

# 防御技術強化を狙いとしたセマンティクス保持型 マルウェア変種生成・拡張手法

渡邊 龍星<sup>1,2,5,a)</sup> 富永 大智<sup>1</sup> 犬塚 祥<sup>2</sup> 森 達哉<sup>2,4,,b)</sup>

**概要：**マルウェア対策において、攻撃者は既存の悪性コードを改変・難読化して検知を回避し、防御側は常に後手に回る構図が続いている。検出器の頑健化には悪性挙動を保持した変種を学習データに取り込むことが重要であるが、マルウェアのソースコードは非公開が多く、バイナリからの再構成も困難であるため、実行ファイルから直接データ拡張を行う必要がある。従来手法は機能破壊によるラベル不整合、現実的な変種との乖離、ルール依存等の課題があり、セマンティクス保持しつつ柔軟に拡張を行うことは困難であった。本研究は、これらの問題を解決するセマンティクス保持型データ拡張手法を提案する。提案手法は実行ファイルのみを入力とし、バイナリを視覚化して得た注意マップにより悪性挙動に関与する領域を特定し、その領域のアセンブリコードをLLMで意味等価に再構成することで、CFGレベルの構造的変換を実現する。BODMASデータセットから選定した100サンプルの評価実験では、動的解析を行い、従来手法では困難であった実行コード内部のCFG構造を改変する変換により、24サンプルで元の悪性挙動の維持を確認した。生成された変種はVirusTotal上で検知数が減少しながら悪性挙動を維持し、検出器の頑健性向上に貢献できる現実的なデータ拡張手法としての可能性を示した。

**キーワード：**マルウェア, CFG, Vision Transformer, 大規模言語モデル, データ拡張

## Defense Enhancement through Semantics-Preserving Malware Variant Generation and Augmentation Techniques

RYUSEI WATANABE<sup>1,2,5,a)</sup> DAICHI TOMINAGA<sup>1</sup> SHO INUZUKA<sup>2</sup> TATSUYA MORI<sup>2,4,,b)</sup>

**Abstract:** In malware defense, attackers modify and obfuscate existing malicious code to evade detection, keeping defenders reactive. While incorporating functionality-preserving variants is crucial for detector robustness, malware source code is often unavailable and binary reconstruction difficult, necessitating direct data augmentation from executables. Existing methods face label inconsistency from functionality corruption, deviation from realistic variants, and rule dependency, making flexible semantic-preserving data augmentation difficult. This research proposes a semantics-preserving data augmentation method addressing these problems. The method takes only executables as input, identifies malicious behavior regions through attention maps from binary visualization, and achieves CFG-level transformations via LLM-based semantic reconstruction of assembly code. Evaluation experiments conducted on 100 samples selected from the BODMAS dataset confirmed preservation of original malicious behavior in 24 samples through dynamic analysis, using CFG structure modifications difficult to achieve with existing methods. Generated variants maintained malicious behavior while reducing VirusTotal detections, demonstrating potential for improving detector robustness as a practical data augmentation method.

**Keywords:** Malware, CFG, Vision Transformer, LLM, Data Augmentation

<sup>1</sup> 株式会社エヌ・エフ・ラボラトリーズ/N.F.Laboratories Inc.

<sup>2</sup> 早稲田大学/Waseda University

<sup>3</sup> 理研 AIP/RIKEN AIP

<sup>4</sup> 情報通信研究機構/NICT

<sup>5</sup> 株式会社 gin/gin. Co., Ltd

a) ryusei.watanabe@nsl.cs.waseda.ac.jp

b) mori@nsl.cs.waseda.ac.jp

## 1. はじめに

マルウェア対策において、攻撃者は既存の悪性コードを小規模に改変・難読化して検知を回避し、防御側は常に後手に回る構図が続いている。この課題に対して、機械学習ベースの椐出器の頑健性向上が重要であり、その有効な手法としてデータ拡張が注目されている。データ拡張により、訓練時に多様な変種を学習データに取り込むことで、汎化性能が向上することが複数の研究で示されている [14, 7]。特に、悪性挙動のセマンティクスを保持した変種を生成することで、同一ラベルで安全に学習データを拡充でき、椐出器の頑健性を効果的に向上させることができる [14]。しかしながら、マルウェアのソースコードは多くが非公開であり、バイナリからのソース再構成も困難であることから [13]、実行ファイルから直接セマンティクスを保持したデータ拡張を行う技術が求められている。

