ChatGPT を用いたPython コード難読化と構造的類似度分析 ツールの評価

氏名　近藤 惇斗
学籍番号　253550

研究背景

- 既存の難読化ツールの多くは「難読化後のコードを生成すること」が主目的でどの難読化手法が使われているのか、難読化前後でコード構造がどれほど変化したのか、をユーザが把握しにくい。
- 研究・評価の観点では、難読化の 強度、可読性・保守性への 影響を客観的指標で比較することが困難なので難読化前後の構造変化を**多角的・定量的に評価できる仕組みが必要である**

提案

- 生成AIは綺麗なコードを書いたり、その内容を詳しく説明してくれる。

↓

生成AIを使って難読化し、その手法を説明してくれるシステムを作成する

提案手法

構文構造ベース

- AST構造類似度
  ソースコードを構文木として表現し、その木構造の一致度を評価する指標である。
- ASTノード頻度に基づくコサイン類似度
AST内に出現するノード種類（If，For，Callなど）の出現回数をベクトル化し、難読化前後のコード間の類似度を測定する手法である。

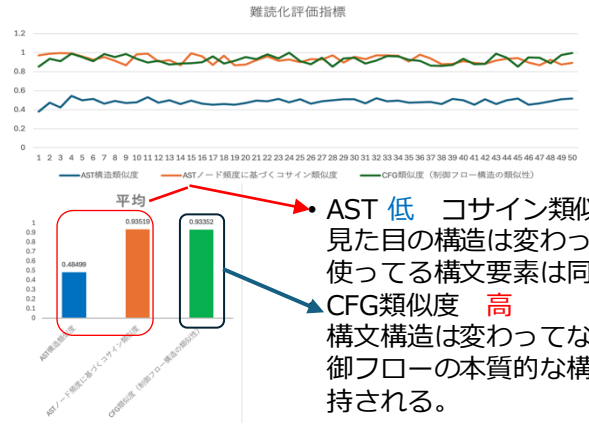
制御構造ベース

- CFG類似度（制御フロー構造の類似性）
  プログラムの実行経路をグラフとして表現し，制御フロー構造の一致度を評価する指標である。
- サイクロマチック複雑度
  プログラム内の分岐数やループ構造の数に基づいて算出される指標であり，制御構造の複雑さを定量的に表すものである。

- 4つの関数が表す結果
- Geminiによる、難読化手法の説明
- 特徴量比較グラフ

プログラムコードをChatGPTで難読化し、どれくらい難読化できたかを数値で表し、Geminiを用いて難読化手法の説明したものをHTMLに書き出す

研究結果



- AST 低　コサイン類似度 高
  見た目の構造は変わったけど、使ってる構文要素は同じ。
- CFG類似度　高
  構文構造は変わってなく、制御フローの本質的な構造は維持される。

難読化前：9.02
難読化後：8.91

難読化後のサイクロマチック複雑度の平均値は低下した。これは処理が複数の関数へ分割されたことにより、各ブロック単位での分岐数が分散したためと考えられる。つまり、これは制御フローそのものを大きく変更するのではなく、構造的な複雑さを増加させることがわかる。

まとめ

ChatGPTによる難読化では、識別子名の変更や処理の分割、冗長な関数呼び出しの追加などが行われる。
これは、ChatGPTが**「意味を保ったまま構文を変形する」**という方針でコードを書いているためだと考えらる。

→ 生成AIによる難読化は、構文の見た目を複雑にする表層的な難読化にとどまる傾向がある。