Whodunit: Classifying Code as Human Authored or GPT-4 Generated — A Case Study on CodeChef Problems

Presenter: Kevin Jie

Accepted by 21st Mining Software Repositories (MSR)

Background & Motivation

```python
for _ in range(int(input())):
l, r = map(int, input().split())
result = 0
for i in range(l, r + 1):
    if str(i) == str(i)[::-1]:
        result += i
print(result)
```

(a) Example of human code

```python
def is_palindrome(n):
    return str(n) == str(n)[::-1]

def palindromic_numbers_sum(l, r):
    total = 0
    for n in range(l, r+1):
        if is_palindrome(n):
            total += n
    return total

t = int(input())

for i in range(t):
    l, r = map(int, input().split())
    result = palindromic_numbers_sum(l, r)
    print(result)
```

(b) Example of ChatGPT code

Background & Motivation

- **AI coding assistants (Copilot, ChatGPT, etc.) widely used**
- **Motivation:**
  - Instructors concern students cheating by using generative-AI tools for their assignments in an **introductory** programming course.
- **Traditional plagiarism detectors (e.g., MOSS) rely on similarity**
  - But LLM code often shows low similarity → hard to flag
- **Stylometry: the statistical analysis of variations in literary style between one writer or genre and another**
  - Layout, lexical, syntactic, complexity metrics

Existing Solutions & Related Works

- **Prior studies:**
  - Bukhari et al. attempt to use machine learning to distinguish between 28 student-authored and 30 AI-generated solutions for a C-language programming assignment.
  - Their approach leverages **lexical and syntactic features** in conjunction with multiple **machine-learning models**, achieving an **accuracy rate of 92%**.
- **Commercial AI-detection (e.g. HankerRank, CoderByte) tools lack independent validation**
- **Need for:**
  - Scalable dataset of human + LLM code
  - Robust, interpretable models
  - Tests across difficulty levels & solution correctness

Research Questions

- **RQ1:** How well can code-stylometry features distinguish human-authored code from GPT-4 generated code?
  - ***Inherent characteristics matter***
- **RQ2:** How influential are non-gameable features in differentiating human-authored vs. GPT-4 generated code?
  - ***"Decoration" independent (e.g., white spaces or indentation length)***
- **RQ3:** How well does the classifier perform when trained and evaluated on only correct solutions?
  - ***Semantic independent***
- **RQ4:** How well does the classifier perform when trained and evaluated across varying levels of problem difficulty?
  - ***Complexity and algorithm independent***

System Design for the Study

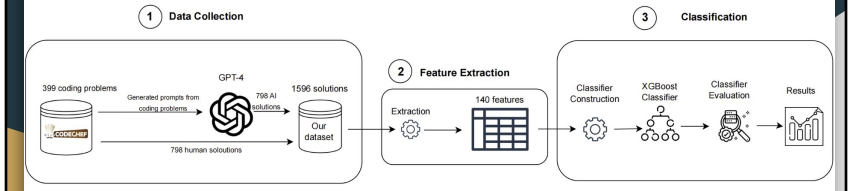


Figure 2: An overview of our approach in detecting GPT-4 generated code

System Design for the Study

Table 1: Difficulty Levels of Selected CodeChef Problems

Level	Range	Count
Beginner	0 - 999	12
1* Beginner	1000 - 1199	45
1* Advanced	1200 - 1399	71
2* Beginner	1400 - 1499	55
2* Advanced	1500 - 1599	56
3* Beginner	1600 - 1699	60
3* Advanced	1700 - 1799	53
4*	1800 - 1999	30
5*	2000 - 2199	14
6*	2200 - 2499	2
7*	2500 - 5000	1
		399

Table 2: Final Problem Set Binned into 3 Classes of Difficulty

Difficulty	Difficulty Scores - Range	Average	Count
Easy	828 - 1417	1224.95	133
Medium	1419 - 1646	1529.51	133
Hard	1647 - 3420	1827.50	133

System Design for the Study

Table 3: Code Stylometry and Code Complexity Features

Feature	Description
ASTNodeTypesTF [11]	Term frequency of 130 possible AST node types excluding leaves
ASTNodeTypeAvgDep [11]	Average depth of 130 possible AST node types excluding leaves.
avgFunctionLength [21]	The average length of lines in a function.
avgIdentifierLength [21]	The average length of identifier names.
avgLineLength [11]	The average length of characters in each line.
avgParams [11]	The average number of parameters across all functions.
branchingFactor [11]	Average branching factor of the code's AST.
cyclomaticComplexity [39]	The number of decisions within a block of code.
emptyLinesDensity [11]	The number of empty lines divided by source code lines.