マルウェアのデータ拡張に関してはこれまでにいくつかの研究が提案されており、それらは(i) ランダムバイト変異、(ii) シーケンス生成、(iii) ソースコード変換、(iv) バイナリ内部変換の4つのアプローチに大別される [7, 3, 5, 8, 9, 14]。ランダムバイト変異手法は実装が容易である一方、プログラムのセマンティクスを破壊する可能性が高く、機能喪失によりラベル整合性が失われる問題がある [7]。RNNベースの MalRNN [3] や LLMベースの MalGPT [5] などのシーケンス生成手法は、バイナリ末尾への付加型アプローチに留まり、「同一サンプル+ノイズ」の域を出ず、実行コードのセマンティクスを変更できない。ソースコードレベルでの変換手法 [8, 9] は高い機能保持率を達成するが、ソースコードアクセスが前提となり適用範囲が限定的である。バイナリ内部での意味保持変換手法 [14] は、実行コードを直接改変しつつセマンティクスを保持する点で有望であり、本研究の提案手法もこのカテゴリに分類される。しかしながら、既存のバイナリ内部変換手法は事前定義されたルールベースの変換に依存し、生成される変種の多様性が用意された変換ルール集合に制限される。

本研究は、上述の問題を解決するセマンティクス保持型データ拡張フレームワークを提案する。提案手法は実行ファイルのみを入力とし、ソースコードへのアクセスを一切必要としない完全にバイナリベースのアプローチである。核心となるアイディアは、マルウェアバイナリを画像化し、Vision Transformer (ViT) の注意機構によってセマンティックに重要な領域を自動特定し、その領域に絞って大規模言語モデル (LLM) で意味等価変換を行うことである。ViT をマルウェアファミリー分類タスクで事前訓練することで、各ファミリーの悪性挙動を実現する中核的機能への注意を学習させる。特定された重要領域のアセンブリコードのみを LLM への入力とすることで、トークン効率を大幅に改善しながら、制御フローグラフ (CFG) レベル

での構造的変換を自動生成する。LLM の活用により、事前定義された変換ルールに依存せず、柔軟で多様なセマンティクス保持変換を実現する。

提案手法の有効性を検証するため、BODMAS データセット [16] から選定した 100 サンプルを対象に、生成された変種の機能維持性と CFG 構造の変換品質を評価する実験を実施した。機能維持とは、変種が元のマルウェアと同一の悪性挙動を実行可能な状態を指し、サンドボックス環境での動的解析により検証した。その結果、24%のサンプルで完全な機能維持を達成した。この数値は、従来の付加型手法とは異なり実行コード内部の CFG 構造を直接改変するという高度な変換において達成されたものであり、実用的な変種生成の可能性を示している。生成された変種を静的解析した結果、制御フロー再構成、不要なエラー処理分岐の削除、命令順序の入替えなど、従来手法では困難であった柔軟で高度な変換が確認された。さらに、VirusTotal での評価において検知数が減少しながら悪性挙動を維持することが確認され、椐出器の頑健性向上に貢献できる実用的なデータ拡張手法としての可能性を示した。

本研究の主な貢献は以下の通りである。

- **バイナリ直接処理による変種生成フレームワーク**：実行ファイルのみから機能保持型変種を自動生成する一気通貫パイプラインを実現。
- **ViT × LLM によるセマンティクス保持変換**：注意機構で特定した重要領域を LLM で意味等価に再構成し、CFG レベルの構造的变化を実現。

本項の以降の章は次のように構成されている。第 2 章では、マルウェアサンプル拡張生成の関連研究について示す。第 3 章で提案手法の詳細、第 4 章で評価実験とその結果について示す。第 5 章では、提案手法の評価結果をもとに考察を行い、最後に第 6 章では結論を示す。

## 2. 関連研究

### 2.1 マルウェアサンプルのデータ拡張

マルウェア椐出器の性能向上において、十分な訓練データの確保は重要な課題である。従来のマルウェアデータ拡張手法は、主に以下の4つのアプローチに分類される。

第一に、ランダムなバイト変異やノイズ付加による単純な拡張手法が存在する [7]。これらの手法は実装が容易である一方、ノイズ付加型でない限りプログラムの意味を破壊する可能性が高く、生成されたサンプルが現実的な変種とは大きく乖離するという問題を抱えている。

第二のアプローチとして、RNN や LLM を用いたシーケンス生成手法が提案されている。Ebrahimi らによる MalRNN [3]、MalGPT [5] は、それぞれ、RNN、GPT-2 を用いて、マルウェアのバイナリ末尾に良性のサンプルから学習した意味のあるバイト列を付加する。これらの付加型手法は、本質的には OS が実行しないノイズのバイナリ末尾

への追加であり、現実的な変種生成手法とは言い難い。

第三のアプローチは、ソースコードレベルでの変換を行う手法である。Lu らは、マルウェアのソースコードを難読化し、再コンパイルすることで機能を完全に保持した変種を生成している [8]。また、Madani は ChatGPT-4 を用いて Python マルウェアの変異（命令置換、順序入替え等）を実現している [9]。これらの手法は高い機能保持率と現実的な変種生成を達成しているが、ソースコードへのアクセスが前提となるため、運用上の制約がある。現実には、多くの場合、入手できるのはバイナリであり、公開アーカイブには一定数のマルウェアソースも存在する一方で、特に作者が隠す意思をもつコードは見つからない。加えて、バイナリからオリジナルの可読ソースへ戻す試みは本質的に難しく、難読化技術がその難度をさらに上げる [13]。

