Computers and Electrical Engineering 132 (2026) 111001

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Real-time phishing uniform resource locator detection based on hybrid embedding transformer and retraining-free inferencing

Dam Minh Linh [a,*], Huynh Trong Thua [a], Tran Cong Hung [b]

[a] Information Security Technology Laboratory and Faculty of Information Technology, Posts and Telecommunications Institute of Technology (PTIT), Ho Chi Minh, Vietnam
[b] School of Computer science & Engineering, The Saigon International University (SIU), Ho Chi Minh, Vietnam

ARTICLE INFO

Keywords:
Artificial intelligence
Cybersecurity
Hybrid embedding Transformer
Levenshtein distance
Retraining-free inferencing
URL data classification

ABSTRACT

Phishing attacks that evade traditional detection mechanisms by exploiting deceptive uniform resource locators (URLs) remain a significant cybersecurity threat. This study proposes an adaptive phishing URL detection framework that integrates Levenshtein distance-based string similarity, a hybrid embedding transformer (HET) encoder-based server-side verification mechanism, and a dynamically updated local blacklist. First, a rapid local lookup is executed to identify known phishing URLs. If the input URL is absent from the blacklist, the Levenshtein distance algorithm detects subtle character-level variations, identifying typosquatting and obfuscation effectively. For ambiguous cases, the HET-based module uses a lightweight post-hoc inference method that classifies URL embeddings via k-nearest neighbor voting based on Euclidean similarity in the latent space, thereby avoiding retraining and enabling real-time adaptation to emerging phishing threats. Confirmed phishing URLs are added iteratively to the local repository to improve detection continuously, enhancing future classification accuracy. Experimental evaluation on a large-scale dataset comprising 235,795 URLs revealed that the proposed method outperforms state-of-the-art approaches, achieving a detection accuracy of 99.8 %, with a false-positive rate of 0.441 % and false-negative rate of 0.0617 %. Additionally, real-time validation using a Chrome browser extension confirmed rapid processing, with an average processing time of 4.43–6.84 ms per URL on a dataset comprising 5,000 URLs. These results highlight the efficiency of the proposed framework in real-world cybersecurity contexts, enabling high detection accuracy, fast response times, and adaptability to evolving phishing strategies, and underscore the importance of proactive threat intelligence and real-time phishing mitigation in developing scalable, high-performance security infrastructures.

1. Introduction

Despite growing awareness of cybersecurity threats, non-technical attacks, particularly phishing, remain prevalent. Phishing leverages social engineering techniques to exploit human vulnerabilities—considered as the weakest link in security systems—to facilitate identity theft. Cyber criminals use persuasive tactics to lure victims into clicking malicious uniform resource locators (URLs), potentially exposing credentials or compromising digital assets. Consequently, effective feedback during anti-phishing training is essential [1–4]. Phishing URLs are often generated using a multistage cycle that enables attackers to refine and launch their schemes

* Corresponding author.
E-mail address: linhdm@ptit.edu.vn (D.M. Linh).

https://doi.org/10.1016/j.compeleceng.2026.111001
Received 29 August 2025; Received in revised form 27 November 2025; Accepted 26 January 2026
Available online 2 February 2026

systematically [5,6]. E-commerce platforms and online transaction environments are prime locations for such attacks [7], offering cyber criminals the opportunity to launch large-scale campaigns and harvest sensitive personal data. The risk of personal information leakage via online scams is increasing, and the two high-risk groups are particularly vulnerable to significant financial losses. Consequently, privacy and data security concerns have become pressing threats amid the rapid expansion of e-commerce and social networking platforms. Although automatic identification technologies, such as barcodes and radio frequency, enhance overall shopping experience, they introduce significant security vulnerabilities [8,9]. First, large-scale data breaches present major challenges for businesses and policymakers regarding the governance of personal data security [10–12]. Second, malware infections in consumer and enterprise environments, along with real-time threats arising from malicious web content, remain critical cybersecurity concerns [13,14]. Such malware infections include deceptive tactics that manipulate users to interact with fraudulent URLs, enabling unauthorized access, credential theft, and the illicit transfer of funds to external accounts [15,16]. The Phishing Activity Trends Report for Q4 2024 [17] provides a detailed account of phishing incidents during that period. As presented in Table 1, the number of reported cases increased from 932,923 in Q3 of 2024 to 989,123. The SaaS/Webmail sector was the most targeted industry, accounting for 23.3 % of all attacks, followed by social media platforms at 22.5 %, reflecting a decline from Q3 onwards. Further, the report highlights a surge in the average financial demand from Business Email Compromise attacks, reaching $128,980 in Q4, which was nearly double the amount in the previous quarter. These trends underscore the increasing sophistication of phishing attacks and the need for advanced mitigation strategies as well as public awareness.

Recently, alerts about fraudulent URLs were issued to customers of Cake by the digital banking services of the Bank (28 February 2024, 17:01). As depicted in Fig. 1, all other reported URLs were identified as phishing attempts.

Phishing continues to be a major cybersecurity threat affecting financial transactions, data privacy, and operational integrity of public institutions. Machine learning (ML)-based mitigation techniques, such as phishing URL detection systems and graph-based models, have been introduced to address the limitations of conventional detection methods. However, prevention relies not only on technology but also on user awareness and organizational culture. Therefore, tailored training programs are critical to foster resilience against phishing attacks in real-world environments [18–20]. Security awareness among users significantly impacts desktop security behavior [21,22]. Password-based authentication remains the most common method of user verification [23]; however, large-scale data breaches have raised concerns about password strength and length [24]. Two-factor authentication (2FA) has been proposed as an additional layer of security beyond passwords [25,26]. Although 2FA improves security, it increases authentication time. A case study on safeguarding against cyber threats [27] described the implementation of 2FA at a university to protect student email accounts. In this system, users authenticate their identity by entering a password utilizing mobile devices, thereby preventing unauthorized access even in the event of password compromise. However, recent research by Kaspersky Lab [28] suggests that 2FA may be vulnerable in certain scenarios. One-time password (OTP) bots represent a new form of sophisticated malware that bypasses 2FA by intercepting OTPs using advanced social engineering techniques. These attacks exploit human error rather than technological flaws and pose a significant risk to individual users and online platforms. Typically, attackers first acquire login credentials and then trigger an OTP. The OTP bot places a phone call using a scripted message that manipulates the user to reveal the code. Once disclosed, the attacker gains unauthorized access to the account.

Modern deep learning (DL) systems typically require retraining on the entire dataset when encountering new, previously unseen URLs or employ complex output layer parameters alongside continuous fine-tuning to optimize model weights. These processes are time-consuming and computationally expensive, rendering them unsuitable for real-time deployment environments where new phishing URLs emerge continuously.

To overcome this challenge, we propose a post-hoc inference framework that leverages the hybrid embedding transformer (HET) encoder combined with k-nearest neighbor (k-NN) voting based on Euclidean similarity within the latent embedding space. This approach facilitates rapid classification, without relying on retraining, by embedding incoming URLs and comparing them with a reference set of embeddings, thereby eliminating the need for repeated model updates or dependence on internal confidence scores.

1.1. Research contributions

Motivated by the limitations of existing methods, this study critically reviews recent phishing URL detection research to identify research gaps, clarify objectives, and propose a novel adaptive multi-tier detection framework. The proposed solution addresses emerging cybersecurity needs by improving real-time detection accuracy and robustness with respect to adversarial phishing techniques.

This study introduces an adaptive real-time framework that combines lightweight local filtering, Levenshtein matching, transformer-based embedding, and post-hoc inference using k-NN voting, enabling scalability, robustness, and high-speed classification.

Table 1
Monthly phishing activity statistics.

	October	November	December
Number of Unique Phishing Websites (Attacks) Detected	345,881	313,288	329,954
Unique Phishing Email Campaigns	28,327	27,668	33,899
Number of Brands Targeted by Phishing Campaigns	315	333	309

Fig. 1. Alerts sent to customers of VPBank Digital Banking for fraudulent URLs.

1. Design of a Novel Multi-Tier Detection Architecture: This study proposes a three-tier detection pipeline consisting of: (i) a locally maintained and dynamically updated URL blacklist for rapid exclusion of known threats at the browser extension layer; (ii) a client-side Levenshtein-distance-based similarity check to identify typosquatting and subtle manipulations effectively; and (iii) a server-side HET encoder model, which leverages advanced natural language processing (NLP) techniques, synthetic data augmentation, and dynamic attention masking to perform deep classification on ambiguous or novel URLs. A key technical innovation is the incorporation of a lightweight post-hoc inference mechanism in the HET, where candidate URL embeddings are tokenized and classified using k-NN voting in the latent space, bypassing complex parametric output layers and the need for retraining. The design communicates real-time phishing alerts to the client and uses a feedback channel for dynamic blacklist updates. This architecture ensures rapid, real-time classification and robust adaptability to sophisticated phishing strategies in dynamic operational environments.
2. Development of the HET: A new DL model is introduced by leveraging customized n-gram tokenization, synthetic augmentation using special tokens, and dynamic attention masking. These innovations enhance the transformer's capability to model structural URL patterns, improving robustness with respect to novel and obfuscated phishing attacks while maintaining low inference latency.
3. Integration of Adaptive Threshold-Based Classification: The HET model incorporates a dynamic thresholding mechanism that adjusts its classification decision boundary in response to evolving phishing patterns. This adaptive approach improves generalization and reduces false-positive rate (FPR) and false-negative rates (FNR) significantly compared to traditional fixed-threshold classifiers.
4. Real-World Deployment via Chrome Browser Extension: A lightweight browser extension is implemented for real-time user protection, which performs on-device Levenshtein filtering and forwards uncertain URLs to an HET model running on a remote server. This hybrid deployment exhibits a processing speed of 4.43–6.84 ms per URL, ensuring scalability and usability in operational environments.
5. Comprehensive Evaluation and Comparative Analysis: The proposed framework exhibits outstanding performance across a large-scale benchmark dataset containing 235,795 URLs (134,850 benign and 100,945 phishing), surpassing additional and DL baselines consistently, including XGBoost (XGB), Gradient Boosting (GB), and DNNs, in terms of multiple classification metrics, such as precision, recall, F1-score, and area under the curve (AUC). The integration of adaptive mechanisms, local filtering, and transformer-based representations enables high robustness, low false-alarm rates, and strong generalizability across varying threat scenarios and URL obfuscation strategies. To quantify the performance in practical environments, the HET model is evaluated on a 70:30 training–test dataset split using 235,795 URLs in aggregate (165,056 for training and 70,739 for testing). The following results are obtained:
   - Accuracy: 99.88 % (training) and 99.77 % (testing).
   - Precision: 99.80 % (training) and 99.65 % (testing).
   - Recall: 99.99 % (training) and 99.93 % (testing).
   - F1-score: 99.90 % (training) and 99.79 % (testing).
   - Loss: 0.00479 (training) and 0.01765 (testing).
   - AUC: 99.87 % (training) and 99.75 % (testing).
   - FPR: 0.441 %.
   - FNR: 0.0617 %.
   - Training time: 1040.82 s.

   Inference time: 108.91 s (for 70,739 test URLs).

   These results highlight the effectiveness of the framework for real-time phishing detection, offering both high classification accuracy and computational efficiency suitable for deployment in browser-server environments.

6. Integration of Post-hoc Inference based on Embedding Similarity and k-NN Voting: To address critical challenges, such as high hardware resource consumption, financial costs, and particularly the extensive retraining time required for updated datasets, recent phishing detection methods have predominantly relied on DL models employing strategies, such as full retraining or fine-tuning. However, these approaches introduce significant bottlenecks that impede real-time deployment. In contrast, the HET model overcomes these limitations by adopting a streamlined post-hoc inference strategy that classifies URL embeddings directly using a

k-NN voting mechanism based on Euclidean similarity in the latent embedding space. This method exhibits an inference latency ranging from 26 to 52 ms per URL, enabling rapid, detection of previously unseen phishing URLs that is not reliant on retraining, reducing operational latency significantly while enhancing adaptability. These capabilities are especially critical for deployment in dynamic environments, including browser-side and edge security systems, where timely responses to emergent threats are imperative.

The remainder of this paper is organized as follows. The current state of research on phishing URL detection is reviewed in Section 2; this includes topics such as browser-integrated mechanisms, blacklist and whitelist strategies, ML- and DL-based approaches, ensemble-based methods, and transformer architectures. Section 3 introduces the proposed multi-tier detection framework, including a browser-side blacklist, a Levenshtein similarity module, an HET model, and a lightweight post-hoc inference mechanism based on embedding similarity and k-NN voting. This novel post-hoc inference approach enables rapid, real-time detection of new phishing URLs without relying on retraining, improving adaptability and deployment efficiency. Data processing and training configurations are also discussed in this section. In Section 4 presents experimental results, including the evaluation of real-time phishing URL detection for browser extensions, assessment in terms of various data splits and classification metrics, comparisons of algorithms based on FPR, FNR, and average AUC, analysis using post-hoc k-NN inference with latent embedding similarity, and a discussion of the overall findings and implications. Finally, in Section 5 concludes the study by summarizing the key contributions, discussing practical implications, and outlining directions for future research.

2. Related works

2.1. Browser extension-based detection

Recent studies have explored browser extensions [29] to enhance user privacy and security effectively by enabling client-side protection independent of server cooperation. For instance, CookieBlock blocks tracking cookies to protect user privacy, demonstrating the potential of user-centric solutions in maintaining control over personal data [30]. Building upon this work, Rawoteea and Bekaroo [31] proposed RXSS Protect, a browser extension that leverages an ML model to detect and block reflected cross-site scripting payloads in real time, thereby strengthening security at the client side.

Similarly, Prasad et al. [32] developed a browser extension based on ML algorithms for real-time URL analysis to detect phishing websites. This extension also enables users to submit suspicious URLs, facilitating a collective defense mechanism within the online community. User feedback collected from 150 participants during testing helped them refine the detection model, achieving 97.5 % accuracy and an average response time of 4.5 s per URL [33]. Further, Jaiswal et al. [34] proposed a browser extension by combining bi-directional long short-term memory (Bi-LSTM) and convolutional neural network (CNN) architectures in a two-tier model for real-time detection and blocking of malicious URLs. They also integrated ad-blocking capability to reduce phishing risks via advertisements.

However, existing solutions, such as the aforementioned examples, rely largely on static training datasets and controlled experimental environments, which limit their adaptability to emerging attack vectors and large-scale real-world deployment. Moreover, many approaches lack mechanisms for continuous learning and prompt model updating based on evolving threat patterns.

To address the aforementioned limitations, this study reconstructs existing detection frameworks using more diverse and dynamic datasets and proposes novel enhancements, including incremental learning techniques and a user feedback-driven adaptive model. This approach aims to improve both detection accuracy and response efficiency in real-world browsing environments.

2.2. Blacklist- and whitelist-based techniques

Blacklist-based mechanisms are a long-standing foundation of phishing URL detection, with notable systems, such as AutoBLG [35, 36], automating blacklist updates based on filtering and expansion. Recent advances, however, have focused on integrating adaptive, client-side blacklists with ML-based analysis to enhance real-time protection and responsiveness against evolving phishing tactics. Notably, some frameworks dynamically manage multiple URL registries (blacklists, whitelists, and user-driven greylists) to improve adaptability and minimize false positives (FPs), thereby enabling a more rapid response to emerging threats [37].

Despite their widespread use, recent studies have revealed persistent weaknesses in real-world blacklist-based protection, particularly under evasion scenarios. For example, Oest et al. [38] deployed over 2,000 phishing websites systematically to test the effectiveness of major browser blacklists and anti-phishing entities under cloaking conditions (e.g., geolocation-, device-, and JS-based filtering). Their results revealed that such evasion techniques reduce blacklist detection rates by over 55 % on average, with even lesser detection on mobile browsers, such as Chrome and Safari. These results highlight the limitations of static blacklists and reinforce the need for adaptive, multilayered defenses in modern phishing detection systems.

The continuous emergence of deceptive domains has outpaced the capabilities of traditional blacklist- and whitelist-based systems, especially in identifying zero-day phishing attacks [39]. To address this issue, Azeez et al. [40] proposed a whitelist-based automated detection system. However, such whitelist-based approaches often struggle to detect novel phishing URLs not present in the list and require frequent updates to maintain adequate accuracy.

Although blacklist- and whitelist-based methods are lightweight and cost-efficient, they are inherently inflexible and suffer from delayed adaptation to emerging phishing techniques. Moreover, their dependence on static domain lists limits their effectiveness in recognizing unknown or zero-day attacks. Therefore, various ML, DL, and neural network models have been proposed to overcome

these constraints.

2.3. Traditional ML approaches

The advent of ML technologies has addressed certain limitations of traditional phishing detection methods, such as blacklist and whitelist techniques, particularly in terms of handling large-scale datasets and achieving faster processing times. Classical ML approaches, especially those employing shortened URL formats, have been demonstrated to be effective in detecting phishing URLs. In particular, Mahboubi et al. [41] classified threat-hunting strategies into ML-based, rule-based, and iterative analysis approaches. A hybrid ensemble feature selection technique was proposed as an ML-based method for phishing URL detection [42,43]. Despite achieving 97.8 % detection accuracy, the study did not analyze computational costs, especially regarding scalability on large datasets. This omission raises concerns regarding the scalability, precision, and practical deployment of the model in real-world anti-phishing systems. Another ML framework was proposed based on 42 distinct feature types [44], including blacklist, lexical, host-based, and content-based attributes. This framework was applied to support vector machine (SVM), Random Forest (RF), and Bayesian network models, yielding an accuracy of 98.95 % and a precision of 98.60%. However, its reliance on static features limits its ability to detect previously unseen phishing URLs. Additionally, the dataset, comprising approximately 5,000 URLs obtained from URLhaus and PhishTank, lacks diversity and reduces the generalizability of the model in real-world scenarios. Similarly, the method proposed by Orunsolu et al. [45] exhibits significant limitations due to its dependence on SVM and Naïve Bayes models. Moreover, the dataset used in this study lacks diversity; it consists of only 2,541 phishing and 2,500 legitimate URLs, which is insufficient for adequately representing the variability of phishing attacks. Thus, the practical applicability of the model is limited by its small dataset, basic ML algorithms, and the lack of real-world validation.

The integration of classical ML techniques with blacklist methods has been used to enhance phishing URL detection, especially for shortened or simple variants, by leveraging known patterns and automated classifiers. However, detecting URLs with long sequences, complex structures, or rare patterns remains challenging. In such cases, blacklists are often outdated and ML models struggle to extract deep semantic features. Phishing URLs featuring lengthy, dynamically generated sequences or employing obfuscation continue to yield missed detections and FPs in practice. Hence, novel approaches with higher generalizability capable of handling complex and lengthy URL sequences robustly and adapting to evolving phishing tactics beyond current model capabilities should be developed urgently.

Overall, although traditional ML techniques provide a solid foundation for phishing detection, their limitations in terms of scalability, feature representation, and adaptability necessitate exploration of more advanced and dynamic models.

2.4. Neural network-based models

Several DL models have been developed to improve phishing URL detection by learning long-context URL sequences effectively. Haq et al. [46] proposed a one-dimensional CNN that classifies URLs directly based on their raw character sequences, avoiding manual feature engineering. This model achieves 99.7 % accuracy on a large dataset of 400,000 URLs. However, although CNNs capture local spatial patterns well, their ability to detect dispersed and non-contiguous phishing features is limited. Moreover, the aforementioned study did not evaluate computational costs for real-time deployment, which is critical for practical applications.

To overcome these limitations, hybrid frameworks have emerged. Shahid et al. [47] combined a DL-based CNN with a cookie analysis engine for real-time detection and mitigation of web attacks, achieving $>$ 99 % accuracy. However, this approach still struggles with novel or obfuscated phishing URLs, such as “paypals.com,” indicating heavy reliance on quality training data.

More advanced architectures incorporate recurrent neural networks (RNNs) with attention mechanisms to better capture sequential dependencies. Asiri et al. [48] employed a BiLSTM network with attention and embedding techniques, achieving a 99 % F1 score across various phishing types. In this architecture, embeddings are used to convert URL tokens into dense vectors for semantic understanding, while attention highlights crucial URL segments. However, it relies heavily on graph neural networks, resulting in high computational and storage demands and limiting its scalability. Further, incomplete or sparse relational data also impairs its performance.

Complementary approaches have been proposed based on classical statistical models enhanced using feature extraction and domain adaptation. Paniagua et al. [49] combined logistic regression with Term Frequency-Inverse Document Frequency, achieving an accuracy of 96.5 % on the PILU-90K dataset. Rashid et al. [50] improved model generalization with unsupervised domain adaptation (UDA), increasing F1 scores by up to 0.2 via feature distribution alignments. These methods mitigate the rigidity of fixed feature sets and enhance adaptability to phishing domain shifts.

Although DL and hybrid models outperform traditional ML techniques, challenges remain in terms of computational efficiency, adaptability to evolving phishing tactics, and real-time feasibility. Most networks rely on CNNs and RNNs, with limited exploration of transformers—models known for capturing long-range dependencies via self-attention with promising phishing detection potential. In summary, despite significant advances, scalable, efficient, and adaptive frameworks integrating state-of-the-art DL architectures with NLP techniques are required to tackle the complex, evolving phishing landscape.

2.5. Blending-based ensemble methods

Varshney et al. [51] surveyed phishing prevention strategies and evaluated various detection methods, highlighting user-protection solutions. However, their study did not account for the rapid evolution of phishing tactics, lacked real-world validation, and overlooked shifts in user behavior and attack platforms, limiting the applicability of their findings.

