

An Empirical Study on Automatically Detecting AI-Generated Source Code: How Far Are We?

Hyunjae Suh
University of California, Irvine
Irvine, CA, USA
hyunjas@uci.edu

Mahan Tafreshipour
University of California, Irvine
Irvine, CA, USA
mtafresh@uci.edu

Jiawei Li
University of California, Irvine
Irvine, CA, USA
jiawl28@uci.edu

Adithya Bhattiprolu
University of California, Irvine
Irvine, CA, USA
abhattip@uci.edu

Iftekhar Ahmed
University of California, Irvine
Irvine, CA, USA
iftekha@uci.edu

2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)
✕arXiv Preprint

背景：コード生成

LLMはコード生成で最先端の性能

- 例：ChatGPT, Gemini Pro, StarCoder2
- 人間が書くコードと似たコードの生成が可能

LLMで生成されたコードの品質

- プロンプトの表現が容易に影響を与える^[61]
- 約35%に様々なセキュリティ問題がある^[20]
- 人間 or LLMどちらで生成されたのかの判断が必要

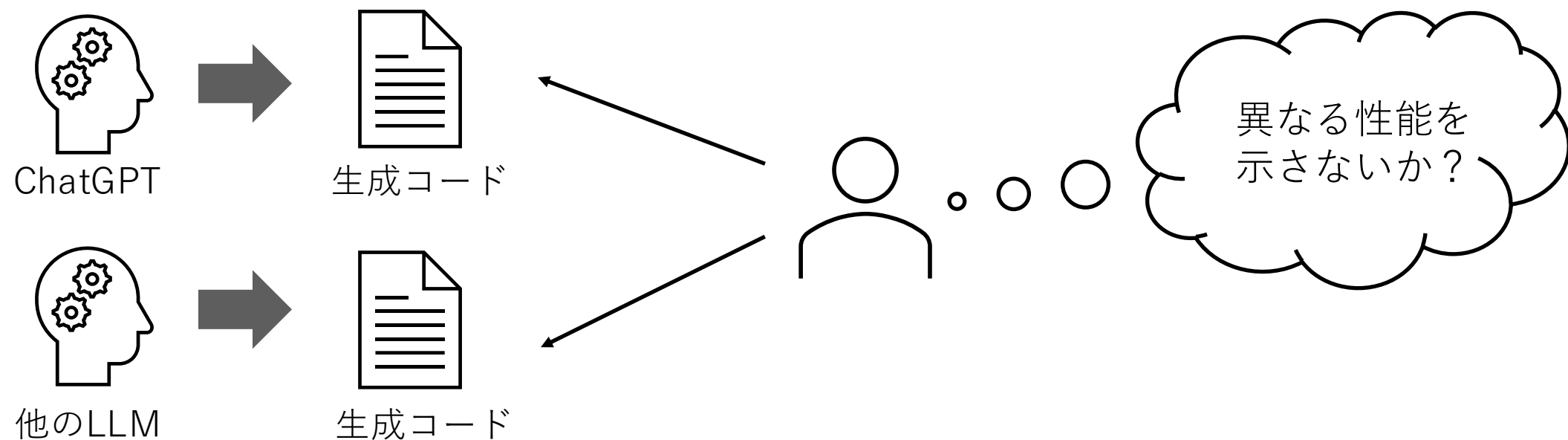
[61] A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, “On the robustness of code generation techniques: An empirical study on github copilot,” in 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023, pp. 2149–2160.

[20] O. Asare, M. Nagappan, and N. Asokan, “Is github’s copilot as bad as humans at introducing vulnerabilities in code?” Empirical Software Engineering, vol. 28, no. 6, p. 129, 2023.

