
SCHOOL OF ENGINEERING, APPLIED SCIENCE, AND TECHNOLOGY

2025-2026

FINAL YEAR PROJECT REPORT

**on**

# FLARE: Federated Learning for Adversarial and Risky Emails

Course: BCS410

Submitted by


| | |
|---|---|
| Ralph Bernard Sengco | 20220001667 |
| Dean Francis Tolero | 20220001125 |
| Rami Alrajei | 20220001825 |
| Kjeldahl Claisen Cadeliña | 20220001658 |


In partial fulfillment of the requirements for a Bachelor of Science in Computer Science/Cybersecurity

# Abstract

This project addressed the growing sophistication of phishing email attacks, which increasingly bypassed conventional defenses through LLM-generated text and socially engineered persuasion. A survey of existing anti-phishing browser extensions found that the majority relied on URL scanning and ad blocking rather than direct analysis of email content, and the few that claimed to incorporate machine learning offered little documentation or transparency. Centralized content-based analysis was also impractical, as raw email text could not be forwarded to a remote server without compromising user privacy. To address this gap, a federated learning-based phishing detection system was designed and implemented. The system was structured into three components: a Chrome Manifest V3 browser extension that automatically extracted opened email content from Gmail and exposed manual flagging and override functionality, a local FastAPI client server that performed inference using a fine-tuned DistilBERT transformer, and a central FastAPI server that aggregated local model weight updates using the Federated Averaging (FedAvg) algorithm with threshold-based and timeout-based round finalization for asynchronous client participation. Raw email text never left the user's network; only model weight deltas were transmitted to the central server. The base model was fine-tuned on the MeAJOR Corpus, using a stratified 80/10/10 train/validation/test split with the AdamW optimizer and standard cross-entropy loss, prioritizing recall during model selection to minimize the misclassification of phishing emails as legitimate. Evaluation showed that the trained model achieved 96.32% accuracy, 95.35% recall, and 96.33% F1-score on the held-out test set, exceeding the 90% non-functional requirement target. End-to-end inference latency through the local client API averaged 4.1 ms for short emails and 13.7 ms for long emails, well below the 100 ms response target. A suite of thirty-one automated unit and integration tests passed across the aggregator, client database, data pipeline, and HTTP API. The results demonstrated that content-based phishing detection could be delivered in a privacy-preserving, federated manner without sacrificing the accuracy or latency required by a practical, browser-integrated tool.

# Acknowledgments & Dedication


### Acknowledgments

The completion of this project would not have been possible without the guidance, support, and contributions of several individuals and institutions.

First and foremost, we would like to express our sincere gratitude to our project advisor Dr. Yasir Faheem and to other faculty of the School of Engineering, Applied Technology, and Science for their continuous guidance, constructive criticism, and technical insights throughout the development of this research. Their recommendations and feedback greatly contributed to improving both the technical implementation and the quality of this report.

We also acknowledge the support provided by our institution through the facilities, laboratory resources, and academic environment that enabled the successful completion of this project.

Special appreciation is extended to the developers and maintainers of the open-source tools, frameworks, and datasets utilized in this project, including PyTorch, FastAPI, SQLAlchemy, pytest, Chart.js, and the MeAJOR Corpus dataset. Their contributions to the broader research and development community made this work significantly more achievable.

Finally, we are deeply grateful to our families for their patience, encouragement, and unwavering support throughout the duration of this project.


### Dedication

This work is dedicated to our families, whose constant encouragement and sacrifices served as our source of motivation throughout the development of this project.

We also dedicate this study to our mentors, instructors, and fellow students in the field of cybersecurity and software engineering, whose passion for learning and innovation continues to inspire us to pursue solutions that contribute toward a safer digital environment.

Lastly, this project is dedicated to future researchers and developers working in cybersecurity, machine learning, and privacy-preserving systems, with the hope that this work may serve as a meaningful reference and foundation for further advancements in phishing defense technologies.

# Table of Contents

# List of Figures

# List of Tables

# 1. Introduction

## 1.1 Background & Motivation

Phishing is a type of cyberattack utilized by cybercriminals to trick victims into revealing sensitive information, often via masquerading as a trusted authority or organization. This attack preys on misplaced trust by the victim, and there are several ways this can be done such as through emails, social media, or “pharming” [1]. Phishing is particularly dangerous as a successful phishing attack can bypass a system’s existing security mechanisms trivially, leading to severe direct and indirect damage to a person or organization [2]. One of the most effective countermeasures would be to use technical safeguards with continuous security awareness and education [3].

This is critical because social engineering accounts for approximately 68% of attacks commonly utilized when breaking into a system [4]. One of the common mediums in which these attacks are staged are via malicious emails designed to trick users into interacting with the email via downloadable files or malicious links or by tricking the user into performing an action they would normally not perform. The trickery involved in executing these tasks requires a significant amount of creativity from attackers. To simulate legitimacy, attackers use different techniques such as inconspicuous typos when imitating legitimate actors or persuasive texts that pressure or convince a victim into acting [5]. It is through these techniques that users identify malicious emails and avoid them.

With the rise of machine learning, the threat of these attacks is ever more significant, especially considering how accessible Large Language Models (LLMs) have become to the broader public. This added layer of complexity makes phishing detection a much more laborious task, increasing the demand for automated systems that check and warn users against phishing [6].

## 1.2 Problem Formulation

A survey of existing anti-phishing implementations found that they rely primarily on URL scanning and ad blocking. Extensions claiming to have machine learning implementations have not provided documentation on how these systems are utilized. This indicates a lack of development in email textology analysis and the implementation of machine learning, signalling a security gap against attacks that focus on persuasion and false intimidation.

## 1.3 Objectives

This project seeks to address the growing sophistication of phishing attacks by designing a federated learning-based phishing detection system capable of performing email analysis while collaboratively improving a shared global model, ensuring consistent service-wide updates. It attempts to accomplish this goal by following the following objectives:

1. Implement and evaluate a federated learning model that utilizes lightweight local models capable of accurately predicting phishing emails across distributed clients.
2. Develop and test a system that processes sensitive emails locally while sharing updated model weights with a centralized global model securely without transmitting raw email data.
3. Design and develop a Google Chrome browser extension that enables users to manually flag phishing emails, correct local model predictions, and perform a targeted input analysis.
4. Enable the system to scrape from existing email service providers (e.g., Gmail) to automatically retrieve, read, and analyze incoming email content in real time for phishing detection.

## 1.4 Report Organization

This report documents and evaluates the progress achieved throughout development of the proposed federated phishing detection system, including technical findings, design decisions, and challenges encountered during development. To provide a structured and comprehensive discussion of the project, the report is organized into the following chapters:

- Chapter 2 presents a review of related work and a comparative analysis of existing phishing detection solutions and federated learning approaches.
- Chapter 3 discusses the system's requirements, design, architectural components, and system workflows.
- Chapter 4 details the implementation of the system which includes development phases, technology stack, and core algorithms.
- Chapter 5 presents testing and evaluation methodology and performance results and metrics of the system.
- Chapter 6 concludes the report by summarizing project outcomes and recommendations for future improvements and directions for future research.

# 2. Literature Review & Comparison

## 2.1 Review of Related Work

Phishing is one of the most persistently damaging categories of cyberattack, and the body of research surrounding it reflects both the depth of the threat and the variety of countermeasures that have been proposed over time. This section reviews existing work across four interconnected areas: the evolving nature of phishing attacks, machine learning-based detection methods, transformer-based natural language processing approaches, and federated learning architectures for privacy-preserving classification.

### 2.1.1 Machine Learning Approaches to Phishing Detection

Gangavarapu, Jaidhar, and Chanduka [7] conducted a systematic review of machine learning applications in spam and phishing email filtering, examining over various published systems across a range of classical and deep learning methodologies. Their review found that feature engineering plays a critical role in the effectiveness of classical approaches such as support vector machines and random forests, and that models relying on handcrafted features tend to degrade in performance when faced with novel phishing campaigns that deliberately alter their surface-level indicators. This brittleness is a key limitation of rule-based and feature-engineered systems.

Abdelhamid, Ayesh, and Thabtah [8] proposed an associative classification approach to phishing detection and demonstrated competitive results with simpler training requirements. However, their model similarly depended on static feature sets derived from URL and HTML content, making it vulnerable to attacks that specifically avoid these signals. Shirazi et al. [9] later addressed adversarial robustness by using autoencoder-synthesized data to augment phishing training sets, improving detection rates against obfuscated campaigns. While their approach produced measurable improvements in robustness, it introduced additional training complexity and still operated under a centralized data collection assumption.

Apruzzese et al. [10] examined the role of machine learning in cybersecurity more broadly, noting that the adversarial nature of the problem means that detection systems face a fundamentally different challenge from typical classification tasks. An attacker who is aware of the detection mechanism can iteratively probe and adapt their attacks until the classifier fails, making continuous model updating a practical necessity rather than an optional enhancement. This observation directly supports the motivation for a federated learning architecture where

local models can be refined on newly observed phishing attempts without requiring raw email data to leave the user's network.

### 2.1.2 Transformer-Based NLP for Email and Text Classification

The transformer architecture introduced by Vaswani et al. [11] fundamentally changed the landscape of natural language processing by replacing recurrent architectures with self-attention mechanisms that better capture long-range dependencies in text. This shift made it possible to train much larger models on text data, leading to a new generation of contextual language representations.

Devlin et al. [12] introduced BERT, a bidirectional transformer pre-trained on large text corpora, and demonstrated that fine-tuning a pre-trained BERT model on downstream tasks such as sentence classification significantly outperformed task-specific architectures trained from scratch. The contextual embeddings produced by BERT encode semantic meaning rather than just lexical frequency, making the model capable of detecting phishing signals that rely on persuasive language construction rather than specific keyword occurrences.

Sanh et al. [13] subsequently proposed DistilBERT as a distilled version of BERT that retains approximately 97% of BERT's performance on downstream benchmarks while reducing the model size by 40% and improving inference speed by 60%. This trade-off makes DistilBERT particularly well-suited for deployment on end-user devices where memory and computational constraints are a real concern. For a phishing detection system embedded in a browser extension running alongside a local inference server, DistilBERT's efficiency characteristics are preferred.

Maneriker et al. [14] applied transformer-based models specifically to phishing web page detection and showed that transformer encoders outperformed traditional approaches when trained on URL string data. While their system focused on web pages rather than email body content, their findings confirm the generalizability of transformer models to phishing-adjacent tasks. The current project extends this line of work by applying a fine-tuned DistilBERT model to full email text, which provides richer contextual signals than URLs alone.

### 2.1.3 Federated Learning: Foundations and Security Implications

McMahan et al. [15] introduced Federated Averaging as a practical algorithm for training machine learning models across distributed clients without sharing raw data. Their key insight was that local gradient updates could be aggregated into a single global model update through weighted averaging, and that this process was robust to a significant degree of data heterogeneity across clients. The original FedAvg paper demonstrated convergence properties

across both IID and non-IID data distributions, establishing the algorithm as the standard baseline for federated learning research.

Bonawitz et al. [16] followed with a systems-level treatment of federated learning at scale, addressing practical concerns such as asynchronous client participation, dropped connections, and round-based aggregation strategies. Their design principles informed many of the architectural decisions in this project, particularly the use of threshold-based and timeout-based round finalization to handle the reality that not all clients will be available simultaneously.

McMahan et al. [17] further extended the federated learning framework to incorporate differential privacy by adding calibrated noise to model updates, showing that privacy guarantees could be maintained even at scale.

Kairouz and McMahan [18] provided the most comprehensive treatment of open problems in federated learning, covering topics from system heterogeneity and communication efficiency to privacy-utility trade-offs and fairness. Their work frames federated learning not as a solved problem but as an active research frontier, which contextualizes the design choices in this project as pragmatic implementations of a developing paradigm rather than deployments of a fully mature technology.

Fang et al. [19] examined the security vulnerabilities of federated learning systems from an adversarial perspective, specifically demonstrating that local model poisoning attacks could significantly degrade the accuracy of the global model even when Byzantine-robust aggregation strategies were in place. Their findings highlight an important limitation of the FedAvg algorithm: it does not inherently protect against malicious or compromised clients. This is a relevant consideration for the long-term robustness of the system described in this report, even though it falls outside the current project scope.

Li et al. [20] conducted a comprehensive survey of federated learning systems, categorizing approaches by their communication architecture, aggregation strategy, and privacy mechanisms. Their taxonomy is useful for situating this project within the broader federated learning landscape. The proposed system falls into the category of a synchronous, parameter-server-based federated learning architecture with partial tolerance for asynchronous participation, consistent with the design patterns identified in Li et al.'s survey as appropriate for small-to-medium scale deployments.

