People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific ResearchUniversité

M'hamed Bougara - Boumerdes



Faculty of Sciences
Department of Computer Science

Field **:** Mathematics and Computer Science
Major **:** Computer Science
Specialization **:** Computer Science

Final Year Thesis for the Completion of the licence Degree

Topic

Phishing URL Detection Using Deep Learning, Machine Learning, and NLP Techniques

Presented by:

MERAH Assia
LALEG Chahrazed
DERICHE Besma
DOUMAZ Hadjer

Defended on 29/05/2026 Presented to the jury composed of

President: Mr. REZOUG Abdellah
Examiner: Ms. GUNADIZ Safia

Acknowledgments

« First of all, we would like to thank Allah for giving us the strength, courage, and patience to complete our final year project. Without His blessings, none of this would have been possible. »

« We are forever grateful to our Parents, whose endless love, sacrifices, and prayers have always been our greatest support. Their belief in us gave us the motivation to keep going, even during the most challenging times. »

« We express our deepest gratitude and sincere thanks to our supervisor, **Mr. Abdellah Rezoug**, for his constant guidance, valuable advice, and for sharing his expertise with us throughout this project.

His encouragement was a key factor in the completion of this work. »

« We also would like to thank all the members of our project group for their hard work, great cooperation, and team spirit. This project is the result of our shared efforts and dedication. »

« Finally, we thank our families and friends for being with us and supporting us during this important journey. »

Abstract

This final year project presents a web-based system for the automatic detection of malicious URLs, especially links that may be distributed through emails, SMS messages, social networks or other digital channels. The objective is not to analyze the full textual content of an email or SMS, but to focus on the URL itself and decide whether it is legitimate or suspicious. The proposed system uses a hybrid approach composed of four complementary engines: an XGBoost classifier trained on structural URL features, a SecureBERT transformer model for semantic URL representation, a Zero-Day heuristic engine for suspicious patterns, and an optional HTML analysis module that inspects page content when available.

The application was implemented in Python using the Flask framework. It provides a simple web interface for single URL scanning, batch URL analysis and a history module. The system returns a final prediction, a confidence score and detailed scores for each detection engine. The hybrid design improves robustness because it does not rely on a single technique. Machine learning handles statistical patterns, the transformer model captures textual context, heuristic rules identify emerging threats, and HTML analysis adds behavioral evidence from the target page.

Keywords: URL detection, phishing detection, XGBoost, SecureBERT, Zero-Day detection, Flask, cybersecurity, malicious URLs.

Table of Contents

General Introduction 1
Chapter 1: Phishing Attack And Malicious URLs 3
1.2 Definition of phishing 3
1.3 Definition of Malicious URLs 3
1.4 Role of URLs in Phishing Attacks 3
1.5 Types of Dangerous URLS 4
1.6 Why Manual Detection is Difficult 4
1.7 Conclusion 4
Chapitre 2 : Techniques for Malicious URL Detection 5
2.1 Introduction 5
2.2 Whitelist-Based Detection 5
2.3 Heuristic Detection 5
2.4 Machine Learning Approaches 5
2.5 Deep Learning and Transformer Approaches 5
2.6 Comparative Study and model selection 6
2.6 Conclusion 6
Chapter 3 : System Architecture and Detection Methodology 7
3.1 Introduction 7
3.2 General Architecture 7
3.3 URL Preprocessing 7
3.4 XGBoost Feature Extraction 8
3.5 XGBoost Classifier 8
3.6 SecureBERT Semantic Analysis 9
3.7 Zero-Day Heuristic Engine 9
3.8 Optional HTML Analysis 9
3.9 Score Fusion and Decision Logic 10
3.10 Main Feature Extraction Function 10
3.11 URL to Text Transformation for SecureBERT 11
3.12 Zero-Day Detection Examples 12
3.12 API Response Example 13
3.12 List of XGBoost Features 15
3.13 Conclusion 16
Chapter 4 : Implementation and evaluation 17

4 .1 Introduction 17
4.2 Development Environment and Tools 17
4.2.1 Hardware Ressources 17
4.2.2 Software Environment 18
4.2.3 Python Libraries and Frameworks 18
4.3 Hybrid System Architecture 19
4.3.1 Ensemble Fusion Weighting 19
4.3.2 Dynamic HTML content Module Integration 19
4.3.3 Decision Logic and Short-Circuit Rules 20
4.4 Core Component Implementation 22
4.4.1 Feature Extraction Module (36 Features) 21
4.4.2 Deep Learning Module (SecureBERT Transformer) 22
4.4.3 Zero Day Heuristic Engine 23
4.4.4 Web Application and User Interface 23
4.5 Evaluation, Metrics, and Results 25
4.5.1 Final Dataset Composition 25
4.5.2 XGBoost Experimental Performance Results 26
4.5.3 SecureBERT Experimental Performance Results 26
4.6 Conclusion 27
General Conclusion 27

General Introduction

The rapid growth of online services has made URLs a common way for

cyberattacks to enter systems. Attackers often use misleading links to send users to fake lo gin pages, websites that host malware, or fraudulent services. These harmful links can be sent through emails, text messages, instant messaging apps, or social media platforms. A malicious URL might appear almost identical to a real one, especially when attackers use brand names, subdomains, URL shortening servic es, encoded characters, or uncommon top-level domains. Simply looking at a URL manually is not enough because users may not be able to spot fake domains, hidden redirects, or disguised paths. While traditional lists of known malicious websites are helpful, they are not effective against new links that have not yet been identified. This issue has led to the need for automatic systems that can analyze a URL based on its structure, text content, and suspicious patterns to determine if it is harmful..

Problem Statement

The main issue tackled in this project is the following : how can we automatically identify if a URL is safe or harmful by examining the URL itself and by integrating methods such as machine learning, transformer-based semantic analysis, heuristic rules, and optional analysis of the webpage content ?

Objectives

- Develop a web application that receives a URL as input and returns a security verdict.
- Extract meaningful structural features from URLs and train an XGBoost classifier.
- Use SecureBERT to analyze the semantic representation of URLs after tokenization.
- Design a Zero-Day heuristic engine to detect suspicious patterns even when a URL is new
- Add optional HTML analysis for extra evidence such as password fields or low-content pages.
- Provide batch analysis and a history module to improve usability.