第四のアプローチとして、Marvolo に代表される“バイナリ内部での静的な意味保持変換”がある [14]。実行ファイルを中間表現にリフトし、基本ブロック／関数単位で事前定義の変換ルール（命令置換・レジスタ再割当・関数インライン化等）を適用することで機能を保持しつつ実行部分を改変したサンプルを生成する。しかしながら、拡張の多様性“用意された変換集合とその組合せ”に制約される。

本研究では、これらの既存手法の限界を克服し、ソースコード非依存でありながら柔軟で現実的な変種生成を自動化する手法を提案する。

## 2.2 マルウェアの画像化とアテンション技術

マルウェアの視覚的表現を用いた解析手法は、2011 年の Nataraj らによる研究 [10] 以降、広く研究されてきた。彼らはマルウェアバイナリを画像として可視化し、テクスチャ特徴を用いた分類を提案した。

その後、深層学習の発展に伴い、CNN（Convolutional Neural Network）を用いたより高度な画像ベースのマルウェア解析手法が提案されている。Cui らは、マルウェア画像に対して CNN を適用し、従来の手動特徴抽出を必要としない end-to-end の分類を実現した [2]。また、Yakura らは、マルウェア画像に対する注意機構を導入し、分類に寄与する重要なバイト領域を可視化する手法を提案している [15]。これらの研究は、マルウェアバイナリの局所的パターンが悪性挙動と強い相関を持つことを示唆している。近年では、自然言語処理分野で革新的な成果を挙げている Transformer アーキテクチャが、マルウェア解析にも応用されている。Belal らの研究では、グローバル-ローカル注意機構を導入し、マルウェア画像の階層的特徴を効果的に抽出している [1]。これらの注意機構ベースの手法は、単に分類精度を向上させるだけでなく、モデルの判断根拠を解釈可能にするという重要な利点を持つ。この解釈可能性により、マルウェアの悪性挙動に寄与する重要なコード領域を視覚的に特定することが可能となり、本研究における

重要領域の選択にも示唆を与えている。

本研究では、これらの視覚的解析手法の知見を活かし、ViT の注意機構を変種生成の領域選択に応用する。従来研究が主に分類や検出を目的としていたのに対し、本手法では注意マップで特定された悪性挙動関連領域を、LLM による意味等価変換の対象として活用する。これにより、マルウェアの本質的な機能部分に焦点を当てた効率的かつ効果的な変種生成を実現する。

## 3. 提案手法

本研究では、実行ファイルのみを入力として機能保持型マルウェア変種を自動生成するフレームワークを提案する。提案手法は、ソースコードへのアクセスを必要とせず、バイナリファイルから直接、悪性挙動を維持した変種を生成する。提案手法のパイプラインを図 1 に示す。本手法は以下の 3 つの主要ステップで構成される：

- (1) **重要領域の特定**：マルウェアバイナリを画像化し、事前訓練済み ViT モデルの注意機構により悪性挙動に寄与する領域を自動抽出する。
- (2) **CFG 構築と変換**：特定された領域から CFG を構築し、基本ブロックとアセンブリコードを抽出する。
- (3) **意味等価な変種生成**：LLM を用いて抽出されたアセンブリコードを意味等価に再構成し、ディスクアセンブルを通じてバイナリヘッダを適用する。

以下の節では、各ステップの詳細を解説する。

### 3.1 ViT モデルの訓練とアテンション機構の活用

本節では、画像化手法と ViT の訓練プロセス、および提案手法における注意機構の役割について述べる。

#### 3.1.1 バイナリの画像化

マルウェアの実行ファイルを視覚的表現に変換するため、1 バイトを 1 ピクセルとして画像化する。具体的には、バイナリファイルの各バイト値 (0-255) をグレースケール画像のピクセル値として扱い、固定幅で折り返すことで 2 次元画像を生成する。画像サイズに満たない場合は 0 値でパディングを行い、超過する場合はトリミングを適用することで、統一されたサイズの画像表現を得る。この手法により、バイナリファイルの構造的特徴（セクション構造、データパターン、コード領域の分布など）を視覚的パターンとして表現できる。

#### 3.1.2 ファミリー分類タスクによる ViT の訓練

画像化したバイナリから抽出領域を効果的に決定するためにマルウェアファミリー分類タスクを用いて ViT モデルを訓練する。本研究では、ViT モデルに分類ヘッドを付加し、画像化されたマルウェアのファミリーを予測するよう学習させる。この訓練プロセスにおいて、モデルは異なるファミリー間の判別に寄与する視覚的特徴を学習する。重要な点は、各ファミリーは固有の悪性挙動パターンを持