既存研究

LLMで生成されたコードを検出するツールが存在

- CodeBERTをFTして，コードを人間が書いたのか，LLMが生成したのかを分類するGPTSnifferを提案^[64]
- Javaで書かれ，ChatGPTで生成したもののみを考慮



他のプログラミング言語やLLMで生成されたコードの汎用性が調査されていない

[64] P. T. Nguyen, J. Di Rocco, C. Di Sipio, R. Rubel, D. Di Ruscio, and M. Di Penta, "Is this snippet written by chatgpt? an empirical study with a codebert-based classifier," arXiv preprint arXiv:2307.09381, 2023.

目的とリサーチクエスチョン

目的

- 既存のLLMコード検出器の能力を調査すること
 - 他のプログラミング言語, LLMではどうか

リサーチクエスチョン (RQ)

1. 既存の検出器はどの程度機能するか？
2. LLM生成コード検出の性能をどの程度向上できるか？

データ収集

先行研究で使用されたコード生成ベンチマークを対象

- MBPP (Python)
- HumanEvalX (Python, C++, Java)
- CodeSearchNet (Java, Python)

AI生成コード：4つの最先端LLMで生成

- ChatGPT, Gemini Pro, GPT-4, Starcoder2-Instruct

	ChatGPT	Gemini Pro	GPT-4	Starcode2-Instruct
MBPP (Python)	1,946	1,948	1,932	1,948
HumanEval-X (Python)	323	327	323	328
HumanEval-X (Java)	281	320	311	328
HumanEval-X (C++)	328	325	328	328
CodeSearchNet (Python)	400	400	400	393
CodeSearchNet (Java)	397	400	384	395

実験設定

5つのAIコード検出器を選択（RQ1）

- GPTZero^[6], GPT-2 Output Detector^[5], DetectGPT^[63], GLTR^[79], Sapling^[7]

データセット分割

- データセットを8:1:1（訓練，検証，テスト）に分割して実施
- 人間のコードを「陽性ラベル」，LLMコードを「陰性ラベル」

[6] “Gptzero,” <https://gptzero.me/>, 2023, 2023.

[5] “Gpt2detector,” <https://github.com/MacroChip/gpt-2-output-dataset>, 2023, 2023.

[63] E. Mitchell, Y. Lee, A. Khazatsky, C. D. Manning, and C. Finn, “Detectgpt: Zero-shot machine-generated text detection using probability curvature,” arXiv preprint arXiv:2301.11305, 2023.

[79] H. Strobelt and S. Gehrmann, “Catching a unicorn with gltr: A tool to detect automatically generated text,” Catching Unicorns with GLTR, 2019.

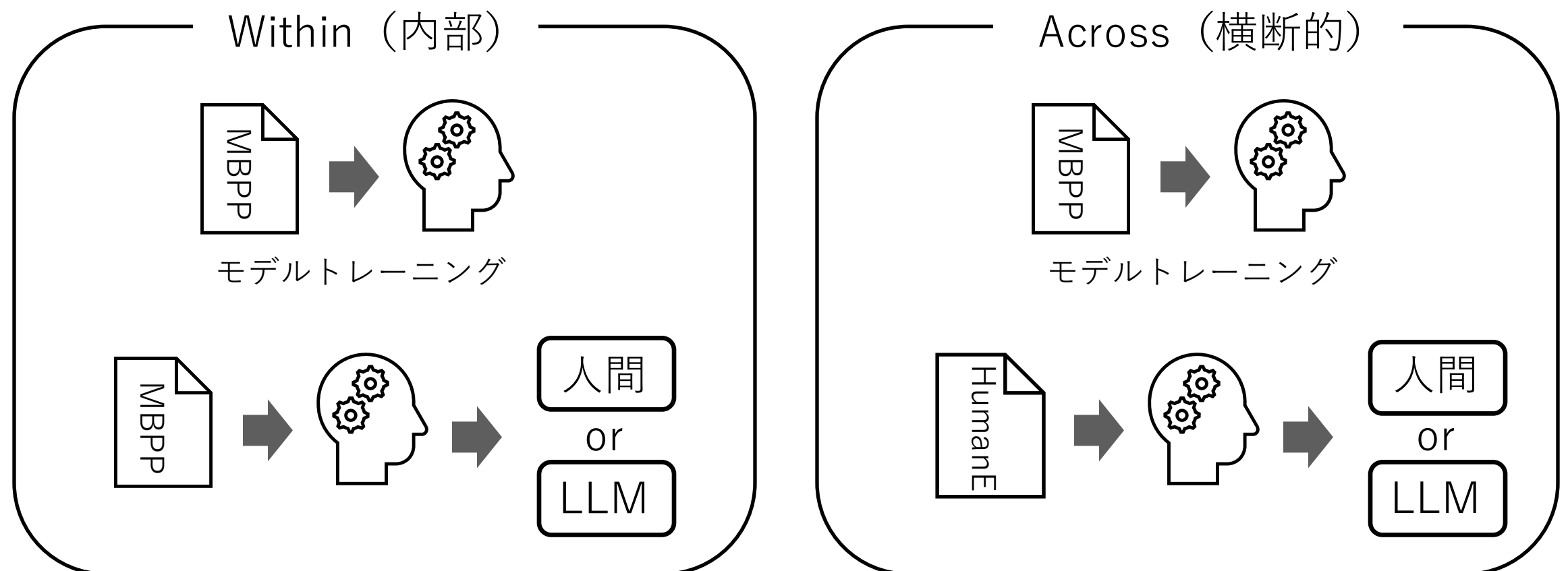
[7] “Sapling,” <https://sapling.ai/ai-content-detector>, 2023, 2023.

