# パッカーの自動解凍に向けた機械学習による解凍処理識別手法の提案

山根 一真[1,a)]　掛井 将平[1]　齋藤 彰一[1]

**概要**：マルウェアの利用するパッカーは進化を続けており，近年には長年有効とされていた情報エントロピーの特徴を利用した検出を免れるものも出現し始めている．本稿では，高度なパッカーに対応した自動解凍の実現に向け，実行した命令列がパッカー特有の解凍処理であるか否かを識別する手法を提案する．識別は，従来のシグネチャベースの識別手法や情報エントロピーに依存した識別手法で不足していた，未知のパッカーに対するロバスト性を高めるため，動的な命令追跡により得られる命令シーケンスを用いて，事前学習を行った機械学習モデルにより判定する．提案手法の有効性を示すため，計 25 種の既知のパッカーを基にデータセットを作成し，既知のパッカーに対する識別性能の評価を行った．また，いくつかのパッカーを学習対象から外した上で学習に用いていないパッカーに対する識別性能の評価を行うことで，未知のパッカーに対する識別性能の評価も行った．加えて，命令シーケンスの長さと識別性能の関係を調査することで，十分な精度を出すために要する命令シーケンスの長さを明らかにした．

**キーワード**：アンパッキング，機械学習，動的解析，命令シーケンス解析

# A Machine Learning-Based Decompression Process Identification Method for Automatic Unpacking

Kazuma Yamane[1,a)]　Shohei Kakei[1]　Shoich Saito[1]

***Abstract:*** Packers used by malware continue to evolve, and in recent years, some have emerged that evade detection methods based on information entropy – an approach that had long been considered effective. This paper proposes a method to identify whether a sequence of executed instructions corresponds to unpacking routines specific to packers, aiming to implement automated unpacking for advanced packers. Unlike conventional signature- or entropy-based techniques, the proposed method enhances robustness against unseen packers by using dynamically traced instruction sequences classified by a pre-trained machine learning model. To evaluate effectiveness, we built a dataset from 25 known packers and assessed identification performance. Additionally, by excluding some packers from the training set and evaluating the model on hold-out packers, we assessed the method's ability to classify unseen packers. We also analyzed the impact of instruction sequence length on classification accuracy, identifying the minimum length needed for reliable performance.

***Keywords:*** Unpacking, Machine Learning, Dynamic Analysis, Instruction Sequence Analysis

## 1. はじめに

Windows ユーザにとって，マルウェアは依然として大きな脅威である．高度なマルウェアの中には，コードの解析を防ぐため，独自のパッカーを適用したものがある [1]．パッカーとは，実行可能ファイルを，実行可能な状態を保ったまま圧縮するツールのことである．パッカーにより圧縮された実行可能ファイルは，圧縮前のプログラムとはまったく異なる特徴を有する．そのため，本来の用途である圧縮によるストレージ容量の削減の他，静的解析や表層

[1] 名古屋工業大学
Nagoya Institue of Technology
[a)] k.yamane.826@nitech.jp

解析を妨害するためにも使用されている [2]．このようなパッカーを用いたマルウェアの解析に対しては，動的解析を用いて解凍後のコードを抽出し，本来のプログラムを得るという，解凍操作を事前に行うことが求められる．

高度なパッカーは，複数の圧縮層を持つことや関数単位で解凍することによって，プログラムの解凍を一層困難としている．これらの高度なパッカーによって圧縮されたプログラムを，第三者が正確に解凍するためには，ある実行可能プログラムが，解凍処理を行うためのコードであるか否かを識別できることが重要である．しかし，近年には，圧縮前後で情報エントロピーが変化しにくいパッカーが確認されたことで，正確な解凍処理箇所の識別が課題となっている [3]．

本稿では，動的解析により実行済み命令シーケンスを生成し，これを用いて事前学習した機械学習モデルによって，プログラムが解凍処理を行っているか識別する手法を提案する．命令シーケンスの作成にあたっては，著者らの提案する軽量な細粒度命令追跡手法 [4] を用いることで，Windows API 中の命令を省略し，解析対象プログラム固有の命令のみを含むようにしている．これにより，パッカーに必然的に共通する特性である，圧縮されたコードを解凍する処理そのものにのみ依存することで，既存手法に比べ検出回避の難しい識別の実現を目指す．

以下，2 章では関連研究の紹介を行い，パッカーの自動解凍の現状と，本研究の位置づけについて述べる．3 章では提案手法の概要について述べ，4 章で提案手法に対して行った各種評価について述べる．5 章では評価結果に対する解釈と今後について述べた後，最後に 6 章で本研究をまとめる．

## 2. 関連研究

パッカーにより圧縮されたプログラムの自動解凍のために，さまざまな研究が行われている．ここでは，複数の解凍レイヤーを持つ高度なパッカーを考慮し，既存研究における展開されたコードの抽出タイミングに焦点を当て，本稿における研究課題を明らかにする．

