TU WIEN Informatics

Encoding Strategies and Explainability in Transformer-Based Malware Family Classification

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Luzia Klauser
Matrikelnummer 12243755

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Mitwirkung: Univ.Lektor Dipl.-Ing. Mag.rer.soc.oec. Rudolf Mayer

Wien, 30. August 2026

Klauser | Andreas Rauber

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Encoding Strategies and Explainability in Transformer-Based Malware Family Classification

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Business Informatics

by

Luzia Klauser
Registration Number 12243755

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Assistance: Univ.Lektor Dipl.-Ing. Mag.rer.soc.oec. Rudolf Mayer

Vienna, August 30, 2026

Klauser | Andreas Rauber

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Luzia Klauser

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 30. August 2026

Klauser

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to thank my supervisors, Andreas Rauber and Rudolf Mayer, for their guidance throughout this thesis, for the many helpful discussions, and for their detailed feedback on my work.

I am also very grateful to my parents for their continuous support throughout my studies. Finally, I would like to thank my partner, David, for always supporting me, especially during the more challenging parts of this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Malware family classification is an essential task in cybersecurity, as it supports malware analysis, reverse engineering, and incident response by identifying previously unseen malware variants. Visualization-based approaches have enabled the application of deep learning by representing malware executables as images, with Convolutional Neural Networks (CNNs) becoming the dominant approach. However, Vision Transformers (ViTs) have recently emerged as a promising alternative due to their ability to model long-range dependencies, and can thus replace CNNs also for malware analysis tasks. Nevertheless, the influence of different malware-to-image encoding strategies on the performance of Vision Transformer models for malware classification, the explainability of these models, and the interpretation of their predictions remain insufficiently understood.

This thesis addressed these gaps and presents a systematic study of Vision Transformer-based malware family classification for Windows Portable Executable (PE) binaries. First, we systematically compare representative malware-to-image encoding strategies in terms of classification effectiveness and computational efficiency, thereby providing a comprehensive evaluation of their suitability for Vision Transformer-based malware classification. Second, we evaluate multiple Vision Transformer explainability methods using quantitative explainability metrics and feature-space visualizations to assess their faithfulness, sparsity, consistency, stability, and computational efficiency. Third, we propose a reverse-traceability framework that maps influential image regions back to executable artifacts. The framework is evaluated through quantitative traceability analyses, perturbation experiments, and reverse-engineering case studies, demonstrating its ability to support the interpretation of Vision Transformer-based malware family classifications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Abstract **ix**

Contents **xi**

1 Introduction **1**
1.1 Problem Statement . . . 1
1.2 Research Questions . . . 2
1.3 Thesis Structure . . . 4

2 Background **5**
2.1 Malware Analysis . . . 5
2.2 Machine learning for malware analysis . . . 7
2.3 Static Malware Analysis and Feature Representations . . . 9
2.4 Deep Learning for Malware Classification . . . 13
2.5 Explainable Artificial Intelligence . . . 22

3 State of the Art **27**
3.1 Malware-to-Image Encoding Strategies in Prior Work . . . 27
3.2 Vision Transformer–Based Malware Classification Approaches . . . 37
3.3 Vision Transformer Explainability Methods and Metrics . . . 43

4 Methodology and Experimental Design **53**
4.1 Dataset and Data Preparation . . . 53
4.2 Malware-to-Image Encoding Strategies . . . 61
4.3 Model Architecture . . . 65
4.4 Training Configuration . . . 67
4.5 Explainability Methods . . . 68
4.6 Reverse Traceability Framework . . . 69
4.7 Evaluation Design . . . 71

5 Results **79**
5.1 Classification Performance . . . 79
5.2 Explainability Evaluation . . . 90
5.3 Reverse Traceability Analysis . . . 97

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.4 Reverse Engineering Case Studies 105

6 Discussion **117**
6.1 Insights and Contributions 117
6.2 Research Questions 118
6.3 Future Work 122

Overview of Generative AI Tools Used **123**

List of Figures **125**

List of Tables **127**

Bibliography **131**

Appendix **143**
Hybrid Metadata Storage 143
Classification Robustness 145
Reverse Engineering Evidence of Malicious Behavior 147

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

1.1 Problem Statement

The rapid evolution of malware poses a growing challenge for cybersecurity. Malware authors employ obfuscation, packing, and polymorphism to evade detection, producing a vast number of variants within malware families and making accurate classification increasingly difficult. In 2024, the BSI, the German Federal Office for Information Security, reported an average of 250,000 new malware variants emerging worldwide each day, alongside a 256% rise in Windows malware variants [Bun24]. This makes timely detection and family-level classification critical for analysts, particularly when dealing with previously unseen zero-day malware, as many such samples are variants of existing families [UAB19]. This is particularly important because research indicates that most new malware is a variant of existing families [NKM15]. Traditional static and dynamic malware analysis remain effective, but face challenges related to scalability, computational cost, and vulnerability to code obfuscation [GMP20, GBS14].

In recent years, deep learning techniques have reduced reliance on manual feature engineering in malware classification and detection. In particular, visualization-based malware classification, in which malware is encoded into image representations, has enabled the application of computer vision models, leading to the widespread adoption of Convolutional Neural Networks (CNNs) in malware analysis [GMP20]. Compared with traditional machine-learning approaches, CNN-based methods have achieved substantial improvements in classification performance [KRM+18]. However, due to their local receptive fields, CNNs may struggle to capture long-range dependencies in malware representations [RCS23].

Vision Transformers (ViTs) represent a promising alternative to CNNs, as their attention mechanisms enable modeling of long-range dependencies, which is particularly relevant for malware that exhibits local and global code patterns [KNH+22]. However, different image

representations preserve different characteristics of malware and may therefore influence how effectively Vision Transformers can learn discriminative representations. While recent work has begun to investigate transformer-oriented malware-to-image encoding strategies, including semantic lossless mappings [YCA+25], existing encoding strategies have not yet been systematically compared for Vision Transformer-based malware family classification.

Beyond classification performance, the attention mechanisms of Vision Transformers offer new opportunities to explain model predictions. Explainability is particularly important in security contexts, as malware family classifications guide subsequent malware analysis, reverse engineering, and incident response [YWLA+21]. Analysts must therefore be able to understand and validate the basis of a model's classification decisions. While deep learning has advanced malware classification, the increased complexity of these models has also resulted in a stronger black-box character compared to traditional approaches. Explainability is therefore crucial for building analysts' trust, yet evaluating the quality of explanations remains an open challenge [SMFC24]. Existing studies primarily rely on qualitative visual inspection, such as attention or activation map visualizations [PC22, LZKN22, JLKC25], while systematic quantitative comparisons of Vision Transformer explainability methods for malware classification remain limited.

Furthermore, explainability methods typically produce relevance maps that indicate which image regions influenced a classification decision, but do not reveal which executable artifacts those regions represent. Consequently, analysts cannot directly relate image-based explanations to the executable artifacts used during malware analysis and reverse engineering [VRM+19, YWLA+21]. Although prior work has demonstrated the potential value of linking explanations to executable artifacts [YSN+19, ICM+21], no systematic reverse-traceability approach has been established for Vision Transformer-based malware family classification of Windows PE binaries.

1.2 Research Questions

To address these research gaps, the following research questions will be explored:

1. **To what extent do different malware-to-image encoding strategies affect the performance of a Vision Transformer model in malware family classification?** Current Vision Transformer-based malware classification studies typically evaluate only a single encoding strategy. To investigate the influence of representation design under comparable conditions, multiple malware-to-image encoding strategies are evaluated using a common Vision Transformer architecture. Their performance is assessed from the complementary perspectives of classification effectiveness and computational efficiency.

    a) **To what extent does the choice of encoding influence effectiveness measured by metrics such as accuracy, F1-score, and false positive**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

rate? The Vision Transformer model is trained on multiple malware-to-image encoding schemes and evaluated using standard classification metrics. Since different encoding strategies preserve different characteristics of malware, this analysis investigates how these differences influence classification effectiveness and identifies which representations provide the best balance between accuracy, F1-score, and false positive rate.

b) **To what extent does the choice of encoding impact efficiency?** Since different encoding strategies preserve different characteristics of malware, they may also differ in their training dynamics and computational requirements. We investigate which encoding strategies achieve the best balance between classification performance and computational efficiency.

2. **To what extent can explainability methods for Vision Transformers (ViT) provide faithful, sparse, consistent, stable, and computationally efficient explanations of malware family classification decisions?** This question addresses the lack of systematic evaluations of explainability methods for Vision Transformers for malware classification, by comparing different explainability methods with respect to the quality and computational efficiency of the explanations they produce. In addition, visualization techniques are used to provide qualitative insight into the model's learned feature representations.

   a) **To what extent do different explainability methods for Vision Transformers provide faithful and sparse explanations?** The analysis focuses on whether the image regions identified by each explainability method meaningfully influence the model's classification decision and how concentrated the resulting explanations are. Faithfulness is evaluated using Insertion and Deletion AUC, while sparsity is assessed using Effective Complexity. The methods are compared separately for each malware-to-image encoding investigated.

   b) **To what extent do different explainability methods for Vision Transformers provide consistent, stable, and computationally efficient explanations?** This question compares the consistency, stability, and computational efficiency of different explainability methods for Vision Transformers. Consistency evaluates whether similar inputs produce similar explanations under class-preserving perturbations, whereas stability measures the agreement between explanations produced by independently trained models. Computational efficiency is assessed using the runtime required to generate explanations.

   c) **To what extent can visualization techniques provide additional insight into the model's learned feature space?** The learned feature representations are analyzed using PCA and t-SNE projections to investigate the organization and separability of malware families within the feature space. In addition, the silhouette score is used to quantitatively assess cluster quality and complement the qualitative visual analysis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3. **To what extent can highly influential regions identified by explainability methods for Vision Transformers be mapped back to underlying program structures and meaningfully interpreted through reverse engineering?** This research question addresses the gap between image-based explanations produced by Vision Transformer models and the executable artifacts used during malware analysis and reverse engineering. To bridge this gap, a reverse-traceability framework is developed to map highly influential image regions back to the corresponding executable artifacts. The framework is evaluated against the derived design requirements through quantitative traceability analyses, perturbation experiments, and reverse-engineering case studies to assess whether the recovered artifacts provide meaningful insight into the model's malware family classification decisions.

1.3 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 introduces the theoretical foundations of malware analysis, malware visualization, Vision Transformers, and explainable artificial intelligence. Chapter 3 reviews the state of the art in malware-to-image encoding, Vision Transformer-based malware classification, explainability, and reverse traceability, and identifies the research gaps addressed in this thesis. Chapter 4 describes the research methodology and experimental design. Chapter 5 presents the experimental results for malware family classification, explainability, and reverse traceability. Finally, Chapter 6 discusses the findings with respect to the research questions, summarizes the contributions of this thesis, and outlines directions for future work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background

This chapter provides the essential knowledge needed to understand the remainder of the thesis. It begins by introducing malware and the fundamental objectives of malware analysis. It then describes how malware can be represented for learning-based systems and how such representations are used in practice, with a particular emphasis on Vision Transformer models. The chapter concludes with an overview of Explainable Artificial Intelligence (XAI) for image-based malware classification.

2.1 Malware Analysis

Malware analysis is the systematic examination of malware to understand its functionality, identify its presence, and support its removal or mitigation. Through malware analysis, security analysts aim to understand what a malware sample does, how its impact can be assessed, and how its harmful effects can be contained, either by eliminating the malware or preventing its activities. As malware continues to grow in scale and complexity, effective malware analysis has become a fundamental component of modern cybersecurity practice [SH12].

2.1.1 Malware Definition and Security Impact

Malware, short for malicious software, refers to software intentionally designed to harm computer systems, networks, or users. It is a broad term encompassing intrusive or hostile programs that perform unauthorized or harmful actions. Typical malware functionalities include disrupting business operations, collecting sensitive information for leakage or exploitation, and gaining unauthorized access to system resources [DJT+22]. Malware attacks most often lead to severe damage in reputation and financial loss. As a consequence, a wide range of anti-malware solutions has been developed and deployed in practice. However, as defensive technologies become more sophisticated, attackers

continuously adapt by designing increasingly elaborate and evasive malware [YLAI17]. Malware variants are created when cybercriminals modify existing samples by applying additional obfuscation techniques or making minor code changes to evade detection [DJT+22].

From a security perspective, malware may threaten one or more aspects of the confidentiality, integrity, and availability (CIA) of information systems. It may compromise confidentiality by stealing sensitive data, integrity by modifying data or system behavior without authorization, and availability by disrupting access to systems or encrypting data, as in ransomware attacks. Modern malware often affects multiple aspects of the CIA triad simultaneously, highlighting the importance of effective malware analysis and detection [BJS25].

2.1.2 Malware Types and Malware Families

Malware analysis can be performed more effectively when the intended objectives of a malware sample are known or can be inferred. To support this, malware is commonly categorized according to different levels of abstraction: malware type and malware family. In the literature, those terms are often used inconsistently. To avoid ambiguity, this thesis follows the definitions adopted by Banin et al. [BD18].

Malware types refer to broad classes of malicious software characterized by their general functionality, that is, what the malware aims to achieve. Typical malware types include viruses, worms, trojans, and backdoors. Viruses replicate by infecting other files; worms spread autonomously across networks; trojans disguise themselves as benign software while performing malicious action; and backdoors provide unauthorized remote access to compromised systems [SDG23]. The boundaries between malware types may overlap – a single malicious software sample may exhibit behavior associated with multiple malware types. As malware type categorization focuses primarily on high-level intent, it does not consider implementation details or code reuse [BJS25].

Malware families, in contrast, group malware samples by their specific functionality, describing how they operate and which methods they employ to achieve their goals, and are characterized by aspects such as their codebase or development origin. Samples within the same family often evolve through versions, retaining identifiable artifacts that allow them to be grouped despite modifications [BJS25]. Examples of malware families include Allaple, a polymorphic worm, Emotet, a modular trojan, and Zeus, a banking trojan [BJS25].

Understanding which malware family a sample belongs to is particularly valuable for security analysts: grouping samples based on similar code structures or development lineage allows analysts to infer likely behavior patterns for previously unseen malware variants. This can therefore guide the selection of appropriate detection, mitigation, and remediation strategies. Additionally, malware family classification facilitates knowledge sharing among analysts [SHKS19].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.1.3 Traditional Malware Analysis Techniques

Early malware research has been primarily driven by the objective of malware detection, that is, deciding whether a given software sample is benign or malicious. Traditional antivirus systems rely mainly on signature- and heuristic-based methods and, despite their name, are not limited to detecting viruses but target various types of malware. Signature-based methods compare software samples against known byte patterns or hashes and are highly effective for detecting previously identified malware. However, they are less effective against new variants, as even minor code modifications or obfuscation techniques can alter the signature and evade detection [GMP20, DJT+22]. Heuristic-based methods use manually defined rules based on structural or behavioral characteristics to identify suspicious samples. While they can detect some previously unseen malware, their effectiveness depends heavily on expert knowledge and manual rule engineering, which limits scalability [GMP20, DJT+22].

While signature-based and heuristic-based methods distinguish malware detection approaches based on how malicious software is identified, malware analysis techniques can also be categorized by how information is extracted from malware samples, namely through static, dynamic, and hybrid analysis [GBS14]. The two primary approaches are:

- Static analysis describes the process of analyzing a malware sample without executing it. Static analysis is efficient and safe as it does not require executing potentially harmful code. However, it might be more susceptible to code obfuscation, which is increasingly employed by modern malware.
- Dynamic analysis involves executing the malware in order to observe its behavior. This approach is generally more robust to obfuscation techniques than static analysis, as it analyzes the program during execution. However, dynamic analysis is time-consuming and more complex because it requires a controlled, secure execution environment. In addition, malware may employ evasion strategies, such as detecting when being executed in a sandbox environment and subsequently suppressing its malicious behavior, thereby limiting the effectiveness of dynamic analysis.

Hybrid analysis combines elements of both approaches [GMP20]. In the literature, memory analysis is sometimes considered a fourth category of malware analysis [SDG23, Khu20]. It examines memory images of running systems to identify forensic artifacts. However, because it is primarily used for post-infection forensic investigation rather than malware detection [KA18], it is treated here as complementary to dynamic analysis rather than as a separate category.

2.2 Machine learning for malware analysis

Traditional malware detection methods, such as signature-based and heuristic approaches, rely on predefined signatures or expert-defined rules. However, these approaches face

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

limitations in keeping pace with evolving malware and the large volume of new attacks and variants, while signature-based methods in particular are unable to detect unknown malware variants. These limitations have motivated the adoption of machine learning methods, which are well-suited for processing large volumes of data and have increasingly been applied to malware detection and classification [GMP20].

Machine learning-based malware analysis formulates tasks such as detection and classification as data-driven learning problems. Traditional machine learning methods typically operate on features extracted from malware samples, which provide an abstract representation of the software. These features are then used to train models for specific malware analysis tasks. A typical machine learning workflow, therefore, involves collecting and preparing malware samples, extracting suitable feature representations, and training and validating models [GMP20].

Ucci et al. [UAB19] categorize machine learning-based malware analysis along three dimensions: objective, feature, and malware analysis algorithm:

- **Objective** defines the task to be solved, such as malware detection, similarity analysis, or category detection. Similarity analysis identifies relationships between malware samples, including variants, families, similarities, and differences, whereas category detection provides a coarse-grained categorization according to prominent behaviors and objectives, such as spyware or ransomware.
- **Feature** describes how malware samples are represented for learning. Features may be derived using static, dynamic, or hybrid analysis and include representations such as byte sequences, opcode sequences representing executable instructions, PE metadata describing structural properties of Windows executable files, and runtime traces.
- **Malware Analysis Algorithm** describes the algorithms used to analyze malware samples, which can be grouped according to learning paradigms such as supervised, unsupervised, or semi-supervised learning.

In this thesis, the primary objective is malware family classification, which corresponds to family detection within the malware similarity analysis objective. The choice of family-level categorization is motivated by prior work indicating that malware families capture implementation-level similarities and code reuse. These characteristics are not reflected by broader malware-type labels and can facilitate effective classification [BD18]. With respect to the feature dimension, the analysis focuses on statically derived features, such as binary byte sequences and opcode sequences. At the algorithm level, supervised learning is employed because labeled malware family information is available and required for classification tasks.

In supervised learning, a labeled dataset is typically divided into three disjoint subsets: a training set, a validation set, and a test set. The training set is used to optimize the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

model parameters. The validation set is used to monitor performance during training and to guide model selection and hyperparameter tuning, which involves selecting settings that control the model or its training process. The test set is reserved for the final evaluation and provides an estimate of the model's ability to generalize to previously unseen data.

2.3 Static Malware Analysis and Feature Representations

Malware analysis approaches are inherently platform-dependent, as, e.g., Windows executables, i.e., Portable Executables (PEs) and Android apps are significantly different in how their operating system and application work [UAB19]. Consequently, malware detection techniques are typically designed for a specific platform, and research efforts commonly focus on a single operating system [LWZ+23]. In practice, Windows continues to be a major target for malicious actors: industry reports indicate that the majority of malware detected in 2024 was associated with Windows systems, while independent governmental analyses report a sustained increase in malware variants exploiting Windows-specific vulnerabilities [Kas24, Bun24]. For these reasons, we focus on Windows PE malware.

2.3.1 Windows Portable Executable File

The Portable Executable (PE) format is the standard file format used for executable programs on 32-bit and 64-bit Microsoft Windows operating systems. Among the various PE-based file types, executable files (.exe) are the most common, while other formats such as dynamic link libraries (.dll), batch files (.bat), and device drivers (.sys) also exist [MMC24]. A PE file is composed of a number of headers and sections that provide the dynamic linker with the information required to load and map the executable into memory (see Fig. 2.1). The PE header stores essential metadata about the executable, including the number of sections, file characteristics, and the size of the optional header. Furthermore, it contains the Import Address Table (IAT), which provides a mapping that the program uses to resolve and invoke functions located in external modules [GMP20]. In addition to the PE header, a Portable Executable is organized into several sections that store the executable's code and data. The following are examples [MMC24]:

- The .data section stores initialized global and static data.
- The .rdata section contains data such as constants, string literals, and, in some cases, debug directory information.
- The .idata section contains import information, including referenced DLLs and API functions.
- The .text section stores the executable machine code that is loaded and executed at runtime.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

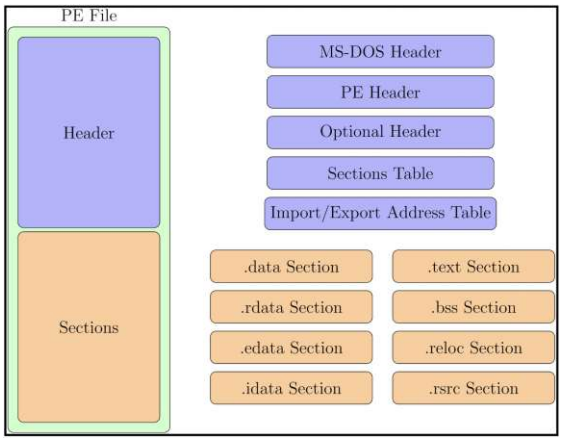


Figure 2.1: Portable executable file format [GMP20].