評価指標

評価指標

- Accuracy, TRR（真陽性率）, TNR（真陰性率）, F1-Score

2つの異なる設定で評価



RQ2：LLMベースのアプローチ

使用するLLM

- 検出器としてChatGPTを採用
- Gemini Pro, GPT-4はFTできないため不採用

3つの異なるソースコード表現

- Codeのみ, ASTのみ, Code + AST

学習設定

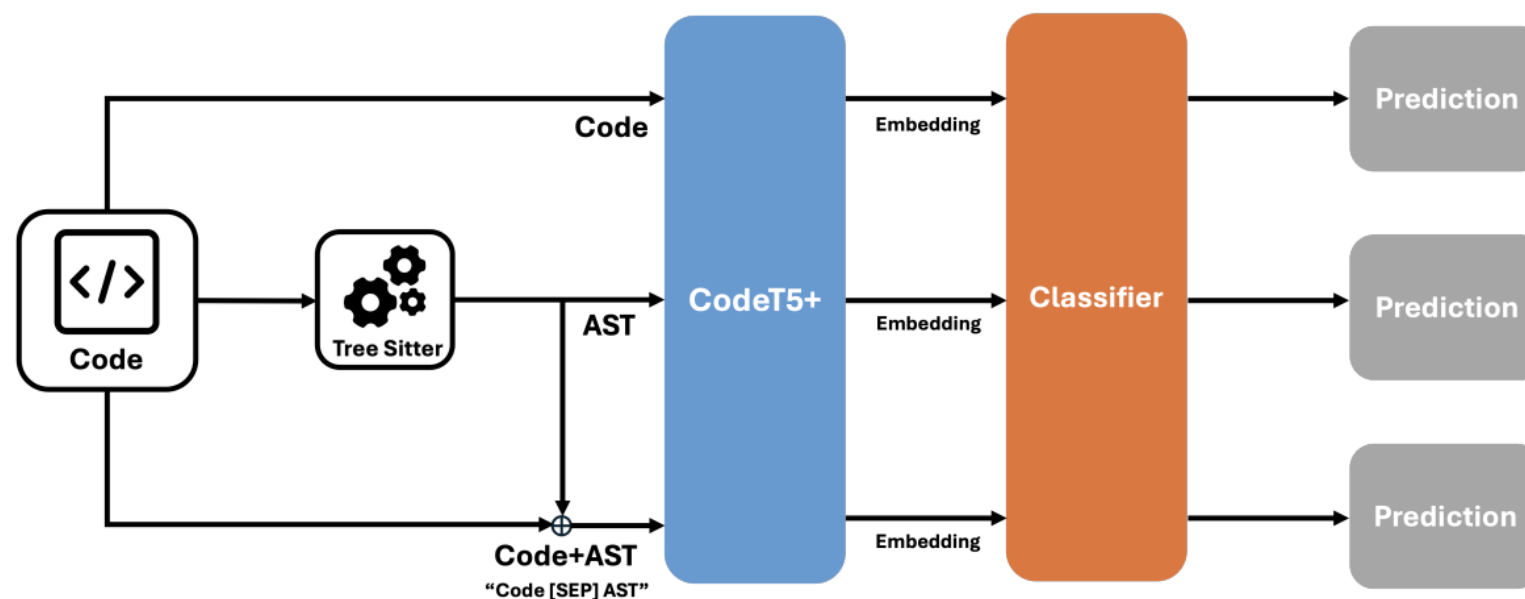
- Zero-shot学習
- インコンテキスト学習
- ファインチューニング

RQ2：機械学習ベースのアプローチ

P1：静的コードメトリクスを使用

- Tree-Sitter, VIF（分散膨張因子）からコードの特徴を抽出
 - 合計8つの特徴を保持
- 使用する分類器（古い機械学習アルゴリズムを使用）
 - ロジスティック回帰, K-近傍法, サポートベクターマシン, ランダムフォレスト, 決定木, 多層パーセプトロン, 勾配ブースティング

P2：ソースコード埋め込みを使用



RQ1：実験結果

	ChatGPT				Gemini Pro				GPT-4				StarCoder2-Instruct			
	ACC	TPR	TNR	AVG_F1	ACC	TPR	TNR	AVG_F1	ACC	TPR	TNR	AVG_F1	ACC	TPR	TNR	AVG_F1
GPT2 Output Detector	45.61	0.00	100.00	31.88	36.66	0.00	100.00	34.54	46.65	49.07	53.07	32.25	50.22	26.49	73.95	45.54
DetectGPT	46.44	2.95	100.00	33.61	38.97	7.21	99.51	39.13	43.71	43.31	52.26	37.24	52.64	7.26	98.01	39.70
Sapling	55.17	94.46	11.60	44.10	33.79	92.73	7.83	37.25	45.90	43.62	35.09	35.55	52.18	47.49	56.88	50.81
