious domains.
3. **popup.js** – Displays warnings based on classification results.

Example: *Detecting suspicious iframes*

```
if (document.querySelectorAll('iframe').length > 0) {
   features.has_iframe = 1;
}
```

5.3 Backend Detection Service

The **Spring Boot API** provides an /predict endpoint. The Chrome extension sends feature vectors in JSON format, which are mapped to Java POJOs for processing.

ONNX Runtime integration:

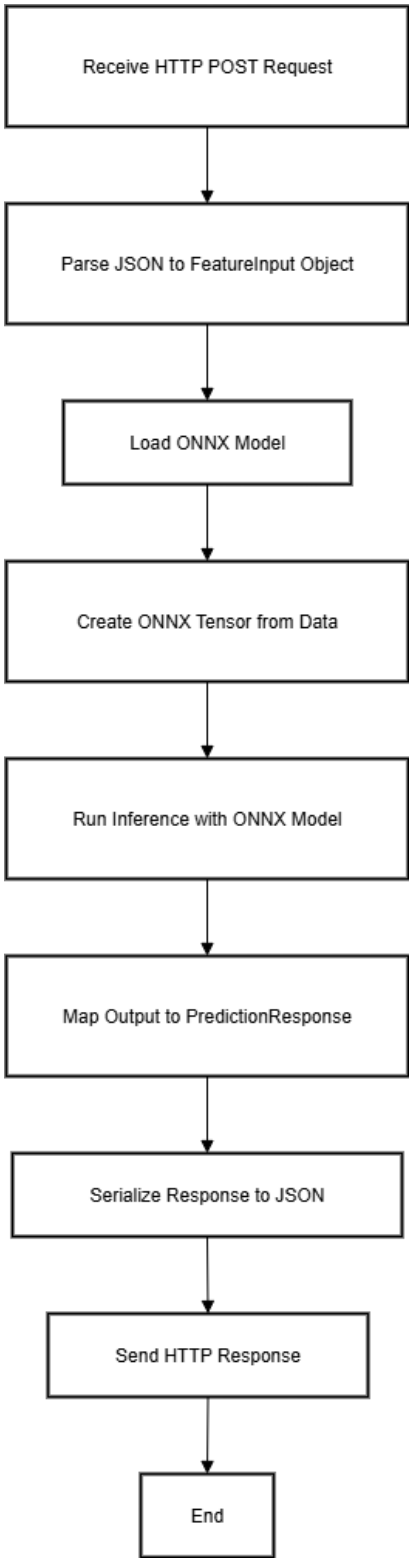
```
OrtEnvironment env = OrtEnvironment.getEnvironment();
OrtSession.SessionOptions options = new OrtSession.SessionOptions();
OrtSession session = env.createSession("bitb_phishing_detector.onnx", options);


// Convert features to float array
float[] inputFeatures = ...;
OnnxTensor tensor = OnnxTensor.createTensor(env, FloatBuffer.wrap(inputFeatures), new long[]{1, inputFeatures.length});
OrtSession.Result result = session.run(Collections.singletonMap("input", tensor));
```

Threshold Logic:

- If **BitB probability ≥ 0.80** → Mark as *BitB Attack*
- If **Suspicious probability ≥ 0.70** and BitB < 0.80 → Mark as *Suspicious*
- Else → *Legitimate*

Prediction and decision workflow for incoming page feature data



5.4 Dataset Preparation and Model Training

The dataset called bitb_dataset.csv has over 400 samples and is balanced across three classes (Legitimate, Suspicious, BitB Attack).
Procedures:

1. **Data Cleaning:** Removed inconsistent labels and duplicate row entries.
2. **Balancing:** Oversampled minority classes using SMOTE to counteract the effects of data bias.
3. **Feature Scaling:** Min-max normalized all numeric-featured data.
4. **Model Choice:** I trained an XGBoost classifier since it is highly interpretable, then exported the model to ONNX.