### 2.1.4 The Dataset Foundation

Mendes, Maia, and Praça [21] introduced the MEAJOR Corpus, a multi-source dataset for phishing email detection that aggregates samples from several previously published phishing

corpora. Its breadth across phishing categories and its availability as a clean, labeled dataset made it a suitable foundation for fine-tuning a binary classification model. The dataset's multi-source construction also reduces the single-corpus bias that has affected several prior phishing detection studies, where models trained on one dataset failed to generalize to phishing emails drawn from different campaigns or time periods.

### 2.1.5 Existing Anti-Phishing Extensions

Existing anti-phishing browser extensions primarily focus on URL scanning and ad blocking. While some claim to utilize machine learning, there is a documented lack of transparency regarding their specific implementations. Recent research has highlighted the demand for real-time identification of phishing attacks through machine learning to combat the rise of LLM-generated threats.

Due to a lack of existing comparative research on anti-phishing browser extensions, the team has conducted their own comparative analysis of existing anti-phishing browser extensions. The metrics used are based on the functional goals and objectives of the project along with restrictive information such as paywalls and account requirements.

## 2.2 Comparative Analysis Table

| Name | Email | URL | Active Prot. | Passive Prot. | Manual Flagging | AI Assisted | Premium Ver. |
|---|---|---|---|---|---|---|---|
| Ublock Origin | x | ✓ | ✓ | ✓ | ✓ | x | x |
| Netcraft Extension | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x |
| Bitdefender Traffic Light | x | ✓ | ✓ | ✓ | x | x | ✓ |
| Avast Online Security | x | ✓ | ✓ | ✓ | x | x | ✓ |
| Malwarebytes Browser Guard | x | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| PhishGuard | x | ✓ | ✓ | ✓ | x | x | x |
| SafeLink Guard | x | ✓ | ✓ | ✓ | x | x | x |
| Norton Safe Web | x | ✓ | ✓ | ✓ | x | x | ✓ |
| Microsoft Defender Browser Protection | x | ✓ | ✓ | ✓ | x | x | x |
| Browsershield | x | ✓ | ✓ | ✓ | x | x | x |
| Antiphish | ✓ | x | x | ✓ | x | ✓ | ✓ |
| EmailPhishGuard | x | ✓ | ✓ | ✓ | x | x | x |
| AegisWeb3 | x | ✓ | ✓ | ✓ | x | ✓ | x |
| Mozo Online Security | x | ✓ | ✓ | ✓ | x | ✓ | ✓ |

| Name | Email | URL | Active Prot. | Passive Prot. | Manual Flagging | AI Assisted | Premium Ver. |
|---|---|---|---|---|---|---|---|
| Ghostery Tracker & Ad Blocker | x | ✓ | ✓ | ✓ | x | ✓ | x |
| NoPhish | x | ✓ | ✓ | ✓ | x | x | x |
| Visilant | x | ✓ | x | ✓ | x | x | x |

A significant portion of existing solutions focus on URL-based phishing prevention rather than direct email content analysis as seen in Table 1. Extensions such as uBlock Origin, Bitdefender TrafficLight, Malwarebytes Browser Guard, and Norton Safe Web mainly utilize blacklist filtering, website reputational services, malicious domain detection, and active webpage blocking. These systems provide strong real-time URL protection but lack the contextual analysis needed against phishing email prevention. As phishing attacks utilize sophisticated language patterns and persuasive text, purely URL-based phishing protectors are inadequate counters against socially engineered phishing.

Although several tools utilize active protection mechanisms that automatically warn or block users from accessing suspicious websites, these methods are heavily dependent on previously identified phishing indicators and centralized threat databases. This raises the likelihood of zero-day phishing campaigns bypassing detection.

The table also highlights that email-focused phishing detection remains limited among mainstream browser extensions. Reviewed solutions primarily focus on URL scanning and ad blocking instead of email body analysis. Email related solutions are also usually requiring a paywalled premium subscription such as Bitdefender, TrafficLight, and Avast Online Security. This presents a significant accessibility limitation for users seeking phishing email protection mechanisms.

Community-driven reporting and manual flagging capabilities are also inconsistently implemented across existing solutions. Extensions such as Netcraft Extension support community reporting workflows while other extensions provide whitelist-only flagging or no user correction mechanisms at all. The lack of user-feedback systems reduces the adaptability of many anti-phishing tools against evolving phishing campaigns.

These limitations further justify development of the proposed system which combines local email content analysis, machine learning enhancement, user-assisted retraining, and browser extension integration through a unified phishing platform. This proposed approach aims to preserve user privacy by ensuring emails and user information remains on the client device

while still enabling collaborative improvement through a centralized global phishing detection model.

## 2.3 Conclusions on Literature

Taken together, the reviewed research and the comparative extension analysis paint a clear picture of where the field currently stands and where meaningful gaps remain.

Early approaches relied on rule-based classifiers and handcrafted URL features, which were effective against the phishing campaigns of their time but proved increasingly brittle as attackers adapted. Machine learning brought greater adaptability, and transformer-based models introduced a qualitative leap in the ability to detect phishing through semantic analysis rather than surface-level pattern matching. The development of DistilBERT [13] made these capabilities practical on resource-constrained devices, and the MEAJOR Corpus [21] provided a consolidated training foundation. What this body of work demonstrates is that the tools for high-accuracy, content-aware phishing detection now exist. The challenge is deploying them in a way that respects user privacy and integrates naturally into the browser environment where phishing attacks are most commonly encountered.

The comparative extension analysis in Section 2.2 reveals that existing browser-based tools have not kept pace with this research progress. The overwhelming majority of reviewed extensions operate at the URL level and depend on centralized threat databases, making them structurally blind to the category of attack that Hazell [6] identified as most concerning: LLM-generated phishing messages that contain no malicious links and succeed entirely through persuasive language. Extensions that do claim email-level analysis are either opaque about their methodology or locked behind premium paywalls, creating an accessibility gap for ordinary users who would benefit most from content-aware protection.

The federated learning literature [15, 16, 18] provides the architectural framework needed to close this gap without introducing new privacy risks. A centralized content analysis service would require raw email text to leave the user's device, which is unacceptable given the sensitive nature of email content. FedAvg resolves this by ensuring that only model weight updates, not email data, are transmitted to the central server. The systems work of Bonawitz et al. [9] shows that this architecture can handle the practical realities of asynchronous participation, and McMahan et al.'s work on differential privacy [17] points toward future extensions that could provide formal privacy guarantees on top of the architectural separation already in place.

The adversarial findings of Fang et al. [19] and Apruzzese et al. [10] serve as important reminders that machine learning-based security systems are not immune to attack. A deployed system will face a distribution shift as phishing campaigns evolve, and the user-correction and federated retraining loop built into this project is a direct response to that challenge. The work of Gangavarapu et al. [7] reinforces that continuous updating is not a nice-to-have feature but a structural necessity for any phishing detection system intended for long-term use.

In summary, the reviewed work demonstrates that content-based phishing detection using transformer models is both technically feasible and demonstrably effective, that federated learning provides a viable privacy-preserving architecture for delivering such detection at the browser level, and that the existing landscape of browser extensions has not yet integrated these capabilities in an accessible, transparent, or user-correctable form. These three observations collectively justify the design of the system described in this report, which combines local DistilBERT inference, FedAvg-based collaborative model improvement, manual correction workflows, and Gmail integration into a unified, privacy-first phishing detection platform.

# 3. System Analysis & Design

This chapter presents the system analysis and design of Flare, a federated learning-based phishing email detection system. It begins with an overview of how requirements were gathered, followed by the functional and non-functional requirements of the system. A system analysis is then conducted through use case diagrams to model user and system interactions, followed by a detailed system design covering the high-level architecture, component responsibilities, and the entity-relationship diagrams for both the client and server databases.

## 3.1 Requirements Gathering

Requirements were gathered primarily through analysis of existing systems as shown in Table 1. However, due to the lack of existing systems with similar objectives as this one, some extrapolation had to be done to adapt the requirements to this current system.

## 3.2 Functional & Non-Functional Requirements

Table 3.1 Functional Requirements

| ID | Requirement Description | Priority | Status |
|---|---|---|---|
| **FR-1** | The system shall classify email text as phishing or legitimate using a fine-tuned DistilBERT model, returning a predicted label and a confidence score. | High | Met |
| **FR-2** | The browser extension shall automatically detect opened Email messages and display a warning popup on the page when the email is classified as phishing. | High | Met |
| **FR-3** | The user shall be able to override a prediction (Mark as Phishing / Mark as Legitimate); the flagged email shall be stored locally and the override cached so rescans return the corrected result. | High | Met |
| **FR-4** | The local client server shall fine-tune the model on user-flagged emails once flag threshold is reached, then hot-reload the updated model without restart. | High | Met |
| **FR-5** | The local client shall transmit the post-training weight delta to the central server, which shall run FedAvg aggregation on a hybrid threshold/timeout schedule and distribute the new global model back to the clients | High | Met |

| | | | |
|---|---|---|---|
| **FR-6** | The client server shall serve an admin dashboard showing flag totals, false positive/negative time series, history of flagged emails, and settings | Medium | Met |
| **NFR-1** | Privacy: Raw email content shall never leave the client server. Only model weight updates are transmitted to the central server. | High | Met |
| **NFR-2** | Accuracy: The base model shall achieve at least 90% accuracy and 90% F1 on the test set. | High | Met |
| **NFR-3** | Compatibility: The extension shall run on Chrome MV3, and the client/server stack shall run on CPU and CUDA-enabled GPUs. | Medium | Met |
| **NFR-4** | Maintainability: The aggregation strategy shall be a swappable component so FedAvg can be replaced (e.g., with FedProx) without changing the API | Medium | Met |
| **NFR-5** | Scalability / Deployment: The same architecture shall support both individual users (personal local server) and enterprise deployments (IT-hosted server with extension pushed to employees) without code changes. | Medium/Low | Met |

## 3.3 High-Level System Design

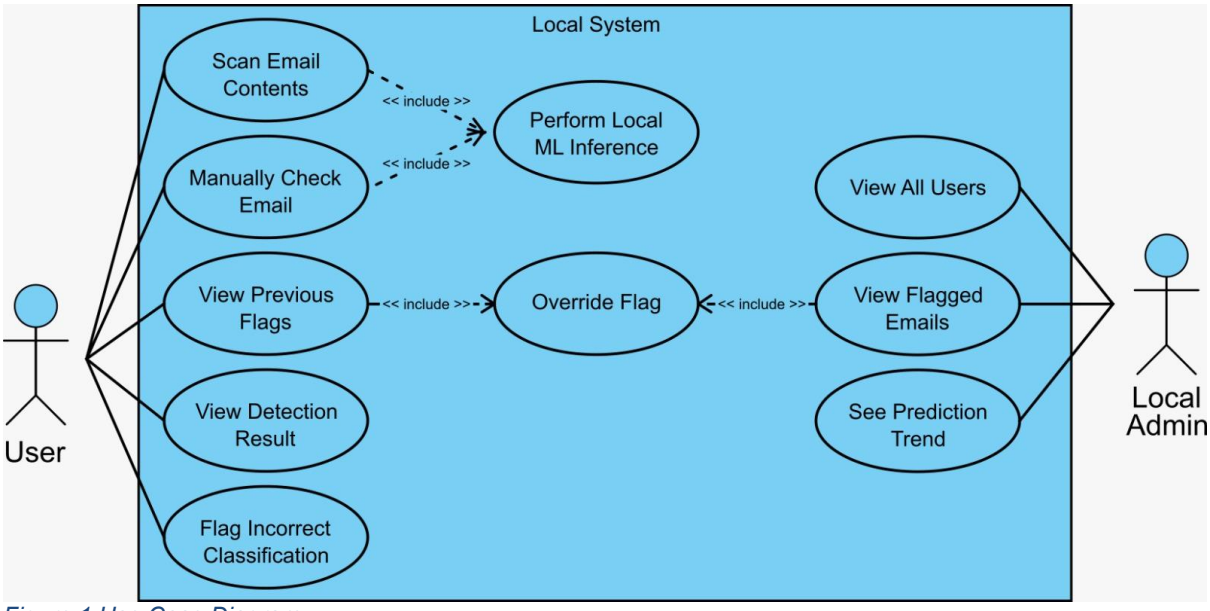



*Figure 1 Use Case Diagram*

Figure 1 presents the use case diagram for the local system, defining the interactions between two primary actors, the User and the Local Admin, within Flare's on-device processing

environment. On the user side, both Scan Email Contents and Manually Check Email include Perform Local ML Inference as a mandatory sub-step, meaning the model runs automatically whenever either action is triggered, keeping inference local to the device in line with the system's privacy-preserving design. Users can also view their previous flags, which extends to Override Flag, as well as view detection results directly and submit feedback through Flag Incorrect Classification when the model produces an incorrect output. On the admin side, the Local Admin has visibility over all users, can view flagged emails, and can monitor model behavior over time through See Prediction Trend. The shared Override Flag use case, reachable from both the user and admin paths, reflects a deliberate design choice to reuse that correction behavior across both roles rather than duplicating it. Collectively, the diagram reflects a system where ML inference is encapsulated locally, human oversight is built in at both the user and administrative levels, and feedback mechanisms are present to support continuous model improvement.

