

Article

SenScanner: An Artificial Intelligence-Based Automatic Password-Related Secret Detection System in Mixed Texts

Zhuofeng He ^{1,2}, Yumeng Guo ^{2,*}, Bo Zhang ^{3,*} and Wenzhi Cao ^{2,4}

¹ Business School, Central South University, Changsha 410083, China; zhuofeng@csu.edu.cn

² Xiangjiang Laboratory, Changsha 410205, China; wenzcao@hutb.edu.cn

³ Phytium Technology Co., Ltd., Beijing 100089, China

⁴ School of Advanced Interdisciplinary Studies, Hunan University of Technology and Business, Changsha 410205, China

* Correspondence: yumeng.guo@xjlab.org.cn (Y.G.); zhangbo1078@phytium.com.cn (B.Z.)

Abstract

The rapid expansion of the Internet has enabled large-scale information sharing while also increasing the risk of sensitive information leakage. Authentication secrets, including passwords and API keys, may be unintentionally exposed in publicly accessible environments such as web pages, network packets, and code-sharing platforms when they are mishandled by developers or operators. Such leakage allows attackers to abuse third-party authentication services and may lead to unauthorized access, fraud, or broader compromise. Therefore, timely and accurate detection of sensitive information in network data is essential for reducing security risks. This paper presents SenScanner, an artificial intelligence-based model for automatically identifying password-related secrets in mixed text. By combining natural language processing and machine learning techniques, SenScanner detects leaked password-related sensitive information across heterogeneous textual contexts. Experimental results on 2000 public data samples show that SenScanner achieves superior precision, recall, and F1-score, demonstrating its effectiveness in reducing false positives and manual review effort.

Keywords: information leakage; passwords; mixed text; natural language processing; machine learning; sensitive information detection

1. Introduction

With the accelerating development of the digital society, Internet penetration and the scale of online users have increased substantially. Online platforms have become a primary medium for constructing, sharing, and reusing digital resources. However, while the Internet provides convenient access to digital services, it also introduces increasingly severe cybersecurity risks, among which large-scale sensitive information leakage is particularly prominent [1]. Most network systems deploy automated vulnerability scanners, web application firewalls, and other security mechanisms for detection and defense [2,3]. Nevertheless, sensitive information leakage is often caused by human errors or logic-level misconfigurations, and is therefore difficult to prevent or identify through conventional automated defenses. For example, GitHub, as a major code-sharing platform [4], may host code that unintentionally contains keys, user passwords, authentication tokens, or private keys, resulting in public exposure of such secrets [5]. A large number of Amazon Web

Academic Editor: Touhid Bhuiyan

Received: 26 May 2026

Revised: 21 June 2026

Accepted: 30 June 2026

Published: 2 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\) license](https://creativecommons.org/licenses/by/4.0/).

Services (AWS) keys have reportedly been leaked on GitHub [6] and maliciously abused by attackers, causing substantial economic losses [7]. The 2018 Facebook data-leakage incident further highlighted the risks associated with sensitive information exposure on the Internet [8]. In February 2021, the U.S. Federal Bureau of Investigation (FBI) also experienced a serious information-leakage incident in which hackers reportedly exposed more than 380 million records from internal files and folders, including personally identifiable information and sensitive documents. These documents reportedly involved prison records, terrorism investigations, and national security-related matters, raising significant concerns regarding national security [9]. In addition, personal information leakage in social networks has become an increasingly serious issue.

Among the various categories of compromised sensitive information, password-related information represents one of the most critical security risks [10]. Although textual passwords have well-known security limitations, they remain the dominant authentication mechanism because of their simplicity, usability, and low deployment cost. Text passwords are widely used to protect personal accounts and sensitive resources. Once compromised, they may allow attackers to access user accounts, obtain additional sensitive information, conduct fraudulent activities, or violate user privacy. Moreover, users often reuse identical or similar passwords across multiple services, such as email, social media, and online banking. As a result, a single leaked password may enable credential-stuffing attacks against multiple accounts and significantly expand the scope of harm [11,12].

Extensive academic and industrial efforts have been devoted to detecting sensitive information leakage in textual data. GitHub has introduced secret-scanning capabilities to reduce the risk of credential exposure in hosted code repositories [13]. Acunetix, a widely used vulnerability scanning tool, also provides basic functionality for detecting sensitive information in traffic and web content [14]. In addition, several open-source tools have been developed to mine key-like secrets from publicly available code [15–17]. Sinha et al. investigated methods for detecting leaked API keys in source-code repositories, including heuristic filtering strategies that improve the discrimination of candidate key strings [18]. Qu studied password detection for Internet-of-Things devices [19], whereas Meli et al. conducted a large-scale characterization of secret leakage in public GitHub repositories [20]. Although these tools and studies provide useful support for sensitive-information detection, most of them focus primarily on key-like strings and rely heavily on regular expressions. Such methods are often insufficient for heterogeneous text and complex password-like secrets, leading to substantial false positives and false negatives.

Given the severe impact of sensitive information leakage on cyberspace security, automatic and intelligent techniques are needed to detect password-related secrets in network data. To identify leaked password-related information in open Internet text more comprehensively and accurately, the following technical challenges must be addressed.

Text-type diversity. Text exposed on the web includes not only source code from code-sharing platforms, but also natural-language content from web pages and plain-text data from network transmission. A single regular-expression-based strategy is difficult to generalize across such heterogeneous formats. For programming languages, syntax and structural conventions vary considerably, and several studies have therefore proposed code-specific analysis methods. Nguyen et al. used an rVSM-based deep neural network (DNN) to process code elements and achieved higher accuracy than traditional approaches in code classification tasks [21]. Mou et al. argued that source code has stricter structural constraints than natural language; they represented such structure using abstract syntax trees and designed coding standards for computing vector representations of code. Their approach captures high-level code semantics through convolutional neural networks to support code classification and search [22]. However, these methods are not applicable to all programming languages and cannot simultaneously analyze source code

and natural-language text. Therefore, handling diverse text types remains a key technical challenge for password-information detection.

Secret-composition complexity. Existing techniques mainly rely on regular expressions to match keys [14–16]. In practice, however, password-related secrets appear in diverse application scenarios and may include keys, user passwords, authentication tokens, and private keys. Unlike structured API keys, textual passwords can be almost arbitrary strings and often lack fixed formats. This variability makes it difficult to detect password-like secrets with high precision using rule-based matching alone.

Need for intelligent detection. Password-related secrets are often embedded within large volumes of surrounding text. Tools such as Gshark and Gsil search GitHub repositories for sensitive information using keywords [23,24], but their results commonly contain many false positives and require manual verification. Google hacking also enables the discovery of vulnerable web applications, exposed file types, error messages, and files containing credentials by using advanced search operators [25]. Although these approaches can retrieve a large amount of potentially relevant information, they still depend heavily on manual screening, which is inefficient and error-prone at scale. Therefore, accurately locating and extracting password-sensitive information from massive and complex text remains an important problem.

To address these challenges, this paper adopts a hybrid detection strategy. For text-type diversity, a trained binary classifier first distinguishes natural-language text from code text, after which different detection models are applied to the corresponding text types. For the complexity of cryptographic composition, Named Entity Recognition (NER) is used to extract password-related secrets from natural language [26], while a machine-learning-based method is used to identify password-related secrets in code text [27]. For intelligent detection, SenScanner replaces manual screening with an automated machine-learning pipeline [28].

This paper proposes SenScanner, an automated and intelligent method for detecting password-related secrets in multi-source mixed text. Different from prior secret-detection tools that mainly rely on regular expressions, high-entropy strings, or code-oriented patterns, SenScanner explicitly models mixed-text secret detection as a two-stage routing and type-specific detection problem. The main contributions are as follows.

