INTERNATIONAL JOURNAL OF INFORMATION SECURITY SCIENCE
M. K. Pehlivanoğlu et al., Vol.14, No.1, pp.1-12
https://doi.org/10.55859/ijiss.1634763

Detection of Malicious Codes Generated by Large Language Models: A Comparison of GPT-3.5, GPT-4o, Gemini, and Claude

Meltem Kurt Pehlivanoğlu[1], Murat Görkem Çoban[1]

[1]Department of Computer Engineering, Kocaeli University, Kocaeli, Turkey
Corresponding Author: meltem.kurt@kocaeli.edu.tr

Research Paper Received: 06.02.2025 Revised: 24.02.2025 Accepted: 15.03.2025

Abstract—This study presents novel machine learning-based approaches for detecting whether source code generated by Large Language Models (LLMs) contains malicious code. To achieve this, comprehensive datasets comprising malicious and benign code samples were created using the GPT-3.5 (ChatGPT), GPT-4o, Gemini, and Claude language models. The extracted code samples were then processed through CodeBERT, CodeT5, and manual feature extraction techniques before being classified using various machine learning algorithms. Experimental results demonstrate that this approach can effectively detect malicious software in code generated by LLMs. This study makes contributions to software security and represents a crucial step toward preventing the misuse of LLMs for malicious purposes. Moreover, the Random Forest algorithm for binary malicious code classification in LLM-generated code achieved the best F_1 score of 94.92% on the ChatGPT-generated dataset (with CodeT5 feature extraction technique). We also showed that the classification models exhibited poor performance on the dataset generated by Claude language model.

Keywords—Large Language Models, Malware Detection, Malicious Code, Machine Learning, CodeBERT, CodeT5, Artificial Intelligence Security

1. Introduction

Large Language Models (LLMs) have brought transformative changes to software development processes. They are now widely used for tasks such as source code generation, code completion, and code refinement. However, as LLMs become more proficient in generating code, concerns have arisen regarding their potential to produce malicious code. This has underscored the need to develop new methods to detect and prevent LLM-generated malicious code.

In this study, we analyze whether the source code generated by large language models, including OpenAI's Generative Pretrained Transformer (GPT) models ChatGPT [1], GPT-4o [2], as well as Gemini 1.5 Flash [3] and the Anthropic's Claude-3.5 Sonnet [4], is malicious or benign. This analysis represents a significant step toward improving software security and preventing the misuse of LLMs for generating malicious code.

1.1. Motivation and Contribution

Software security is one of the most critical concerns in today's technology landscape. With the widespread adoption of LLMs, evaluating the security of the code they generate has become increasingly important. The primary motivation of this study is to perform a comprehensive security analysis of the code produced by popular LLMs, including ChatGPT, GPT-4o, Gemini, and Claude. To our knowledge, this is the first preliminary study exploring the potential use of these LLMs for the generation of malicious code in the Python programming language.

This paper addresses two key research questions: "How capable are LLMs of generating malicious code when prompted?" and "Can LLM-generated malicious code be detected using machine learning algorithms?"

The summarization of the main contributions is as follows:

- A comparative analysis of the code generation capabilities of different LLMs from a security perspective has been conducted.
- Novel four different datasets containing both malicious and benign source codes generated by ChatGPT, GPT-4, Gemini, and Claude LLMs have been generated.
- Different binary classification approaches have been proposed using CodeBERT [5] and CodeT5 [6] as well as manual methods as feature extraction models.
- We have made the extracted features of the source codes and the source code for the binary classification models publicly available at https://github.com/MGCoban/LLM-MalCode-Detection.

1.2. Organization

We give related works in Section 2. Section 3 presents all the steps generating the data set. In Section 4, we give all the details of the proposed methodology, and the experimental results are presented in Section 5. Finally, we conclude the paper in Section 6.

2. Related Works

In this section, the related works given in the literature are addressed under two subheadings: detecting whether source code is malicious and extracting the features of source codes.