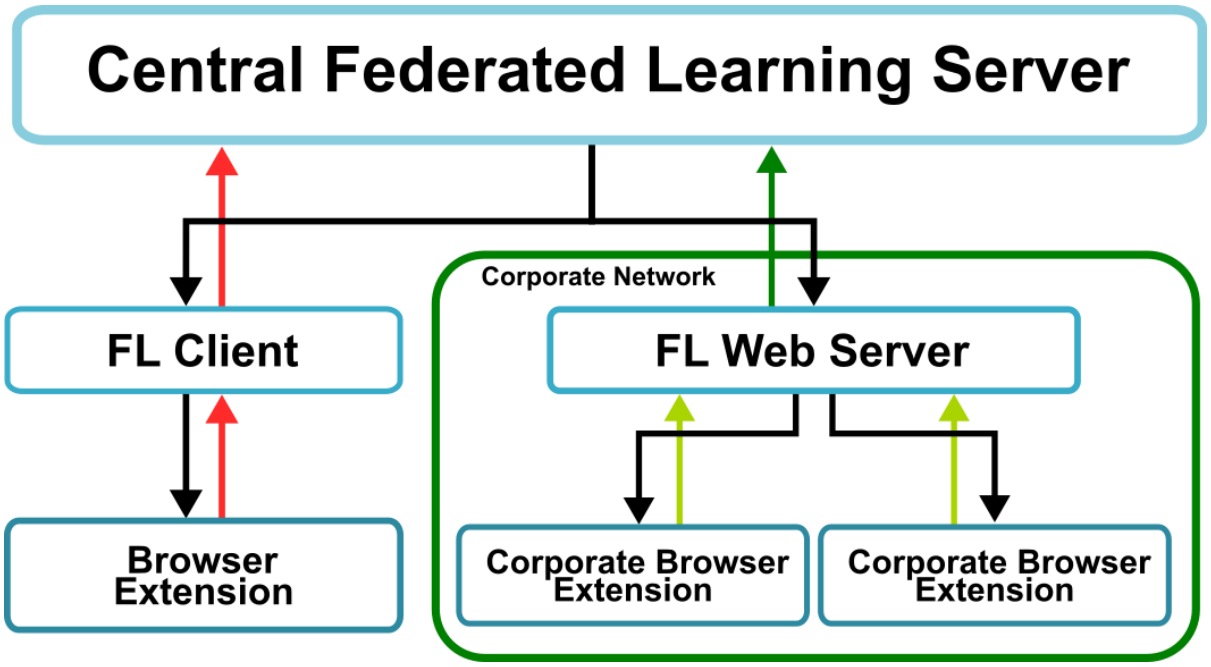



*Figure 2 System architecture with a single-device implementation (left) and a corporate implementation (right)*

The architecture as represented by figure 2 follows a local security model (left) and a corporate implementation of the service (Right) where the browser extension acts as the primary monitoring and response layer. All email classification is done locally within the user's machine or corporate network, only model weight updates are transmitted to the central server through a federated learning workflow. The system can be generalized into three primary layers:

1. Central Federated Learning Server

2. Local Client Desktop Server
3. Chrome Browser Extension

**Central Federated Learning Server**

The central server is responsible for coordinating federated learning across multiple smaller models existing among clients. The server performs model aggregation that combines these updates into a new global model using averaged weights after a sufficient number of updates is received or after a timeout period expires. This ensures individualized privacy and non-IID email distributions across users.

**Local Client Desktop Server**

The local client server acts as the core interface and implements a distilBERT Transformer Model for phishing classification in a user's machine. It downloads the latest global model from the central server during initialization and locally and stores incorrect predictions until a configurable threshold is reached, after which the client sends model weight updates to the federated server.

**Browser Extension**

The browser extension serves as the user-facing interface of the system. The extension displays phishing warnings directly and provides additional features such as manual email checking, confidence visualization, and user override functionalities. User correction are locally cached allowing rescans to preserve manually corrected classifications.

# 4. Implementation Details

## 4.1 Technology Stack

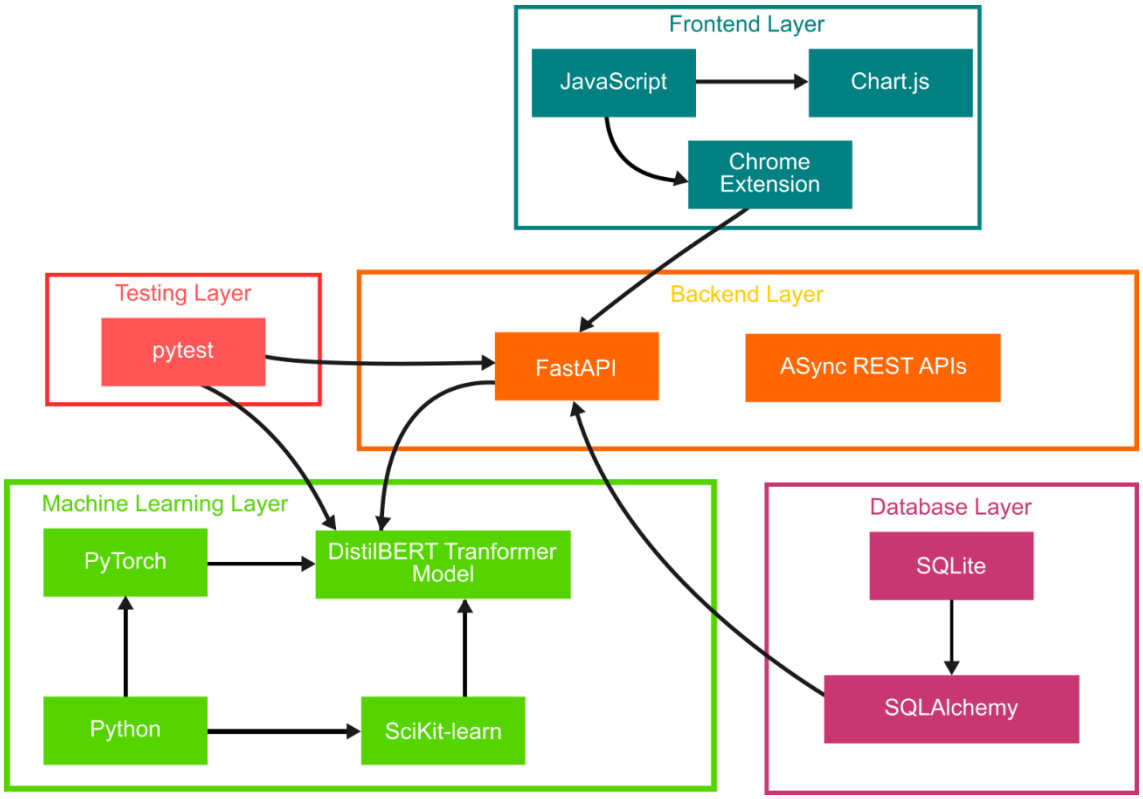



*Figure 3 Tech Stack*

Figure 3 portrays the tech stack of the system.

**Programming Languages:**

- Python 3.14: Used for training and running the model, and for running the HTTP web servers for the federated server and client. Python is the standard for machine learning projects and provides a vast array of ML libraries to utilize, and frameworks like FastAPI allow writing fast and easy to use web APIs.
- Javascript: Needed for writing all logic for the MV3 Chrome extension and for interacting with the python web API. Native javascript was mostly used due to the ease of development (no bundling or built pipeline required) and the requirements not needing any complex state or component management. Only the Chart.js library was used for a simple line graph.

**Frameworks:**

- FastAPI 0.135.3: Used as the backend framework for both the local client server and the central federated server. FastAPI was chosen over Flask and Django REST

Framework because it provides native asynchronous support and automatic request and response validation through Pydantic models. Flask and Django either required extra libraries or introduced too much overhead for a system that only needed to expose a small number of JSON endpoints. FastAPI met all requirements while providing an easy development interface.

- Chrome Manifest V3: Used as the extension framework. Manifest V3 was chosen because Manifest V2 has been deprecated by Google and new extensions submitted to the Chrome Web Store must comply with the V3 specification, making it the only viable option for long-term distribution. The MV3 service-worker and content-script isolation model also provided good isolation between components, resulting in better structured code.

**Database:**

- SQLite 3 (via SQLAlchemy 2.0.49): Used for persistent storage on both the local client and the central server. SQLite was chosen over PostgreSQL and MySQL because it runs as an embedded database without requiring a separate server process, which is essential for the client component that must run on an end-user's machine with no external service dependencies. The workload is also single-writer with low concurrency, where SQLite's simplicity outweighs the scalability benefits of a full database server. SQLAlchemy was layered on top to provide an ORM abstraction, allowing the data access layer to remain unchanged if a future enterprise deployment requires migrating the central server to PostgreSQL.

**Key Libraries / APIs:**

- PyTorch 2.11.0: Chosen as the deep learning framework because it is the standard for transformer-based research and provides mature CUDA acceleration alongside a flexible API, which is better suited to the iterative fine-tuning workflow required by federated learning than the static-graph approach of older TensorFlow versions.
- pytest 9.0.3: Chosen as the test runner over Python's built-in unittest because it offers a more concise syntax, a powerful fixture system, and better support for parametrized tests, all of which reduced the amount of boilerplate required to write the unit and integration test suite.
- Chart.js: Chosen for chart rendering in the admin dashboard because it is a lightweight, declarative charting library that works directly with vanilla JavaScript, avoiding the

steep learning curve and verbose syntax of lower-level libraries such as D3.js for what amounted to a single line graph.

## 4.2 Development Phases

1. **Phase 1 - Planning (Early February 2026):** The planning phase established project scope, technology stack, and team responsibilities. The system was scoped as a privacy-preserving phishing email detection tool combining a fine-tuned language model with federated learning, allowing the model to improve from user corrections without transmitting raw email data off-device. The MeAJOR Corpus was selected as the training dataset. Core technologies were agreed upon, those being PyTorch, HuggingFace Transformers, FastAPI, and SQLite for the ML and server-side components, with Chrome Manifest V3 selected for the browser extension to keep deployment friction low for both individual users and enterprise IT. A clean API boundary was defined between the ML and SE sub-teams so that browser extension development could proceed independently of the ML pipeline; the local client server was identified as the privacy boundary, with the extension communicating only with that local server and never reaching the central aggregation server directly. Two deployment topologies were planned for: an individual configuration where the local server runs on the same machine as the extension, and an enterprise configuration where IT hosts the local server on a LAN-reachable machine and deploys the extension across employee workstations.

2. **Phase 2 - Design (February 12-14, 2026):** The design phase translated requirements into architectural decisions. On the model side this included the data preprocessing and splitting strategy, the selection of DistilBERT as the base model, the federated learning protocol (weight deltas, FedAvg aggregation, hybrid threshold/timeout round management), and database schemas for both the client and server. On the server and extension side, the five-endpoint API contract shared between the two sub-teams was finalized (POST /predict, POST /flag, GET /flagged, plus admin endpoints), along with the message-passing protocol between the extension's three components (content script, background service worker, and popup). Wireframes were produced for the extension popup (status circle, collapsible manual check, flag correction button) and for two dashboard surfaces: the per-user extension dashboard and the multi-user admin dashboard served directly by the local client server.