- (1) We propose a lightweight mixed-text detection framework that first distinguishes code text from natural-language text at the line level and then applies type-specific detection models. This design enables SenScanner to handle heterogeneous inputs in which source code, comments, configuration-like text, and natural-language descriptions may coexist.
- (2) We design a context-aware code-text detection module for password-related secrets. Instead of relying only on regular expressions, the module combines candidate extraction, non-secret filtering, assignment-style features, keyword-context features, file-level occurrence features, and intrinsic string-composition features. This improves the ability to distinguish real password-related secrets from ordinary literals, placeholders, and repeated constants.
- (3) We fine-tune a lightweight ELECTRA-based NER sub-model for identifying password-related secrets in natural-language contexts. Compared with general-purpose NER models or large language models, this task-specific NER component is more suitable for lightweight deployment and large-scale scanning.
- (4) We construct a controlled mixed-text benchmark and conduct component-level and end-to-end evaluations. The results show that SenScanner achieves better performance than selected open-source and regex-based baselines under the default non-dataset-specific configurations used in this study. We also discuss data authenticity,

baseline configuration, error patterns, and practical limitations to clarify the scope and reliability of the results.

2. Related Work

Password security. Among different types of sensitive information, password leakage is particularly harmful. Password-based authentication remains a core mechanism for web login and access control; once attackers obtain high-privilege passwords, they can acquire unauthorized service permissions and cause serious damage. Consequently, password security has attracted sustained research attention. Morris and Thompson emphasized that passwords should balance security and usability and presented a pioneering study on password cracking [29]. More recently, researchers have shown that machine learning and deep neural networks can support password-security assessment and leakage detection. Feng et al. proposed a machine-learning-based method for detecting password leakage in public GitHub repositories. Their approach uses two models to determine whether a candidate string is key-like and whether its surrounding context is related to secret information, and then combines the two judgments to identify leaked credentials [30]. Sinha et al. studied leaked API-key detection in source code and proposed heuristic filtering based on string location and entropy [18]. Qu investigated password-related secrets leakage in Internet-of-Things devices [19]. Sibai et al. applied artificial neural networks to secure password detection [31]. Vijaya et al. and Hong et al. used machine learning techniques for password-strength evaluation [9,32]. Hitaj et al. employed deep generative adversarial networks to learn password distributions and generate high-quality password guesses [33], while Melicher et al. developed a recurrent neural network model for estimating password guessability [34]. In contrast to studies focused on password guessing or key detection, SenScanner aims to identify password-like cryptographic strings embedded in complex mixed text.

Classification models. Classification is a fundamental task in machine learning, where an input instance is assigned to one or more predefined categories. Over the past decades, numerous classification algorithms have been proposed, including support vector machines [35], decision trees [36], Naive Bayes [37], and random forests [38]. With the development of deep learning, neural-network-based classifiers have also become widely studied [39]. Jackins et al. combined artificial intelligence with Naive Bayes and random forest classifiers to predict clinical diseases from multiple datasets [40]. Gu and Lu proposed an intrusion-detection framework that integrates support vector machines with Naive Bayes feature embedding [41]. Itoo et al. compared logistic regression, Naive Bayes, and k-nearest-neighbor algorithms for credit card fraud detection [42]. Sheykhou et al. systematically compared support vector machines and random forests for remote sensing image classification and discussed related challenges and future directions [43]. Motivated by these studies, SenScanner formulates password detection as a classification problem and uses lightweight binary classifiers, including Naive Bayes and decision trees, to support efficient training on relatively small datasets.

Named Entity Recognition (NER). Natural Language Processing (NLP) studies computational methods for processing and understanding human language [44]. It has been widely applied to text classification, sentiment analysis, information extraction, and machine translation. Named Entity Recognition (NER), a core NLP task, aims to identify text spans that refer to entities with specific semantic types, such as persons, locations, organizations, and domain-specific concepts [26,45]. Wang et al. implemented NER using large language models and demonstrated strong performance in low-resource and few-shot settings [46]. Fields et al. combined NLP and network analysis to study social interactions and relationship patterns in historical diaries [47]. Lison et al. proposed a weak-supervision method for training NER models without labeled data [48]. Yu et al. formulated NER

as a dependency-parsing problem and achieved State-of-the-Art performance on multiple corpora [49].

Building on these studies, SenScanner addresses the difficulty of obtaining large-scale labeled password-leakage data by combining Named Entity Recognition with machine learning-based classification. This hybrid strategy enables the model to detect password-related secrets in both natural-language and code-text contexts.

3. SenScanner Model Design

3.1. Design Overview

This paper models password-like information detection in mixed text as a combined classification and extraction problem. Specifically, SenScanner processes input text line by line, classifies each line as either code text or natural-language text, and then applies the corresponding detection model to extract password-like sensitive information. Figure 1 illustrates the overall workflow.

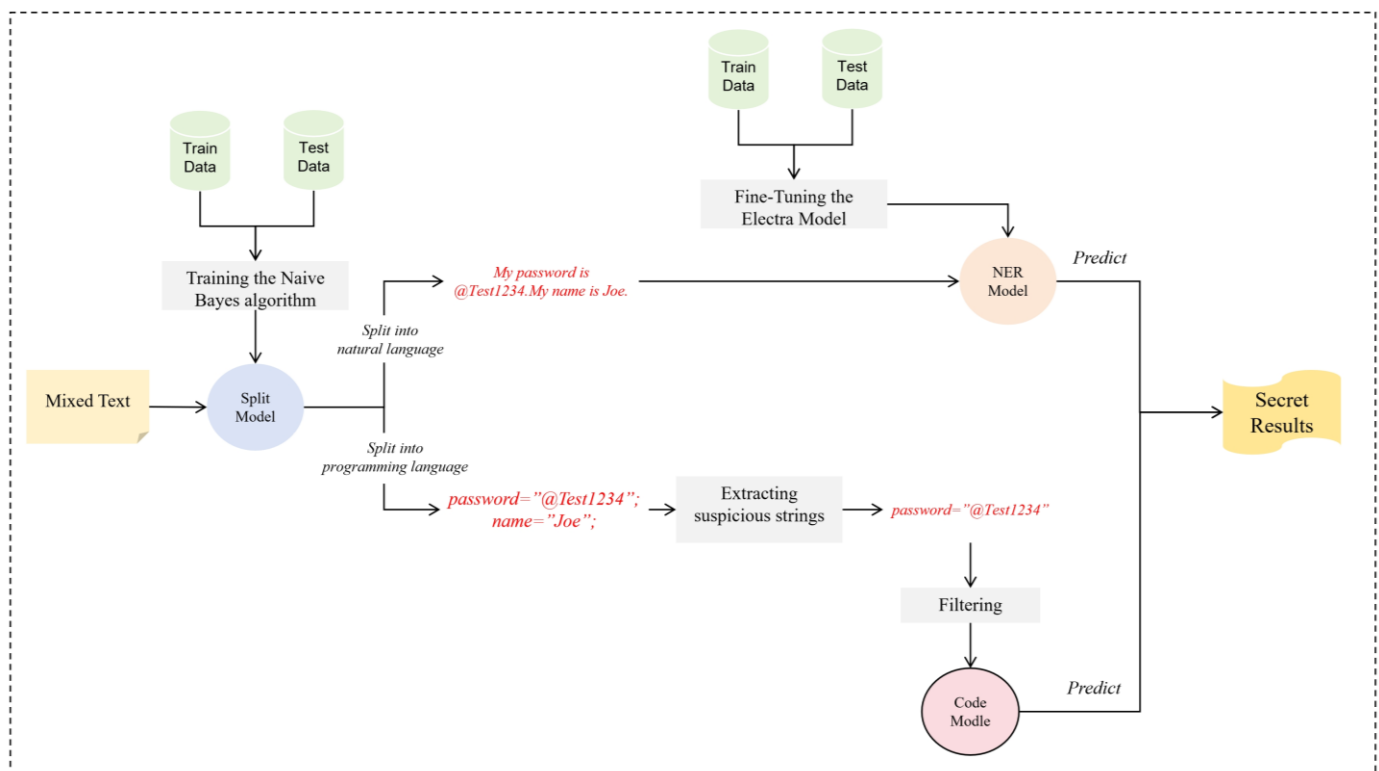


Figure 1. Overall architecture and workflow of SenScanner.

