ObfSwin: Transformer-based Multi-class Obfuscation Classification for Windows Portable Executable[*†]

Junhyuk Kang[1], Jaewoo Lee[2], Wonsuk Choi[3], and Dong Hoon Lee[4]

Korea University, Seoul, South Korea
{kangsecu, azan03160, beb0396, donghlee}@korea.ac.kr

Abstract

Windows remains the primary target of malware attacks, and attackers routinely deploy obfuscation—often in overlapping combinations—to evade detection and frustrate reverse engineering. When protections are layered, their effects compound: transformations interleave and mask one another, standard heuristics break, and efficient deobfuscation becomes difficult without first knowing the protection set. Yet efficient, execution-free methods that clearly identify the protections present in a binary remain scarce.

We present a purely static learning approach that infers applied protections directly from raw bytes. We model three-byte transitions as a 3D Markov tensor, convert it into a single RGB image representation, and feed that image to a Swin Transformer. We train combination size specific heads: one covering all pairwise two option settings and another covering all triple three option settings, using Windows PE binaries protected by a commercial tool. Evaluated on 100k obfuscated samples, consisting of six two option combinations and four three option combinations, our framework achieves 94% accuracy on the two option subset and 96% on the three option subset. The approach avoids execution and disassembly, substantially reduces memory and training cost compared with naïve 3D slice representations, and provides a practical basis for large scale static identification of overlapping protection options.

Keywords: Obfuscation, AI Security, Malware Analysis, Classification

1 Introduction

At the scale of modern enterprise and consumer deployments, Windows presents a broad, persistent attack surface. A heterogeneous installed base, strong backward compatibility, and expressive PE/loader semantics preserve stable primitives for persistence and evasion, sustaining malware activity[3]. For defenders, effective response hinges on reversing Windows PE binaries to recover behavior, unpack protected artifacts, derive signatures and rules, and guide remediation across fleets.

To frustrate such analysis, Windows malware routinely employs overlapping multi-option configurations that weaken static detectors and slow reverse engineering. As protections are layered, transformations interleave and mask one another, breaking standard heuristics; without first identifying the protection set, efficient deobfuscation is unlikely. Hence the urgent need for an automated, execution-free method to determine which protections each binary contains.

Despite steady progress in learning-based malware detection, much prior work ignores obfuscated samples or focuses on Android[5], and Windows-centric studies often stop at packer

[*]This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2024-00399389, Generative AI based Binary Deobfuscation Technology and Its Evaluation Metrics).

[†]Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. 86, December 16-18, 2025, Sapporo, Japan. © The copyright of this paper remains with the author(s).

identification or cover only techniques from academic or open-source tools[8]. In practice, commercial obfuscators are widely used and multi-option combinations are commonly applied at once, yet this setting remains underexplored. To the best of our knowledge, existing approaches largely infer single obfuscation options in isolation [7]. Single-option evaluations can overstate performance because interactions are non-additive; for example, packing reshapes byte statistics while changes to imports and resources perturb metadata. Identification should therefore be treated as a multi-class problem and tested across the full combination space.

Multi-option inference is inherently harder than single-option classification. When protections co-occur, their effects are non-additive: a dominant mechanism (e.g., pack-like transformations that reshape byte statistics) can mask weaker cues from imports or resources, narrowing margins between neighboring combinations that differ by only one option. Multi-option protections also induce feature entanglement across code and non-code regions, reducing linear separability unless interactions are modeled explicitly. The effective label space becomes denser as more co-occurrences are possible, increasing class proximity and fragmenting decision boundaries. Finally, valid combinations obey structural dependencies, so naïve per-option decisions can yield inconsistent sets unless interactions are modeled jointly. These considerations motivate a design that respects interactions while keeping decision spaces tractable.

To meet this need, we present a static framework that predicts the multi-option protections applied to VMProtect-protected Windows PE binaries[15]. Our method models a binary's three-byte transitions as a 3D Markov image, compresses it to a single image via principal component analysis (PCA)[10], and then employs a Swin Transformer[9]. We train *combination-size–specific* classifiers—one head over all pairwise two-option settings and another over all triple three-option settings—which aligns the label space with realistic multi-option usage and reduces confusion between distant classes. Evaluated on the pairwise and triple subsets, the framework attains high accuracy without requiring execution or disassembly.

Our contributions are threefold. First, we introduce a 3D Markov representation that maps raw-byte three-gram transition probabilities into a single RGB image while preserving distributional signals under stacked obfuscation. Second, we propose a static framework with a Swin backbone that performs *combination-size–aware* multi-class prediction by training dedicated heads—six classes for the two-option subset and four classes for the three-option subset. This is the first study to model and evaluate *layered*, multi-option configurations. Third, we empirically validate the approach on Windows PE binaries generated from BODMAS seeds[16] across these ten settings—six pairwise and four triple—showing strong accuracy with reduced memory and training cost thanks to PCA compression.