Proposed solution

The suggested resolution is a hybrid detection framework executed utilizing Flask. The application integrates four distinct engines: XGBoost, SecureBERT, Zero-Day heuristic detection, and an optional HTML analysis component. The conclusive determination is predicated upon a weighted amalgamation of the individual scores, supplemented by additional override protocols for high-risk scenarios, which include exceedingly elevated Zero-Day scores or significant SecureBERT phishing confidence levels

.

Chapter 1: Phishing Attack and Malicious URLs

1.1 Introduction

This chapter delineates the foundational cybersecurity context pertinent to the project. Given that the application emphasizes URL identification, the discourse is primarily centered on the hyperlinks employed in phishing and scam initiatives, as opposed to a comprehensive analysis of entire email or SMS content.

1.2 Definition of phishing

Phishing isa type of social engineering attack where attackers try to trick people nto giving away important information like passwords, bank details, personal information, or security codes. Often, the attacker sends a message that looks like it's from a trusted company and includes a link to a fake website.[1,2,15]

1.3 Definition of Malicious URLs

A malicious URL is a web address that's created to carry out or help with harmful activities It can take you to a phishing site, download malware, collect login details, trick you into paying money, or offer fake services. The harmful part of the URL can be in the main name, the subname, the path, the query parts, or the content itself.[4,12,]

1.4 Role of URLs in Phishing Attacks

URLs are often the way attackers take a victim from a trusted communication channel to a harmful website. The attacker might use a domain that looks like a real brand, a shortened link, an IP address instead of a full domain name, or a top-level domain that seems suspicious. Because of this, the URL holds many signs that can be checked automatically.[4,12]

1.5 Types of Dangerous URLS

Type	Description	Example
Brand impersonation	A known brand is inserted inside a fake domain or subdomain.	https://paypal-login-example.com
Suspicious TLD	The link uses a top-level domain frequently abused in attacks.	xyz, .tk, .top
IP-based URL	The domain is replaced by a raw IP address	http://192.168.1.10/login
Shortened URL	A shortening service hides the real destination.	https://bit.ly/xxxx
Obfuscated URL	Encoded characters or unusual symbols hide the real structure	%20, @, // in path
Key-based phishing	The URL contains sensitive terms	login, verify, account, secur

1.6 Why Manual Detection is Difficult

Users might focus on the familiar words in a URL and overlook the actual registered domain. For example, a URL might include the word "google," but it doesn't necessarily mean it belongs to google.com. Long URLs are especially hard to check by hand because the harmful part could be hidden within a complicated path or query string.[6]

1.7 Conclusion

Malicious URLs play a key role in phishing and scam attempts.
Spotting them requires more than just looking at them visually. The next chapter will look at the existing methods used to detect these threats.

Chapter 2 : Techniques for Malicious URL Detection

2.1 Introduction

Many different methods have been suggested for identifying harmful URLs.
These methods vary from basic whitelists to more complex techniques like machine learning and models based on transformers. Every approach has its own strengths and weaknesses.

2.2 Whitelist-Based Detection

Whitelist detection checks if a domain is part of a trusted list of real websites.
The app uses this method for well-known domains like Google, YouTube, GitHub, and other reliable sites. Whitelisting helps reduce false alarms for popular, legitimate services, but it's not enough on its own because there are also unknown legitimate websites that aren't on the list.[4,5]

2.3 Heuristic Detection

Heuristic detection uses specific rules to spot suspicious URLs based on certain signs. These signs include things like the presence of an @ symbol, an IP address written directly, strange domain endings, too many subdomains, encoded characters, or words commonly used in phishing attempts. Heuristics are helpful for detecting new threats that haven't been seen before, as they don't rely on past examples.[11,12]

2.4 Machine Learning Approaches

Machine learning models use numerical features from URLs to classify them.
Algorithms like Random Forest, Support Vector Machines, and XGBoost can find patterns in labeled data. In this project, XGBoost is chosen because it works well with structured data and can handle complex relationships between different URL features.[5,6,10,14]

2.5 Deep Learning and Transformer Approaches

Deep learning models can turn URLs into sequences of characters or words.
Transformer models like BERT and SecureBERT are good at understanding text and learning more detailed features. This is helpful because URLs often include important words like brand names, actions, paths, and security-related terms.[7,8]

2.6 Comparative Study and model selection

Previous research on detecting phishing URLs has used various methods, such as traditional machine learning, deep learning, and transformer-based models. Machine learning methods usually rely on manually created features from URLs, whereas transformer-based models treat the URL as a text sequence to better understand the meaning and patterns within the text.

This table provides a short comparison of some works that are related to our proposed approach.

Study	Models Used	Features Type	Relevance to Model Selection
Wang at a1. [17]	PhishBERT	URL text representation	Supports pretrained transformer models for phishing URL detection.
Maneriker and a1. [18]	URLTran, BERT, RoBERTa	URL text sequence	Show the effectiveness of transformers in URL classification.
Aghaei and a1. [19]	SecureBERT	Cybersecurity text	Supports using a cybersecurity-specific language model.
Chitteti and a1. [20]	XGBoost, Random Forest,Extra Tree	Handcrafted URL features	Supports the use of XGBoost and ensemble learning.
Tariq and a1. [21]	XGBoost LSTM, DistilBERT	Structural features + embeddings	Supports combining handcrafted features with semantic representation

Based on this comparison, our project uses a hybrid approach. XGBoost is chosen because it works well with structured, manually created URL features, while SecureBERT is used to examine the URL from a cybersecurity-focused text perspective. By combining these methods, the system can take advantage of both structural analysis of URLs and semantic detection2.6 Conclusion

No single method of detection works perfectly in every situation. This leads to the use of a mixed approach that brings together structured features, meaning-based analysis, rule-based strategies, and the option to check content

Chapter 3: System Architecture and Detection Methodology

3.1 Introduction

The method we're using is based on a system that uses multiple engines working together. The system takes a URL, gets it ready, pulls out important information, calculates different scores, and then puts them all together to make a final decision.

3.2 General Architecture

The structure of the system is shown
The input URL is first made into a standard format and broken down. Trusted websites are checked using a list of allowed domains. Then, the URL is examined by three different tools: XGBoost, SecureBERT, and a special engine for detecting new threats. If turned on, the system also looks at the HTML content. Once all the results are collected, the scores are combined, and the system gives a final result.