The SenScanner workflow consists of two major stages. The first stage distinguishes code language from natural language, and the second stage detects and extracts password-related secrets. The password-related secrets considered in this study include keys, user passwords, authentication tokens, private keys, and related secrets. Because code and natural language may coexist within the same document, SenScanner performs line-level text-type classification. Section 3.2 describes this classification procedure. For code text, SenScanner first extracts hard-coded candidate strings using regular expressions and then applies a machine-learning model to identify password-related secrets, as described in Section 3.3. For natural-language text, SenScanner uses a trained NER sub-model to recognize cryptographic entities, as described in Section 3.4.

3.2. Text-Type Detection in Mixed Text

SenScanner first determines whether each line of text belongs to natural language or code language, because the two text types require different detection strategies. Since a single document may contain both types, SenScanner splits the input by line and performs line-level classification. Based on the observation that code text and natural-language text exhibit distinct lexical and structural patterns, this study trains a Naive Bayes binary classifier for text-type identification [37]. CountVectorizer [50,51] is used to transform text into word-frequency vectors, where each row represents a document, each column represents a token, and each matrix element records the frequency of the corresponding token. MultinomialNB [52] is then used as the classifier. In the dataset construction process, 16,394 code lines were extracted from 4008 GitHub code files as negative samples, while 12,255 natural-language sentences were extracted from Wikipedia [53] as positive samples.

3.3. Password Detection in Code Text

In code text, passwords usually appear as strings. Therefore, SenScanner first extracts candidate strings using the regular expressions listed in Table 1. Because developers may include spaces before or after strings, leading and trailing spaces are removed during pre-processing.

Table 1. Rules for extracting candidate strings from code text.

Regex	Example
<code>(?<=\\').*?(?=\\')</code>	password='test123'
<code>'(?<=\\").*?(?=\\")'</code>	password="test123"
<code>(?<=:).*?(?=\\s \\\$)'</code>	password:test123
<code>(?<=).*?(?=& \\s \\\$)'</code>	password=test123

The extracted candidate strings are then filtered using regular expressions to remove common non-secret strings, including email addresses, domain names, IP addresses, illegal special-character strings, purely numeric strings, and dates. The filtering rules are summarized in Table 2.

Table 2. Filtering rules for excluding non-secret candidate strings.

Email
<code>r'[A-Za-z0-9\\.\\+_]+@[a-z0-9\\.\\+_]+\\.\\[a-z]+''</code>
Example: test@gmail.com
Domain
<code>r'(!\\-)(?:[a-zA-Z\\d\\-]{0,62}[a-zA-Z\\d\\-]{1,126})(?!\\d+)[a-zA-Z\\d]{1,63}'</code>
Example: www.google.com
ip
<code>r'\\d*\\.\\d*\\.\\d*\\.\\d*'</code>
Example: 192.168.1.1
Purely Numeric
<code>r'(^?[1-9]\\d*\$) (^[^?[1-9]\\d*\\.\\d+ 0\\.\\d*[1-9]\\d*\$) (^[^?[1-9]\\d*\\.?[\\d*(e E)[+-]?\\d+)\$'</code>
Example: 123456789
Special Characters
<code>r'[^~@/_\\+:\\.\\w\\s]'</code>
Example: test%&===123
Date
<code>r'\\d*-\\d*-\\d*\\/\\d*\\/\\d*'</code>
Example: 2008-08-08 2008/08/08

After candidate extraction and filtering, SenScanner derives features from both the password string itself and the surrounding code context. Passwords are typically composed of lowercase letters, uppercase letters, digits, and special characters, with more complex passwords containing multiple character types.

An analysis of password-containing samples reveals four common structural characteristics: (1) the line containing the password string often includes an assignment statement, such as `password=123@abc` or `password:123@abc`; (2) password-related keywords, such as `password`, `pwd`, or `secret`, often appear near the candidate string; (3) the surrounding code fragment may contain multiple password-related keywords; and (4) password strings usually appear only a small number of times within the same code file.

Based on these observations, this paper defines the 10-dimensional feature set shown in Table 3, where MAX = 999,999 denotes a missing or nonexistent value.

The selected features are intended to capture four types of evidence: assignment structures, keyword-based context, file-level repetition, and intrinsic string composition. Assignment and keyword-context features reflect typical code patterns in which password-related secrets are hard-coded near terms such as “password”, “pwd”, “secret”, “token”, or “key”. The occurrence-count feature helps filter repeated non-secret constants or template values. Character-composition features describe the complexity and format of candidate strings, which are useful for distinguishing password-like secrets from ordinary literals. Although no separate feature-level ablation study was conducted in this version, the compact and interpretable feature set was designed based on empirical observations of password-leakage samples. A detailed feature-ablation analysis will be considered in future work.

Table 3. Candidate-string features used by the code-language classifier.

No.	Feature	Value Range
1	Whether an assignment statement appears on the line containing the suspicious string	0/1
2	Character distance from the nearest keyword to the suspicious string, excluding symbols; positive values indicate that the keyword precedes the string and negative values indicate the opposite	−999,999 to 999,999
3	Line distance from the nearest keyword to the suspicious string	0 to 999,999
4	Number of keyword occurrences within the two lines before and after the suspicious string	0 to 999,999
5	Number of occurrences of the suspicious string in the code	0 to 999,999
6	Proportion of lowercase letters in the suspicious string	0 to 1
7	Proportion of uppercase letters in the suspicious string	0 to 1
8	Proportion of digits in the suspicious string	0 to 1
9	Proportion of special characters in the suspicious string	0 to 1
10	String length	0 to 999,999

For example, in the following code snippet, the feature vector extracted for the password candidate `1qaz!QAZ` is [1, 0, −1, 1, 1, 0.375, 0.375, 0.125, 0.125, 8].

```

$mail->SMTPSecure='ssl';
$mail->Host="smtp.gmail.com";
//$mail->Host="smtp.hostinger.in";
$mail->Port='465';
$mail->isHTML();
$mail->Username=<EMAIL>;
$mail->password="1qaz!QAZ";
$mail->SetFrom(<EMAIL>);
$mail->AddReplyTo($email);
$mail->FromName=$first_name." ".$last_name;
$mail->Subject="New Message from digiOmega Contact Form";
$email_content = "Name: $first_name + $last_name <br>";
$email_content .= "Phone: $phone <br>";

```

Based on the above feature analysis, this paper constructs a password-recognition model using a decision tree classifier. The decision tree was selected for three main reasons. First, the proposed feature set is low-dimensional, structured, and interpretable, consisting of assignment-style features, keyword-context features, occurrence-count features, and character-composition features. Such heterogeneous numerical and binary features can be naturally handled by tree-based threshold splitting. Second, the relationships between these features and password-related secrets are often nonlinear. For example, a candidate string may be more suspicious when it appears in an assignment statement, is close to a keyword such as “password” or “token,” and has a complex character composition. A decision tree can capture such rule-like interactions without requiring feature scaling or strong linear assumptions. Third, compared with more complex models such as SVMs or ensemble methods, a decision tree provides better interpretability and lower inference overhead, which is useful for analyzing false positives and false negatives in secret-detection scenarios.

We acknowledge that decision trees may have a risk of overfitting. Therefore, the effectiveness of the decision tree component was evaluated on a held-out test set, and additional algorithm combinations, including logistic regression, SVM, and random forest, are compared in Section 4.2.2. The results show that the original combination used by SenScanner achieves the best overall performance in our experimental setting. More complex ensemble models and additional regularization strategies will be further explored in future work.