Building on these insights, Alsariera et al. [52] introduced three meta-learning models based on the forest penalizing attributes algorithm, achieving a phishing URL detection accuracy of 96.26 %. In our previous study [5], we evaluated multiple ML and DL algorithms on 651,191 URLs, identifying an ensemble CNN-LSTM model as the best performer (accuracy of 97.6 %), demonstrating the benefits of combining neural architectures.

More recently, Omolara and Alawida [53] proposed DaE2, an ensemble framework merging Adaboost (AB), Bagging, Stacking, and Voting techniques. Although Bagging and AB exhibited F1 scores of 0.976 and 0.980, respectively, the study's novelty remains limited. Similarly, Aljofey et al. [54] developed a multichannel temporal convolutional network by fusing URL and HTML content, achieving 99.81 % accuracy on private data and over 98 % on benchmarks. However, this study did not assess deployment factors, e.g., training time or FPs and false negatives (FNs), which limited its practical implementation.

Yang et al. [55] achieved high accuracies on various datasets using a complex architecture combining autoencoders, transductive LSTMs, GANs, multilayer perceptrons (MLP), and reinforcement learning. Despite its effectiveness, its high computational overhead limits its real-time applicability. In contrast, our approach employs an HET module with post-hoc k-NN inference, which avoids retraining and enables accurate, low-latency phishing detection suitable for practical deployment.

In summary, combining ensemble methods effectively is known to leverage complementary model strengths to reduce misclassification rates. However, such methods often require increased training time and incur additional computational costs. The system proposed in this study addresses these limitations by offering a scalable, efficient hybrid framework that balances accuracy and practicability, enabling real-world phishing mitigation.

2.6. Traditional inferencing via retrained models

Traditional phishing detection models often rely on supervised DL architectures that require retraining or fine-tuning to maintain performance quality with evolving threats. Otieno et al. [56] fine-tuned bidirectional encoder representations from transformers (BERT) for phishing detection and achieved high accuracy without complex feature engineering. However, the authors noted that repeated retraining reduces its scalability in real-time environments. Similarly, Zhou et al. [57] introduced CSPPC-BiLSTM using spatial and channel attention, which exhibits improved robustness but lacks support for lightweight or adaptive inferencing. Even classical ML approaches are affected. Adap et al. [58] applied XGB and RF to URL-based detection with favorable results (96.51 % accuracy, $\kappa = 0.930$). However, the system showed degraded performance when it was not retrained on new data.

To improve generalization, Prabakaran et al. [59] proposed a hybrid VAE+DNN framework to extract latent URL features automatically. Although effective (97.45 % accuracy), it incurs inference delays (1.9 s per URL) and requires full retraining. Recent works have incorporated large language models (LLMs). For instance, Trad and Chehab [60] demonstrated that fine-tuned GPT-2 and Bloom outperform prompt-based usage (F1 = 97.29 % vs. 92.74 %) but require expensive adaptation. Likewise, Ali and Subba [61] introduced PhishURLDetect using LoRA to fine-tune RoBERTa and GPT-2 efficiently. Although they achieved an accuracy of 99.86 %, the method relies on task-specific updates.

The aforementioned discussion demonstrates the strengths of model retraining in enhancing accuracy and robustness, but also highlights their limitations in terms of latency, scalability, and adaptability. In particular, reliance on retraining restricts practical deployment in real-time or resource-constrained environments. To address these limitations, the present study proposes a retraining-free, post-hoc inference framework that balances detection performance with real-time and client-side applicability.

2.7. Transformer-based approaches

Vaswani et al. [62] introduced a transformer architecture based on an attention mechanism to address the limitations of sequential models, such as RNNs and LSTMs. In this mechanism, attention is used to compute a weighted sum of value vectors, enabling the model to dynamically focus on relevant positions in the sequence. The transformer processes inputs in parallel, resulting in substantial improvements in training efficiency. The model consists of an encoder–decoder architecture, in which the encoder uses multi-head self-attention and position-wise feed-forward networks (FFNs), whereas the decoder incorporates masked attention to preserve the autoregressive property of sequence generation.

Despite their widespread success in NLP and other domains, transformers remain underutilized in phishing URL detection. Although prior studies have leveraged DL methods for this task, few have tailored transformer-based designs to capture the structural patterns unique to URLs, such as n-gram dependencies or position-specific cues.

Although transformer-based architectures, such as BERT, have exhibited strong performance in phishing detection, they often remain unsuitable for real-time deployment due to reliance on full-page feature extraction, high inference latency, and limited support for lightweight URL-level analysis. These limitations hinder their utilization in browser-side environments and delay adaptation to emerging phishing tactics. To address these issues, our proposed HET framework builds upon the transformer design while simultaneously incorporating deployment-oriented adaptations, including a retraining-free, post-hoc inference mechanism and browser-side integration via extension-based deployment. Instead of relying on repeated retraining, HET enables real-time classification of unseen URLs using k-NN voting within the embedding space. Additionally, it supports dynamic local blacklist updates to improve detection speed and scalability.

Table 2 compares HET with recent transformer-based phishing URL detection models in terms of various architectural, functional, and deployment-related aspects. The data highlight HET's unique integration of retrain-free inference and client-side efficiency, features not addressed in prior transformer-based works.

As summarized, HET differs qualitatively from existing transformer-based phishing detection methods in terms of both architecture

and deployability, offering a scalable, inference-efficient approach tailored for dynamic phishing detection in real-world environments.

3. Proposed method

3.1 Multi-tier phishing detection framework

Phishing URL detection is a non-trivial and evolving challenge, as attackers continually develop more sophisticated evasion techniques. To address this, we propose a multi-tier detection framework that combines traditional blacklist filtering, similarity-based matching, and advanced ML techniques. Fig. 2 illustrates the stepwise architecture and processing flow of the proposed system.

As depicted in Fig. 2, the detection pipeline proceeds through three main tiers:

– Tier 1: Local Blacklist Check

The initial stage involves a fast lookup with respect to a local browser-managed blacklist of known phishing URLs. If the input URL matches an entry in this list, a phishing alert is immediately triggered, enabling low-latency protection for already identified threats.

– Tier 2: Similarity-based Filtering

If the URL is not present in the blacklist, a Levenshtein distance-based similarity check is used to compare the input URL with those in the blacklist. This step is designed to detect phishing attempts that leverage visually similar or typo-squatted URLs to evade direct matches. URLs flagged as highly similar are also considered to be phishing URLs.

– Tier 3: Server-side artificial intelligence (AI)-based Classification

For URLs that pass both previous tiers, a more sophisticated classification is performed on the server side. Here, the URL is tokenized using n-gram techniques and encoded using an HET model to capture its semantic and structural features. The resulting embeddings are compared with precomputed labeled neighbors using a k-NN approach. The final prediction is derived based on majority voting among the nearest neighbors, thereby leveraging learned representations effectively for robust detection.

This multi-tier strategy enables efficient filtering of clear phishing threats at the early stages, while reserving more resource-intensive AI-based analysis for ambiguous cases. This layered approach improves both the accuracy and computational efficiency of phishing detection in real-world scenarios.

Our proposed method effectively overcomes the limitations commonly encountered in two predominant model adaptation strategies in phishing detection—retraining and fine-tuning.

– Retraining involves training a model from scratch or from an initial checkpoint on entirely new or combined datasets, which is typically necessary when data distributions shift significantly. Despite its adaptability, retraining is computationally expensive and time-consuming, and thus impractical for real-time or large-scale deployments.
– Fine-tuning starts from a pretrained model and adjusts some or all weights based on new, domain-specific data, enabling faster adaptation. However, it remains vulnerable to catastrophic forgetting, requires careful hyperparameter tuning, and exhibits poor generalizability in highly dynamic or noisy environments.

In contrast, the proposed approach utilizes a hybrid browser–server architecture coupled with a transformer-based, embedding-rich model and a k-NN post-hoc inference mechanism. This framework circumvents the need for frequent retraining or extensive fine-tuning by leveraging efficient similarity searches over fixed embeddings, thereby achieving high detection accuracy with low latency. Consequently, it enables scalable, real-time phishing detection with high robustness with respect to data distribution shifts and adversarial manipulations, without incurring the high computational overhead or fragility associated with traditional adaptation methods.

3.2 Browser-integrated blacklist storage

Algorithm 1 integrates a predefined URL blacklist into the local storage of the browser, allowing efficient access to phishing site data without requiring continuous server communication. Initially, the system reads a blacklist.txt file containing malicious URLs sourced from Cisco Systems, Inc. [39]. The blacklist is converted into a JSON-formatted string to ensure compatibility with the storage mechanisms of JavaScript. Then, a JavaScript script is dynamically generated to store this blacklist using the localStorage.setItem() function. This script is saved as save_blacklist.js and can be executed while installing the browser extension. The overall computational complexity of the approach is $O(n)$, where n denotes the number of blacklisted URLs, which ensures scalability for real-time phishing detection in client-side applications.

The proposed algorithm ensures rapid URL lookup by optimizing local blacklist queries, thereby reducing latency and minimizing reliance on network connectivity by storing malicious URLs in local browser storage. This approach alleviates server load because the system requests blacklist updates only when necessary. Further, direct URL verification within the browser improves privacy by

Table 2
Comparison between HET and recent transformer-based phishing detection models.

Feature	Asiri et al. [63]	Yoon et al. [64]	Do et al. [65]	Majgave and Gavankar [66]	HET + Retrain-Free Post-hoc Inference (Proposed)
Input type	URL only	URL + HTML Document Object Model graph structure	URL only	URL only	URL only
Base model / Task	CNN + Transformer encoder / Phishing detection (10,000 URLs)	Graph Convolutional Networks (GCNs) + CNN + Transformer: Hybrid architecture for phishing detection (2012–2024 dataset, >80M nodes)	Transformer encoder + Deep Belief Network (DBN): Pretrained language representation for phishing URL detection (Ebbu2017, PhishCrawl, 420K-PD, 1M-PD)	Transformer + Deep Belief Network	Levenshtein filtering + Transformer encoder (n-gram tokenization, synthetic augmentation, dynamic attention masking, etc.) + post-hoc k-NN / Phishing detection (235,795 URLs)
Tokenization	Raw	Raw	Raw	One-hot encoding	n-gram + special token (“..”)
Synthetic Data Generation	No	No	No	No	Yes
Dynamic Attention Masking	No	No	No	No	Yes
Adaptive Thresholding	No	No	No	No	Yes
Real-Time Deployment (Browser-side API)	No	No	No	No	Yes
Avg. Processing Time (ms/URL)	-	-	-	-	4.43–6.84
Local Blacklist (Browser-side)	No	No	No	No	Yes (adaptive, local updates)
Post-hoc Inference (ms/URL)	No	No	No	No	Post-hoc k-NN voting 26–52
Task-Specific Performance	F1 = 99 %, Precision = 99 %, Recall = 99 %	Accuracy = 98.84 %, Precision = 96.77 %, Recall = 97.59 %, F1-score = 97.18%	Max Accuracy 99.71 %	Accuracy = 99.4%, Precision = 99.2%, Recall = 99.3 %, F1-score = 99.2 %	**Accuracy** = 99.8 %, **FPs** = 0.441 %, **FNs** = 0.0617 %; **AUC** = 99.87 % (training) and 99.75 % (testing); **Training time** = 1040.82 s per URL **Inference time** = 108.91 s (for 70,739 test URLs); **Recall** = 99.99 % (training)/ 99.93 % (testing); **F1-score** = 99.90 % (training)/ 99.79 % (testing).

Notes:
- All listed models are based on transformer architectures, unless otherwise indicated in the “Base model” row.
- Task-specific performance metrics (e.g., Accuracy, Precision, Recall, and F1-score) are taken directly from each source publication.
- “–” denotes that the respective publication did not report runtime or inference latency.
- The proposed HET model integrates multiple innovations not addressed in prior works, including:
(i) retraining-free inference via post-hoc k-NN voting in the embedding space,
(ii) dynamic attention masking and adaptive thresholding for improved token-level discrimination, and
(iii) browser-side, real-time deployment using a lightweight extension with adaptive local blacklist updates and Levenshtein-based filtering.

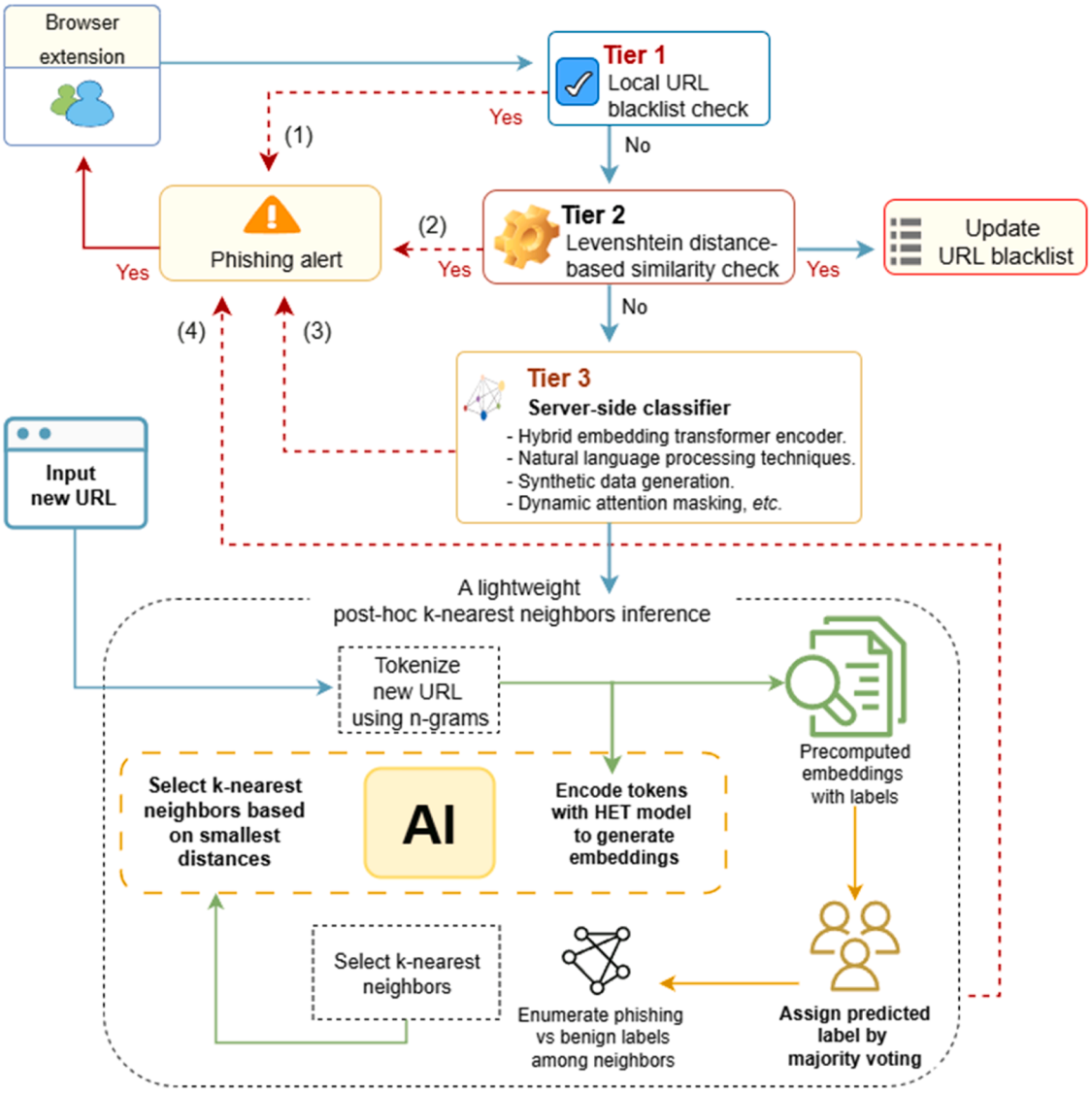

Fig. 2. Workflow of the proposed phishing detection framework combining blacklist filtering, similarity matching, and a server-side AI classifier.

preventing external query information and improves user experience by enabling faster detection of malicious websites. Moreover, the algorithm supports dynamic updates and maintains detection accuracy and resilience against evolving phishing threats.

3.3 Browser-based detection of phishing URL variants

The proposed phishing detection framework integrates four algorithms (Algorithms 2–5) to enable real-time detection of phishing URL variants in a browser environment. This system employs a hybrid approach that combines local storage lookups, string similarity analysis, server-side verification, and an interactive user interface (UI) to enhance computational efficiency and detection accuracy.

Algorithm 2 (manifest.json) conFig.s the browser extension to ensure seamless integration within the browser environment. It specifies essential permissions for interacting with active webpages, retrieving URLs, and communicating securely with a remote verification server. Additionally, it initializes the core components, including `background.js` for detection logic and `popup.html` for UI rendering. Algorithm 3 (`background.js`) is the primary phishing detection engine that implements a multi-layered verification process. Initially, the system checks each URL against a locally stored phishing database using the Levenshtein distance metric, thereby enabling the identification of suspicious URLs with minimal computational overhead. The URL is immediately flagged as a phishing threat if the similarity threshold is exceeded. Otherwise, it undergoes secondary validation via a server-side check using an up-to-date global phishing repository. If the server confirms that the URL is malicious, it is added to the local blacklist to enhance future detection performance. Algorithm 4 focuses on rendering the UI and presenting phishing detection results in an accessible and

Algorithm 1
Store blacklist in the local storage of the browser.

Input: blacklist.txt (a text file containing a list of blacklisted URLs)
Output: save_blacklist.js (a JavaScript file that stores the blacklist in local storage)

Step 1: Blacklist file processing
1 Define filePath as the absolute path to blacklist.txt.
2 Read all lines from blacklist.txt and store them in a list blacklist.
3 If the file reading operation fails, log the error and terminate execution.
Step 2: JSON conversion
4 Construct a JSON-formatted string blacklistJson based on blacklist:
- Concatenate all URL entries in blacklist using ", " as a delimiter.
- Enclose the concatenated string in square brackets [and] to create a valid JSON array.

Step 3: JavaScript code generation
5 Construct the JavaScript command script for storing the blacklist in the local storage: script = "localStorage.setItem('blacklist', 'blacklistJson')";
6 Replace 'blacklistJson' with the actual JSON-formatted blacklist string.
Step 4: Script file generation
7 Define scriptPath as the target location for save_blacklist.js.
8 Write script to save_blacklist.js.
9 If the file writing operation fails, log the error and terminate execution.
10 Upon successful completion, output the message: text{" File save_blacklist.js has been successfully created!"}
Complexity analysis
– File reading complexity: $O(n)$, where n denotes the number of URLs in blacklist.txt.
– String processing complexity: $O(n)$ as the transformation to JSON format requires the concatenation of $n$elements.
– File writing complexity: $O(1)$ since the output file is a single JavaScript script of limited size.
– Overall computational complexity: $O(n)$, ensuring scalability for handling thousands of blacklisted URLs efficiently.

Algorithm 2
Browser extension configuration for phishing URL detection (manifest.json).

Input: Browser environment configuration
Output: ConFig.d browser extension with permissions and service integration

Step 1: Manifest definition
1 Initialize JSON structure $\mathscr{M}$ for the browser extension:
$\mathscr{M} = \{"\text{manifest\_version}" : 3, "\text{name}" : \mathscr{N}, "\text{version}" : \text{v}\}$
where $\mathscr{N}$ = "phishing URL Checker" and $v = 1.0$.
Step 2: Permission allocation
2 Define permission set $\mathscr{P}$:$\mathscr{P} = \{"\text{storage}", "\text{activeTab}", "\text{scripting}"\}$
3 Assign host access permissions: $\mathscr{H} = \{' < \text{all\_urls} > '\}$
Step 3: Background service initialization
4 Define background script $\mathscr{B}$:
$\mathscr{B} = \{"\text{service\_worker}" : \mathscr{S}_B\}$, where $\mathscr{S}_B = "\text{background.js}"$.
Step 4: UI configuration
5 Specify user interaction parameters $\mathscr{U}$:
$\mathscr{U} = \left\{"\text{default\_popup}" : \mathscr{P}_H\right\}$ where $\mathscr{P}_H = "\text{popup.html}"$.
Step 5: Final manifest construction
6 Construct the final manifest file: $\mathscr{M} = \mathscr{M} \cup \{\mathscr{P}, \mathscr{H}, \mathscr{B}, \mathscr{U}\}$
7 Deploy $\mathscr{M}$ in the browser environment
Complexity analysis
– Manifest parsing complexity: $O(1)$ as it consists of fixed JSON key-value pairs.
– Permission validation complexity: $O(p)$, where $p = |\mathscr{P}|$ (constant in practice)
– Service worker initialization complexity: $O(1)$ as it loads a predefined script.

comprehensible format. The interface (`popup.html`) is designed using an adaptive layout and visual indicators using CSS-based styling to differentiate between benign and malicious websites. Algorithm 5 establishes a dynamic link between the detection engine and UI. Upon activation, `popup.js` extracts the URL of the currently active browser tab, invokes the detection module, and updates the UI accordingly. If a phishing threat is detected, the interface provides immediate feedback using interactive elements, such as color-coded warnings and alert messages.