2 Preliminaries

This section summarizes the basic concepts and notation used in this paper. Section 2.1 introduces binary obfuscation and the VMProtect options; Section 2.2 formalizes the Markov transition matrix; Section 2.3 presents the Swin Transformer model; and Section 2.4 reviews principal component analysis.

2.1 Binary Obfuscation

Binary obfuscation is a technique developed to hinder software analysis and to protect intellectual property embedded in software—such as algorithms, secret keys, and other sensitive information—by deliberately transforming an executable's structure and semantics[4]. However, adversaries also abuse these techniques to impede reverse engineering of malware and to evade security solutions, thereby preventing analysts from understanding the code's logic[6].

In this study, we consider the following obfuscation options provided by the commercial tool VMProtect:

- **Pack the Output File (packing):** Reduces the size of the protected file by compressing/encrypting it and automatically unpacking it at runtime *in memory*; no plaintext code is written to disk during this process, which complicates static, signature-based detection and OEP recovery[15].
- **Import Protection:** Conceals the list of APIs used by the protected program from crackers by hiding/minimizing the Import Address Table (IAT) and resolving APIs dynamically (e.g., `LoadLibrary` /`GetProcAddress`) or via hash-based lookups[15].
- **Memory Protection:** Prevents modification of the in-memory image to strengthen security (data integrity is verified for all sections without the `WRITABLE` attribute). The integrity check is performed before transferring control to the original entry point (OEP); if the check fails, a message is displayed and execution is terminated[15].
- **Resource Protection:** Encrypts/obfuscates the PE's resource contents (e.g., icons, dialogs, manifests, version info, and embedded data) and may relocate them to custom sections. At runtime, a custom loader decrypts/decodes resources on demand (instead of leaving them in plaintext under `.rsrc`), which hinders static extraction, resource-only scanning, and straightforward diffing[15].

Option overlap. In current practice, protections are rarely applied in isolation[11]. They are layered, and the layering induces coupled changes across code, data, and metadata: binaries are reorganized, byte-level statistics shift, control-flow surfaces are rewritten, and externally visible interfaces shrink. These effects interact rather than add, which breaks hand-crafted rules and makes deobfuscation contingent on the exact protection set. Accordingly, identification should be posed as a multi-class problem over overlapping option sets and performed before any deobfuscation step.

We target malware protected by the above techniques and, in this study, set evaluation to the pairwise and triple settings: six two-option combinations and four three-option combinations. We focus on obfuscated samples and report results separately for the two-option and three-option subsets.

2.2 Markov Matrix

A *Markov matrix* (also called a *transition probability matrix*) is a square, row-stochastic matrix that encodes the one-step transition probabilities of a discrete-time stochastic process[13]. Let $\mathcal{S} = \{1, \ldots, n\}$ be a finite state space and let X_t denote the state at time t. In a time-homogeneous Markov chain, the next state depends only on the current state:

$$\Pr\big(X_{t+1} = j \,\big|\, X_t = i_t,\, X_{t-1} = i_{t-1}, \ldots, X_0 = i_0\big) = \Pr\big(X_{t+1} = j \,\big|\, X_t = i_t\big). \tag{1}$$

Collecting the one-step transition probabilities into a matrix $P = [p_{ij}] \in \mathbb{R}^{n \times n}$ gives

$$p_{ij} = \Pr(X_{t+1} = j \mid X_t = i), \qquad P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}. \tag{2}$$

Each entry is nonnegative and each row sums to one:

$$p_{ij} \geq 0, \qquad \sum_{j=1}^{n} p_{ij} = 1, \quad \forall i \in \{1, \ldots, n\}, \tag{3}$$

so every row of P is a probability vector. This representation enables us to quantify the probability of moving from any current state to any subsequent state; in our application, we exploit this property to encode binaries as transition-probability matrices.

2.3 Swin Transformer Model

Swin—short for Shifted Window Transformer—is a hierarchical vision Transformer that partitions the input into fixed-size windows and applies self-attention only within each window, a mechanism called windowed multi-head self-attention (W-MSA)[9]. In the next block, the windows are shifted by a small offset; this shifted-window multi-head self-attention (SW-MSA) lets tokens interact with neighbors that fell in adjacent windows in the previous block. The design avoids the quadratic cost of global attention while preserving locality from windowing and adding cross-window context via shifting, achieving a balanced efficiency–expressiveness trade-off even for high-resolution inputs. In addition, patch merging progressively halves the spatial resolution and increases the channel width to form a feature pyramid at one-quarter, one-eighth, one-sixteenth, and one-thirty-second scales, which suits dense prediction tasks.

Notation. Let H and W denote the image height and width, respectively; P the patch size; C the embedding/channel dimension; M the window side length; $s = \lfloor M/2 \rfloor$ the shift size; $G = (H'/M) \cdot (W'/M)$ the number of windows at a given stage with spatial size $H' \times W'$.