The training dataset includes 20,000 code snippets released by Feng et al. [30], 3000 normal user passwords from the LinkedIn password dataset [54], 2500 passwords containing four character types, 2500 passwords containing at least two character types, 4000 key-replacement code snippets in which randomly selected passwords of length 20–50 are substituted into code, and 4142 GitHub code snippets containing sensitive information. Following the proposed extraction and labeling procedure, 56,562 training samples were obtained, including 16,185 positive samples and 40,377 negative samples.

3.4. NER Sub-Model for Password Detection in Natural Language

Named Entity Recognition (NER) is an NLP technique for extracting structured information from unstructured text [55]. In this paper, a password is treated as a domain-specific entity and extracted through NER. Because constructing a large manually labeled dataset is time-consuming, this study uses a pretrained model for subsequent fine-tuning [56]. Specifically, a NER sub-model is trained to recognize cryptographic entities in natural-language text.

To train the password-oriented NER sub-model, this study first constructs a synthetic training dataset. ChatGPT 5.4 [57] is used to generate 5000 sentences containing password placeholders, where each password is replaced with @PASSWORD@. Then, 5000 passwords are randomly sampled from the LinkedIn leaked-password dataset [54] and substituted for the placeholder to generate labeled training instances. ELECTRA [58] is selected as the pretrained language model and fine-tuned for this task. ELECTRA is a light-weight pretrained model for natural language processing; it outperforms GPT on the GLUE benchmark [59], achieves performance comparable to RoBERTa [60] and XLNet [61], and requires less than one quarter of their computational cost under comparable settings.

To improve the diversity and realism of the synthetic natural-language training data, ChatGPT was instructed to generate sentences in multiple password-leakage contexts, including user notes, configuration descriptions, troubleshooting messages, software documentation, issue reports, commit-like descriptions, log explanations, and informal communication. The generated sentences were required to contain a placeholder “@PASSWORD@” representing the secret value, while the surrounding context was required to include natural expressions related to password use, storage, sharing, configuration, or authentication. A representative prompt used for data generation is as follows: “Generate English sentences that may appear in real software-related documents, comments, logs, issue reports, or user notes. Each sentence should contain one placeholder @PASSWORD@ as a password-related secret. Use diverse writing styles and contexts, and avoid repeating the same sentence pattern.” After generation, the placeholder was replaced with passwords sampled from the public leaked-password dataset to construct labeled NER training instances.

4. Evaluation

This section evaluates SenScanner from three perspectives: (1) the performance of each component model; (2) the end-to-end performance of SenScanner in detecting sensitive information; and (3) comparisons with classical baselines and recent related models using precision, recall, and F1-score.

To improve reproducibility, we provide the main implementation and evaluation details used in this study. The text-type classifier was implemented using CountVectorizer and MultinomialNB. The code-text detector used the candidate-extraction and filtering rules listed in Tables 1 and 2, followed by the 10-dimensional feature set listed in Table 3 and a DecisionTreeClassifier. The natural-language detector was implemented by fine-tuning an ELECTRA-based NER model using the synthetic training data described in Section 3.4. For dataset construction, the generated natural-language statements were inserted into GitHub code files using the line-level random insertion strategy described in Section 4.2.1. Exact duplicate files, snippets, lines, and candidate strings were removed based on normalized textual content before training and evaluation. Detect-Secrets and TruffleHog were executed using their default or recommended configurations. The evaluation metrics were computed at the secret level after aggregating line-level outputs within each file.

4.1. Component-Level Model Performance

For the Naive Bayes-based text-type classifier, the dataset described in Section 3.2 is divided into training and test sets at an 8:2 ratio. The trained classifier is evaluated on the test set. As shown in Table 4, the model achieves a precision of 95%, a recall of 97%, and an F1-score of 96%.

For the decision tree-based password classifier, the dataset described in Section 3.3 is also divided into training and test sets at an 8:2 ratio. As shown in Table 4, this classifier achieves a precision of 94.19%, a recall of 95.40%, and an F1-score of 94.79%.

For the NER sub-model, the dataset described in Section 3.4 is divided into training, test, and validation sets at an 8:1:1 ratio. Hyperparameters are tuned and evaluated on the validation set. As shown in Table 4, the NER sub-model achieves a precision of 94.47%, a recall of 97.69%, and an F1-score of 96.04%.

Table 4. Component-level performance of SenScanner models.

	Precision	Recall	F1-Score
MultinomialNB Model	95%	97%	96%
DecisionTreeClassifier	94.19%	95.40%	94.79%
NER Model	94.47%	97.69%	96.04%

4.2. End-to-End Sensitive Information Detection

This section evaluates the end-to-end performance of SenScanner, focusing on its ability to detect secret-type sensitive information in mixed text.

4.2.1. Experimental Data

The evaluation dataset was constructed from 2019 GitHub code files containing 2607 password-related secrets. In addition, ChatGPT was used to generate 1044 English natural-language statements, of which 719 contained password-related secrets. Secret values were randomly sampled from the LinkedIn dataset described in Section 3.3 and inserted into these statements. The generated natural-language statements were then embedded into the GitHub code files to construct mixed-text samples containing both code and natural language. The final dataset included 1247 files containing code-context password-related secrets, 616 files containing natural-language password-related secrets, and 389 files containing both types, with a total of 3326 password-related secrets.

To construct the mixed-text samples, we adopted a line-level random insertion strategy. Specifically, each GitHub code file was first split into lines, and one or more generated natural-language statements were randomly inserted as independent lines at randomly selected positions within the file. The inserted statements were not restricted to specific syntactic locations, such as comments or docstrings. This design was intended to create heterogeneous files in which code lines and natural-language lines coexist, so that SenScanner's line-level text-type classifier and type-specific detection modules could be evaluated in an end-to-end setting. We acknowledge that this construction does not fully reproduce all organic mixed-text formats, such as Markdown documents, long code comments, issue discussions, or structured documentation strings. Nevertheless, it provides a controlled benchmark for evaluating secret detection in mixed code and natural-language contexts.

Since real-world mixed-text datasets containing leaked credentials are difficult to obtain and release due to privacy, ethical, and security constraints, this study adopts a controlled data-construction strategy. The generated natural-language statements and embedded code contexts were reviewed by cybersecurity experts to ensure that the constructed samples reflect common password-leakage scenarios, such as hard-coded credentials, configuration-like assignments, and password descriptions in natural-language contexts. Nevertheless, we acknowledge that constructed data cannot fully represent all real-world leakage distributions.

Data Deduplication and Splitting Strategy. To reduce the risk of data leakage between training and evaluation, we performed de-duplication checks before model training and testing. Exact duplicate code files, code snippets, code lines, and extracted candidate strings were removed based on normalized textual content. The normalization process included removing leading and trailing whitespace, unifying line endings, and standardizing repeated blank characters. For GitHub-based samples with available repository metadata, we ensured that samples from the same repository were not simultaneously included in the training and test sets. For public datasets or extracted snippets without complete repository metadata, we performed file-level and snippet-level de-duplication before splitting.

After de-duplication, we further checked the final partitions and found no identical files, snippets, or candidate strings shared between the training and test sets. We acknowledge that complete repository-level splitting is difficult when repository metadata are missing from public snippets. Therefore, we regard this as a limitation and will construct fully repository-level separated benchmarks in future work.

4.2.2. Model Validity and Baseline Comparison

End-to-end effectiveness. Two open-source tools are selected as baselines. The first is Detect-Secrets, a GitHub project with approximately 3000 stars, indicating broad attention from the developer community. Detect-Secrets relies on regular expressions and heuristic rules to reduce false positives. The second is TruffleHog [62], a data-mining tool for discovering high-entropy strings and sensitive information in target GitHub repositories; the project has approximately 12,000 stars. In addition, this study extends the regular expressions used by Feng et al. [30] to construct a regex-based baseline. The resulting expressions are shown in Table 5, where Secret_keyword is defined as:

[db_pass|password|passwd|secrets|login|auth|admin|mongo|cred|secure|db|mysql|login|pw|secret|token|key|pass|pwd].