System Design for the Study

- No comments
- Zero shots

You are an expert Python Programmer. Your job is to look at a programming puzzle provided by the user and output 2 different ways to solve the solution in python.
The Input is provided with the following contents:
{The problem statement}
{How the input would be formatted},
{Format to be followed in the output generated},
{Constraints on the variables specified in the problem}
Make sure to take the input from the user considering the input format Output should be printed as defined in the output format
Do not attempt to explain the solution only output the code in the following format:
[PYTHON1]
{Solution to given puzzle in Python}
[\PYTHON1]
[PYTHON2]
{Alternate solution to given puzzle in Python}
[\PYTHON2]

Figure 3: Prompt used for generating 2 AI code solutions

Evaluation

- Answers to RQ1 and RQ2

Table 4: Classifier Performance Comparison among Different Approaches for Distinguishing between AI-generated and Human-authored Code

	Our Approach		Baseline		
	All	Non-Gameable	Naive	n-grams + L n = 2	n = 3
Accuracy	0.91	0.89	-	0.86	0.88
Precision	0.91	0.89	0.5	0.86	0.87
Recall	0.91	0.89	0.5	0.88	0.88
F1-score	0.91	0.89	0.5	0.87	0.88
AUC-ROC	0.91	0.89	-	0.86	0.88

Evaluation

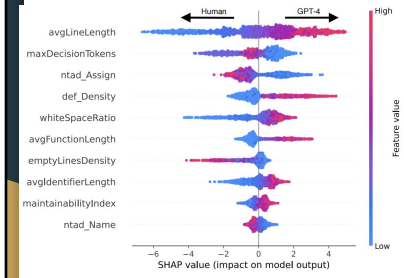


Figure 4: SHAP feature importance of our approach

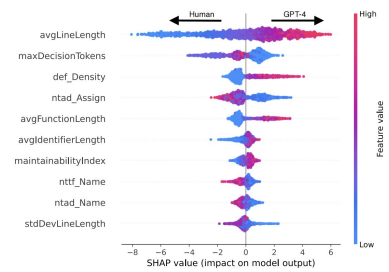


Figure 5: SHAP feature importance of non-gameable features

Evaluation

- Answers to RQ3

Table 5: Classifier Performance Comparison on Correct and Randomly Sampled Solutions

	Our Approach		Baseline (n-grams + L)			
	C	R	n = 2		n = 3	
			C	R	C	R
Accuracy	0.86	0.87	0.83	0.84	0.87	0.86
Precision	0.87	0.87	0.83	0.84	0.87	0.86
Recall	0.86	0.88	0.81	0.85	0.87	0.85
F1-score	0.86	0.87	0.82	0.84	0.87	0.86
AUC-ROC	0.86	0.87	0.83	0.84	0.87	0.86

C = Correct Solutions, R = Random Solutions, L = Lexical Features

Evaluation

Table 6: Classifier Performance Comparison Across Levels of Problem Difficulty

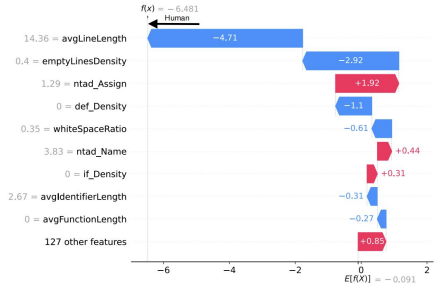
	Our Approach			Baseline (n-grams + Lexical Features)					
				n = 2			n = 3		
	Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard
Accuracy	0.89	0.89	0.87	0.87	0.79	0.80	0.89	0.86	0.80
Precision	0.87	0.88	0.89	0.85	0.80	0.79	0.89	0.87	0.80
Recall	0.91	0.90	0.86	0.89	0.77	0.82	0.88	0.85	0.81
F1-score	0.89	0.89	0.87	0.87	0.79	0.80	0.89	0.86	0.80
AUC-ROC	0.89	0.89	0.87	0.87	0.79	0.80	0.89	0.86	0.80

- **Answers to RQ4**

Evaluation

```python
n, m = map(int, input().split())
mi = 2
ma = n + m
ans = [1 for i in range(ma + 1)]

for i in range(2, int(ma**0.5) + 1):
    for j in range(i + i, ma + 1, i):
        ans[j] = 0
ans[0] = 0
ans[1] = 0
print(ans.count(1))
```



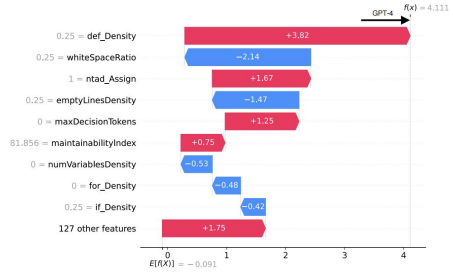
(b) SHAP waterfall plot showing how features impact the model's decision for this code