2.3.2 Static Feature Extraction from Windows Portable Executables

Static features are derived from a program without executing it. In the context of Windows Portable Executable files, such features are extracted from low-level representations that are directly available in the executable. Although malware is typically written using high-level programming languages such as C++ or C#, the distributed PE file contains only machine code produced during compilation [SH12]. Consequently, static malware analysis operates on raw byte sequences at the machine-code level and on assembly-level representations reconstructed through disassembly (see Fig. 2.2). Packing can complicate this analysis by transforming the executable, for example, through compression or encryption, while including unpacking code that reconstructs the original program during execution. As a result, the code and features directly observable in the packed executable can differ from those of the original program. These representations enable the extraction of characteristics such as byte patterns, section contents, and instruction-level features. Commonly used disassembly tools for Windows executables include IDA Pro and Radare2 [GMP20].

In line with the feature dimension introduced in Section 2.2, different static feature types may be derived from the same extracted information; they vary in how they are represented for learning-based analysis. Common examples include string-based features, which capture printable character sequences such as file paths and URLs, and Byte N-Grams and Opcode N-Grams, which model local byte-level and instruction-level patterns. API Function Call features represent imported operating system functions, while Entropy-based features characterize the statistical randomness of byte sequences. Graph-based representations, such as Control Flow Graphs and Function Call Graphs, capture structural relationships within the program. Visualization-based Features transform malware artifacts, such as raw byte sequences or structural characteristics, into image representations that can be processed by computer vision models [GMP20].

Of particular relevance to this thesis are Visualization-based Features, which are discussed

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

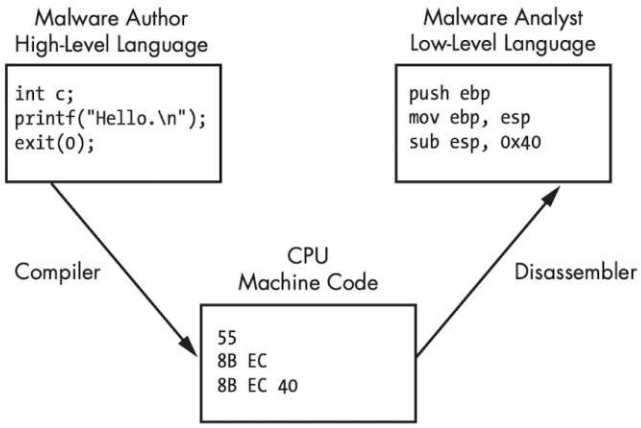


Figure 2.2: Relationship between high-level source code, machine code, and disassembly used in static malware analysis [SH12].

in more detail in the following section.

2.3.3 Visualisation-based Features

Visualization-based features transform malware artifacts into image-like representations by mapping low-level program information, such as byte values, opcode sequences, or structural properties, to pixel intensities [BPA+24]. Throughout this thesis, we follow this definition; in particular, only representations that explicitly map to pixel intensities are considered.

It should be noted that malware visualization has been used for various malware analysis purposes, including detection and classification, by illustrating malware characteristics and behavior in visual form, for example, through graphs, maps, or other graphical representations of relationships and activities. [EH20]. In this thesis, visualization, however, is considered specifically as a feature representation for learning-based malware detection and classification.

The underlying idea is the transformation of program information into a two-dimensional visual representation. As illustrated in Figure 2.3, malware binaries are first interpreted as numerical sequences, typically by reading raw byte values directly from the executable. These values are then reshaped or organized into fixed-size numerical structures (e.g., an 8-bit vector) and mapped to pixel intensities, resulting in grayscale or color images that can be processed by learning-based models [KRM+18]. This representation pipeline provides a generic abstraction that underlies many visualization-based malware detection approaches.

Visualization-based malware representations can be characterized according to the type of information encoded in the generated images, including the following categories [BPA+24]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

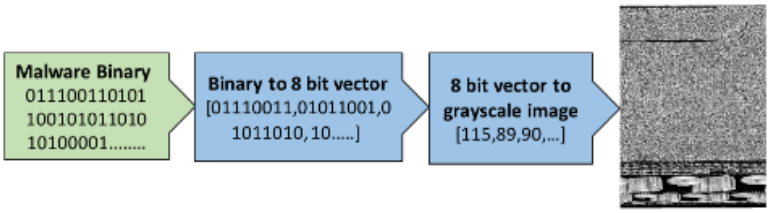


Figure 2.3: Illustration of the visualization-based feature generation process, transforming a malware binary into an image by mapping byte values to pixel intensities [KRM+18].

- Raw bytecode–based representations interpret executable files as sequences of byte values and map these values directly to grayscale or color pixels, thereby preserving low-level binary structure in visual form.
- Metadata-based representations encode static structural information embedded in the executable, such as header fields, section layouts, or import relationships, emphasizing the organization and composition of malware binaries.
- Run-time–based representations are constructed from dynamic analysis data collected during execution, for example, system calls, API interactions, or network activity, and reflect behavioral characteristics observed at run time

In line with the static analysis focus of this thesis, only raw bytecode–based and metadata-based representations are considered, while run-time–based representations are excluded.

Visualization-based features offer several advantages for malware analysis. They enable the exploitation of visual similarities between malware samples, provide a global view of the executable while preserving local details, and allow the use of computer vision models without requiring extensive manual feature engineering [BAC+26]. In addition, byte-level image representations have shown resilience to certain packing techniques, which compress or transform executable code to obscure its original representation and hinder static analysis [NYPZ11, RCS23]. At the same time, certain visualization-based representations, particularly grayscale images derived directly from raw byte sequences, exhibit inherent limitations related to the image construction process. Executable binaries are not inherently two-dimensional objects, and reshaping byte sequences into images introduces inductive biases that do not directly correspond to program semantics. In particular, the selection of image dimensions constitutes an additional design parameter and may introduce artificial spatial relationships, as bytes that are distant in the original one-dimensional sequence may become spatially adjacent after reshaping, depending on the chosen image width. Furthermore, as with many static feature representations, byte-level image-based approaches may be vulnerable to obfuscation techniques like encryption [GMP20].

While visualization-based features enable the application of computer vision models to malware analysis, their effectiveness depends on both the chosen representation and the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

expressive power of the underlying learning architecture, thereby motivating the use of modern deep learning techniques; these are discussed in the next section.

2.4 Deep Learning for Malware Classification

Deep learning is a subfield of machine learning, which is based on artificial neural networks with many layers [YT21]. It has enabled significant progress in solving problems that remained challenging for the machine learning community and has been applied to domains such as image and speech recognition, natural language processing, and scientific data analysis [LBH15]. This includes applications in malware detection and classification, where several types of deep learning architectures, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and autoencoders, have been applied to detect Windows malware using different feature representations derived from both benign and malicious files [MMC24].

Traditional machine learning approaches for malware detection and classification rely on classifiers such as support vector machines (SVMs) and Naive Bayes, among other machine learning algorithms [KRM+18]. However, these approaches inherently face limitations when processing raw data and thus require (often handcrafted) feature extraction to transform raw inputs into feature representations, a process that demands substantial domain expertise [LBH15].

Deep learning reduces the need for manual feature engineering by enabling end-to-end learning, in which feature representations and the mapping to the desired output are learned jointly. Representation learning more generally refers to methods that enable a model to automatically discover representations from raw data rather than relying solely on manually designed features. Deep learning extends this principle by learning multiple levels of representation through the composition of simple but non-linear transformations, where each level maps the representation from the previous layer to a more abstract one. By stacking multiple such transformations, deep learning models can learn increasingly complex functions [LBH15]. Due to their multi-layered structure, deep learning models are well-suited for high-dimensional data. This capability is particularly relevant to malware analysis, which involves complex and high-dimensional data [DA23]. Deep learning models have shown promising results in efficiently handling unstructured or unlabeled data as well as large datasets [MMC24]. As a result, deep learning–based approaches for malware detection and classification have attracted increasing attention in recent research.

2.4.1 Deep Learning Fundamentals

The general architecture of a deep learning network consists of three types of layers. The first is the input layer, which receives the input data. The hidden layers process the input by extracting, selecting, and learning abstract and informative feature representations. The final layer is the output layer, which produces the model's prediction [MMC24].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Depending on the learning task, the form of the output may differ. In this thesis, deep learning is applied to classification. Let $D = \{(x_i, y_i)\}_{i=1}^{N}$ denote a labeled dataset, where x_i is the input representation and $y_i \in \{1, \ldots, C\}$ is the corresponding class label. A classification model f_θ with parameters θ maps an input x_i to predicted class probabilities $\hat{\mathbf{p}}_i = f_\theta(x_i)$. The predicted class label is then obtained as

$$\hat{y}_i = \arg \max_{c \in \{1,\ldots,C\}} \hat{p}_{i,c}. \tag{2.1}$$

In supervised deep learning, models are trained using labeled samples from the training set. During training, the network produces predictions for each input sample, which are compared to the corresponding ground-truth labels using a loss function measuring the prediction deviation. For multi-class classification problems, the cross-entropy loss is commonly used and is defined as

$$\mathcal{L} = -\frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \sum_{c=1}^{C} y_{i,c} \log(\hat{p}_{i,c}), \tag{2.2}$$

where N_{train} denotes the number of samples in the training set, C the number of classes, $y_{i,c} \in \{0, 1\}$ indicates whether sample i belongs to class c, and $\hat{p}_{i,c}$ is the predicted probability that sample i belongs to class c.

The model parameters, including weights and biases, are adjusted to minimize this loss. This optimization process often relies on gradient-based learning methods, which compute the gradient of the loss function with respect to the model parameters and iteratively update them in the direction that reduces the prediction error.

The separation into training, validation, and test sets helps reduce the risk of overfitting, i.e., the situation where a model learns patterns specific to the training data but fails to generalize well to previously unseen samples.

Transfer learning is widely used to improve training efficiency and predictive performance, particularly when labeled data is limited. It enables models to leverage representations learned from large-scale source datasets and adapt them to a target setting through fine-tuning. Transfer learning has also been shown to be effective for malware classification [GMP20].

2.4.2 Evaluation Metrics

The performance of classification models can be assessed from the complementary perspectives of effectiveness and efficiency. Effectiveness describes how accurately a model predicts the correct target labels, while efficiency captures the computational resources required during training and inference. Effectiveness metrics are computed on the test set D_{test}, to estimate their generalization capabilities.

For multi-class classification, effectiveness is often evaluated using metrics such as accuracy, precision, recall, and F1-score. The following quantities are used to define these metrics for a class $c \in \{1, \ldots, C\}$:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- **True Positive** (TP_j) denotes the number of samples of class c_j that are correctly classified as c_j.
- **False Positive** (FP_j) denotes the number of samples belonging to other classes that are incorrectly classified as c_j.
- **False Negative** (FN_j) denotes the number of samples of class c_j that are incorrectly classified as another class.
- **True Negative** (TN_j) denotes the number of samples not belonging to class c_j that are correctly classified as not belonging to c_j.

We can then define the following metrics:

- **Accuracy** (Acc) measures the overall proportion of correctly classified samples:

$$Acc = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbf{1}(\hat{y}_i = y_i), \tag{2.3}$$

  where N_{test} denotes the number of samples in the test set and $\mathbf{1}(\cdot)$ the indicator function.

- **Precision** (P_c) measures the proportion of samples predicted as class c that are correctly classified:

$$P_c = \frac{TP_c}{TP_c + FP_c}. \tag{2.4}$$

- **Recall** (R_c) measures the proportion of samples of class c that are correctly identified:

$$R_c = \frac{TP_c}{TP_c + FN_c}. \tag{2.5}$$

- **F1-score** ($F_{1,c}$) combines precision and recall as their harmonic mean:

$$F_{1,c} = 2 \cdot \frac{P_c \cdot R_c}{P_c + R_c}. \tag{2.6}$$

- **False Positive Rate** (FPR_c) measures the proportion of samples not belonging to class c that are incorrectly classified as class c:

$$FPR_c = \frac{FP_c}{FP_c + TN_c}. \tag{2.7}$$

- **Confusion Matrix** summarizes class-wise classification outcomes. Each entry (i, j) represents the number of samples of class i predicted as class j, with diagonal entries indicating correct classifications and off-diagonal entries indicating misclassifications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

In multi-class classification, the macro-averaged F1-score (F_1^{macro}) is obtained by computing the F1-score independently for each class and averaging the resulting values:

$$F_1^{\text{macro}} = \frac{1}{C} \sum_{c=1}^{C} F_{1,c}. \tag{2.8}$$

This averaging scheme assigns equal importance to all classes regardless of their frequency and is therefore particularly suitable for imbalanced datasets. This is often the case in malware classification datasets, with some malware families represented by substantially more samples than others. Consequently, macro-averaged metrics are particularly important because they prevent the majority class(es) from dominating the overall evaluation.

In addition to predictive performance, computational efficiency can be evaluated using metrics such as training and inference times, while convergence behavior characterizes how quickly a model reaches stable validation performance during training.

2.4.3 Convolutional Neural Networks for Malware Classification

Convolutional neural networks (CNNs) are a class of artificial neural networks designed to process spatially structured data by exploiting local connectivity and hierarchical feature learning. Within the field of computer vision, CNNs are widely adopted for image classification and object detection tasks because of their ability to efficiently capture spatial patterns and structured visual information [GMPV19]. These strengths have also led to their widespread use in malware analysis, where CNNs currently represent the most commonly used architecture for image-based malware classification [BAC+26]. As such, CNNs provide an important conceptual baseline for understanding alternative architectures, including the Vision Transformers investigated in this thesis.

The effectiveness of CNNs in malware analysis is closely linked to visualization-based representations, as discussed in 2.3.3. By treating malware classification as an image classification problem, CNNs can learn discriminative visual patterns of malware families directly from malware images [KRM+18].

While neural network layers can be distinguished by their role as input, hidden, or output layers, they can also be characterized by the operations they perform. A typical CNN uses convolutional, pooling, and fully connected layers. Convolutional layers apply learnable filters to local regions of the input. Each filter slides across the input and computes a value at each position based on the corresponding local region. The values obtained across all positions form a feature map, which represents the presence of the feature learned by the filter at different spatial locations. Multiple filters are typically applied within a convolutional layer, resulting in multiple feature maps. Through this process, convolutional layers can learn features such as edges, color variations, and textures, while deeper layers can capture increasingly complex patterns. The resulting feature maps are typically passed through a non-linear activation function, such as the rectified linear unit (ReLU) [KRM+18].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Pooling layers downsample the feature maps by reducing their spatial dimensions. For example, max pooling retains the largest value within each local region, thereby reducing the amount of data processed by subsequent layers. Fully connected layers combine the features learned by the preceding layers to perform classification. In these layers, each neuron is connected to every neuron in the previous layer [KRM+18]. Dropout may be applied during training to reduce overfitting by randomly setting a proportion of neuron activations to zero, thereby improving the model's generalization [WG15].

CNNs offer several advantages for image-based malware classification. A key strength of CNN-based approaches lies in their ability to automatically extract spatial features from malware binaries that are represented as images. This reduces the need for handcrafted features. When applied to images derived from static information, CNN-based approaches can classify malware without requiring execution of the analyzed binaries. Additionally, CNNs are computationally efficient during inference, making them suitable for real-time and large-volume malware detection systems. Their compatibility with transfer learning further allows pre-trained models to be adapted for malware classification tasks, supporting scalability across diverse datasets [AM25]. CNNs also incorporate strong inductive biases, such as locality, two-dimensional neighborhood structure, and translation equivariance. Locality means that convolutional operations process local regions of the input, while a two-dimensional neighborhood structure reflects the spatial relationships between neighboring regions in an image. Translation equivariance means that shifting a feature in the input results in a corresponding shift in the resulting feature representation [DBK+21]. These inductive biases can be beneficial when training data is limited, as they provide prior assumptions about image structure that help CNNs learn effectively from smaller amounts of training data [DBK+21].

Despite these advantages, the local nature of convolution can also impose limitations when relationships between spatially distant regions within an individual input need to be captured. The size of the local region processed by a convolutional operation is determined by the size of its filter. Using larger filters allows each convolutional operation to incorporate information from a larger local region, but generally increases the number of learnable parameters and the computational cost of the operation. The receptive field, in contrast, describes the region of the original input that can influence a particular activation. Although individual convolutional operations are local, the receptive field increases as multiple convolutional layers are stacked, allowing information from increasingly distant regions of the input to be combined. Consequently, modeling long-range dependencies may require information to be propagated across multiple layers before spatially distant regions can influence the same representation [RCS23].

This limitation is particularly relevant for malware images constructed from linear byte sequences, as their two-dimensional spatial layout does not directly reflect program structure. Consequently, semantically related regions of a binary may be spatially distant in the image representation and, therefore, difficult for CNNs to connect through local convolutional operations. In addition, the low-level visual patterns typically captured by convolutional filters, such as edges or texture variations, do not necessarily correspond

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

to meaningful features of the underlying executable. These limitations motivate the exploration of architectures that can more explicitly model global interactions.

2.4.4 Attention Mechanism

The attention or self-attention mechanism enables a model to explicitly capture relationships between all elements of an input sequence by dynamically weighting their relevance to one another. It is applied in deep learning architectures such as Transformers and graph attention networks across numerous fields, including natural language processing, graph mining, or computer vision [RZ24].

Architecturally, self-attention updates each input element by computing a weighted aggregation over the representations of all elements in the input sequence. Given an input sequence of n elements represented as $X \in \mathbb{R}^{n \times d}$, each input element is represented by a d-dimensional embedding vector, where d denotes the embedding dimension, i.e., the number of features in the vector representation of each element. Self-attention then projects these representations using three learnable weight matrices, referred to as the query ($W_Q \in \mathbb{R}^{d \times d_q}$), key ($W_K \in \mathbb{R}^{d \times d_k}$), and value ($W_V \in \mathbb{R}^{d \times d_v}$) projection matrices, with $d_q = d_k$. The corresponding query, key, and value representations are obtained as

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \tag{2.9}$$

The query and key representations are used to determine the relevance between input elements, while the value representations contain the information to be aggregated. Specifically, the similarity between queries and keys yields attention scores indicating how strongly each element attends to the others. These scores are normalized using a softmax function, which converts them into non-negative attention weights that sum to one. The resulting weights are subsequently used to form a weighted combination of the value representations. The output of a self-attention layer can be expressed as

$$Z = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_q}}\right)V. \tag{2.10}$$

where the resulting representation $Z \in \mathbb{R}^{n \times d_v}$ encodes contextual information aggregated from the entire input sequence [KNH+22].

Multi-head attention generalizes self-attention by computing multiple attention mappings in parallel, each based on different learned projections of the input. By combining these mappings, the model can capture diverse interaction patterns and increase representational capacity while preserving the core attention formulation [KNH+22].

In contrast to convolutional operations, the self-attention mechanism operates dynamically, whereas convolution uses static filters that remain unchanged across inputs. Additionally, self-attention does not require a grid structure like convolution, as it is invariant to permutations and can flexibly handle sequences with different numbers of input points. Since

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

self-attention alone does not encode the order or position of input elements, positional information can be incorporated to provide the model with information about their relative or absolute positions. With such positional information, self-attention can flexibly model both local and global patterns and, under appropriate parameterizations, approximate convolutional behavior while adaptively determining receptive fields [KNH+22].

These properties make self-attention particularly well-suited for representations where relationships are not necessarily linked to spatial closeness. In malware analysis, visualization-based representations derived from binary code may exhibit semantically related regions that are spatially distant due to the reshaping process. By enabling interactions among arbitrary input elements, attention mechanisms provide a principled foundation for modeling such global dependencies, motivating their application to visual representations beyond natural images.