Table 5. Regular expressions used by the regex-based baseline.

Target Regular Expression
(secret_keyword)\s*:=\s*["'](.?)[“\’]
(secret_keyword)[\"]{0,2}?:\s*?([\"]?)([^\s]+\2
(secret_keyword)[\']{0,2}?:\s*?([\']?)([^\s]+\2
(secret_keyword)(\[\])?\[s\]*=[s\]*(@)?"([^\s]+)"
(secret_keyword)(\[\"]{0,2})?\s*=\s*([^\s]+)
(secret_keyword)(\[\']{0,2})?\s*=\s*([^\s]+)\3
(secret_keyword)\s+("[^"]+"' '[^']*'+\');

Baseline Configuration. Detect-Secrets and TruffleHog were selected as representative open-source baselines because their detection objectives are consistent with that of SenScanner, namely automatically identifying credential-like secrets in code or textual data. To ensure a practical and reproducible comparison, both tools were executed using their default or recommended configurations rather than dataset-specific tuning. Specifically, we did not manually optimize entropy thresholds, customize regular-expression rules, or construct dataset-specific allowlists based on the evaluation set. This setting reflects the typical out-of-the-box usage of these tools and avoids introducing additional bias by tuning the baselines toward the constructed benchmark. All methods were evaluated on the same mixed-text dataset, and the regex-based baseline used the rules listed in Table 5.

Aggregation Rule for End-to-End Evaluation. SenScanner processes each input file in a line-by-line manner. For each file, the text is first split into lines, and each line is classified

as either code text or natural-language text. The corresponding detection module is then applied according to the line type. For code lines, the code-language detector extracts and classifies password-related secret candidates. For natural-language lines, the NER sub-model extracts password-related entities. The detection results from all lines are then aggregated as the output of the file. To avoid repeated counting, identical detected secret values within the same file are normalized and counted only once.

Table 6 reports secret-level detection results over the entire mixed-text benchmark, rather than file-level binary classification results. A detected item is counted as a true positive if its normalized value matches a labeled password-related secret in the same file. A detected item that does not match any labeled secret in the same file is counted as a false positive. Conversely, a labeled secret that is not matched by any detected item in the same file is counted as a false negative.

As shown in Table 6, SenScanner achieves a precision of 84.40%, a recall of 83.76%, and an F1-score of 84.08% on the mixed-text samples. In contrast, among the three baselines, the highest precision is 65.23%, the highest recall is 47.32%, and the highest F1-score is 36.77%. These results show that, under the default non-dataset-specific configurations used in this study, SenScanner achieves better performance than the selected open-source and regex-based baselines. This indicates that the hybrid design of SenScanner is effective for detecting password-related secrets in mixed text, especially in scenarios where secrets appear in both code and natural-language contexts.

Table 6. End-to-end comparison results on mixed text data.

	SenScanner	Detect-Secrets	TruffleHog	Regex-Based
Detected password-related secrets	3301	630	213	5235
True positives	2786	411	9	1574
False positives	515	219	204	3661
False negatives	535	2915	3113	1752
Precision	84.40%	65.23%	4.23%	30.07%
Recall	83.76%	12.36%	0.27%	47.32%
F1-score	84.08%	20.78%	0.54%	36.77%

The regex-based baseline produces a large number of false positives because it mainly relies on keyword-triggered pattern matching. Once a line contains a sensitive keyword such as “password,” “token,” or “key,” the regular expressions tend to extract nearby strings as suspicious values, even when these strings are not actual password-related secrets. For example, example values, placeholders, configuration constants, file names, hash-like strings, or descriptive text may be incorrectly detected. Such errors could be reduced by adding additional filtering strategies, such as entropy checks, blacklist/whitelist rules for common placeholders, context-aware validation, and candidate-value type analysis.

SenScanner addresses this limitation through a multi-stage detection pipeline. First, it distinguishes code text from natural-language text and applies different detection strategies to different text types. Second, for code text, SenScanner filters common non-secret strings and uses contextual features, including assignment structures, distances to password-related keywords, local keyword occurrences, and candidate occurrence counts. Third, the decision-tree classifier further determines whether a candidate string is likely to be a password-related secret instead of accepting all regex-matched strings. For natural-language text, the NER sub-model extracts credential-like entities based on semantic context. Therefore, compared with the regex-based baseline, SenScanner can better suppress

false positives caused by simple keyword matching while maintaining detection capability in mixed-text scenarios.

SenScanner achieves higher precision, recall, and F1-score than the baseline methods for two main reasons. First, SenScanner explicitly distinguishes code text from natural-language text and applies different detection strategies to different scenarios, whereas the baselines apply largely uniform detection rules and therefore fail to identify many secrets embedded in natural language. Second, Detect-Secrets, TruffleHog, and the regex-based baseline rely primarily on keywords, high-entropy patterns, or regular expressions, and are thus better suited to detecting structured key strings than arbitrary password-like strings. This limitation increases false negatives in complex text. The regex-based baseline also produces a large number of false positives because it lacks sufficient filtering after extracting suspicious strings.

Error Analysis. Although SenScanner achieves better end-to-end performance than the baseline methods, it still produces false positives and false negatives. As shown in Table 6, SenScanner reports 515 false positives and 535 false negatives on the constructed mixed-text benchmark. To better understand these errors, we manually inspected representative misclassified cases and summarized the main error patterns, as shown in Table 7.

The false positives mainly fall into three categories. First, some placeholder or demonstration strings are incorrectly identified as password-related secrets, such as “password=example123” or “token=test_token”. These strings are syntactically similar to real secrets but are used only for documentation, examples, or testing. Second, some high-complexity non-secret strings, such as hashes, random identifiers, file names, and configuration constants, are misclassified because they contain digits, uppercase letters, lowercase letters, and special characters. Third, in natural-language text, some sentences describing passwords, keys, or tokens are incorrectly recognized by the NER model even when no actual secret value is present.

The false negatives also show three major patterns. First, some secrets are missed when they appear without clear contextual keywords such as “password”, “pwd”, “token”, or “key”. In such cases, the code-language classifier lacks sufficient semantic context to distinguish a secret from an ordinary string. Second, some secrets are missed when they are split across multiple lines or embedded in complex structured formats, because SenScanner performs line-level text-type classification and candidate extraction. Third, some natural-language secrets are missed when they appear in non-standard writing styles, abbreviations, or non-English contexts, which are less represented in the current training data.

These error patterns suggest several possible improvements. Placeholder-aware filtering and document-context analysis could reduce false positives caused by examples and test values. Additional features for distinguishing hashes, identifiers, and configuration constants from real secrets could further improve precision. For false negatives, incorporating multi-line context, file-level aggregation, and more diverse multilingual or domain-specific training data may improve the robustness of SenScanner in complex real-world scenarios.

Table 7. Representative misclassified cases and the main error patterns

Error Type	Pattern	Example	Possible Reason	Potential Improvement
FP	Placeholder/test value	password=example123	Similar to real assignment-style secret	Placeholder-aware filtering
FP	High-complexity non-secret string	file_hash=9f2a...	Character composition resembles a secret	Add hash/ID-specific features

FP	Descriptive natural-language sentence	The password field should be encrypted.	NER detects context without actual value	Entity-boundary and value-existence constraints
FN	Secret without keyword	auth="x7@kL9p"	Weak contextual signal	Broader context modeling
FN	Multi-line secret	password = next line "abc123"	Line-based processing loses context	Multi-line candidate extraction
FN	Non-standard or multilingual expression	Kou ling shi abc123	Underrepresented in training data	Multilingual/stress-test data

Algorithm comparison. To examine the influence of classifier selection, this study compares different combinations of the two classification components in SenScanner: the text-type classifier and the code-language password-related secret classifier. In the original SenScanner pipeline, MultinomialNB is used for line-level text-type classification, and DecisionTreeClassifier is used for detecting password-related secret candidates in code text. In the alternative settings, the corresponding classification components are replaced with logistic regression, support vector machine (SVM), and random forest classifiers, while the remaining pipeline, feature set, training data, and evaluation data remain unchanged. Therefore, the comparison focuses on how different classifier combinations affect the overall end-to-end detection performance. All variants are trained on the same training dataset and evaluated on the same test dataset. Precision, recall, and F1-score are shown in Figure 2.