The proposed system achieves an optimal balance between computational efficiency and detection accuracy by integrating the four aforementioned algorithms into a structured detection pipeline. The use of local-first heuristics reduces the dependency on remote servers, whereas server-assisted validation ensures robustness against evolving phishing strategies. This hybrid design outperforms conventional single-layer detection approaches, making it suitable for real-time deployment in modern browser environments.

Algorithm 2 outlines the deployment strategy of the phishing detection model using a Chrome-compatible browser extension, enabling real-time URL interception and classification. Constructed using the Manifest V3 architecture, the extension adopts a service worker-based design (background.js) to facilitate asynchronous background execution. It requests only essential permissions—storage, activeTab, scripting, and <all_urls>—thereby ensuring proactive detection across all tabs while adhering to the principle of least privilege. The core logic handles runtime feature extraction, model inference, and threat response, bridging the detection engine and

Algorithm 3
Phishing URL detection via Levenshtein distance and server-side verification (background.js).

Input: U (URL string)
Output: $Y \in \{P, B, U\}$ (Classification: Phishing, Benign, Unknown)

Step 1: Check local storage for phishing URLs
1 Retrieve stored phishing URLs P_{local} from local storage.
2 If $U \in P_{local}$, classify as $Y = \mathbb{P}$ and trigger alert $N(P, U)$, then terminate.
Step 2: Compute Levenshtein distance
3 Define $M \in Z^{(|U|+1)\times(|V|+1)}$, where $V \in P_{local}$ (local phishing database).
4 Initialize boundary conditions:
$M_{i,0} = i, \forall i \in [0, |U|]$, $M_{0,j} = j, \forall j \in [0, |V|]$
5 Iterate over $i \in [1, |U|]$ and $j \in [1, |V|]$:
$M_{i,j} = min\{M_{i-1,j} + 1, M_{i,j-1} + 1, M_{i-1,j-1} + \delta(U_i, V_j)\}$,where
6 Compute $d_L(U,V) = M_{|U|,|V|}$.
Step 3: Local phishing list matching
7 Retrieve stored phishing URLs P_{local}.
8 Define threshold τ:
If $d_L(U,V) \le \tau$, classify $Y = P$ and terminate
Step 4: Server-side verification
9 If Y is undetermined, send request:
Case: If $U \notin P_{local}$ and $d_L(U,V) > \tau$, classify $Y = U$(*Unknown*),
then: $\mathscr{R} = \mathrm{POST}(S, \{U\})$ (for instance, $\mathscr{R} = \mathrm{SendRequest}(S, U)$), where S denotes server-side verification.
10 Retrieve server response Y', where:
$Y' = \begin{cases} P, & \text{if the server detects phishing;} \\ B, & \text{if the server detects benign} \end{cases}$
11 If $Y' = P$:
- Update the local phishing set: $P_{local} \leftarrow P_{local} \cup \{U\}$.
- Set $Y = P$ and proceed to Step 5.

12 If $Y' = B$, assign $Y = B$ (*no further action is required*).
Step 5: Notification trigger
13 If $Y = P$, trigger alert: $\mathscr{N}(P, U)$.
Complexity analysis
– Levenshtein distance: $O(mn)$, where $m = |U|$, and$n = |V|$.
– Database lookup: $U(k \cdot mn)$, where $k = |P_{local}|$.
– Overall complexity: $O(k \cdot mn)$, assuming k to be small for real-time efficiency.

Algorithm 4
UI for phishing URL detection (`popup.html`).

Input: Browser popup interaction
Output: Real-time phishing detection result displayed to the user

Step 1: Document structure definition
1 Define the document object $\mathscr{D}$ as an HTML structure:
where:
o $\mathscr{H}_D$ represents the document head.
o $\mathscr{B}_D$ represents the document body.
o $\mathscr{S}_D$ represents the embedded script.
Step 2: Style configuration
2 Define CSS styling set $\mathscr{C}$: $\mathscr{C} = \{c_1, c_2, c_3\}$, where:
o c_1: body{width: 200px; font-family: Arial, sans − serif; }
o c_2: .safe{color: green; }
o c_3: .phishing{color: red; font-weight: bold; }
Step 3: UI component initialization
3 Define interface components $\mathscr{U}$:
$\mathscr{U} = \langle \mathscr{H}_{\mathscr{U}}, \mathscr{P}_{\mathscr{U}}, \mathscr{S}_{\mathscr{U}} \rangle$,where:
o $\mathscr{H}_{\mathscr{U}} \le$ h3 > Phishing Checker < /h3 >
o $\mathscr{P}_{\mathscr{U}} \le$ p id = 'result' > Checking... < /p >
o $\mathscr{S}_{\mathscr{U}} \le$ script src = 'popup.js' > < /script >
Step 4: Integration and rendering
4 Construct the final UI document:
5 Deploy $\mathscr{D}$ in the browser environment to enable phishing detection visualization.
Complexity analysis
– DOM initialization complexity: $O(1)$ as the document structure is static.
– CSS application complexity: $O(c)$, where $c = |\mathscr{C}|$, representing the number of styles applied.
– Script execution complexity: $O(1)$ since `popup.js` is linked externally and is executed asynchronously.

user interface effectively for seamless phishing prevention in real-world browsing scenarios.

Algorithm 3 detects phishing URLs using a hybrid approach that combines the Levenshtein distance metric [67,68] with server-side verification to balance detection efficiency and accuracy. First, it compares the input URL with entries in the locally stored phishing list

Algorithm 5
Real-time phishing detection in browser popup (`popup.js`).

Input: Active browser tab URL U
Output: Phishing detection result displayed in the popup

Step 1: Retrieve active tab URL
1 Query the active browser tab:
T = chrome.tabs.query({active: true, currentWindow: true})
2 Extract the URL U from the retrieved tab object:
Step 2: URL classification via detection function
3 Invoke phishing detection function:
R = checkURL(U),
where R denotes the classification result with$R \in \{\mathscr{S}, \mathscr{P}\}$,where:
- $\mathscr{S}$ (Safe), representing U is benign.
- $\mathscr{P}$ (Phishing), representing that U is detected as a phishing URL.

Step 3: Update UI with detection result
4 Modify the document object model (DOM) to reflect R:
$\text{document.getElementById}(I_R).\text{innerHTML} \leftarrow \begin{cases} \text{'Safe' if } R = S \\ \text{'Phishing' if } R = P \text{'} \end{cases}$
where I_R denotes the ID of the result display element

Complexity analysis
– Tab query complexity: $O(1)$ as the active tab is retrieved with a single API call.
– URL classification complexity: $O(f)$, where f denotes the complexity of checkURL(U) depending on the phishing detection method.
– DOM update complexity: $O(1)$ since a single element is modified.
– Overall complexity: $O(f)$, dominated by the detection function.

within the memory of the browser. If an exact match is detected, the URL is immediately flagged as phishing; otherwise, the system computes its Levenshtein distance to each stored entry. If this distance is below an empirically optimized threshold (τ), the URL is classified as phishing, and the user is notified. If the URL remains unclassified, it is transmitted to the server-side verification module

Table 3
Conceptual comparison between blacklist/whitelist-based methods and the proposed hybrid algorithm.

Criterion	Blacklist/Whitelist-Based Method	Algorithm 3: Phishing URL Detection via Levenshtein Distance and Server-Side Verification
Operational Principle	Relies on a fixed list: The blacklist comprises malicious URLs, while the whitelist includes safe URLs. If a URL is found in the blacklist, it is classified as phishing; if found in the whitelist, it is classified as benign; if absent from both, its classification remains undetermined.	Utilizes a locally stored blacklist for instant phishing detection, minimizing response time. If the URL is not found in this list, the Levenshtein distance algorithm is applied to detect phishing variants. A verification request is sent to the backend server if classification remains uncertain.
Novelty	Unable to handle URLs not present in the list, making it vulnerable to slightly modified versions of phishing URLs.	- Incorporates Levenshtein distance to detect variants of phishing URLs. - Integrates server-side verification to update the phishing list in real-time.
Innovativeness	Relies on frequent list updates, which may delay the detection of emerging phishing URLs.	- Enhances detection accuracy by analyzing the similarity between the input URL and blacklisted entries. - Supports dynamic updating of the phishing list based on server-side verification.
Improvement	- Highly dependent on blacklist/whitelist, prone to errors if updates are not timely. - Fails to classify URLs that are absent from the predefined lists.	- Reduces reliance on static lists, enabling detection of novel phishing URLs and their variants. - Improves efficiency by leveraging the Levenshtein distance algorithm with complexity = $O(mn)$. - Leverages local storage for rapid phishing detection, reducing server-side verification. If a phishing URL is confirmed via server-side verification, it is automatically added to the local blacklist, enhancing future detection efficiency and enabling offline phishing detection.
Ability to Detect New Phishing URLs	Low, as it can only identify URLs explicitly listed in the database.	- High, due to the combination of Levenshtein distance and server-side verification. - Handling of undetermined cases: If the backend server fails to classify a URL, the system alerts the user and allows them to decide whether to store the URL, improving the detection of emerging phishing threats.
Performance (Runtime Complexity)	$O(1)$ (direct lookup in the list).	$O(k \cdot mn)$ (Levenshtein distance computation + server-side verification), but remains efficient in real-time scenarios since k is small.
Scalability	Requires manual or automated updates from external sources.	Easily scalable by refining the matching algorithm and integrating AI-based verification into the backend server.

Note: This comparison highlights architectural differences and inferred strengths based on the design characteristics of the respective systems, and does not account for experimental benchmarks.

for further analysis using the HET framework. If the server confirms that the URL is phishing, it is added to the local phishing database and updated to improve future detection. The user is promptly alerted to ensure real-time response. This mechanism reduces FPs and FNs by incorporating server-side verification and user feedback, thereby allowing the system to learn and adapt to emerging phishing techniques continuously. The algorithm exhibits a time complexity of $O(k \cdot mn)$, where k denotes the number of stored URLs, and m and n denote the lengths of the input and stored phishing URLs, respectively.

The scalability of the Levenshtein-based similarity check grows linearly with the size of the local blacklist. The latency can be approximated by the model $T(B) \approx B \times O(mn)$, where B is the number of blacklist entries and m, n denote the lengths of the compared URLs. This implies that, as the blacklist expands beyond tens of thousands of entries, the client-side computation cost becomes non-negligible. In such cases, unresolved or ambiguous URLs are escalated to the server-side HET classifier, whose inference latency is largely invariant of the blacklist size. To further improve scalability in future deployments, indexing structures such as BK-trees or trie-based pruning can be integrated to reduce the number of full Levenshtein evaluations required per query.

Traditional phishing URL detection methods rely primarily on static blacklists and whitelist databases, which are limited in their ability to handle novel phishing variants. To overcome this drawback, the proposed algorithm integrates Levenshtein distance with server-based verification to improve the detection of previously unrecorded phishing attempts. Notably, it updates the local phishing list dynamically, reduces reliance on repeated server checks, and enhances detection efficiency in subsequent analyses. Table 3 compares the two approaches in terms of novelty, innovation, improvement, and scalability.

Algorithm 4 outlines the UI design for displaying the phishing URL detection results using an interactive browser popup, providing real-time feedback to users. The document structure (D)comprises three main components—head (H_D), body (B_D), and embedded script (S_D)—ensuring a modular and scalable layout. The styling configuration (C) defines visual cues—green for safe URLs and bold red for phishing alerts—enhancing user comprehension. The UI components (U) include a header, dynamic result placeholder, and a JavaScript code that enables real-time phishing analysis and user interaction. Finally, the complete UI document is assembled and deployed within the browser environment during the integration and rendering steps, enabling seamless phishing detection visualization. Given the varying browser security vulnerabilities, real-time phishing alerts are crucial for user protection, as highlighted in previous studies [69]. Therefore, incorporating interactive elements into the detection UI may further enhance the ability of users to recognize and respond to phishing threats in real-time.

Algorithm 5 implements a real-time phishing detection mechanism by extracting the URL from an active browser tab and classifying it using a predefined function. First, the algorithm retrieves the current URL, which is then transmitted to the phishing detection module. This module categorizes the URL as either safe (S) if the URL is benign or phishing (P) if it is identified as malicious. The result is displayed in the UI by updating a designated document object model element in the pop-up window of the extension. This approach enables fast, intuitive threat identification immediately following page access, improves user awareness, and minimizes the risk posed by malicious websites.

In summary, the proposed phishing URL detection algorithm integrates Levenshtein distance computation with server-side verification, employs an HET-based DL classifier, and uses a multilayered detection framework to achieve high detection accuracy while maintaining real-time performance. The system first checks a locally stored phishing database for immediate classification. If the queried URL is not found, its Levenshtein distance from the known phishing URLs is calculated to assess the similarity. URLs with similarity scores below a predefined threshold are flagged as phishing; otherwise, the URL is forwarded to a backend server where the DL model performs the final classification. In addition, a dynamic update mechanism is performed to refresh the phishing URL repository continuously, enhancing the adaptability and robustness of the system against emerging threats. This architecture enables scalable and effective phishing mitigation in dynamic, real-world environments.

3.4. HET model for phishing URL variant detection

The dynamic masking mechanism specifically targets phishing-indicative n-gram patterns that frequently appear in obfuscated or evasive URLs. The synthetic token ".." was selected because it is a commonly observed artifact in phishing URLs created through path obfuscation (e.g., repeated directory traversals, hidden redirect chains, or adversarial insertion of redundant separators), and it strongly correlates with malicious intent in several public phishing datasets. By masking these positions during attention computation, the model is encouraged to redistribute attention toward more structurally meaningful regions of the URL, thereby enhancing robustness against obfuscation-based attacks.

The original dataset used in this study is the PhiUSIIL Phishing URL Dataset [73], comprising labeled benign and phishing URLs with diverse lexical and structural patterns. The detailed dataset statistics (including total size and class distribution) are provided in Section 3.6. This subsection focuses on the preprocessing pipeline applied prior to model training.

To enhance robustness against obfuscation-based phishing techniques, a synthetic augmentation step is applied. For each benign URL in the training set, one to five occurrences of the special token ".." are randomly inserted at arbitrary positions to create adversarial-style phishing variants. This procedure yields approximately 5,000 additional phishing samples, enriching the dataset and improving generalization to unseen attack patterns.

Before feeding data to the HET encoder, all URLs are standardized into a token sequence using character-level bi-gram encoding. Formally, a URL $U = (c_1, c_2, \ldots, c_L)$ is transformed into overlapping bi-grams: $G(U) = \{ g_i = c_i c_{i+1} | i = 1, \ldots, L-1 \}$.

A vocabulary is constructed from all bi-grams in both the original and synthetic URLs, and each bi-gram is mapped to an integer index through a tokenizer. Sequences are then zero-padded to a fixed length of 200 tokens to allow efficient batching during training and inference. Bi-gram tokenization is deliberately chosen over character-level tokenization for URL modeling. This is because, unlike natural language text, URLs exhibit highly localized lexical patterns—such as "//", ".com", "..", "/?", "=&"—which serve as strong

phishing indicators. Bi-grams capture these short-range dependencies more effectively than single characters. This design yields a compact yet discriminative representation tailored to phishing-specific structures.

The resulting preprocessed dataset therefore encompasses:

(i) the original labeled URLs,
(ii) 5,000 synthetic adversarial variants containing the ".." token,
(iii) bi-gram encoded token sequences with fixed-length padding, and
(iv) vocabulary indices optimized for transformer-based embedding.

Algorithm 6 presents the HET-based phishing URL detection framework proposed in this study. The model departs from conventional transformer architectures by introducing targeted innovations tailored to the challenges of real-world phishing detection. Unlike the original transformer structure, which comprises both encoder and decoder components optimized for sequence transduction, HET adopts an encoder-only design that reduces computational overhead and supports faster inference. These architectural modifications enhance the model's suitability for classification tasks and enable its integration into client-side or real-time environments.

The first phase of Algorithm 6 transforms the input URL sequence U into structured representations for model processing. This transformation comprises three key steps. (1) First, n-gram encoding partitions the URL into overlapping segments, generating a token sequence $G(U) = \{g_1, g_2, \ldots, g_m\}$. This enables the model to capture the structural and lexical patterns present within the URL. (2) Synthetic data generation is performed by randomly inserting '..' into legitimate URLs present in the training dataset D_{train} to construct a synthetic dataset $D_{synthetic}$ in order to enhance robustness against obfuscation attacks. (3) Vocabulary encoding constructs a vocabulary V from the encoded dataset, mapping each token to an integer index using an encoding function T. This ensures that critical special characters, such as "..', retain their semantic significance. Subsequently, an attention mask computation $M \in \mathbb{R}^{|U|\times|U|}$ is computed to guide the attention of the model toward the most significant parts of the URL. For standard tokens, $M_{i,j} = 0$ is assigned to enable conventional self-attention. If U_i = '..', then $M_{i,j} = 1$, thereby emphasizing critical URL segments. The mask is then expanded across multiple self-attention heads, for instance, $M' = \text{expand}(M, \text{num}_{\text{heads}})$, optimizing the learning process. The primary steps of transformer-based URL representation include (1) token and positional encoding, where the input sequence is embedded via token embeddings $E(T(U))$ and positional embeddings $P(T(U))$; (2) the multi-head self-attention mechanism, governed by the following Eq.:

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M'\right)V, \tag{1}$$

where Q, K, and$V$ denote linear projections of the input sequence; and (3) the FFN, where the attention output is passed through two non-linear layers, as follows:

$$Z = \text{ReLU}(W_1 A + b_1), \; O' = W_2 Z + b_2 \tag{2}$$

The classification process uses global average pooling to extract a feature vector $H = \text{GlobalAveragePooling}(O)$, followed by computing the classification probabilities.

$$P(Y) = \text{softmax}(W_H H + b_H) \tag{3}$$

The classification decision is based on predefined thresholds—if $P(Y = P) \geq \tau$ (default threshold of $\tau = 0.5$, adjustable), the URL is classified as phishing (P); if $P(Y = B) \geq \tau$, it is categorized as benign (B). If a URL is identified as phishing, the system issues an alert, updates the malicious URL repository, and sends a warning to the user. Finally, the computational complexity analysis of Algorithm 6 reveals that (1) tokenization and vocabulary encoding exhibits a complexity of $O(n)$, where n denotes the URL length; (2) attention mask computation exhibits a complexity of $O(n^2)$; and (3) forward propagation through the transformer exhibits a complexity of $O(h \cdot d_k \cdot n^2)$, where$h$ represents the number of attention heads in the multi-head self-attention mechanism, d_k denotes the dimensionality of the vector space for each attention head, and n denotes the length of the input sequence corresponding to the number of tokens in the URL. In self-attention, the query (Q), key (K), and value (V) matrices exhibit dimensions of ($n \times d_k$). The weighted attention computation is based on matrix multiplication between Q and K:

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M'\right), \tag{4}$$

where QK^T represents the matrix multiplication between two matrices of dimensions ($n \times d_k$) and ($d_k \times n$), resulting in an attention matrix of size ($n \times n$), with a computational complexity of $O(n^2 d_k)$. Multiplication of the weighted attention matrix with the value matrix (V), with dimension $= n \times d_k$, further maintains the complexity at $O(n^2 d_k)$. Since the transformer employs h attention heads, the total complexity for self-attention computation becomes $O(h \cdot d_k \cdot n^2)$.

Although a complexity of $O(n^2)$ raises concerns about computational cost, several factors mitigate its impact in the context of phishing URL detection:

Algorithm 6
Phishing URL detection via HET.

Input: U (URL string)
Output: $Y \in \{P, B\}$ (Classification: Phishing, Benign)

Step 1: Preprocessing and feature extraction
1 Tokenization using n-grams
– Initialize an empty sequence $G(U)$.
– For $i = 1$ to $|U| - n + 1$ do:
  • Extract n-gram $g_i = U[i: i + n]$.
  • Append g_i to $G(U)$.
– Store $G(U)$ as input tokens.
2 Synthetic data generation
– For each $U' \in D_{train}$ do:
  • Select a random index $k \in [1, |U'|]$.
  • Insert special token '..' at position k.
  • Store U' in synthetic dataset $D_{synthetic}$.
3 Vocabulary encoding
– Construct vocabulary V from $G(D_{train} \cup D_{synthetic})$.
– Define a mapping function $T : V \rightarrow Z$.
– Assign a special index $T('..') = t_{special}$.
Step 2: Attention mask computation
4 Initialize attention mask
– Define a matrix $M \in R^{|U| \times |U|}$, initialized as follows:
  $M_{i,j} = 0, \forall i, j$.
– For each $i = 1$ to $|U|$ do:
  • If $U_i = '..'$ then: $M_{i,j} = 1, \forall j$.
5 Expand mask for multi-head attention
– Compute multi-head mask:
  $M' = \text{expand}(M, \text{num\_heads})$
Step 3: Transformer-based URL representation
6. Token & positional embedding
– Encode U within token embeddings:
  $X = E(T(U)) + P(T(U))$,
– where E denotes a token embedding and P denotes a positional embedding.
– If $T(U_i) = t_{special}$, then add an additional embedding.
7. Multi-head self-attention computation
– Compute query, key, and value projections:
  $Q = W_Q X, K = W_K X, V = W_V X$
– For each head h do the following:
  • Compute attention weights:
  $$A_h = \text{softmax}\left(\frac{Q_h K_h^T}{\sqrt{d_k}} + M'\right)$$
  • Compute output:
  $O_h = A_h V_h$
– Concatenate outputs:
  $O = \text{concat}(O_1, O_2, \ldots, O_h)$
8. FFN
– Apply a two-layer FFN with ReLU activation: where O' is directly derived from the intermediate ReLU-transformed output Z
  • The first linear layer transforms the attention output O and applies a ReLU activation:
$Z = \text{ReLU}(W_1 O + b_1)$
  • The second linear layer maps Z back to the original dimension without activation:
Step 4: Classification & decision-making
9 Global feature extraction
– Compute global feature vector:
  $H = \text{GlobalAveragePooling}(O')$
– Compute probability scores:
  $P(Y) = \text{softmax}(W_H H + b_H)$
10 Threshold-based classification
– Set threshold $\tau = 0.5$ (default or dynamically adjustable)
– If $P(Y = P) \geq \tau$, then:
  • Set $Y = P$ (Phishing).
  • Go to Step 5.
– Else $P(Y = B) \geq \tau$, then:
  • Set $Y = B$ (Benign).
Step 5: Post-processing and alerting
• If $Y = P$, then:
  – Trigger phishing alert: $N(P, U)$.
  – Store U within the phishing list.
  – Display user warning: "Please be cautious!"
Complexity analysis
– Tokenization and encoding: $O(n)$, where $n = |U|$.