2.4.5 Vision Transformers for Malware Classification

Self-attention-based architectures have become the de facto standard in natural language preprocessing (NLP). Since Transformer architectures rely primarily on self-attention mechanisms rather than convolutional operations, their success has motivated extensions to computer vision. Because Transformer architectures operate on sequences, they can also be applied to images when the images are represented as sequences of patches. Images can be divided into patches, which are projected into vector embeddings that form a sequence of tokens serving as input to a Transformer architecture, analogous to words in a sentence [DBK+21].

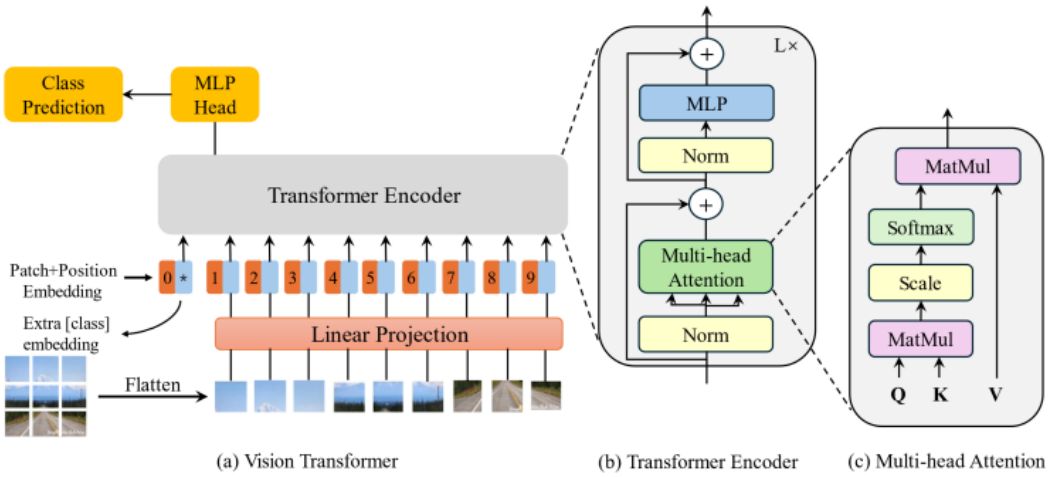


Figure 2.4: Example of a Vision Transformer architecture [Lon24].

An overview of the Vision Transformer (ViT) architecture is illustrated in Figure 2.4. The main components of the architecture are described in detail in the following paragraphs. In the figure, "MatMul" represents matrix multiplication, while "Scale" corresponds to the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

division of the query-key similarity scores by $\sqrt{d_q}$. "Norm" refers to layer normalization, and "MLP" to the multilayer perceptron. The MatMul, Scale, and softmax operations within the multi-head attention block correspond to the self-attention computation described in Section 2.4.4.

Vision Transformers adopt the encoder component of the original Transformer architecture and apply it directly to sequences of embedded image patches. As the Transformer architecture expects a 1D sequence of token embeddings, the 2D image first needs to be reshaped from $x \in \mathbb{R}^{H \times W \times C}$ into fixed-sized patches of resolution $P \times P$. This results in a sequence of patches, with $N = \frac{HW}{P^2}$ patches. After partitioning the image into patches, each patch is flattened into a vector. Since the Transformer operates on token embeddings of fixed dimensionality D, each flattened patch is mapped into a D-dimensional latent space using a learnable linear projection matrix $\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$. The resulting patch embeddings form the basis of the input token sequence to the Transformer encoder [DBK+21].

A learnable classification token $\mathbf{x}_{\text{class}}$ is added to the sequence of embedded patches. The output representation corresponding to this token after the final Transformer encoder layer serves as the global image representation. This representation is then passed to a classification head for predicting the target class [DBK+21].

Since the self-attention mechanism itself does not encode positional information, learnable one-dimensional positional embeddings $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D}$ are added to the patch embeddings and the classification token. These embeddings provide information about the ordering of the patches within the sequence. The resulting token sequence serves as input to the Transformer encoder and, according to [DBK+21] is given by:

$$\mathbf{z}_0 = \left[\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \ldots; \mathbf{x}_p^N\mathbf{E}\right] + \mathbf{E}_{\text{pos}}. \tag{2.11}$$

The Transformer encoder follows the architecture introduced by Vaswani et al. and consists of L identical layers. Each layer consists of two sub-layers: the multi-head self-attention mechanism, as described in Section 2.4.4, and the position-wise feed-forward network (FFN) [VSP+17]. In the Vision Transformer, the FFN is implemented as a multilayer perceptron (MLP). The term position-wise means that the same MLP is applied independently to each token representation. For each layer $\ell = 1, \ldots, L$, the updates are defined as:

$$\mathbf{z}'_\ell = \mathbf{z}_{\ell-1} + \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1}))\,, \tag{2.12}$$

$$\mathbf{z}_\ell = \mathbf{z}'_\ell + \text{MLP}(\text{LN}(\mathbf{z}'_\ell))\,. \tag{2.13}$$

The MLP consists primarily of fully connected layers combined with non-linear activation functions such as the Gaussian Error Linear Unit (GELU). The GELU activation is defined as

$$\text{GELU}(x) = x\Phi(x), \tag{2.14}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

where $\Phi(x)$ denotes the cumulative distribution function of the standard normal distribution. GELU introduces non-linearity by weighting an input according to its value. The MLP enhances the model's expressive power by enabling the learning of more complex feature transformations [Lon24].

In the Vision Transformer, layer normalization is applied before each sub-layer to normalize the features of each token representation based on their mean and variance. Residual connections are employed after every block to add the input of each sub-layer to its output. In the equations above, this is represented by the addition of $\mathbf{z}_{\ell-1}$ to the output of the multi-head self-attention sub-layer and of $\mathbf{z}'_\ell$ to the output of the MLP sub-layer. This allows information from the original representation to be preserved in the resulting representation. After the final encoder layer, an additional layer normalization is applied to the resulting token representations, following the same normalization procedure as within the encoder blocks. The representation corresponding to the classification token is then extracted and passed to a classification head:

$$\mathbf{y} = \mathrm{LN}(\mathbf{z}_L^{(0)}). \tag{2.15}$$

Here, $\mathbf{z}_L^{(0)}$ denotes the embedding of the classification token after the final encoder layer. Due to their weaker inductive bias compared to convolutional neural networks, Vision Transformers typically require larger training datasets to achieve competitive performance [DBK+21, KNH+22]. Therefore, Vision Transformers are often pre-trained on large datasets to learn general representations and subsequently fine-tuned on task-specific data to adapt the pre-trained model to the target task [DBK+21]. In the original Vision Transformer, the classification head is implemented as a multilayer perceptron during pre-training and as a linear layer during fine-tuning [DBK+21].

Visualization-based malware representations can be constructed by transforming linear byte sequences into two-dimensional grids. This encoding introduces artificial spatial relationships that do not necessarily reflect the underlying program structure. As a result, close pixels in a malware image may correspond to semantically unrelated instructions, while semantically related code fragments may be separated spatially. As discussed previously, Vision Transformers exhibit less image-specific inductive bias than CNN architectures [DBK+21] and enable interactions between spatially distant regions through self-attention [KNH+22]. These characteristics might be beneficial for modeling global structural dependencies in malware images.

An example application of a Vision Transformer to visualization-based malware representations is illustrated in Figure 2.5.

While the reduced image-specific inductive bias of Vision Transformers may help address artificial spatial relationships in malware images, their ability to capture meaningful dependencies still depends on the information preserved by the underlying image representation. Feature representation is a critical design choice, as it impacts the model's ability to accurately analyze malicious patterns using transformer-based architectures [BPA+24]. When malware binaries are transformed into images, important structural

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

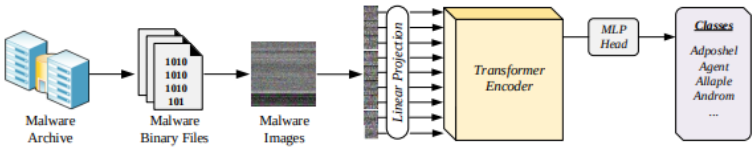


Figure 2.5: Example of a Vision Transformer architecture for malware classification [KY24].

relationships may be distorted or lost depending on the encoding method [YCA+25]. Since Vision Transformers operate on fixed-size patches, the chosen encoding strategy directly influences how structures are partitioned into tokens and, therefore, how attention mechanisms can capture dependencies.

Despite the previously discussed advantages of Vision Transformers for modeling dependencies in malware images, they also introduce trade-offs that must be considered. As discussed previously, their weaker inductive bias can result in larger data requirements compared to convolutional neural networks [DBK+21, KNH+22]. Additionally, the self-attention mechanism scales quadratically with the number of input tokens, leading to higher computational costs for high-resolution inputs or small patch sizes [KNH+22]. These aspects must be considered for malware classification scenarios where dataset size and computational resources may be constrained.

Vision Transformers are therefore adopted as the primary classification architecture in this thesis to investigate how different encoding strategies influence model performance.

2.5 Explainable Artificial Intelligence

Despite their strong performance, deep learning models are often criticized for their lack of transparency in the decision-making process. Their complex internal representations make it difficult to understand how specific inputs lead to particular predictions, resulting in so-called black-box behavior. This limitation is especially problematic in domains where decisions must be interpretable, e.g., security-critical tasks such as malware analysis [SMFC24].

Explainable Artificial Intelligence (XAI) has emerged as a research field that aims to make the behavior of machine learning models, including Deep Learning models, more understandable to humans by providing insights into their decision-making processes. As a result, XAI seeks to improve the transparency, interpretability, and explainability of AI systems [SMFC24].

To clarify the terminology used in this work, it is important to distinguish between interpretability and explainability. Interpretability is defined as the degree to which a human can understand the cause of a model's decision [BC17]. In contrast, explainability can be understood as a form of post-hoc interpretability, referring to the mechanisms or processes that provide such understanding [Lip18]. Although related, these terms are used

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

inconsistently in the literature and are often treated interchangeably [Mil19, SMFC24]. Moreover, their definitions lack sufficient precision for formalization and depend on user expertise and context [DBH18]. In this thesis, the distinction is used as a guiding framework, while acknowledging some remaining overlap.

2.5.1 Types of Explainability Methods

A range of methods has been proposed to analyze model behavior. These methods can be categorized along three dimensions, which distinguish them based on their inherent interpretability, their scope of explanation, and their dependence on the underlying model [AB18]:

- Intrinsic vs. Post-hoc (Extrinsic): Intrinsic methods aim to achieve interpretability by design, using models that are inherently interpretable, often at the expense of predictive performance. In contrast, post-hoc methods generate explanations for complex black-box models after training without modifying their internal structure.
- Local vs. Global: Local methods focus on giving explanations for individual predictions. Global methods, on the other hand, aim to describe the overall behavior and decision-making logic of the model as a whole.
- Model-specific vs. Model-agnostic: Model-specific methods are designed for particular types of models and rely on their internal structure. Model-agnostic methods treat the model as a black box and can be applied independently of the underlying model type.

In this thesis, the focus is primarily on post hoc explainability methods, as the considered ViT models are not inherently interpretable, despite providing internal attention mechanisms that can be leveraged for explanation. The main emphasis is on local explanations that describe individual predictions, while global analyses are additionally used to study overall model behavior. The employed methods are model-specific, as they rely on the internal structure of the vision transformer architecture.

2.5.2 Explainability in Computer Vision

Explainability in computer vision is especially relevant for image-based malware approaches, where understanding visual patterns is essential for interpreting model decisions.

As models in computer vision can be categorized into CNNs and attention-based models, like ViTs, XAI approaches can also be categorized into those two types of methods, with some overlap remaining of model-agnostic explainers. Typically, saliency maps, Layer-wise Relevance Propagation (LRP), Integrated Gradients, or Class Activation Mapping (CAM) methods are applied for CNNs [MLW+24]. Saliency maps highlight input regions based on their influence on the model prediction, while LRP propagates relevance scores backward through the network to determine the contribution of individual input features. Integrated

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Gradients attributes a prediction to input features by accumulating gradients between a reference input and the analyzed input, whereas CAM methods use convolutional feature maps to identify image regions that contribute to a particular class prediction. In the context of image-based malware analysis, explainability methods remain relatively rare. However, gradient-based visualization techniques such as Gradient-weighted Class Activation Mapping (Grad-CAM) have been frequently applied to highlight the regions of malware images that contribute most to classification decisions, making them a representative approach for interpreting CNN-based malware classifiers [SMFC24]. Grad-CAM uses the gradients associated with a target class to weight convolutional feature maps and generate a heatmap of the image regions relevant to the classification decision.

As Vision Transformers gain increasing attention, additional challenges for explainability arise from their reliance on self-attention mechanisms and increased architectural complexity. As a result, specialized explainability approaches have been developed, including attention visualization and methods such as Attention Rollout. Attention visualization examines the attention weights between image patches to provide insights into their interactions. Since attention is distributed across multiple heads and layers, individual attention maps provide only a partial view of how information is propagated through the model. Attention Rollout combines attention information across successive Transformer layers to capture how information is propagated between image patches throughout the network [MLW+24].

Given the increasing use of Vision Transformers in image-based malware classification, this work focuses on explainability techniques suitable for such models, while CNN-specific approaches are not considered in further detail.

2.5.3 Explainability in Malware Analysis

In XAI-based malware analysis, explainability can be integrated into the overall analysis process rather than considered as a standalone component. In the literature, this process is described as consisting of classification or detection, performance evaluation, explanation/interpretation generation, and explanation assessment, as illustrated in Figure 2.6. Within this process, explainability is applied after model evaluation to provide insight into how predictions are made [SMFC24].

In image-based malware classification, explainability methods typically produce visual representations such as saliency or relevance maps. Grad-CAM, as introduced in the previous subsection, has been applied to highlight regions of malware images that contribute most to classification decisions. Figure 2.7 illustrates such relevance maps for two different malware families [SMFC24].

Explainability is particularly relevant for image-based malware classification, where visual representations are derived from executable program data and do not directly correspond to human-interpretable semantics. As a result, the interpretation of model predictions depends not only on the chosen representation but also on the underlying learning architecture, such as Vision Transformers. Explainability methods, therefore,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

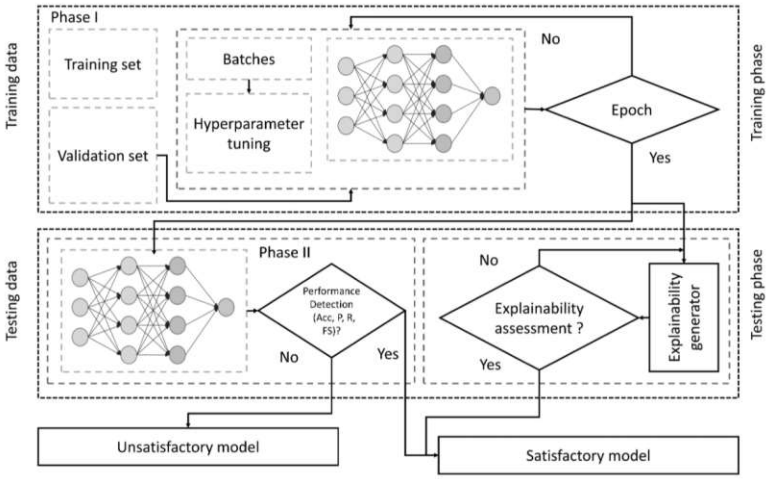


Figure 2.6: Workflow of an explainable malware analysis pipeline, integrating model training, performance evaluation, and explainability generation and assessment [SMFC24].

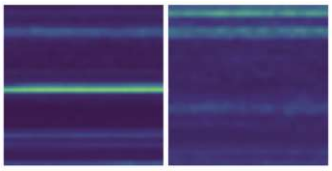


Figure 2.7: Example of Grad-CAM relevance maps for image-based malware classification [SMFC24].

play an important role in bridging this gap by providing insight into how models process such representations and which parts of the input contribute to a decision. However, the extent to which these insights can be meaningfully interpreted depends on whether the highlighted regions can be related back to the underlying program structures. This semantic gap and existing approaches for Vision Transformer explainability are examined in more detail in Section 3.3.

These challenges motivate a closer examination of existing approaches to malware representation, Vision Transformer-based classification, and Vision Transformer explainability methods, which are reviewed in the following chapter.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

State of the Art

This chapter reviews the state of the art relevant to this thesis. It first examines malware-to-image encoding strategies proposed in prior work, then Vision Transformer–based malware classification approaches and explainability methods for transformer models. Finally, it discusses evaluation practices for explainability and examines approaches for relating model explanations to malware analysis workflows.

3.1 Malware-to-Image Encoding Strategies in Prior Work

Visualization-based malware classification relies on transforming executable code into image-like representations. These image-based encoding strategies correspond to the visualization-based features defined in Section 2.3.3, as they describe how program information is mapped to pixel intensities.

Recent work conceptualizes this process as part of a sequential pipeline consisting of dataset collection, image transformation, feature processing, model generation, performance evaluation, and model robustness [BAC+26]. Within this pipeline, the transformation of malware samples into images corresponds to the second stage: image transformation. The choice of encoding determines which aspects of the underlying program are preserved, emphasized, or removed in the resulting image. Consequently, since images serve as input, encoding strategies directly influence the types of structural patterns that learning-based models can capture.

In the literature, a wide range of malware-to-image encoding techniques, differing in their level of semantic abstraction and structural preservation, have been proposed. Brosolo et al. [BAC+26] organize these approaches according to the type of image representation used during transformation, and distinguish five main categories: (i) grayscale-based encodings, (ii) RGB-based encodings, (iii) Markov-based encodings, (iv) hashing-based encodings, and (v) entropy-based encodings.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

While this categorization can provide a useful perspective on how malware samples are transformed into images, it primarily reflects the image construction technique rather than the level of abstraction or the type of underlying data representation. In practice, similar image types can be derived from different sources, such as raw byte or disassembled opcode sequences. In addition, all five categories can range from direct mappings of raw data to representations that incorporate intermediate feature processing steps. Accordingly, the resulting images may reflect either raw content or transformed feature representations.

As a result, it is important to note that individual encoding approaches within each category may differ significantly in terms of their representational properties and interpretability. Nevertheless, this taxonomy is adopted to structure the following discussion of encoding strategies while explicitly highlighting variations in the underlying data sources. Therefore, in the subsequent subsections, each encoding category is described in detail, followed by a discussion of layout strategies. A comparison of the trade-offs between the categories is provided in Section 3.1.7.

3.1.1 Grayscale-Based Encodings

Grayscale-based encodings are among the earliest and most widely adopted approaches in visualization-based malware detection [BAC+26]. In this approach, malware binaries are interpreted as sequences of numerical values, typically raw bytes, which are mapped to pixel intensities in the range of 0 to 255. The resulting one-dimensional sequence is then reshaped into a two-dimensional image (see Figure 3.1).

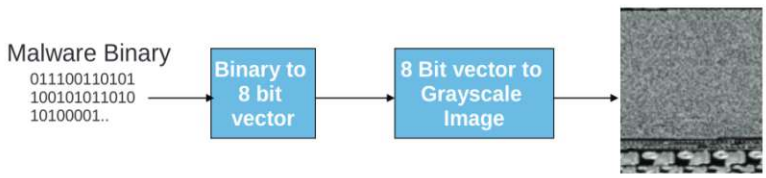


Figure 3.1: Transformation of malware binaries into grayscale images [NKJM11]

Raw Byte-Based Grayscale Representations. Nataraj et al. [NKJM11] were among the first to turn malware into grayscale images for classification. They proposed visualizing malware binaries by mapping raw byte values to pixel intensities and rendering them as images whose width adapts to the file size. The image height varies accordingly, allowing the representation to preserve the executable's structural characteristics. For classification, the authors extract global texture features using GIST descriptors and apply a k-nearest neighbor classifier. As the method operates directly on raw binary content, it distinguishes itself from many traditional malware analysis pipelines.

Subsequent work has adopted grayscale representations primarily due to their simplicity and computational efficiency [BAC+26]. While the fundamental byte-to-pixel mapping remains largely unchanged, later approaches differ in how images are constructed. In

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

contrast to the adaptive image layout proposed by Nataraj et al. [NKJM11], many studies enforce fixed-size image representations. This ensures compatibility with deep learning models such as CNNs and is typically achieved through preprocessing steps such as truncation, padding, or resizing of the byte sequence [LZKN22, CJKK17, LLLZ20].