3. **Phase 3 - Implementation (February 12 – April 7, 2026):** Implementation proceeded in three sequential sub-phases. The data pipeline and centralized training loop were built first, followed by model refinement, best-checkpoint saving, and bug fixes. The federated learning infrastructure, the client node and central aggregation server, was then built out, completing the full federated learning pipeline. In parallel with FL infrastructure work, the client server's API was implemented as a FastAPI application backed by SQLAlchemy and SQLite, exposing inference, flagging, per-user flag retrieval, and a set of administrative endpoints. The browser extension was built as a Manifest V3 Chrome extension consisting of a email content script (only Gmail was implemented but more can be added for each email service) that detects opened emails, a background service worker that manages all HTTP calls to the local server and persists results through chrome storage APIs, and a popup UI that displays a colored status circle (idle, legitimate, phishing) with model confidence, a collapsible manual input panel, and a flag-correction button. Two dashboard surfaces were built on top of the same API: a per-user extension dashboard and a multi-user admin dashboard backed by Chart.js.

4. **Phase 4 - Testing (April 1 – April 9, 2026):** Testing combined unit-level validation of individual modules with end-to-end integration testing of the live FL pipeline. On the model side, two dedicated test scripts were written to simulate client flagging, local training, weight submission, and model aggregation. Six defects were identified and resolved during this phase. On the server and extension side, a pytest-based test suite was added. Unit tests cover the client database query helpers, the FedAvg aggregation math, and the Extract data pipeline. Integration tests were written that drive the live FastAPI application through Starlette's TestClient and verify the end-to-end behavior of every extension-facing endpoint.

## 4.3 Core Algorithms

The system implements two primary algorithms:

- Local Phishing Detection Algorithm
- Federated Aggregation Algorithm (FedAvg)

### Local Phishing Detection Algorithm

This algorithm performs local phishing classification on the user's machine using a fine-tuned DistilBERT model. It is triggered whenever the browser extension extracts email contents and sends it to the local server through FastAPI

```
Algorithm 1: Local Phishing Detection
──────────────────────────────────────────────────────
Input:  email_text — raw email body extracted from Gmail
Output: p_phish, label
Require: Pre-loaded tokenizer T and fine-tuned DistilBERT model M
──────────────────────────────────────────────────────
1: tokens ← T(email_text)
2: x ← Pad-and-Truncate(tokens)
3: logits ← M(x)
4: p ← Softmax(logits)
5: p_phish ← p[phishing_class]
6: if p_phish ≥ τ then
7:     label ← "Phishing"
8: else
9:     label ← "Legitimate"
10: end if
11: return (p_phish, label)
```

**Step-by-step Description**

1. The Chrome extension extracts the contents of a specific email.
2. The email text is sent to the local server through the prediction endpoint.
3. The DistilBERT tokenizer converts email text into transformer tokens.
4. Input sequences are padded and truncated to match model dimensions.
5. The fine-tuned DistilBERT model performs inference.
6. Output logits are converted into probabilities using Softmax.
7. The phishing probability is compared against a classification threshold.
8. The prediction result is returned to the browser extension UI.

Time Complexity: O(n), where n is the character length of the email body. The tokenization step is linear in the email's character length, while the transformer forward pass runs in constant time because the truncation step caps the sequence length at a fixed value and DistilBERT's depth and hidden dimension are architectural constants.

Space Complexity O(P), where P is the total parameter count of the model. The model parameters dominate inference memory, since the per-layer activations stored during the forward pass depend only on fixed architectural constants

### Design Justification

DistilBERT was specifically chosen because it provides:

- Reduced computational overhead with marginal performance decay compared to full BERT
- Faster inference, which is desired in more normal deployments
- Strong contextual performance

Unlike keyword-based systems, transformer embeddings allow for semantic analysis instead of textual information, enabling phishing detection even when phishing keywords are absent. The model specifically uses local inference to contain user emails and information within the client device, enforcing privacy.

### Federated Learning Aggregation Algorithm (FedAvg)

FedAvg allows for combining lightweight local client models into a centralized, more powerful global model without transferring raw email data.

```
Algorithm 2: Federated Averaging Aggregation
──────────────────────────────────────────────────────────────────
Input:  client_updates = {w_1, w_2, ..., w_K} — local weight updates from K clients
        sample_counts  = {n_1, n_2, ..., n_K} — local sample sizes
Output: w_global — updated global model weights
──────────────────────────────────────────────────────────────────
1: N ← Σ_{i=1..K} n_i
2: Initialize w_agg ← 0 (same shape as w_1)
3: for i = 1 to K do
4:     α_i ← n_i / N
5:     for each parameter layer j do
6:         w_agg[j] ← w_agg[j] + α_i · w_i[j]
7:     end for
8: end for
9: w_global ← w_agg
10: return w_global
```

### Step-by-step Description

1. Clients perform local DistilBERT fine-tuning using flagged emails.
2. Updated model weights are calculated locally.
3. Clients send weight updates and local sample counts to the central server.
4. The server computes the total number of samples across all clients.
5. Each client's contribution is proportionally weighted based on its dataset size.
6. Model layers are averaged parameter-by-parameter.
7. The aggregated global model replaces the previous global model.

8. Clients download the updated model.

Time & Space Complexity: O(nP), where n is the number of client updates and P is the total parameter count of the model. The time complexity is O(nP) because the aggregator computes a weighted sum across all n client deltas for every one of the P parameters in the model. The space complexity is also O(nP) because the n deltas are held in memory simultaneously before aggregation, with each delta containing P parameters.

**Design Justification**

FedAvg is implemented for the following reasons:

- Privacy preservation by only transmitting model weights
- Network traffic efficiency compared to centralized dataset uploads
- Better scalability as users only need to join training rounds when joining the system
- Clients can fine-tune their model while still contributing to the shared global model

Additionally, the model also implements the following mechanisms to support asynchronous client participation:

- threshold-based aggregation rounds
- timeout-based round finalization
- hot-reloading of updated models

## 4.4 Interface Implementation

There are two main components that require some form of User Interface (UI). First, the browser extension, which needed UI for the extension popup, phishing alerts, and user dashboard. Second, the FL client needed some form of admin dashboard in order to monitor the flags of each individual extension user and observe general trends.

The UI was mostly made in native Javascript and HTML, with the Chart.js library being used for the chart in the admin dashboard. This decision was made because the UI consisted of only a few render-once surfaces with limited interactive state that a UI framework would need to manage. Frameworks such as React or Vue would have also introduced a build pipeline, bundler configuration, and a runtime overhead disproportionate to the size of the interface, in addition to friction with the Manifest V3 policy which restricts the inline scripts and eval-based tooling that most framework development workflows rely on. Native JavaScript kept the extension build process to a simple edit-and-reload cycle while still meeting all functional UI requirements.

### 4.4.1 Extension Popup

The first UI element is the extension popup. Figure 2 shows the extension popup on startup, with no email scanned yet. The only interactive elements are the dashboard button on the top right, and a “Manual Check” expandable section.

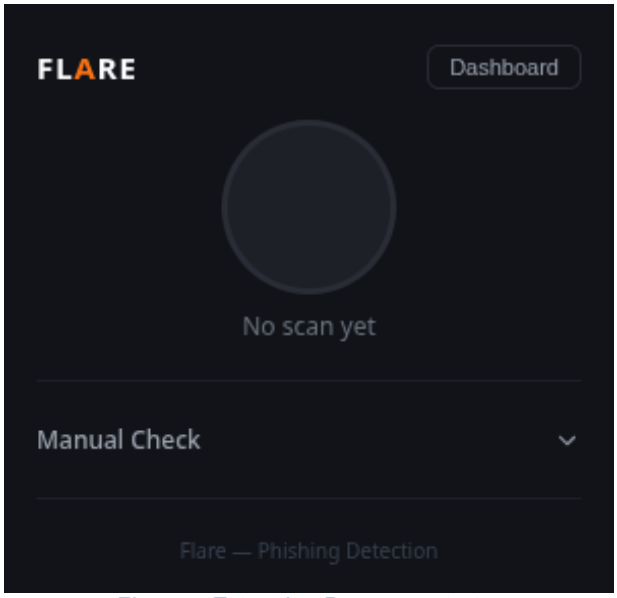



*Figure 4 Extension Popup on startup*

Figure 3 shows the expanded “Manual Check” section which contains a text input box and an “Assess” button. This section is for manually pasting in email contents to be evaluated directly by the model.

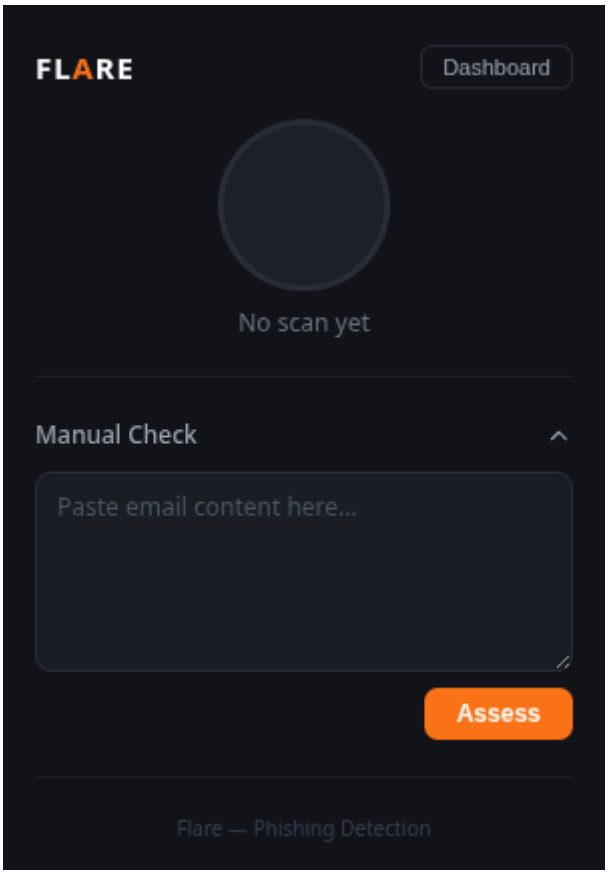



*Figure 5 Extension popup with Manual Check expanded*

The result of an evaluation either through manual assessment or through automatic scraping of an email is shown in Figure 4. If the email is judged to be legitimate, a green check is shown, otherwise, a red cross is displayed. In both cases, the confidence level of the model is provided to the user. There is also a “Mark as X” (X being the opposite of whatever the current judgement is) button for each situation, which is where the user can flag a detection as incorrect. In that case, the flag is saved to be used later for training, and the user is shown their own

flagged result as in Figure 5, which shows the case where a user has marked an email that was predicted to be phishing as legitimate.

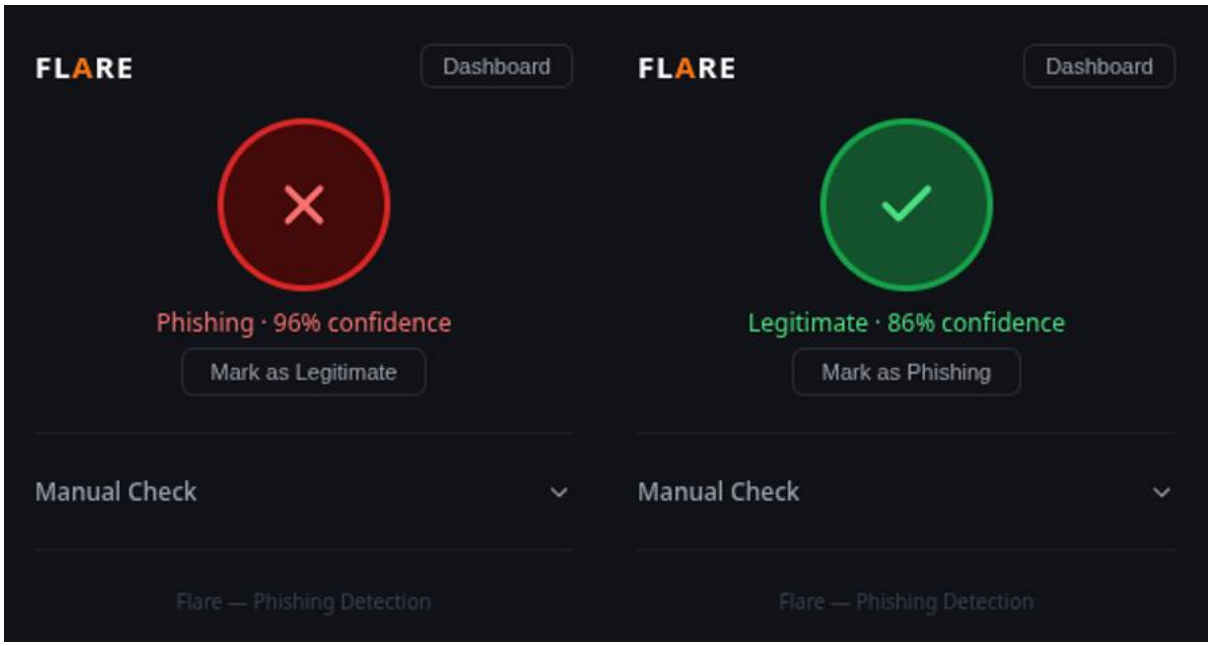



*Figure 6 Assessment if phishing (left) and legitimate (right) with confidence scores*