### 2.1 古典的な研究

この節では，過去の著名な研究の成果から，コードの抽出タイミングに関して前提となる背景について紹介する．

#### 2.1.1 PolyUnpack[5]

PolyUnpack は，静的解析により得られたコードと動的解析により得られたコードの差分を抽出することで解凍後のコードを取得する．差分検出をベースにしていることから，抽出対象は主に最初に展開されたコードとなるため，後から展開されて既存コードを上書きして現れるコードを見落とす可能性がある．そのため，初めに圧縮された元のコードを解凍する処理を行うコードを展開し，その後，同じメモリに圧縮された元のコードを展開するパッカーに対しては，正常に動作しない課題が存在する．

#### 2.1.2 Renovo[6]

Renovo は，動的に書き込まれたメモリ上のコードが実行されるたびにメモリをダンプすることで解凍後のコードを抽出する．書き込み操作が行われたメモリが実行される度にコードの抽出・更新を行うため，最終的には最後に展開されたコードのみが抽出される可能性が高い．これにより，PolyUnpack に見られた課題は解消された一方で，解凍後のコードの実行後にダミーコードを上書きするようなパッカーに対しては正常に動作しない課題が存在する．

#### 2.1.3 OmniUnpack[7]

OmniUnpack は，実際に実行したタイミングで，実行したコードのみを抽出することで解凍後のコードを抽出する．これにより，PolyUnpack および Renovo での課題を解消する一方，実行されなかった分岐先のコードを抽出することができないといった課題が存在する．

### 2.2 近年の研究

前節では，古典的な研究による成果から，抽出タイミングを機械的に設けるのみでは十分に対応できないケースが存在することを明らかにした．本節では，近年の研究の抽出タイミングに対するアプローチを述べ，現在の課題について明らかにする．

#### 2.2.1 API-Xray[8]

API-Xray は，解凍終了後には IAT 再構築が行われることを前提とし，再構築された IAT が利用された時点を解凍完了と見做す．抽出をメモリマップ単位で行うことで，関数単位での復号が行われるケースにも対応を可能としている．しかし，解凍終了を IAT の利用に依存していることから，IAT が用いられない場合，コードを抽出できない課題が存在する．また，マルウェアは，フックのバイパスのために IAT を用いないことがあるため，マルウェア解析には抽出精度が悪くなる特徴が存在する．

#### 2.2.2 LoopHPCs[9]

LoopHPCs は，ダンプ対象メモリ中のループに際してハードウェアパフォーマンスカウンタ（HPCs）を計測し，事前学習済みモデルに HPCs プロファイル情報を与えることで，実行中のコードが解凍処理か識別する．正確な判定に事前学習済みのモデルを必要とする上に，HPCs はハードウェアに強く影響を受けるため，ハードウェアごとにモデルの学習が必要となることから，汎用性に課題が存在する．

#### 2.2.3 Xunpack[10]

Xunpack は，展開されたコードすべてを一時的にダンプした上で，最終的に SelectiveDump という手法により最も圧縮前のオリジナルのコードであると考えられるダンプを選択することで解凍後のコードを抽出する．Selective-

Dump では，情報エントロピーを用いた判定を利用しているため，情報エントロピーに特徴が表れないパッカーに対しては正常に動作しない課題が存在する．

## 3. 提案手法

2 章より，現状の自動解凍には解凍されたコードが，圧縮前のオリジナルのコードであるかを識別する手法に課題が存在することが明らかとなった．本章では，この課題を解決するために本稿が提案する新たな解凍処理識別手法について述べる．本提案手法が対象とするプログラムは以下のとおりである．

- **動作 OS:** Windows
- **フォーマット:** PE
- **ISA:** IA-32

提案手法の全体像を図 1 に示す．提案手法は，機械学習モデルの学習と，機械学習モデルによるコード推定の二つのフェーズから成る．以下に，各フェーズの詳細について述べる．

### 3.1 学習フェーズ

学習フェーズでは，パッカー特有の解凍処理を行うプログラムと，解凍処理を行わないプログラムを含むリポジトリから，データセットを作成し，機械学習モデルを作成する．

#### 3.1.1 データセット [11] の作成

データセットは，各プログラムの持つ解凍処理及び，解凍処理以外の処理から作成した命令シーケンスと，各命令シーケンスの作成元コードが解凍処理であるかを示すラベルのペアにより JSON 形式のファイル群として構成する．本稿では，命令シーケンスの作成の元となるリポジトリとして，良性プログラムと，それらを計 25 種類のパッカーを用いてパックしたプログラムを含むデータセット [12] を利用した．

各プログラムの命令シーケンスは，我々が提案 [4] している動的解析を用いた命令追跡により，各プログラムのエントリーポイントから 10,000 命令を追跡し，それぞれの命令シーケンスを作成した．パッカーにより圧縮される前のプログラムの先頭 10,000 命令には解凍処理を含んでいない一方で，パッカーにより圧縮されたプログラムの先頭 10,000 命令には解凍処理が含まれると考え，それぞれラベルを付けた．これらの構成比率を図 2 に示す．