2.1. Studies on the Detection of Malicious and Benign Source Code

In [7], GPT-2 model is used for malware detection. Static analysis of PE (Portable Executable) format files is performed, and the processed code is fed into the GPT-2 model. The model is designed to capture the features of both malicious and benign source codes. Experimental results show that the proposed model successfully detects malware with an accuracy rate of 85.4%.

In [8], the potential use of GPT-3 by attackers to generate malicious software is investigated. The experimental results show that while GPT-3 struggles to generate fully functional malware examples, it can create malware using building block definitions. The study also highlights that GPT-3 is capable of producing multiple variants of malware with the same semantics, and the detection rates of these variants vary.

Monje et al. [9] explore the potential misuse of GPT 3.5 (ChatGPT) to generate malicious software. The study demonstrates that by directing ChatGPT with seemingly innocuous commands, it is possible to bypass its built-in ethical constraints and create ransomware.

Iqbal et al. [10] investigate the malicious uses of ChatGPT, presenting attack scenarios and usage examples. The experiments demonstrate that ChatGPT can be used for malicious activities such as generating malware, preparing phishing emails, and scanning for system vulnerabilities.

Begou et al. [11] explore the potential use of ChatGPT for developing advanced phishing attacks and automating large-scale deployments. The study demonstrates that ChatGPT can perform functions such as copying target websites, integrating code to steal credentials, hiding code, automating distribution, registering phishing domains, and integrating with reverse proxies. This indicates that ChatGPT is capable of bypassing security measures against malicious use and enabling the rapid creation and deployment of phishing sites.

Pa Pa et al. [12] explore the potential of ChatGPT, the "text-davinci-003" model from OpenAI Playground, and Auto-GPT for malware development. The study demonstrates that with these tools, seven malware programs and two attack tools were developed. It also shows that, despite OpenAI's security controls, functional malware and attack tools can be produced within 90 minutes. Although Auto-GPT struggles to generate correct commands, it is capable of bypassing OpenAI's security checks. The study emphasizes that current antivirus and endpoint detection solutions are insufficient for detecting AI-generated malware, highlighting the need for improvements in the security controls of ChatGPT and the text-davinci-003 models.

2.2. Studies on Feature Extraction of Source Code

In [13], the "Large Movie Review Dataset," consisting of movie reviews, is used to test the BERT model against traditional machine learning models. In this study, TF-IDF and BERT-based feature extraction methods are employed. When compared with models such as Logistic Regression, Linear SVC, Multinomial Naïve Bayes, Ridge Classifier, and Passive Aggressive Classifier, the BERT model demonstrates the highest performance with an accuracy rate of 93.87%.

In [14], the COPER dataset, consisting of Iranian user comments and news content, is used. One-hot encoding and sentence transformers are chosen for feature extraction. Various deep learning models Dense Neural Network (DNN), Long Short Term Memory (LSTM), CNN + LSTM) are tested across different categories, and in the economy category, the CNN + LSTM + Dense model achieves the best accuracy (77%), precision (79%), and recall (75%).

In [15], source code classification is performed on code files obtained from 293,319 GitHub repositories. For feature extraction, unigram, bigram, and trigram methods are used. In this study, where the Maximum Entropy and Naive Bayes models are compared, the Maximum Entropy model demonstrates superior performance with an accuracy rate of 99.1%.

In [16], the Wikipedia Huggingface dataset is used to differentiate between machine-generated and human-written texts. Count Vectorizer is used for feature extraction, and the Multinomial Naive Bayes algorithm is chosen for classification. As a result, the proposed model successfully identifies the author of the texts with an accuracy of 86% and an F1 score of 83%.

In [17], various natural language processing techniques are applied for the automatic classification of PDF documents. In the study, Part-of-Speech (POS) based features are extracted using ECSS PDF documents. Models such as Naive Bayes, SVM, RF, and Feedforward Neural Network (FNN) are tested, with the highest accuracy rate (98.2%) achieved by the Decision Tree (DT) model.