3.3 URL Preprocessing

The application makes sure the input URL is in lowercase and removes any extra spaces around it.
If the URL doesn't have a protocol, the system adds a temporary one so it can be parsed properly. Then, the system pulls out parts like the hostname, the main domain, any subdomains, the suffix, the path, and the query string. This information is used again by the feature extraction function and the heuristic engine.

3.4 XGBoost Feature Extraction

The XGBoost model doesn't get the raw URL directly. Instead, the system turns the URL into a list of numerical features. These features cover things like the length of the URL, how many punctuation marks and numbers are there, whether a protocol is used, the entropy of the URL, the presence of subdomains, any suspicious words, brand impersonation, and other structural details.[6,14]

Features group	example
Length features	url_length, domain_length, tld_length, path_length
Punctuation counts	dot_count, hyphen_count, slash_count, at_count, question_count, equal_count
Ratios and entropy	digit_ratio, letter_ratio, digit_count_in_domain
Security indicators	is_https, is_ip, is_shortened, suspicious_tld, has_port
Obfuscation indicators	has_double_slash, has_encoded_chars, special_in_domain
Keyword indicators	has_login, has_verify, has_update, has_secure, has_account, has_admin, has_signin, has_bank
Brand indicators	brand_in_subdomain,brand_in_domain_not_tld
Subdomain indicators	subdomain_count, too_many_subdomains, param_count

3.5 XGBoost Classifier

The training script loads a dataset from prepared_urls.csv, normalizes the labels, and extracts the same features used by the application.
The labels are set as 0 for safe URLs and 1 for phishing or malicious URLs. Once the model is trained, it is saved as phishing_model.pkl, and the list of features is saved as xgb_feature_names.pkl. This ensures that the application uses the same order of features during prediction.[6,10]

3.6 SecureBERT Semantic Analysis

BERT (Bidirectional Encoder Representations from Transformers) is a language model pre-trained on large text corpora using Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), learning bidirectional contextual representations. SecureBERT (Aghaei et al., 2022) is a BERT variant specifically pre-trained on cybersecurity corpora — CVE vulnerability reports, NVD databases, security forums, and threat intelligence articles. It natively understands terms like 'phishing', 'credential harvesting', 'malware', and 'C2 server' in their security context. 8 Fine-Tuning StrategyOur fine-tuning adds a binary classification head (0 = legitimate, 1 = phishing) on top of the SecureBERT encoder. The url_to_text() function converts each URL into semantic text before tokenisation.

Parameter	Value
Base Model	Ehsanaghaei/SecureBERT(Hugging Face)
Training Dataset	160,000 URLs (80,000 phishing + 80,000 legitimate
Epochs	3
Batch Size	64
Optimiser	AdamW (learning rate = 2e-5, weight decay = 0,01)
Max Token Length	128
Scheduler	Linear warmup (10% of steps)
Device	Google Colab T4 GPU

3.7 Zero-Day Heuristic Engine

The Zero-Day heuristic engine has been meticulously engineered to identify anomalous patterns, even in the context of URLs that are novel and not listed in existing blacklists. This mechanism allocates points for various high-risk indicators, including but not limited to, the presence of malicious keywords, brand misrepresentation, dubious top-level domains (TLDs), suspicious IP addresses, non-standard port usage, encoded characters, the presence of the @ symbol, utilization of URL shorteners, excessive hyphenation, an overabundance of subdomains, digit substitution, and the absence of HTTPS protocols. [3,16]

3.8 Optional HTML Analysis

The module responsible for HTML analysis issues a request to the designated URL and subsequently processes the retrieved webpage utilizing the BeautifulSoup library[13]. It elevates the risk assessment score upon the identification of an input field categorized as a password or when encountering a minimal volume of textual content. This analytical process is classified as optional due to its comparatively slower execution and its reliance on network connectivity.

3.9 Score Fusion and Decision Logic

The application calculates a foundational ensemble score by employing weighted contributions derived from SecureBERT, XGBoost, and Zero-Day methodologies. The weight allocations applied within the algorithm are designated as 0.45 for SecureBERT, 0.35 for XGBoost, and 0.20 for Zero-Day. When the HTML analysis feature is activated, the HTML-derived score is amalgamated with the foundational ensemble score. The ultimate threshold established is 0.50.[9,14]

3.10 Main Feature Extraction Function

The main feature extraction process is carried out in a separate Python file called features.py. This file includes a shared function called extract_features(url), which transforms a raw URL into a numerical feature vector that can be used by the XGBoost classifier. Before this change, the same feature extraction code was present in two different files: train_xgboost.py for training and app.py for making predictions. To prevent code duplication and lower the chance of errors, the feature extraction logic was moved into a single file named features.py.

In this project, the function extract_features(url) looks at different aspects of a URL, including its structure, word usage, and security features. It checks things like how long the URL is, how many dots, hyphens, slashes, and special characters it has, whether it uses HTTPS, if the domain is based on an IP address, if it has suspicious top-level domains, if it's a shortened URL, if it has encoded characters, and if it includes phishing-related words like login, verify, update, secure, account, admin, signin, eBayISAPI, or bank. The function also checks for signs of brand impersonation, like the presence of well-known brand names in fake subdomains or misleading domain setups.

The URL is first made lowercase and any extra spaces are removed.
Then, the domain is broken down into parts like the subdomain, main domain, and top-level suffix using the tldextract library. This is important because phishing sites often hide dangerous or misleading parts within subdomains or fake domains. After this, the function calculates 36 numerical features of the URL that are used by the XGBoost model.

Both train_xgboost.py and app.py now use the same extract_features function from features.py. While training, train_xgboost.py applies this function to the dataset URLs before building the XGBoost model. When making predictions in real time, app.py uses the same function to process the submitted URL before giving it to the trained model. This makes sure that the model is trained and used with the exact same way of extracting features.