Given an RGB image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$, we split it into non-overlapping $P \times P$ patches and apply a linear embedding to obtain:

$$N = \frac{H}{P} \cdot \frac{W}{P}, \qquad \mathbf{X} \in \mathbb{R}^{N \times C}, \tag{4}$$

where N is the number of patches and C is the embedding dimension.

At resolution $H' \times W'$, after partitioning into $M \times M$ windows, W-MSA for window g with tokens $\mathbf{X}^{(g)} \in \mathbb{R}^{M^2 \times C}$ (single-head notation) is

$$\mathrm{Attn}\Big(\mathbf{X}^{(g)}\Big) = \mathrm{Softmax}\Big(\tfrac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{B}\Big)\mathbf{V}, \quad \mathbf{Q} = \mathbf{X}^{(g)}\mathbf{W}^Q, \;\; \mathbf{K} = \mathbf{X}^{(g)}\mathbf{W}^K, \;\; \mathbf{V} = \mathbf{X}^{(g)}\mathbf{W}^V, \tag{5}$$

where d_k is the key/query dimension and $\mathbf{B} \in \mathbb{R}^{M^2 \times M^2}$ is the relative position-bias matrix.

In the next block, we circularly shift by $s = \lfloor M/2 \rfloor$ and re-partition into $M \times M$ windows. An attention mask $\mathcal{M} \in \mathbb{R}^{M^2 \times M^2}$ is applied to maintain window-local attention under the batched computation after the shift, yielding the shifted-window attention (SW-MSA):

$$\mathrm{Attn}_{\mathrm{SW}} = \mathrm{Softmax}\Big(\tfrac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{B} + \mathcal{M}\Big)\mathbf{V}. \tag{6}$$

In terms of complexity, the number of windows is $G = (H'/M) \cdot (W'/M)$ and the per-window attention cost is $\Theta(M^4)$, so the total cost is

$$\Theta\Big(\tfrac{H'W'}{M^2} \cdot M^4\Big) = \Theta(H'W' \cdot M^2), \tag{7}$$

which scales nearly linearly with spatial size when M is fixed.

Between stages, patch merging halves the resolution and doubles the channels:

$$\mathbb{R}^{H' \times W' \times C} \xrightarrow{\text{merge}} \mathbb{R}^{\frac{H'}{2} \times \frac{W'}{2} \times 2C}. \tag{8}$$

In summary, Swin combines (i) windowed attention with cross-window interactions via shifting, (ii) stable positional encoding through relative position bias, and (iii) multi-scale representations via patch merging, yielding an efficient accuracy–cost trade-off on high-resolution inputs. For our Markov-based images—where transition patterns are local yet distributed—window attention captures local transitions while shifted windows propagate context across boundaries, providing advantages over CNN backbones.

2.4 Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality-reduction technique that maps high-dimensional data to a lower-dimensional space while preserving as much variance as possible[10]. Given a data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ consisting of n samples and d features, we first center each feature by subtracting its mean:

$$\mathbf{X}_{\text{centered}} = \mathbf{X} - \mathbf{1}_n \boldsymbol{\mu}^\top, \qquad \boldsymbol{\mu} = \tfrac{1}{n}\,\mathbf{X}^\top \mathbf{1}_n. \tag{9}$$

We then compute the sample covariance matrix

$$\boldsymbol{\Sigma} = \frac{1}{n-1}\,\mathbf{X}_{\text{centered}}^\top \mathbf{X}_{\text{centered}}. \tag{10}$$

Next, we perform eigenvalue decomposition of $\boldsymbol{\Sigma}$ to find the principal directions:

$$\boldsymbol{\Sigma}\,\boldsymbol{v}_i = \lambda_i\,\boldsymbol{v}_i, \quad i = 1, \ldots, d, \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d. \tag{11}$$

To reduce the dimensionality to k, we select the top-k eigenvectors and form the projection matrix

$$\mathbf{V}_k = \big[\boldsymbol{v}_1,\, \boldsymbol{v}_2,\, \ldots,\, \boldsymbol{v}_k\big] \in \mathbb{R}^{d \times k}. \tag{12}$$

Finally, we project the centered data onto the new k-dimensional subspace:

$$\mathbf{X}_{\text{reduced}} = \mathbf{X}_{\text{centered}} \mathbf{V}_k. \tag{13}$$

This procedure preserves the dominant variance in the original data while reducing dimensionality and improving computational efficiency.

3 Related Work

Deobfuscation depends strongly on the techniques that were applied, so reliably identifying those techniques is a prerequisite. Consequently, academic research on identifying obfuscation is active.

Dhanya K. A.[5] studied Android-based IoT applications and, using Obfuscapk[2], classified twelve obfuscation techniques applied to APKs. However, their setting is not Windows, and Android obfuscation operates mainly at the source or bytecode level, such as string transformations and control-flow obfuscation. Therefore, these approaches are not adequate for the

Windows PE setting, where commercial binary obfuscation acts at the binary and section levels via mechanisms such as import protection, packing, and related techniques.