Upon reviewing the existing literature, no study specifically addresses two key research questions: "How capable are LLMs of generating malicious

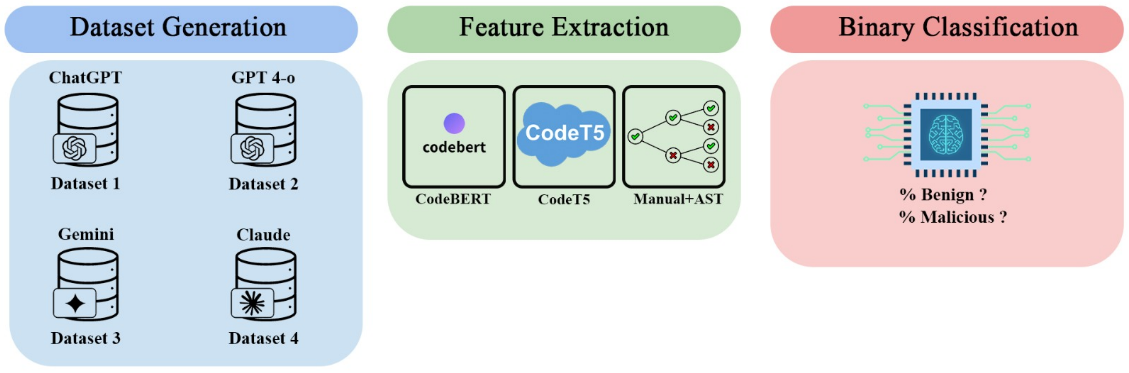

Figure 1. General System Architecture

code when prompted?" and "Can LLM-generated malicious code be detected using machine learning algorithms?" Furthermore, while some studies have attempted to generate malicious code, none have used multiple LLMs to generate both malicious and benign Python source code and performed binary classification of these code snippets using various feature extraction methods.

3. Proposed Datasets

Figure 1 illustrates the general system overview. First, we generate all datasets, which include both malicious and benign source codes, and then extract the features of these source codes using the CodeT5, CodeBERT, and Manual + AST methods. Next, we apply binary classification models to classify the code snippets.

In this study, four different datasets—Dataset 1, Dataset 2, Dataset 3, and Dataset 4—are created using ChatGPT, GPT-4o, Gemini, and Claude, respectively. All these datasets contain both malicious and benign source codes. For the malicious code snippet class, ransomware, keyloggers, screenshot grabbers, remote access tools (Remote Access Trojan—RAT), software for controlling webcams and microphones, software for accessing social media accounts, adware, worms, and trojan malware were generated. In the generation of benign code snippets, code snippets that use the same libraries as those used in the generated malicious code snippets but produce benign outputs were created by providing the same prompts to the large language models.

Table 1 presents the number of malicious and benign source code snippets in each dataset. According to the table, Dataset 1, generated by ChatGPT, comprises 65 malicious and 65 benign code snippets. Dataset 2, produced by GPT-4o, consists of 50 malicious and 50 benign code snippets. Dataset 3, created by Gemini, includes 50 malicious and 60 benign code snippets, while Dataset 4, generated by Claude, contains 45 malicious and 45 benign code snippets. The reason for the different number of samples in some classes of the datasets is that the malicious code was not generated using that specific large language model.

Table 1.
Detailed information on malicious/benign source codes in each dataset.

Type	**#of samples (Dataset 1 generated by ChatGPT)**	**#of samples (Dataset 2 generated by GPT-4o)**	**#of samples (Dataset 3 generated by Gemini)**	**#of samples (Dataset 4 generated by Claude)**
Trojan	40	27	30	26
Ransomware	3	3	3	3
Spyware	17	15	15	11
Adware	5	5	2	5
Benign	65	50	60	45

4. Methodology

4.1. Evaluation Metrics

We use accuracy, precision, recall, and F_1 score metrics to evaluate the performance of different models in detecting malicious code. True Positive (TP) refers to cases where the model correctly identifies a malicious code snippet as malicious, while True Negative (TN) represents instances where benign code is accurately classified as benign. False Positive (FP) occurs when the model incorrectly flags benign code as malicious, leading to false alarms, whereas False Negative (FN) happens when the model fails to detect a malicious code snippet and mistakenly classifies it as benign.