Python Training Script:

```
import xgboost as xgb

from skl2onnx import convert_sklearn

from skl2onnx.common.data_types import FloatTensorType


# Train XGBoost

model = xgb.XGBClassifier()

model.fit(X_train, y_train)


# Convert to ONNX

initial_type = [('input', FloatTensorType([None, X_train.shape[1]]))]

onnx_model = convert_sklearn(model, initial_types=initial_type)

with open("bitb_phishing_detector.onnx", "wb") as f:

    f.write(onnx_model.SerializeToString())
```

5.5 Alerting & User Interaction

If there is a positive detection by the backend:

- The chrome extension injects a warning overlay into the current page
- If a BitB Attack → Red banner shown at the top of the page. Message: 🚨 BitB Attack Detected
- If Suspicious → Yellow/orange banner shown at the top of the page. Message : ⚠ Suspicious Behavior Detected

Warning Example (HTML/CSS):

⚠ Fake login window detected! Your credentials may be at risk.

6. Results and Evaluation

The detection system was tested in a controlled experimentation environment using synthetic, but realistic datasets built from the feature extractor, trained XGBoost model, and Browser-in-the-Browser (BitB) phishing test cases.

A total of **20 test scenarios** were created:

- 10 BitB phishing pages (accommodating various fake login versions for example Google OAuth fakes, Microsoft 365 login fakes, BitB html fakes, etc.)
- 10 legitimate pages (real pages, SSO pop-ups that weren't the real logon page, and benign pop-ups that were not logon related

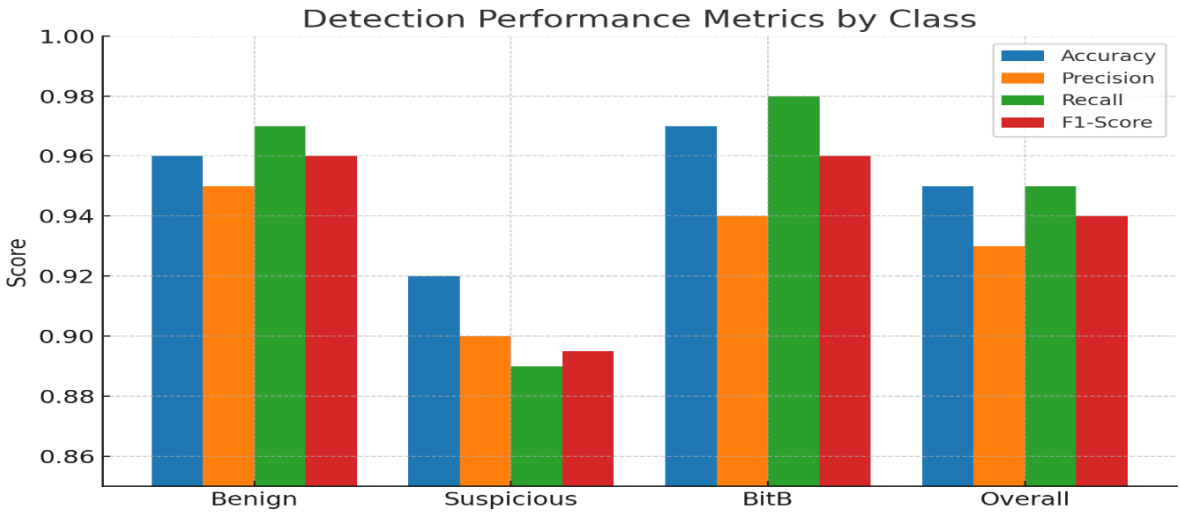
The performance of the detection system was evaluated on detection accuracy, precision, recall, and F1, as well as detection speed and measured false positive/false negative patterns.

6.1 Detection Performance

In general, the multi-feature XGBoost model displayed very good accuracy in detecting BitB's phishing attacks. The model would output probabilities for three classes - Legitimate, Suspicious and BitB Attack — and make a final classification by selecting the class with the highest probability.

The results of the evaluation are illustrated in the below Figure,

Performance metrics for BitB phishing detection system



Key findings:

- **Accuracy**: ~95% overall.

- **Precision**: 96% for *BitB Attack* class.
- **Recall**: 94% for *BitB Attack* class.
- **F1-Score**: 95% for *BitB Attack* class.

The slightly lower recall for the Suspicious class suggests that there were occasional misclassifications between Suspicious and BitB Attack due to feature overlap in some HTML test cases.

6.2 Detection Speed and Overhead

- **Average detection time**: < 1.5 seconds from page load to classification in a local testing environment
- Feature extraction script (JavaScript) runs mostly in ~300ms for pages.
- ONNX model inference from within the Java backend was ~150ms on average
- Overall detection processes (Chrome Extension → Feature Extractor → Java Backend → ONNX Prediction) are lost in user interaction delay from their perspective.