Li et al.[8] proposed PackGenome to automate packer detection by extracting packer-specific genes and auto-generating YARA rules[1], achieving effective identification across twenty tools. However, this work focuses on packers and does not cover the broader spectrum of binary obfuscation, including memory and resource protection. It also does not evaluate the concurrent application of multi-option techniques that is common in practice, such as packing combined with import protection.

Parker et al.[14] converted binaries into grayscale images and used a small CNN to classify obfuscation techniques. However, the dataset is limited to Tigress/OLLVM-based source/IR-level transformations, which leaves a substantial gap to commercial binary obfuscation for Windows PE; core techniques used by commercial tools—such as packing and import protection—are not evaluated. At the representation level, mapping bytes directly to pixels and resizing to a fixed size causes information loss, and as binaries grow larger the images exhibit nonlinear scaling issues.

Kang et al.[7] converted VMProtect-obfuscated malware binaries into 3D Markov images, applied PCA for dimensionality reduction, and fed the result to a CNN to classify obfuscation techniques. While effective, comparatively large 3D Markov images leave room for improvement in modeling long-range dependencies and parameter efficiency with conventional CNNs. In addition, the study focuses on identifying single techniques, making it difficult to account for the frequent concurrent use of multi-option techniques in real-world settings.

To address these limitations, we statically analyze Windows PE binaries—without execution and without relying on high-level metadata—and construct a 3D Markov image from raw byte transition probabilities. This representation spans both code and non-code regions, reducing information loss and remaining robust to diverse obfuscation mechanisms. We adopt a Swin Transformer with shifted windows to integrate local and global information even for large inputs, and we train *combination-size–specific* classifiers. In particular, we evaluate on the pairwise two-option and triple three-option subsets and train dedicated heads for each subset, which aligns the label space with realistic multi-option combinations usage and reduces inter-class confusion. On a malware dataset obfuscated with the commercial tool VMProtect, our empirical evaluation shows that the proposed framework efficiently identifies techniques under overlapping obfuscation while maintaining strong accuracy and efficiency.

4 Framework

This section describes the framework for classifying obfuscation techniques proposed in the paper.

4.1 Overview

Windows-targeting malware is typically distributed as PE executables and often layers multi-option anti-analysis techniques, which makes it difficult for analysts to infer intent. Dynamic analysis carries infection risk and is further impeded by anti-sandbox and anti-virtualization features, so a static method that does not execute the target is required[17]. Code-centric features obtained from disassembly (e.g., opcodes and operands) suffer from information loss and bias because commercial obfuscation also modifies non-code sections such as the import address table and the resource section; anti-disassembly techniques further hinder analysis[12].