GPTZero	53.13	85.00	0.83	35.22	45.34	100.00	0.00	31.16	38.82	39.65	29.85	34.98	53.87	28.74	79.00	47.39
GLTR	44.53	48.73	44.64	41.63	41.83	45.32	68.46	47.80	51.83	45.48	46.59	46.59	47.21	47.08	47.34	44.22

- **Accuracy**

全ての場合で， 0.6未満になっている

→ LLMソースコード検出において非効果的

- **TPR・TNR**

DetectGPTが低いTPR値， 高いTNR値

→ 人間が書いたコードをLLMコードと判断してしまっている

- **F1-score**

StarCoder2-Instructが比較的高いスコアを出した

RQ2：実験結果（LLMベース）

	GPT-4			
	ACC	TPR	TNR	AVG_F1
GPT-3.5 (Zero shot)-Code	54.15	100.00	0.05	35.11
GPT-3.5 (In-context)-Code	55.58	83.80	25.76	49.80
GPT-3.5 (Fine-tuned)-Code	88.89	93.33	84.18	88.22
GPT-3.5 (Zero shot)-AST	43.92	64.61	21.51	39.85
GPT-3.5 (In-context)-AST	53.32	83.38	21.72	46.91
GPT-3.5 (Fine-tuned)-AST	65.64	83.22	48.96	62.61
GPT-3.5 (Zero shot)-Code+AST	52.05	98.19	1.77	35.51
GPT-3.5 (In-context)-Code+AST	52.57	61.81	43.50	50.25
GPT-3.5 (Fine-tuned)-Code+AST	75.37	90.42	61.87	71.72