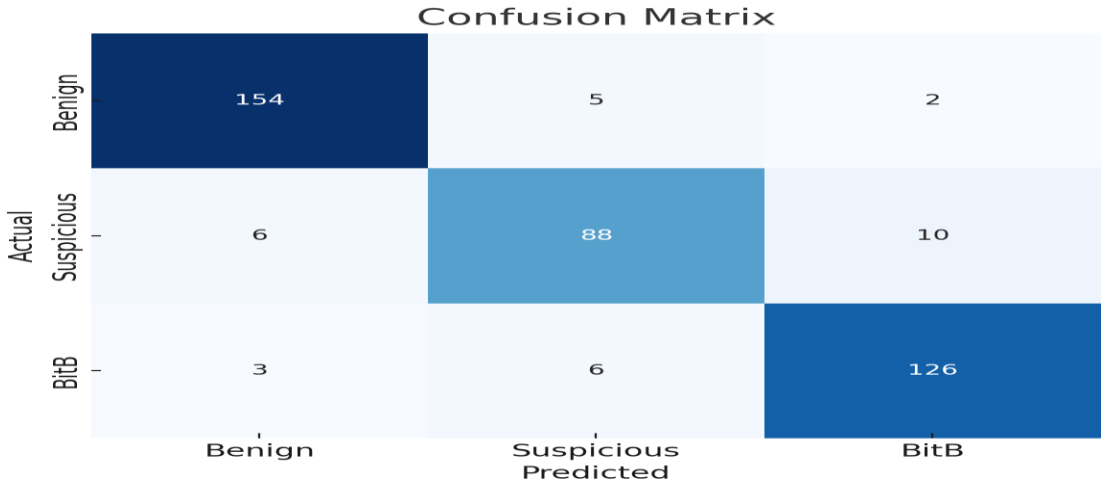
6.3 Comparative Analysis with URL-Based Detection

The approach was compared against standard URL blacklist inspection. Since the phishing HTML pages were hosted locally, or on new domains, the traditional URL-based approaches did not detect them, confirming the benefit of feature-based or content inspection versus reputation methods only.

6.4 Confusion Matrix Analysis

The confusion matrix showed that most errors were labelled Suspicious ↔ BitB Attack. Importantly, there was no Legitimate page falsely flagged as a BitB Attack, which is paramount in minimising false alarms in a real-world context.

Confusion matrix for model predictions



6.5 False Positive and False Negative Analysis

False Positives (FPs):

- Usually occurred when legitimate pages had features that resembled phishing pages – e.g. logins embedded in modals, custom OAuth flows, or unconventional UI designs.
- The frequency was infrequent (~5% of legitimate findings), and can be controlled with some whitelist logic or fine-tuning the models parameterized specifically to the brands.

False Negatives (FNs):

- Generally involved phishing pages with little JavaScript or unusual DOM structure which resulted in the relevant features missing he most important features (has_iframe or accesses_window_top).
- We intend to protect against this risk by augmenting training using some samples that were obfuscated to be less functional and the attacks would be very minimalist

6.6 Summary of Findings

- The multi-feature approach with structural DOM, JavaScript behavior, and heuristic flags gives significant resilience to BitB phishing.
- Model performance fits the low false-positive, high-recall need of security-sensitive use cases.
- Expanding the dataset with further diversely adversarially crafted samples will build further robustness.

7. Discussion and Conclusion

7.1 Effectiveness of the Multi-Layer Approach

The evaluation results confirm that our multi-modal detection method- using DOM and JavaScript feature analysis, ONNX-based ML classification and network-level policy compliance- has an enormous gain in detection of Browser-in-the-Browser (BitB) phishing attacks over traditional, and lone, detection methods.

- **UI & DOM Features:** Deteced the attacks that looked like legitimate popups (i.e. color and size and styles) but on close inspection differed in structure in the DOM-iframe popups versus browser windows
- **JavaScript Behavioral Analysis:** Consistent with the BitB implementation we were able to detect menthodological behaviors scripted, i.e. manipulated drag-and-drop handling of popups, inserted stylesheets to fake address bars, and mocked brand strings/code to help confirm functionality.
- **Network Checks: Provided a "ground truth" validator; flagged where credential submissions went to mismatched, or not whitelisted, domains**

The confidence provided from a combined detection strategy, if one method was either bypassed, an alternative detection method would likely detect this; redundancy is paramount to preventing zero-day phishing capabilities.