Opcode-Based Grayscale Representations Grayscale images can, in principle, also be constructed from higher-level representations such as opcode sequences. In such approaches, opcode sequences are transformed into numerical representations, for instance by assigning integer identifiers, which are then arranged into matrices and interpreted as grayscale images [BAC+26]. However, in practice, direct opcode-to-pixel mappings are rarely employed. Instead, most approaches introduce intermediate processing steps that transform opcode sequences into more compact or discriminative representations prior to image construction.

For example, Darem et al. [DAM+21] combine grayscale image representations with opcode-level and structural features derived from disassembled binaries. In their approach, opcode sequences are first processed through feature engineering techniques, including n-gram analysis, ranking, and feature selection, before being transformed into a binary representation for a grayscale image. Similarly, Khan et al. [KZK19] construct grayscale images by using opcode bigrams weighted with statistical measures such as probability and information gain. In this approach, they use opcode pairs to capture local relationships within the instruction sequence, and the importance of these pairs is quantified through statistical weighting. In both approaches, the images encode structural or statistical properties instead of raw opcode sequences. This suggests a trade-off between representational expressiveness and direct learning from the data.

Strengths and Limitations. Grayscale encodings are relatively easy to implement and computationally efficient, since they typically operate on raw byte sequences. This makes them well-suited for large datasets [BAC+26] and robust to certain obfuscation techniques, such as packing and section encryption [NKJM11]. However, they enforce a two-dimensional spatial structure on inherently one-dimensional byte sequences, which may not reflect actual semantic relationships within the code. In addition, the use of fixed-size image representations requires resizing or truncation, which potentially leads to information loss and degradation of structural relationships [BAC+26]. Especially for larger or highly variable binaries, the original sequence coherence may be disrupted, resulting in reduced effectiveness of feature extraction [YCA+25]. Furthermore, as single-channel representations, grayscale images are limited in their ability to capture multiple complementary aspects of malware simultaneously.

3.1.2 RGB-Based Encodings

RGB-based encodings extend grayscale representations by mapping malware data to three-channel images, in which each pixel is represented by red, green, and blue intensities. This enables the simultaneous capture of multiple aspects of a malware sample [BAC+26,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

LWSW24]. Existing work proposes several approaches to constructing RGB malware images, which we categorize by how the three channels are defined.

RGB from raw bytes. A straightforward approach is to construct RGB images by only using raw binary data without additional feature extraction. Groups of three bytes are mapped to the red, green, and blue channels of a pixel. For example, a hexadecimal sequence such as 0x868816 can be interpreted as a pixel with values (R = 134, G = 136, B = 22). This type of encoding is often used as a baseline, for instance by Wang et al. [WZLZ19], and has also been adopted in later studies [BM24, DGS+23].

RGB from opcode and higher-level features. More advanced approaches construct RGB channels from higher-level representations. Rather than using raw bytes, they incorporate information such as opcode sequences, assembly instructions, or API calls. Jian et al. [JKR+21], for example, combine byte-level and opcode-based features within a three-channel representation. Similarly, Li et al. [LWSW24] propose a tri-channel encoding that integrates binary data, assembly-level information, and API call features. Overall, these approaches aim to capture more information about how the program is structured and behaves.

Hybrid / multi-feature RGB encodings. A further extension of RGB-based methods combines different types of features into a single image representation. In these hybrid encodings, each channel represents a different aspect of the malware sample, such as statistical properties, structural patterns, or behavioral indicators. Pinhero et al. [PLP+21] present a comparative study of encoding strategies, including RGB, grayscale, and Markov-based representations, and demonstrate that combining multiple feature extraction techniques can improve performance. Specifically, approaches that integrate complementary information sources tend to outperform those relying on a single representation.

Strengths and Limitations. RGB-based encodings allow more information to be represented by using multiple channels. This makes it possible to combine different types of features, such as byte-level structure, opcode information, or statistical properties, within a single image. This can improve classification performance by capturing richer structural patterns [LWSW24, PLP+21]. However, RGB representations still impose a two-dimensional structure on data that is naturally sequential. This does not reflect the true organization of executable code and may require resizing or truncation, similar to grayscale encodings. Multi-channel designs can increase preprocessing complexity, particularly when incorporating higher-level features such as opcode sequences or API calls.

3.1.3 Markov-Based Encodings

Markov-based encodings model statistical relationships using transition probabilities between elements of a sequence (bytes/opcodes). Precisely, the sequence can be modeled as a first-order Markov process, where the probability of transitioning from one element

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

i to another element j is defined as:

$$P_{i,j} = P(j \mid i) = \frac{f(i,j)}{\sum_k f(i,k)} \tag{3.1}$$

where $f(i,j)$ denotes the frequency of observing element j immediately after element i in the sequence, and the denominator represents the total number of transitions originating from state i. The resulting transition matrix can be mapped to an image representation, with the specific mapping depending on the encoding strategy and described in detail in Section 4.2. This approach is based on the assumption that elements in executable files exhibit locality, meaning that the occurrence of a value is influenced by previous elements [BAC+26]. The Markov-based encodings capture how elements relate to each other within the sequence rather than their absolute values.

Markov from raw bytes. One approach is to construct transition probability matrices directly from raw byte sequences. In this case, each element corresponds to a byte value in the range 0 to 255, resulting in 256 possible states and a 256×256 transition matrix, where each entry represents the probability of a transition between two byte values [BAC+26]. For example, Yuan et al. [YWL+20] construct Markov images from raw byte sequences and map them into fixed-size matrix representations, thereby avoiding the need for truncation or resizing while capturing global statistical patterns across the entire binary.

Markov from opcode/assembly. More advanced approaches derive transition matrices from higher-level representations such as opcode sequences. In this case, the encoding captures dependencies between instructions instead of raw byte values. For example, Deng et al. [DGS+23] construct Markov-based representations from assembly instruction sequences extracted from the code section of executable files. So instead of directly modeling opcode transitions, the method transforms assembly instructions into sequences of alphanumeric symbols, reducing the state space to a fixed set of 62 symbols and enabling compact transition matrices. These are combined into a three-channel image capturing (i) general transitions, (ii) opcode dependencies, and (iii) statistical properties of instruction structure.

Strengths and Limitations. Markov-based encodings provide a more structured representation by explicitly modeling transition relationships between elements of a sequence. As a result, they capture global statistical dependencies that are not preserved in raw byte mappings. They also allow relationships to be learned independently of position, which may improve classification performance compared to simpler encodings such as grayscale and RGB [PLP+21]. In addition, the resulting transition matrices are typically of fixed size, allowing the full sequence to be represented without truncation. However, Markov-based representations rely on aggregated transition statistics and therefore provide only a simplified view of program structure, without explicitly capturing detailed semantic information such as instruction context or operand-level behavior. Consequently, important structural or semantic characteristics of the executable may be lost in the transformation process [DGS+23].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.1.4 Hashing-Based Encodings

Hashing-based encodings rely on locality-sensitive hashing (LSH), which aims to preserve similarity relationships between inputs by mapping elements that are close to each other to the same hash value with a higher probability than elements that are far apart [IM98]. Early work by Broder et al. [Bro97] introduced the use of shingles and MinHash to represent documents as sets of features, enabling similarity estimation from fixed-size sketches. As a specific hashing technique for sets, MinHash generates compact signatures such that similar sets have a higher probability of producing the same hash value. In the general context of similarity estimation, Charikar [Cha02] formalized locality-sensitive hashing as a family of hash functions with the property that the probability of hash collisions reflects the similarity between inputs. These principles are directly adopted in malware visualization, where executable files are represented as sets of extracted features, such as opcode sequences or n-grams, which are then transformed into fixed-size representations using hashing techniques and interpreted as images for classification [BAC+26].

Hashing from opcode / higher-level features. Most existing work applies hashing-based encodings to opcode sequences derived from static analysis. For instance, Ni et al. [NQZ18] propose the Malware Classification using SimHash and CNN (MCSC) approach, in which opcode sequences extracted from disassembled malware are encoded with SimHash, a locality-sensitive hashing technique that maps similar inputs to similar binary hash values, and then converted into grayscale images for CNN-based classification. In their method, each opcode is hashed individually, and the resulting hash values are combined bit-wise to produce a fixed-length binary SimHash value. This value is then mapped to pixel values to generate an image representation. Similarly, Sun and Qian [SQ21] propose the RMVC method, whose name combines the initials of its four main components: RNN, MinHash, visualization, and CNN. In their approach, opcode sequences are processed by a bidirectional RNN using a sliding window, where the surrounding opcodes are used to predict an opcode within the window. Repeating this process produces a predictive opcode sequence that captures information learned from the surrounding opcode context. MinHash is then applied to produce fixed-length signatures that can be mapped into images of uniform size.

A key characteristic of these approaches is that hashing is applied after feature extraction rather than directly to raw binary data. In both cases, hashing operates on opcode-level representations obtained through disassembly rather than directly on the raw byte sequence.

Strengths and Limitations. Hashing-based encodings produce compact, fixed-size representations that enable consistent image construction without requiring resizing or truncation of the original data [SQ21]. In addition, the encodings are reported to achieve strong classification performance while maintaining low computational cost [NQZ18]. However, this representation might introduce information loss due to hash collisions and does not preserve explicit structural or sequential relationships, limiting its ability to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

capture fine-grained program structure.

3.1.5 Entropy-Based Encodings

Entropy-based approaches focus on the distribution and randomness of byte values in malware samples. Entropy, derived from information theory, is commonly used in malware analysis to identify patterns associated with obfuscation techniques such as packing or encryption [BAC+26]. Formally, the entropy of a byte distribution within a given segment can be defined using the Shannon entropy as:

$$H(X) = -\sum_{i=0}^{255} p(x_i) \log p(x_i), \tag{3.2}$$

where $p(x_i)$ denotes the probability of observing byte value i within the segment. Higher entropy values indicate a more uniform distribution of byte values, which is often associated with encrypted or packed regions [VNN+20].

Entropy from raw bytes. In most approaches, entropy is computed by dividing binaries into segments and calculating the entropy for each segment. The resulting values can then be transformed into visual representations, for example, by mapping them to pixel intensities to form images or by constructing entropy graphs, in which the entropy values of consecutive binary segments are plotted to represent changes in entropy across the file [BAC+26, XLCL20]. For example, Xiao et al. [XLCL20] construct entropy graphs by computing structural entropy over binary segments and transforming the resulting values into visual representations for classification using deep learning models. In addition, entropy information can be incorporated into more complex visual encodings, where it is combined with other feature types, such as grayscale and SimHash representations, within multi-feature image construction pipelines [JAV+24, FXW+18].

Strengths and Limitations. Entropy-based encodings capture general structural patterns of executable files by modeling the distribution of byte values. This makes them particularly effective for identifying patterns associated with packing or encryption [VNN+20]. However, entropy-based representations provide only an abstract summary of the data, as they capture value distributions within a region while ignoring sequential order. When visualized as images, this produces an artificial spatial structure that does not reflect the actual organization of the executable. In addition, entropy-based representations primarily capture variations in byte-value distributions and therefore provide limited information about the underlying program semantics.

3.1.6 Image Construction and Layout Strategies

While the previous subsections focused on five frequent encoding categories and the type of information represented in malware images, prior work also considers how this information is arranged in two-dimensional space. Image construction and layout strategies determine how one-dimensional sequences, such as byte streams or opcode sequences, are mapped

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

to two-dimensional pixel grids. In the literature, image construction is most commonly discussed in the context of byte-based encodings, in which raw binary data is directly transformed into image representations [BAC+26, OS22]. More advanced layout strategies have also been proposed, including approaches in which feature representations are constructed prior to image formation and subsequently arranged in two-dimensional space [VNN+20].

A widely used approach is the *line-by-line* (or *carriage return*) method, in which pixels are arranged sequentially until a predefined row width is reached, after which the next pixel is placed at the beginning of the following row [NKJM11, LZKN22, CJKK17, LLLZ20]. This approach is simple and computationally efficient but neglects locality, as bytes that are close in the original sequence may be separated across different rows [BAC+26].

To address this limitation, alternative layout strategies have been proposed. In particular, space-filling curves (SFCs) define a traversal over a two-dimensional grid such that consecutive elements in a sequence are mapped along a continuous path through the grid, thereby keeping consecutive elements spatially adjacent. Compared to row-wise layouts, this improves the preservation of local relationships within the data. In malware visualization, SFCs have been used to preserve such relationships when mapping binary data or derived features into two-dimensional representations [OS22]. These properties can be described in terms of locality, stability, and order adjacency. Locality ensures that neighboring elements in the sequence are mapped to spatially nearby regions, stability implies that small changes in the input result in limited changes in the layout, and order adjacency preserves relationships between consecutive elements [Wat05]. O'Shaughnessy and Sheridan [OS22] propose a malware visualization framework based on space-filling curves, in which binary data is mapped to color-coded pixels and arranged using space-filling curves such as Hilbert and Z-order. Similarly, Vu et al. [VNN+20] incorporate a Hilbert curve into their HIT4Mal framework after feature extraction, where entropy and semantic features are encoded into image channels before defining the spatial arrangement of pixels.

It is important to note that while alternative layout strategies such as space-filling curves have been proposed to improve locality preservation in malware image representations, they introduce an additional design dimension beyond the choice of encoding method. As a result, variations in classification performance may depend not only on the information encoded in the image, but also on how this information is spatially arranged. However, to maintain a focused experimental scope and isolate the influence of the underlying encoding strategy, layout strategies are not considered as an independent comparison dimension in this thesis.

3.1.7 Comparison and Discussion of Encoding Strategies

The reviewed encoding strategies can be compared along several dimensions, including their underlying data sources, levels of abstraction, preprocessing requirements, and the type of relationships they capture. Going beyond focusing on individual approaches,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

this section summarizes the main characteristics and trade-offs across the five encoding categories introduced.

Based on the reviewed literature, Table 3.1 provides a comparative synthesis developed in this thesis of the five encoding categories in terms of their underlying data sources, levels of abstraction, and the types of relationships they capture.

Table 3.1: Comparison of malware-to-image encoding strategies across key dimensions

Image Type	**Reference**	**Data Source**	**Abstraction**	**Traceability**	**Relationships Captured**
Grayscale	[NKJM11] [LZKN22] [CJKK17] [LLLZ20]	Bytes	Low	Direct byte correspondence	Local spatial adjacency
RGB (simple)	[WZLZ19] [BM24] [DGS+23]	Bytes	Low–Medium	Near-direct byte correspondence	Local grouped-byte patterns
RGB (multi)	[JKR+21] [LWSW24] [PLP+21]	Bytes + features	Medium–High	Channel-dependent	Multi-feature channel relationships
Markov	[YWL+20] [DGS+23] [PLP+21]	Bytes/opcode	Medium–High	Indirect	Transition and statistical dependencies
Hashing	[NQZ18] [SQ21]	Opcode/features	High	Indirect	Feature similarity
Entropy	[XLCL20] [JAV+24] [FXW+18]	Bytes	Medium	Segment-level	Byte-distribution and randomness patterns

The columns summarize the underlying data source, abstraction level, traceability, and the primary type of relationships captured by each encoding strategy. Traceability describes how directly the resulting representation can be related back to information in the underlying executable.

A primary distinction between encoding strategies lies in their level of abstraction and preprocessing complexity. Grayscale and simple RGB encodings typically operate directly on raw byte sequences and therefore preserve low-level structural information of executable files while requiring minimal preprocessing [NKJM11, WZLZ19]. In addition, such representations are largely platform-independent, as they do not rely on architecture-specific disassembly [RCS23]. In contrast, Markov-, entropy-, and hashing-based encodings introduce increasing levels of abstraction. Markov-based approaches model transition probabilities between elements [YWL+20], entropy-based methods summarize local byte distributions [XLCL20], and hashing-based techniques compress extracted features into compact representations [NQZ18]. While higher levels of abstraction may improve the ability to capture discriminative patterns, they also reduce the direct correspondence between the image representation and the original binary. In addition, approaches that rely on opcode sequences or engineered features require disassembly and may be sensitive to obfuscation [NQZ18, SQ21]. More expressive encodings may introduce additional overhead during image construction, particularly for multi-channel or structured

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

representations.

Encoding strategies also differ in how they handle variable-length binaries. Grayscale and RGB representations often require resizing, truncation, or padding to produce fixed-size inputs compatible with machine learning models [LZKN22, BAC+26]. This may lead to information loss or distortion of structural relationships within the data. In contrast, Markov-based and hashing-based approaches naturally produce fixed-size representations through transition matrices or hash signatures, allowing the entire sequence to be incorporated without explicit truncation [YWL+20, NQZ18]. Entropy-based representations similarly provide compact summaries by aggregating information over segments, although at the cost of reduced granularity [XLCL20].

Furthermore, the different encoding strategies capture distinct types of relationships within malware data. Grayscale and basic RGB encodings primarily preserve local relationships based on spatial adjacency in the constructed image. More advanced RGB approaches extend this by combining multiple feature types across channels, increasing representational richness [LWSW24]. Markov-based encodings explicitly capture transition relationships between elements, enabling the modeling of global statistical dependencies [YWL+20, DGS+23]. In contrast, entropy-based methods focus on the distribution of byte values within regions, capturing coarse-grained structural patterns and highlighting statistical irregularities often associated with packing or encryption [XLCL20]. Hashing-based approaches, on the other hand, emphasize similarity between feature sets rather than preserving explicit structural relationships [NQZ18].

Implications for Transformer-Based Models

Overall, the literature suggests that no single encoding strategy is universally superior. Simpler byte-based encodings require minimal preprocessing, but may fail to capture higher-level structural or semantic information. More structured approaches, including multi-channel RGB, Markov-based, and hashing-based encodings, provide richer representations, but require additional preprocessing and reduce the direct correspondence to the underlying binary. Although preprocessing requirements differ between encoding strategies, their practical relevance also depends on the computational cost they introduce relative to the overall classification pipeline. Therefore, both classification effectiveness and computational efficiency are considered when evaluating the encoding strategies in this thesis.

In addition, it can be observed that many earlier works adopt grayscale encodings as a default representation without explicitly justifying this choice compared to alternatives or discussing the associated trade-offs [BS23, LZKN22]. On the other hand, works that propose more specialized encoding strategies frequently report strong performance improvements over baseline methods [DGS+23, NQZ18]. However, these results are typically obtained under specific experimental settings (e.g., particular datasets, training splits, and model configurations) and may therefore not generalize across different datasets or model architectures. Consequently, encoding strategies should be evaluated alongside

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the model architecture, as their effectiveness depends on how well the model can capture local and global patterns in the data.

Given that this thesis focuses on Vision Transformer-based models (as described in 2.4.5), this interaction becomes particularly relevant. Since ViTs partition images into fixed-size patches and process them as token sequences, the chosen malware-to-image encoding directly influences how structural information is distributed across patches and how relationships between different regions are modeled.

The level of abstraction introduced by an encoding also has implications for interpretability. Representations that remain closer to the original byte sequence preserve a more direct correspondence between image regions and the underlying executable, thereby supporting interpretation. In contrast, more abstract representations weaken this correspondence, making it more difficult to relate model outputs back to program structure.

To summarize, the comparison of existing malware-to-image encoding strategies reveals several open challenges. First, many approaches are evaluated in specific model settings, often relying on traditional machine learning methods or CNN-based architectures, while the influence of different encoding strategies in Vision Transformer-based settings remains less explored. Second, literature primarily focuses on classification performance, while computational costs and preprocessing requirements introduced by different encodings receive comparatively less attention [BAC+26]. Finally, different encoding strategies introduce varying levels of abstraction, which affects the correspondence between the generated image representation and the underlying executable structure.

3.2 Vision Transformer–Based Malware Classification Approaches

Vision Transformers (ViTs), introduced in Section 2.4.5, have recently been adopted for malware classification, complementing image-based approaches that have traditionally been dominated by convolutional neural networks (CNNs) [RCS23]. Building on malware-to-image representations, transformer-based models aim to capture global dependencies within visualized executable data through self-attention mechanisms.

Compared to the large body of CNN-based approaches, the corresponding literature on Vision Transformers is relatively limited and still evolving; it mainly focuses on adapting ViT architectures to malware image classification, often emphasizing performance improvements while relying on fixed encoding strategies. In addition to standard ViT models, recent work also explores enhanced transformer architectures and hybrid CNN–Transformer approaches that combine local feature extraction with global attention mechanisms.