(*continued on next page*)

Algorithm 6 (*continued*)

Input: U (URL string)
Output: $Y \in \{P, B\}$ (Classification: Phishing, Benign)

- Attention masking: $O(n^2)$.
- Transformer forward pass: $O(h \cdot d_k \cdot n^2)$, where h denotes the number of heads.
- Overall complexity: $O(h \cdot d_k \cdot n^2)$, ensuring real-time processing capability.

- URLs are shorter than the contents of typical NLP texts. Unlike typical NLP tasks, which process lengthy sequences (often hundreds of tokens), URLs are relatively short and typically contain fewer than 100 tokens. This reduces the computational burden substantially.
- HET refines attention masking. The attention mask mechanism is optimized to emphasize critical URL components (such as domain names and suspicious query parameters). This optimization reduces the computational overhead by filtering out less relevant tokens.
- Efficient parallelization on graphic processing units (GPUs): Self-attention matrix operations are highly parallelizable and well-suited for GPU acceleration, improving the inference speed significantly.

Because of these optimizations, Algorithm 6 (HET) achieves a high detection accuracy while maintaining real-time performance, making it suitable for deployment in large-scale cybersecurity systems.

Recently proposed phishing detection models have demonstrated promising results but continue to exhibit critical limitations. For instance, Sahingoz et al. [70] achieved 97.98 % accuracy using NLP-based features but relied on a small dataset, reducing generalizability. Zhu et al. [71] proposed a hybrid CNN-BiLSTM model, but its real-time performance and adversarial robustness were not evaluated. Do et al. [72] adopted n-gram embeddings, but failed to report inference latency, incorporate modern transformer-based comparisons, or integrate whitelist mechanisms. These research gaps motivate the development of a more robust, adaptive, and real-time solution, which is addressed by the proposed HET model.

In summary, despite the advances in recent works, limitations, such as lack of scalability, absence of transformer-based comparisons, and inadequate evaluation of inference latency and robustness, remain ubiquitous. To address these challenges, the proposed HET model incorporates several novel strategies optimized for phishing URL detection, including mixed embedding schemes, n-gram tokenization, dynamic attention masking, and adaptive classification thresholds. These innovations aim to enhance robustness, response adaptability, and deployment feasibility.

Table 4 summarizes the key architectural and training parameters of the proposed HET model. The configuration includes one encoder block with four attention heads and a 64-dimensional embedding space optimized for URL tokenization using bi-gram encoding. A special synthetic token (“..”) is incorporated to enhance phishing-specific feature learning through dynamic attention masking. The model is trained using the Adam optimizer with a learning rate of 1×10^{-3}, mini-batch size of 32, and 20 epochs across various train–test splits. These settings ensure balanced generalization and real-time inference efficiency for phishing versus benign URL classification.

The synthetic token “..” is selected because it frequently appears in real phishing URLs created through path obfuscation, redundant separators, and hidden redirect chains. Public phishing datasets show strong correlation between this pattern and malicious intent. Injecting “..” during augmentation exposes the model to realistic obfuscation behavior and improves robustness against adversarial URL manipulations. The dynamic attention mask suppresses attention on phishing-indicative n-grams—especially the “..” pattern—by reducing their attention logits. This forces the encoder to focus on stable, semantically meaningful URL components (e.g., domain, subdomain) rather than adversarial noise. Combined with synthetic augmentation, this mechanism enhances resilience against obfuscation-based and evasive phishing strategies.

Table 4
Configuration of the HET model, including architectural and training hyperparameters used for phishing–benign URL classification.

Component	Configuration (from implementation)	Description and Purpose
Input sequence length	L = 200	Maximum URL sequence length after n-gram tokenization.
Tokenizer & vocabulary	Character-level bi-gram (n = 2), vocabulary built from dataset	Captures local character adjacency and short-range dependencies.
Special token	“..” with trainable 64-dim embedding	Synthetic token inserted for augmentation and used in masking.
Embedding dimension	$d_{emb} = 64$ (includes positional)	Dimension of token, positional, and special embeddings.
Transformer blocks	N = 1	Single encoder block used in HET model.
Attention heads	h = 4	Multi-head self-attention projections.
Feed-forward dimension	d_ff = 64	Internal dense layer width inside Transformer block.
Dropout rate	p = 0.2	Applied to attention + FFN sublayers during training.
Normalization	LayerNorm (ε = 1e-6)	Stabilizes intermediate activations.
Dynamic attention mask	create_attention_mask() → positions = special_token_id	Suppresses positions of “..” (score += mask × −1e9).
Pooling layer	GlobalAveragePooling1D	Aggregates token-level features into a fixed vector.
Output layer	Dense (2, softmax)	Binary classification: benign (1) vs phishing (0).
Optimizer / Loss	Adam (lr = 1×10^{-3}), Sparse CE	Adaptive optimization suitable for integer labels 0/1.
Batch size / Epochs	B = 32, E = 20	Mini-batch training configuration.
Training/testing ratio	0.4–0.9 (optimal 0.7/0.3)	Used to determine best generalization split.

Table 5 summarizes the key technical differences between HET and prior approaches, highlighting the theoretical and practical contributions of the proposed model.

Beyond algorithmic design, the HET framework supports practical deployment based on integration with a lightweight browser extension. This plugin facilitates real-time phishing detection by combining local blacklist verification with Levenshtein distance-based analysis for unknown URLs. The system detects deceptive URLs closely resembling legitimate domains accurately using character-level similarity metrics. Additionally, unlabeled URLs are assessed dynamically via server-side HET inference, eliminating the need for manual blacklist updates. This hybrid configuration ensures responsive, scalable, and efficient defense across real-time browsing environments.

3.5. Post-hoc inference via embedding-based similarity

The post-hoc inference mechanism performs phishing URL detection by comparing the deep semantic representations of input URLs obtained from the HET encoder with a precomputed database of labeled embeddings. Specifically, each input URL is first tokenized using an n-gram strategy and transmitted to the HET encoder (the core AI component) to generate an embedding vector that captures its structural and lexical features. This embedding is then compared to existing labeled embeddings in terms of Euclidean distance. The top-k most similar reference URLs are selected via nearest-neighbor search, and a majority voting scheme is used to determine the final prediction (phishing or benign). This lightweight inference process supports fast and retraining-free classification while preserving adaptability to novel threats, making it suitable for real-time deployment scenarios. The overall workflow of this inference pipeline is illustrated in Fig. 3.

The overall procedure of this post-hoc inference mechanism is formalized in Algorithm 7, which outlines the complete pipeline beginning with embedding extraction and extending till prediction via nearest-neighbor voting. Traditional phishing URL detection approaches often require retraining or fine-tuning of the model when encountering new or unseen samples. Although these are effective in controlled environments, such procedures introduce substantial computational overhead and hinder real-time adaptability in dynamic deployment environments.

To mitigate this limitation, we propose a lightweight post-hoc inference mechanism that leverages the latent embedding representations learned by the proposed HET model. Instead of leveraging the final classification layer for decision-making, the proposed method infers the label of a new URL by computing its similarity with a set of previously seen and labeled URLs in the embedding space.

This embedding-based inference strategy reduces computational costs substantially by eliminating the need for model retraining. It also facilitates rapid adaptation to emerging phishing threats, enabling real-time, on-the-fly detection of new URLs based solely on their embedding vectors. Moreover, the approach provides a flexible foundation for potential integration with few-shot learning or continual learning paradigms.

Eq. (5) – Euclidean Distance in the Embedding Space

Let U_{new} be a newly observed URL, and $\boldsymbol{e}_{\text{new}} \in \mathbb{R}^d$ be its corresponding embedding vector obtained from the trained HET model. Let

Table 5
Technical Innovations in HET.

Aspect	Previous Approaches	Proposed Innovation (HET)	Novelty in Contribution
Embedding Design	Use only character-level or token-level embeddings without explicit encoding for phishing-specific patterns	Introduces special embeddings (e.g., for '..') and mixed-token encoding	Enables the model to capture obfuscation strategies commonly used in phishing URLs
Tokenization Strategy	Standard character-level or word-based tokenizers (e.g., BPE, WordPiece), insensitive to URL structure	Uses n-gram-based tokenization tailored for URLs	Captures localized patterns and segment-level semantics unique to phishing structures
Synthetic Data Generation	Rarely applied in phishing URL detection; mostly used in NLP or computer vision	Random insertion of special tokens (e.g., '..') into benign samples to generate adversarial-style phishing URLs	Enhances training diversity and generalizability without requiring access to additional real phishing datasets
Attention Mask Mechanism	Basic padding masks or uniform attention across all tokens	Dynamically masks attention to prioritize phishing-relevant tokens (e.g., suspicious subdomains, special characters)	Directs model capacity toward high-risk regions in URLs, improving discriminative power
Multi-Head Attention Adaptation	Generic fixed attention heads with uniform weight updates	Integrates phishing-sensitive attention by adjusting head weights for indicative tokens	Improves sensitivity to subtle contextual variations between benign and malicious URLs
Classification Strategy	Single-stage softmax classifier; decision boundary fixed at training time	Multi-stage classification: pre-filtering (Levenshtein), transformer-based scoring, and dynamic server-side fallback	Reduces FPs/FNs by using flexible decision boundaries based on risk level
Adaptive Thresholding	Static thresholds or raw softmax-based prediction	Adjusts classification threshold based on URL distribution and model uncertainty	Enables dynamic response to evolving URL types, enhancing real-time adaptability
Post-Processing/ User Feedback	Absent or static decision pipeline; does not account for uncertain or ambiguous predictions	Interactive post-processing: warns users of borderline cases and logs new patterns	Supports continual learning and human-in-the-loop evaluation, improving robustness against zero-day phishing patterns
Inference Efficiency	Often unreported or unsuitable for real-time applications; complex pipelines with high latency	All inference stages optimized: lightweight URL input, modular logic, optional server offload	Enables browser-based or client-side deployment with inference latency (26–52 ms per URL)

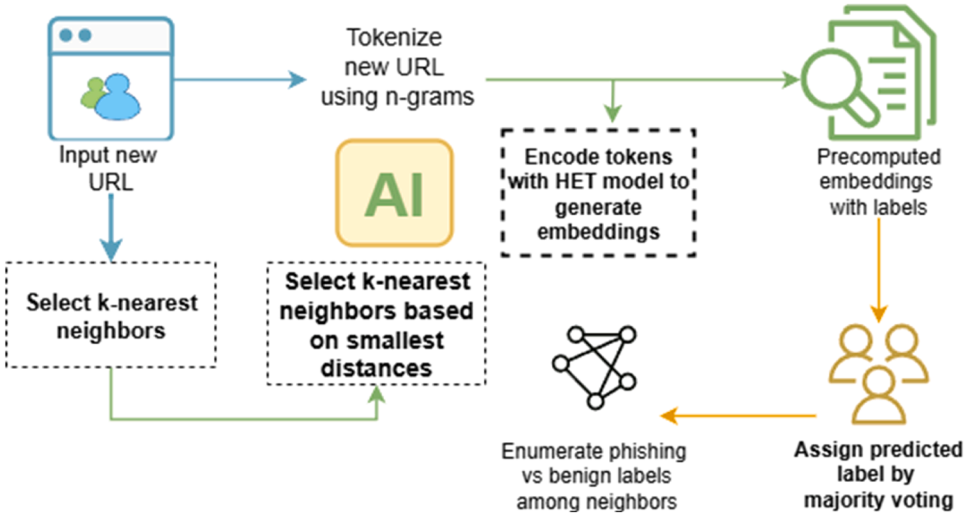

Fig. 3. Flowchart for post-hoc inference mechanism based on HET embeddings and k-NN voting.

Algorithm 7
Post-hoc inference using embedding-based similarity.

Input:
- U_{new}: New URL to classify
- $\{(U_i, y_i)\}_{i=1}^{N}$: Known labeled URLs and their labels
- k: Number of nearest neighbors
- Precomputed embeddings $\{e_i\}$: corresponding to known URLs $\{U_i\}$, obtained using the same HET model

Output:
- $\hat{y} \in \{0,1\}$: Predicted label for U_{new}

Step 1: Embedding Extraction
1 Tokenize U_{new} using n-grams
2 Encode tokens via HET model to obtain embedding $e_{\text{new}} \in \mathbb{R}^d$
Step 2: Distance Computation
3 For each $i = 1$to N:
  Compute Euclidean distance $d_i = \|e_{new} - e_i\|_2$
Step 3: k-Nearest Neighbor Selection
4 Identify index set $K \subseteq \{1, \ldots, N\}$ of the top-k URLs with smallest d_i
Step 4: Majority Voting
5 Count number of phishing labels $y_i = 0$ and benign labels $y_i = 1$among $i \in K$
6 Assign predicted label:
  If majority is benign (1), then $\hat{y} = 1$; else $\hat{y} = 0$
Step 5: Incremental Update Without Retraining
7 If the true label of U_{new} becomes available
– Compute its embedding e_{new}using the same HET encoder.
– Append the new embedding and label directly:
  $e_{N+1} \leftarrow e_{\text{new}}, y_{N+1} \leftarrow y_{\text{new}}$
Step 6: Return $\hat{y}$

$\{\boldsymbol{e}_i\}_{i=1}^{N}$ denote the set of embedding vectors corresponding to known URLs $\{\boldsymbol{U}_i\}_{i=1}^{N}$, each associated with a binary label $y_i \in \{0,1\}$, where 0denotes phishing and 1denotes benign.

The Euclidean distance between $\boldsymbol{e}_{\text{new}}$ and each known embedding $\boldsymbol{e}_i$is computed as follows:

$$d_i = \|\boldsymbol{e}_{new} - \boldsymbol{e}_i\|_2 = \sqrt{\sum_{j=1}^{d} \left(\boldsymbol{e}_{new,j} - \boldsymbol{e}_{i,j}\right)^2} \tag{5}$$

This metric quantifies the similarity between the new URL and each known sample in the latent embedding space.

Eq. (6) – Label Prediction via k-NN Voting

Based on the distance set $\{d_i\}_{i=1}^{N}$, we identify the top-k most similar URLs. Let $K \subseteq \{1, 2, \ldots, N\}$ be the index set corresponding to the k nearest neighbors, and let $y_i \in \{0,1\}$ be the ground-truth label for the i-th known URL. The predicted label $\hat{y}$ for the new URL is obtained via majority voting as follows:

$$\hat{y} = \arg\max_{y \in \{0,1\}} \sum_{i \in \mathscr{K}} 1(y_i = y), \tag{6}$$

where$1(y_i = y)$ is the indicator function, returning 1 if $y_i = y$, and 0 otherwise, and $\hat{y}$ denotes the final predicted label for U_{new}.

Note: Here, k refers to the number of nearest neighbors considered during majority voting. The set $K \subseteq \{1, 2, \ldots, N\}$ contains the indices of the top-k most similar URLs, determined in terms of Euclidean distance. This clarification distinguishes between the scalar k and the index set $\mathscr{K}$ used in Eq. (6).

Eq. (7) – Voting Confidence Metric

$$C(U_{\text{new}}) = \frac{1}{k} \sum_{i \in \mathscr{K}} 1(y_i = \hat{y}\ ), \tag{7}$$

where $C(U_{\text{new}}) \in [0, 1]$ denotes the confidence score based on neighbor agreement, k is the number of nearest neighbors, K is the corresponding index set with $|K| = k$, and $\hat{y}$ is the predicted label obtained via majority voting from Eq. (6). A higher value of $C(U_{\text{new}})$ reflects stronger local consensus and higher prediction reliability.

Eq. (8) – Accuracy Approximation under Local Density

$$\text{Acc}(k) \approx 1 - \frac{1}{k} \sum_{i \in K} \left[1 - \exp\left(-\lambda d_i^2\right)\right], \tag{8}$$

where d_i is the Euclidean distance from Eq. (5), $\lambda \propto \rho^{-1}$ is inversely proportional to the local embedding-space density ρ, and $k = |K|$ is the neighborhood size. This empirical formulation models how denser local clusters (smaller d_i) yield higher accuracy, whereas *excessively large k* introduces samples from different clusters, reducing discriminative precision. Eq. (8) thus captures the trade-off between locality and robustness in post-hoc k-NN refinement.

Eq. (9) – Latency Complexity per Inference

$$T_{\text{inf}}(k, N) = T_{\text{embed}} + \alpha N d + \beta N \log k, \tag{9}$$

where T_{embed} is the embedding extraction time from the HET encoder, αNd is the cost of computing Euclidean distances for Nembeddings of dimension d, and $\beta N \log k$ reflects the top-k selection cost (using a partial sort), followed by k-NN voting. For the experimental configuration (N ≈ 2000, d = 64, k = 5), the end-to-end latency ranges 26–52 ms, consistent with the empirical mean, median, and p95 values reported in Section 4.5.

Note: The value $N \approx 2000$ refers to the reference embedding subset used during inference-time testing. The full training dataset (including synthetic augmentation) was significantly larger, with ≈ 5000 additional samples.

Unlike conventional approaches that require retraining or fine-tuning when encountering unseen URLs, the proposed inference mechanism enables immediate, real-time classification solely based on embedding similarity. This characteristic makes it particularly suitable for practical deployments, such as browser-based phishing detection systems or network security gateways.

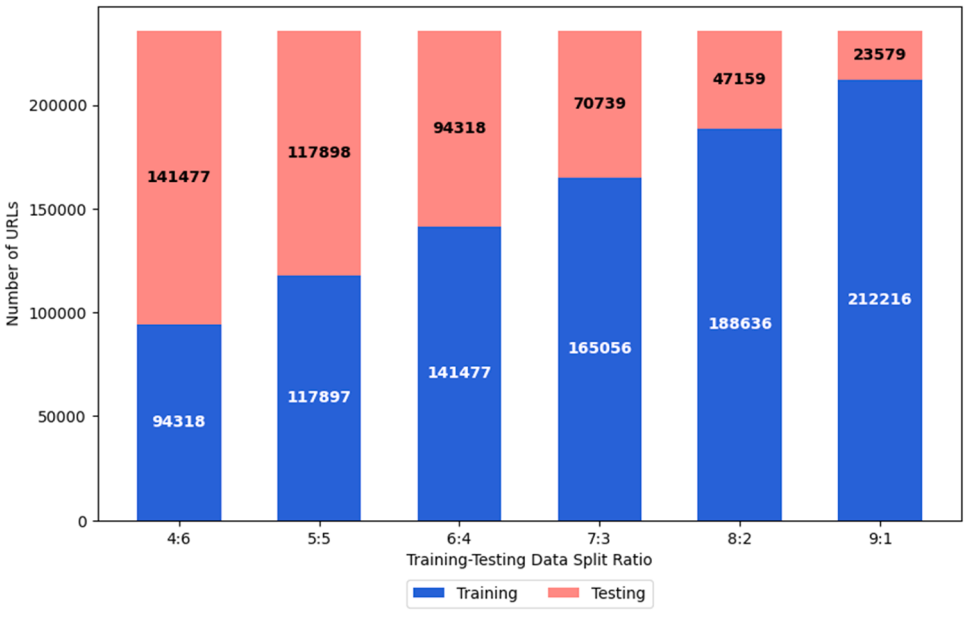

Fig. 4. Dataset splits used for training and testing.

3.6 Training and testing split ratios

This study utilizes the PhiUSIIL Dataset [73], a benchmark dataset designed for classifying phishing and benign URLs obtained from a reputable journal specializing in cybersecurity. The PhiUSIIL Dataset consists of 235,795 URLs, including 134,850 benign and 100, 945 phishing URLs collected from diverse sources to ensure data variety and label accuracy. Different training–testing data ratios are considered and key performance metrics, including accuracy, loss, precision, recall, F1-score, training and testing times, and AUC are evaluated for Gradient Boost (GB), AB, CatBoost (CB), XGB, MLP, KDR (an ensemble baseline comprising k-Nearest Neighbors (KNN), Decision Tree (DT), and RF), and HET. Specifically, the size of the testing dataset represents the proportion of data used for model evaluation, whereas the training dataset comprises the remaining portion used for model learning. The following Eq. defines the experimental configuration and determines the optimal training-to-testing data ratio for predictive modeling:

$$Testing\ (i) = 1 - Training\ (i),$$

where i represents the simulation scenario index, ranging between 4 and 9. The specific distributions of the training and testing datasets are depicted in Fig. 4. The corresponding number of testing samples decreases as the number of URLs in the training set increases.