リポジトリに含まれるパッカーにより圧縮される前のプログラム計 435 個のうち，IA-32 ではない 4 個，管理者権限を要求する 39 個，実行時に要求される DLL が存在せず実行を開始できない 189 個を除いた 203 個を対象として命令シーケンスを取得した結果，174 個のプログラムから命令シーケンスの作成に成功した．このうち，10,000 命令の追跡を終える前に終了した 3 個を除いた，計 171 個の命令シーケンスを，非解凍処理を意味する「False」ラベル付きの命令シーケンスとした．同様に，リポジトリに含まれる，各パッカーにより圧縮されたプログラム計 3,080 個のうち，IA-32 ではない 3 個，管理者権限を要求する 405 個，実行時に要求される DLL が存在せず実行を開始できない 90 個を除いた 2,582 個を対象とし，結果，2,292 個のプログラムから命令シーケンスの作成に成功した．このうち，10,000 命令の追跡を終える前に終了した 288 個を除いた，計 2,004 個の命令シーケンスを，解凍処理を意味する「True」ラベル付きの命令シーケンスとした．なお，計 25 種のパッカーのうち，追加のウィンドウの表示によって命令実行を阻害する特徴を持つ「Exe32pack」および「EXpressor」の 2 つのパッカーについては，作成に成功したすべてのケースにおいて命令シーケンスが 10,000 命令に満たなかったため，最終的に 23 種のパッカーのみを対象としたデータセットとなった．

#### 3.1.2 命令表現形式

各命令を表現する形式として，オペコードで表現する形式やニーモニックにより表現する形式が考えられる．これらの手法は容易に実装できる一方，オペランドを考慮していないため，大きく異なる動作をする命令を区別することなく表現している．近年では，より正確に命令を表現するため，ニーモニックと抽象化したオペランドの情報を組み合わせて一つの命令を表現する手法が提案されている [13]．本稿では，命令シーケンスにおける各命令を表現する形式として，図 3 に示す，オペコードのみで命令を表現する形式，ニーモニックのみで命令を表現する形式に加え，ニーモニックとオペランドがメモリを参照の有無と，メモリを参照している場合には参照先のメモリ保護情報を合わせて命令を表現する形式の三つを検討する．

#### 3.1.3 モデルの作成

本稿では，ロジスティック回帰に基づく機械学習モデルを作成し，推定を行う．学習では，作成したデータセットのうち，命令シーケンス群はベクトライザにより値行列 $\mathbf{X}$ に，ブール値のベクトルは 1 / 0 のバイナリベクトル $\mathbf{y}$ に変換し，パラメータ $\mathbf{w}$ を最適化する．今回作成した命令シーケンスは，各命令を 1 単語と見做すことで，一般的な文章のように単語のリストと見做すことが可能である．このような文字列に対するベクトル化手法としては，TF-IDF アルゴリズムによるベクトル化，ワンホットエンコーディング，ラベルエンコーディングが考えられる．IA-32 自体の命令の多様性に加え，独自の命令表現形式においては各オペランドの情報を含めて一つの単語を成すことから，語彙数が爆発的に増加する可能性があること，命令の出現頻度に応じた重要度の重み付けが効果的に働くことを期待し，本稿では TF-IDF アルゴリズムに基づくベクトライザを利用する．TF-IDF アルゴリズムによりベクトル化を行う場合，ベクトル化に際してコーパスを共有する必要がある．