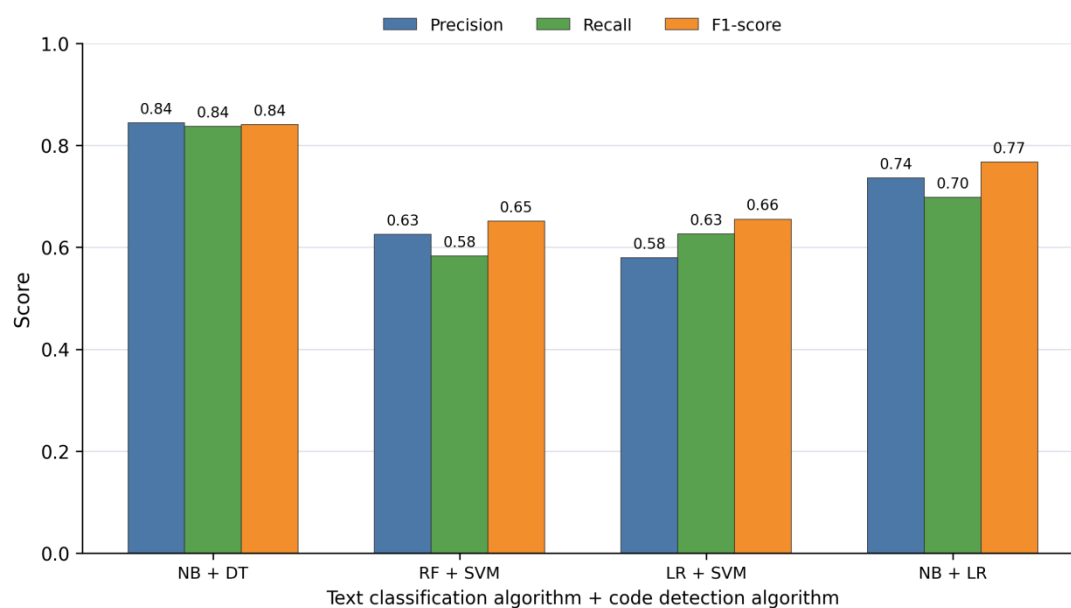


Figure 2. Performance comparison of different classification-algorithm combinations.

The results show that the combination of Naive Bayes for text-type classification and decision tree for code-language password detection performs better than the alternative combinations. SenScanner uses Naive Bayes to classify text types and a decision tree to detect password candidates in code text. Both algorithms can handle noisy or incomplete data, and decision trees are effective for feature sets with nonlinear relationships. Naive Bayes offers fast training and low storage overhead, making it suitable for large-scale text classification. Its feature-independence assumption also helps reduce the risk of high-dimensional sparsity in word-frequency representations. Since text-type classification is performed before password detection, classification errors can directly affect downstream

detection. Consequently, variants using Naive Bayes for text-type classification achieve better overall performance.

Although the NER sub-model plays an important role in detecting password-related secrets from natural-language text, we did not include additional generic NER or large language model-based baselines for this component. The main reason is that general-purpose NER models are typically designed to recognize conventional entity types, such as persons, organizations, locations, or dates, rather than credential-like strings such as passwords, API keys, tokens, or private keys. Therefore, directly applying generic NER systems would not provide a fair or task-aligned comparison.

In contrast, the NER sub-model used in SenScanner is fine-tuned specifically for the password-related secret detection task. It is trained to identify credential-like entities in natural-language contexts and is integrated into the lightweight mixed-text detection pipeline. We selected ELECTRA because it provides a favorable balance between detection performance and computational efficiency, making it more suitable for large-scale scanning than heavier language models. Large language models may also be used for secret extraction, but their high inference cost, large model size, and deployment overhead make them less suitable as practical baselines in the lightweight scanning scenario considered in this study.

Therefore, this paper focuses on comparing SenScanner with widely used secret-detection tools and a regex-based baseline in an end-to-end mixed-text setting. The component-level results of the NER sub-model are reported separately to demonstrate its effectiveness. More comprehensive comparisons with task-specific NLP-based secret-detection models and large language model-based extraction methods will be considered in future work.

5. Limitations

This section discusses the main limitations of SenScanner.

5.1. Password Availability Testing

SenScanner focuses on detecting sensitive information in mixed text, but it does not verify whether detected password-related secrets are valid or usable. Most authentication systems require both usernames and passwords [63]. Therefore, if only a password is detected without the corresponding username, the detected secret may not be directly usable, and its actual validity cannot be determined. Moreover, SenScanner does not attempt to validate detected password-related secrets because passwords are time-sensitive and may expire or be changed by users. Although validation could improve the practical usefulness of detection results, testing credentials against online systems is risky, may be unethical or illegal, and could affect normal users or services. Future work should investigate safe, privacy-preserving, and non-intrusive approaches for assessing the usability of detected password-related secrets.

Although the default configurations reflect common practical usage, different entropy thresholds, rule sets, or allowlists may affect the performance of existing tools. A systematic sensitivity analysis of baseline configurations will be considered in future work.

5.2. Data Authenticity

Real-world mixed-text datasets containing password-related sensitive information are difficult to obtain at scale because many real leakage records involve private credentials, personal information, internal logs, or repository histories that cannot be publicly released for ethical, legal, and security reasons. Therefore, the training and evaluation data used in this study were constructed through a combination of ChatGPT-generated

natural-language statements, transformed public password datasets, GitHub-based code samples, and manually designed hybridization. The construction process was examined by cybersecurity experts, who considered the generated samples to be reasonable approximations of typical password-leakage contexts.

However, we acknowledge that constructed datasets may still differ from real-world leakage distributions. In particular, ChatGPT-generated sentences may have limited stylistic diversity, and passwords injected from leaked datasets may not fully capture the contextual, linguistic, and structural characteristics of real issues, commits, logs, configuration files, or operational documents. As a result, the reported performance may partially reflect the constructed data distribution. Future work will focus on collecting or building privacy-preserving real-world mixed-text benchmarks and conducting additional stress tests across different writing styles, languages, domains, and leakage contexts to further evaluate the generalization ability of SenScanner.

In addition, although we performed file-level, snippet-level, and candidate-string-level de-duplication, complete repository-level separation may not be available for all public snippets because some datasets do not provide original repository metadata. This may still leave a small risk of distributional similarity between training and evaluation data. Future work will construct benchmarks with complete repository metadata and strict repository-level splitting.

Although the synthetic NER training data provides controllable labels and enables the model to learn common password-related contexts, it may still introduce distributional bias. ChatGPT-generated sentences may overrepresent fluent and regular English expressions, while real-world password leakage may appear in incomplete sentences, informal comments, multilingual text, logs, Markdown files, configuration documents, or noisy operational records. In addition, replacing placeholders with leaked passwords may not fully reproduce the way secrets are inserted, formatted, or obfuscated in real environments. Therefore, the synthetic data should be regarded as an approximation of natural-language password-leakage contexts rather than a complete substitute for real-world data. Future work will collect privacy-preserving real-world natural-language and mixed-text benchmarks, and will further evaluate SenScanner across more diverse writing styles, languages, and leakage contexts.