3.7 Evaluation metrics

The effectiveness and generalization capability of the proposed HET framework is evaluated rigorously in the context of phishing URL detection in terms of a suite of widely accepted classification metrics. These metrics provide comprehensive insights into the model's ability to distinguish between phishing and benign URLs under various deployment scenarios. All metrics are computed based on the confusion matrix components:

- True Positives (TP): The number of phishing URLs that are correctly identified as phishing by the model.
- True Negatives (TN): The number of benign URLs that are accurately classified as benign.
- FP: The number of benign URLs that are incorrectly classified as phishing.
- FN: The number of phishing URLs that are incorrectly classified as benign.

The evaluation metrics are defined as follows:

- Accuracy, which reflects the overall correctness of predictions:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

- Precision, which quantifies the proportion of correctly identified phishing URLs among all predicted phishing cases:

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

- Recall (also known as sensitivity), which measures the model's ability to detect phishing URLs:

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

- F1-Score, the harmonic mean of precision and recall, offering a balanced view especially corresponding to imbalanced class distributions:

$$F1 - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{13}$$

- AUC Score (Area Under the Receiving Operating Characteristic Curve), which evaluates the model's ability to discriminate between classes across thresholds:

$$AUC = \int_0^1 TPR(f)\ df \tag{14}$$

- Binary Cross-Entropy Loss

For a binary classification task with ground-truth label $y_i \in \{0,1\}$ and predicted probability $\hat{y}_i \in [0,1]$, the binary cross-entropy (BCE) loss for a single sample is:

$$\ell_i = -[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)].$$

The total loss over N samples is computed as:

$$L_{\text{BCE}} = \frac{1}{N}\sum_{i=1}^{N} \ell_i, \tag{15}$$

where N is the number of training samples, y_i denotes the true label (0 = phishing, 1 = benign), and $\hat{y}_i$ is the predicted probability that the URL belongs to the benign class produced by the HET model. This loss penalizes incorrect predictions using a logarithmic scaling, ensuring very high penalties when the model is confident but wrong. BCE remains the standard objective function for binary deep-learning classifiers.

The integration of embedding-based post-hoc inference within phishing URL detection presents a computationally efficient alternative to conventional retraining strategies. By leveraging latent vector representations and k-NN voting in the embedding space, the proposed mechanism enables fast and accurate classification of unseen URLs without requiring additional model updates. This design is particularly well-suited for real-time cybersecurity applications, such as browser-integrated detection systems or network-level intrusion prevention frameworks, where adaptability and low latency are critical.

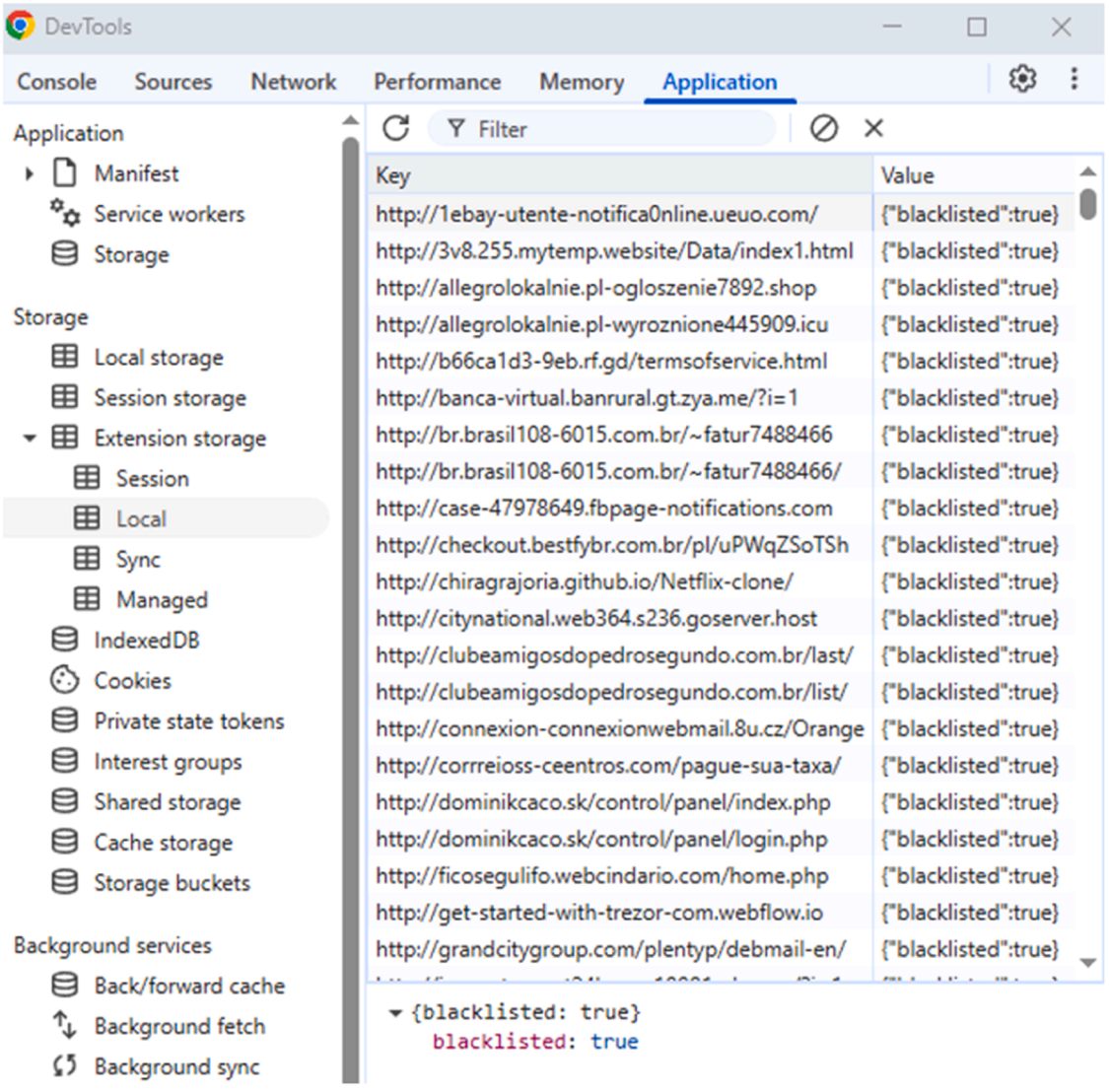

Fig. 5. Implementation of URL blacklist using local storage.

Empirical evaluation in terms of multiple standard classification metrics, including Accuracy, Precision, Recall, F1-score, Loss, and AUC, demonstrates the proposed model's robust generalizability across varied data distributions and experimental splits. These metrics are computed using well-established formulas reported in ML literature, where Accuracy measures overall prediction correctness, Precision captures the proportion of TPs among predicted positives, Recall reflects the model's ability to identify actual positives, F1-score balances Precision and Recall, and AUC assesses the trade-off between TP rates and FPRs across thresholds. This evaluation framework is consistent with previous approaches in phishing detection and cybersecurity classification tasks [74,75], reinforcing the reliability and relevance of these performance indicators.

4. Results

Experimental evaluations are conducted on a high-performance computing server with dual Intel Xeon E5-2696 v3 processors operating at 2.30 GHz, offering 36 cores and 72 threads. The system is conFig.d with 64 GB of DDR4 RAM and powered by an ASUS TUF 1200 W Gold ATX 3.0 to ensure stability and efficiency during intensive computational workload. DL computations are accelerated using an NVIDIA GeForce RTX 3090 XC3 Ultra Hybrid GPU with 24 GB GDDR6X memory and 10,496 CUDA cores. The experimental analysis evaluates phishing URL detection performance of various ML and DL models, including GB, AB, CB, XGB, and MLP, comprehensively. Additionally, KDR, an ensemble method integrating KNN, DT, and RF, is examined for its potential to enhance classification performance. Moreover, we incorporate an advanced HET model to further improve detection capabilities by leveraging DL-based feature extraction and representation.

4.1 Real-time phishing URL detection for browser extensions

Fig. 5 depicts the implementation of a phishing URL blacklist containing 5,000 entries, sourced from Cisco Systems, Inc. [39]. This blacklist is embedded in the local storage of the browser and is deployed automatically upon installation of the browser extension. This design facilitates efficient phishing detection because the stored blacklist can be accessed quickly without requiring constant server queries. Consequently, the system enhances URL lookup performance, reduces server load, and achieves a high detection accuracy for real-time phishing mitigation.

Fig. 6 illustrates the real-time phishing detection process in a Chrome browser environment. When a user accesses a URL, the system first employs the Levenshtein distance algorithm to evaluate its similarity to 5,000 blacklisted URLs stored in the local storage of the browser. A real-time warning is immediately displayed to the user if an exact match is found or if the similarity score exceeds a predefined threshold. Conversely, if the URL is not identified in the blacklist, it is further analyzed using the proposed HET algorithm to determine its phishing status before delivering the final detection result.

Fig. 7 depicts the execution times of the Levenshtein distance algorithm. The first URL (https://account.venmo.com/u/Roland-Richard) is processed in approximately 4.43 ms, and the second (http://dominikcaco.sk/control/panel/index.php) in 6.84 ms. These results highlight the minimal processing overhead of the algorithm and its suitability for real-time applications, ensuring high

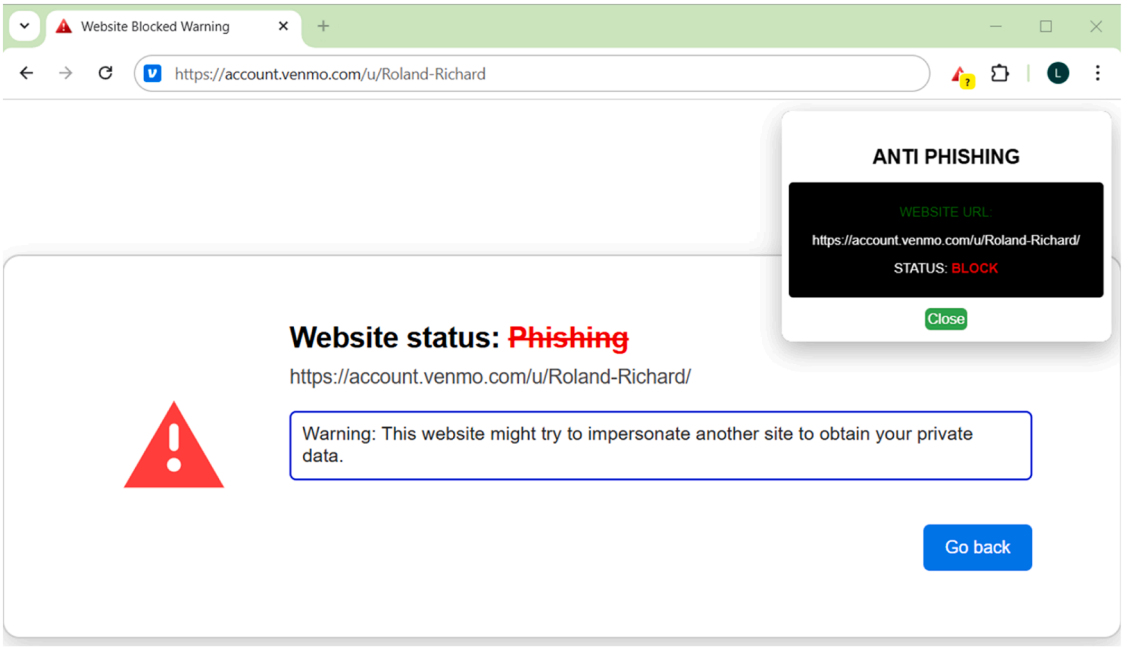

Fig. 6. Phishing warning notification displayed to users.

performance in verifying URLs against locally stored blacklists.

Fig. 8 illustrates the URL storage mechanism following detection. When a phishing URL is identified either via the Levenshtein distance algorithm or the HET-based DL model, the system promptly notifies the user and updates the local blacklist within the browser. This mechanism enables rapid retrieval and continuous expansion of the blacklist, thereby enhancing its phishing detection capability. Additionally, the detected URLs are integrated seamlessly into the blacklist to enable future real-time verification, allowing the system to adapt to emerging threats and improve overall security and responsiveness.

4.2. Evaluation under different data splits and metrics

To ensure a fair and representative comparison, the baseline models of classical machine learning and neural architectures that are commonly used in phishing URL detection were selected. Among machine learning models, GB, AB, CB, and XGB were selected as they represent strong tree-based ensemble methods widely adopted in cybersecurity tasks and consistently reported as competitive baselines in URL-based threat classification. For neural baselines, MLP and KDR were included to capture nonlinear feature interactions and provide lightweight deep-learning counterparts comparable in complexity to the proposed HET model. Methods such as SVM and RF were not included because they either underperform substantially on high-dimensional tokenized URL features or do not scale efficiently under real-time inference constraints, making them less suitable benchmarks for the deployment-oriented scenario examined in this study.

Table 6 reports the ablation results obtained by removing the key components of the proposed HET architecture. Across all four variants, performance differences remain extremely small ($\leq$ 0.0002 in accuracy and $\leq$ 0.0003 in F1-score), indicating that the PhiUSIIL Dataset is highly separable and that the core Transformer encoder already captures most discriminative URL patterns.

Fig. 9 shows the loss curves of the four HET variants. The full HET model achieved one of the highest accuracies (0.9978) and the highest recall (0.9999), confirming that the combination of bi-gram encoding, synthetic obfuscation samples, and dynamic masking improves phishing-sensitive feature extraction. The corresponding loss curve (Fig. 9(a)) shows stable convergence with a smoothly decreasing training loss and a consistently low test loss across epochs. Removing synthetic samples ("HET_no_synth") yields a slightly higher F1-score (0.9981), but at the cost of reduced robustness to obfuscation-based attacks. As shown in Fig. 9(b), the test loss exhibits mild fluctuations, indicating reduced regularization when adversarial-style samples are absent.

Disabling the masking mechanism ("HET_no_mask") slightly decreases recall and increases loss, indicating that masking provides a mild regularization effect by emphasizing phishing-indicative tokens. Its loss curve (Fig. 9(c)) displays higher test-loss variance than the full model, consistent with the removal of token-level attention suppression. As expected, the character-level tokenizer ("HET_char_level") performs worst across all metrics, validating the effectiveness of bi-grams in modeling short-range URL structures. The corresponding loss curve (Fig. 9(d)) shows slower convergence and the highest oscillation in test loss, further supporting the benefit of bi-gram locality modeling. Overall, these results demonstrate that each HET component contributes complementary benefits, and the full model maintains strong generalization and robustness across all evaluation metrics.

In summary, the ablation analysis highlights the complementary role of bi-gram encoding, synthetic obfuscation, and dynamic masking, with each component improving a specific aspect of URL discriminability and robustness. Their combined effect explains why the full HET variant achieves the most stable and generalizable performance.

Furthermore, the extremely small variations across all four variants ($\leq$ 0.0002 in accuracy and $\leq$ 0.0003 in F1-score) demonstrate

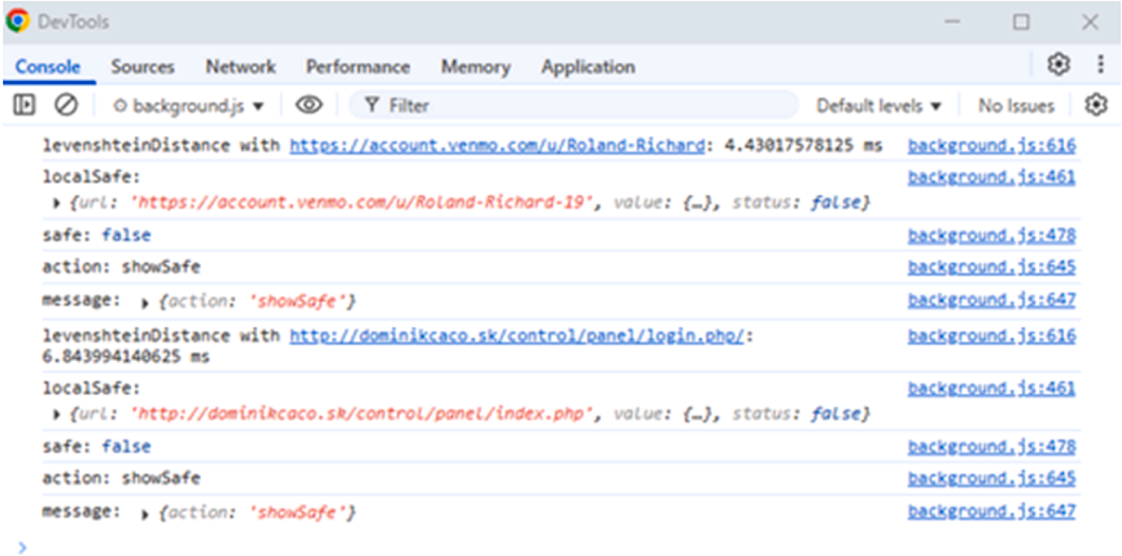

Fig. 7. Analysis of execution time of the Levenshtein distance-based algorithm.

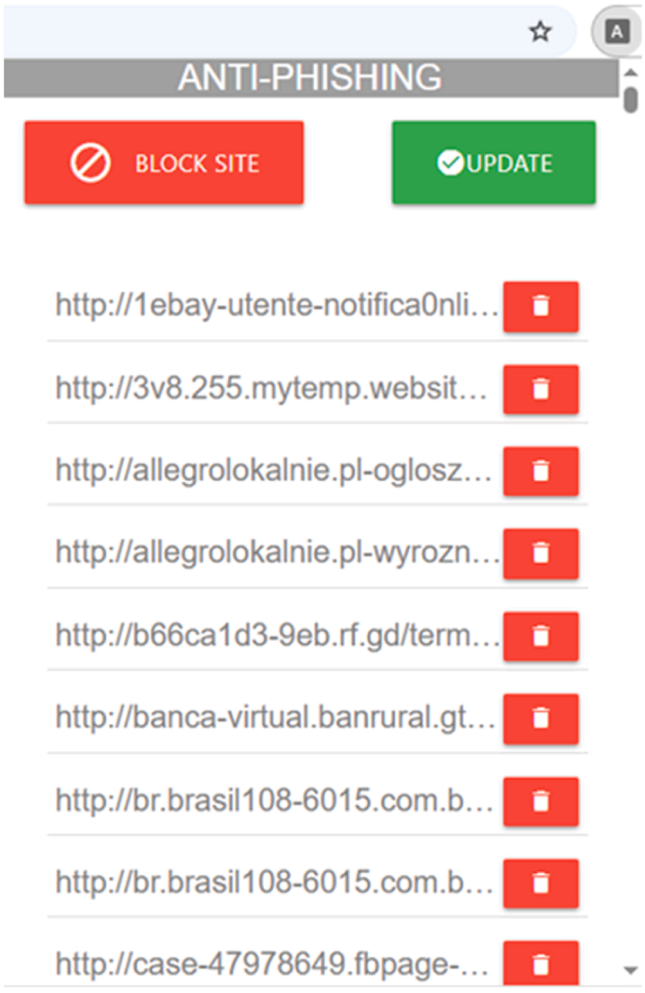

Fig. 8. Post-detection storage mechanism for phishing URLs.

Table 6
Ablation study results for the HET model across four architectural variants.

Variant	Loss	Accuracy	Precision	Recall	F1-score	Total params
HET_full	0.0167	0.9978	0.9962	0.9999	0.998	199,874
HET_no_synth	0.0163	0.9978	0.9965	0.9997	0.9981	199,682
HET_no_mask	0.018	0.9976	0.9965	0.9994	0.9979	199,938
HET_char_level	0.0173	0.9971	0.9949	0.9999	0.9974	42,818

the high stability of the training process and reproducibility of the experimental setup. This consistency indicates that the transformer backbone is inherently robust under different tokenization, masking, and augmentation choices, and that the observed differences can be reliably attributed to the controlled removal of components rather than stochastic effects.

In the ablation study, the four variants of the HET architecture—HET_full, HET_no_synth, HET_no_mask, and HET_char_level—were trained under identical experimental conditions to ensure a fair comparison. The PhiUSIIL Dataset was split into training and testing subsets using a 0.2 test ratio and a fixed random state of 42. Each variant used a maximum sequence length of 200, an embedding dimension of 64, four attention heads, and a feed-forward dimension of 64, and was optimized with Adam optimizer (learning rate 1×10^{-3}) for up to 15 epochs with a batch size of 32. Bi-gram tokenization was applied for all variants except HET_char_level, which used character-level encoding. Synthetic URLs generated by inserting the obfuscation token ".." were included in all variants except HET_no_synth, and dynamic masking based on the same token was disabled only in HET_no_mask. Through these controlled modifications, we were able to isolate the impact of each component, thereby enabling the identification of the functionalities driving the performance variations among the variants.