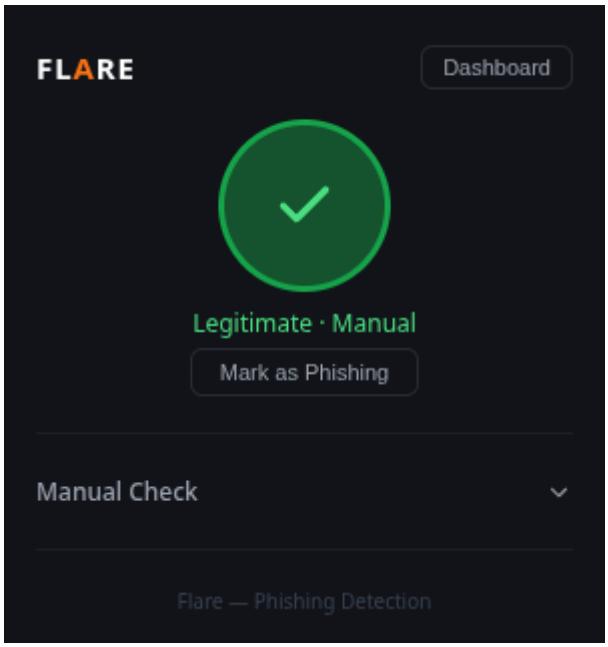



*Figure 7 Popup after manual flag*

### 4.4.2 Phishing Alert

When a scanned email is predicted to be phishing, an alert, shown in Figure 6, is shown to the user in their email webpage.

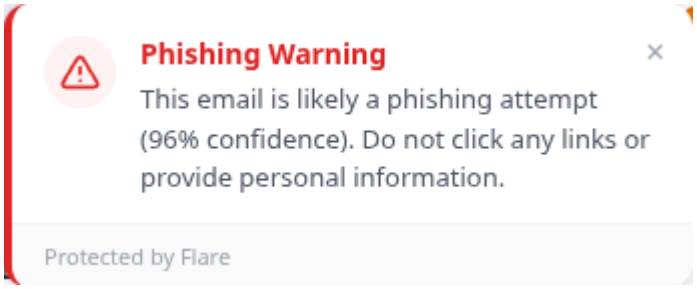



*Figure 8 Phishing Warning Alert*

The warning alerts the user, shows the confidence level, and warns them to avoid behavior that would lead to a security risk.

### 4.4.3 Extension Dashboard

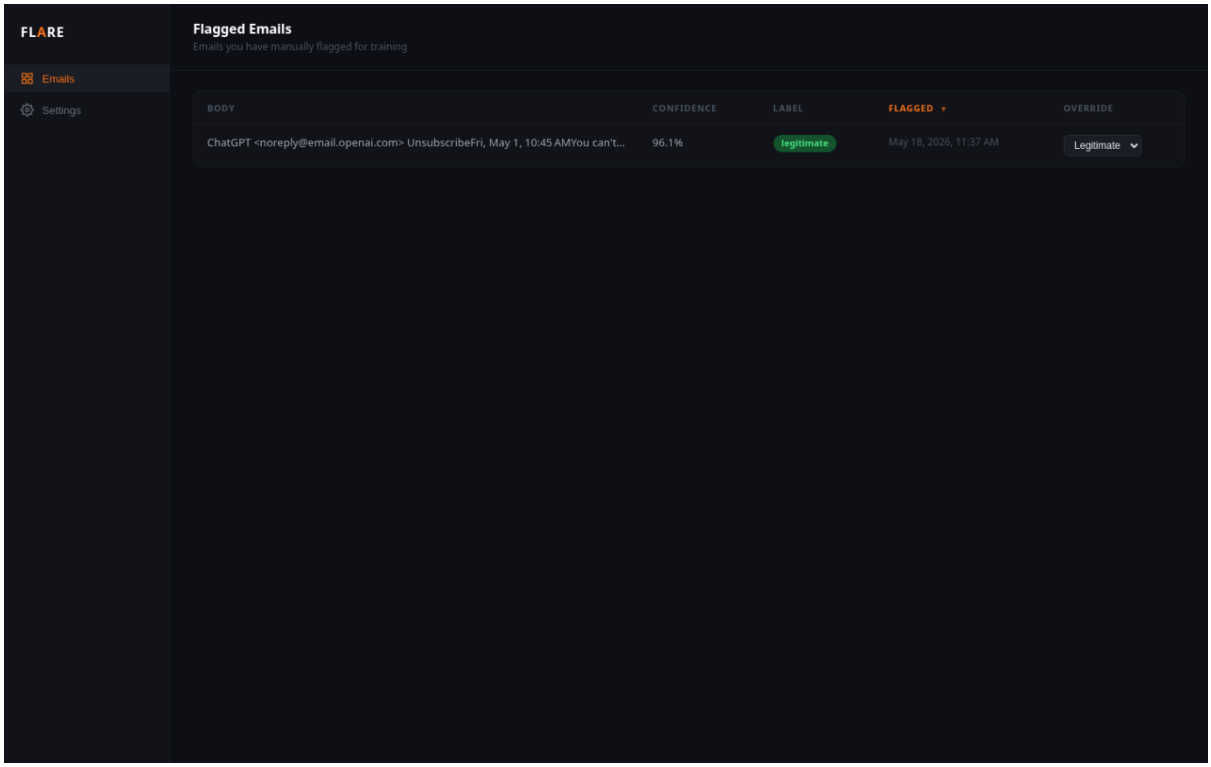



*Figure 9 Extension dashboard*

Shown in Figure 7 is the dashboard for the chrome extension that is reached by clicking the extension button. The dashboard displays information about each email that was manually flagged by the user. For each email, the text body of the email, the confidence level, and the flagged label, and the date of the flagging are shown. There is also an override dropdown for each email that allows the user to change label flag again in case the initial flag was incorrect. By default, emails are sorted by flag date; however, the user can sort by any valid column by clicking on the column name. Clicking the current sort click again allows the user to switch between ascending and descending order.

### 4.4.4 Admin Dashboard

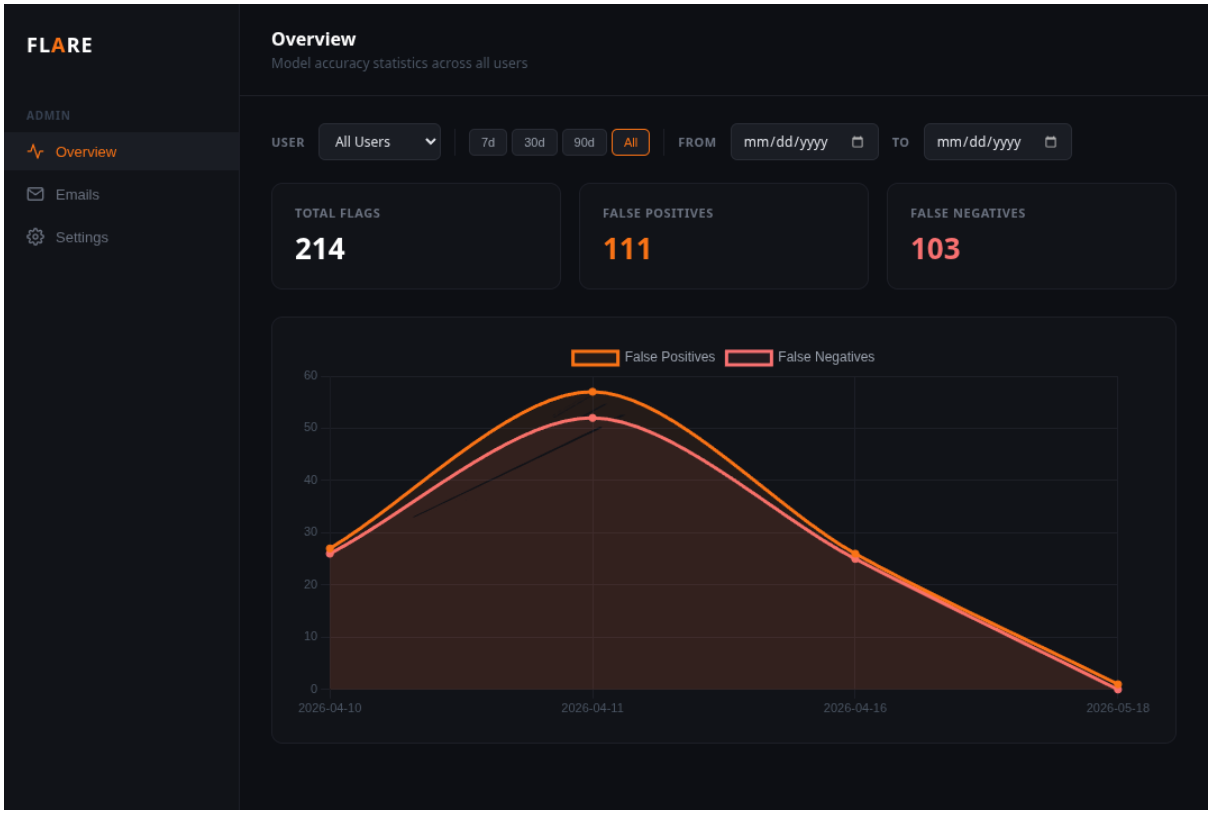



*Figure 10 Admin dashboard overview*

Shown in Figure 8 is the admin dashboard, which can be accessed by whoever is in control of the client server. It is a separate component from the extension, and can be accessed by visiting the /dashboard endpoint of the client server. The admin dashboard contains an additional tab, “Overview”, which is not present in the extension dashboard. In the overview tab, the admin can view the total amount of flags, the amount of false positives (phishing marked as legitimate) and false negatives (legitimate marked as phishing), and a line chart displaying their trends over a specified time period. The admin can choose from several preset filters for the date range or manually specify their own range. They can also filter by extension user to view the data for that specific user.

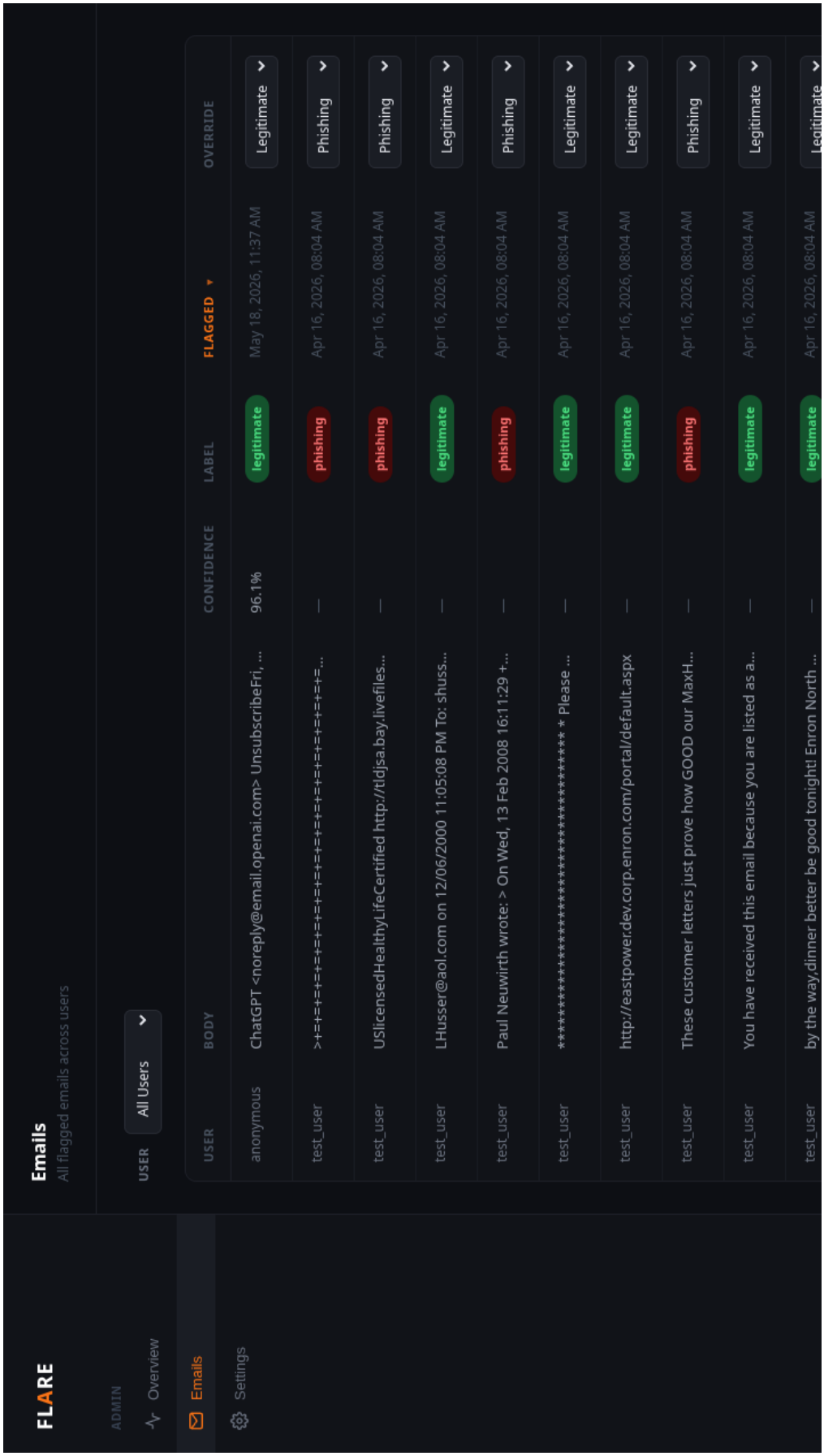



*Figure 11 Admin dashboard Emails*

Figure 9 shows the "Emails" tab of the admin dashboard. It has all the capabilities of the extension dashboard in addition to being able to view all users' emails and filter by specific users.