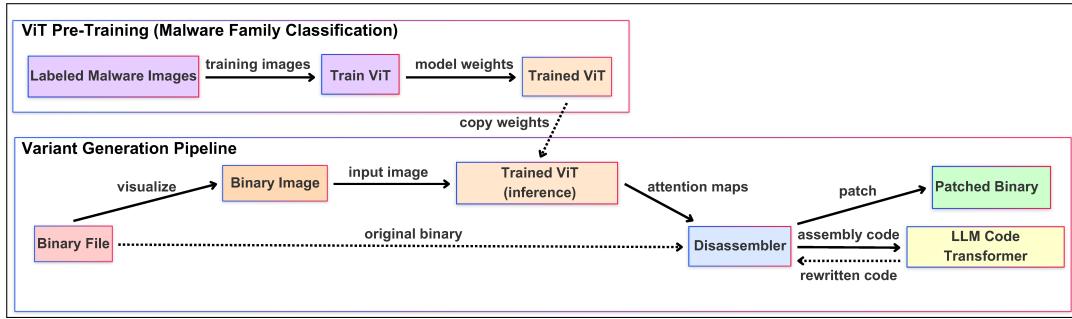


図 1 提案手法のパイプライン

つため、ViT の注意機構はこれらの挙動を実装する領域に焦点を当てるよう学習される。本手法では、この事前訓練済みモデルの注意機構が特定した重要領域を LLM Code Transformer による変種生成のターゲットとして利用する。この戦略は以下の観点から有効性が期待される：

- **機能保持の観点**：注意スコアの高い領域は、そのファミリーの特徴的箇所となるため、この領域を意味等価に変換することで、悪性挙動部分を変換できる。
- **検知回避の観点**：既存の検出器も同様にこれらの領域に基づいてファミリー分類を行う可能性が高いため、該当領域の構造的変更は検知回避に効果的である。
- **効率性の観点**：重要な領域のみを変換対象とすることで、LLM への入力サイズを絞り、かつ不要な変更による機能破壊のリスクを最小化できる。

このように、ファミリー分類タスクで学習された ViT の注意機構は、単なる領域選択以上の意味を持ち、提案手法における変種生成の質と効率を向上させる重要な要素となる。

### 3.1.3 注意マップによる重要領域の特定

変種生成の対象となるマルウェアの実行可能ファイルに対して、事前訓練済み ViT モデルを適用し、注意マップを利用して悪性挙動に寄与する重要領域を特定する。本プロセスでは、マルウェアのバイナリファイルそのものを入力として使用する。まず、拡張生成のターゲットとなるマルウェアバイナリを前述のバイナリ画像化の手法と同じ手順でグレースケール画像に変換し、前処理（訓練時の設定に合わせたリサイズおよび正規化）を適用する。次に、この画像を事前訓練済み ViT モデルに入力し、最終層の [CLS] トークンに対する各パッチの注意重みを全ヘッドで平均化する。得られた注意スコアの上位  $k$  個のパッチを選択し、これらのパッチに対応するバイナリオフセットを画像座標からの逆変換により計算する。この手法により、変種生成対象のバイナリから、その悪性挙動に重要な役割を果たすコード領域を自動的に特定できる。抽出された重要領域のみを LLM への入力としてすることで、トークン効率の大幅な向上と、より焦点を絞った効果的な変種生成を実現する。

### 3.2 重要コードブロックの抽出と CFG 構築

ViT の注意マップで特定された重要領域から、実際のアセンブリコードと制御フロー構造を抽出するプロセスを以下に示す。本プロセスは 3 つのステップで構成される：

#### (1) バイナリオフセットからアドレスへの変換

注意マップから得られたバイナリオフセットを、実行時のメモリアドレスに変換する。画像の各ピクセル位置は元のバイナリファイル内の 1 バイトに対応しているため、注意スコアの高いパッチの座標からファイルオフセットを逆算できる。この変換には実行可能ファイルのヘッダ情報とセクション構造を解析し、ファイルオフセットから仮想アドレスへのマッピングを行う。

#### (2) ディスアセンブルによる逆アセンブルと CFG 構築

変換されたアドレス範囲に対して、ディスアセンブルを用いて逆アセンブルを実行する。ここでは、以下の情報を抽出する：

- 基本ブロックの境界とアセンブリ命令列：各基本ブロックの開始・終了アドレスと、含まれる全命令の構造化された表現
- CFG：条件分岐、無条件ジャンプ、関数呼び出しなどの制御フロー遷移
- 関数の開始・終了アドレス：プロローグ・エピローグを含む関数境界の特定
- クロスリファレンス情報：データ参照、コード参照を含む相互参照関係

#### (3) 重要基本ブロックの選定

前述の注意マップから特定された重要領域に対応する基本ブロックに加えて、これらから直接到達可能な基本ブロック（例：深さ 10 の CFG 近傍）も抽出対象に含める。変換対象を重要領域の近傍まで拡張することで、局所的な制御フロー構造全体を考慮した変換を実現し、生成される変種の多様性を拡大する。

このプロセスにより、ViT が注目した視覚的パターンから、実際の実行可能コードとその制御構造を正確に特定し、後続の LLM Code Transformer への入力として準備される。

