AI生成したプログラミングコードの識別に関する研究

Study on Identifying AI-Generated Program code

西村太孝・マネジメント分科会・情報セキュリティ大学院大学

Abstract - In this study, we analyze Python programs generated by multiple generative AI models from identical natural language prompts, focusing on both generating-model classification and inter-model code similarity analysis. Our framework uses CodeT5+-based code embeddings and integrates code normalization, abstract syntax tree (AST) representations, and UniXcoder-based structural information within a unified embedding space. Experiments across diverse model families, including distilled models and different generation settings, show that incorporating structural information improves classification performance. Moreover, normalization and structural features clearly reveal model families and distillation relationships as similarities in the embedding space, demonstrating the effectiveness of structural analysis for quantitatively understanding relationships among generative AI models.

1.背景と研究課題

生成AIによるコード生成が普及する一方、生成元や生成特性の把握が必要である。本研究では、コードを構造的に分析することで、生成モデルの識別と類似性分析を試みる。

RQ1:生成AIモデルを識別できるか？

RQ2:モデル間の類似性はどのように現れるか？

本研究の貢献

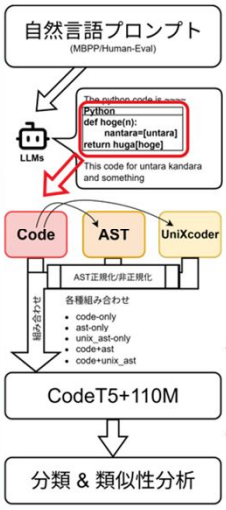
CodeT5+ によるコード埋め込みを共通基盤に分類と類似性分析を統合

AST、正規化、UniXcoder を組み合わせた体系的なアブレーション

モデル系列や蒸留関係が埋め込み空間にどう現れるかを実証

2.提案手法

コード表層情報、構文情報、AST表現、正規化処理の有無の4点に着目した、コードの表層表現だけでなく、構造（AST）にも着目した分析フロー



全体フロー概要

1.同一プロンプトからコード生成

2.生成結果からコード部分のみを抽出（Code-only 化）

3.AST の構築および表現方式の切替

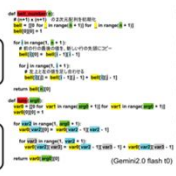
4.AST 正規化の有無を切り替えた前処理

5.CodeT5+110M によるコード表現の埋め込み生成

6.分類および類似度分析による評価

AST正規化の例

変数名の変更やコメントの削除が行われている。



使用したプロンプトの例

You are a senior Python engineer.

Follow these strict rules:

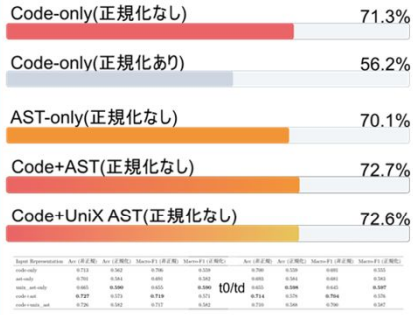
- Output ONLY one Python function that solves the task.
- No prints, no input(), no files, no network, no I/O, no main block.
- Use Python standard library only.
- Keep it deterministic and efficient; respect time and space complexity.
- Return the result from the function; do not execute it.

Task: Write a function to find the number of ways to partition a set of bell numbers.

(Gemini2.0 flash t0)

3.実験結果（RQ1:分類）

GPT-4o-mini、Gemini2.5-flash、LLama3.3の3つのLLMで生成したコードを使用し、分類の実験を行った。AST正規化による表層情報（コメントや変数名）の削除により、精度が低下。



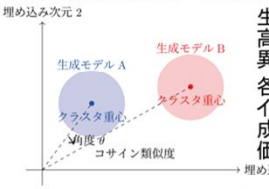
AST情報を付加することで、Code-Onlyよりも分類精度が向上。Code+ASTの組み合わせが最良の結果を出した。

T=dにおいても性能関係は概ねT=0の時と一致している。

T（Temperature）について:Temperatureは LLMの出力のランダム性を決定するパラメータの一つ。低くすると一貫性のある回答をし、高くすると創造性のある回答をするようになる。

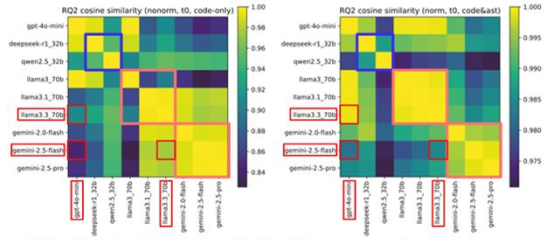
3.実験結果（RQ2:類似性）

モデル間の埋め込みベクトル（重心）のコサイン類似度を分析



生成したコードは、CodeT5+により高次元ベクトル空間でモデルごとに異なる分布（クラスタ）を形成する。

各クラスタの重心ベクトル間のコサイン類似度を計算することで、生成モデル間の構造的類似性を評価する。



系列モデル(Gemini,LLaMA)は明確にクラスタ化

蒸留モデル(Deepseek,Qwen)は近く（0.951→0.987）

ただし、差は非常に小さく（差分は最大でも約0.03 程度）、高い類似性の中に微細な差がある。

赤枠で囲ったモデルは、RQ1で使用したモデルであるが、埋め込み空間上での近接や遠隔が見られる。近接、遠隔によってRQ1の結果に影響を与えた可能性がある。

4.考察

Temprature の影響：

Temprature=0（決定的生成）の方がモデルの特徴がはっきりとし、識別精度が高くなる。

正規化の効果：

AST正規化を行うと、系列モデル内の類似度が上がり、系列モデル間の類似度が下がる。ブロック構造が明確に現れるようになった。

構造的指紋（フィンガープリント）の存在：

正規化（変数名隠蔽）を行っても識別可能であることから、LLMにはコードロジック構築レベルでのクセが存在する。

5.結論

本研究は、AI生成コードを構造的・意味的特徴を持つ対象として捉えることで、識別が可能であることを示した。

RQ1では、codeとAST表現を併用することで、分類性能の向上に寄与することを示した。

RQ2では、モデル間のコード類似性を埋め込み空間上で把握することで、同一系列に属するモデル同士は比較的高い類似性を示すことが確認された。

特にAST正規化およびUniXcoder を用いた構造表現を導入することで，モデル間類似性の構造がより明瞭に可視化された。

しかしながらコサイン類似度は全体として0.95以上と非常に高く、高い類似性の中に微小な差異が含まれている可能性を示唆するに留まる。

参考文献

[1] H. Suh et al, "An Empirical Study on Automatically Detecting AI-Generated Source code",ICSE 2025. DOI:10.1109/ICSE55347.2025.00064

[20] Y.Wang et al, "CodeT5+: Open code large language models for code understanding and generation", 2023. DOI:10.18653/v1/2023.emnlp-main.68

[36] D.Guo et al, "Unixcoder: Unified cross-modal pre-training for code representation", 2022. DOI:10.18653/v1/2022.acl-long.499