$$\text{Precision (P)} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

$$\text{Recall (R)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

$$F_1 = \frac{2 \times \text{P} \times \text{R}}{\text{P} + \text{R}} \tag{3}$$

4.2. Feature Extraction

In this study, CodeT5, CodeBERT, AST, and Manual features methods are used for the feature extraction. During the manual feature extraction process, seven distinct features were identified and utilized to differentiate between malicious and benign code samples. The extracted features are detailed in below:

- CodeBERT Features: The embedding vectors are obtained using CodeBERT for the code snippets.
- Manual Features:
  - Code Length: The total number of characters in the code snippet is calculated.
  - Unique Token Count: The number of unique words (tokens) used in the code snippet is determined.
  - Line Count: The total number of lines in the code is analyzed.
  - Complexity Score: The number of opening and closing parentheses (, (), []) is counted to generate a complexity score.
  - Function Count: The total number of function definitions in the code (e.g., “def” or “function”) is identified.
  - Library Import Count: The total number of libraries imported in the code is analyzed.

Table 2.
Model performance comparison on Dataset 1.

	CodeT5			**CodeBERT**			**Manual+AST**		
Models	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**
GaussianNB	0.8469	0.7366	1.0000	0.7009	0.7000	0.7500	0.5203	0.6800	0.4667
LR	0.9269	0.8699	0.8900	0.8920	0.9099	0.9000	0.7080	0.7833	0.6833
KNN	0.7861	0.9500	0.6833	0.7176	0.8166	0.6833	0.7159	0.7933	0.6833
DT	0.6176	0.5700	0.6833	0.5452	0.7166	0.4666	0.5810	0.6667	0.5833
SVM	0.9206	0.9099	0.9500	0.8714	0.9000	0.8500	0.7060	0.6966	0.7500
RF	0.8984	0.9199	0.9000	0.8869	0.9199	0.8833	0.6355	0.7800	0.5833
GB	0.6857	0.7000	0.6833	0.6211	0.7100	0.5833	0.6088	0.7133	0.5833
AdaBoost	0.7936	0.8699	0.8000	0.6687	0.7433	0.6333	0.6522	0.6533	0.7500

  – Arithmetic Operator Count: The total number of arithmetic operators (such as addition (+), subtraction (-), multiplication (*), and division (/)) in the code is calculated.

- AST (Abstract Syntax Tree) Features: The number of loops, conditionals, and functions extracted from the AST analysis.
- CodeT5 Features: CodeT5 represents code snippets (code fragments) as high-dimensional embedding vectors.

For CodeBERT and CodeT5, 767 features were extracted for each dataset, while 9 features were extracted using the Manual + AST method. We applied the CodeBERT, CodeT5, and Manual + AST methods separately for feature extraction on each dataset and then compared the performance of all these methods.

4.3. Models

In this study, various machine learning algorithms were applied to classify malicious code in a binary manner. To ensure a balanced distribution during the training and evaluation process, the stratified k-fold cross-validation method was employed, with the value of k set to 5. This preserved the ratio of malicious to benign code samples in each fold.

For hyperparameter optimization, the grid search technique was utilized, allowing the identification of optimal hyperparameter combinations for each model. Eight machine learning classifiers, including Gaussian Naive Bayes (GaussianNB), Logistic Regression (LR), k-Nearest Neighbors (KNN), DT, SVM, Random Forest (RF), Gradient Boosting (GB), and Adaptive Boosting (AdaBoost), were tested. By testing a variety of classifiers, we aim to identify the most suitable model for given datasets. GaussianNB is efficient for high-dimensional data and probabilistic classification, while LR is well-suited for linear relationships. KNN is effective for nonlinear decision boundaries, whereas DT provides interpretability and easy visualization. Ensemble methods like RF, GB, and AdaBoost enhance predictive performance by combining multiple weak learners to create a stronger model, reducing overfitting and improving generalization.