### 3.3 LLM によるコード・グラフ構造改変とパッチ適用

ディスアセンブルから抽出された基本ブロック情報を基に、LLM Code Transformer が変換を行い、生成されたコードを元のバイナリに適用するプロセスを以下に示す。

#### (1) LLM への入力形式

選定された各基本ブロックのアセンブリコードを個別に LLM Code Transformer に入力する。入力データは以下の形式で構成される：

- 変換対象の基本ブロックのアセンブリコード
- 変換要求と期待される変換の量的度

#### (2) 意味等価な変換の生成

LLM Code Transformer は、入力されたアセンブリコードに対して複数の変換手法を適用し、機能的に等価な新しいコードを生成する。適用される主要な変換パターンには例えば以下が挙げられる：

- 命令の順序入れ替え：データ依存関係を保持しながら、独立した命令の実行順序を変更
- 等価命令への置換：同じ結果を生成する異なる命令シーケンスへの変換
- 安全な命令の挿入：実行結果に影響を与えない命令（例：nop, push/pop ペア）の追加
- 制御フローの難読化：直接ジャンプを間接ジャンプテーブルに変換

生成された変換は、信頼度スコアで評価され、基準を満たさない場合は再試行される。

#### (3) バイナリパッチの適用

LLM Code Transformer が生成した変換済みアセンブリコードを、ディスアセンブルのパッチ機能を用いて元のバイナリに適用する。パッチ適用プロセスでは、該当アドレスのオペコードを新しい命令列で直接置換する。必要に応じて、以下の編集を適用する：

- 新規コード領域の確保：実行可能セクション内の未使用領域、または新規セクションの追加
- トランポリン構造の構築：元の位置からジャンプ命令で新規領域へ遷移し、処理後に戻る構造
- アドレス参照の更新：移動したコードへの参照を更新このプロセスにより、ViT とディスアセンブルで特定・抽出された重要コード領域が、LLM Code Transformer によって意味等価に変換され、新たな変種として出力される。

### 3.4 実装

提案手法の実装環境を表 1 に示す。本実装では、Python 3.12 を基盤として各コンポーネントを統合した。ViT モデルは google/vit-large-patch16-384 [6] を採用した。ファミリー分類タスクのための分類ヘッドは、ViT の最終層出力に全結合層を追加することで実装した。訓練時のパラメータは表 2 に示す通りである。逆アセンブルと CFG 抽出には IDA Pro 8.3 を使用し、IDAPython スクリプトを通じて

表 1 実装環境

項目	詳細
ViT 訓練・実行環境	Google Colab Pro+ (A100 GPU)
ViT モデル	google/vit-large-patch16-384
ディスアセンブル	IDA Pro 8.3
LLM	OpenAI API (o4-mini モデル)

表 2 ViT モデルの訓練パラメータ

パラメータ	設定値	パラメータ	設定値
解像度	384	パッチサイズ	16
エポック数	20	損失関数	CE
学習率	1e-4		

自動化を実現した [4]。LLM Code Transformer としては OpenAI API [12] の o4-mini モデルを使用し、生成されたアセンブリコードのバイナリへの再アセンブルには Keystone Engine [11] を採用した。パッチ適用は、IDAPython を用いて生成されたオペコードを直接バイナリに書き込むことで実現した。

提案手法の全体的なパイプラインは、ViT 推論部を Google Colab Pro+ 上で実行し、逆アセンブルとパッチング処理はローカルの Windows 環境で動作する IDA Pro と連携させることで構築した。各コンポーネント間のデータ交換には構造化された形式を採用し、拡張性を確保した。

## 4. 評価

### 4.1 データセット

本研究では、マルウェアバイナリとファミリー分類情報を含む BODMAS データセット [16] を使用した。BODMAS データセットは多様なマルウェアファミリーの実行可能ファイルを収録した大規模データセットであり、本研究ではこのうち悪性バイナリのみを実験対象とした。

実験の信頼性を確保するため、10 サンプル以上を含むマルウェアファミリーのみを対象とし、各ファミリーからランダムにサンプルを抽出した。この基準により、196 個のマルウェアファミリーから合計 56,342 サンプルを選定した。選定されたサンプルは、ViT モデルの訓練用とテスト用に 4:1 の比率でランダムに分割し、訓練データ 44,991 サンプル、テストデータ 11,351 サンプルを得た。

マルウェア変種生成の評価には、テストデータセットから正常に動作し悪性挙動が確認できたサンプルを選定する必要があった。動的解析環境において各サンプルの実行可能性と悪性挙動の発現を確認し、最終的に 100 サンプルを変種生成の対象として選定した。この選定プロセスにより、機能保持率を正確に評価できる実験環境を構築した。

表 3 に、実験で使用したデータセットの構成を示す。多様なファミリーからの選定により、提案手法の汎用性を評価できるデータセット構成とした。

表 3 実験データセットの構成

項目	数量
対象ファミリー数	196 ファミリー
総サンプル数	56,342 サンプル
訓練データ	44,991 サンプル
テストデータ	11,351 サンプル
変種生成対象	100 サンプル