### 4.4.5 Interface Discussion

As a passive system that mainly runs in the background, a complex and heavily interactive UI was not required. The main function of the system was to alert and inform the user about the potential phishing risk of the emails they encounter, and the UI for the popup and alert were mainly designed to complement that goal. The admin dashboard in comparison is more powerful, allowing the system admin to monitor trends and control all users' flags. Perhaps more development could be done on the analytics capabilities of the admin dashboard, allowing better insights to be made, or a way to export gathered data for input in existing analytics systems.

Having gone over the system design and user interface, chapter 5 presents the testing methodology utilized and the performance of the full system, relating it to the functional and non-functional requirements presented in chapter 3.

# 5. Results, Testing & Evaluation

Several testing strategies were utilized in order to evaluate different performance metrics. Firstly, evaluation of the trained model was performed, computing accuracy, precision, recall, and F1-score metrics. Secondly, automated unit and integration tests were written and performed using the pytest library. Lastly, manual end-to-end testing was done to test the functionality of the most common use-cases of the system.

## 5.1 Comprehensive Test Results

Table 5.2 Comprehensive Test Results

| Test ID | Test Case Description | Expected Result | Actual Result | Status |
|---|---|---|---|---|
| TC-01 | Accuracy of the model evaluated before applying Federated Learning | >= 0.90 | 0.9632 | Pass |
| TC-02 | Recall of the model evaluated before applying Federated Learning | >= 0.90 | 0.9535 | Pass |
| TC-03 | F1 of the model evaluated before applying Federated Learning | >= 0.90 | 0.9633 | Pass |
| TC-04 | FedAvg aggregator testing. Verifies the central server's weight-averaging logic: single-update passthrough, equal-weight averaging of two deltas, weighting proportional to num_samples, tolerance of partial state-dict key mismatches, and rejection when all updates have mismatched keys. | All 5-unit tests pass. Aggregated tensors equal the analytically expected values within torch.allclose tolerance; an all-mismatch input raises ValueError. | All 5 tests passed. Sample-weighted average (75/25 split of deltas 1 and 5) returned 2.0 as expected; mismatched-key updates were dropped without affecting valid ones. | Pass |
| TC-05 | Client flagged-email database testing. Exercises insert/query helpers against an in-memory SQLite: per-user scoping, get_all_flagged_emails filtering, distinct user IDs, the trained flag transition via mark_emails_trained, and the FP/FN grouping in get_flag_stats (with and without user_id filter). | All 8-unit tests pass. count_flagged_emails() counts only trained=False rows; stats group label=0 as false positives and label=1 as false negatives per date. | All 8 tests passed. Inserts persisted with correct user scoping; marking emails trained correctly excluded them from get_untrained_emails; FP/FN counts matched the seeded data. | Pass |
| TC-06 | | All 6-unit tests pass. Cleaned data has no | All 6 tests passed. From a balanced 100-row input, | Pass |

| | | | | |
|---|---|---|---|---|
| | Data extraction & splitting tests. Validates the Extract class against synthetic CSV data (via monkeypatched pd.read_csv): null-body drop, duplicate-body drop, string-to-int label coercion, 80/10/10 train/val/test split sizes, class stratification across splits, and pairwise disjointness of splits. | nulls/duplicates and integer labels; splits sum to the original size with 80/10/10 proportions, preserved 50/50 class balance, and zero row overlap. | splits returned exactly 80/10/10 rows, each with equal class counts and no shared bodies. | |
| TC-07 | Client API integration tests. End-to-end FastAPI TestClient over the local-server endpoints with a stubbed model detector and patched training trigger: /predict response shape, /flag persistence with and without confidence, per-user /flagged scoping, /admin/users, /admin/flagged, /admin/stats, /dashboard HTML response, parametrized code 422 validation on malformed /flag payloads, and FLAG_THRESHOLD (monkeypatched to 2) correctly setting training_triggered=true on the second flag. | All 11 integration cases pass. Endpoints return the documented JSON shapes; invalid payloads return HTTP 422; reaching the flag threshold returns training_triggered: true. | All 11 cases passed. The threshold test returned false on the first flag and true on the second; /dashboard returned text/html content. | Pass |
| TC-08 | Client API Latency testing. A sample short and long email was sent to be evaluated by the model 200 times (100 for short and long each). The time it took to receive each response from the client server was measured using python's time.perfcounter, and useful metrics such as the mean, min, and max were calculated.<br><br>Test ran on ~14 GB RAM and CUDA-enabled GPU. | Response time of less than 100 ms | Short email (n=100)<br>mean: 4.1 ms<br>stdev: 0.2 ms<br>p50: 4.1 ms<br>p95: 4.5 ms<br>p99: 4.7 ms<br>min: 3.9 ms<br>max: 4.7 ms<br><br>Long email (n=100)<br>mean: 13.7 ms<br>stdev: 0.3 ms<br>p50: 13.7 ms<br>p95: 14.1 ms<br>p99: 14.3 ms<br>min: 13.1 ms<br>max: 14.3 ms | Pass |

As shown in Table 5.2, TC-01 to TC-3 cover the bare model performance after pretraining, evaluated using the sklearn.metrics from the scikit-learn python library. Accuracy, recall, and F1 score were all evaluated, and expected to reach at least 0.90 each in alignment with NFR-2. As seen in the table, all three metrics were able to pass that threshold, scoring 0.9632,

0.9535, and 0.9633 respectively. Figure 11 displays the same results as a confusion matrix for more illustration.

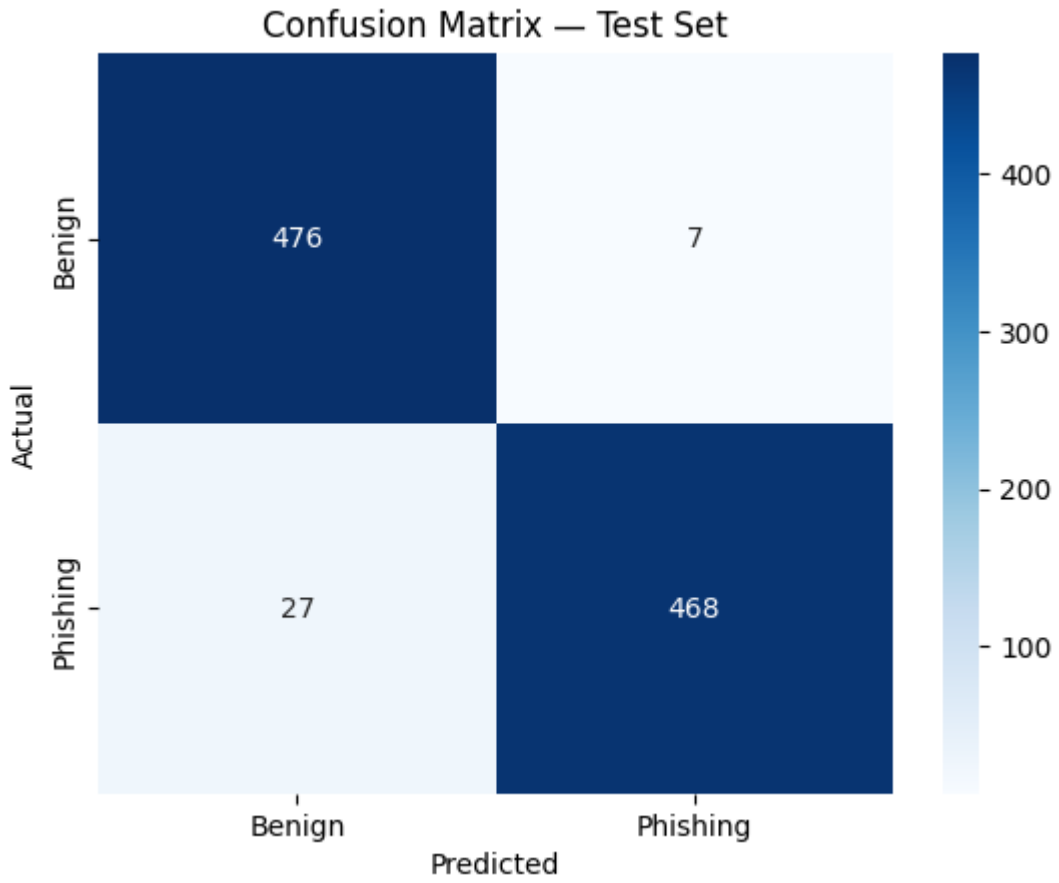



*Figure 12 Confusion Matrix evaluated on the Test Set*

TC-04 to TC-07 were the results of automated testing using the pytest python library. TC-04 to TC-6 were groups of unit tests, and TC-07 was a collection of integration tests, making 31 total tests. The results of each test can be seen in Figure 10, which the path to the file containing each test, the name of the test, and whether it passed or failed. As presented in both Figure 10 and in Table 5.2, all automated tests passed.

```
============================= test session starts ==============================
platform linux -- Python 3.14.4, pytest-9.0.3, pluggy-1.6.0 -- /home/sapling/Flare/.venv/bin/python
cachedir: .pytest_cache
rootdir: /home/sapling/Flare
configfile: pytest.ini
plugins: anyio-4.13.0
collected 31 items

tests/integration/test_client_api.py::test_predict_returns_expected_shape PASSED [  3%]
tests/integration/test_client_api.py::test_flag_persists_and_appears_in_flagged_with_confidence PASSED [  6%]
tests/integration/test_client_api.py::test_flag_without_confidence_stores_null PASSED [  9%]
tests/integration/test_client_api.py::test_flagged_is_scoped_per_user PASSED [ 12%]
tests/integration/test_client_api.py::test_admin_users_returns_distinct PASSED [ 16%]
tests/integration/test_client_api.py::test_admin_flagged_returns_all_or_filtered PASSED [ 19%]
tests/integration/test_client_api.py::test_admin_stats_counts_fp_and_fn PASSED [ 22%]
tests/integration/test_client_api.py::test_flag_triggers_training_at_threshold PASSED [ 25%]
tests/integration/test_client_api.py::test_dashboard_endpoint_serves_html PASSED [ 29%]
tests/integration/test_client_api.py::test_flag_rejects_invalid_payload[payload0] PASSED [ 32%]
tests/integration/test_client_api.py::test_flag_rejects_invalid_payload[payload1] PASSED [ 35%]
tests/integration/test_client_api.py::test_flag_rejects_invalid_payload[payload2] PASSED [ 38%]
tests/unit/test_aggregator.py::test_single_update_applies_full_delta PASSED [ 41%]
tests/unit/test_aggregator.py::test_equal_weighting_averages_two_deltas PASSED [ 45%]
tests/unit/test_aggregator.py::test_weighting_is_proportional_to_num_samples PASSED [ 48%]
tests/unit/test_aggregator.py::test_mismatched_keys_are_dropped_but_others_aggregate PASSED [ 51%]
tests/unit/test_aggregator.py::test_all_mismatched_keys_raises_value_error PASSED [ 54%]
tests/unit/test_client_database.py::test_insert_and_get_by_user PASSED [ 58%]
tests/unit/test_client_database.py::test_confidence_round_trips_and_is_optional PASSED [ 61%]
tests/unit/test_client_database.py::test_get_all_flagged_with_and_without_filter PASSED [ 64%]
tests/unit/test_client_database.py::test_get_distinct_user_ids PASSED [ 67%]
tests/unit/test_client_database.py::test_count_only_includes_untrained PASSED [ 70%]
tests/unit/test_client_database.py::test_mark_trained_excludes_from_untrained_query PASSED [ 74%]
tests/unit/test_client_database.py::test_flag_stats_groups_fp_and_fn_per_date PASSED [ 77%]
tests/unit/test_client_database.py::test_flag_stats_filters_by_user PASSED [ 80%]
tests/unit/test_extract.py::test_clean_drops_null_bodies PASSED [ 83%]
tests/unit/test_extract.py::test_clean_drops_duplicate_bodies PASSED [ 87%]
tests/unit/test_extract.py::test_clean_coerces_label_to_int PASSED [ 90%]
tests/unit/test_extract.py::test_get_splits_returns_80_10_10 PASSED [ 93%]
tests/unit/test_extract.py::test_splits_preserve_class_stratification PASSED [ 96%]
tests/unit/test_extract.py::test_splits_are_disjoint PASSED [100%]

============================== 31 passed in 3.19s ==============================
```