Evaluation

```python
def solve(n, m):
    return 1 if min(n, m) > 1 else 2

n, m = map(int, input().strip().split())
print(solve(n, m))
```

(a) GPT-4 generated code

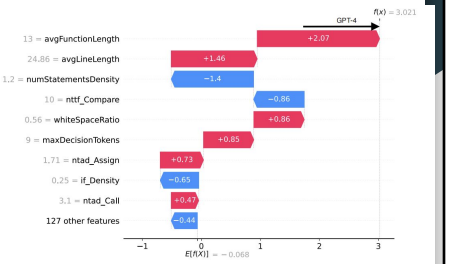


(b) SHAP waterfall plot showing how features impact the model's decision for this code

Evaluation

```python
def getCount(h,m,i) :
    h=int(h)
    m=int(m)
    h1=0
    lst=[11,22,33,44,55,66,77,88,99]
    while(h1<h) :
        for m1 in range(0,m) :
            if h1<10 :
                if(m1<10 and h1==m1) : count[i]+=1
                if (m1 in lst and m1%10==h1) :  count[i
    ]+=1
            else :
                if(m1 in lst and h1==m1) : count[i]+=1
                if(h1 in lst and h1%10==m1) : count[i]+=1
        h1+=1
t=int(input())
count=[0]*t
for i in range(0,t) :
    h,m=input().split()
    getCount(h,m,i)

for i in count : print(i)
```



(b) SHAP waterfall plot showing how features impact the model's decision for this code

Positives

- **Robust methodology:**
  - Equal numbers of human and AI solutions, spanning Easy/Medium/Hard difficulty.
  - Feature set of 140 stylometry + complexity metrics.
  - GroupKFold by problem ensures no data leakage.
  - Tested baselines (random + n-grams) for fair comparison.
  - Robustness checks:
    - Works even with only non-gameable features.
- **Interpretability:** Uses SHAP for global & local feature explanation → supports human trust in detection.
- **Open science:** Dataset, feature lists, replication package made public for reuse.

Limitations

- **Platform dependence:** Study is based only on CodeChef; coding practices may differ on other platforms.
- **Problem sampling:** Used the most popular 100 problems per difficulty (with Python solutions). While this spans from beginner to expert, it might not capture all coding styles.
- **Representativeness & Model coverage:** Only GPT-4 was used for AI-generated code; results may not generalize to outputs from other AI assistants.
- **Correctness checking**: AI-generated code was validated only against public test cases from CodeChef, not private ones.

Limitations

Table 1: Difficulty Levels of Selected CodeChef Problems

Level	Range	Count
Beginner	0 - 999	12
1* Beginner	1000 - 1199	45
1* Advanced	1200 - 1399	71
2* Beginner	1400 - 1499	55
2* Advanced	1500 - 1599	56
3* Beginner	1600 - 1699	60
3* Advanced	1700 - 1799	53
4*	1800 - 1999	30
5*	2000 - 2199	14
6*	2200 - 2499	2
7*	2500 - 5000	1
		399

Table 2: Final Problem Set Binned into 3 Classes of Difficulty

Difficulty	Difficulty Scores - Range	Average	Count
Easy	828 - 1417	1224.95	133
Medium	1419 - 1646	1529.51	133
Hard	1647 - 3420	1827.50	133

Future Works

- **Large-Scale, Real-World Validation**
  - Test on real course submissions (with consent + anonymization).
  - Compare classifier judgments against instructor-labeled ground truth.
- **Multiple Languages**
  - Current study: Python only.
  - Future: test across Java, C++, JavaScript, etc., to see if stylometric signals hold.

Rating

- **4 / 5:** rigorous paper, interesting topic, but applicable…?

Discussion

- **How to utilize the research result?**

Discussion

- **How to utilize the research result?**
- **The authors used GroupKFold by problem to avoid leakage.**
  - Does this truly guarantee independence, or could stylistic overlap between problems still bias results?
- **SHAP interpretability is highlighted, but…**
  - Do SHAP explanations meaningfully help instructors, or are they too abstract for non-ML experts?

Reference

Oseremen Joy Idialu, Mathews, N.S., Rungroj Maipradit, Atlee, J.M. and Nagappan, M. 2024. Whodunit: Classifying Code as Human Authored or GPT-4 Generated - A case study on CodeChef problems. MSR '24: 21st International Conference on Mining Software Repositories. (Jul. 2024), 394–406. DOI:https://doi.org/10.1145/3643991.3644926.