## 4.2 処理時間と機能維持率

### 4.2.1 ViT の訓練時間と分類精度・LLM 変換処理時間

提案手法の第一段階として、マルウェアファミリー分類タスクによる ViT モデルの訓練を実施した。訓練は 3.4 節で示した構成において、Google Colab Pro+ の A100 GPU を使用して行った。4.1 節で述べた訓練データセット 44,991 サンプルを用いた 20 エポックの訓練に要した時間は 19 時間 15 分 5 秒であった。この訓練時間は事前学習でのみ必要となる処理であり、推論時には影響しない。実際の変種生成パイプラインにおいては、各サンプルの処理時間はモデルの推論時間のみに依存する。したがって、実運用においてはこの訓練時間がボトルネックとなることはない。

訓練済みモデルの性能評価として、テストデータセット 11,351 サンプルを用いてマルウェアファミリー分類精度を測定した。その結果、196 クラス分類タスクにおいて 86.9% の正解率を達成した。この精度は多クラス分類問題として十分な性能を示しており、モデルが各マルウェアファミリーの特徴的なパターンを適切に学習できていることを示唆している。本研究における ViT の利用目的はマルウェア分類そのものではなく、訓練された ViT の注意機構を活用した重要領域の特定にある。したがって、分類精度の向上は追求せず、得られたモデルを変種生成のための領域特定に適用することとした。

提案手法のパイプライン全体の処理時間を評価するため、表 3 で示す変種生成対象 100 サンプルに対して拡張サンプルを生成し、処理時間を測定した。その結果、1 サンプルあたりの平均処理時間は 998 秒であった。この処理時間の内訳を分析したところ、LLM によるアセンブリコードの意味等価変換にほぼ全ての時間が費やされており、その他の処理（バイナリの画像化、訓練済み ViT による注意マップ取得、生成されたパッチのバイナリへの適用）に要した時間は、合計で 1~2 秒未満であった。このことは、提案手法における処理時間のボトルネックが LLM の API 呼び出しと応答生成にあることを示している。しかしながら、全体としての処理時間は並列化により大幅に短縮可能であり、実用的な時間での変種生成が実現できる。また、より高速な LLM モデルの採用や、ローカル環境での推論実行により、さらなる高速化の余地がある。

### 4.2.2 機能維持率

提案手法の実用性を評価するため、表 3 で示す変種生成対象 100 サンプルに対して拡張サンプルを生成し、機能維

持率を測定した。各オリジナルサンプルから 1 つずつ変種を生成し、合計 100 個の変種サンプルを得た。生成された変種の機能維持確認は、手動による静的解析とサンドボックス環境での動的解析の両面から実施した。静的解析では、生成されたアセンブリコードの意味的等価性と CFG 構造の妥当性を検証した。動的解析では、オリジナルサンプルと同様の悪性挙動（ファイル操作、ネットワーク通信、レジストリ変更等）が維持されているかを確認した。

評価の結果、100 サンプル中 24 サンプルにおいて完全な機能維持が確認された。この数値は一見低く見えるが、提案手法が実現した CFG レベルでの構造的変換の観点から見ると重要な成果である。従来のノイズ付加型手法とは異なり、実行コードの内部構造を直接改変している。このような高度なレベルでの変換において 24% の成功率は、実用的な変種生成手法としての可能性を示している。機能維持に成功した変種の変換の質については 4.3 節で詳述する。

## 4.3 生成された変種

提案手法により生成された変種の具体的な変換内容を分析し、その質的特性を評価した。図 2 および図 3 に、実際に生成された変種と元のサンプルの局所的な差分を CFG で示す（各図、左が拡張サンプル、右が元のサンプル）。

図 2 は、元のアセンブリコードが削除され、代わりに “jmp” 命令によるトランポリン構造が生成された例である。右の青色部分の命令は左の赤色 “jmp” 先に移動されている。この変換では、元の位置から別のメモリ領域へジャンプし、そこで意味的に等価な命令を実行することで、CFG の構造を変更しながらもセマンティクスを保持している。特筆すべきは、このような実際の攻撃者が用いる高度なトランポリン構造が、LLM へのプロンプトのみで生成され、事前の特定パターンの定義なしで実現された点である。

図 3 の例では、CFG レベルでの大幅な構造簡略化が実現されている。右で灰色に示された複数の基本ブロックが左では削除されていることが視覚的に確認できる。削除されたブロックは正常系の実行パスにおいて不要なエラーハンドリング分岐の内容であった。この変換は、プログラムの本質的な機能を保持しながら、CFG 全体の構造を根本的に変更することに成功している。従来の手法では、このような CFG 構造全体を理解した上での大規模な構造的変換は不可能であり、提案手法の優位性を示している。