In the following, transformer-based approaches proposed for image-based malware classification are reviewed and categorized according to their architectural characteristics. We selected papers based on the following criteria: (i) the use of transformer-based

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

architectures, (ii) the application to malware detection or malware family classification tasks, and (iii) the use of image-based representations of executable code. Studies focusing solely on CNN architectures, non-image representations, or tasks other than malware classification were excluded.

3.2.1 Standard Vision Transformer Approaches

Some works treat Vision Transformers largely as direct alternatives to convolutional neural networks within malware classification pipelines, without introducing major architectural modifications. Their main goal is to prove that vision transformer-based models can achieve competitive or superior performance compared to established CNN-based methods.

Park et al. [PC22] were among the first to apply Vision Transformers to malware classification in 2022, introducing an additional patch-level encoding to enhance local feature representation and demonstrating improved performance compared to CNN-based approaches on the Microsoft Malware Dataset (BIG2015)[1]. These findings are supported by later studies. Katar and Yıldırım [KY24] apply a fine-tuned Vision Transformer to grayscale malware images on the MaleVis dataset[2], while Ben Abdel Ouahab et al. [BaoEB23] show that standard ViT models outperform CNN baselines on the Malimg dataset[3]. In addition, Lu et al. [LZKN22] propose ImgConvAttn, a Vision Transformer–based model that outperforms CNN baselines by incorporating a convolutional layer for patch extraction. This approach represents a practical adaptation of the ViT architecture, introducing local inductive bias without fully adopting a hybrid CNN–Transformer design.

In addition to performance-oriented evaluations, some works explore practical adaptations of ViTs. Ravi et al. [RCS23] propose a lightweight Vision Transformer optimized for edge devices, demonstrating that ViTs can be adapted to resource-constrained environments without relying on a hybrid CNN–Transformer architecture. Furthermore, Seneviratne et al. [SSRK22] analyze self-supervised training strategies based on masked autoencoders. They show that improved representation learning can lead to strong performance without modifying the underlying ViT architecture.

3.2.2 Enhanced Vision Transformer Architectures

Beyond standard Vision Transformer applications, recent work focuses on enhancing the ViT architecture to better capture structural characteristics of malware representations. While the reduced inductive bias of standard ViTs enables flexible modeling of global dependencies, it provides less explicit support for learning local patterns than convolutional architectures. Recent approaches therefore introduce architectural modifications that incorporate local or multi-scale feature modeling while retaining the global modeling capabilities of self-attention.

[1] https://www.kaggle.com/datasets/muhammad4hmed/malwaremicrosoftbig
[2] https://web.cs.hacettepe.edu.tr/~selman/malevis/
[3] https://www.kaggle.com/datasets/ikrambenabd/malimg-original

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

One line of work incorporates mechanisms that explicitly combine local and global feature modeling. For example, Belal and Sundaram [BS23] propose a butterfly construction-based Vision Transformer (B-ViT). The architecture divides the image patches into segments that are processed in parallel by local transformer encoders to capture local spatial representations. The resulting local representations are subsequently combined and processed by a global transformer encoder to capture information across the entire image, which improves classification performance compared to standard ViT variants. Similarly, Zhao and Gan [ZG24] introduce the Malware Detector Based on Enhanced Vision Transformer (MDViT), a hierarchical transformer architecture based on the Swin Transformer and its shifted-window attention mechanism, enabling the model to capture local and global patterns. This hierarchical design resembles convolutional neural networks, as it processes features at multiple scales and progressively aggregates local information into higher-level representations. Additionally, deformable attention is used to identify important regions within each image by dynamically selecting relevant spatial locations for attention, allowing the model to focus on informative features while reducing attention to less relevant regions. Cui et al. [CWQ+25] similarly propose the Hybrid Enhanced Vision Transformer based on Regional-Local Attention (HERL-ViT), a hierarchical Vision Transformer architecture based on a pyramid structure that enables multi-scale feature extraction. The model introduces a region-to-local attention mechanism that combines local attention for fine-grained feature modeling with regional attention to capture global context, while improving computational efficiency by reducing the cost of global self-attention.

Beyond architectural modifications, other approaches focus on enhancing the representation learning capacity of Vision Transformers. Jeong et al. [JLKC25] propose MalFormer, which combines multi-scale embedding and long–short distance attention to model local and global dependencies, while adaptive token masking and Sharpness-Aware Minimization improve generalization and robustness against obfuscated malware.

3.2.3 Hybrid CNN–Transformer Approaches

Motivated by the strong performance of Vision Transformers in malware classification, recent work explores hybrid approaches that combine convolutional neural networks with transformer-based models. These methods aim to leverage the strengths of CNNs and ViTs, where CNNs are effective at capturing local spatial patterns through inductive biases such as locality and translation invariance, while transformers excel at modeling long-range dependencies through self-attention mechanisms.

A common design strategy is to employ sequential hybrid pipelines, in which convolutional layers are used for early-stage feature extraction and transformer modules are applied subsequently for global feature modeling. For example, Chen et al. [CLH+22] propose a hybrid framework that integrates convolutional processing with a Vision Transformer architecture, replacing standard self-attention with a Lambda layer to reduce computational complexity. In addition, the model introduces a line-based patch embedding strategy tailored to malware representations, enabling more efficient modeling of semantic

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

relationships within opcode-derived images. Similarly, Bavishi and Modi [BM24] introduce a two-stage hybrid approach combining DenseNet for binary classification with a lightweight Vision Transformer (LeViT) for multiclass classification, achieving both high accuracy and fast inference.

Beyond sequential designs, more integrated hybrid architectures aim to jointly model local and global features within a unified framework. For example, Yu et al. [YCA+25] propose a hybrid CNN–Transformer model that combines multi-scale convolutional feature extraction with transformer-based processing. Their approach introduces a row-patch Transformer design that processes feature maps as sequential structures, enabling modeling of long-range dependencies while preserving local feature relationships. In addition, the model incorporates a lossless encoding strategy, in which each byte value is mapped to a corresponding grayscale pixel and the image dimensions are adapted to the bytecode length, thereby preserving the complete byte sequence without requiring cropping or resizing and reducing information loss during malware-to-image transformation.

3.2.4 Synthesis and Limitations

This subsection summarizes the reviewed Vision Transformer-based approaches and identifies key trends and limitations. Table 3.2 provides a structured overview of the discussed methods, including their architectural characteristics, encoding strategies, datasets, explainability approaches, and main contributions. Based on this comparison, several consistent patterns emerge across existing work.

Table 3.2: Overview of transformer-based malware classification approaches using image representations

Reference	Type	Architecture	Encoding	Dataset	Explainability	Key Contribution
[BaoEB23]	Standard ViT	ViT	Grayscale (raw bytes)	Malimg	None	ViT vs CNN comparison
[KY24]	Standard ViT	ViT-B/16	Grayscale (raw bytes)	MaleVis	None	Fine-tuned baseline
[PC22]	Standard ViT	ViT + patch encoding	Grayscale (raw bytes)	BIG2015	Limited	Patch-level enhancement
[RCS23]	Standard ViT	Lightweight ViT	RGB (raw bytes)	IMG_DS	None	Efficient edge model
[SSRK22]	Standard ViT	ViT (MAE)	Grayscale (raw bytes)	MalNet	None	Self-supervised learning
[LZKN22]	Adapted ViT	ViT + conv patch	Grayscale (frequency-based)	BIG2015 + BODMAS	Limited	Conv-based patch embedding
[BS23]	Enhanced ViT	B-ViT	Grayscale (raw bytes)	Malimg; BIG2015	None	Global–local attention
[ZG24]	Enhanced ViT	MDViT	RGB (raw bytes)	Malimg	None	Hierarchical attention
[CWQ+25]	Enhanced ViT	HERL-ViT	Grayscale (raw bytes)	BIG2015	None	Efficient local attention
[JLKC25]	Enhanced ViT	MalFormer	Grayscale (raw bytes)	Multiple	Limited	Multi-scale embedding and long–short attention
[CLH+22]	Hybrid CNN–Transf.	ViT + CNN	Grayscale (feature-based)	VirusShare; BIG2015	None	Lambda attention for efficient modeling
[BM24]	Hybrid CNN–Transf.	DenseNet + LeViT	RGB (raw bytes)	MaleVis	None	Efficient two-stage CNN–ViT pipeline
[YCA+25]	Hybrid CNN–Transf.	CNN + ViT	Grayscale (lossless encoding)	BIG2015; BDCI-21	None	Lossless encoding

Explainability: None = no explainability method; Limited = explainability method applied primarily for visual inspection, without further systematic qualitative or quantitative analysis of the resulting explanations (e.g., attention/activation maps).

One observation is that most approaches rely on relatively simple encoding strategies, like grayscale or RGB representations derived from raw byte sequences. We can conclude that the choice of encoding is rarely treated as a *primary* design variable, despite the availability of more advanced encoding techniques, as discussed in Section 3.1. In addition, the image encoding is often adopted without systematic evaluation or explicit

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

justification. One exception is the work of Yu et al. [YCA+25], which explicitly accounts for the encoding process in the model design by proposing a lossless encoding strategy that preserves structural and semantic information. However, it is important to note that their approach combines this encoding with a complex hybrid CNN-Transformer architecture. This makes it difficult to isolate the contribution of the encoding itself. In addition, no comparison with alternative encoding strategies is provided, and therefore, it remains unclear to what extent the reported performance improvements can be attributed to the encoding method or the model architecture. Overall, the interaction between different image encoding choices and transformer-based architectures remains unexplored.

Another aspect that stands out is the literature's strong emphasis on architectural improvements, with the primary objective of achieving higher classification performance than CNN-based approaches. Among the reviewed works, Vision Transformer–based models consistently demonstrate competitive or superior performance [BaoEB23, KY24, CLH+22]. This suggests an increasing relevance of ViT-based approaches in image-based malware classification, gradually complementing or replacing purely convolutional methods. A notable trend is the development of architectures that explicitly combine local and global feature modeling, for example, through hierarchical transformers or hybrid CNN–Transformer designs.

Table 3.3: Comparison of evaluation practices, dataset splits, and efficiency considerations in transformer-based malware classification approaches

Reference	Dataset	Split	OS	Acc	Prec	Rec	F1	Other	Efficiency
[YCA+25]	BIG2015	60/40 (no val)	No	96.7%	–	–	95.2%	FPR	–
[YCA+25]	BDCI-21	60/40 (no val)	No	99.7%	–	–	99.7%	FPR	–
[BM24]	MaleVis	70/30 (no test)	No	96.6%	–	–	–	–	Inference speed
[CLH+22]	VirusShare	80/20 (no test)	No	91.00%	90.26%	90.64%	0.9044	–	Train time/epoch
[CLH+22]	BIG2015	80/20 (no test)	No	99.30%	98.29%	97.45%	0.9785	–	Train time/epoch
[JLKC25]	BIG2015	80/10/10	No	97.33%	95.41%	92.27%	0.9349	–	Latency & Throughput
[JLKC25]	Malimg	80/10/10	No	99.29%	99.32%	99.29%	0.9929	–	Latency & Throughput
[JLKC25]	MalNet-Tiny	80/10/10	No	88.73%	89.08%	88.73%	0.8873	–	Latency & Throughput
[CWQ+25]	BIG2015	80/10/10	No	99.2%	99.0%	99.0%	0.992	AUROC, Loss	Inference delay
[ZG24]	Malimg	Not reported	Yes	99.35%	98.7%	98.5%	98.6%	–	–
[BS23]	BIG2015	80/20	Yes	99.49%	99.77%	99.77%	99.77%	–	Parallel efficiency metrics
[LZKN22]	BIG2015	80/10/10	No	96.35%	–	–	96.33%	–	Latency
[LZKN22]	BODMAS-11	80/8/12	No	95.02%	–	–	94.96%	–	Latency
[LZKN22]	BODMAS-49	80/8/12	No	91.23%	–	–	90.13%	–	Latency
[SSRK22]	MalNet	Not reported	No	80.2%	56.8%	46.1%	0.491	AUROC	Inference time
[RCS23]	IMG_DS	Not reported	Yes	97.66%	98.0%	98.0%	0.98	FPR, FNR	Latency, speedup, energy
[PC22]	BIG2015	10-fold CV	No	96.83%	–	–	–	Std.	–
[KY24]	MaleVis	80/10/10	No	98.80%	–	–	–	Specificity	–
[BaoEB23]	Malimg	Not reported	No	98.38%	–	–	–	Loss	–

OS = Oversampling applied during training.

Furthermore, we also observed that the evaluation practices lack standardization, which limits the comparability of reported results, a trend that has also been observed in broader surveys of malware classification approaches [BAC+26]. While many papers report very high classification accuracy, often exceeding 98–99%, the selection and reporting of evaluation metrics vary considerably across works (see Table 3.3). Some studies report a comprehensive set of metrics, including Precision, Recall, and F1-score, whereas others rely primarily on accuracy or provide only a limited set of additional metrics.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

In addition, experimental setups are often described inconsistently, with variations in dataset splits (e.g., train/validation/test ratios, cross-validation, or unspecified splits) and limited reporting on preprocessing or sampling strategies. For example, Belal and Sundaram [BS23] report superior performance of their butterfly vision transformer (B-ViT) model compared to prior CNN- and ViT-based approaches. In particular, their method achieves one of the highest reported accuracies on the BIG2015 dataset, at 99.49%. However, a closer examination of their methodology reveals additional factors that may influence the reported results. In particular, the authors apply both data augmentation and class-balancing strategies to address dataset imbalance, which can affect performance. Moreover, while the paper reports classification performance in terms of test accuracy, an inspection of additional materials provided by the authors suggests that the reported results may correspond to **validation performance** rather than evaluation on a strictly held-out test set. This lack of clarity further complicates the direct comparison with other approaches, as differences in evaluation strategies can lead to unjustified optimistic performance estimates. More generally, the absence of standardized evaluation practices limits comparability across studies and raises concerns about the robustness and generalization of reported results.

Beyond evaluation practices, several works highlight a trade-off between model performance and computational efficiency when comparing Vision Transformers to CNN-based approaches. In particular, ViT-based models are generally more computationally intensive, especially for larger input images, and often require longer training times due to the complexity of the self-attention mechanism [BS23, BaoEB23]. However, only a limited number of studies explicitly address this limitation. For example, Ravi et al. [RCS23] propose a lightweight Vision Transformer (ViT4Mal) optimized for edge deployment, demonstrating significant efficiency improvements compared to standard ViT models. While some studies additionally report efficiency-related metrics (e.g., latency or inference time), these aspects generally remain secondary to classification performance in most existing work [JLKC25, SSRK22].

Finally, we observed that most existing approaches lack meaningful explainability and traceability. While Vision Transformers inherently rely on attention mechanisms, these are rarely leveraged to provide interpretable insights into model decisions. As summarized in Table 3.2, the majority of works do not include any explainability method, and those that do typically rely on basic visual inspection techniques, such as attention or activation maps, without including metrics [PC22, LZKN22]. However, these approaches do not establish a clear link between model predictions and the underlying code semantics and, therefore, fail to provide meaningful semantic interpretation. For example, Jeong et al. [JLKC25] use t-SNE visualizations and Grad-CAM-based attention maps to analyze learned feature representations and highlight influential regions in the image. However, the models still remain effectively black-box systems, making it difficult to understand which parts of the malware contribute to a given family prediction, a limitation that is particularly important in practical malware analysis.

To summarize, existing ViT–based malware classification approaches demonstrate that

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

transformer architectures can achieve competitive or superior performance compared to established CNN-based methods. However, several important limitations remain. First, most studies evaluate only a single encoding strategy and focus primarily on architectural improvements, leaving the impact of representation design on model effectiveness and efficiency insufficiently understood. Second, heterogeneous evaluation practices limit the comparability of reported results across studies. Third, explainability is rarely considered, and the few studies that include it rely primarily on visualizations without providing a systematic qualitative or quantitative analysis of the explanations or establishing a meaningful link to the underlying program structures.

3.3 Vision Transformer Explainability Methods and Metrics

Explainability methods for Vision Transformers aim to provide insight into how these models make classification decisions based on image representations. In the context of malware analysis, such methods are particularly important, as analysts require transparency to understand and validate model predictions, as discussed in Section 2.5. However, explainability remains largely underexplored in existing Vision Transformer-based malware classification approaches (see Section 3.2.4). For this reason, we consider explainability methods proposed for Vision Transformers in the broader computer vision literature. These methods provide a more systematic basis for evaluating whether explanations are faithful, stable, computationally efficient, and whether they produce outputs that can be meaningfully related to underlying executable structures.

This section first introduces explainability methods relevant to Vision Transformers, followed by commonly used evaluation metrics. Finally, it discusses the semantic gap between model-generated explanations and their practical interpretation in malware analysis, which motivates the design requirements for the reverse-traceability framework proposed in this thesis.

3.3.1 Explainability Methods for ViTs

To categorize the various explainability methods for ViTs, we follow the taxonomy proposed by Kashefi et al. [KBSA26]. This taxonomy distinguishes five groups: (1) attribution-based methods estimate the contribution of input features to a prediction, while (2) attention-based methods leverage the self-attention mechanism of Vision Transformers to provide explanations; (3) pruning-based methods aim to improve interpretability by removing less informative model components, such as tokens or patches, whereas (4) inherently explainable methods incorporate interpretability directly into the model architecture; (5) task-specific methods, finally, are designed for particular applications such as segmentation or detection.

In line with the scope defined in Section 2.5.1, this work focuses on post-hoc explainability methods for pre-trained models that do not modify the architecture. Therefore, pruning-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

based and inherently explainable methods are not considered further. Task-specific methods are also excluded because they rely on task-dependent outputs such as segmentation masks or spatial localization, which typically require ground-truth annotations. Since image-based malware classification does not provide such outputs, these methods are not directly applicable in the considered setting.

The following sections provide a detailed overview of attribution- and attention-based methods, which constitute the primary post-hoc approaches for Vision Transformers.

1. Attribution-based Methods

Attribution-based methods aim to explain model predictions by estimating the contribution of individual input features to the final decision. Among these, Gradient-weighted Class Activation Mapping (Grad-CAM), introduced by Selvaraju et al. [SCD+17], is a commonly used technique for generating visual explanations in image-based classification tasks. Originally designed for CNNs, Grad-CAM computes the gradients of a target class score with respect to feature maps in a convolutional layer and aggregates them to obtain importance weights, which are then combined with the corresponding activations to produce a class-specific heatmap highlighting relevant image regions [SCD+17].

In image-based malware classification, Grad-CAM is one of the few explainability methods applied in combination with CNN-based models [SMFC24]. This highlights its role as a practical baseline in the domain. With the growing adoption of Vision Transformers in image classification, the method has also been extended to transformer-based architectures [KBSA26] and, consequently, transferred to malware classification settings. For example, Jeong et al. [JLKC25] employ Grad-CAM to highlight regions of malware images that influence model predictions.

In addition to Grad-CAM, several attribution-based explainability methods have been adapted for Vision Transformers, including Integrated Gradients, SHAP, and Layer-wise Relevance Propagation (LRP) [KBSA26]. Among these, Layer-wise Relevance Propagation (LRP) estimates feature relevance by redistributing prediction scores backward through the network [BBM+15].

2. Attention-based Methods

Attention-based methods use the self-attention mechanism of Vision Transformers to provide explanations. While Kashefi et al. [KBSA26] do not provide a sub-categorisation, a conceptual distinction can be made between attention-only methods and hybrid approaches that combine attention with additional signals such as gradients or relevance propagation.

Attention-only methods The simplest attention-based explanation approach is the visualization of raw attention weights, which directly reflect how tokens attend to each other within a single transformer layer. However, raw attention can become difficult to interpret in deeper layers because token representations increasingly mix contextual

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

information across the network. To address this issue, Abnar and Zuidema proposed Attention Rollout and Attention Flow [AZ20], methods later adapted to ViTs.

Attention Rollout estimates how information propagates through a transformer by recursively multiplying the aggregated attention matrices across layers while incorporating residual connections. In contrast, Attention Flow models the transformer as a flow network and computes information propagation using maximum-flow estimation. While Attention Rollout provides a computationally efficient approximation of information propagation, it relies on simplified linear assumptions that may reduce faithfulness. Attention Flow addresses some of these limitations by modeling information transfer more explicitly, although at substantially higher computational cost. Both methods derive explanations solely from attention relationships, which limits their ability to represent class-dependent feature contributions [KBSA26], motivating approaches that include additional signals.