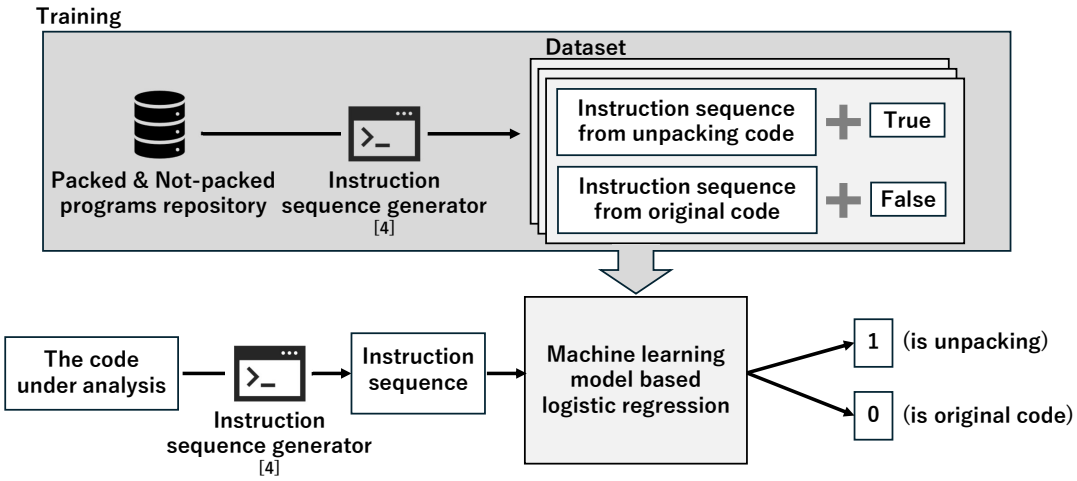


図 1: 提案手法の全体像

**Fig. 1** Overview of the proposed method.

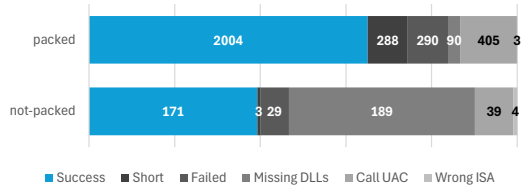


図 2: 作成に成功した命令シーケンスの割合

**Fig. 2** Ratio of successfully generated instruction sequences.

```
{
  "opcode": "A3"
}
```

(a) オペコードのみによる命令表現の例

```
{
  "mnemonic": "mov"
}
```

(b) ニーモニックのみによる命令表現の例

```
{
  "mnemonic": "mov",
  "dest": "MemoryAccess{RW}",
  "src1": "NonMemoryAccess",
  "src2": "None"
}
```

(c) 独自形式による命令表現の例

図 3: 各命令表現形式の例

**Fig. 3** Examples of each instruction expression.

学習フェーズと推定フェーズでコーパスを共有するため，作成したモデルと同時にコーパス入力後のベクトライザも併せて保存する．

### 3.2 推定フェーズ

推定フェーズでは，前節で述べた学習フェーズで作成した機械学習モデルを用いて，対象コードが解凍処理であるかを識別する．

はじめに，解析対象コードから，データセット作成時同様に，我々の提案手法 [4] により命令シーケンスを作成する．その後，モデル作成時に保存したベクトライザを用いてベクトル化することで，ベクトル $\mathbf{x}$ に変換する．作成したベクトルをモデルに入力することで解凍処理であるかの識別結果を 1 / 0 のバイナリ値として得る．1 であれば入力が解凍処理であることを，0 であれば入力が解凍処理ではないことを示す．

## 4. 評価

提案手法に対し，未知のパッカーに対する判定精度の評価と，命令表現形式別の判定精度の比較を行った．加えて，各命令表現形式について命令シーケンスの長さを変更したときの判定精度の変化を調査した．

### 4.1 未知のパッカーに対する判定精度の評価

作成したデータセットに含まれる計 23 種のパッカーを二つのグループに分け，一方のグループを学習に，他方のグループをテストに用いることで，学習に含まれないパッカーの解凍処理コードに対する判定精度を評価し，未学習のパッカーに対する判定精度を調べることで，疑似的に未知のパッカーに対する判定精度を調査した．

#### 4.1.1 パラメータの設定

Lim ら [14] は，パッカーによる解凍処理に対する観察から，5,000 バイトを超える書き込みがあるメモリセクションは命令が展開されている可能性が高いことを報告した．5,000 バイト中に含まれる命令は最大で 5,000 命令であることから，ここでは 5,000 命令追跡して得られる命令シーケンスで学習・推定を行う．5,000 命令の命令シーケンスについては，3.1.1 節で作成した 10,000 命令の命令シーケンスの先頭 5,000 命令を用いて作成し，データセットを再構築した．また，命令シーケンス中の各命令は，独自形式による表現とした．

本稿における学習および推定では，データセット作成に用いたリポジトリの特性上，True ラベル付きデータと False ラベル付きデータの数に大きな差が生じている不均衡データとなっている．実際の運用においては，必ずしも同様の偏りが生じているとは考えられないことから，学習時における不均衡性を排するため，学習データ量に応じて自動でクラス間の重み付けを実施した上で学習する *class_weight* 設定を有効とした．

ロジスティック回帰分析において設定可能な正則化パラメータ C および，正則化項は，グリッドサーチによって最適な設定値の探索を行った．探索範囲は下記の通りである．

- **C:** $10^{-4}$ から $10^{4}$ より対数スケールで均等に 10 個
- **正則化項:** $L1$, $L2$

各組合せのスコアを測る評価指数としては，不均衡データに対して公平に評価可能な下記 5 つを用い，それぞれの評価指標において最適と判断された組み合わせすべてを平行にテストし，最も結果の良いものを最終的に得られる結果とした．

- Average Precision
- Balanced Accuracy
- F1 Macro
- 対数損失 (Log Loss)
- ROC 曲線下面積 (AUC)

#### 4.1.2 データセットの分離

本節では，データセットをテストデータと学習データに分離する方法について述べる．

未知のパッカーは，既知のパッカーに対し，出現時期が遅くなっていることから，より高度な解凍処理を持つように変化していると予想される．したがって，計 23 種のパッカーのうち，比較的出現時期の早いものを学習に用い，出現時期の遅いものをテストに利用することを考える．しかし，多くのパッカーは正確な起源の特定が困難であり，また，パッカーの中には，登場時期は早くとも，更新が続けられ，公開初期とは大きく仕組みを変えたものも存在すると考えられる．そこで，古典的な静的パッカー識別器である PEiD[15] を利用し，PEiD により識別できるパッカーについては登場時期が早く，そうでないものについては比較的新しいパッカーであると考えることとし，パッカーの分類を行った．表 1 に，PEiD による各パッカーの検出精度を示す．