これらの変換例が示すように、提案手法は単なるノイズ付加や機械的な命令置換を超えた、高度な変換を実現している。生成された変種は、BinDiff などの既存のバイナリ差分解析ツールでも元のサンプルとの対応付けが困難なレベルの構造的变化を含むケースもあることが確認された。

## 4.4 検知回避効果の評価

生成された変種の検知回避効果を評価するため、Virus-

```

0041c474: jmp 0x41c480
0041c477: add esp, 0xc
0041c47a: push ss:[ebp+var_4], 0x19
0041c47d: lea edx, ss:[ebp+var_3C]
0041c480: push edx

```

図 2 生成された変換例 1：左（変種）の赤色部分は“jmp”命令、右（オリジナル）の青色部分が遷移先に移動。

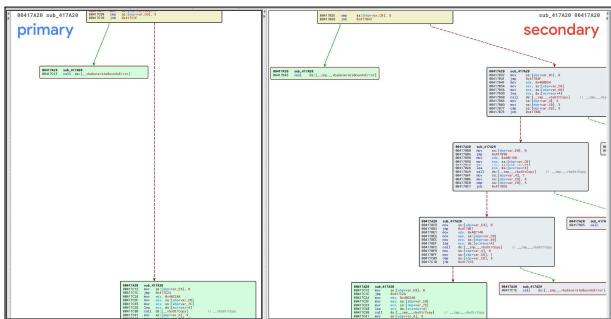


図 3 生成された変換例 2：右（オリジナル）の灰色ブロック群（エラーハンドリング）が左（変種）では削除されている。

Totalにおける検知状況を測定した。図2で示した変換例のサンプルにおいて、元のサンプルは72エンジン中65エンジンで検知されていた。一方、提案手法により生成された変種は、72エンジン中61エンジンで検知された。

この検知率の低下は限定的ではあるものの、重要な示唆を含んでいる。現段階の提案手法は検知回避を主目的として設計されていないにもかかわらず、一部の検出エンジンが検知を失敗している。これは、該当エンジンがサンプル固有の静的シグネチャ（固定バイト列、セクション特性、粗い構造特徴等）に強く依存していることを示唆している。現状の検知回避効果は、静的シグネチャベースの検出に対する偶発的な回避にとどまっており、より高度な動的解析や機械学習ベースの検出器に対する効果は限定的である可能性が高い。しかしながら、防御側が攻撃者に先んじて変種を生成し、訓練データとして活用することで、未知変種への対応能力を向上させる可能性は示されたと言える。

今後、検知回避を明示的な目的として変換戦略を最適化することで、より効果的な検知回避を実現できると考える。例えば、ViTをマルウェア検知タスクで訓練し、検知に寄与する領域を抽出、変換することで、検知回避率の向上が期待できる。このような拡張により、検出器強化により貢献できる技術へと発展する可能性がある。

## 5. 議論

### 5.1 制限と展望

本研究の制限事項と展望について、以下の点を挙げる。

まず、現状の24%という機能維持率は、実用化に向けて改善の余地がある。機能維持に失敗したサンプルの分析により、LLM出力の問題パターンを発見した。主要な失敗要因として、(1) 無効アドレスへのジャンプ命令生成（実行可能領域外への参照）、(2) 制御フローの循環構造による無限ループ、(3) 関数呼び出し規約の破壊（プロローグ・

エピローグの不整合）、(4) x86命令形式の誤用（サイズ指定欠落、不正な間接参照等）が確認された。これらは、LLMがアセンブリ言語の構文的側面は学習しているものの、実行時制約やメモリレイアウト、アーキテクチャ固有の規則を完全には理解していないことに起因する。これらの問題に対して、ガイドラインベースの制御機構の導入と失敗時の再生成機構の導入が有効と考えられる。具体的には、変換時の制約条件を明示的なルールセットとして定義し、LLMの出力を制御する手法である。例えば、(1) 有効アドレス範囲の明示的指定、(2) スタック操作の対称性保証、(3) レジスタ使用規約の遵守、(4) 制御フロー整合性の維持、といった制約をガイドラインに組み込むことで、多くの失敗パターンを回避できると考えられる。

次に、生成された変種が実際にマルウェア検出器の性能向上に寄与するかについては、更なる検証が必要である。今後、生成された変種が検出器の汎化性能向上に貢献するかを従来手法と比較評価する必要がある。さらに、ViTの注意機構が特定する領域と、実際の同一ファミリーサンプル間の差分が一致するかについても定量的な評価が必要である。この相関を明らかにすることで、提案手法における領域選択の妥当性を示すことができる。

また、本研究では複数のLLMによる変種の質や機能維持率の変化については未評価である。

本研究の展望として、変種生成以外にも応用可能であると考えている。特に有望な応用として、パッキングや難読化されたマルウェアの自動解析支援が挙げられる。ViTの注意機構を用いてパッカーのスタブ部分を特定し、LLMにアンパッキング用のパッチを生成させることで、自動的な難読化解除が実現できる可能性がある。