Hybrid methods Hybrid approaches combine attention information with additional signals such as gradients or relevance propagation to generate more class-specific explanations. One representative method is *Grad-SAM*, proposed by Barkan et al. [BHC+21], which combines attention maps with gradient information to estimate the importance of token interactions for a specific prediction. The method computes gradients with respect to the attention matrices and integrates them with the corresponding attention weights to identify the input tokens that contribute most strongly to the model's output. By incorporating gradient information, Grad-SAM introduces class sensitivity into attention-based explanations while operating directly on standard transformer architectures.

Another hybrid approach, Beyond Attention, proposed by Chefer et al. [CGW21], combines attention information, gradient-based weighting, and relevance propagation to generate class-specific explanations for transformer architectures. By propagating relevance through the self-attention structure and residual connections, the method aims to produce explanations that more closely reflect the model's decision process.

Several other hybrid explainability methods have also been proposed. The Transition Attention Maps (TAM) [YLX+21] model information flows across transformer layers to derive class-discriminative relevance maps. The Beyond Intuition method [CLY+23] estimates token contributions using attention-based reasoning and gradient information to improve the faithfulness of explanations. PartialLRP [VTM+19] adapts Layer-wise Relevance Propagation to transformer architectures by focusing specifically on the relevance of attention heads within the self-attention mechanism. Compared to Beyond Attention and Grad-SAM, these approaches have received comparatively less emphasis in current Vision Transformer explainability research [KBSA26].

3.3.2 Explainability Evaluation Metrics

Unlike classification tasks, where predictions can be directly compared against known labels, explanations generally lack corresponding ground-truth labels against which they can be directly evaluated [SCMS23]. This makes the evaluation of explanations inherently

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

difficult and often subjective. Consequently, many works rely primarily on qualitative assessments rather than quantitative evaluation metrics [HWB+23], and malware analysis research is no exception [SMFC24]. Although increasing efforts have been made to establish more systematic and standardized quantitative evaluation approaches [HWB+23], there is still no universally accepted framework for assessing explanation quality, and the suitability of a particular metric depends strongly on the application domain, the underlying model architecture, and the needs of the intended audience [TK25].

To the best of our knowledge, no prior work has systematically discussed evaluation metrics for explainability in Vision Transformer-based malware classification. Therefore, this section first reviews evaluation dimensions commonly used in explainable malware analysis and then relates them to quantitative metrics employed in the broader Vision Transformer explainability literature.

Explainability Evaluation in Malware Analysis

Recent survey papers provide comprehensive overviews of the evaluation criteria most commonly used in explainable malware analysis. Tanha and Kafaie [TK25] focus on explainable Android malware detection, while Saqib et al. [SMFC24] consider explainable malware analysis more broadly across multiple platforms and data representations, including image-based approaches. Although Tanha and Kafaie focus on Android malware, the evaluation criteria they summarize are defined independently of the underlying malware platform and are therefore applicable to Windows malware classification as well. Together, these surveys provide a comprehensive overview of the quantitative criteria used to evaluate the quality of explanations in malware analysis.

Tanha and Kafaie [TK25] identify seven commonly used criteria: effectiveness, similarity, stability, robustness, efficiency, completeness, and sparsity. Many of these criteria build on the evaluation framework proposed by Warnecke et al. [WAWR20], who distinguish between general properties of explanation quality and security-specific evaluation criteria for cybersecurity applications, including completeness, stability, robustness, and efficiency. Warnecke et al. argue that these criteria are particularly important in cybersecurity, where explanations must remain complete, stable, robust, and efficient even under adversarial conditions. Saqib et al. [SMFC24] discuss closely related concepts, including fidelity, consistency, robustness, sparsity, efficiency, completeness, and expressiveness. Table 3.4 maps the terminology used in both surveys to a common set of synthesized evaluation dimensions.

Although these studies use partially different terminology, the underlying evaluation criteria are largely consistent and can be summarized as follows:

1. **Faithfulness**: Jacovi and Goldberg [JG20] define a faithful explanation as one that accurately represents the reasoning process underlying a model prediction. Although terminology is not standardized across the explainability literature, subsequent surveys note that the terms faithfulness and fidelity are often used interchangeably

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 3.4: Comparison of evaluation criteria discussed in recent malware explainability surveys.

Synthesized Evaluation Dimension	**Tanha and Kafaie (2025) [TK25]**	**Saqib et al. (2024) [SMFC24]**
Faithfulness	Effectiveness	Fidelity
Similarity	Similarity	–
Interpretability Result	–	Interpretability Result
Stability	Stability	–
Consistency	–	Consistency
Robustness	Robustness	Robustness
Sparsity	Descriptive Sparsity	Sparsity
Computational Efficiency	Efficiency	Efficiency
Completeness	Completeness	Completeness
Expressiveness	–	Expressiveness

[LACB24]. Consistent with this terminology, Saqib et al. [SMFC24] use the term fidelity and define it based on the agreement between model-generated and ground-truth explanations. Tanha and Kafaie [TK25] define effectiveness as the extent to which the features identified by an explanation are responsible for an individual prediction. An explanation is considered effective if removing the features deemed important changes the model's prediction. This perturbation-based evaluation procedure is closely related to the faithfulness evaluations discussed by Jacovi and Goldberg [JG20] and conceptually similar to Deletion AUC [PDS18], a faithfulness metric that evaluates whether features identified as important by an explanation genuinely influence model predictions [KBSA26]. Since both fidelity and effectiveness assess whether an explanation captures the factors driving a model's decision, we synthesized them into a single dimension named faithfulness.

2. **Similarity:** Measures the agreement between explanation results generated by different explainability methods for the same input. Tanha and Kafaie [TK25] describe similarity in terms of the overlap between the top-k features identified by two explanation methods.

3. **Interpretability Result**: Compares model-generated explanations with expert-provided reference explanations [SMFC24].

4. **Stability**: Measures whether repeated executions of an explanation method on the same input produce similar explanations. Tanha and Kafaie [TK25] summarize stability using the following top-k overlap formulation:

$$\text{stab}(m, T) = \frac{1}{|T|} \sum_{x_i \in T} \text{stab}(m, x_i), \qquad \text{stab}(m, x_i) = \frac{1}{\binom{|S|}{2}} \sum_{a,b \in S} \frac{|S_a \cap S_b|}{k}. \quad (3.3)$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

where T denotes the evaluated dataset, S denotes the set of explanation instances being compared for a given input, and S_a and S_b denote the sets of the top-k most relevant features obtained from two explanation instances a and b, respectively.

5. **Consistency**: Measures whether an explanation method produces similar explanations for similar inputs. Saqib et al. [SMFC24] quantify consistency as

$$K(m) = 1 - \frac{\mathrm{Var}(e(x))}{\mathrm{Var}(x)}. \tag{3.4}$$

where m denotes the explanation method, $e(x)$ the corresponding explanation, and Var represents variability.

6. **Robustness**: Assesses explanation behavior at the class level by evaluating whether explanations remain coherent across related samples while remaining distinguishable across different classes. Tanha and Kafaie [TK25] emphasize that malware samples belonging to the same family should produce similar explanations, whereas Saqib et al. [SMFC24] quantify robustness through the similarity of explanations generated for samples from the same and different classes.

7. **Sparsity**: Measures how concentrated an explanation is on a limited number of salient features [TK25, SMFC24].

8. **Computational Efficiency**: Captures the runtime required to generate explanations [TK25, SMFC24].

9. **Completeness**: Refers to whether an explanation method can generate valid, non-degenerated explanations for all evaluated inputs [TK25]. Saqib et al. [SMFC24] extend this notion by quantifying how much of the model's decision process is covered by the explanation.

10. **Expressiveness**: Describes how informative and semantically meaningful an explanation is to a human analyst [SMFC24].

It is important to note that while these evaluation dimensions provide a useful conceptual framework, not all of them are equally applicable in practice. In particular, several metrics discussed by Saqib et al. [SMFC24], including Interpretability Result, Fidelity, Sparsity, Completeness, and Expressiveness, assume the availability of ground-truth explanations against which model-generated explanations can be compared. In malware image classification, however, such pixel-level reference annotations are generally unavailable. Moreover, malware labels typically reflect high-level properties such as family membership or maliciousness rather than a single localized malicious function, making it difficult to map image regions and semantically meaningful behavior.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Explainability Evaluation in Vision Transformers

Recent work on Vision Transformer (ViT) explainability discusses quantitative metrics that relate to several evaluation dimensions identified in malware explainability research. Kashefi et al. [KBSA26] discuss several quantitative metrics used in the literature to evaluate explanations for Vision Transformer models, including perturbation-based measures such as Deletion and Insertion AUC, complexity-based measures such as Effective Complexity, runtime measurements, and annotation-based metrics such as Intersection over Union (IoU) and Pointing Game.

The metrics discussed by Kashefi et al. can be related to the synthesized evaluation dimensions introduced above as follows:

- **Faithfulness** is commonly evaluated using Deletion and Insertion AUC, Faithfulness (Correlation), Infidelity, and Sensitivity-n. These metrics quantify how strongly the model prediction changes when highly relevant image regions are perturbed and how well attribution scores correspond to observed changes in model output. More specifically, Deletion AUC measures how quickly the model's score for the target class decreases as important features are progressively removed, whereas Insertion AUC measures how quickly the target-class score increases as important features are progressively introduced. Faithfulness (Correlation) measures the correlation between feature attribution scores and changes in the model's predicted output when corresponding features are removed. Infidelity measures the discrepancy between attribution scores and changes in model output caused by input perturbations, while Sensitivity-n evaluates the correlation between the summed attribution scores of subsets of n features and their effect on the model's prediction.
- **Sparsity** is quantified by Effective Complexity, which measures how concentrated an explanation is on a limited number of image patches.
- **Computational Efficiency** is measured by the runtime required to generate an explanation.
- **Interpretability Result** is commonly evaluated using annotation-based metrics such as Intersection over Union (IoU), Pointing Game, and segmentation tests. IoU measures the overlap between the regions highlighted by an explanation and human-annotated reference regions. Pointing Game evaluates whether the most relevant point identified by an explanation falls within the annotated region of the target object. Segmentation tests assess how well the regions highlighted by an explanation correspond to human-annotated segmentation masks.

As in malware explainability research, not all of these metrics are equally applicable to malware image classification. In particular, annotation-based metrics such as IoU, Pointing Game, and segmentation tests require pixel-level reference masks, which are generally unavailable for malware images. By contrast, perturbation-based metrics such

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

as Deletion and Insertion AUC evaluate explanations solely through their effect on model predictions and therefore do not depend on manually annotated ground-truth explanations.

3.3.3 Semantic Interpretation of Explainability Results in Malware Analysis

Although the explainability methods and evaluation metrics discussed above provide important insights into how well an explanation reflects the model's internal decision process, they do not address whether the resulting explanations are actually meaningful to malware analysts. In particular, heatmap-based explanations indicate which image regions influenced the model's prediction, but they do not reveal which executable artifacts are represented by these regions or how they relate to the model's classification decision. This subsection first reviews prior work that attempts to link explainability results to program artifacts and then derives design requirements for the reverse-traceability framework proposed in this thesis.

Explainability-Guided Malware Analysis

A small number of studies have explored the use of explainability techniques to support malware analysis by linking salient image regions to executable artifacts.

Yakura et al. [YSN+19] employed a CNN with an attention mechanism to identify the byte sequences that are most influential for malware family classification. The authors motivate their approach by the goal of making manual malware analysis more efficient through the identification of important byte sequences that can serve as hints for subsequent investigation. Their qualitative analysis shows that highly salient regions can contain behaviorally relevant functions. However, the paper presents only representative examples and discusses individual functions identified within highlighted regions without systematically analyzing all executable artifacts associated with those regions. Given that explanations are derived from attention regions in a 64×64 image and a 60×60 attention map, a salient region may overlap multiple functions, partial functions, strings, or other binary artifacts. Consequently, it remains unclear whether the reported function represents the primary source of evidence for the classifier's prediction or only one of several artifacts contained within the highlighted region. Furthermore, the study does not quantify how much of a highlighted region corresponds to executable code, how many functions overlap the region, or whether the recovered functionality is causally important for the model's decision. As a result, the interpretation remains largely manual and does not provide a formal, reproducible mapping from explanations to executable structures.

Iadarola et al. [ICM+21] proposed a semi-automated explainability-driven approach for Android malware analysis. Their method applies Grad-CAM to a CNN trained on image representations of opcode sequences and maps salient image regions back to smali classes, which represent classes in the human-readable assembly-like representation of Android bytecode, producing a ranked list of potentially relevant classes for manual

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

inspection. The authors motivate their approach through two objectives: improving the interpretability of image-based malware classification and reducing analyst effort by narrowing the search space for subsequent code analysis. Their results demonstrate the practical value of combining explainability with reverse mapping to program artifacts. However, the reverse-mapping procedure is tailored to Android applications, where decompiled smali code provides explicit class boundaries that can be used to associate salient image regions with individual classes. Native Windows PE binaries do not generally provide an equivalent explicit class-level representation. Applying the same principle to Windows PE binaries would therefore require a different mapping from salient image regions to suitable binary-level structures, such as sections, functions, or instructions.

These studies demonstrate the practical value of linking explainability results to executable artifacts. However, the proposed approaches still rely partly on manual interpretation and have not been applied to Vision Transformer-based malware classifiers for Windows PE binaries. To the best of our knowledge, no prior work has proposed a systematic reverse-traceability framework for this setting. This highlights the need to consider which requirements such a framework must satisfy to support the workflows of malware analysts.

Requirements for Reverse Traceability

Votipka et al. [VRM+19] studied workflows of reverse engineers and found that they rely on recognizable cues, such as strings, API calls, constants, and control-flow patterns, to infer program functionality and identify points of interest for further investigation. Wong et al. [YWLA+21] extend these insights to malware analysis by interviewing 21 professional malware analysts from 18 organizations. Their study shows that analysts aim to understand malicious behavior, identify malware families, generate reports, and track tactics, techniques, and procedures (TTPs). To achieve these goals, analysts commonly combine dynamic analysis, static analysis, and reverse engineering, moving to more detailed code inspection when additional insight is required. The study also highlights recurring challenges, including distinguishing relevant malicious activity from benign system behavior and efficiently analyzing large numbers of samples.

Wong et al. [YWLA+21] further identified several common analysis workflows. One workflow emphasizes static analysis to identify malware families, understand sample capabilities, and generate reports. In this workflow, one participant described performing an “API scout” by extracting statically referenced Windows API functions and comparing them across samples. This example illustrates that malware analysts interpret samples in terms of executable artifacts such as API calls. Consequently, mapping highlighted image regions back to the corresponding executable artifacts can make image-based explanations more directly interpretable, actionable for malware analysts, and support the generation of analysis reports.

Overall, these works provide a basis for deriving requirements that explainability methods should meet to support malware analysis and reverse-engineering workflows. Based on

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the observed workflows and information needs reported by malware analysts and reverse engineers, three design requirements are derived for the reverse-traceability framework proposed in this thesis.

The three requirements address complementary aspects of bridging the representation mismatch between image-based explanations and malware-analysis workflows. **Traceability** concerns how an explanation can be linked to executable artifacts, **interpretability** concerns what the recovered artifacts reveal about the malware and the model prediction, and **efficiency** concerns analyst effort, specifically how explanations can guide analysts toward the most relevant artifacts and thereby reduce the amount of information that must be manually inspected.

1. **Traceability.** Malware analysts and reverse engineers understand program behavior through executable artifacts such as functions, API calls, strings, and control-flow structures identified through static analysis and reverse engineering [YWLA+21, VRM+19]. In the setting of image-based malware classification, this creates a representation mismatch: explainability methods highlight regions in the image representation, whereas analysts work with executable artifacts. Consequently, explanation regions should be traceable to the corresponding locations in the original executable, enabling them to be related to the executable artifacts examined during malware analysis and reverse engineering.

2. **Explanation interpretability.** Wong et al. [YWLA+21] show that analysts aim to understand malicious behavior and identify malware families, while Votipka et al. [VRM+19] show that reverse engineers rely on recognizable cues such as strings, API calls, and other program artifacts. Once a highlighted region has been mapped to the executable, it should correspond to recognizable analysis artifacts, such as imports, PE sections, or functions, that analysts can use to relate the model's prediction to observable malware functionality.

3. **Efficiency.** Wong et al. [YWLA+21] report that malware analysts often process and prioritize large numbers of samples under time constraints. Therefore, explanations should reduce the effort required to locate and investigate potentially relevant executable artifacts and support the prioritization of subsequent manual analysis.

To summarize, although a wide range of explainability methods and evaluation metrics have been proposed for Vision Transformers, several important limitations remain in the context of malware analysis. First, the application of these methods to malware image classification has not been systematically investigated. Second, not all existing evaluation metrics are directly applicable, as ground-truth explanation annotations are missing. Third, existing explainability methods produce relevance maps that do not directly reveal which executable artifacts influenced the model's prediction. Finally, while prior work has demonstrated the potential value of linking explanations to program artifacts, existing approaches remain limited in scope, rely partly on manual interpretation, and have not been applied to Vision Transformer-based malware classification of Windows PE binaries.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 4

Methodology and Experimental Design

This chapter provides a detailed description of the experimental design used in this thesis. The overall pipeline consists of transforming malware binaries into image representations using different encoding strategies, followed by training classification models and applying explainability methods. The primary focus is on Vision Transformer (ViT) architectures, while a convolutional neural network (CNN) is included as a baseline for comparison. To ensure comparability across experiments, training procedures and evaluation protocols are defined consistently across the evaluated models. Finally, a reverse-traceability approach is introduced to map model explanations from image regions back to underlying program structures using reverse-engineering techniques. The complete implementation of the experimental pipeline described in this chapter is available at `https://github.com/LuziaKl/malware-vit-public`.

4.1 Dataset and Data Preparation

The choice of the datasets was driven by several factors: First, the datasets must contain Windows Portable Executable (PE) files, as the focus of this work is on malware analysis in Windows environments. Second, preference is given to publicly available datasets commonly used in prior research, thereby enabling reproducibility and comparability of results. Third, the datasets must provide either precomputed representations, such as byte sequences (`.bytes`) and disassembled code (`.asm`), or raw PE files from which these representations can be derived. Finally, the datasets must provide labeled malware families rather than only binary labels (benign vs. malicious), as family-level classification is the primary task in this work.

As highlighted in prior work [BAC+26], many Windows PE malware datasets provide only byte-level representations or preprocessed image data, limiting their suitability for

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

systematic evaluation of different encoding strategies. Consequently, several datasets had to be excluded early on, leaving only a limited set of viable candidates.

Based on these criteria, initially four candidate datasets were identified: BDCI-21 [YCA+25], BIG2015 [PC22], BODMAS [YCL+21], SOREL-20M [HR20]. However, BDCI-21 and SOREL-20M were excluded from further consideration due to accessibility and suitability limitations. While BDCI-21 meets the defined selection criteria and is nominally publicly available, access is restricted in practice, as registration on the Chinese competition platform requires a Chinese mobile phone number. Obtaining such a number through the standard registration procedure for foreign nationals generally requires identity verification and in-person registration with a telecommunications provider in China. Since no alternative official access mechanism for the dataset was available, BDCI-21 was excluded from this study. In contrast, SOREL-20M provides Windows PE files but lacks malware family labels, as it is primarily designed for binary classification, making it unsuitable for this work.

4.1.1 BODMAS

The BODMAS dataset [YCL+21] is selected as the primary dataset for this work, as it provides raw PE files that allow full control over the encoding process. In addition, it enables the extraction of various types of information directly from the executable files, which is essential for the traceability pipeline. Furthermore, access to the original binaries enables manual inspection and detailed analysis using disassembly tools such as IDA.[1]

BODMAS is an open dataset designed to address key limitations of existing PE malware datasets, particularly the lack of recent samples and carefully labeled family information. It contains a total of 134,435 samples, including 57,293 malware samples and 77,142 benign samples, and covers 582 malware families. The dataset was collected over the period from August 29, 2019, to September 30, 2020, providing temporally structured data suitable for analyzing evolving malware characteristics. For each malware sample, the dataset includes the original PE binary and pre-extracted feature vectors. In addition, timestamp information is provided, indicating the first time the malware was submitted to VirusTotal[2]. It is important to note that, while the dataset includes benign samples in theory, their corresponding binaries are not publicly distributed due to copyright restrictions. Instead, only hash values are provided, which may limit their practical availability. The malware family labels were primarily generated by aggregating and interpreting detection results from multiple antivirus vendors using internally developed scripts and following approaches similar to AVClass[3]. A small portion of samples (approximately 1%) was additionally verified through manual analysis by the Yang et al. [YCL+21].