表 1 の結果より，検出精度が 0 でない 12 種をより古いパッカーと見なし，計 910 個の命令シーケンスを学習に用いる．同様に，検出精度が 0 となった 11 種をより新しいパッカーと見なし，計 1,094 個の命令シーケンスをテストに用いる．

また，同一のパッカーにより圧縮されたプログラムは，命令シーケンスの大部分が共通していることがある．この特徴による重複したデータの学習を避けるため，各長さ 10,000 の命令シーケンスについて，他の学習対象の長さ 10,000 の命令シーケンスとのコサイン類似度を算出し，類似度が 0.99 を超えるものについては，それらの内一つのみを学習に用いることとした．その後，長さ 10,000 の命令シーケンスの重複排除の結果を基に，長さ 5,000 の命令シーケンスの一部を学習対象から排除した．その結果，最終的にパッカー適用済みのプログラムから得られた命令シーケンスについては，計 12 種のパッカーから 144 個の命令シーケンスのみを学習に用いる．

一方で，パッカー適用前のプログラムは，互いに無関係のプログラムであることから，単純にランダムに等分割し，一方のすべてを学習に用い，他方のすべてをテストにのみ用いる．

表 1: PEiD による各パッカーの検出精度
**Table 1** Detection accuracy of each packer using PEiD.

| パッカー名 | 検出対象数 | 検出成功数 | 検出精度 [%] |
|---|---|---|---|
| Alienyze | 92 | 0 | 0 |
| Amber | 145 | 0 | 0 |
| ASPack | 97 | 97 | 100 |
| BeRoEXEPacker | 94 | 0 | 0 |
| Enigma Virtual Box | 67 | 0 | 0 |
| Eronana Packer | 124 | 0 | 0 |
| FSG | 37 | 37 | 100 |
| JDPack | 8 | 8 | 100 |
| MEW | 114 | 108 | 95 |
| Molebox | 102 | 0 | 0 |
| MPRESS | 93 | 0 | 0 |
| Neolite | 74 | 74 | 100 |
| NSPack | 100 | 0 | 0 |
| Packman | 92 | 0 | 0 |
| PECompact | 103 | 59 | 57 |
| PEtite | 129 | 108 | 84 |
| RLPack | 92 | 0 | 0 |
| TELock | 30 | 30 | 100 |
| Themida | 93 | 0 | 0 |
| UPX | 99 | 37 | 37 |
| WinUpack | 99 | 98 | 99 |
| Yoda-Crypter | 91 | 91 | 100 |
| Yoda-Protector | 29 | 29 | 100 |

#### 4.1.3 評価結果

Balanced Accuracy および F1 Macro を評価指数としたとき，正則化パラメータ C が 21.544，正則化項が $L2$, また, Log Loss を評価指数としたとき，正則化パラメータ C が 1291.5, 正則化項が $L2$ となり，最良な結果が得られた．各評価指標ごとの結果を表 2 に示す．

ここで，各モデルの統一的な評価指標として，Balanced Accuracy および Matthews Correlation Coefficient（MCC）を導入した．Balanced Accuracy は，評価データの総数の違いによる影響を受けにくく，直感的な精度がわかりやす

表 2: 未学習のパッカーに対する判定精度の評価結果
**Table 2** Results of judgment accuracy for untrained packers.

| | テストデータに対する結果 | | | | | | 学習済みデータに対する結果 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scoring | TP | FN | FP | TN | MCC | Balanced Acc. | TP | FN | FP | TN | MCC | Balanced Acc. |
| Average Precision | 812 | 282 | 0 | 85 | 0.41462 | 0.87112 | 902 | 8 | 0 | 86 | 0.95229 | 0.99560 |
| Balanced Accuracy | 903 | 191 | 0 | 85 | 0.50419 | 0.91271 | 910 | 0 | 0 | 86 | 1.0 | 1.0 |
| F1 Macro | 903 | 191 | 0 | 85 | 0.50419 | 0.91271 | 910 | 0 | 0 | 86 | 1.0 | 1.0 |
| Log Loss | 903 | 191 | 0 | 85 | 0.50419 | 0.91271 | 910 | 0 | 0 | 86 | 1.0 | 1.0 |
| ROC AUC | 812 | 282 | 0 | 85 | 0.41462 | 0.87112 | 902 | 8 | 0 | 86 | 0.95229 | 0.99560 |

い．その一方で，データ数と均衡の程度が等しい不均衡クラスにおいては，MCC を用いることでより公平な比較が可能である．

表 2 より，テストデータと学習済みデータのいずれにおいても，Balanced Accuracy および F1 Macro, Log Loss で MCC が最高値となっており，この三つの評価指標を用いた探索の結果最良となったことがわかる．またいずれの評価指標による結果も，テストデータに対する結果では特異度に比べて再現率が低くなっている．すなわち，Negative データ，つまり，パッカー未適用のプログラムから作成された命令シーケンスに対する推測に強い一方で，Positive データ，つまり，パッカー適用後のプログラムから作成された命令シーケンスに対する推測が悪くなっている傾向が見て取れる．