This section presents a comprehensive performance evaluation of various ML algorithms, i.e., GB, AB, CB, XGB, MLP, and KDR, as

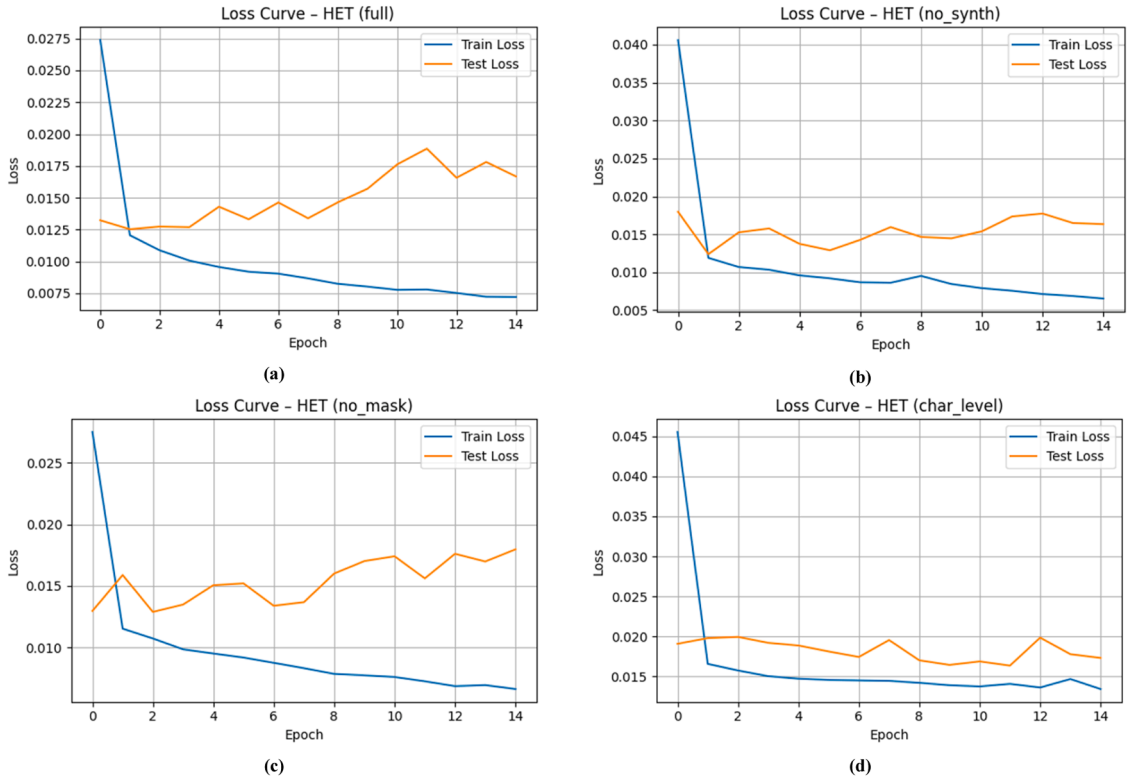

Fig. 9. Loss curves for four HET variants. (a) Loss curve for the full HET model (HET_full); (b) Loss curve for HET without synthetic augmentation (HET_no_synth); (c). Loss curve for HET without dynamic masking (HET_no_mask); (d). Loss curve for character-level tokenization (HET_char_level).

well as the proposed HET model, on the PhiUSIIL dataset [73], which comprises 235,795 URLs (134,850 benign and 100,945 phishing). Experiments are conducted with respect to varying data split ratios ranging from 4:6 to 9:1 to assess the stability and generalizability of each model. Each algorithm is evaluated in terms of key performance metrics, including accuracy, precision, recall, F1-score, AUC, and training and testing times. Additional metrics, such as FPs and FNs, are analyzed to gain a more comprehensive assessment.

Table 7 presents a comparative analysis of model accuracy with respect to varying training-to-testing ratios. GB and HET achieve remarkably high training accuracies (approximately 0.9999 and 0.9989–0.9994, respectively), reflecting strong learning capacities. However, AB exhibits relatively low test accuracy (0.9921–0.9931), possibly owing to its sensitivity to imbalanced or noisy data. MLP and KDR yield consistent test accuracies but underperform compared to boosting-based models, such as GB, CB, and XGB. HET exhibits the best generalizability, achieving the highest test accuracy (0.9971–0.9979). As the training proportion is increased from 4:6 to 9:1, test accuracy of the boosting models declines slightly, possibly due to overfitting. However, HET maintains high accuracy, underscoring its effectiveness in feature representation learning.

Table 8 highlights model performances in terms of precision corresponding to different training-testing ratios. HET consistently achieves the highest testing precision (0.9963–0.9967), demonstrating excellent phishing-benign discrimination with minimal misclassification. On the other hand, GB exhibits near-perfect training precision (0.9997–0.9998) but lower testing precision (0.9924–0.9932), indicating a tendency to overfit. Moreover, AB exhibits the lowest precision (0.9867–0.9884), likely because of its limited ability to capture phishing-specific patterns or handle class imbalance. Although MLP and KDR exhibit stable precision, they underperform compared to boosting-based models.

Tables 9 and 10 present the F1-scores corresponding to varying training-testing ratios, confirming the superiority of HET. HET exhibits the highest F1-scores (0.9974–0.9981), indicating its capacity to balance precision and recall while minimizing the misclassification of malicious URLs. As the training ratio is increased from 4:6 to 9:1, most models exhibit a slight decrease in the testing F1-scores. However, HET outperforms the others, with its accuracy peaking at 0.9981, underscoring its robust generalization capability even when trained on larger proportions of the dataset. Conversely, AB records the lowest F1-scores (0.9932–0.9940), indicating limited representational capacity.

Table 11 presents a comparative analysis of the training and testing times of the eight ML and DL models corresponding to varying train–test ratios (from 4:6 to 9:1). MLP and KDR exhibit significant increases in training time as the training proportion is increased—with MLP rising from 15,308.7 to 30,617.4 s and KDR from 4,272.01 to 11,318.4 s. On the contrary, the proposed HET model exhibits a more gradual and significantly lower increase in training time, from 606.2 to 1,339.1 s. Notably, HET maintains consistently low and stable testing times, ranging from 103.4 to 113.4 ms, which is significantly lower than that of KDR (406.8 to 2,322.1 s). These

Table 7
Comparison of various models in terms of accuracy corresponding to various training–testing ratios.

Tr–Te Ratio	GB		AB		CB		XGB		MLP		KDR		HET	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
4:6	0.9999	0.9951	0.9931	0.9931	0.9947	0.9944	0.9949	0.9948	0.9998	0.9948	0.9973	0.9961	0.9994	0.9971
5:5	0.9998	0.9953	0.9931	0.9931	0.9943	0.9943	0.9952	0.9950	0.9998	0.9946	0.9971	0.9964	0.9990	0.9974
6:4	0.9998	0.9952	0.9931	0.9930	0.9946	0.9944	0.9952	0.9951	0.9997	0.9944	0.9975	0.9962	0.9992	0.9975
7:3	0.9998	0.9950	0.9932	0.9928	0.9947	0.9943	0.9953	0.9950	0.9997	0.9943	0.9974	0.9962	0.9988	0.9977
8:2	0.9997	0.9954	0.9932	0.9924	0.9948	0.9940	0.9955	0.9946	0.9997	0.9949	0.9975	0.9958	0.9989	0.9975
9:1	0.9997	0.9949	0.9932	0.9921	0.9949	0.9938	0.9955	0.9944	0.9997	0.9940	0.9977	0.9956	0.9989	0.9979

Table 8
Comparison of various models in terms of precision corresponding to various training–testing ratios.

Tr–Te Ratio	GB		AB		CB		XGB		MLP		KDR		HET	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
4:6	0.9998	0.9930	0.9884	0.9883	0.9911	0.9907	0.9916	0.9917	0.9997	0.9920	0.9956	0.9939	0.9989	0.9963
5:5	0.9998	0.9932	0.9883	0.9884	0.9904	0.9904	0.9921	0.9921	0.9997	0.9918	0.9954	0.9942	0.9985	0.9966
6:4	0.9998	0.9932	0.9884	0.9882	0.9909	0.9906	0.9921	0.9920	0.9998	0.9930	0.9961	0.9944	0.9986	0.9966
7:3	0.9998	0.9932	0.9885	0.9879	0.9911	0.9905	0.9922	0.9921	0.9998	0.9934	0.9958	0.9944	0.9980	0.9965
8:2	0.9997	0.9928	0.9886	0.9872	0.9913	0.9900	0.9925	0.9912	0.9996	0.9925	0.9960	0.9938	0.9982	0.9967
9:1	0.9997	0.9924	0.9885	0.9867	0.9915	0.9895	0.9926	0.9908	0.9997	0.9918	0.9964	0.9931	0.9980	0.9966

Table 9
Comparison of various models in terms of F1-score corresponding to various training–testing ratios.

Tr–Te Ratio	GB		AB		CB		XGB		MLP		KDR		HET	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
4:6	0.9999	0.9958	0.9940	0.9940	0.9954	0.9951	0.9956	0.9955	0.9998	0.9955	0.9977	0.9966	0.9994	0.9974
5:5	0.9999	0.9959	0.9940	0.9940	0.9950	0.9950	0.9958	0.9957	0.9998	0.9953	0.9975	0.9968	0.9991	0.9977
6:4	0.9998	0.9958	0.9940	0.9939	0.9953	0.9951	0.9958	0.9957	0.9998	0.9951	0.9978	0.9967	0.9992	0.9978
7:3	0.9998	0.9957	0.9941	0.9938	0.9954	0.9950	0.9959	0.9957	0.9997	0.9951	0.9977	0.9968	0.9990	0.9979
8:2	0.9997	0.9960	0.9941	0.9934	0.9955	0.9948	0.9960	0.9953	0.9997	0.9955	0.9978	0.9963	0.9990	0.9977
9:1	0.9997	0.9956	0.9941	0.9932	0.9955	0.9946	0.9961	0.9951	0.9997	0.9948	0.9980	0.9961	0.9990	0.9981

Table 10
Comparison of various models in terms of recall corresponding to various training–testing ratios.

Tr–Te Ratio	GB		AB		CB		XGB		MLP		KDR		HET	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
4:6	1	0.9986	0.9997	0.9997	0.9997	0.9995	0.9996	0.9993	0.9999	0.9990	0.9997	0.9994	0.9999	0.9986
5:5	1	0.9986	0.9997	0.9997	0.9996	0.9997	0.9995	0.9992	0.9999	0.9987	0.9996	0.9995	0.9997	0.9989
6:4	0.9999	0.9984	0.9997	0.9997	0.9997	0.9997	0.9996	0.9994	0.9998	0.9973	0.9996	0.9989	0.9999	0.9990
7:3	0.9999	0.9982	0.9997	0.9997	0.9997	0.9996	0.9996	0.9994	0.9997	0.9967	0.9996	0.9991	0.9999	0.9993
8:2	0.9997	0.9991	0.9997	0.9996	0.9997	0.9997	0.9996	0.9993	0.9998	0.9986	0.9996	0.9988	0.9998	0.9988
9:1	0.9998	0.9988	0.9997	0.9997	0.9997	0.9997	0.9996	0.9995	0.9997	0.9977	0.9996	0.9992	1	0.9996

results suggest that HET preserves the computational efficiency as the training data volume increases, supporting real-time inference. Owing to its balance between training cost and minimal inference latency, HET stands out as a strong candidate for applications that require fast and accurate detection.

Table 12 presents the comparative loss values of seven machine learning and deep learning models across different training–testing ratios ranging from 4:6 to 9:1. As the training proportion increases, GB, AB, CB, and XGB exhibit only marginal variations in loss, whereas neural models MLP and KDR show greater sensitivity—MLP's testing loss fluctuates between 0.18 and 0.21, whereas KDR's loss ranges from 0.13 to 0.16. In contrast, the proposed HET model consistently achieves the lowest training and testing losses, with values gradually declining from 0.00191 to 0.00513 (training) and 0.02633 to 0.01817 (testing) as the training ratio increases, indicating both efficient learning and strong generalization.

4.3 Comparison of algorithms in terms of FP and FN rates

As depicted in Fig. 10, HET outperforms traditional models in terms of several metrics—it achieves the highest TN count (31,601), minimizing the misclassification of benign URLs; the highest TP count (40,473), maximizing phishing detection; the lowest FP count (140), reducing erroneous alerts and improving practical applicability; and a low FN count (25), minimizing undetected phishing cases. The effectiveness of HET stems from the integration of advanced techniques, including n-gram-based tokenization, which captures lexical patterns rather than isolated characters; a hybrid embedding approach that combines token, positional, and special embeddings to improve contextual URL understanding; adaptive attention masking, which emphasizes critical URL components while suppressing noise; Levenshtein distance, which aids in the detection of obfuscated phishing variations; and synthetic data augmentation using an additional 5,000 URLs, enhancing model robustness against adversarial attacks. HET enhances robustness by generating diverse phishing variants using character modifications (ph1shing.com, phishingg.com), special token insertions (www.bank-login.com → www.bank..login.com), and altered subdomain structures (secure.bank.com → bank.secure-login.com). Further, the model is optimized computationally for real-time deployment, with complexities of $O(n)$ for tokenization, $O(n^2)$ for attention, and $O(h \cdot d_k \cdot n^2)$ for the forward pass of the transformer. This balance between performance and efficiency reinforces the suitability of HET for real-world phishing URL detection systems.

Fig. 11 further compares FP and FN rates of the eight classification models corresponding to a 7:3 training-to-testing ratio, highlighting the effectiveness of the proposed HET method. HET achieves the lowest total misclassification rate, with an FP count of 140 and an FN count of 25, outperforming the other models significantly. These observations validate the robustness and reliability of the proposed model for phishing URL detection, particularly in minimizing false alarms and undetected threats—two critical factors in real-world cybersecurity applications. Among the boosting-based models, AB exhibits the highest FP count (493) despite a low FN count (12), indicating a tendency to misclassify benign URLs. CB and XGB maintain lower FN rates but exhibit relatively high FP counts, reflecting performance instability. MLP exhibits the highest FN count (132), indicating reduced capability to detect phishing URLs, likely due to overfitting. The ensemble model (KNN + DT + RF) performs better than some boosting models but exhibits high FP (227) and FN (35) counts. In contrast, the proposed HET model reduces the FP rate by 0.441 %, maintained an FN rate of only 0.0617 %, and achieves an average AUC of 99.7354 % corresponding to various test ratios.

4.4. Comparison of average AUC across data splits

The box plots in Fig.s 12 and 13 indicate that the proposed HET model consistently achieves high AUC scores corresponding to all training–testing ratios, indicating high accuracy and stability. Unlike other ML models, HET exhibits minimal performance fluctuation, ensuring robust phishing URL detection, even corresponding to varying data distributions. This stability, further supported by the results presented in Tables 13 and 14, is crucial in real-world applications, where data imbalance commonly degrades detection performance. A detailed performance comparison highlights the advantages of the proposed approach. Specifically, HET achieves an average AUC of 0.998951 (99.8951 %) during training and 0.997354 (99.7354 %) during testing, with lower variance than competing models, indicating strong generalizability. In contrast, traditional boosting-based methods, such as AB, exhibit lower AUC scores ($\sim$ 0.992), suggesting a higher susceptibility to performance degradation under data distribution shifts. Although GB and CB exhibit relatively high AUCs, their greater variance indicates lower stability under different training conditions.

Although MLP achieves high AUC scores, its low dispersion suggests overfitting, i.e., it performs well on the training dataset but lacks generalizability. In contrast, HET maintains high accuracy and consistency corresponding to all training–testing ratios. These observations agree with recent research on phishing detection.

4.5 Post-hoc k-NN inference using latent embedding similarity

To demonstrate the practical effectiveness of the proposed post-hoc similarity inference strategy, we conduct a qualitative analysis over a curated subset of both phishing and benign URLs, focusing on challenging cases, such as homograph attacks, typosquatting variants, and fake governmental domains.

In this experiment, each input URL is first tokenized and converted into an embedding vector using the trained HET model. Embedding similarity is computed against a reference database of labeled URL embeddings, and the top five nearest neighbors are selected to determine the predicted label via majority voting.

Table 15 presents representative examples from the testing set, highlighting the model's ability to identify visually deceptive phishing URLs correctly (e.g., goog1e.com, paypa1.com, dichvucong.ggo.com) while avoiding FPs on legitimate domains (e.g., paypal.

Table 11
Comparison of various models in terms of training and testing times corresponding to various training–testing ratios.

Tr–Te Ratio	GB		AB		CB		XGB		MLP		KDR		HET	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
4:6	1861.43	7.064	86.790	1.192	620.56	8.839	14.5041	3.789	15308.7	1.925	4272.01	2322.06	606.218	113.372
5:5	2767.87	7.035	130.74	1.082	685.27	7.794	14.579	3.708	16279.0	1.607	5541.88	1931.28	746.350	111.129
6:4	3816.73	7.033	182.18	1.057	758.01	6.036	15.0479	3.666	19253.9	1.603	7096.12	1566.83	872.348	111.695
7:3	5041.21	6.943	242.05	1.046	856.12	5.819	15.3658	3.652	23976.9	1.578	8631.30	1196.08	1040.82	108.905
8:2	6481.91	6.77	309.02	1.008	918.78	5.353	15.8295	3.617	26443.8	1.545	10126.3	827.833	1170.10	111.380
9:1	8037.10	6.749	384.75	0.991	973.72	5.213	15.8995	3.602	30617.4	1.518	11318.4	406.772	1339.12	103.379

Table 12
Comparison of various models in terms of loss corresponding to various training–testing ratios.

Tr–Te Ratio	GB		AB		CB		XGB		MLP		KDR		HET	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
4:6	0.18840	0.17324	0.24648	0.24865	0.18840	0.20024	0.18152	0.18521	0.00535	0.18445	0.09439	0.13706	0.00191	0.02633
5:5	0.20452	0.16875	0.24855	0.24702	0.20452	0.20299	0.17181	0.17701	0.00642	0.19352	0.10211	0.12901	0.00461	0.01727
6:4	0.19387	0.17273	0.24534	0.25145	0.19387	0.20062	0.17069	0.17578	0.00738	0.19833	0.08713	0.13604	0.00290	0.02020
7:3	0.18911	0.17731	0.24370	0.25731	0.18911	0.20534	0.16705	0.17833	0.00851	0.20330	0.09258	0.13349	0.00466	0.02004
8:2	0.18476	0.16508	0.24151	0.27285	0.18476	0.21323	0.16184	0.19413	0.00936	0.18266	0.08904	0.14980	0.00479	0.01765
9:1	0.18224	0.18189	0.24406	0.28125	0.18224	0.22317	0.16016	0.20024	0.01070	0.21399	0.08152	0.15744	0.00513	0.01817

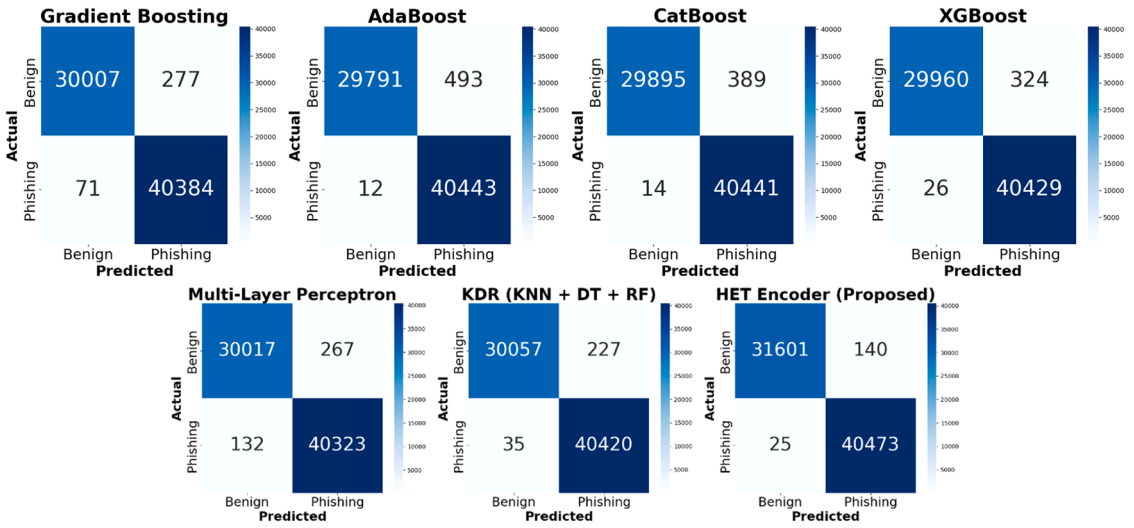

Fig. 10. Evaluation of the numbers of TPs, TNs, FPs, and FNs for all algorithms corresponding to a training-to-testing data split ratio of 7:3.

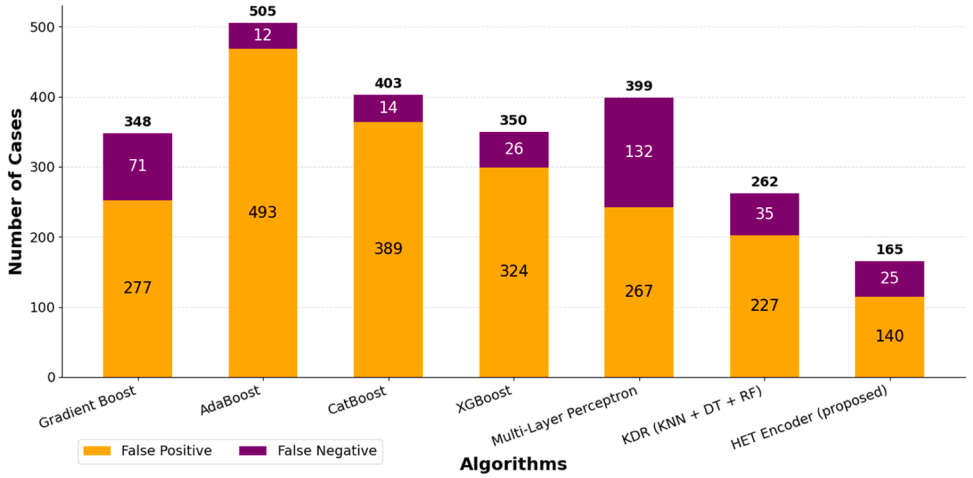

Fig. 11. Comparison of the numbers of FPs and FNs of various algorithms corresponding to a training-to-testing data split ratio of 7:3.

com, bankofamerica.com). Inference latency per URL ranges between 26 and 52 ms, confirming the method's real-time applicability.