5.3. Practical Deployment Limitations

SenScanner still has several practical limitations. First, although the proposed pipeline is lightweight compared with large language model-based extraction methods, its scalability on extremely large repositories or real-time scanning scenarios has not been fully evaluated. Large-scale deployment may require additional engineering optimizations, such as incremental scanning, file-change tracking, parallel processing, and caching of intermediate detection results.

Second, SenScanner may have limited effectiveness when password-related secrets are intentionally obfuscated or hidden. For example, secrets may be split across multiple variables, encoded using Base64 or other transformations, concatenated at runtime, or stored in dynamically generated strings. Since the current implementation mainly analyzes visible textual content and line-level candidates, it may miss secrets that require deeper semantic analysis or runtime reconstruction.

Third, the generalization ability of the code-text detection component may be affected by programming languages and file formats that are underrepresented in the training data. Although SenScanner uses lightweight features that do not depend on language-specific parsers, different programming languages and configuration formats may follow different conventions for variable naming, assignment, comments, and string representation. Therefore, additional evaluation on more diverse languages and file types is needed.

Fourth, the natural-language NER component depends on the quality and diversity of the synthetic training data. ChatGPT-generated sentences provide controllable labels and useful training examples, but they may not fully capture noisy, multilingual, incomplete, or domain-specific natural-language leakage patterns in real-world documents, logs, comments, or issue discussions.

Finally, as an AI-based detection system, SenScanner may be affected by adversarial inputs. Attackers may deliberately modify the surrounding context, insert misleading keywords, use unusual formatting, or construct adversarial strings to evade the classifier or increase false positives. Future work will investigate adversarially robust training, multi-line and file-level context modeling, broader language and file-format coverage, and more efficient large-scale deployment strategies.

5.4. Ethical Considerations and Impact of False Positives

Research on password-related secret detection involves ethical and security concerns because real leaked credentials may expose users, organizations, or services to additional risks. Therefore, this study does not attempt to validate whether detected secrets are usable against real systems. The datasets and examples used in the paper are either publicly available, transformed, or synthetically constructed, and the reported examples are anonymized or illustrative. This design reduces the risk of exposing active credentials while still allowing the detection method to be evaluated.

False positives are another practical concern. Although false positives do not directly expose credentials, they may increase manual review effort and cause unnecessary alerts in real deployment. In large-scale repository scanning, excessive false positives may reduce user trust in the tool and increase the workload of security analysts. SenScanner reduces this problem through text-type classification, candidate filtering, context-aware features, and NER-based extraction, but false positives cannot be completely eliminated. Future work will explore risk-based ranking, confidence calibration, and user-feedback mechanisms to help prioritize detected secrets and reduce unnecessary manual inspection.

In addition, deployment in real software-development environments requires careful integration with access-control policies, privacy protection, and responsible disclosure procedures. Detected secrets should be handled securely, and scanning systems should avoid storing or exposing sensitive values unnecessarily. Future versions of SenScanner will further consider privacy-preserving reporting, secure logging, and deployment guidelines for practical use.

6. Conclusions

This paper presents SenScanner, an AI-based method for detecting password-related sensitive information in mixed text. To address the limitations of existing methods in handling code language and natural language simultaneously, SenScanner first distinguishes between the two text types and then applies type-specific detection models. For password-related secrets detection, SenScanner combines intrinsic features of password-like strings, structural features of code context, Named Entity Recognition, and machine-learning-based classification. Experimental results show that SenScanner outperforms open-source tools and traditional rule-based methods, achieving a precision of 84.40%, a recall of 83.76%, and an F1-score of 84.08%. These results demonstrate that SenScanner is effective for detecting password-like secrets in the constructed mixed-text benchmark, while further validation on real-world mixed datasets remains an important direction for future work.

Author Contributions: Conceptualization, Z.H., Y.G. and B.Z.; methodology, Z.H., Y.G. and B.Z.; software, Z.H.; validation, Z.H.; formal analysis, Z.H.; investigation, Z.H. and W.C.; writing—original draft, Z.H.; writing—review and editing, Z.H., Y.G., B.Z. and W.C.; visualization, Z.H.; and project administration, Z.H., Y.G., B.Z. and W.C. All authors have read and agreed to the published version of the manuscript.

Funding: Xiangjiang Laboratory Projects (25XJ03001, 25XJ03004, 25XJ03016, 25XJJB03, 24XJJCYJ01005); Hunan Provincial Natural Science Foundation (No.2025JJ60393); Changsha Natural Science Foundation (No.106271).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to sincerely thank Phytium Technology Co., Ltd. for its financial support of this research.

Conflicts of Interest: Author Bo Zhang was employed by Phytium Technology Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Zhang, Y.; Shabaz, M.; Beram, S.M. Information Leakage Tracking Algorithms in Online Social Networks. *Recent Adv. Electr. Electron. Eng. (Former. Recent Pat. Electr. Electron. Eng.)* **2023**, *16*, 126–134.
2. Applebaum, S.; Gaber, T.; Ahmed, A. Signature-based and machine-learning-based web application firewalls: A short survey. *Procedia Comput. Sci.* **2021**, *189*, 359–367.
3. Pingale, S.V.; Sutar, S.R. Analysis of Web Application Firewalls, Challenges, and Research Opportunities. In *ICDSMLA 2020: Proceedings of the 2nd International Conference on Data Science, Machine Learning and Applications*; Springer: Singapore, 2022; pp. 239–248.
4. Tsay, J.; Dabbish, L.; Herbsleb, J. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, 1–7 June 2014; pp. 356–366.
5. Github Kills Search After Hundreds of Private Keys Exposed. Available online: <https://it.slashdot.org/story/13/01/25/132203/github-kills-search-after-hundreds-of-private-keys-exposed> (accessed on 25 January 2013).
6. Aws Urges Devs to Scrub Secret Keys from Github. Available online: <https://developers.slashdot.org/> (accessed on 2 April 2022).
7. Seeja, K.R.; Zareapoor, M. Fraudminer: A novel credit card fraud detection model based on frequent itemset mining. *Sci. World J.* **2014**, *2014*, 252797.
8. Financial Times. Available online: <https://www.ft.com/content/87184c40-2cfe-11e8-9b4b-bc4b9f08f381> (accessed on 25 June 2026).
9. Hong, K.H.; Lee, B.M. A deep learning-based password security evaluation model. *Appl. Sci.* **2022**, *12*, 2404.
10. Kuka, E.; Bahiti, R. Information Security Management: Password Security Issues. *Acad. J. Interdiscip. Stud.* **2018**, *7*, 43.
11. Delnevo, G.; Deluigi, L.; Evangelisti, D.; Magnani, S. On increasing password security awareness using a serious game. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*; IEEE: New York, NY, USA, 2022; pp. 82–87.
12. Taneski, V.; Heričko, M.; Brumen, B. Systematic overview of password security problems. *Acta Polytech. Hung.* **2019**, *16*, 143–165.
13. About Secret Scanning. Available online: <https://docs.github.com/en/code-security/concepts/secret-security/secret-scanning> (accessed on 2 April 2022).
14. Vulnerabilities—Acunetix. Available online: <https://www.acunetix.com/vulnerabilities/web/> (accessed on 25 June 2026).
15. Detect-Secrets Release Version. Available online: <https://github.com/Yelp/detect-secrets/releases> (accessed on 6 October 2022).
16. Git-Secrets. Available online: <https://github.com/aws-labs/git-secrets> (accessed on 22 September 2023).
17. Shhgit. Available online: <https://github.com/eth0izzle/shhgit> (accessed on 22 September 2023).