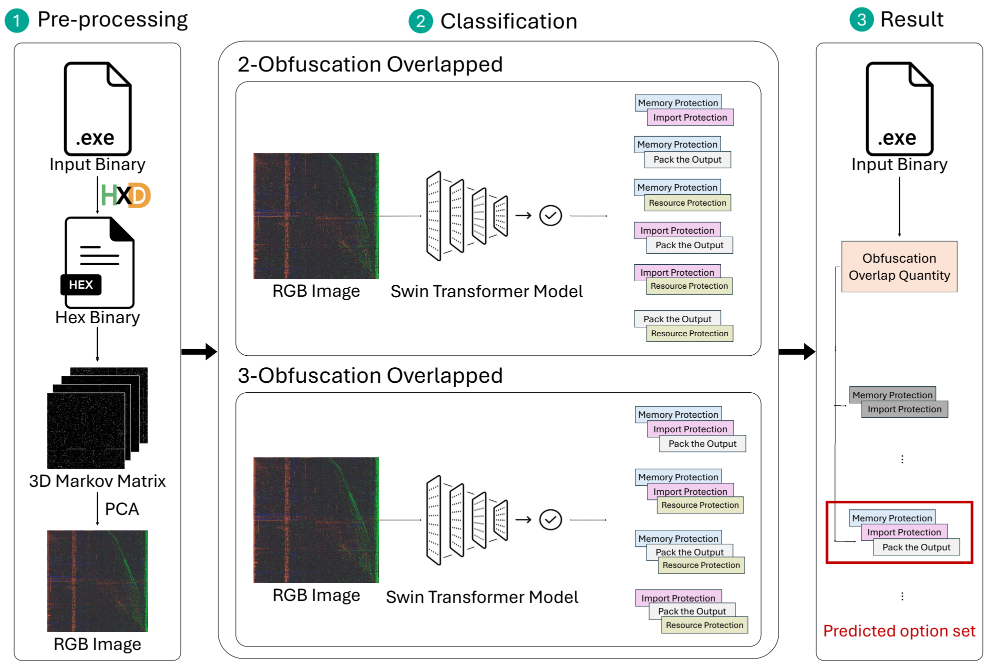


Figure 1: Framework Overview

We therefore avoid high-level metadata and instead encode each binary as a 3D Markov image constructed solely from raw-byte transition probabilities. This representation spans both code and non-code regions and captures byte-transition statistics without relying on section boundaries or absolute locations. Building on this idea, as illustrated in Figure 1, we propose a static, fully automated framework that converts the 3D Markov image into a fixed-size RGB image via PCA and feeds it to a Swin Transformer. we train *combination-size–specific* heads and evaluate on the pairwise two-option and triple three-option subsets. This design aligns the label space with realistic multi-option usage and reduces inter-class confusion, yielding safe analysis without execution and robust accuracy and speed on realistic malware datasets.

4.2 Pre-Processing

Preprocessing converts malware binaries into inputs suitable for neural-network classification. Feeding raw bytes directly to the model is inefficient, so we transform each binary into a fixed-size image. A malware binary is a sequence of bytes; we read the byte stream directly and compute transition probabilities over consecutive three-byte sequences to form a 3D Markov tensor. We then apply PCA to obtain a single RGB image, which serves as the input to the Swin Transformer. The procedure is summarized below.

4.2.1 3D Markov Matrix

To represent a binary as a three-dimensional Markov tensor, we partition its byte stream into overlapping triples and record the frequency of each transition. Let the byte sequence be

$H = \{h_1, h_2, \ldots, h_N\}$ with $h_t \in \{0, \ldots, 255\}$. We accumulate counts into $M \in \mathbb{R}^{256\times256\times256}$ as

$$M[i,j,k] \;=\; \sum_{t=1}^{N-2} \delta\big(h_t = i\big)\, \delta\big(h_{t+1} = j\big)\, \delta\big(h_{t+2} = k\big), \tag{14}$$

where $i, j, k \in \{0, \ldots, 255\}$ index byte values and $\delta(\cdot)$ is the indicator function that returns 1 if the condition holds and 0 otherwise. The tensor M thus stores frequencies of all byte 3-gram transitions. In the next step, we normalize M to transition probabilities and apply PCA to obtain a single 256×256×3 RGB image.

4.2.2 Normalization

The constructed three-dimensional Markov tensor M contains only transition counts, so it must be normalized to probabilities. For each start state $i \in \{0, \ldots, 255\}$, we define the conditional probability of the two-step transition $i \to j \to k$ as

$$P(i \to j \to k) \;=\; \frac{M[i,j,k]}{\displaystyle\sum_{j',\,k'} M[i,j',k']}, \qquad j, k \in \{0, \ldots, 255\}. \tag{15}$$

Here, $P(i \to j \to k)$ denotes the probability of transitioning from state i to j and then from j to k. The numerator is the observed count $M[i,j,k]$, and the denominator is the sum of counts over all (j', k') transitions that start with i. This mapping sends counts to $[0,1]$ and ensures

$$\sum_j \sum_k P(i \to j \to k) = 1 \quad \forall\, i \text{ with } \sum_{j',k'} M[i,j',k'] > 0.$$

Numerical stability. If $\sum_{j',k'} M[i,j',k'] = 0$ for some i (i.e., i never appears as a start byte), we either (i) set $P(i \to j \to k) = 0$ for all (j,k) and exclude that i during normalization, or (ii) apply add-ε smoothing:

$$P_\varepsilon(i \to j \to k) = \frac{M[i,j,k] + \varepsilon}{\sum_{j',k'} \big(M[i,j',k'] + \varepsilon\big)},$$

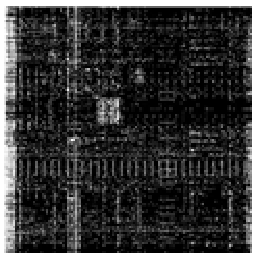
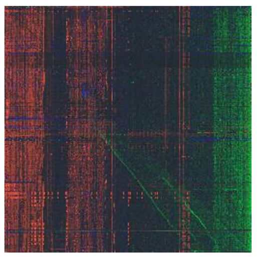
	2D Markov	**3D Markov**	**3D Markov-PCA**
Image			
Image size	256×256	256×256×256	256×256×3
Count	160,000	40,960,000	160,000
Memory usage	∼0.8 GB	∼120 GB	∼0.5 GB
Training time	∼ 1–2 hours	∼10 days	∼ 1–2 hours

Table 1: Comparison of image representations.