*Figure 13 Output of pytest with all tests passing*

Lastly TC-08 show a latency test for the client API when making prediction requests. Part of the reason DistilBERT was chosen as the classification model was due to the performance benefits on end user devices, so this test serves to show the results of this performance gain. As shown in the table, the maximum possibly acceptable latency would be about around 100 ms, or 1/10 of a second. Considering that the system must warn the user as soon as possible in the case of a possible phishing email, the delay could not reasonably be any longer. The system was able to achieve an average latency of 4.1 ms on short emails, and 13.7 ms on long emails (worst case email length for the model parameters). Both averages are considerably lower than 100 ms, easily clearing the threshold. A distribution of the response time latency over all 200 test requests (100 short – 100 long) can be seen in figure 12.

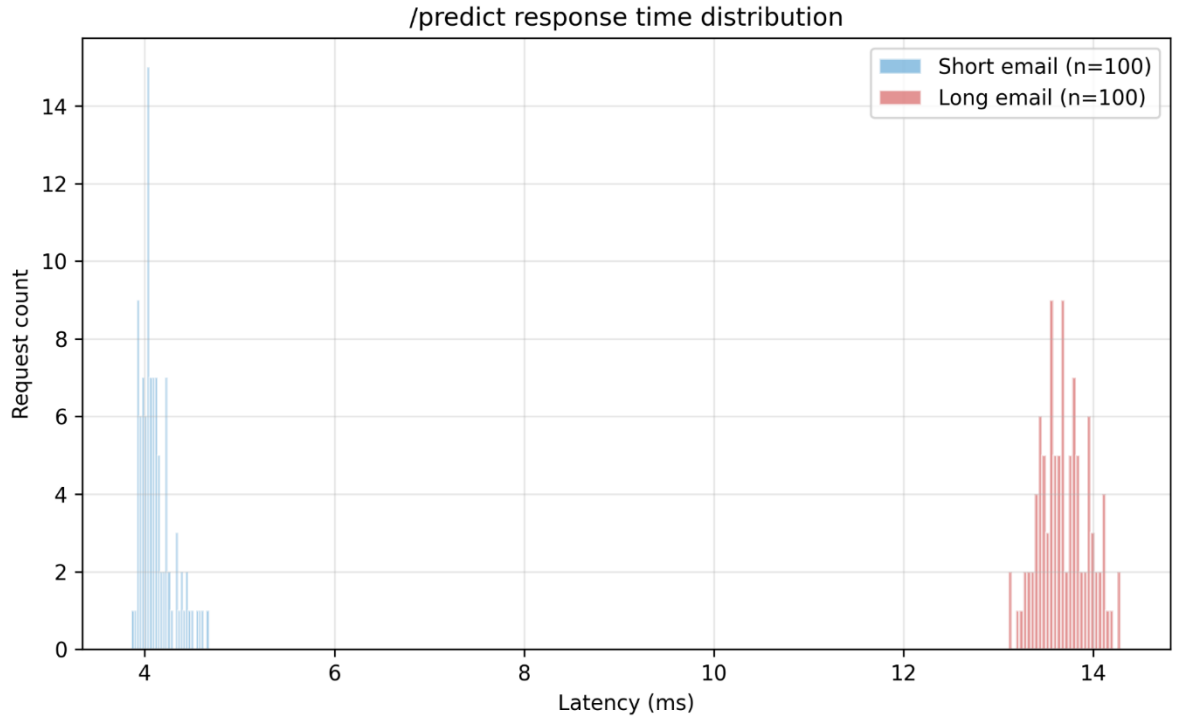



*Figure 14 Model prediction latency distribution for short and long emails*

## 5.2 Analysis of Strengths & Limitations

**Strengths**

- Strength 1: The mode achieved Accuracy, Recall, and F1 >= 90%, which satisfies NFR-2 (TC-1 - TC-3)
- Strength 2: The client server achieved very low response time (<20ms) on worst case email length scenario (TC-8)

**Limitation**

- Limitation 1: The classifier consumes only the email body string, with no parser for URLs, attachments, or images. Image-based phishing attacks can not be detected through this architecture, and would require a possible ensemble setup with multiple models.

- Limitation 2: The dataset used for training is primarily in English, making non-English phishing attacks much less likely to be detected.

# 6. Conclusion & Future Work

## 6.1 Project Conclusion

This project set out to address the growing sophistication of phishing attacks by designing a federated learning-based phishing detection system that combines local email content analysis with collaborative global model improvement. The four objectives defined in Section 1.3 were achieved. Objective 1 was met through the implementation of the FedAvg-based federated learning pipeline, in which lightweight DistilBERT clients contribute weight deltas to a central aggregator that redistributes a new global model to all participating clients; the aggregator's weighted averaging, partial state-dict tolerance, and mismatched-key rejection behaviors were verified through TC-04. Objective 2 was demonstrated by the client-side classification architecture, in which raw email content never leaves the user's network and only model weight updates are transmitted to the central server, satisfying NFR-1 while still allowing the system to reach 96.32% accuracy, 95.35% recall, and 96.33% F1 on the held-out test set (TC-01 through TC-03), exceeding the 90% threshold set by NFR-2. Objective 3 was achieved through the Manifest V3 Chrome extension, which exposes a popup-based scanning interface, a manual "Mark as Phishing / Mark as Legitimate" override flow, a per-user flagged emails dashboard, and an admin overview surface (FR-2, FR-3, FR-6). Objective 4 was fulfilled through the Gmail content script, which automatically extracts opened email content and forwards it to the local client server for inference, with end-to-end response times measured at 4.1 ms (short emails) and 13.7 ms (long emails) in TC-08. Overall, the prototype demonstrates that content-based phishing detection can be deployed in a privacy-preserving, federated manner without sacrificing the accuracy or latency that a practical browser-integrated tool requires.

## 6.2 Recommendations for Future Improvements

While the system meets all defined objectives, several concrete extensions would broaden its coverage and robustness:

- Future Improvement 1: Extend the classifier into a multimodal ensemble that supplements the DistilBERT text branch with dedicated analyzers for URLs, attachments, and embedded images. The current pipeline consumes only the email body string, so image-only phishing and link-obfuscation campaigns can bypass detection; adding a URL reputation/feature model and an OCR-driven image classifier behind the

same /predict endpoint would close this gap without changing the federated aggregation logic.

- Future Improvement 2: Add multilingual support by fine-tuning a multilingual transformer backbone such as mBERT or XLM-RoBERTa on Arabic, French, and other non-English phishing corpora. The MeAJOR-trained model is heavily English-biased, which limits deployment in regions where phishing is increasingly written in local languages; swapping the backbone is backwards-compatible with the existing FedAvg client because only the tokenizer and pretrained weights change.
- Future Improvement 3: Integrate an optional cloud LLM fallback (e.g., GPT, Claude, or Gemini) that is invoked either automatically when the local model's confidence falls below a configurable threshold, or on demand as a user-triggered "second opinion" from the extension popup. This would preserve the privacy-by-default behavior of the system while giving users access to a stronger reasoning model for ambiguous or socially-engineered emails that small local classifiers tend to mishandle.

# References

[1] A. A. Alani and A. Al-Azzawia, “Phishing Attacks Detection and Prevention Techniques: An Overview,” *Journal of Al-Qadisiyah for Computer Science and Mathematics*, vol. 17, no. 1, pp. 166–178, Mar. 2025, doi: 10.29304/jqcsm.2025.17.11972.

[2] J. Hong, “The state of phishing attacks,” *Communications of the ACM*, vol. 55, no. 1, pp. 74–81, Dec. 2011, doi: 10.1145/2063176.2063197.

[3] W. Syafitri, Z. Shukur, U. A. Mokhtar, R. Sulaiman, and M. A. Ibrahim, “Social Engineering Attacks Prevention: A Systematic Literature Review,” *IEEE Access*, vol. 10, pp. 39325–39343, Jan. 2022, doi: 10.1109/access.2022.3162594.

[4] Verizon, *2024 Data Breach Investigations Report*. Verizon Business, 2024. [Online]. Available: https://www.verizon.com/business/en-gb/content/business/us/en/index/resources/reports/dbir/

[5] M. Jakobsson and S. Myers, *Phishing and Counter-Measures: Understanding the Increasing Problem of Electronic Identity Theft*. Hoboken, NJ, USA: Wiley, 2006. doi: 10.1002/9780470086100.

[6] J. Hazell, “Spear phishing with large language models,” *arXiv preprint arXiv:2305.06972*, 2023. doi: 10.48550/arXiv.2305.06972.

[7] T. Gangavarapu, C. D. Jaidhar, and B. Chanduka, “Applicability of machine learning in spam and phishing email filtering: review and approaches,” *Artificial Intelligence Review*, vol. 53, no. 7, pp. 5019–5081, Feb. 2020, doi: 10.1007/s10462-020-09814-9.

[8] N. Abdelhamid, A. Ayesh, and F. Thabtah, “Phishing detection based Associative Classification data mining,” *Expert Systems With Applications*, vol. 41, no. 13, pp. 5948–5959, Mar. 2014, doi: 10.1016/j.eswa.2014.03.019.

[9] H. Shirazi, S. R. Muramudalige, I. Ray, and A. P. Jayasumana, “Improved Phishing Detection Algorithms using Adversarial Autoencoder Synthesized Data,” *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pp. 24–32, Nov. 2020, doi: 10.1109/lcn48667.2020.9314775.

[10] G. Apruzzese *et al.*, “The role of machine learning in cybersecurity,” *Digital Threats Research and Practice*, vol. 4, no. 1, pp. 1–38, Jul. 2022, doi: 10.1145/3545574.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Advances in Neural Information Processing Systems, vol. 30, 2017.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Jan. 2019, doi: 10.18653/v1/n19-1423.

[13] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, Oct. 2019. doi: 10.48550/arXiv.1910.01108.

[14] P. Maneriker, J. W. Stokes, E. G. Lazo, D. Carutasu, F. Tajaddodianfar, and A. Gururajan, “URLTran: Improving phishing URL detection using transformers,” *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, pp. 197–204, Nov. 2021, doi: 10.1109/milcom52596.2021.9653028.

[15] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized\n Data,” *arXiv (Cornell University)*, Feb. 2016, doi: 10.48550/arxiv.1602.05629.

[16] K. Bonawitz et al., “Towards federated learning at scale: system design,” *arXiv (Cornell University)*, Feb. 2019, doi: 10.48550/arxiv.1902.01046.

[17] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, “Learning differentially private recurrent language models,” *arXiv (Cornell University)*, Oct. 2017, doi: 10.48550/arxiv.1710.06963.

[18] P. Kairouz and H. B. McMahan, “Advances and open problems in federated learning,” *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, Dec. 2020, doi: 10.1561/2200000083.

[19] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local Model Poisoning Attacks to Byzantine-Robust Federated Learning," *in Proc. 29th USENIX Security Symposium, Boston, MA, USA, 2020*, pp. 1623–1640.