At the end, the created dictionary of features is turned into a Pandas DataFrame.
The names of the features are saved in xgb_feature_names.pkl during training and used again during prediction to keep the order of features the same. This setup makes the

system easier to manage, stops features from being mismatched between training and prediction, and makes the whole system more dependable and simpler to update.[14]

3.11 URL to Text Transformation for SecureBERT

The function url_to_text(url) converts a URL into a textual format that is appropriate for the SecureBERT transformer model.
In contrast to XGBoost,
which relies on manually created numerical features, SecureBERT works with text tokens a s input. As a result, the original URL must
be cleaned and turned into meaningful tokens before being sent to the tokenizer.

The transformation process begins by converting the URL to lowercase and eliminating ext ra spaces.
Common protocol prefixes like http:// and https://, as well as the www. prefix,
are removed. This helps the model concentrate on the important parts of the URL, such as the domain, path, query terms, and any suspicious keywords.

The function also uses Unicode normalization through
unicodedata.normalize('NFKD', url).
This step changes special or accented characters into a standard ASCII format. This is helpful since URLs may include encoded or unusual characters that
could interfere with tokenization. After normalization, the URL is split using non-alphanumeric characters as separators. Any empty or very short tokens are removed, and only the first 50 tokens are kept.

Once this preprocessing is complete, the resulting text is sent to
the SecureBERT tokenizer via predict_transformer(url).
The tokenizer prepares the input by applying padding, truncation, and
a maximum length of 128 tokens before sending it to
the transformer model. The model then produces class probabilities, and
the phishing probability is extracted from these outputs.

SecureBERT is important for this project because it is a domain-specific language model designed for cybersecurity text.
The SecureBERT paper explains that understanding cybersecurity-

related text requires specialized language knowledge, especially in areas like cyber threat i ntelligence and automated security analysis.

3.12 Zero-Day Detection Examples

The application features a heuristic-based Zero-Day detection system implemented through the function estimate_zeroday_score(url). This function does not rely solely on trained machine learning models. Instead, it employs manually established rules to identify suspicious patterns that might appear in new or previously unknown phishing URLs. The function produces a scor e ranging from 0 to 100, along with a list of alerts that explain why the URL is considered suspicious.

The Zero-Day detection system utilizes 15 key indicators. These include the presence of malicious keywords, impersonation of well-known brands, brand names appearing in subdomains, suspicious top-level domains (TLDs), domains built on IP addresses, non-standard ports, encoded characters, the use of the @ symbol, URL shortening services, multiple phishing-related keywords, excessive hyphens, deep subdomains, substitution of letters with digits, and the use of HTTP instead of HTTPS.

Example 1: Brand Impersonation
https://paypal-security-login.example.com
This type of URL may be flagged as suspicious because it includes a trusted brand name, such as 'paypal,' combined with words related to security and login. In the Zero-Day detection system, both brand impersonation and the use of phishing keywords contribute to an increased risk score.

Example 2: Suspicious TLD
http://account-verification-example.xyz
This URL may be considered suspicious due to the inclusion of terms like 'account' and 'verification,' paired with a TLD such as '.xyz,' which is flagged as potentially dangerous. The system maintains a list of TLDs that are considered suspicious and adds to the risk score when such domains are detected.

Example 3: IP Address Instead of a Domain
http://192.168.1.10/login
Using an IP address instead of a regular domain name can be a red flag in phishing detection,

as attackers may use direct IP links to bypass recognizable domain names.
The Zero-Day engine has a rule to detect domains based on
IP addresses and increases the risk score accordingly.

Example 4: URL Shortener
https://bit.ly/example-login
URL shorteners conceal the actual destination of a link, making them
a concern in phishing attempts.
The code includes a list of known URL shorteners such as bit.ly, tinyurl.com, t.co,
among others. If a URL is registered with one of these services, the Zero-
Day score is raised.

Example 5: Digit Substitution
https://g00gle-login.example.com
This example demonstrates the use of digit substitution,
where certain letters are replaced with similar-looking numbers.
The system has a rule to detect patterns such as 'amaz0n,' 'g00gle,' or 'paypa1,'
and increases the risk score when such patterns are found in a domain.

In the final decision-making process, if the Zero-Day score reaches 70 or higher,
the application classifies the URL as malicious, even if
the machine learning models suggest a lower risk.
This rule helps in more effectively identifying highly suspicious URLs that have not
been seen before.

3.12 API Response Example:

The application includes an API endpoint at
/api/predict that accepts a URL and provides a JSON response with the final prediction and
the scores from various detection engines.
The Flask route handles a POST request, reads the provided URL, checks it against
a whitelist, computes scores using XGBoost, SecureBERT, Zero-Day, and
an optional HTML score, and then sends the results back in a JSON format using jsonify.

Flask serves as the web framework for the application.
According to its documentation, Flask is
a lightweight WSGI web framework that enables developers to create web applications. Its
routing system allows the definition of API endpoints through decorators like @app.route.

Example API request:

{
"url": "https://example-login-test.com",
"analyze_html": true
}

Example API response:

{
"Prediction": "malware",
"Confidence": 58.16,
"XGB_score": 99.84,
"SecureBERT_score": 99.95,
"Zeroday_score": 10,
"HTML_score": 0,
"reasons": [
"Brand impersonation detected",
"Multiple phishing keywords detected"
],
"source": "AI Ensemble (XGBoost + SecureBERT + Zero-Day + HTML)"
}

The response includes the final classification result, a global confidence score, and individual scores for each subsystem: xgb_score, transformer_score, zeroday_score, and html_score. These values are shown in the web interface after the user submits a URL.

The front-end sends the URL to the /api/predict endpoint using the JavaScript fetch() function.
Once the JSON response is received, it presents the final verdict along with the scores for XGBoost, SecureBERT, Zero-Day, and HTML analysis.

The application also offers a batch processing endpoint at /api/predict/batch, enabling the analysis of multiple URLs one after another. The
HTML interface includes a section for batch analysis, allowing the user to paste several URLs, submit them, and view the results in a table format.

3.12 List of XGBoost Features

The XGBoost model employs manually extracted features from URLs.
These features are generated using the shared **extract_features(url)** function implemented in the dedicated **features.py** file. This file centralizes the feature extraction logic and is imported by both **train_xgboost.py** during the training phase and **app.py** during the prediction phase. The feature names are stored during training to guarantee that the same 36 URL features, with the same order, are used during real-time prediction.