Within

	GPT-4			
	ACC	TPR	TNR	AVG_F1
GPT-3.5 (In-context)-Code	55.12	86.78	19.75	46.24
GPT-3.5 (Fine-tuned)-Code	53.26	49.10	54.55	46.01
GPT-3.5 (In-context)-AST	51.38	42.30	59.02	48.74
GPT-3.5 (Fine-tuned)-AST	51.47	60.54	40.53	44.67
GPT-3.5 (In-context)-Code+AST	54.08	60.78	44.40	50.46
GPT-3.5 (Fine-tuned)-Code+AST	54.48	56.08	51.90	48.72

Across

- Within
 - ファインチューニングにより，Accuracy，F1-scoreが80を超えたものも存在した
 - 既存のLLMコード検出器よりも良く検出することができた
- Across
 - ファインチューニングをしても，Accuracy，F1-scoreは40程度しか示さなかった
 - 異なるプログラミング言語からのコードでの汎化可能性に欠けている

RQ2：実験結果（静的コードメトリクス）

機械学習分類器の性能を評価

- 勾配ブースティング（GB）が最も良い性能を示した

WithinとAcrossでの評価結果

	ACC	TRR	TNR	F1		ACC	TRR	TNR	F1
GB	78.72	84.81	73.62	78.28	GB	56.79	54.43	59.11	51.82
Within					Across				

- Within
 - 既存のAIコード検出器を上回った
- Across
 - F1スコアが50程度，モデルの性能が限定的になっている

RQ2：実験結果（ソースコード埋め込み）

機械学習分類器としてロジスティック回帰を選択

	ACC	TRR	TNR	F1		ACC	TRR	TNR	F1
Code	81.15	83.37	80.41	80.91	Code	51.15	39.01	63.72	42.20
AST	89.89	91.76	88.34	89.80	AST	53.47	38.67	69.42	46.83
C+A	84.85	84.67	85.45	84.46	C+A	54.85	42.37	67.89	46.64
Within					Across				

- Within
 - ASTでの埋め込みが最良の性能を示した
- Across
 - LLM生成コードの検出にあまり良い性能を示さなかった

考察

ChatGPTは様々なSEタスクで最先端の性能を示した

- LLM生成コード検出ではAccuracyが40程度と低い性能
- LLMでもFTされていない場合、識別能力に欠ける

本研究でのモデルでは、優れた性能を示した

- AccuracyとF1スコアで80を超えることができた
- 「Across」では依然として低い性能

ASTでのコード埋め込みでは、最高の性能を示した

- マルチモーダルなコード表現が必要であることが示唆される

まとめ

目的

- 既存のLLMコード検出ツールの性能を調査

結果

- 既存のツールは低い性能を表した
- LLMベース, コードメトリクスベース, ソースコード埋め込みベースを提案
- ソースコード埋め込みを用いた場合で, 最高のF1スコアを出すことができた

補足：コードメトリクス

Features	Definitions	
SumCyclomatic	Sum of cyclomatic complexity of all nested functions or methods.	ネストされた関数やメソッドの循環的複雑度
AvgCountLineCode	Average number of lines containing source code for all nested functions or methods.	ソースコードの平均行数
CountLineCodeDecl	Number of lines containing declarative source code	変数宣言や関数の行数
CountDeclFunction	Number of functions	関数の数
MaxNesting	Maximum nesting level of (if, while, for, switch etc.)	最大のネストレベル
CountLineBlank	Number of blank lines	空白行の数
Keywords	The ratio between the number of language keyword tokens to the number of total tokens	キーワードトークン数の数と全トークン数の数の比率
Operators in if, else, and while statements	The ratio between the number of operators in if, else, and while statements to the number of total tokens	if, else, whileステートメント内の演算子の数と全トークン数の数の比率