5. Experimental Results

All primary models were evaluated on Dataset 1 using precision, recall, and F_1 score, and their performance results are provided in Table 2. Among these models, the three best-performing models, SVM, LR, and RF, were selected, and the results of

Table 3.
Hyperparameters used in each classification algorithm for datasets.

Dataset	Models	Feature Extraction Method	Hyperparameters
Dataset 1	LR	CodeBERT	C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 0.1, 'gamma': 'scale', 'kernel': 'linear'
	RF		max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100
	LR	CodeT5	C': 1, 'penalty': 'l2', 'solver': 'liblinear'
	SVM		C': 1, 'gamma': 'scale', 'kernel': 'rbf'
	RF		max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100
	RF	Manual+AST	max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200
	LR		C': 1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 10, 'gamma': 'scale', 'kernel': 'rbf'
Dataset 2	LR	CodeBERT	C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'
	RF		max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50
	LR	CodeT5	C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'
	RF		max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200
	RF	Manual+AST	max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200
	LR		C': 1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'
Dataset 3	LR	CodeBERT	C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 1, 'gamma': 'auto', 'kernel': 'rbf'
	RF		max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100
	LR	CodeT5	C': 1, 'penalty': 'l2', 'solver': 'liblinear'
	SVM		C': 1, 'gamma': 'scale', 'kernel': 'rbf'
	RF		max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100
	RF	Manual+AST	max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200
	LR		C': 10, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 1, 'gamma': 'auto', 'kernel': 'rbf'
Dataset 4	LR	CodeBERT	C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 1, 'gamma': 'scale', 'kernel': 'rbf'
	RF		max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100
	LR	CodeT5	C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'
	RF		max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100
	RF	Manual+AST	max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50
	LR		C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
	SVM		C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'

the other datasets (Dataset 2, Dataset 3, and Dataset 4) were evaluated using just these three models.

In Table 3, we give the hyperparameters used in each model for Dataset 1, Dataset 2, Dataset 3, and Dataset 4.

Table 4.
Precision metric results for each dataset

Dataset	Models	CodeT5	CodeBERT	Manual+AST
Dataset 1	SVM	0.9099	0.9000	0.7700
	LR	0.8699	0.9099	0.7833
	RF	0.9099	0.9199	0.7800
Dataset 2	SVM	0.7507	0.5557	0.5557
	LR	0.7691	0.6752	0.5964
	RF	0.6733	0.5964	0.6571
Dataset 3	SVM	0.7216	0.7027	0.7428
	LR	0.6725	0.6178	0.7998
	RF	0.7214	0.7105	0.7355
Dataset 4	SVM	0.5419	0.6690	0.5419
	LR	0.5571	0.5674	0.7000
	RF	0.6448	0.6639	0.6600

Table 4 presents the precision scores of the models trained on generated datasets. The performance of SVM, LR, and RF was compared across CodeT5, CodeBERT, and Manual+AST feature combinations.

For Dataset 1, the SVM model achieves 90.99% precision with CodeT5 features and 90% precision with CodeBERT features. LR shows 86.99% precision with CodeT5 and 90.99% with CodeBERT. RF achieves the highest precision, with 90.99% for CodeT5 and 91.99% for CodeBERT. The precision values for all models using Manual+AST features are lower (around 77-78%), indicating that CodeT5 and CodeBERT features are more effective for Dataset 1. For Dataset 2, the precision values show a general decline. SVM achieves 75.07% precision with CodeT5, while it drops to 55.57% with CodeBERT. LR's precision ranges from 76.91% with CodeT5 to 67.52% with CodeBERT. RF shows the lowest precision values. This suggests that Dataset 2 is more challenging compared to the other datasets, GPT-4o model has been more successful at producing secure and benign code structures. For Dataset 3, SVM achieves 72.16% precision with CodeT5 and 70.27% with CodeBERT. LR shows 67.25% precision with CodeT5, while it increases to 79.98% with Manual+AST features. RF's precision values range from 72.14% with CodeT5 to 71.05% with CodeBERT. This suggests that Manual+AST features are more suitable for Dataset 3, and LR performs better with these features. For Dataset 4, the precision values are the lowest. SVM achieves 54.19% precision with CodeT5, and 66.90% with CodeBERT. Logistic Regression shows precision between 55.71% (CodeT5) and 56.74% (CodeBERT). Random Forest achieves precision of 64.48% with CodeT5 and 66.39% with CodeBERT. This indicates that Dataset 4 is more complex than the other datasets, and Claude has been more successful in creating secure and benign code by taking safety measures into account. Machine learning models struggle to classify this dataset as malicious, further supporting the idea that Claude can create more security-aware code.