[20] Q. Li et al., “A survey on Federated Learning Systems: Vision, Hype and Reality for data Privacy and protection,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3347–3366, Nov. 2021, doi: 10.1109/tkde.2021.3124599.

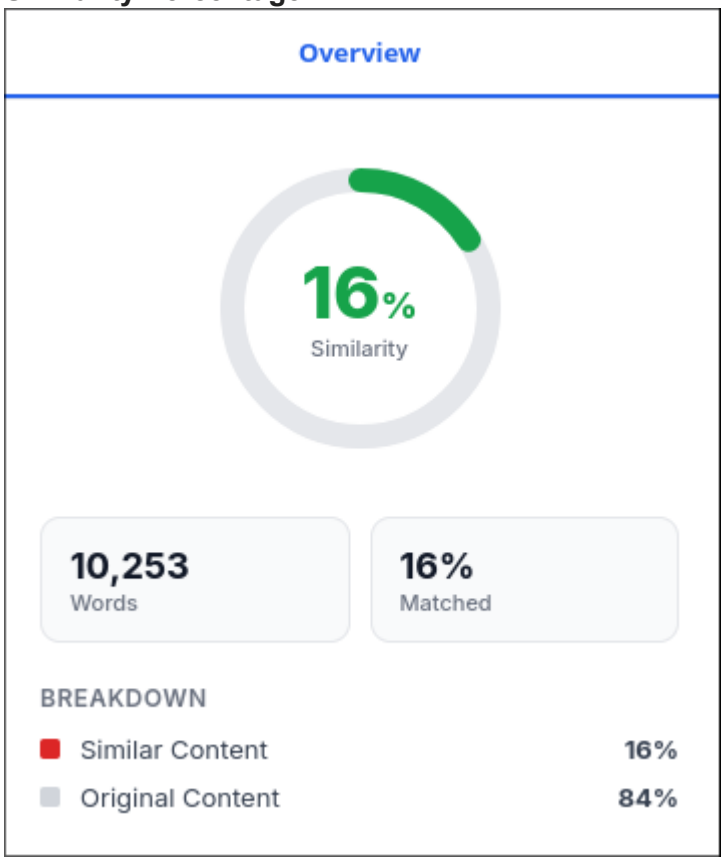
[21] P. Mendes, E. Maia, and I. Praça, “MEAJOR Corpus: a Multi-Source dataset for Phishing email detection,” *ArXiv.org*, Jul. 2025, doi: 10.48550/arxiv.2507.17978.

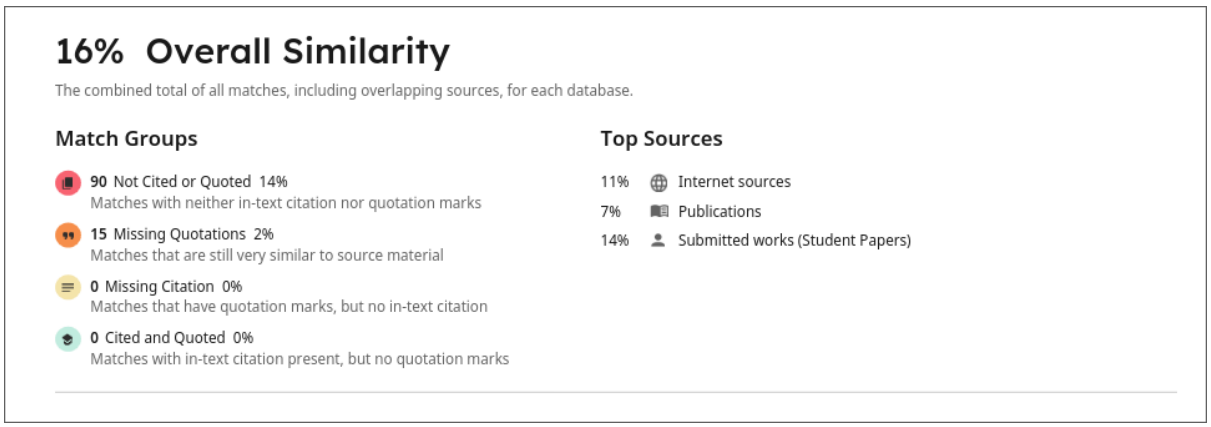
# Appendix A: Plagiarism Report

## Submission Date & Document Title

| File name | Upload Date | |
|---|---|---|
| Phishing Defence - Final_Report Draft-1.pdf | May 19th, 11:31 PM | View Results Download Delete |

## Similarity Percentage




## Source Breakdown Chart

## Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

| | | |
|---|---|---|
| 1 | Internet | |
| ukcatalogue.oup.com | | 2% |
| 2 | Student papers | |
| King's College on 2024-02-14 | | <1% |
| 3 | Internet | |
| www.mais-journal.ru | | <1% |
| 4 | Internet | |
| arxiv.org | | <1% |
| 5 | Student papers | |
| Asia Pacific Instutute of Information Technology on 2025-10-29 | | <1% |
| 6 | Student papers | |
| Melbourne Institute of Technology on 2024-06-02 | | <1% |
| 7 | Internet | |
| ijai.iaescore.com | | <1% |
| 8 | Internet | |
| portal.sinteza.singidunum.ac.rs | | <1% |
| 9 | Publication | |
| Maria Fernanda Cazares, Roberto Andrade, Gustavo Navas, Walter Fuertes, Jhona... | | <1% |
| 10 | Student papers | |
| Montclair State University on 2023-09-10 | | <1% |

| | | |
|---|---|---|
| 11 | Internet | |
| ijrar.com | | <1% |
| 12 | Internet | |
| www.rame.org.in | | <1% |
| 13 | Internet | |
| ijarcs.info | | <1% |
| 14 | Student papers | |
| South Bank University on 2025-03-09 | | <1% |
| 15 | Internet | |
| www.dfki.de | | <1% |
| 16 | Student papers | |
| Liverpool John Moores University on 2026-03-22 | | <1% |
| 17 | Student papers | |
| Middlesex University on 2025-04-11 | | <1% |
| 18 | Student papers | |
| Bahrain Polytechnic on 2025-12-27 | | <1% |
| 19 | Student papers | |
| CSU, Fullerton on 2026-05-06 | | <1% |
| 20 | Student papers | |
| Coventry University on 2025-12-11 | | <1% |
| 21 | Student papers | |
| Universität Bayreuth on 2025-10-22 | | <1% |
| 22 | Student papers | |
| University of Cincinnati on 2026-04-22 | | <1% |
| 23 | Internet | |
| www.itm-conferences.org | | <1% |
| 24 | Student papers | |
| Manchester Metropolitan University on 2024-10-04 | | <1% |

25 Internet
norma.ncirl.ie <1%

26 Publication
Tarafdar, A.. "Predicate control: synchronization in distributed computations with... <1%

27 Student papers
University of Bedfordshire on 2024-05-16 <1%

28 Student papers
University of Westminster on 2026-02-15 <1%

29 Internet
fredoniaregionalhospital.org <1%

30 Internet
journal.lembagakita.org <1%

31 Internet
repository.iutoic-dhaka.edu <1%

32 Publication
Arvind Prasad, Vibhu Yadav, Chirag Solanki, Harshit Goswami, Tanmay Jha, Dushy... <1%

33 Student papers
Brunel University on 2026-03-27 <1%

34 Student papers
Galgotias University, Greater Noida on 2025-12-11 <1%

35 Student papers
Islington College,Nepal on 2026-04-21 <1%

36 Student papers
Lithan Academy Pte Ltd on 2026-05-15 <1%

37 Student papers
University of Wales Institute, Cardiff on 2026-01-14 <1%

38 Publication
Zihang Xu, Chiawei Chu, Shiyang Song. "An Effective Federated Recommendation ... <1%

39 Internet
docplayer.net <1%

40 Internet
dokumen.pub <1%

41 Student papers
FICT on 2026-04-29 <1%

42 Student papers
Liverpool John Moores University on 2024-08-01 <1%

43 Student papers
University of the Highlands and Islands Millennium Institute on 2026-05-01 <1%

44 Student papers
Western Balkans University on 2026-02-25 <1%

45 Internet
khazna.ku.ac.ae <1%

46 Internet
www.diva-portal.org <1%

47 Student papers
Bahrain Polytechnic on 2025-12-28 <1%

48 Student papers
Indira Gandhi Delhi Technical University for Women on 2026-04-27 <1%

49 Student papers
King's College on 2025-08-07 <1%

50 Internet
auai.org <1%

51 Internet
ijser.aliraqia.edu.iq <1%

52 Internet
rosap.ntl.bts.gov <1%

53 Student papers
Glasgow Caledonian University on 2021-07-31 <1%

54 Student papers
Liverpool John Moores University on 2026-04-21 <1%

55 Student papers
Liverpool John Moores University on 2026-04-22 <1%

56 Student papers
MCC Training Institute on 2011-08-05 <1%

57 Publication
Suneeta Satpathy, Álvaro Rocha, Sachi Nandan Mohanty, Tanupriya Choudhury. "... <1%

58 Student papers
Technological University of the Shannon on 2024-08-27 <1%

59 Internet
api-depositonce.tu-berlin.de <1%

60 Internet
digibug.ugr.es <1%

61 Internet
jair.org <1%

62 Internet
www.mdpi.com <1%

63 Publication
"The 22nd International Conference on Information Technology-New Generation... <1%

64 Student papers
Chester College of Higher Education on 2025-10-16 <1%

65 Student papers
Colorado State University Fort Collins on 2020-12-10 <1%

66 Student papers
Coventry University on 2025-12-04 <1%

67 Publication
Dayin Zhang, Xiaojun Chen, Dakui Wang, Jinqiao Shi. "A Survey on Collaborative D... <1%

68 Publication
Joseph Schneible, Alex Lu. "Anomaly detection on the edge", MILCOM 2017 - 2017 ... <1%

69 Student papers
King Abdulaziz University - Scientific Research on 2026-05-10 <1%

70 Student papers
Liverpool John Moores University on 2026-04-10 <1%

71 Publication
Rahim Taheri, Mohammad Shojafar, Mamoun Alazab, Rahim Tafazolli. "FED-IIoT: ... <1%

72 Publication
Song Gao, Yingjie Hu, Wenwen Li. "Handbook of Geospatial Artificial Intelligence"... <1%

73 Student papers
Universiti Kuala Lumpur on 2026-01-23 <1%

74 Student papers
University of Leicester LTI on 2026-04-24 <1%

75 Student papers
University of Nottingham on 2024-09-07 <1%

76 Student papers
University of Sunderland on 2024-05-24 <1%

77 Student papers
University of Warwick on 2024-09-03 <1%

78 Internet
csusm-dspace.calstate.edu <1%

79 Internet
dspace.uii.ac.id <1%

80 Internet
impa.usc.edu <1%

81 Internet
sappan-project.eu <1%

82 Internet
www.hindawi.com <1%

83 Internet
www.techscience.com <1%

84 Student papers
Federal University of Technology on 2023-12-21 <1%

85 Student papers
Glasgow Caledonian University on 2026-04-17 <1%

86 Publication
Uwe Engel, Anabel Quan-Haase, Sunny Xun Liu, Lars Lyberg. "Handbook of Comp... <1%

87 Student papers
Columbia University on 2023-05-09 <1%

88 Student papers
Heriot-Watt University on 2024-08-15 <1%

89 Student papers
Heriot-Watt University on 2026-05-11 <1%

90 Student papers
Lahore Garrison University-Lahore. on 2026-04-08 <1%

# Appendix B: Team Contribution Statement

**Table B.1 Individual Team Contributions**

| Student ID | Student Name | Responsibilities | Contribution % |
|---|---|---|---|
| 20220001125 | Dean Francis Tolero | Dataset preprocessing, DistilBERT model fine-tuning and evaluation, local client server development, central federated learning server development, federated learning pipeline integration, and system testing. | **25%** |
| 20220001658 | Kjeldahl Claisen Cadeliña | Local client server development, literature review, documentation, system testing, presentation and poster preparation. | **25%** |
| 20220001667 | Ralph Bernard Sengco | Dataset selection and preprocessing, literature review, model feedback and quality control, report writing and formatting. | **25%** |
| 20220001825 | Rami Alrajei | Extension logic development, central federated learning server development, front-end development, report writing, federated learning pipeline integration. | **25%** |

Member 1 Signature:  Date: May 15, 2026

Member 2 Signature:  Date: May 15, 2026

Member 3 Signature:  Date: May 15, 2026

Member 4 Signature:  Date: May 15, 2026