Table 16 reports the effect of varying the neighborhood size k on the post-hoc embedding-based k-NN refinement module. Although the precision, recall, and F1-score remain in the moderate range (0.50–0.67), this behavior is expected because the refinement module is not designed to operate as a standalone classifier. Its decision is derived solely from local embedding similarity, and the small number of phishing samples in the test split further amplifies metric fluctuations. Instead, the module serves as a lightweight, interpretable, and retraining-free adjustment layer that complements, rather than replaces, the primary HET classifier.

Across $k = 3$ and $k = 5$, the accuracy remains stable at 88.23%, which aligns with the local-density formulation in Eq. (8)—smaller neighborhoods preserve cluster purity, while larger neighborhoods introduce samples from adjacent regions in the embedding space. The latency characteristics follow the complexity trends predicted by Eq. (9), where increasing k reduces the cost of partial sorting and leads to faster inference.

For this post-hoc refinement module, the key performance indicators include not only the classification metrics but also are dependent on (i) behavioral stability across k, (ii) latency efficiency, and (iii) interpretability through neighbor-agreement consistency. These results confirm that the module fulfills its intended purposes of enabling incremental, retraining-free inference and providing low-overhead similarity-based reasoning for previously unseen URLs.

4.6. Discussion

This study presents a novel and adaptive multi-tier framework for phishing URL detection, combining local heuristic filtering

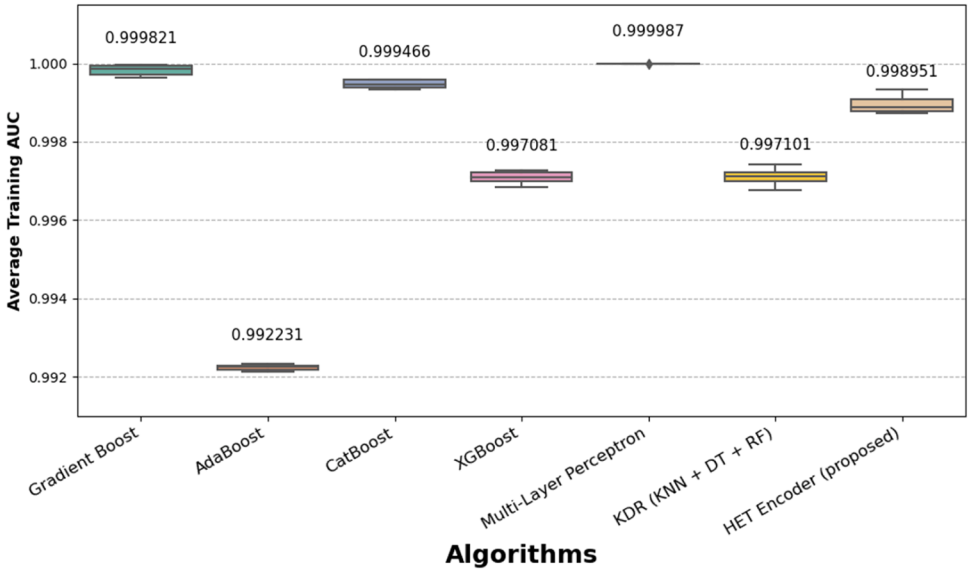

Fig. 12. AUC score distribution corresponding to varying training ratios.

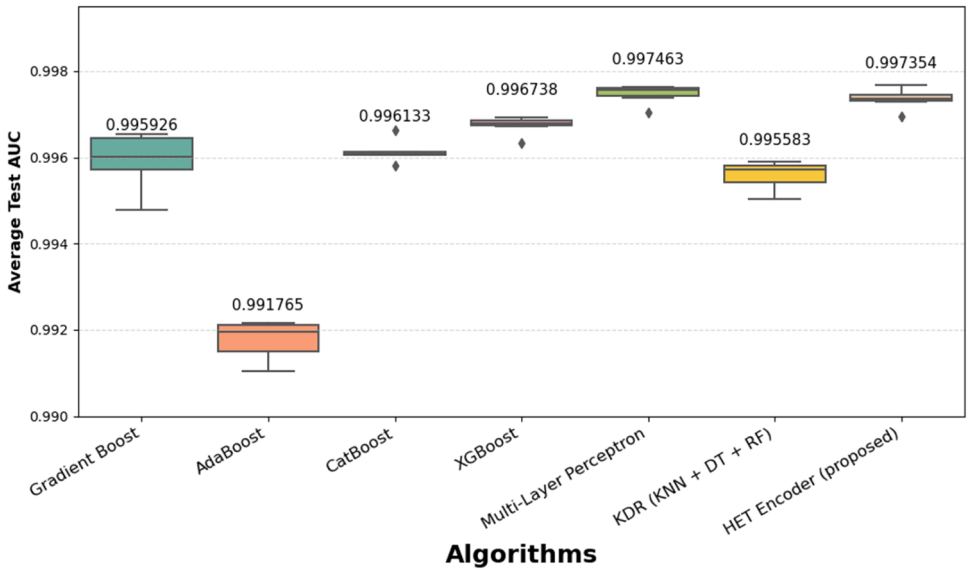

Fig. 13. AUC score distribution corresponding to varying testing ratios.

Table 13
Comparison of various models in terms of the AUC score corresponding to various training ratios.

Tr–Te Ratio	GB	AB	CB	XGB	MLP	KDR	HET
4:6	0.999955822	0.992167941	0.999373878	0.996850926	0.999989	0.996975651	0.999334093
5:5	0.999941455	0.992128434	0.999595278	0.996955665	0.999987	0.996751015	0.998920105
6:4	0.99988219	0.992218364	0.999596198	0.997046029	0.999991	0.997230646	0.999123735
7:3	0.999833988	0.992268258	0.999522158	0.997151705	0.999988	0.997054913	0.998733989
8:2	0.999634606	0.992336777	0.999323261	0.997225677	0.99998	0.99717392	0.998829992
9:1	0.999675527	0.992268527	0.999384951	0.997258996	0.999986	0.997421936	0.998766529

Table 14
Comparison of various models in terms of the AUC score corresponding to various testing ratios.

Tr-Te Ratio	GB	AB	CB	XGB	MLP	KDR	HET
4:6	0.996504169	0.992129542	0.996061515	0.996723657	0.997520592	0.995652	0.996948409
5:5	0.996533496	0.992161397	0.996120854	0.996795456	0.997624243	0.995902	0.997297355
6:4	0.996242601	0.992034727	0.996077433	0.996928408	0.997602809	0.995766	0.997376583
7:3	0.995702382	0.99185711	0.996616845	0.996883265	0.997608184	0.99582	0.997485993
8:2	0.995797077	0.991377393	0.996106422	0.99676371	0.997381614	0.995333	0.99732778
9:1	0.994777228	0.991032481	0.995813002	0.996335991	0.997038263	0.995023	0.997685723

Table 15
Prediction results in terms of post-hoc similarity inference.

URL	Ground-truth (0 = phishing, 1 = benign)	Predicted label	Inference time (ms)	Notes
https://www.google....com	0	phishing	33	Homograph
https://www.paypal..com	0	phishing	27	Modified variant
https://www.bankofamerica.com	1	benign	43	Legitimate
https://www.goog1e..com/	0	phishing	37	Lookalike
https://www.paypa1.com/vn/home	0	phishing	37	Variant of PayPal
https://viet.lgovn.cc	0	phishing	44	Fake gov domain
https://vietvp.cc	1	benign	26	Non-phishing short domain
https://dichvucong.hgov.net	0	phishing	43	Fake public service domain
https://dichvucong-gov.com	0	phishing	31	Another .gov mimic
https://dichvucong.agovn.net	0	phishing	37	Fake gov domain
https://dichvucong.dancuquocgia.net	0	phishing	31	Fake gov domain
https://dichvucong.kgov.net	0	phishing	52	Another .gov mimic
https://dichvucong.lgov.net	0	phishing	51	Another .gov mimic
https://dichvucong.vgovn.net	0	Phishing	33	Fake gov domain
https://dichvucong.ggo.com	0	Phishing	33	Homograph
https://www.paypal.com	1	benign	35	Original PayPal domain

These observations further validate the robustness and practical readiness of the post-hoc inference mechanism. Across diverse adversarial URL structures, the proposed model maintains high classification accuracy and low latency, making it suitable for real-time deployment in modern browser or security gateway environments.

Table 16
Effect of the k parameter on embedding-based post-hoc inference performance.

k	Accuracy (%)	Precision	Recall	F1-score	Mean latency (ms)	Median latency (ms)	p95 latency (ms)
3	88.235294	0.666667	0.666667	0.666667	130.499489	133.822680	156.940174
5	88.235294	0.666667	0.666667	0.666667	64.689370	63.472509	73.023462
7	82.352941	0.500000	0.666667	0.571429	67.362631	63.940763	80.486584
9	82.352941	0.500000	0.666667	0.571429	62.205483	61.534166	67.804289

techniques with a sophisticated HET model that incorporates dynamic attention mechanisms alongside a lightweight post-hoc inference scheme. Unlike many existing methods that focus primarily on model accuracy or isolated components, the proposed framework strikes an optimal balance between scalability, real-time efficiency, and adaptive robustness—all essential factors for practical deployment in dynamic, large-scale environments.

Comprehensive evaluation on a large and diverse benchmark dataset demonstrates that the proposed method exhibits superior accuracy, robustness against obfuscated and novel phishing attacks, and greater computational efficiency than current state-of-the-art alternatives.

The main novel contributions if this study are presented below:

- Multi-level detection pipeline: A three-tiered architecture synergistically integrates (i) a dynamically updated local blacklist enabling swift exclusion of known threats, (ii) a Levenshtein distance-based similarity module adept at detecting subtle typos-quatting and URL manipulations, and (iii) a server-hosted HET model delivering fine-grained, context-aware classification of ambiguous or novel URLs. This hierarchical design balances processing speed and analytical depth, addressing critical research gaps in prior singular or non-adaptive frameworks.
- Innovative inference strategy: We pioneer the use of k-NN voting directly within the model's latent embedding space, removing the need for regular retraining. This reduces operational overhead and enhances adaptability to rapidly evolving phishing tactics. This lightweight post-hoc inference achieves real-time classification with latency ranging between 26 and 52 ms per URL, representing the pure server-side prediction time. This performance highlights the model's capacity to deliver rapid, adaptive inference suitable for responsive phishing detection in dynamic deployment environments (including local filtering, communication delays).

- Detailed inference profiling: Notably, the system supports real-time classification with an extremely low end-to-end latency of 4.43–6.84 ms per URL in actual browser environments. This latency measurement reflects the complete processing pipeline, where the Levenshtein distance computation plays a key part in assessing URL similarity against a locally stored URL blacklist of 5,000 entries. By maintaining this critical similarity index in local storage, the system achieves rapid filtering of known threats and typosquatting attempts directly on the client side, thereby reducing the need for remote queries or server-side computation for the majority of requests.
- Adaptive thresholding mechanism: The dynamic threshold adjustment technique allows the classification decision boundary to be self-tuned responsively as phishing patterns evolve. This improves precision and recall significantly by minimizing FPs and FNs, a crucial factor for sustained reliability in real-world, continuously shifting threat landscapes.
- Robust empirical performance: Evaluation on a large-scale, diverse benchmark dataset (~236,000 URLs) reveals that the HET model consistently achieves top-tier testing accuracy (99.7–99.8 %), precision and recall (> 99.6 %), and F1-scores (~ 99.8 %), outperforming classical ML baselines significantly. Importantly, it maintains the lowest FPR (as low as 0.14 %), striking an optimal balance critical for practical phishing detection.
- Real-world deployment and scalability: The proposed framework is successfully implemented as a lightweight Chrome browser extension integrated with the multi-tier backend architecture, ensuring both usability and scalability in real-world environments. The framework balances client-side filtering and server-side DL inference efficiently to deliver rapid, reliable phishing protection.
- Comprehensive evaluation and stability: The framework is tested corresponding to multiple training/testing split ratios, demonstrating strong generalizability and resistance to overfitting. It is observed to outperform popular boosting-based models and traditional ML approaches particularly in imbalanced and noisy data contexts.
- Furthermore, the ablation study provides additional insights into how each architectural component contributes to the effectiveness of the HET model. Removing synthetic obfuscation samples or disabling dynamic masking results in measurable increases in test loss and recall degradation, and eliminating bi-gram encoding causes the most significant decline in accuracy and precision. These results confirm that each component contributes a complementary role—while bi-grams preserve locality-sensitive URL structures, synthetic samples enhance robustness against adversarial obfuscation and masking stabilizes attention interactions. The extremely small performance variations across variants (≤ 0.0002 in accuracy and ≤ 0.0003 in F1-score) further demonstrate the stability and reproducibility of the proposed framework.

When compared with notable related works, such as Akçam et al. [76] which employed a BiLSTM model integrating character- and word-level features for phishing URL detection, achieving high performance across multiple benchmark datasets, or Liu et al. [77] (PMANet), which focused on sophisticated post-training of pre-trained transformers with multi-layer attention and spatial pyramid pooling, our approach emphasizes practical deployment aspects uniquely. The combination of the HET framework with a post-hoc k-NN inference mechanism enables fast classification without relying on retraining and flexible deployment via a browser-server hybrid architecture. Consequently, our approach demonstrates superior scalability and practical applicability in diverse, real-time environments compared to the aforementioned methods.

Moreover, although MADONNA [78] serves as a practical browser-based domain detection solution, its reliance on a shallow neural network architecture restricts adaptability and detection depth when compared to our transformer-based embedding-rich model.

This research addresses critical limitations in existing phishing URL detection systems, notably the high computational cost and latency associated with frequent retraining and fine-tuning of DL models when encountering novel phishing URLs. By proposing a novel multi-tier architecture combining rapid client-side heuristic filtering (local blacklist and Levenshtein similarity verification) with a server-side HET encoder and an innovative lightweight post-hoc k-NN inference strategy, this study overcomes the bottlenecks hindering real-time, large-scale deployment effectively.

Although the proposed multi-tier framework demonstrates strong empirical performance, a dedicated adversarial robustness evaluation remains an important direction for further strengthening the system. The Levenshtein similarity module may be susceptible to homoglyph-based obfuscation techniques (e.g., Unicode confusables) that preserve visual similarity while altering edit distance, and the HET encoder may also be affected by targeted perturbations designed to manipulate the embedding space. Bi-gram encoding, synthetic obfuscation samples, and dynamic masking partially mitigate these risks; however, a formal adversarial evaluation suite is necessary to quantify robustness across all tiers. We therefore identify adversarial robustness assessment as a key priority for future work to ensure comprehensive resilience against determined attackers.

5. Conclusions

5.1 Theoretical contributions

This study advances the theoretical foundations of phishing URL detection research by introducing a novel multi-tier detection architecture that integrates rapid client-side heuristic filtering with a sophisticated server-side DL model. Specifically, the framework combines a locally maintained and dynamically updated blacklist, a Levenshtein distance-based similarity module for capturing typosquatting and subtle URL manipulations, and an HET model incorporating advanced NLP techniques, such as n-gram tokenization, synthetic data augmentation, multi-head self-attention, and dynamic attention masking.

A key theoretical innovation is the lightweight post-hoc inference mechanism implemented in the HET model, which classifies URL embeddings using k-NN voting in the latent space. This approach eliminates the need for complex parametric output layers and frequent retraining or fine-tuning, thereby addressing significant computational bottlenecks commonly observed in existing DL-based

phishing detectors. We evaluate the performances of traditional ML and DL models—including GB, AB, CB, XGB, MLP, and KDR—alongside the proposed HET method on the PhiUSIIL dataset, which contains 235,795 URLs (134,850 benign and 100,945 phishing). Experiments are conducted corresponding to varying training-testing split ratios (ranging from 4:6 to 9:1) to assess stability and generalizability using key evaluation metrics, such as accuracy, precision, recall, F1-score, AUC, and training and testing times. Empirical results on this large-scale benchmark dataset demonstrate the superior stability, precision, recall, and robustness of the HET method, particularly in the presence of data imbalance and real-world constraints, outperforming classical and ensemble baseline models.

Importantly, the ablation tests in this study further strengthen the theoretical contributions of this work by isolating the role of each architectural component within the HET framework. The analysis clearly demonstrates that bi-gram encoding captures short-range structural patterns in URLs, synthetic obfuscation samples enhance robustness against adversarial manipulations, and dynamic masking acts as an effective regularization mechanism for suppressing misleading token interactions. Their combined effect yields the most stable and generalizable configuration, thereby providing rigorous empirical justification for the architectural design choices. Moreover, the extremely small variance observed across all ablation variants confirms the inherent stability and reproducibility of the HET methodology, reinforcing its suitability for real-world deployment scenarios.

By elucidating the synergy between local heuristic filtering and transformer-based representation learning and non-parametric inference systematically, this study makes a substantial contribution to the design of scalable, adaptive, and robust DL frameworks for cybersecurity. In particular, it bridges the gap between theoretical innovation in model architectures and practical requirements for deployable, real-time phishing detection systems.

Although the study provides a complete empirical validation of the k-NN post-hoc inference mechanism, a formal theoretical analysis of its behavior under varying embedding-space densities and conditions that guarantee stable neighbor-based prediction remains beyond the scope of this study. Although the proposed explanation of local-density effects (Eq. 8) offers an intuitive interpretation, a more formal treatment of embedding topology, convergence properties, and robustness under "embedding drift'' can be realized in future theoretical works. These aspects are not trivial and typically require longitudinal datasets capturing evolving attack patterns, which extends beyond the objectives of this study. Nonetheless, the empirical results of this study consistently support the design rationale of the inference module and demonstrate its practical viability.

5.2. Practical implications

The proposed multi-tier phishing detection framework offers strong practical benefits for real-world deployment and cybersecurity operations. The client-side browser extension enables rapid filtering of known malicious URLs using a local blacklist comprising approximately 5,000 entries, complemented by real-time Levenshtein-based similarity checks that detect phishing URL variants efficiently without requiring server interaction. This architecture minimizes network overhead and accelerates the filtering workload handled locally.

Complex or ambiguous URLs are escalated to the server-side HET model, which executes adaptive, low-latency inferencing by incorporating embedding-based k-NN voting. The system achieves an end-to-end latency of 4.43–6.84 ms per URL in browser environments, reflecting user-perceived responsiveness, and an isolated model inference latency of 26–52 ms per URL on the server, confirming real-time applicability.

On the training dataset, the proposed method achieves an accuracy of 99.94 %, a precision of 99.80 %, an F1-score of 99.90 %, and an AUC of 99.89 %. On the test dataset, it attains an accuracy of 99.79 %, a precision of 99.65 %, an F1-score of 99.79 %, and an AUC of 99.73 %. The method maintains low error rates, with an FPR of 0.441 % and FNR of 0.0617 %, which directly fosters user trust and reduces alert fatigue. Further, the modular and dynamically updateable nature of the blacklist, combined with the adaptive thresholding mechanism, enables continuous maintenance and rapid adaptation to evolving phishing tactics. The framework's scalability and responsiveness satisfy the stringent requirements of edge and browser-side cybersecurity applications, where swift threat detection and mitigation enhance defense capabilities significantly. These results underscore the effectiveness of the proposed hybrid approach in delivering rapid, adaptive, and resource-efficient phishing detection. Collectively, this study's contributions reinforce cybersecurity defense strategies and advance the development of AI-driven security solutions, ensuring both scalability and real-world applicability.

Modern DL-based phishing URL detection systems commonly rely on retraining on the entire dataset upon encountering new, unseen URLs or on employing complex output layers combined with continuous fine-tuning to optimize model weights. These make them computationally intensive and time-consuming, posing significant challenges for deployment in real-time environments where phishing URLs evolve rapidly and continuously. To address these limitations, the proposed framework integrates a post-hoc inference mechanism that leverages the HET encoder alongside a k-NN voting scheme based on Euclidean similarity within the latent embedding space. This method enables efficient, retraining-free classification by embedding incoming URLs and comparing them to a pre-existing reference embedding set, thereby circumventing the need for frequent model updates or reliance on internal confidence metrics. Consequently, the framework provides a scalable and adaptive solution that satisfies the stringent demands of real-time phishing detection in dynamic operational contexts.

In addition, the ablation study provides concrete practical insights into how each architectural component of the HET model contributes to real-world detection effectiveness. Removing synthetic obfuscation samples ("HET_no_synth") slightly increased the F1-score (0.9981 vs. 0.9980) but reduced robustness when confronting manipulated URLs that frequently appear in operational environments. Disabling dynamic masking ("HET_no_mask") resulted in a measurable increase in test loss (0.0180 vs. 0.0167) and a decrease in recall (0.9994 vs. 0.9999), indicating less stable inference when misleading or noisy URL tokens are present. Eliminating

bi-gram encoding ("HET_char_level") caused the most significant practical degradation, lowering the accuracy from 0.9978 to 0.9971 and reducing precision from 0.9962 to 0.9949—an important consideration for deployment scenarios where false positives directly impact user trust and system usability.