Table 5 compares the recall scores for each model. For Dataset 1, the SVM model achieves a recall of 94% with CodeT5 features, which decreases to 85% with CodeBERT features. The LR model demonstrates a high recall of 95% with CodeT5, but this

Table 5.
Recall metric results for each dataset

Dataset	Models	CodeT5	CodeBERT	Manual+AST
	SVM	0.9400	0.8500	0.7500
Dataset 1	LR	0.9500	0.9000	0.6833
	RF	0.9099	0.8833	0.5833
	SVM	0.7272	0.5636	1.000
Dataset 2	LR	0.6727	0.6363	0.6727
	RF	0.7818	0.7272	0.6545
	SVM	0.8163	0.7400	0.8527
Dataset 3	LR	0.7381	0.7418	0.7963
	RF	0.8527	0.7600	0.7600
	SVM	0.5107	0.7964	1.0000
Dataset 4	LR	0.5678	0.6178	0.6392
	RF	0.6928	0.7428	0.7464

drops to 90% with CodeBERT. The RF algorithm attains a recall of 90.99% with CodeT5 and 88.33% with CodeBERT. These results indicate that CodeT5 features are more effective for detecting malicious code in Dataset 1. Furthermore, the Manual+AST features yield the lowest recall values across all models, highlighting their inadequacy for Dataset 1. In Dataset 2, the SVM model shows a recall of 72.72% with CodeT5, which decreases to 56.36% with CodeBERT. The LR model achieves a recall of 67.27% with CodeT5, maintaining 63.63% with CodeBERT. The RF algorithm demonstrates a recall of 78.18% with CodeT5, but this drops to 72.72 with CodeBERT. For Dataset 3, the SVM model achieves recall values of 81.63% with CodeT5 and 74.00% with CodeBERT. The LR model demonstrates a recall of 73.81% with CodeT5, which increases slightly to 74.18% with CodeBERT. The RF algorithm reaches a high recall of 85.27% with CodeT5 but drops to 76.00% with CodeBERT. These results suggest that CodeT5 and CodeBERT features show similar performance for Dataset 3, with the RF model being the most effective model in detecting malicious code for this dataset. In Dataset 4, the recall values are generally lower. The SVM model shows a recall of 51.07% with CodeT5, which increases to 79.64% with CodeBERT. The LR model attains a recall of 56.78% with CodeT5, rising to 61.78% with CodeBERT. The RF algorithm achieves a recall of 69.28% with CodeT5 and 74.28% with CodeBERT. These findings suggest that Claude is more successful in concealing the characteristic features of malicious code, making it harder to detect. This indicates that the LLM is more adept at generating safer code.