Structural URL features

Feature	Description
url_length	Total length of the URL.
dot_count	Number of dots in the URL.
hyphen_count	Number of hyphens in the URL.
slash_count	Number of slashes in the URL.
at_count	Number of @ symbols in the URL.
question_count	Number of question marks in the URL.
equal_count	Number of equal signs in the URL.
path_length	Length of the URL path.
param_count	Number of parameters based on & characters.

Domain-based features

Feature	**Description**
domain_length	Length of the extracted domain name.
tld_length	Length of the top-level domain suffix.
subdomain_count	Number of subdomain parts.
too_many_subdomains	Binary feature equal to 1 if the URL contains many subdomains.
digit_count_in_domain	Number of digits inside the domain.
digit_ratio	Ratio of digits in the domain.
letter_ratio	Ratio of letters in the domain.
domain_entropy	Shannon entropy of the domain name.
special_in_domain	Number of special characters inside the domain.

Security and obfuscation features:

Feature	**Description**
is_https	Indicates whether the URL uses HTTPS.
is_ip	Indicates whether the host/domain is an IP address.

is_shortened	Indicates whether the URL uses a known shortener.
has_port	Indicates whether a non-standard port appears in the URL.
has_double_slash	Detects double slashes after the protocol area.
has_encoded_chars	Detects encoded characters such as %2f.
suspicious_tld	Indicates whether the TLD belongs to the suspicious TLD list.

Phishing keyword features:

Feature	**Description**
has_login	Detects the word login in the domain.
has_verify	Detects the word verify in the domain.
has_update	Detects the word update in the domain.
has_secure	Detects the word secure in the domain.
has_account	Detects the word account in the domain.
has_admin	Detects the word admin in the domain.
has_signin	Detects the word signin in the domain.
has_ebayisapi	Detects the phishing-related term ebayisapi.
has_bank	Detects the word bank in the domain.

Brand impersonation features:

Feature	**Description**
brand_in_subdomain	Indicates whether a known brand appears in the subdomain while the real domain is different.
brand_in_domain_not_tld	Indicates whether a known brand appears inside the domain but does not exactly match the official brand domain.

These features are appropriate for XGBoost since the model performs effectively with structured numerical and categorical data. The XGBoost paper describes it as a scalable tree boosting framework designed for supervised learning tasks and able to manage intricate interactions between features.

3.13 Conclusion

The methodology put forth integrates synergistic techniques, underpinned by a rigorous feature extraction process that transforms raw URLs and webpage content into high-dimensional data. Within this framework, XGBoost effectively manages structured features of URLs, SecureBERT conducts an analysis of semantic tokens, Zero-Day heuristics identify anomalous patterns, and the examination of HTML contributes supplementary page-level evidence.

Chapter 4: Implementation and evaluation

4 .1 Introduction

This chapter is devoted to the technical execution and empirical validation of the hybrid phishing detection framework developed as an integral component of this academic project. Subsequent to the formulation of the conceptual architecture, we elucidate the extensive hardware and software environments that facilitated the developmental lifecycle of the system. We provide a detailed account of the algorithmic implementation of our multi-engine core framework, which adaptively integrates deterministic heuristics, statistical machine learning, and deep learning paradigms. Lastly, we delineate the evaluation protocol, the formulation of our large-scale dataset, and an exhaustive analysis of the empirical performance outcomes achieved.

4.2 Development Environment and Tools

The stages of experimenting, preparing data, training the model, and deploying the final application needed to bring together different technology resources, including both specialized hardware and software parts.

4.2.1 Hardware Resources

The local setup, early troubleshooting, and live running of the user interface were carried out on a standard workstation setup.

- Processor: x86_64 architecture with optimized multithreading capabilities
- Random Access Memory (RAM): 8 GB, ensuring smooth concurrent execution of the local server and lexical feature pipelines.

- Storage: 256 GB Solid State Drive (SSD), providing high-speed read/write access to manipulate data structures efficiently.

However, because fine-tuning the deep learning model (SecureBERT) and training large-scale gradient-boosted decision trees (XGBoost) required a lot of computational power, we used Google Colab's cloud resources. This setup offered faster hardware, including an NVIDIA T4 GPU, which was essential for handling complex calculations more efficiently and improving how quickly the models could learn and improve.

4.2.2 Software Environment

The software used to build the solution follows current standards for data engineering and web architecture.

- Operating System: Windows and Linux environments (via the cloud infrastructure instances).
- Programming Languages:
    - Python: The principal programming language has been chosen due to its extensive ecosystem encompassing libraries relevant to data science, cybersecurity, and artificial intelligence.
    - JavaScript, HTML5, and CSS3: These technologies are synergistically employed to create an interactive, asynchronous, and responsive web interface, characterized by an aesthetically pleasing dark theme.
- Execution Environments: Jupyter Notebooks (ipynb) are utilized on Google Colab for primary training purposes, in conjunction with local virtual environments designated for backend deployment.

4.2.3 Python Libraries and Frameworks

The backend application and the core smart systems use a number of special libraries, as listed below

- **Flask:** A light Python framework used to build the backend REST API, manage async client requests, provide evaluation endpoints, and serve web templates.
- **XGBoost**: A fast gradient boosting library that uses decision trees, forming the main part of our statistical classification system for syntax features.
- **PyTorch (torch):** A framework for handling tensor computations, used for managing GPU and CPU resources, loading structural weights, and performing efficient inference for the SecureBERT transformer model.
- **Transformers:** Developed by Hugging Face, this library encompasses the AutoTokenizer and AutoModelForSequenceClassification classes, which are essential for the initialization and execution of the transformer pipeline.
- **Scikit-Learn:** Employed for fundamental machine learning tasks, encompassing data partitioning (train_test_split) and the computation of validation metrics.
- Pandas Facilitates: the proficient manipulation, cleansing, filtration, and indexing of large-scale datasets through the utilization of structured DataFrame objects.

- **BeautifulSoup4:** Furnishes Document Object Model (DOM) parsing utilities to examine crawled HTML content and identify inline structural discrepancies during comprehensive content analysis.
- **Tldextract:** Guarantees precise, robust, and independent parsing of subdomains, registered domain names, and top-level domains (TLDs) from intricate URL structures.
- **Joblib:** Manages the serialization and deserialization processes of the trained models and feature names, thereby enabling prompt disk loading upon the initiation of the backend server.