[1] https://docs.hex-rays.com/
[2] https://www.virustotal.com
[3] https://github.com/malicialab/avclass

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 4.1: BODMAS family sample distribution.

No. of samples per family	Number of families	Number of samples
1	198	198
2–50	299	3,191
51–100	28	1,930
101–500	30	7,529
501–1,000	13	10,077
1,001–2,000	7	9,743
>2,000	7	24,625
SUM	582	57,293

The dataset exhibits a strong class imbalance across malware families, as shown in Table 4.1. A large majority of families are represented by only a small number of samples: 497 families contain fewer than 50 instances, of which 198 consist of a single sample. In contrast, a small number of families account for a substantial proportion of the dataset, with just seven families containing more than 2,000 samples.

To ensure comparability with prior work, we follow the approach of Lu et al. [LZKN22] and restrict the dataset to the 11 most frequent malware families (BODMAS-11). This restriction mitigates the extreme class imbalance in the full dataset and ensures that each class has a sufficient number of samples (more than 1,000 instances) for statistically meaningful evaluation.

Data Preparation We first reconstructed the original executable structure from the disarmed Portable Executable (PE) files provided by the BODMAS dataset. This was done using the arming procedure provided by the BODMAS authors, which reverses the modifications applied to disarm the malware samples and restores their original executable structure. All subsequent processing was conducted on Ubuntu 24.04.4 LTS using static analysis only, ensuring that no malware was executed during data preparation.

From the resulting PE files, we extracted two representations, as they are the foundation for the subsequent encoding strategies. The overall data preparation process from the reconstructed PE file to the extracted representations is illustrated in Figure 4.1:

1. Byte sequence representation (`.bytes`): We generated this representation by reading each file in binary mode and writing the content as a hexdump. The output is structured into fixed-width 16-byte rows, each preceded by its corresponding file offset, preserving the positional information of the original binary.

2. Disassembled representation (`.asm`): We generated this representation using the static disassembly pipeline based on `Capstone`[4] (version 5.0.7). The output is written as a flat instruction stream in Intel syntax, including the file offset range,

[4] `https://www.capstone-engine.org/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

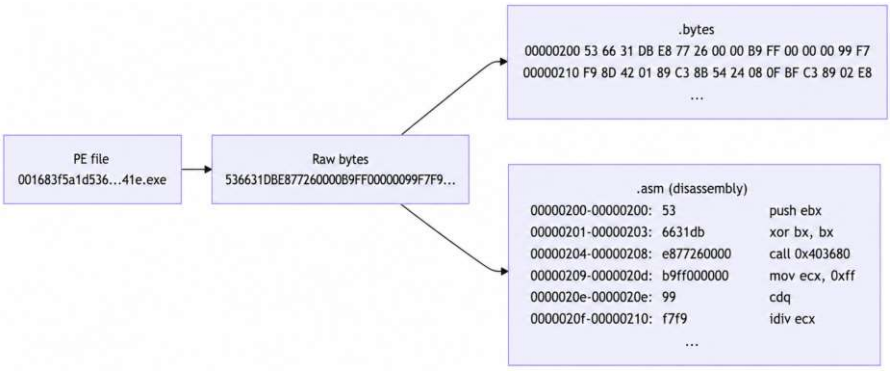


Figure 4.1: Example of the data preparation process for a Windows PE file.

opcode bytes, mnemonic, and operands for each decoded instruction. Only PE sections marked as executable were considered, i.e., sections with the execute flag set. This includes code-bearing sections such as `.text`, `CODE`, or packed executable sections such as `UPX1` when they are marked executable, while non-executable sections such as `.rsrc` are excluded. This produces a code-centric view of the binary that can be used consistently for opcode extraction and downstream image generation.

For comparison, we additionally generated an alternative ASM representation using a Ghidra-based disassembly pipeline[5] (version 12.0.4, public release). As with the Capstone-based approach, only instructions located in executable sections are included. Unlike the Capstone-based approach, which produces a uniform linear instruction stream, Ghidra employs more advanced analysis techniques that can yield a more structured and semantically richer disassembly, particularly for packed or irregular binaries. This representation is therefore included as an alternative view of the underlying program structure, rather than a direct replacement. Nevertheless, the Capstone-based output remains our primary source for downstream opcode extraction and image generation due to its consistency and suitability for large-scale processing: in our experiments, the Capstone-based pipeline required, on average, approximately 0.26 seconds per sample (median: 0.085 s), whereas the Ghidra-based pipeline required approximately 19.44 seconds per sample (median: 11.40 s). This corresponds to a runtime increase of roughly 74×, reflecting the significantly higher computational cost of Ghidra's analysis-driven disassembly. This runtime difference is particularly relevant during dataset preparation for model training, where thousands of samples must be processed. Although disassembly is performed only once per sample, the substantially lower runtime of Capstone makes it more suitable for large-scale preprocessing.

While some datasets, such as BIG2015, provide assembly files generated using IDA Pro, this approach is not adopted in this work. Large-scale automated analysis with IDA

[5] `https://github.com/NationalSecurityAgency/ghidra`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 4.2: BODMAS-11 class counts before and after filtering samples without a valid ASM representation for both disassembly backends. All samples were successfully converted to byte-level (.bytes) format.

Class	Total samples (.bytes)	ASM (Capstone)	ASM (Ghidra)
sfone	4,523	4,487	4,486
wabot	3,148	3,058	3,058
upatre	2,517	2,448	2,418
small	2,500	2,464	2,465
ganelp	2,040	2,040	2,040
wacatac	1,904	1,745	1,806
berbew	1,389	1,389	1,389
dinwod	1,383	1,383	1,383
sillyp2p	1,308	1,283	1,308
mira	1,303	1,302	1,303
ceeinject	1,040	1,017	1,039
Total	23,055	22,616	22,695

may require commercial licensing, introducing an additional dependency and cost for reproducing the preprocessing pipeline. Therefore, Capstone and Ghidra are used because they provide scriptable and consistent interfaces for scalable and reproducible analysis.

All samples were successfully converted to the byte-level representation. However, a small fraction could not be reliably disassembled using Capstone and were therefore excluded from ASM-based experiments (see Table 4.2). In total, *439* samples (approximately *1.9%* of the dataset) lacked a valid Capstone ASM representation. This is primarily due to limitations of linear disassembly when applied to packed, obfuscated, or structurally irregular binaries, where valid instruction boundaries cannot always be recovered, or decoding fails in executable regions. In contrast, the Ghidra-based pipeline failed for a slightly smaller number of samples (approximately *1.6%*), reflecting its ability to recover additional instructions through more advanced analysis techniques. The failures of the two disassembly backends were partially overlapping: 262 samples failed with both approaches, 177 failed only with Capstone, and 98 failed only with Ghidra. Thus, although Ghidra successfully disassembled more samples overall, neither set of failures was a strict subset of the other.

As the Capstone ASM-based dataset represents a reduced subset, all experiments comparing different image encodings are conducted on this subset. Accordingly, the corresponding byte-level data is filtered to the same samples to ensure a fair comparison. This may introduce a minor selection bias, as more heavily obfuscated or non-standard binaries are more likely to be excluded.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.1.2 BIG2015

In addition to BODMAS, we include the Microsoft Malware Classification Challenge (BIG2015)[6] dataset as a secondary benchmark to enable direct comparison with prior work. This dataset is widely adopted in malware classification research and has been used in numerous studies for evaluating machine learning-based approaches [BAC+26].

The dataset comprises more than 20,000 instances from nine distinct malware families [RRF+18]. However, only the 10,868 samples in the training set are labeled, while the remaining 10,873 samples are unlabeled and intended for evaluation. Since the test set labels are not publicly available, only the labeled training portion can be used for supervised classification in this work. Each malware sample is provided in two representations, consisting of a `.bytes` file containing the hexadecimal representation of the binary content and a corresponding `.asm` file generated using the IDA Pro disassembler. The latter includes not only assembly instructions but also additional metadata such as function calls, strings, and other structural information extracted from the binary [RRF+18], resulting in a mixed-content representation in which not all parts correspond to executable code.

Compared to BODMAS, BIG2015 represents an earlier malware collection, having been released in 2015, and its labeled portion contains approximately half as many samples as the BODMAS-11 subset used in this work. While these differences should be considered when comparing the datasets, the more relevant limitation for this thesis is that BIG2015 does not provide the original PE binaries. This restricts the ability to perform custom reverse engineering or extract additional information beyond the provided `.bytes` and `.asm` representations, limiting its suitability for the reverse-traceability analysis proposed in this work.

Data Preparation When using BIG2015 for assembly-based analysis, some prior work does not utilize the full set of 10,868 training samples. For example, Ni et al. [NQZ18] report experiments on a dataset of 10,805 samples, as reflected in their class distribution, while Deng et al. [DGS+23] exclude 61 samples whose corresponding `.asm` files do not contain a valid code section. However, the exact preprocessing criteria and identities of the removed samples are not fully specified, making it difficult to reproduce these results. Differences in filtering procedures, including how valid code regions are identified from the IDA-generated `.asm` files, can lead to variations in the resulting dataset.

To ensure transparency and reproducibility, we therefore define an explicit filtering criterion. Specifically, we exclude samples whose `.asm` files do not contain a recognizable code section, identified by markers such as a `.text:` label, an IDA `; Segment type: Pure code` comment, or a segment declaration of type `CODE`. We retain only samples with at least one such indicator and a non-empty opcode stream extracted from these sections; we discard those for which the opcode count within the identified code regions is zero.

[6] https://www.kaggle.com/c/malware-classification

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Applying this procedure removes 67 samples, reducing the dataset from 10,868 to 10,801 samples. The largest reductions occur in the Tracur family (751 to 735 samples) and Vundo (475 to 453 samples), while several families remain unaffected. Consequently, differences may arise when comparing results to prior work that employs alternative filtering procedures. For example, Ni et al. [NQZ18] report 751 Tracur samples (compared to our 735). Inspection of the excluded cases shows that they still contain valid instructions, but IDA exports the corresponding regions as `seg001 segment para public 'BSS' use32`. Semantically, the `BSS` segment denotes uninitialized data rather than executable code and therefore does not satisfy our requirement for explicitly identified code regions. To facilitate reproducibility and comparison with alternative filtering procedures, a CSV file containing the identifiers of all samples included after filtering is provided in the accompanying repository.

4.1.3 Dataset Splits

BODMAS For the BODMAS-11 dataset, we follow the split protocol of Lu et al. [LZKN22], using a stratified 80/8/12 split. Lu et al. provide the number of samples from each malware family assigned to the training, validation, and test sets, but not the corresponding sample identifiers. We therefore independently assign samples to the three sets while preserving these reported sample counts. The resulting distribution is shown in Table 4.3a, while an overview of all dataset variants used in the experiments is provided in Table 4.4. Encoding comparisons are conducted on a Capstone-based shared bytes+ASM subset, excluding samples without valid ASM output.

To assess the robustness of the results to different sample assignments, we generate three splits using random seeds 42, 7, and 123. The split with seed 42 is used as the reference, while seeds 7 and 123 serve for robustness checks. Because the split procedure is stratified, the class distributions are identical across seeds. However, the exact sample assignments differ considerably. For each pair of seeds, approximately 66% of the 22,616 samples are assigned to the same subset (training, validation, or test) in both splits, while approximately 34% are assigned to different subsets. The full byte-level dataset is used for experiments that do not require ASM, avoiding unnecessary exclusions due to disassembly failures. Encoding comparisons are conducted on the Capstone-based shared bytes+ASM subset to ensure that byte- and ASM-based encodings are evaluated on the same samples. For direct Capstone–Ghidra comparisons, experiments are conducted on a separate intersection subset containing only samples available in both representations, ensuring identical sample composition across the two disassembly backends. Thus, the choice of dataset variant is determined by the representation requirements of each experiment. To facilitate reproducibility, the exact sample identifiers assigned to the training, validation, and test sets for each split are provided as CSV files in the accompanying repository.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 4.3: Class-wise distributions of the reference dataset splits.

(a) BODMAS-11 shared bytes+ASM split (80/8/12).

Family	Train	Val	Test
sfone	3,590	359	538
wabot	2,446	245	367
small	1,971	197	296
upatre	1,958	196	294
ganelp	1,632	163	245
wacatac	1,396	139	210
berbew	1,111	111	167
dinwod	1,106	111	166
mira	1,042	104	156
sillyp2p	1,026	103	154
ceeinject	814	81	122
Total	18,092	1,809	2,715

(b) BIG2015 filtered split (70/10/20).

Family	Train	Val	Test
Kelihos_ver3	2,057	294	587
Lollipop	1,733	248	495
Ramnit	1,073	153	307
Obfuscator.ACY	853	122	244
Gatak	709	101	203
Tracur	515	73	147
Vundo	317	45	91
Kelihos_ver1	274	39	79
Simda	29	5	8
Total	7,560	1,080	2,161

Table 4.4: Summary of BODMAS-11 dataset variants used in the experiments.

Dataset variant	Total	Train	Val	Test
Full byte-level dataset	23,055	18,444	1,844	2,767
Shared bytes+ASM subset (Capstone)	22,616	18,092	1,809	2,715
Capstone–Ghidra intersection subset	22,518	18,014	1,801	2,703

BIG2015 For BIG2015, we use a stratified 70/10/20 partition with a fixed random seed (42). Unlike for BODMAS, where the 80/8/12 split is adopted specifically to enable comparison with Lu et al. [LZKN22], prior work on BIG2015 employs varying partitioning schemes and does not establish a common split protocol [BAC+26]. We therefore choose a 70/10/20 split to retain the majority of samples for training while reserving a comparatively large held-out test set for evaluating and comparing the different encoding and architecture configurations. Stratification preserves the observed malware-family proportions across the three subsets, making each subset representative of the overall dataset distribution and reducing differences in class composition as a potential confounding factor when comparing methods. The resulting distribution is shown in Table 4.3b.

As described previously, the split is applied to the already-filtered dataset, from which 67 samples without valid code sections were removed. All final encoding and architecture comparisons are evaluated on this same filtered split to ensure consistent comparability across methods. The exact split assignments are provided in the accompanying repository.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.2 Malware-to-Image Encoding Strategies

The selection of encodings was motivated by the analysis of prior work in Chapter 3.1, which highlights substantial variation in how executable data is transformed into image representations. While a wide range of encoding strategies exists, transformer-based malware classification predominantly relies on simple grayscale or RGB representations derived from raw byte sequences. As a result, we noted, the choice of encoding is rarely treated as a primary design variable, and alternative representation types remain largely unexplored in the context of ViTs (see Section 3.2.4). To address this limitation, the selection in this work deliberately covers different encoding paradigms, enabling a systematic comparison of how varying levels of abstraction and structural representation influence Vision Transformer models under controlled experimental conditions.

The following criteria were used to guide the selection of encoding strategies:

- **Grounding in prior work:** Each encoding is based on established approaches from the literature.
- **Conceptual clarity:** The transformation principles are sufficiently well defined to enable a reproducible implementation, with any unspecified preprocessing choices made explicit.
- **Controlled preprocessing:** Preprocessing is kept consistent and limited where possible to reduce the influence of additional feature engineering and allow differences between models to be attributed more directly to the encoding strategy.
- **Diversity of representation:** The selected encodings capture different aspects of the data, including raw content, structural relationships, and higher-level or statistical abstractions.

Based on these criteria, approaches that combine multiple feature types (e.g., byte, opcode, and API features) [JKR+21, LWSW24] or rely on heavily processed opcode features (e.g., n-gram selection and weighting) [KZK19, DAM+21] are excluded, as they introduce additional feature-selection, transformation, or combination choices that make it more difficult to isolate the influence of the encoding strategy. Rather than excluding all transformations beyond direct byte mapping, we exclude multi-stage hybrid pipelines whose performance depends on several interacting feature-engineering choices. The selected encodings each follow a clearly defined encoding approach, enabling a controlled comparison of how different representation types influence model behavior.

The selected encoding strategies include byte-based representations (grayscale and RGB), Markov-based encodings at both the byte and opcode level, as well as entropy-based and hashing-based approaches. As discussed in Section 3.1.6, where a one-dimensional sequence must be arranged into a two-dimensional image, a line-by-line layout is used,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

arranging values sequentially row by row. Encodings that inherently define a two-dimensional structure, such as transition matrices or entropy graphs, retain their method-specific layout. Alternative layout strategies are not evaluated independently, allowing the experiments to focus on the influence of the malware-to-image encoding strategy.

Within each encoding strategy, the specific methods were selected from the approaches identified in the literature review. For grayscale, the widely established raw-byte representation is used as the baseline, following the fixed-size construction of Lu et al. [LZKN22]. For RGB, the direct byte-to-RGB mapping of Wang et al. [WZLZ19] is selected instead of multi-feature RGB approaches to avoid introducing additional feature sources. For Markov-based encoding, both the byte-level approach of Yuan et al. [YWL+20] and the opcode-level approach of Deng et al. [DGS+23] are included to represent transition-based encoding at different levels of abstraction. For entropy, the entropy-graph approach of Xiao et al. [XLCL20] is selected as a direct entropy-based representation rather than a hybrid encoding combining entropy with other features. Finally, for hashing, the SimHash approach of Ni et al. [NQZ18] is selected because it directly transforms opcode information into a hash-based image representation without the additional learned feature-processing stage used by alternatives such as the RNN–MinHash approach of Sun and Qian [SQ21].

Grayscale Byte Encoding. Grayscale byte encoding is adopted as the baseline representation, as it is one of the earliest and most widely used malware-to-image encodings [NKJM11, LZKN22]. In this approach, the executable byte stream $B = (b_1, \ldots, b_n)$ is interpreted directly as pixel intensities, where each byte value $b_i \in [0, 255]$ corresponds to one grayscale pixel. Following Lu et al. [LZKN22], a fixed-width row-wise layout is used instead of adaptive image dimensions to ensure consistent input representations across samples. The byte stream is padded with zeros where necessary, reshaped into an intermediate grayscale image, and resized to the target resolution $S \times S$ using bilinear interpolation.

RGB-Based Encoding. We additionally include an RGB-based encoding derived from raw byte sequences, following Wang et al. [WZLZ19]. In this representation, consecutive bytes are grouped into RGB triples, where each triplet represents a single color pixel, with the three bytes assigned to the red, green, and blue channels, respectively. As the original work does not fully specify the image layout, padding strategy, and interpolation procedure, these implementation details are defined explicitly to ensure reproducibility and comparability with the grayscale baseline. Following the same fixed-width row-wise layout used for grayscale encoding, the byte stream is padded where necessary, reshaped into an intermediate RGB image, and resized to the target resolution $S \times S$ using bilinear interpolation.

Byte-Level Markov Encoding. We include a byte-level Markov encoding following Yuan et al. [YWL+20]. Unlike direct byte-to-pixel mappings, this approach represents a binary through a 256×256 transition matrix over consecutive byte values. For each pair of adjacent bytes (b_i, b_{i+1}), the corresponding transition count is accumulated in the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

matrix and subsequently row-normalized to obtain transition probabilities. The resulting matrix is then scaled to the grayscale intensity range $[0, 255]$, converted into an image, and resized to the target resolution $S \times S$ using bilinear interpolation where necessary. In contrast to grayscale and RGB encodings, which primarily preserve the sequential arrangement of bytes, the Markov representation captures global transition statistics within the byte stream. This encoding is included to evaluate whether such structural information improves ViT performance compared with direct byte-based representations.

Opcode-Level Markov Encoding. We include an opcode-level Markov encoding following Deng et al. [DGS+23]. This approach operates on disassembled code and constructs a three-channel (RGB) representation based on transition patterns in the instruction sequence. Compared to byte-level encodings, opcode sequences provide a higher-level abstraction more closely aligned with program structure, while the multi-channel design enables the representation of complementary information. This encoding is included to evaluate whether such structured, instruction-level representations improve ViT performance compared to byte-based approaches.

The representation is constructed from three 62×62 transition matrices, based on a character alphabet consisting of uppercase letters, lowercase letters, and digits. The first channel captures transitions between consecutive characters in the instruction stream, the second channel models transitions between the first characters of successive opcodes, and the third channel records pairs formed by the final two characters of each opcode. Each matrix is row-normalized and scaled to the pixel intensity range $[0, 255]$ before being assigned to one RGB channel and merged into a single image. The resulting image is subsequently resized to the target resolution $S \times S$ using bilinear interpolation.