Module (stage & type)	**Input shape**	**Output shape**	**Notes**
Patch embedding	(256,256,3)	(64,64,96)	patch 4×4, embed dim 96
Stage 1 — Swin block(s)	(64,64,96)	(64,64,96)	window 7×7
Stage 2 — Patch merging	(64,64,96)	(32,32,192)	2×2 merge, channels ×2
Stage 2 — Swin block(s)	(32,32,192)	(32,32,192)	window 7×7
Stage 3 — Patch merging	(32,32,192)	(16,16,384)	2×2 merge, channels ×2
Stage 3 — Swin block(s)	(16,16,384)	(16,16,384)	window 7×7
Stage 4 — Patch merging	(16,16,384)	(8,8,768)	2×2 merge, channels ×2
Stage 4 — Swin block(s)	(8,8,768)	(8,8,768)	window 7×7
Classification head — GAP + Linear	(8,8,768)	(C_{sub})	$C_{\text{sub}} \in \{6, 4\}$

Table 2: Swin Transformer architecture for Markov-image classification.

with a small ε (e.g., 10^{-8}) to avoid division by zero and improve numerical stability. We follow the Markov normalization in Section 2.2 and use the resulting probability tensor for the subsequent PCA step (Section 2.4).

4.2.3 PCA for Image

A naïve use of the 256×256×256 3D Markov matrix renders each sample as 256 separate 256×256 slices—one per index along the third axis—which bloats memory usage and lengthens training time. Instead of slicing, we apply principal component analysis to the 3D Markov matrix, retain the top three components, and stack them as a single RGB image of size 256×256×3. This conversion preserves most of the informative variance and substantially lowers I/O, memory footprint, and wall-clock training time with our Swin-based classifier. Table 1 summarizes the efficiency gains.

4.3 Model Implementation

We split the task by combination size and train two classifiers that share the same Swin backbone but use separate heads. Let $\mathcal{O} = \{\text{memory protection}, \text{ import protection}, \text{ resource protection}, \text{pack the output file}\}$. For the two-option subset we define

$$\mathcal{Y}_2 = \{\, o_a \cup o_b \mid o_a, o_b \in \mathcal{O},\ a < b \,\}, \quad |\mathcal{Y}_2| = 6,$$

and for the three-option subset,

$$\mathcal{Y}_3 = \{\, o_a \cup o_b \cup o_c \mid o_a, o_b, o_c \in \mathcal{O},\ a < b < c \,\}, \quad |\mathcal{Y}_3| = 4.$$

Given a 256×256×3 Markov–PCA image, the backbone produces a feature map that is globally averaged and fed to a linear head, yielding logits $\mathbf{z} \in \mathbb{R}^{C_{\text{sub}}}$ with $C_{\text{sub}} \in \{6, 4\}$. We train with cross-entropy and predict

$$\hat{y} = \arg \max_{c \in \{1,\ldots,C_{\text{sub}}\}} \text{softmax}(\mathbf{z})_c.$$

Layer-wise shapes and window settings are summarized in Table 2.

Why Swin instead of CNN. Our 256×256×3 Markov images contain fine-grained local byte-transition textures as well as broader contextual signals. Swin captures both: windowed self-attention learns local patterns efficiently, while shifted windows propagate information

Subset	**Combination**	# samples
Two options	Import Protection + Memory Protection	10k
	Import Protection + Resource Protection	10k
	Import Protection + Pack the Output File	10k
	Memory Protection + Resource Protection	10k
	Memory Protection + Pack the Output File	10k
	Resource Protection + Pack the Output File	10k
	Subtotal	60k
Three options	Import Protection + Memory Protection + Resource Protection	10k
	Import Protection + Memory Protection + Pack the Output File	10k
	Import Protection + Resource Protection + Pack the Output File	10k
	Memory Protection + Resource Protection + Pack the Output File	10k
	Subtotal	40k
Total		**100k**

Table 3: Dataset composition.

across neighboring windows to recover longer-range dependencies. In parallel, patch merging builds a hierarchical pyramid ($64 \rightarrow 32 \rightarrow 16 \rightarrow 8$) with expanding channels, forming multi-scale features while keeping attention cost near-linear in image size for a fixed window. This makes Swin a better fit than CNNs—which rely on fixed kernels and depth to reach comparable receptive fields—for learning and classifying overlapping obfuscation-option patterns from Markov images.

5 Evaluation

This section presents our experimental methodology and evaluation. We describe the datasets, the hardware and software environment, the model configuration and training protocol, and the evaluation metrics; we then report results, and error analysis.

5.1 Experimental Setup

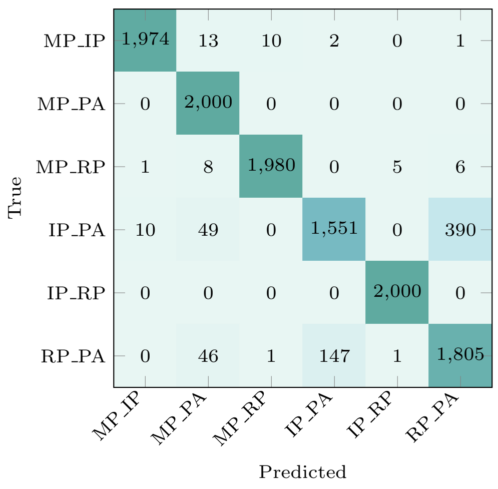
Preprocessing was executed on a local machine with an Intel Core i7-12700 CPU. Swin Transformer training and test-time classification were run on a server equipped with an NVIDIA RTX A6000 GPU. Unless noted otherwise, data preparation was CPU-only, and GPU acceleration was used solely for model training and inference.