しかし，比較的低い再現率も 82.5%を超えていることから，既知のパッカーの情報のみを用いて，より新しい未知のパッカーに対する推測をすることは十分に可能であると考える．また，学習済みデータに対する結果は，いずれも 100%の精度となった．特に，Positive データに関しては，実際には 144 個しか学習には用いていなかったが，910 個すべてで正しく推測できていることから，重複を排除して学習データを減らすアプローチは，過学習を避けつつ既知データの精度を維持するのに効果的であったと考える．

### 4.2 命令表現形式別の判定精度の比較

他の設定を変えず命令の表現形式のみを「オペコードのみ」,「ニーモニックのみ」,「独自形式」に変化させることによ ri 判定精度の変化について調査する．これにより，独自の命令表現形式導入による精度向上を評価する．

#### 4.2.1 パラメータの設定

命令シーケンスの長さは，4.1 節での評価と同様，5,000 命令とした．また，*class_weight* の設定も同様に有効とした．他，ロジスティック回帰分析における正則化パラメータ C および，正則化項の探索範囲についても同様とした．

#### 4.2.2 データセットの分離

データセットの分離方法についても，4.1 節での評価と同様とした．一方で，重複排除後のパッカー適用済みプログラムから作成された命令シーケンスの数については，それぞれの命令表現形式毎に差が生じ，それぞれ,「オペコードのみ」が 145 個,「ニーモニックのみ」が 141 個,「独自形式」が 144 個となった．ただし，いずれもパッカーの種類数は 23 種で共通しているため，学習の結果に及ぼす差は少ないと考える．

#### 4.2.3 評価結果

「オペコードのみ」では，Balanced Accuracy および F1 Macro を評価指数としたとき，正則化パラメータ C が 21.544，正則化項が $L2$ となり，最良の結果が得られた．また,「ニーモニックのみ」では，Balanced Accuracy および F1 Macro を評価指数としたとき，正則化パラメータ C が 1291.5，正則化項が $L2$，Log Loss を評価指標としたとき, 正則化パラメータ C が 1291.5，正則化項が $L1$ となり，最良の結果が得られた.「独自形式」については 4.1 節での評価結果と同様である．各評価指標ごとの結果を表 3 に示す．

表 3 を見ると，テストデータに対する結果より,「オペコードのみ」よりも僅かに「ニーモニックのみ」が優れており,「独自形式」は他と比べ，より高い精度で推測出来ていることがわかる．これにより，少なくとも長さ 5,000 の命令シーケンスから推測するにあたっては,「独自形式」が「オペコードのみ」や「ニーモニックのみ」で命令を表現する以上に命令の情報を伝えることができるようになっていると言える．

### 4.3 命令シーケンス別の判定精度の比較

他の設定を変えず，命令シーケンスの長さのみを変化させることで，命令シーケンスと判定精度の関係を調査した．

#### 4.3.1 パラメータの設定

命令シーケンスの長さは，1,000 から 10,000 まで，1,000 刻みで変化させる．各長さの命令シーケンスの作成については，長さ 10,000 の命令シーケンスの先頭からそれぞれ抜き出すことで用意するものとし，各長さ毎のデータセットを再構築した．命令表現形式毎に長さを変化させたときに異なる変化を見せる可能性を考慮し，いずれの長さにおいても三種類の命令表現形式により判定精度の確認を行うこととする．その他，*class_weight* の設定ならびに，ロジスティック回帰分析における各種パラメータの探索範囲につ

表 3: 命令表現形式別の判定精度の比較結果
**Table 3** Results of comparison of accuracy of judgment for each instruction expression.

| | テストデータに対する結果 | | | | | | 学習済みデータに対する結果 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Expression | TP | FN | FP | TN | MCC | Balanced Acc. | TP | FN | FP | TN | MCC | Balanced Acc. |
| Opcode | 710 | 384 | 0 | 85 | 0.34296 | 0.82450 | 910 | 0 | 0 | 86 | 1.0 | 1.0 |
| Mnemonic | 719 | 375 | 0 | 85 | 0.34849 | 0.82861 | 910 | 0 | 0 | 86 | 1.0 | 1.0 |
| Original | 903 | 191 | 0 | 85 | 0.50419 | 0.91271 | 910 | 0 | 0 | 86 | 1.0 | 1.0 |