These measured performance shifts, though small in absolute value, have clear operational implications. They confirm that bi-gram encoding is critical for preserving locality-sensitive URL patterns, synthetic samples improve resilience against adversarial obfuscation strategies, and dynamic masking reduces variance in real-time inference. The extremely low cross-variant metric variations (≤ 0.0002 in accuracy and ≤ 0.0003 in F1-score) also demonstrate that the framework remains stable and reproducible under different component configurations, enabling practitioners to choose lighter or heavier variants of HET based on available computational budgets or the threat landscape of a given environment.

5.3. Limitations and future research

Despite the excellent performance of the proposed multi-tier framework, several limitations must be acknowledged. The system has been experimentally validated only on the Chrome browser and has not yet been evaluated on alternative environments such as Firefox or mobile platforms. Furthermore, the current architecture does not support multilingual URL analysis, which limits its applicability to diverse phishing landscapes. Future work should include broadening platform compatibility, launching a community-based trial version to collect failure cases in the wild, and expanding multilingual capabilities. Another important direction is enhancing robustness against emerging phishing variants, including social engineering-driven methods such as voice phishing impersonation attacks, which have increased significantly in recent years.

Additionally, this study does not include a dedicated adversarial robustness evaluation, such as homoglyph-based obfuscation, gradient-driven embedding perturbations, or similarity-preserving adversarial URL transformations. A comprehensive adversarial assessment will be carried out in future research to further strengthen the framework's practical resilience.

Finally, this study does not include a comparison against fine-tuned transformer baselines, such as BERT or DistilBERT, which are widely used in phishing URL detection. Future extensions will benchmark the proposed HET architecture against these models under matched inference-latency constraints to provide a more comprehensive evaluation of accuracy, efficiency, and deployment feasibility.

Glossary

Accuracy	Ratio of correctly predicted URLs (phishing and benign) to the total number of predictions.
AdaBoost	A boosting algorithm that adds weak learners sequentially, focusing on incorrectly classified instances, to improve the overall model performance.
Area Under the Curve (AUC)	A metric that represents the ability of the model to distinguish between classes, calculated based on the Receiver Operating Characteristic (ROC) curve.
Autoencoder (AE)	A type of neural network used for unsupervised learning that aims to compress the input into a lower-dimensional representation and then reconstruct it.
Bagging	An ensemble learning method that trains multiple base learners on different data subsets and combines their predictions to reduce variance.
Blacklist-based Methods	Phishing detection techniques that rely on static lists of known malicious URLs. These require regular updates, and may fail to detect new threats.
Boosting	An ensemble learning strategy that constructs models sequentially, each correcting an error from the previous one, to improve the classification accuracy.
Browser Extension	Lightweight software components that add features to a web browser. In this study, real-time phishing detection is enabled using a browser extension.
Cross-Entropy Loss	A loss function used in classification problems to measure the distance between predicted probabilities and actual class labels.
Dynamic Attention Mask	A modified attention mechanism that applies variable weights to emphasize the critical components of the input data, thereby improving model focus.
False Negative (FN)	An instance in which a phishing URL is incorrectly classified as benign.
False Positive (FP)	An instance where a benign URL is incorrectly flagged as phishing.
Firebase Firestore	A cloud-based NoSQL database used for real-time storage and synchronization of the phishing URL data.
F1-Score	The harmonic mean of precision and recall used to evaluate models in which data are imbalanced or both metrics are important.
Generative Adversarial Network (GAN)	A deep learning model consisting of two competing networks—a generator and a discriminator—which is used for synthetic data generation.
Hybrid Embedding	A method that combines different token representations (e.g., n-grams and special tokens) to enhance model generalization and classification.
Levenshtein Distance	A metric that measures the minimum number of edits (insertions, deletions, or substitutions) required to transform one string into another.
Local Repository	A phishing URL database stored on the user's device, allowing offline checks and reducing detection latency.
Multi-head Attention	A component of transformer architectures that allows the model to attend to different parts of the input simultaneously using multiple attention mechanisms.
N-gram Tokenization	The process of breaking strings into overlapping substrings of length n that is useful for preserving contextual information during URL analysis.
Phishing URL	Fraudulent web addresses designed to deceive users and steal personal or financial information by mimicking legitimate sites.
Precision	Proportion of correctly identified phishing URLs among all URLs labeled as phishing by the model.
Real-time Processing	The ability of a system to analyze and classify URLs with minimal latency, which is crucial for in-browser phishing prevention.

(*continued on next page*)

(*continued*)

Recall	Proportion of actual phishing URLs correctly identified by the model.
Reinforcement Learning	A learning paradigm in which models learn to make sequences of decisions through trial-and-error interactions with the environment.
Server-side Verification	A secondary classification step, where URLs that are not found locally are transmitted to a server for deep learning-based analysis.
Stacking	An ensemble method in which outputs of multiple models are combined using a meta-classifier to improve predictive performance.
Threshold-based Classification	A flexible classification strategy in which decisions are based on whether the prediction scores transcend predefined thresholds.
Transductive LSTM (TLSTM)	A variant of long short-term memory networks tailored for semi-supervised learning, focusing on test-set-specific generalization.
Transformer Architecture	A neural network model that relies on self-attention to weigh the importance of different input parts for sequence-modeling tasks.
Voting	An ensemble approach, where multiple classifiers vote and the final prediction is determined by the majority decision.
Whitelist	A list of URLs explicitly deemed safe, used in contrast to blacklists to allow only trusted content through.

Data availability

Data used in this study are available upon request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

None

References

[1] Bustio-Martínez L, Herrera-Semenets V, García-Mendoza JL, Álvarez-Carmona MÁ, González-Ordiano JÁ, Zúñiga-Morales L, et al. Uncovering phishing attacks using principles of persuasion analysis. J Netw Comput Appl 2024;230:103964.
[2] Ejaz A, Mian AN, Manzoor S. Life-long phishing attack detection using continual learning. Sci Rep 2023;13:11488.
[3] Hussain M, Cheng C, Xu R, Afzal M. CNN-Fusion: an effective and lightweight phishing detection method based on multi-variant ConvNet. Inf Sci 2023;631:328–45.
[4] Pan S, University Tianjin, Kwak DH, et al. Roles of feedback and phishing characteristics in antiphishing training performance: perspectives of goal setting and skill acquisition. J Assoc Inf Syst 2024;25:1037–78.
[5] Minh Linh D, Hung HD, Minh Chau H, Sy Vu Q, Tran TN. Real-time phishing detection using deep learning methods by extensions. Int J Electr Comput Eng IJECE 2024;14:3021.
[6] Tang L, Mahmoud QH. A survey of machine learning-based solutions for phishing website detection. Mach Learn Knowl Extr 2021;3:672–94.
[7] Liu X, Ahmad SF, Anser MK, Ke J, Irshad M, Ul-Haq J, et al. Cyber security threats: A never-ending challenge for e-commerce. Front Psychol 2022;13:927398.
[8] Muhammad SS, Dey BL, Bala H, Alwi SFS, Asaad Y. A typology and model of privacy- and security-concerned users' Attitudes toward digital footprints and the consequent influence on their social Media adaptation. J Assoc Inf Syst. 2024;25:1240–73.
[9] Tu YJ, Zhou W, Piramuthu S. Critical risk considerations in auto-ID security: barcode vs. RFID. Decis Support Syst. 2021;142:113471.
[10] Nikkhah HR, Grover V. Strategizing responses to data breaches: A multi-method study of organizational responsibility and effective communication with stakeholders. J Manag Inf Syst 2024;41:1042–77.
[11] Khan S, Kabanov I, Hua Y, Madnick S. A systematic analysis of the Capital one data breach: critical lessons learned. ACM Trans Priv Secur 2023;26:1–29.
[12] Thomas L, Gondal I, Oseni T, Sally Firmin S. A framework for data privacy and security accountability in data breach communications. Comput Secur 2022;116:102657.
[13] Dambra S, Bilge L, Balzarotti D. A comparison of systemic and systematic risks of malware encounters in consumer and enterprise environments. ACM Trans Priv Secur 2023;26:1–30.
[14] Sharif M, Urakawa J, Christin N, Kubota A, Yamada A. Predicting impending exposure to malicious content from user behavior. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security; 2018. p. 1487–501. https://doi.org/10.1145/3243734.3243779.
[15] Bera D, Ogbanufe O, Kim DJ. Towards a thematic dimensional framework of online fraud: an exploration of fraudulent email attack tactics and intentions. Decis Support Syst 2023;171:113977.
[16] Ou CX, Zhang X, Angelopoulos S, Davison RM, Janse N. Security breaches and organization response strategy: exploring consumers' threat and coping appraisals. Int J Inf Manag 2022;65:102498.
[17] APWG. Anti-phishing Working Group. (2025), https://docs.apwg.org/reports/apwg_trends_report_q4_2024.pdf.
[18] Alani MM, Tawfik H. PhishNot: A cloud-based machine-learning approach to phishing URL detection. Comput Netw 2022;218:109407.
[19] Spithoven R, Drenth A. Who will take the bait? Using an embedded, experimental study to chart organization-specific phishing risk profiles and the effect of a voluntary microlearning among employees of a Dutch municipality. J Cybersecurity 2024;10:tyae010.
[20] Wright R, Johnson S, Kitchens B. Phishing Susceptibility in Context: A Multilevel Information Processing Perspective on Deception Detection, 47. MIS Q; 2023. p. 803–32.
[21] Iliou C, Kostoulas T, Tsikrika T, Katos V, Vrochidis S, Kompatsiaris I. Detection of advanced web bots by combining web logs with mouse behavioural biometrics. Digit Threats Res Pract 2021;2:1–26.
[22] Hanus B. Y. "Andy" Wu. Impact of users' Security awareness on desktop security behavior: A protection motivation theory perspective. Inf Syst Manag 2016;33:2–16.
[23] Jubur M, Shrestha P, Saxena N. An In-depth analysis of password managers and two-factor authentication tools. ACM Comput Surv 2025;57:1–32.

[24] Harsha B, Morton R, Blocki J, Springer J, Dark M. Bicycle attacks considered harmful: quantifying the damage of widespread password length leakage. Comput Secur 2021;100:102068.
[25] Shirvanian M, Agrawal S. 2D-2FA: A new dimension in two-factor authentication. In: Annual Computer Security Applications Conference; 2021. p. 482–96. https://doi.org/10.1145/3485832.3485910.
[26] Berrios J, Mosher E, Benzo S, Grajeda C, Baggili I. Factorizing 2FA: forensic analysis of two-factor authentication applications. Forensic Sci Int Digit Investig 2023;45:301569.
[27] Ng KC, Zhang X, Thong JYL, Tam KY. Protecting against threats to information security: an attitudinal ambivalence perspective. J Manag Inf Syst 2021;38:732–64.
[28] Kaspersky Lab. Spam and phishing: bypassing 2FA with phishing and OTP bots. (2024), https://securelist.com/2fa-phishing/112805/.
[29] Varshney G, Misra M, Atrey P. Secure authentication scheme to thwart RT MITM, CR MITM and malicious browser extension based phishing attacks. J Inf Secur Appl 2018;42:1–17.
[30] Schöni L, Kubicek K, Zimmermann V. Block cookies, not websites: analysing mental models and usability of the privacy-preserving browser extension CookieBlock. Proc Priv Enhancing Technol 2024;2024:192–216.
[31] Y. Rawoteea, G. Bekaroo. RXSS Protect: A browser extension for detection of reflected cross-site scripting attacks in real-time using machine learning. 2024 International conference on next generation computing applications (NextComp). (2024) p. 1–7, https://doi.org/10.1109/NextComp63004.2024.10779651.
[32] Prasad D, Suhasini S, Parthasarathy B, Praveen J. Enhancing internet security: A machine learning-based browser extension to prevent phishing attacks. In: 2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE); 2024. p. 896–900. https://doi.org/10.1109/IC3SE62002.2024.10593201.
[33] Wu CY, Kuo CC, Yang CS. Phishing detection with browser extension based on machine learning. In: 2023 18th Asia Joint Conference on Information Security (AsiaJCIS); 2023. p. 81–7. https://doi.org/10.1109/AsiaJCIS60284.2023.00023.
[34] Jaiswal O, Asare P, Gadgilwar J, Rahangdale K, Adekar P, Bitla L, et al. Malicious address identifier (MAI): A browser extension to identify Malicious URLs. In: 2023 11th International Conference on Emerging Trends in Engineering & Technology - Signal and Information Processing (ICETET - SIP); 2023. p. 1–5. https://doi.org/10.1109/ICETET-SIP58143.2023.10151582.
[35] Sun B, Akiyama M, Yagi T, Hatada M, Mori T. AutoBLG: automatic URL blacklist generator using search space expansion and filters. In: 2015 IEEE Symposium on Computers and Communication (ISCC); 2015. p. 625–31. https://doi.org/10.1109/ISCC.2015.7405584.
[36] Vidyakeerthi S, Nabeel M, Elvitigala C, Keppitiyagama. Demo C. PhishChain: A decentralized and transparent system to blacklist phishing URLs. In: Companion Proceedings of the Web Conference 2022; 2022. p. 286–9. https://doi.org/10.1145/3487553.3524235.
[37] Elgharbi SE, Ait Yahia M, Ouchani S. Online phishing detection: A heuristic-based machine learning framework. In: 2024 13th Mediterranean Conference on Embedded Computing (MECO); 2024. p. 1–4. https://doi.org/10.1109/MECO62516.2024.10577848.
[38] Oest A, Safaei Y, Doupe A, Ahn GJ, Wardman B, Tyers K. PhishFarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists. In: 2019 IEEE Symposium on Security and Privacy (SP); 2019. p. 1344–61. https://doi.org/10.1109/SP.2019.00049.
[39] Cisco Systems Inc. PhishTank: A collaborative anti-phishing site. Phishtank.Org; 2025.
[40] Azeez NA, Misra S, Margaret IA, Fernandez-Sanz L, Abdulhamid SM. Adopting automated whitelist approach for detecting phishing attacks. Comput Secur 2021;108:102328.
[41] Mahboubi A, Luong K, Aboutorab H, Bui HT, Jarrad G, Bahutair M, et al. Evolving techniques in cyber threat hunting: A systematic review. J Netw Comput Appl 2024;232:104004.
[42] Jayaraj R, Pushpalatha A, Sangeetha K, Kamaleshwar T, Udhaya Shree S, Damodaran D. Intrusion detection based on phishing detection with machine learning. Meas Sens 2024;31:101003.
[43] Chiew KL, Tan CL, Wong K, Yong KSC, Tiong WK. A new hybrid ensemble feature selection framework for machine learning-based phishing detection system. Inf Sci 2019;484:153–66.
[44] Rafsanjani AS, Binti Kamaruddin N, Behjati M, Aslam S, Sarfaraz A, Amphawan A. Enhancing malicious URL detection: A novel framework leveraging priority coefficient and feature evaluation. IEEE Access 2024;12:85001–26.
[45] Orunsolu AA, Sodiya AS, Akinwale AT. A predictive model for phishing detection. J King Saud Univ - Comput Inf Sci 2022;34:232–47.
[46] Haq QEU, Faheem MH, Ahmad I. Detecting phishing URLs based on a deep learning approach to prevent cyber-attacks. Appl Sci 2024;14:10086.
[47] Shahid WB, Aslam B, Abbas H, Khalid SB, Afzal H. An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling. J Netw Comput Appl 2022;198:103270.
[48] Asiri S, Xiao Y, Alzahrani S, Li T. PhishingRTDS: A real-time detection system for phishing attacks using a Deep learning model. Comput Secur 2024;141:103843.
[49] Sanchez-Paniagua M, Fernandez EF, Alegre E, Al-Nabki W, Gonzalez-Castro V. Phishing URL detection: A real-case scenario through login URLs. IEEE Access 2022;10:42949–60.
[50] Rashid F, Doyle B, Han SC, Seneviratne S. Phishing URL detection generalisation using Unsupervised Domain Adaptation. Comput Netw 2024;245:110398.
[51] Varshney G, Kumawat R, Varadharajan V, Tupakula U, Gupta C. Anti-phishing: A comprehensive perspective. Expert Syst Appl 2024;238:122199.
[52] Alsariera YA, Elijah AV, Balogun AO. Phishing website detection: forest by penalizing attributes algorithm and its enhanced variations. Arab J Sci Eng 2020;45:10459–70.
[53] Omolara AE, Alawida M. DaE2: Unmasking malicious URLs by leveraging diverse and efficient ensemble machine learning for online security. Comput Secur 2025;148:104170.
[54] Aljofey A, Bello SA, Lu J, Xu C. Comprehensive phishing detection: A multi-channel approach with variants TCN fusion leveraging URL and HTML features. J Netw Comput Appl 2025;238:104170.
[55] Yang J, Wu Y, Yuan Y, Xue H, Bourouis S, Abdel-Salam M, et al. LLM-AE-MP: web attack detection using a large language model with autoencoder and multilayer perceptron. Expert Syst Appl 2025;274:126982.
[56] Otieno DO, Abri F, Namin AS, Jones KS. Detecting phishing URLs using the BERT Transformer model. In: 2023 IEEE International Conference on Big Data (BigData); 2023. p. 2483–92. https://doi.org/10.1109/BigData59044.2023.10386782.
[57] Zhou J, Zhang K, Bilal A, Zhou Y, Fan Y, Pan W, et al. An integrated CSPPC and BiLSTM framework for malicious URL detection. Sci Rep 2025;15:6659.
[58] V.A.V. Adap, G.A. Castillo, E.J.M. Delos Reyes, E.B. Ronquillo, L.A. Vea. Do not feed the phish: phishing website detection using URL-based features. 2023 the 5th World Symposium on Software Engineering (WSSE). (2023) p. 135–41, https://doi.org/10.1145/3631991.3632011.
[59] Prabakaran MK, Sundaram PMeenakshi, Chandrasekar AD. An enhanced deep learning-based phishing detection mechanism to effectively identify malicious URLs using variational autoencoders. IET Inf Secur 2023;17:423–40.
[60] Trad F, Chehab A. Prompt engineering or fine-tuning? A case study on phishing detection with large language models. Mach Learn Knowl Extr 2024;6:367–84.
[61] Ali I, Subba B. PhishURLDetect: A parameter efficient fine-tuning of LLMs using LoRA for detection of phishing URLs. In: Proceedings of the 26th International Conference on Distributed Computing and Networking; 2025. p. 278–9. https://doi.org/10.1145/3700838.3703658.
[62] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, et al. Attention is all you need. (2017), https://doi.org/10.48550/ARXIV.1706.03762.
[63] Asiri S, Xiao Y, Li T. PhishTransformer: A novel approach to detect phishing attacks using URL collection and Transformer. Electronics 2023;13:30.
[64] Yoon JH, Buu SJ, Kim HJ. Phishing webpage detection via multi-modal integration of HTML DOM graphs and URL features based on graph convolutional and transformer networks. Electronics 2024;13:3344.
[65] Do NQ, Selamat A, Fujita H, Krejcar O. An integrated model based on deep learning classifiers and pre-trained transformer for phishing URL detection. Future Gener Comput Syst 2024;161:269–85.
[66] Majgave AB, Gavankar NL. Automatic phishing website detection and prevention model using transformer deep belief network. Comput Secur 2024;147:104071.
[67] Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. Sov Phys Dokl 1966;10:707–10.

[68] Zouina M, Outtaj B. A novel lightweight URL phishing detection system using SVM and similarity index. Hum-Centric Comput Inf Sci 2017;7:17.
[69] Chalyi O, Driaunys K, Rudžionis V. Assessing browser security: A detailed study based on CVE metrics. Future Internet. 2025;17:104.
[70] Sahingoz OK, Buber E, Demir O, Diri B. Machine learning based phishing detection from URLs. Expert Syst Appl 2019;117:345–57.
[71] Zhu E, Yuan Q, Chen Z, Li X, Fang X. CCBLA: a lightweight phishing detection model based on CNN, BiLSTM, and attention mechanism. Cogn Comput 2023;15:1320–33.
[72] Do NQ, Selamat A, Krejcar O, Fujita H. Detection of malicious URLs using Temporal Convolutional Network and Multi-Head self-attention mechanism. Appl Soft Comput 2025;169:112540.
[73] Prasad A, Chandra S. PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning. Comput Secur 2024;136:103545.
[74] Bustio-Martínez L, Álvarez-Carmona MA, Herrera-Semenets V, Feregrino-Uribe C, Cumplido R. A lightweight data representation for phishing URLs detection in IoT environments. Inf Sci 2022;603:42–59.
[75] Opara C, Modesti P, Golightly L. Evaluating spam filters and stylometric detection of AI-generated phishing emails. Expert Syst Appl 2025;276:127044.
[76] Akçam ÖŞ, Tekerek A, Tekerek M. Development of BiLSTM deep learning model to detect URL-based phishing attacks. Comput Electr Eng 2025;123:110212.
[77] Liu R, Wang Y, Xu H, Qin Z, Zhang F, Liu Y, et al. PMANet: malicious URL detection via post-trained language model guided multi-level feature attention network. Inf Fusion 2025;113:102638.
[78] Senanayake J, Rajapaksha S, Yanai N, Kalutarage H, Komiya C. MADONNA: browser-based malicious domain detection using Optimized Neural Network by leveraging AI and feature analysis. Comput Secur 2025;152:104371.