5.2 Dataset

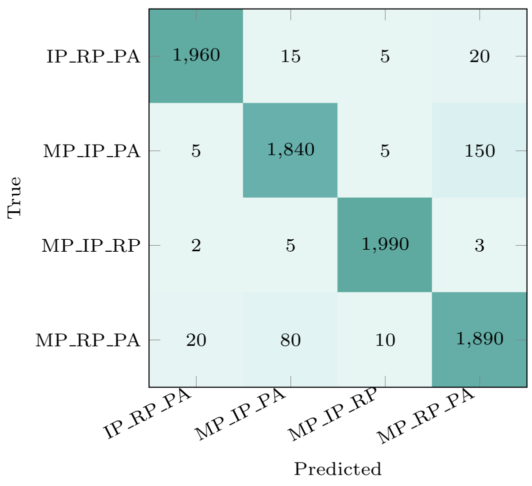
This paper uses a widely deployed commercial protector to generate labeled obfuscation combinations. To keep the corpus controlled and reproducible, we fix one commonly used tool. In the Windows ecosystem, VMProtect is among the most commonly used protectors in practice, so it is a realistic choice; at the same time, it exposes the four options at the binary/section level and

Metric	**Two-option**				**Three-option**			
	Acc.	**Prec.**	**Recall**	**F1-score**	**Acc.**	**Prec.**	**Recall**	**F1-score**
Train	0.95	0.95	0.95	0.95	0.97	0.97	0.97	0.97
Test	0.94	0.94	0.94	0.94	0.96	0.96	0.96	0.96

Table 4: Summary metrics by subset (Time removed for clarity).



(a) Two-option (acc.=0.94)



(b) Three-option (acc.=0.96)

Figure 2: Classification confusion-matrix heatmaps.

Abbreviations. MP: Memory Protection; IP: Import Protection; RP: Resource Protection; PA: Pack the Output File. Labels such as `MP_IP` and `IP_RP_PA` indicate two- and three-option combinations, respectively.

supports deterministic application in combination, which lets us produce ground-truth labels for all non-empty option sets at scale. Our approach itself is protector-agnostic and requires no change to the representation or model. The four target options are: *import protection*, *memory protection*, *resource protection*, and *pack the output file*.

As Windows PE malware seeds, we randomly sampled 10,000 executables from the BODMAS dataset. In this study we set evaluation to the pairwise and triple option settings, which are both common and challenging in practice. Specifically, we generated 10,000 variants for each of the six pairwise combinations ($\binom{4}{2} = 6$) and each of the four triple combinations ($\binom{4}{3} = 4$), yielding $6 + 4 = 10$ distinct settings and a total of 100,000 obfuscated samples.

We performed a stratified split by combination into 70% training (70,000) and 30% test (30,000). A dataset summary appears in Table 3.

5.3 Classification Results

Table 4 summarizes the evaluation of the proposed 3D Markov–PCA + Swin framework on the test set of 30,000 executables (18,000 two-option and 12,000 three-option samples). With combination-size–specific heads, we achieve 94% accuracy on the two-option subset and 96%

on the three-option subset. Accuracy scores are comparable across subsets, and the entire pipeline operates statically—without execution—making it suitable for large-scale evaluation. The training time for the two-option subset is approximately 4,086 seconds, and 3,482 seconds for the three-option subset. During testing, each sample is processed in roughly 8.12 seconds for the two-option subset and 7.24 seconds for the three-option subset. Obfuscation characteristics that alter code, resources, and metadata are reflected in the byte-transition patterns of the 3D Markov image, while Swin's windowed and shifted self-attention captures both local textures and broader contextual signals, enabling effective discrimination of multi-option obfuscations.

As shown in Figure 2, the confusion matrix indicates that errors predominantly occur between adjacent combinations that differ by a single option. In the two-option subset, models most frequently confuse combinations that both include Pack the Output File (e.g., *Import Protection + Pack the Output File* vs. *Memory Protection + Pack the Output File*), as packing strongly reshapes byte distributions and induces similar Markov patterns. In the three-option subset, confusion is higher among the combinations that share Pack the Output File (e.g., *Import Protection + Memory Protection + Pack the Output File* vs. *Memory Protection + Resource Protection + Pack the Output File*); the distinctive signal of the differing option can be partially masked by the shared effects of packing and the other common option.

In summary, training by combination size reduces confusion between distant classes and aligns the label space with realistic multi-option usage, yielding stable performance—94% for two-option and 96% for three-option classification—and establishing a practical basis for large-scale static analysis in commercial protector settings, with efficient training and inference times as noted above.