Table 6 presents the F_1 scores for each model. For Dataset 1, the SVM model achieves an F_1 score of 92.06% with CodeT5 features, while it decreases to 87.14% with CodeBERT features. The LR model demonstrates a high F_1 score of 92.69% with CodeT5, but it drops to 89.20% with CodeBERT. The RF algorithm achieves the highest F_1 score of 94.92% with CodeT5 and drops to 88.69% with CodeBERT. The Manual+AST features result in the lowest F_1 score for all models, indicating that these features are less effective for Dataset 1. These results suggest that CodeT5 features, particularly with the RF algorithm, perform best for malware

Table 6.
F_1 score metric results for each dataset

Dataset	Models	CodeT5	CodeBERT	Manual+AST
Dataset 1	SVM	0.9206	0.8714	0.7060
	LR	0.9269	0.8920	0.7080
	RF	0.9492	0.8869	0.6355
Dataset 2	SVM	0.7303	0.7144	0.7144
	LR	0.7072	0.6337	0.6169
	RF	0.7205	0.6491	0.6456
Dataset 3	SVM	0.7647	0.7200	0.7932
	LR	0.6975	0.6722	0.7966
	RF	0.7806	0.7330	0.7451
Dataset 4	SVM	0.7025	0.7222	0.7025
	LR	0.5591	0.5865	0.6649
	RF	0.6635	0.6972	0.6967

detection in Dataset 1. In Dataset 2, F_1 scores for the models generally show a decrease. The SVM model achieves an F_1 score of 73.03% with CodeT5 features and 71.44% with CodeBERT. The LR model shows an F_1 score of 70.72% with CodeT5, which decreases to 63.37% with CodeBERT. The RF algorithm reaches an F_1 score of 72.05% with CodeT5 and drops to 64.91% with CodeBERT. This trend indicates that the malicious software in Dataset 2 is more complex and harder to detect. Although a decrease is observed, CodeT5 features still perform better than CodeBERT features in this dataset. For Dataset 3, the SVM model achieves F_1 score of 76.47% with CodeT5, 72.00% with CodeBERT, and 79.32% with Manual+AST. The LR model shows F_1 score of 69.75% with CodeT5, 67.22% with CodeBERT, and 79.66% with Manual+AST. The RF algorithm reaches F_1 score of 78.06% with CodeT5, 73.30% with CodeBERT, and 74.51% with Manual+AST. These results suggest that Manual+AST features are more suitable for Dataset 3, with SVM and LR models performing better with these features. For Dataset 4, the F_1 score of the models are lower compared to other datasets. The SVM model achieves F_1 score of 70.25% with CodeT5, 72.22% with CodeBERT, and 70.25% with Manual+Ast. The LR model demonstrates F_1 score of 55.91% with CodeT5, 58.65% with CodeBERT, and 66.49% with Manual+AST. The RF algorithm reaches F_1 score of 66.35% with CodeT5, 69.72% with CodeBERT, and 69.67% with Manual+AST.

From the malicious code detection perspective, recall and F_1 score metrics are crucial. While the experimental results are evaluated, if one of these scores is low, this is a critical concern, as it means that the malicious code is not being detected effectively. This implies a major risk that LLM-generated malicious code is undetectable. Meanly, the fine-tuned LLM can generate malicious code that bypasses security mechanisms, which poses a significant threat. If an AI-generated source code snippet is not labeled as malicious, it suggests that attackers could exploit such models to create harmful software that evades detection. Dataset 4, which includes Claude-generated source code, had the lowest recall and F_1 scores, meaning the generated malicious codes were the hardest to detect.

Moreover, SVM and LR struggled more compared to RF. From the feature extraction perspective, CodeBERT performed better in some cases, meaning that different feature representations impact the model's ability to catch threats.

6. Conclusion

A low malicious code detection score means this malicious code is actively being missed. That underscores the need to improve detection models and highlights the potential risk of LLMs being exploited to generate undetectable malicious code. In this paper, we compare this potential risk of ChatGPT, GPT4-o, Gemini, and Claude language models. Our findings indicate that while the malicious codes generated by ChatGPT can be successfully detected, it is hard to detect malicious codes generated by Claude.