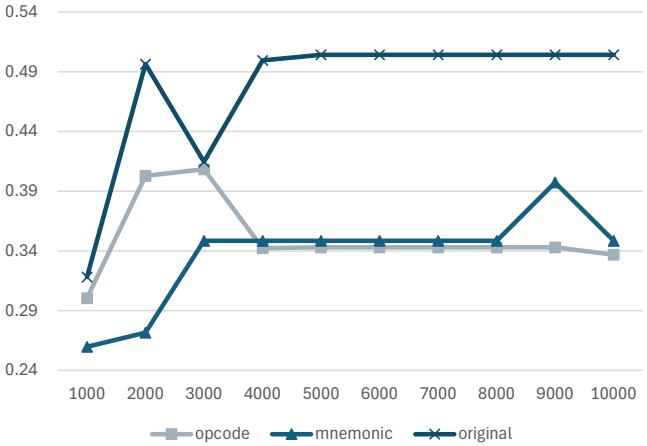


図 4: 各命令表現形式における命令シーケンス長の変化に伴う MCC の遷移
**Fig. 4** Transition of MCC with changes in instruction sequence length for each instruction expression.

いては 4.1 節での評価と同様とした．

#### 4.3.2 データセットの分離

データセットの分離方法についても，4.1 説での評価と同様とした．ただし，重複排除については，それぞれの長さでの重複排除結果に関係なく，長さ 10,000 での結果に基づき重複排除を行った．また，4.2 節同様，重複排除の結果，実際に学習に用いるパッカー適用済みプログラムから作成された命令シーケンスの数は，命令表現形式毎に異なる．

#### 4.3.3 評価結果

各命令表現形式別命令シーケンス長別の結果を表 4 に，各命令表現形式における命令シーケンス長の変化に伴う判定精度の遷移を図 4 に示す．

図 4 および表 4 から，いずれの命令表現形式においても，細かな差異は存在するものの，長さ 1,000 から長さ 4,000 程度までは長さを増やすごとに精度が増加する一方で，それ以降は長さを増加させても精度はほとんど変化しない共通の傾向が見て取れる．抽出したい解凍処理の命令シーケンス長が短いとすると，長さ 10,000 の命令シーケンス長の前半で解凍処理を終えるため，後半部分については判定精度の向上に寄与しない．このことから，現行のパッカーの識別には，命令表現形式に依らず，今回調査した範囲では，長さ 4,000 が最適な長さだと考えられる．

## 5. 議論

評価結果は，いずれも Positive データに対する判定精度が，Negative データに対する判定精度よりも大きく劣る結果となった．多くのパッカー適用済みプログラムでは，元のプログラムのエントリーポイントに先んじて解凍処理が行われる．このため，プログラムのヘッダ情報から得られるエントリーポイントは，圧縮されたコードの解凍を含むパッカー固有の処理の先頭を指すこととなる．したがって，パッカー適用済みプログラムについては，エントリーポイントから追跡することで，解凍処理を含む命令シーケンスを得られる可能性が高いと考える．一方，解凍処理を含まないコードの取得は，本稿では，パッカー未適用のプログラムのエントリーポイントから追跡することで行っている．パッカー適用済みプログラムとパッカー未適用のプログラムとの差を明らかにするという意味では妥当であるが，一般にエントリーポイント直後から独自の処理を行うプログラムは少なく，多くはコンパイラによって挿入されるスタートアップルーチンが存在していることから，「解凍処理でないコード」の大部分がスタートアップルーチンとなっていたのではないかと考える．その結果，非解凍処理に存在する多様性を十分に学習することができず，そのコードが解凍処理か否かではなく，スタートアップルーチンに近いか否かで判断されてしまった可能性がある．また，パッカー適用済みプログラムから作成された命令シーケンスの数に対し，パッカー未適用のプログラムから作成された命令シーケンスの数が少ないために，少ないデータに対する誤判定が大きなマイナスとなるといったバイアスが働き，悪影響を与えた可能性が考えられる．したがって，単にエントリーポイントからの命令シーケンスだけでなく，各関数先頭アドレスからの追跡を開始させ命令シーケンスを生成させることで，「解凍処理でないコード」に多様性を取り入れつつ，クラス間の不均衡を解消することが考えられる．

## 6. まとめ

本稿では，関連研究から現状の自動解凍において，動的に展開されるコードのうち，圧縮前の処理を含んだコードのみを抽出することが課題となっていることを確認した．

表 4: 命令表現形式別命令シーケンス長別の判定精度の比較結果
**Table 4** Results of comparison of judgment accuracy by instruction expression and instruction sequence length.

| | Opcode | | Mnemonic | | Original | |
|---|---|---|---|---|---|---|
| Length | MCC | Balanced Acc. | MCC | Balanced Acc. | MCC | Balanced Acc. |
| 1,000 | 0.30049 | 0.78765 | 0.25977 | 0.75103 | 0.31815 | 0.80484 |
| 2,000 | 0.40289 | 0.85981 | 0.27151 | 0.76234 | 0.49644 | 0.90637 |
| 3,000 | 0.40835 | 0.86523 | 0.34849 | 0.82861 | 0.41462 | 0.87112 |
| 4,000 | 0.34235 | 0.82404 | 0.34849 | 0.82861 | 0.49946 | 0.91088 |
| 5,000 | 0.34296 | 0.82450 | 0.34849 | 0.82861 | 0.50419 | 0.91271 |
| 6,000 | 0.34296 | 0.82450 | 0.34849 | 0.82861 | 0.50419 | 0.91271 |
| 7,000 | 0.34296 | 0.82450 | 0.34849 | 0.82861 | 0.50419 | 0.91271 |
| 8,000 | 0.34296 | 0.82450 | 0.34849 | 0.82861 | 0.50419 | 0.91271 |
| 9,000 | 0.34296 | 0.82450 | 0.39723 | 0.86107 | 0.50419 | 0.91271 |
| 10,000 | 0.33687 | 0.81861 | 0.34849 | 0.82861 | 0.50419 | 0.91271 |