### 5.2 倫理的配慮

本研究は、以下の倫理的配慮のもとで研究を実施した。

まず、本研究の目的は、防御技術の向上を通じたセキュリティの強化にある。生成される変種は、検出器の訓練データとして活用することで、未知マルウェアへの対応能力向上に寄与することを意図している。これは、攻撃者が継続的に新たな変種を生成する現状において、防御側が先回りして対策を行うための重要な取り組みである。

次に、悪用防止の観点から、本研究では以下の措置を講じている。第一に、実装コードの公開を制限し、信頼できる研究者に対してのみ、個別審査のうえでアクセスを許可する方針を採用している。第二に、生成された変種についても同様に、公開データセットとしての配布は行わず、正当な研究目的での利用に限定している。第三に、本論文では手法の詳細を記述しつつも、悪意ある実装を容易にするような実装の詳細については意図的に省略している。また、本研究はすべての実験を隔離された環境下で実施し、生成された変種が外部に流出することがないよう管理した。

## 6. 結論

本研究では、ViT と LLM を組み合わせた機能保持型マルウェア変種自動生成フレームワークを提案した。提案手法の核心は、実行ファイルを画像化して ViT の注意機構によりマルウェアの重要領域を特定し、その領域のアセンブリコードを LLM で意味等価に変換することである。この手法により、ソースコードへのアクセスを必要とせず、バイナリファイルから直接、機能を保持した変種を生成することが可能となった。実験では、BODMAS データセットから選定した 100 サンプルの内、24 サンプルにおいて完全な機能維持を確認し、CFG レベルでの構造的変換を実現した。生成された変種には、トランポリン構造による制御フロー再構成や不要なエラーハンドリング分岐の削除など、高度な変換が含まれていた。これは従来手法では実現困難であった成果であり、検出器の頑健性向上に有効なデータ拡張手法としての可能性を示している。

今後の研究課題として、LLM 出力制御の改善による機能維持率の向上、生成変種の検出器強化への貢献度の定量的評価、ViT 注意機構による注目箇所の妥当性検証が挙げられる。また、パッキングや難読化されたマルウェアの自動解析支援に応用することで、直接的な貢献も期待できる。

## 参考文献

- [1] M. M. Belal and S. D. Meena. Global-local attention-based butterfly vision transformer for visualization-based malware classification. *IEEE Access*, 11:69337–69355, 2023.
- [2] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-G. Wang, and J. Chen. Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics*, 14(7):3187–3196, 2018.
- [3] M. Ebrahimi, N. Zhang, J. Hu, M. T. Raza, and H. Chen. Binary black-box evasion attacks against deep learning-based static malware detectors with adversarial byte-level language model. *CoRR*, abs/2012.07994, 2020.
- [4] Hex-Rays. Ida pro - the interactive disassembler. <https://www.hex-rays.com/ida-pro/>, 2023.
- [5] J. L. Hu, M. Ebrahimi, and H. Chen. Single-shot black-box adversarial attacks against malware detectors: A causal language model approach. In *2021 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 1–6, 2021.
- [6] A. Kolesnikov, A. Dosovitskiy, D. Weissenborn, G. Heigold, J. Uszkoreit, L. Beyer, M. Minderer, M. Dehghani, N. Houlsby, S. Gelly, T. Unterthiner, and X. Zhai. An image is worth 16x16 words: Trans-
- [7] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. *CoRR*, abs/1803.04173, 2018.
- [8] F. Lu, Z. Cai, Z. Lin, Y. Bao, and M. Tang. Research on the construction of malware variant datasets and their detection method. *Applied Sciences*, 12(15):7546, 2022.
- [9] P. Madani. Metamorphic malware evolution: The potential and peril of large language models. *arXiv*, 2024.
- [10] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, 2011.
- [11] A. Q. Nguyen and H.-V. Dang. Keystone: The ultimate assembler. <https://www.keystone-engine.org/>, 2016.
- [12] OpenAI. Openai api reference. <https://platform.openai.com/docs/>, 2024.
- [13] M. O. F. Rokon, R. Islam, A. Darki, E. E. Papalexakis, and M. Faloutsos. SourceFinder: Finding malware Source-Code from publicly available repositories in GitHub. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 149–163, San Sebastian, Oct. 2020. USENIX Association.
- [14] M. Wong, E. Raff, J. Holt, and R. Netravali. Marvolo: Programmatic data augmentation for deep malware detection. In *Machine Learning and Knowledge Discovery in Databases: Research Track: European Conference, ECML PKDD 2023, Turin, Italy, September 18–22, 2023, Proceedings, Part I*, page 270–285, Berlin, Heidelberg, 2023. Springer-Verlag.
- [15] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma. Making malware for what it is: Malware classification with attention mechanism on byte-level image representations. *Computers & Security*, 87:101592, 2019.
- [16] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang. Bodmas: An open dataset for learning based temporal analysis of pe malware. In *4th Deep Learning and Security Workshop*, 2021.