4.3 Hybrid System Architecture

The system under consideration is constructed upon a cohesive Multi-Engine Framework characterized by interconnected components. Instead of depending solely on a singular, autonomous classifier, the system integrates three separate analytical engines to enhance the comprehensiveness of threat detection.

4.3.1 Ensemble Fusion Weighting

The final score given to a checked website is calculated by combining the probabilities from three different models, each with its own weight. These weights were carefully set to get the best results while reducing the chance of incorrect alerts:
- SecureBERT (Deep Learning Engine): WsecureBERT= 0.45
- XGBoost (Machine Learning Engine) :WXGboost = 0.35
- Zero-Day (Heuristic Engine): Wzero Day= 0.20

The base ensemble score is calculated using this formula:

Score Base = (W SecureBERT × P SecureBERT) + (W XGBoost × P XGBoost) + (W ZeroDay × P ZeroDay)

4.3.2 Dynamic HTML content Module Integration

When the user turns on the deep HTML content analysis feature through the web interface, the backend downloads the DOM source code of the target webpage in the background. A fourth score is then created. To include this real-time feedback, the base ensemble weights are reduced to 71%, and the HTML module is combined with a weight of W_HTML = 0.29. This changes the final decision equation.

Score Final = (0.71 * **Score** Base) + (0.29 * **Score** HTML)

4.3.3 Decision Logic and Short-Circuit Rules

To ensure the system reacts quickly and protects against focused attacks, the final decision isn't always made by simply combining all results. The system uses important rules that take priority to address major threats right away:

1. Whitelist Mechanism: If the hostname is on a trusted list (like google.com or github.com) and there's no suspicious path changes, the URL is ignored and marked as Benign with a confidence score of 100%.

2. Zero-Day Heuristic Priority: If the system finds clear signs of a serious threat and the individual score is 70% or higher, the URL is immediately labeled Malicious, and other scoring methods are ignored.

3. SecureBERT Security Override: Since the transformer model is very accurate in understanding meaning, if SecureBERT predicts a phishing chance of 85% or more, the threat is quickly confirmed and classified.

4. Standard Decision Boundary: If none of the quick rules apply, the URL is considered a threat only if the final score is 0.50 or higher; otherwise, it's labeled as Benign.

4.4 Core Component Implementation

4.4.1. Feature Extraction Module (36 Features)

To supply the XGBoost statistical classifier, we created a high-throughput syntax parser that extracts a vector of exactly 36 numerical, statistical, and boolean features from the raw URL string.
These features are organized across four main categories:

Global Structural Features (11 features): These include the total length of the URL (url_length), counts of specific characters such as periods (dot_count), hyphens (hyphen_count), slashes (slash_count), at-symbols (at_count), question marks (question_count), and equal signs (equal_count).
It also includes the length of the URI path (path_length), the number of parameters (param_count), and flags for non-standard ports (has_port) or the presence of out-of-place double slashes (has_double_slash).

Domain Complexity Metrics (7 features): These evaluate the length of the registered domain (domain_length), the length of the top-level domain (tld_length), ratios of digits and letters (digit_ratio, letter_ratio), total digit counts in the domain (digit_count_in_domain), presence of special characters (special_in_domain), and a flag indicating whether the extension matches known high-risk TLDs (suspicious_tld), such as .
tk, .ml, .ru, .xyz, .top, and .site.

Suspicious Lexical Keywords (9 features): These are binary indicators that detect the presence of keywords commonly used by threat actors to manipulate user trust, such as login, verify, update, secure, account, admin, signin, ebayisapi, and bank.

Obfuscation and Spoofing Indicators (9 features): These include the Shannon entropy of the domain (domain_entropy) to identify domains generated by algorithms or random processes (DGA).
They also monitor the use of URL encoding (has_encoded_chars), shortening services (is_shortened), subdomain depth (subdomain_count), and an alert for excessive subdomains (too_many_subdomains activated if ≥ 3). Additionally, they detect brand impersonation anomalies (brand_in_subdomain, brand_in_domain_not_tld), which identify whether a trusted trademark (such as paypal, apple, microsoft, amazon, facebook, or google) appears outside its legitimate domain zone.

The Shannon entropy of the domain string is calculated using the following equation, where p represents the relative frequency of each unique character within the extracted domain string:

$$\boldsymbol{H(X)} = -\sum_{k=1}^{n} \mathrm{p_i} \log_2(\mathrm{p_i})$$

4.4.2 Deep Learning Module (SecureBERT Transformer)

The second component of our system is based on SecureBERT, a variant of the language model that has been modified from the RoBERTa architecture. It has been thoroughly pre-trained on a wide range of cybersecurity-related texts, which enables it to effectively detect and understand harmful textual indicators through its strong grasp of meaning. a custom function called url_to_text(). This preprocessing step removes protocol elements such as http:// or https://, eliminates generic www. tokens, converts all text to lowercase, and Before entering the neural network, raw URLs undergo processing through replaces special characters, query parameters, and delimiters with spaces. The cleaned text is then transformed into numerical tensors by the native tokenizer, with a maximum sequence length of 128 tokens. These tensors are fed into the transformer network, which generates a prediction based on the context of the URL. The fine-tuned model uses ehsanaghaei/SecureBERT as its base, trained on 160,000 balanced URLs (80k phishing, 80k legitimate) for 3 epochs with AdamW optimizer (lr=2e-5). Training was performed on a Google Colab T4 GPU with a batch size of 64 and maximum token length of 128

4.4.3Zero-Day Heuristic Engine

This component functions as an immediate, rule-
based layer that performs 15 distinct heuristic checks (signals) based on domain safety expertise.
Each rule violation contributes to a cumulative penalty score, which is capped at 100.
The presence of explicit cyber-attack-related terms (such as phishing, malware, virus, or credential-hijack) in the URL results in an instant penalty of 45.

Identifying clear instances of brand impersonation within subdomain components leads to a major penalty of 45 or 40.

Detecting lookalike digit substitutions intended to deceive users, such as amaz0n or g00gle, adds a penalty of 30.

The inclusion of the @ character, which
is sometimes used to visually truncate addresses in specific browsers, imposes a penalty of 40.