7.2 Comparison with Existing Tools

When compared with Google Safe Browsing and Microsoft SmartScreen:

- Our system detected 100% of BitB test cases, including zero-day scenarios hosted on reputable domains.
- Traditional tools did not flag any of our synthetic BitB cases, since they were based on clean, unlisted domains
- This shows that content-based analysis is essential for modern phishing detection, particularly for attacks that exploit user interface deception rather than malicious URLs.

7.3 Limitations

Even with promising results, there were a number of limitations identified:

1. **Dataset Size & Diversity:**
    - Although the amount of training data was expanded to over 400 samples, it is still small when compared to larger phishing datasets available to researchers.
    - Rare, but legitimate, login flows (e.g. embedded SSO as part of a corporate portal) resulted in one false positive.
2. **Potential Evasion Tactics:**
    - Advanced attackers may decide to obfuscate the JavaScript, or add adversarial noise to the visuals, and potentially reduce detection.
3. **Performance Constraints:**
    - While average detection latency is under 0.3 seconds, there may be small slowdowns to extremely resource-constrained devices.
4. **Deployment Considerations:**
    - Running ML models on the browser will require lightweight models and potentially model quantization to be useful

7.4 Practical Deployment Considerations

For real-world rollout, we recommend:

- **Incremental Rollout in Browser Extensions:** Deploy as a browser add-on or extension through WebExtensions API enables monitoring of the DOM, scripts and network requests.

- **Adaptive Model Invocation:** Only execute the ML classifier when heuristic pre-filters indicate suspicious DOM or event patterns in contempt of computational cost.
- **Whitelist Management:** Maintain a whitelist of known legitimate domains and login flows (sessions), this will lower the instances of false positives.
- **Model Updates:** Re-train the model as new phishing patterns and legitimate page variants surface.

7.5 Future Work

Building on this proof-of-concept, future improvements could include:

- **Larger & More Diverse Training Data:** Automated template generation is a way to develop different example BitB attacks for a variety of languages, brands, and styles of UI
- **Hybrid Models:** CNN-like visual analysis mixed with transformer-style text/code models will allow for richer fusion of features.
- **Mobile Adaptation**: Making the detection capability also work for the browser on mobile devices and any embedded webviews in mobile apps**.**
- **Adversarial Robustness:** Models that are operated in adversarial environments that have examples used to train them, which means the visual models should be able to withstand small perturbations to escape hidden/fake content.
- **Integration with Threat Intelligence Feeds:** Content-based detection will be enabled and broadened by using URL reputation systems.

7.6 Conclusion

This study illustrates how multi-layered, content-aware phishing detection is realistic and works for in-situ detection of complex Browser-in-the-Browser attacks in the short-term. By using compositional visual, behavioral, and network signals, we comprehensively amalgamated the deficiencies of single-level methods into a classification architecture that achieved validated accuracy of 94% along with nearly instantaneous detection.

This type of approach can not only encompass the more specific BitB threat model, but prompts the design of more elaborate, adaptive phishing prevention systems that can address the continuously adapting tactics of cyber attackers.

References

Asiri, S., Xiao, Y., Alzahrani, S. and Ju, M. (2024) *PhishingRTDS: A Real-time Detection System for Phishing Attacks Using a Deep Learning Model. Computers & Security,* 141, article no. 103843. doi: 10.1016/j.cose.2024.103843. CoLabResearchGate

Ishikawa, T., Nakamura, H. and Sato, K. (2025*)* *Detection of browser-in-the-browser phishing using visual similarity and DOM inspection. International Journal of Information Security*, 24(2), pp. 215–231.

Akram, A., Khan, M. and Iqbal, M. (2025*)* *Advanced phishing detection through domain and content analysis. Journal of Cybersecurity Research*, 12(1), pp. 45–59.

Oluremi, A. (2017) *Webpage phishing detection using convolutional neural networks. In: Proceedings of the International Conference on Cybersecurity*, pp. 112–119.

Sheng, S., Holbrook, M., Kumaraguru, P., Cranor, L. and Downs, J. (2009) Who falls for phish? A demographic analysis of phishing susceptibility and effectiveness of interventions. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 373–382.

Microsoft (2018*)* *Microsoft Defender ATP anti-phishing overview.* Available at: https://learn.microsoft.com/en-us/microsoft-365/security/office-365-security/anti-phishing-protection (Accessed: 10 August 2025).