18. Sinha, V.S.; Saha, D.; Dhoolia, P.; Padhye, R.; Mani, S. Detecting and mitigating secret-key leaks in source code repositories. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*; IEEE: New York, NY, USA, 2015; pp. 396–400.
19. Qu, J. Research on password detection technology of iot equipment based on wide area network. *ICT Express* **2022**, *8*, 213–219.
20. Meli, M.; McNiece, M.R.; Reaves, B. How bad can it git? characterizing secret leakage in public GitHub repositories. In *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium 2019*, San Diego, CA, USA, 24–27 February 2019.
21. Nguyen, A.T.; Nguyen, T.N. Automatic categorization with deep neural network for open-source java projects. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*; IEEE: New York, NY, USA, 2017; pp. 164–166.
22. Mou, L.; Li, G.; Jin, Z.; Zhang, L.; Wang, T. TBCNN: A tree-based convolutional neural network for programming language processing. *arXiv* **2014**, arXiv:1409.5718.
23. Gshark. Available online: <https://github.com/madneal/gshark/> (accessed on 25 June 2026).
24. Gsil. Available online: <https://github.com/FeeiCN/gsil> (accessed on 25 June 2026).
25. Gardner, B.; Long, J.; Brown, J. *Google Hacking for Penetration Testers*; Elsevier: Amsterdam, The Netherlands, 2011.
26. Nadeau, D.; Sekine, S. A survey of named entity recognition and classification. *Linguisticae Investig.* **2007**, *30*, 3–26.
27. Jordan, M.I.; Mitchell, T.M. Machine learning: Trends, perspectives, and prospects. *Science* **2015**, *349*, 255–260.
28. Ertel, W. *Introduction to Artificial Intelligence*; Springer: New York, NY, USA, 2018.
29. Morris, R.; Thompson, K. Password security: A case history. *Commun. ACM* **1979**, *22*, 594–597.
30. Feng, R.; Yan, Z.; Peng, S.; Zhang, Y. Automated detection of password leakage from public github repositories. In *Proceedings of the 44th International Conference on Software Engineering*, Pittsburgh, PA, USA, 22–27 May 2022; pp. 175–186.
31. Sibai, F.N.; Shehhi, A.; Shehhi, S.; Shehhi, B.; Salami, N. Secure password detection with artificial neural networks. In *2008 International Conference on Innovations in Information Technology*; IEEE: New York, NY, USA, 2008; pp. 628–632.
32. Vijaya, M.S.; Jamuna, K.S.; Karpagavalli, S. Password strength prediction using supervised machine learning techniques. In *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*; IEEE: New York, NY, USA, 2009; pp. 401–405.
33. Hitaj, B.; Gasti, P.; Ateniese, G.; Perez-Cruz, F. A deep learning approach for password guessing. In *Proceedings of the International Conference on Applied Cryptography and Network Security*; Springer International Publishing: Cham, Switzerland, 2019; pp. 217–237.
34. Melicher, W.; Ur, B.; Segreti, S.M.; Komanduri, S.; Bauer, L.; Christin, N.; Cranor, L.F. Fast, lean, and accurate: Modeling password guessability using neural networks. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, USA, 10–12 August 2016; pp. 175–191.
35. Joachims, T. *Making Large-Scale SVM Learning Practical*; Technical Report No. 1998,28; University Dortmund: Dortmund, Germany, 1998.
36. Quinlan, J.R. Induction of decision trees. *Mach. Learn.* **1986**, *1*, 81–106.
37. Lewis, D.D. Naive (Bayes) at forty: The independence assumption in information retrieval. In *European Conference on Machine Learning*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 4–15.
38. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32.
39. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*.
40. Jackins, V.; Vimal, S.; Kaliappan, M.; Lee, M.Y. AI-based smart prediction of clinical disease using random forest classifier and Naive Bayes. *J. Supercomput.* **2021**, *77*, 5198–5219.
41. Gu, J.; Lu, S. An effective intrusion detection approach using SVM with naïve Bayes feature embedding. *Comput. Secur.* **2021**, *103*, 102158.
42. Itoo, F.; Meenakshi; Singh, S. Comparison and analysis of logistic regression, Naïve Bayes and KNN machine learning algorithms for credit card fraud detection. *Int. J. Inf. Technol.* **2021**, *13*, 1503–1511.
43. Sheykhmousa, M.; Mahdianpari, M.; Ghanbari, H.; Mohammadimanesh, F.; Ghamisi, P.; Homayouni, S. Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2020**, *13*, 6308–6325.
44. Chowdhary, K.R.; Chowdhary, K.R. Natural language processing. In *Fundamentals of Artificial Intelligence*; Springer, New Delhi, India, 2020; pp. 603–649.
45. Ehrmann, M.; Hamdi, A.; Pontes, E.L.; Romanello, M.; Doucet, A. Named entity recognition and classification in historical documents: A survey. *ACM Comput. Surv.* **2023**, *56*, 1–47.

46. Wang, S.; Sun, X.; Li, X.; Ouyang, R.; Wu, F.; Zhang, T.; Li, J.; Wang, G.; Guo, C. Gpt-ner: Named entity recognition via large language models. *arXiv* **2023**, arXiv:2304.10428.
47. Fields, S.; Cole, C.L.; Oei, C.; Chen, A.T. Using named entity recognition and network analysis to distinguish personal networks from the social milieu in nineteenth-century Ottoman–Iraqi personal diaries. *Digit. Scholarsh. Humanit.* **2023**, *38*, 66–86.
48. Lison, P.; Hubin, A.; Barnes, J.; Touileb, S. Named entity recognition without labelled data: A weak supervision approach. *arXiv* **2020**, arXiv:2004.14723.
49. Yu, J.; Bohnet, B.; Poesio, M. Named entity recognition as dependency parsing. *arXiv* **2020**, arXiv:2005.07150.
50. Patel, A.; Meehan, K. Fake news detection on reddit utilising CountVectorizer and term frequency-inverse document frequency with logistic regression, MultinomialNB and support vector machine. In *2021 32nd Irish Signals and Systems Conference (ISSC)*; IEEE: New York, NY, USA, 2021; pp. 1–6.
51. Turki, T.; Roy, S.S. Novel hate speech detection using word cloud visualization and ensemble learning coupled with count vectorizer. *Appl. Sci.* **2022**, *12*, 6611.
52. Maulana, A.; Afifah, I.K.; Mubarrak, A.; Fauzan, K.R.; Dwintara, A.; Zen, B.P. Comparison of Logistic Regression, Multinomialnb, Svm, and K-Nn Methods on Sentiment Analysis of Gojek App Reviews on the Google Play Store. *J. Tek. Inform. (JUTIF)* **2023**, *4*, 1487–1494.
53. Wikipedia. Available online: <https://en.wikipedia.org/wiki/> (accessed on 22 September 2023).
54. LinkedIn Leaked Passwords. Available online: <https://hashes.org/public.php> (accessed on 22 September 2023).
55. Liu, Z.; Jiang, F.; Hu, Y.; Shi, C.; Fung, P. NER-BERT: A pre-trained model for low-resource entity tagging. *arXiv* **2021**, arXiv:2112.00405.
56. Shelar, H.; Kaur, G.; Heda, N.; Agrawal, P. Named entity recognition approaches and their comparison for custom ner model. *Sci. Technol. Libr.* **2020**, *39*, 324–337.
57. ChatGpt. Available online: <https://chat.openai.com> (accessed on 22 September 2023).
58. Clark, K.; Luong, M.T.; Le, Q.V.; Manning, C.D. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv* **2020**, arXiv:2003.10555.
59. Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; Bowman, S. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv* **2018**, arXiv:1804.07461.
60. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
61. Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R.R.; Le, Q.V. Xlnet: Generalized autoregressive pretraining for language understanding. *Adv. Neural Inf. Process. Syst.* **2019**, *32*.
62. Trufflehog. Available online: <https://github.com/dxa4481/truffleHog> (accessed on 22 September 2022).
63. Conklin, A.; Dietrich, G.; Walz, D. Password-based authentication: A system perspective. In *37th Annual Hawaii International Conference on System Sciences*; IEEE: New York, NY, USA, 2004; 10p.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.