6 Future Work

Future work will focus on realizing a single model that, given a binary once, directly predicts the exact protection combination. A preliminary single-shot classifier over the full combination space reached only about 67% accuracy, indicating that learning all option interactions at once is harder than expected. To address this, we plan to apply curriculum learning that starts from our working two-option and three-option heads and gradually expands the label space. In parallel, we will explore a hierarchical strategy that first predicts the number of options and then the specific combination, reducing routing errors. Our goal is to lift the current 67% single-model accuracy substantially while preserving the practical advantages of our static pipeline and its low memory and time costs.

7 Conclusion

This paper presents a multi-class classification framework that statically identifies obfuscation options in Windows PE malware protected by commercial tools. The method combines a 3D Markov image constructed from raw-byte transition probabilities with a Swin Transformer. Using 10,000 seeds from the BODMAS dataset, we evaluate on the pairwise and triple subsets of the four VMProtect options—six two-option combinations and four three-option combinations—yielding a corpus of 100,000 obfuscated samples. We train separate heads for the two-option and three-option settings while sharing the same backbone, and we observe strong accuracy on both subsets: 94% on the two-option subset and 96% on the three-option subset.

The framework is purely static and does not rely on high-level metadata or disassembly, which improves analysis safety and deployability. In preprocessing, we build the 3D Markov matrix and apply a PCA-based three-channel reduction to form a single RGB image per binary, substantially lowering memory and I/O while preserving informative transition statistics. Leveraging window and shifted-window attention in the Swin Transformer, the model captures fine-grained local patterns and reliably detects overlapping obfuscation options within each combination-size subset. These results establish a practical foundation for large-scale static classification in commercial protector settings and demonstrate efficiency and accuracy suitable for integration into operational malware-analysis workflows.

References

[1] V. M. Alvarez. "YARA —The patternmatching Swiss knife for malwareresearchers," VirusTotal. [Online]. `Available:https://virustotal.github.io/yara/`. (accessed: Dec. 9, 2022).

[2] Simone Aonzo, Gabriel Claudiu Georgiu, Luca Verderame, and Alessio Merlo. Obfuscapk: An open-source black-box obfuscation tool for Android apps. *SoftwareX*, 11:100403, 2020.

[3] AV-TEST. "Total malware," Malware Statistics & Trends Report, [Online]. `Available:https://www.av-test.org/en/statistics/malware/`. (accessed: Oct.20, 2024).

[4] Asish Kumar Dalai, Shakya Sundar Das, and Sanjay Kumar Jena. A code obfuscation technique to prevent reverse engineering. In *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 828–832. IEEE, 2017.

[5] KA Dhanya, P Vinod, Y Yerima Suleiman, T Abhiram, K Shavanas Ashil, T Gireesh Kumar, et al. Obfuscated malware detection in IoT Android applications using Markov images and CNN. *IEEE Systems Journal*, 17(2):2756–2766, 2023.

[6] Claudia Greco, Michele Ianni, Antonella Guzzo, and Giancarlo Fortino. Explaining binary obfuscation. In *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 22–27. IEEE, 2023.

[7] Junhyuk Kang, Jiwon Lee, Hongjoo Jin, Dong Hoon Lee, and Wonsuk Choi. Classification of Binary Obfuscation Techniques in Windows Binaries Using a 3D Markov Matrix. *Journal of the Korea Institute of Information Security & Cryptology*, 35(3):563–572, 2025.

[8] Shijia Li, Jiang Ming, Pengda Qiu, Qiyuan Chen, Lanqing Liu, Huaifeng Bao, Qiang Wang, and Chunfu Jia. PackGenome: Automatically generating robust YARA rules for accurate malware packer detection. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 3078–3092, 2023.

[9] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[10] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (PCA). *Computers & Geosciences*, 19(3):303–342, 1993.

[11] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security*, 51:16–31, 2015.

[12] Kenneth Miller, Yonghwi Kwon, Yi Sun, Zhuo Zhang, Xiangyu Zhang, and Zhiqiang Lin. Probabilistic disassembly. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1187–1198. IEEE, 2019.

[13] J.R. Norris. Markov chains. Cambridge University Press, pp. 1-200, Dec. 1997.

[14] Colby Parker, Jeffrey Todd McDonald, and Dimitrios Damopoulos. Machine Learning Classification of Obfuscation using Image Visualization. In *SECRYPT*, pages 854–859, 2021.

[15] VMProtect Software. "User manual," VMProtect, [Online]. `Available:https://vmpsoft.com/vmprotect/user-manual`. (accessed: Oct.20, 2024).

[16] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. Bodmas: An open dataset for learning based temporal analysis of pe malware. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 78–84. IEEE, 2021.

[17] Bin Zeng. Static analysis on binary code. *Tech. rep. Tech-report*, page 17, 2012.