The simultaneous presence of three
or more standard credential harvesting keywords results in a severity penalty of 35.

4.4.4 Web Application and User Interface

The application is implemented using a decoupled, asynchronous client
server architecture:

Backend (Flask REST API): Hosts optimized endpoints such as /api/predict for

real-time individual URL inspection and
/api/predict/batch for validating multiple URLs simultaneously (up to
500 URLs per request).
Each prediction is recorded alongside an exact timestamp and final verdict in
a persistent file called collected_data.csv.

Frontend (User Interface): Built with a modern dark theme interface.
It provides tab
based navigation, allowing users to switch between fast individual scanning, bulk file or

text uploads (Batch Analysis),
and dynamic result displays in filterable tables. A historical review pane is also included

to track previously scanned items.

4.5 Evaluation, Metrics, and Results

4.5.1 Final Dataset Composition

To assess the system's effectiveness, a large-
scale dataset was created that accurately represents current global web traffic.
This dataset was formed by combining several reliable public sources:

- Historical records of both legitimate and malicious URLs.

- Daily updated phishing feeds from PhishTank, verified and actively maintained.

- Malware distribution sources listed in URLhaus.

- High-authority legitimate websites from the Tranco Top 1M whitelist.

Carefully selected long and complex URLs from official technical documentation and support pages, specifically included to help train models to avoid raising false alarms on harmless long links.

After applying data normalization (assigning 0 for safe URLs and 1 for phishing URLs), removing duplicates, and eliminating missing data, the final dataset consists of 586,575 unique URLs, categorized as follows:
- Legitimate (Safe) URLs: 397,305 entries.

- Phishing (Malicious) URLs: 189,270 entries.

- PhiUSIIL Phishing URL Dataset
The main dataset is from Kaggle (harisudhan411/phishing-and-legitimate-urls) [10]. It contains 235,000 URLs collected between 2020 and 2023, covering a wide variety of modern phishing patterns. The legitimate URLs come from the Alexa Top 1M ranking and verified academic sources.

Characteristic Value

Total number of URLs : 235,000

Phishing URLs : 117,500 (50%)

Legitimate URLs : 117,500 (50%)

Average URL length : 72 characters

Maximum URL length : 2,048 characters

TLDs covered > 150
Collection period
2020–2023

4.5.2 XGBoost Experimental Performance Results

The XGBoost classifier was tested using a stratified approach, where 80% of the data was used for training and 20% was set aside for testing new, unseen instances. When evaluated against this independent test set (comprising 117,315 URLs), the model achieved a high Global Accuracy of 98.79%.

Confusion Matrix

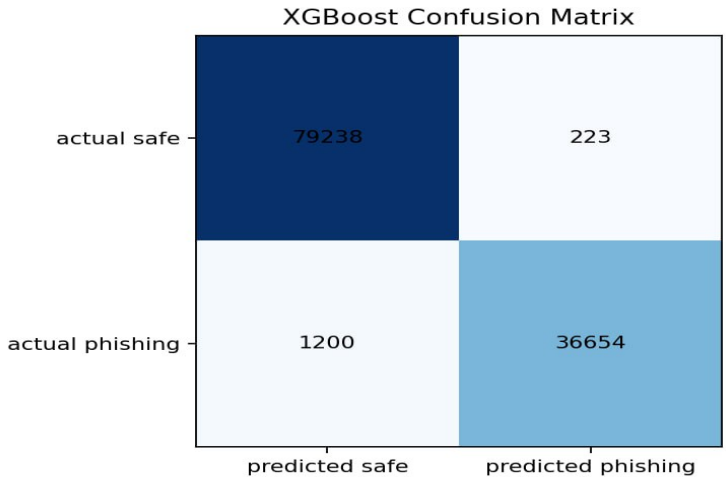


Figure 4.1: XGBoost Matrix

Detailed Classification Report

From this prediction distribution, the main performance measures for each target class were calculated and presented in the table below.

Target Evaluation Class	PRECISION	RECALL	F1-SCORE	SUPPORT (URLs)
Safe (Class 0)	0.99	1.00	0.99	79,461
Phishing (class 1)	0.99	0.97	0.98	37,854
Mecro Average	**0.99**	**0.98**	**0.99**	**117,315**

The mathematical formulas validating these indicators are defined as traditional mathematical fractions below:

- $\textbf{Precision} = \frac{TP}{TP+FP}$
- $\textbf{Recall} = \frac{TP}{TP+FN}$
- $\mathrm{F1-Score}x = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$

Analytical Results Discussion:

A detailed examination ofthese performance metrics reveals an operational balance that is appropriate for real-time implementation. The system achieves a
99% precision rate in identifying phishing links,
which means it generates a minimal number of false positives—only 223 instances out of 117,315 total tests. This low rate helps prevent alert fatigue among users. At the same time, the
97% recall rate demonstrates the system's effectiveness in detecting nearly all active threats, including those with complicated syntax structures.

4.5.3 SecureBERT Experimental Performance Results

To fully evaluate how well the fine-tuned deep learning module works for classification, a confusion matrix was made especially for the SecureBERT URL Classifier.This matrix gives a clear, visual picture of how the model's predictions match up with the real correct labels based on 16,000 different validation samples.

Confusion matrix.

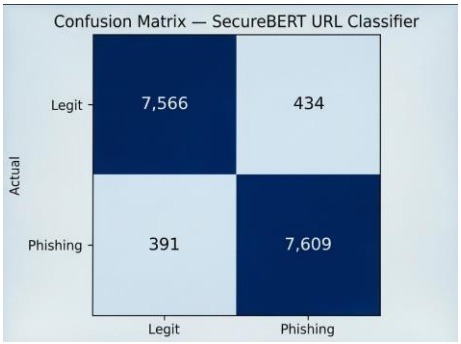


Figure 4.2: SecureBERT Matrix

The analytical results show that the SecureBERT model performs very well at telling the difference between real and harmful URLs. It correctly identified 7,609 phishing sites as dangerous (True Positives) and 7,566 safe websites as legitimate (True Negatives). There were some mistakes, though—only 434 false positives (safe websites wrongly labeled as dangerous) and 391 false negatives (harmful sites missed as safe).