これを踏まえて，機械学習による解凍処理識別手法により，情報エントロピーやハードウェアに依存せずに，展開されたコードを抽出するべきか識別する新たな手法を提案した．評価では，学習に用いたデータや，学習済みのパッカーに対してだけではなく，未学習のパッカーに対しても 80%以上の精度が得られることを示した．また，オペコードの情報に加え，オペランドの情報を合わせた独自の命令表現形式が精度の向上に繋がっていることを確認した．加えて，学習，推定に用いる命令シーケンスの長さと判定精度の関係を確認し，最適な命令シーケンスの長さを明らかにした．今後は，パッカー未適用のプログラムに対し，エントリーポイント以外から追跡を開始させることで多様な命令シーケンスの作成と不均衡の是正を目指す．また，今回は機械学習モデルとしてロジスティック回帰をベースとした機械学習を行ったが，深層学習ベースの機械学習モデルを用いた場合にどのように精度が変化するかを検討する．

## 参考文献


[1] LiŢă, C. V., Cosovan, D. and GavriluŢ, D.: Anti-emulation trends in modern packers: a survey on the evolution of anti-emulation techniques in UPA packers, *Journal of Computer Virology and Hacking Techniques*, Vol. 14, No. 2, pp. 107–126 (2018).

[2] Gibert, D., Totosis, N., Patsakis, C., Zizzo, G. and Le, Q.: Assessing the Impact of Packing on Machine Learning-Based Malware Detection and Classification Systems (2024).

[3] Mantovani, A., Aonzo, S., Ugarte-Pedrero, X., Merlo, A. and Balzarotti, D.: Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem, *NDSS 2020, Network and Distributed System Security Symposium, 23-26 February 2020, San Diego, CA, USA* (2020).

[4] 山根一真，掛井将平，齋藤彰一：パッカーの自動解凍に向けた軽量な細粒度フック手法の提案，SIG Technical Reports 2025-CSEC-109, 情報処理学会 (2025).

[5] Royal, P., Halpin, M., Dagon, D., Edmonds, R. and Lee, W.: PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware, *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pp. 289–300 (2006).

[6] Kang, M. G., Poosankam, P. and Yin, H.: Renovo: a hidden code extractor for packed executables, *Proceedings of the 2007 ACM Workshop on Recurring Malcode*, WORM '07, p. 46–53 (2007).

[7] Martignoni, L., Christodorescu, M. and Jha, S.: Omni-Unpack: Fast, Generic, and Safe Unpacking of Malware, *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 431–441 (2007).

[8] Cheng, B., Ming, J., Leal, E. A., Zhang, H., Fu, J., Peng, G. and Marion, J.-Y.: Obfuscation-Resilient Executable Payload Extraction From Packed Malware, *30th USENIX Security Symposium (USENIX Security 21)*, pp. 3451–3468 (2021).

[9] Cheng, B., Leal, E. A., Zhang, H. and Ming, J.: On the Feasibility of Malware Unpacking via Hardware-assisted Loop Profiling, *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 7481–7498 (2023).

[10] Kawakoya, Y., Akabane, S., Iwamura, M. and Okamoto, T.: Xunpack: Cross-Architecture Unpacking for Linux IoT Malware, *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '23, p. 471–484 (2023).

[11] Kazuma, Y.: InstructionSequenceDataset, https://github.com/Yamanekazuma/InstructionSequenceDataset. (Accessed 2025-08-21) (2025).

[12] D'Hondt, A.: Dataset of packed PE samples, https://github.com/packing-box/dataset-packed-pe. (Accessed 2025-08-20).

[13] Zhang, Z., Tao, G., Shen, G., An, S., Xu, Q., Liu, Y., Ye, Y., Wu, Y. and Zhang, X.: PELICAN: Exploiting Backdoors of Naturally Trained Deep Learning Models In Binary Code Analysis, *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 2365–2382 (2023).

[14] Lim, C., Syailendra Kotualubun, Y., Suryadi and Ramli, K.: Mal-Xtract: Hidden Code Extraction using Memory Analysis, *Journal of Physics: Conference Series*, Vol. 801, No. 1, p. 012058 (2017).

[15] snaker, Qwerton, Jibs and xineohP: PEiD detects most common packers, cryptors and compilers for PE files., https://github.com/wolfram77web/app-peid. (Accessed 2025-08-20) (2008).