One of the limitations of this study is the need to increase the number of samples for each class and design balanced datasets. Expanding the dataset is crucial for improving the performance of multi-class classification models, as a larger and more balanced dataset can enhance model generalization and reduce bias. Additionally, the datasets used in this study contain a limited number of malicious source code samples, necessitating future testing on much larger datasets to ensure robustness and real-world applicability. However, despite these limitations, this study serves as a guiding use case, as it is one of the pioneering efforts in this field. In future work, we plan to focus on collecting and incorporating more diverse and extensive datasets, exploring advanced data augmentation techniques, and experimenting with deep learning-based approaches to further enhance classification performance.

Furthermore, integrating additional LLM models and exploring more advanced feature extraction techniques, such as deeper semantic analysis or neural network-based methods, to better capture the complex characteristics of malicious code and enhance the overall performance of the detection models will be part of future work. In addition, further investigations are needed to analyze why certain malicious code samples evade detection. Techniques like explainable AI (XAI) could be used to understand misclassified cases and improve detection models.

Acknowledgment

MGÇ and MKP provided software and contributed to data curation. MKP was involved in supervision, conceptualization, methodology, writing, reviewing, and editing. All authors have reviewed and approved the submitted manuscript.

References

[1] J. Ye, X. Chen, N. Xu, C. Zu, Z. Shao, S. Liu, Y. Cui, Z. Zhou, C. Gong, Y. Shen *et al.*, "A comprehensive capability analysis of gpt-3 and gpt-3.5 series models," *arXiv preprint arXiv:2303.10420*, 2023.

[2] R. Islam and O. M. Moushi, "Gpt-4o: The cutting-edge advancement in multimodal llm," *Authorea Preprints*, 2024.

[3] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[4] J. Bae, S. Kwon, and S. Myeong, "Enhancing software code vulnerability detection using gpt-4o and claude-3.5 sonnet: A study on prompt engineering techniques," *Electronics*, vol. 13, no. 13, p. 2657, 2024.

[5] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[6] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," *arXiv preprint arXiv:2109.00859*, 2021.

[7] N. Şahin, "Malware detection using transformers-based model gpt-2," Master's thesis, Middle East Technical University, 2021.

[8] M. Botacin, "Gpthreats-3: Is automatic malware generation a threat?" in *2023 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2023, pp. 238–254.

[9] A. Monje, A. Monje, R. A. Hallman, and G. Cybenko, "Being a bad influence on the kids: Malware generation in less than five minutes using chatgpt," 2023, unpublished.

[10] F. Iqbal, F. Samsom, F. Kamoun, and Á. MacDermott, "When chatgpt goes rogue: exploring the potential cybersecurity threats of ai-powered conversational chatbots," *Frontiers in Communications and Networks*, vol. 4, p. 1220243, 2023.

[11] N. Begou, J. Vinoy, A. Duda, and M. Korczyński, "Exploring the dark side of ai: Advanced phishing attack design and deployment using chatgpt," in *2023 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2023, pp. 1–6.

[12] Y. M. Pa Pa, S. Tanizaki, T. Kou, M. Van Eeten, K. Yoshioka, and T. Matsumoto, "An attacker's dream? exploring the capabilities of chatgpt for developing malware," in *Proceedings of the 16th Cyber Security Experimentation and Test Workshop*, 2023, pp. 10–18.

[13] S. González-Carvajal and E. C. Garrido-Merchán, "Comparing bert against traditional machine learning text classification," *arXiv preprint arXiv:2005.13012*, 2020.

[14] K. Mohammadi, "Human vs machine generated text detection in persian," 2023.

[15] S. Zevin and C. Holzem, "Machine learning based source code classification using syntax oriented features," *arXiv preprint arXiv:1703.07638*, 2017.

[16] A. Bhandarkar, M. A. DM, D. Vishwachetan, A. Mushtaq, D. Kadam, and S. Saxena, "Unmasking the ai hand: A machine learning approach to deciphering authorship," in *2024 3rd International Conference for Innovation in Technology (INOCON)*. IEEE, 2024, pp. 1–6.

[17] N. Abdoun and M. Chami, "Automatic text classification of pdf documents using nlp techniques," in *INCOSE International Symposium*, vol. 32, no. 1. Wiley Online Library, 2022, pp. 1320–1331.