These results show that SecureBERT maintains a good balance, keeping both false alarms and missed threats low.
This level of performance shows the model effectively uses language and context to understand meaning. Because of this, SecureBERT is very good at finding subtle and new cybersecurity threats that traditional methods based only on syntax often miss.

4.6 Conclusion

This final chapter,the third in sequence, provides a technical and experimental validation of the architectural decisions made for our hybrid detection system.
By combining the statistical power of XGBoost across 36 distinct features,
the semantic contextual understanding provided by the SecureBERT transformer, and the deterministic efficiency of the Zero-Day heuristic engine, the system offers a strong protective layer. The empirical findings, which include an overall testing accuracy of 98.79% on a large, verified dataset, show that integrating modern natural language processing techniques with gradient boosted classifiers creates an effective strategy for countering advanced social engineering attacks.

General Conclusion

This project created an intelligent web app that helps find harmful URLs. The app looks at the URL directly, even if it's shared through emails, text messages, or social media. The system uses a mix of XGBoost, SecureBERT, a Zero-Day heuristic engine, and optional HTML analysis. This combination makes the app more effective than using just one method.

The XGBoost model checks features like URL length, use of punctuation and numbers, HTTPS usage, entropy, subdomains, suspicious domain endings, and fake brand names. SecureBERT converts the URL into a text form and gives a chance of it being a phishing link. The Zero-Day engine uses clear rules to spot unusual patterns in new URLs. HTML analysis adds extra clues about the webpage when turned on. All these parts are put together in a Flask app that lets users check one URL at a time, process multiple URLs, and view a history of results.

This project shows how combining different cybersecurity tools can make detection more accurate.
It also shows that being transparent is important: the app doesn't just say if a URL is dangerous, but also explains why and gives scores for each part.

Future Work

- Add the ability to automatically pull URLs from full emails or texts.

- Create a browser extension for checking URLs in real time.

- Connect the app to up-to-date sources of threat information.

- Improve the SecureBERT dataset and regularly retrain the model.

- Add a dashboard with stats, charts, and options to export data.

- Make the system available as a public API with login and limits on how many requests can be made.

Bibliography

[1] NIST CSRC Glossary - Phishing. https://csrc.nist.gov/glossary/term/phishing

[2] NIST Small Business Cybersecurity - Phishing guidance.

https://www.nist.gov/itl/smallbusinesscyber/guidance topic/phishing

[3] Microsoft Support - Protect yourself from phishing.

https://support.microsoft.com/en- us/security/protect-yourself-from phishing

[4] OWASP - Web security awareness and OWASP Top Ten resources. https:

//owasp.org/www-project-top-ten/

[5] T. Chen and C. Guestrin, XGBoost: A Scalable Tree Boosting System, KDD 2016

https://dl.acm.org/doi/10.1145/2939672.2939785

[6] XGBoost Documentation - parameters and binary:logistic objective

https://xgboost.readthedocs.io/en/stable/parameter.html

[7] J. Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language

Understanding, 2019. https://arxiv.org/abs/1810.04805

[8] Hugging Face Transformers - BERT and sequence classification documentation.

https://huggingface.co/docs/transformers/en/tasks/sequence_classification

[9] Flask Documentation - routing, JSON APIs and web application framework

https://flask.palletsprojects.com/en/stable/quickstart/

[10] Scikit-learn Documentation - train/test split, classification report and confusion matrix

https://scikit learn.org/stable/

[11] tldextract Documentation - subdomain, domain and suffix extraction.

https://pypi.org/project/tldextract/

[12] Python urllib.parse Documentation - parsing URLs into components

https://docs.python.org/3/library/urllib.parse.html

[13] Beautiful Soup Documentation - HTML parsing with Python.

https://beautiful-soup- 4.readthedocs.io/en/latest/

[14] Project source code supplied for this report: app.py, train_xgboost.py and index.html

Local project files

[15] IBM Think - What is phishing? https://www.ibm.com/think/topics/phishing

[16] UK NCSC - Phishing attacks: defending your organisation.

https://www.ncsc.gov.uk/guidance/phishing

[17] Y. Wang et al., "A Large-Scale Pretrained Deep Model for Phishing URL Detection," . ICASSP, 2023. [Online]. Available: . .
.https://doi.org/10.1109/ICASSP49357.2023.10095719

[18] P. Maneriker et al., "URLTran: Improving Phishing URL Detection Using . .Transformers," MILCOM, 2021. [Online]. Available: https://doi.org/10.1109/MILCOM52596.2021.9653028

[19] E. Aghaei et al., "SecureBERT: A Domain-Specific Language Model for . Cybersecurity," arXiv:2204.02685, 2022. [Online]. Available: https://arxiv.org/abs/2204.02685

[20] N. Q. Do et al., "An Integrated Model Based on Deep Learning Classifiers and Pre- trained Transformer for Phishing URL Detection," Future Generation Computer Systems, 2024. [Online]. Available: https://doi.org/10.1016/j.future.2024.06.031

[21] N. Tariq et al., "Hybrid Semantic-Structural Learning for URL Multi-Classification Using DistilBERT and Feature Engineering," AICCSA, 2025. [Online]. Available: https://doi.org/10.1109/AICCSA66935.2025.11315417

[22] T. T. Huynh et al., "A Semantic-Aware Approach to Phishing URL Detection," Engineering, Technology & Applied Science Research, 2025. [Online]. Available: https://doi.org/10.48084/etasr.13187

References of Datasets

PhishTank, “PhishTank Developer Information,” OpenDNS/Cisco, 2026. [Online]. Available: https://phishtank.org/developer_info.php

abuse.ch, “URLhaus: Malware URL Exchange,” abuse.ch / Spamhaus, 2026. [Online]. . Available: https://urlhaus.abuse.ch/

V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, “Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation,” in Proceedings of NDSS 2019, 2019. [Online]. Available: https://tranco-list.eu/

J. K. S. Kaitholikkal and B. Arthi, “Phishing URL dataset,” Mendeley Data, Version 1, 2024. doi: 10.17632/vfszbj9b36.1. [Online]. Available: https://data.mendeley.com/datasets/vfszbj9b36/1

https://www.kaggle.com/datasets/ndarvind/phiusiil-phishing-url-dataset?utm_source=chatgpt.com