Entropy-Based Encoding. Entropy-based encoding follows Xiao et al. [XLCL20], where Shannon entropy is computed over fixed-length byte segments and visualized as an entropy graph. This representation is included to assess whether statistical byte-distribution patterns, rather than explicit sequential or structural opcode relations, are sufficient for effective ViT-based malware family classification. Since the original work does not explicitly specify a numeric segment length (l), we set $l = 256$ bytes in our implementation as a fixed granularity that balances local entropy information with the resulting sequence length. The byte stream is partitioned into consecutive segments of length (l), and the Shannon entropy of each segment is computed to form an entropy sequence. This sequence is rendered as a grayscale line graph at a resolution of 300×300 pixels, following the original method of Xiao et al. [XLCL20], and subsequently resized to the target resolution $S \times S$ using bilinear interpolation.

SimHash Opcode Encoding. This encoding is a re-implementation of the MCSC approach proposed by Ni et al. [NQZ18], following the methodological description provided in the original paper. In this method, opcodes are extracted from the disassembled malware sample and treated as a single sequence. The sequence is then transformed into a fixed-length binary representation using SimHash, in which each opcode contributes

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

equally to the resulting hash vector. Unlike byte-based encodings, SimHash produces a compact, similarity-preserving representation whose dimensionality is independent of the original opcode sequence length. This encoding is included to assess whether a compressed opcode-level representation can retain sufficient information for effective ViT-based malware family classification. Ni et al. [NQZ18] evaluate multiple SimHash configurations and report that the SimHash768-z variant achieves the best overall performance. In this configuration, SHA-512 and SHA-256 are concatenated to produce a 768-bit hash. The resulting SimHash fingerprint is mapped to grayscale pixel values by assigning bit value 0 to intensity 0 and bit value 1 to intensity 255. The 768-bit fingerprint is reshaped into a native image of size 24×32 and then resized using bilinear interpolation. To ensure consistency with the remaining encodings in this work, the resulting image is resized to the common target resolution $S \times S$. While the original paper describes the SimHash construction and image generation procedure, it does not fully specify the exact preprocessing criteria used to obtain the final subset of 10,805 samples from BIG2015. We therefore used the dataset filtering defined in Section 4.1.2.

4.2.1 Image Resolution and Standardization

Existing studies adopt different image resolutions. For example, grayscale-based approaches use sizes such as 64×128 in Park et al. [PC22], 224×224 in Belal et al. [BS23], and 256×256 in Lu et al. [LZKN22], typically without explicitly justifying this choice.

In this work, the image resolution is not considered a variable of interest and is therefore fixed to 256×256 for all experiments. This ensures a consistent input representation across encoding strategies and allows performance differences to be attributed to the encoding itself rather than variations in image size. The chosen resolution aligns with prior work using the BODMAS dataset [LZKN22], facilitating comparison of results.

All images are resized to the target resolution using bilinear interpolation, which may introduce artificial intensity values, particularly for byte-based representations derived from discrete values. Depending on the encoding, inputs are represented as either single-channel (grayscale) or three-channel images (e.g., RGB or opcode-level Markov). Images are then converted to tensors, with pixel values scaled from $[0, 255] to [0, 1]$, and subsequently normalized using $\mu = 0.5$, $\sigma = 0.5$, resulting in inputs approximately in $[-1, 1]$.

It should be noted that this standardization introduces a trade-off. Encoding strategies that inherently produce lower-resolution or highly compact representations (e.g., Markov matrices or SimHash-based encodings) may be disproportionately affected by resizing, as interpolation can alter or smooth structural patterns. Consequently, while a fixed resolution improves experimental control and comparability, it may introduce a bias that favors encodings with richer spatial structure in their original form. Encoding-specific resolution optimization was not performed, as this would introduce an additional method-dependent hyperparameter and substantially expand the experimental scope beyond the primary focus on encoding strategies. Where alternative resolutions are motivated by

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

prior work, their influence is evaluated separately. This limitation is considered when interpreting the experimental results.

4.3 Model Architecture

After generating the malware image representations, classification models are trained on the resulting inputs. The Vision Transformer is used as the primary model, since the main objective of this work is to analyze the influence of different encoding strategies on Vision Transformer-based malware family classification. A ResNet-18 convolutional neural network is included as a baseline to provide a reference point against a well-established image classification approach. Both models are trained under comparable settings, as described in Section 4.4.

4.3.1 Vision Transformer Architecture

The Vision Transformer (ViT) architecture employed in this work is based on the ViT-B/16 model proposed by Dosovitskiy et al. [DBK+21].

A standard ViT backbone is deliberately chosen to isolate the effect of the encoding strategy as the primary variable under investigation. In contrast to more complex or task-specific transformer architectures proposed in recent work, advanced models often introduce additional architectural components, such as hierarchical designs or hybrid CNN–Transformer structures, which may influence performance independently of the input representation. Using a well-established architecture ensures that observed differences in performance can be more directly attributed to the encoding methods rather than to architectural variations. Preliminary experiments indicated that this configuration achieves competitive performance on the BIG2015 dataset compared to more complex architectures, supporting its suitability as a reference model (see Chapter 5.1.1).

The core architectural parameters, including the patch size, embedding dimension, number of Transformer layers, number of attention heads, and MLP hidden dimension, follow the standard ViT-B/16 configuration of Dosovitskiy et al. [DBK+21]. The input resolution is fixed according to the standardization described above, and the number of input channels is adapted to the respective malware image representation, while the number of output classes is determined by the respective dataset. The detailed model configuration is summarized in Table 4.5.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 4.5: Vision Transformer (ViT-B/16) configuration used in this work.

Parameter	Value
Backbone	ViT-Base/16
Input resolution	256×256
Input channels	1 (grayscale) or 3 (RGB)
Patch size	16×16
Patch embedding	Linear projection
Number of patches	$16 \times 16 = 256$
Sequence length	$256 + 1 = 257$ (patch tokens + CLS token)
Embedding dimension	768
Transformer layers	12
Attention heads	12
MLP hidden dimension	3072
Pooling strategy	CLS token
Positional encoding	Learnable
Classification head	Fully connected linear layer
Output classes	11 (BODMAS-11) / 9 (BIG2015)

4.3.2 Convolutional Neural Network Baseline

To contextualize the performance of the transformer-based model, a convolutional baseline is included. The CNN baseline is based on the ResNet-18 architecture [HZRS16]. ResNet-18 is selected as an established residual CNN architecture to provide a representative convolutional reference for the ViT-based experiments. The purpose of this baseline is not to identify the best-performing CNN architecture, but to assess whether the observed effects of the malware-to-image encodings are specific to the transformer architecture or also occur with a conventional convolutional model. Its relatively shallow depth of 8 layers and approximately 11.2 million parameters makes it suitable for controlled comparisons, while still retaining the benefits of residual learning.

Table 4.6: CNN baseline configuration (ResNet-18) used in this work.

Parameter	Value
Backbone	ResNet-18
Input resolution	256×256
Input channels	1 (grayscale) or 3 (RGB)
Pooling strategy	Global average pooling
Classification head	Fully connected linear layer
Output classes	11 (BODMAS-11) / 9 (BIG2015)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.4 Training Configuration

This section describes the training setup used for all experiments, ensuring a consistent configuration across models so that performance differences can be attributed primarily to the encoding strategies. The training configuration for both architectures is summarized in Table 4.7.

Table 4.7: Training configuration used for ViT and ResNet experiments, with selected hyperparameters determined through the searches described below.

Setting	ViT (ViT-B/16)	CNN (ResNet-18)
Pretraining	ImageNet-1k pretrained	ImageNet-1k pretrained
Batch size	16	16
Epochs	100	100
Optimizer	AdamW	AdamW
Learning rate	5×10^{-5}	1×10^{-4}
Weight decay	0.0	0.01 (default)
Augmentation	none	none
Seed	42	42

The models are trained using the cross-entropy loss function for multi-class classification. Following Lu et al. [LZKN22], the models are trained for 100 epochs, with the model achieving the highest validation accuracy retained and subsequently used for final evaluation on the test set. Using the same batch size, number of epochs, input resolution, and data splits ensures that differences between models and encodings are not caused by inconsistent training conditions.

The scope of hyperparameters tuning differed between the two architectures, with a broader search space considered for the Vision Transformer than for the CNN baseline. For the Vision Transformer, a small grid search over learning rates (3×10^{-5}, 5×10^{-5}, 1×10^{-4}) and weight decay values (0.0, 0.01, 0.05) was performed. The learning-rate range was chosen based on the range of values reported in related malware-transformer studies [LZKN22, KY24]. Weight decay was included as a regularization hyperparameter, following the optimization considerations described for Vision Transformers by Dosovitskiy et al. [DBK+21]. The evaluated values cover no weight decay (0.0), the default value of 0.01 in the PyTorch AdamW optimizer[7], and a stronger regularization setting of 0.05. In contrast, the CNN baseline was tuned using a limited set of learning rates (5×10^{-5} and 1×10^{-4}), selecting the better-performing configuration. This reflects the focus on Vision Transformers, while the CNN serves as a reference model. Data augmentation is not applied in this work to preserve the original structure of malware representations and avoid introducing artificial samples that do not correspond to real malware instances. While augmentation is commonly used to address class imbalance, prior work shows that

[7] https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

it generates synthetic variations and that its application to malware images is non-trivial [BAC+26, MCS+26].

All experiments are conducted using the following computing resources:

- CPU: Intel Xeon E5-2667 v4 (16 cores)
- RAM: 64 GB
- GPU: NVIDIA TITAN RTX (24 GB VRAM)

4.5 Explainability Methods

This section describes the post-hoc explainability methods selected to analyze the Vision Transformer models and provides the rationale for their inclusion.

As discussed in Section 2.5, explainability methods for Vision Transformers can broadly be divided into attribution-based and attention-based approaches. As summarized by Kashefi et al. [KBSA26], these two categories are among the most widely used explainability approaches for Vision Transformers, largely because they can be applied to pretrained models without architectural modifications and incur relatively low computational overhead. To obtain a balanced comparison while keeping the experimental scope manageable, one representative method is selected from each of three complementary categories: an attribution-based baseline, an attention-only method, and a hybrid attention-based method.

Grad-CAM [SCD+17] is selected as the attribution-based baseline. It is one of the most widely used explainability methods in computer vision and has already been applied in image-based malware classification studies, such as MalFormer by Jeong et al. [JLKC25]. Compared with other attribution-based methods, Grad-CAM is computationally efficient, straightforward to implement, and well supported by existing libraries. It therefore serves as a practical and well-established reference method. In this thesis, Grad-CAM is applied to the last transformer block of the Vision Transformer, and attribution scores are computed with respect to the predicted class unless stated otherwise.

Attention Rollout [AZ20] is selected as the representative attention-only method. Compared with raw attention visualization, it aggregates attention information across all transformer layers while accounting for residual connections, yielding explanations that more accurately reflect information propagation through the model. For the experiments conducted in this work, Rollout uses mean head fusion and no token discarding, i.e., `head_fusion = mean` and `discard_ratio = 0.0`. Mean head fusion follows the original Attention Rollout formulation of Abnar and Zuidema [AZ20]. As the original method does not apply attention-value filtering, a discard ratio of 0.0 is used to retain all attention values and avoid introducing this additional modification. Attention Flow, proposed alongside Attention Rollout, is not included because it is substantially more computationally expensive and has been reported to be too slow for large-scale evaluations

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[AZ20, KBSA26]. Attention Rollout, therefore, represents a practical and widely adopted transformer-specific explainability method.

Grad-SAM [BHC+21] is selected as the representative hybrid explainability method, as it combines attention maps with gradient information. In contrast to Beyond Attention [CGW21], which typically requires an LRP-compatible transformer implementation, Grad-SAM can be applied directly to the pretrained Vision Transformer models used in this work. In this work, Grad-SAM is computed with respect to the predicted class and aggregates gradient-weighted attention information across all heads and transformer layers. Recent comparative evaluations report that Grad-SAM achieves strong performance on faithfulness metrics such as Insertion and performs competitively with more complex hybrid approaches [KBSA26]. Grad-SAM therefore provides a practical compromise between explanation quality and implementation complexity.

Although the selected explainability methods differ in how they compute relevance scores, all produce patch-level relevance maps aligned with the Vision Transformer patch grid, where each score indicates the relative importance of a patch to the predicted class. To ensure comparability across samples and methods, each relevance map is normalized independently to the range $[0, 1]$. The normalized relevance maps are used for visualization and all subsequent explainability analyses.

4.6 Reverse Traceability Framework

The explainability methods considered in this thesis produce patch-level relevance maps that assign an importance score to each image patch processed by the Vision Transformer. Although the underlying algorithms differ, all methods ultimately yield a class-specific relevance map that can be represented as

$$R \in \mathbb{R}^{H_p \times W_p}, \tag{4.1}$$

where P denotes the patch size, and $H_p = H/P$ and $W_p = W/P$ denote the height and width of the Vision Transformer patch grid, such that $N = H_p W_p$ is the total number of image patches. Each element $R_{i,j}$ represents the relevance assigned to a single image patch.

The reverse-traceability framework maps these patch-level relevance scores to executable artifacts, such as functions, instructions, imported API calls, and PE sections, by linking image patches to their corresponding byte ranges in the original executable.

Conceptually, the framework consists of two main components:

1. **Metadata Extraction and Storage**, which extracts and stores the provenance and structural metadata required for reverse mapping.
2. **Traceability Pipeline**, which combines explanation maps with the stored metadata to map image-space relevance scores to executable artifacts.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Figure 4.2 provides an overview of these components and their interaction, while we describe both components in detail below.

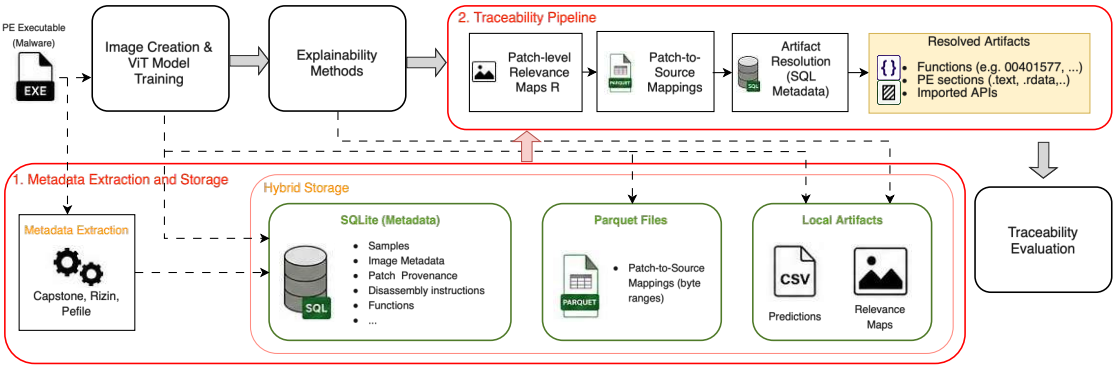


Figure 4.2: Overview of the proposed reverse-traceability framework, including the hybrid metadata storage architecture and the artifact-resolution pipeline.

4.6.1 Metadata Extraction and Storage

During preprocessing, metadata is extracted from the original Portable Executable (PE) files and from the malware-to-image encoding process. PE section boundaries and imported functions are extracted using `pefile`[8]. Instruction-level disassembly is performed with `Capstone`, and function boundaries are recovered using `Rizin`[9]. The encoding process also records image-generation parameters and patch-to-source mappings that map Vision Transformer patches to byte ranges in the original executable.

Hybrid Storage Design

The extracted metadata is stored using a hybrid architecture that combines a SQLite database, per-sample Parquet files, and local explainability artifacts.

The SQLite database serves as the central metadata store, containing sample-level information, image metadata, patch provenance, PE section metadata, and disassembly-derived instruction and function metadata. The relational schema used in the reverse-traceability analysis is provided in Appendix 6.3.

The patch-to-source mappings are stored separately as per-sample Parquet files. Each row records the association between an image patch and the source byte ranges that contributed to that patch. Because these mappings are substantially larger than the remaining metadata, they are stored in Parquet, which provides efficient storage and analysis. Storing all metadata in Parquet was avoided because the remaining metadata is relational and requires queries and joins across different metadata types, for which SQLite is more suitable. The substantially larger patch-to-source mappings, in contrast,

[8] `https://pypi.org/project/pefile/`
[9] `https://rizin.re/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

are primarily accessed as per-sample tabular data and therefore benefit from Parquet's compact columnar storage.

Explainability outputs, including prediction summaries and relevance maps, are stored as local artifacts in the corresponding experiment directories and are loaded on demand during reverse-traceability analysis.

4.6.2 Traceability Pipeline

For a selected sample, the corresponding relevance map R is loaded from the explainability outputs. Each element R_p represents the relevance assigned to a Vision Transformer patch p. The relevance scores are subsequently linked to the source byte ranges recorded in the patch-to-source mapping generated during malware-to-image encoding. These mappings are represented as half-open byte intervals $[s, e)$, where s is inclusive and e is exclusive. Through this mapping, the analysis transitions from image space to the original executable representation.

The recovered source byte ranges are then resolved against executable artifacts stored in the traceability database. Resolution is performed by computing overlap between the source byte ranges associated with a patch and the byte ranges of executable structures, including instructions, functions, imported API calls recovered via IAT overlap (i.e., overlap with the import address table entries referenced by an instruction), and PE sections. A single patch may be associated with multiple source ranges and may therefore overlap multiple executable artifacts. Conversely, a single executable artifact may overlap multiple image patches and can therefore accumulate relevance from several regions of the explanation. If no matching artifact is found, the corresponding source range remains unresolved.

Finally, the resulting patch-to-artifact relationships are recorded together with the corresponding patch coordinates, patch attribution scores, source offsets, and resolved artifact metadata. These traceability outputs can then be aggregated at different levels of granularity. At the PE level, the mappings indicate whether image patches can be associated with executable sections. At the function level, the same mappings are used to identify executable functions and imported APIs associated with highly attributed regions. These outputs form the basis for the reverse-traceability evaluation described in Section 4.7.4, including the assessment of mapping coverage and explanation interpretability.

4.7 Evaluation Design

This section describes the evaluation procedures used to assess classification performance, explanation quality, and reverse traceability.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.7.1 Evaluation of Classification Performance

Unless stated otherwise, the following metrics, introduced in Section 2.4.2, are used throughout the experiments and serve primarily to compare the selected malware-to-image encoding strategies.

Classification effectiveness is assessed using accuracy, macro-averaged F1-score, and false positive rate (FPR). Accuracy measures overall classification performance, while the macro-averaged F1-score is emphasized because the two benchmark datasets exhibit different degrees of class imbalance. While BODMAS-11 has a comparatively balanced class distribution, BIG2015 remains substantially more imbalanced, particularly due to minority families such as Simda. Macro averaging ensures that each malware family contributes equally to the evaluation, regardless of its frequency in the dataset. Confusion matrices are additionally analyzed to identify family-specific misclassification patterns.

Computational efficiency is evaluated using image preprocessing time and total training time. Preprocessing efficiency is assessed using mean and median encoding times per sample, while training efficiency is assessed through the overall training time. Training behavior is additionally analyzed through validation accuracy curves across epochs.

All classification results are reported on the dataset splits described in Section 4.1.3. For BODMAS-11, the primary dataset, all six encodings are additionally evaluated with both architectures on the two alternative dataset splits generated with seeds 7 and 123, in addition to the reference split generated with seed 42. To separately assess variance due to training randomness, the Vision Transformer experiments for all six encodings are repeated using training seeds 42, 7, and 123 while keeping the reference dataset split (seed 42) fixed. The resulting classification performance is compared across training seeds.

4.7.2 Evaluation of Explainability Methods

The explainability evaluation follows the evaluation framework discussed in Section 3.3.2. The quantitative evaluation considers five dimensions: faithfulness, consistency, stability, sparsity, and computational efficiency. The metrics are evaluated separately for each explainability method and malware-to-image encoding. In addition to the aggregate analysis over the complete test set, a family-level analysis is conducted for a representative encoding to investigate whether explainability characteristics vary across malware families.

Faithfulness. Faithfulness is evaluated using Insertion AUC and Deletion AUC, two perturbation-based metrics widely used in the literature on explainability for Vision Transformers [KBSA26]. For each explanation, image patches are ranked according to their relevance scores. In the deletion procedure, the most relevant patches are progressively replaced by a baseline value of 0, starting from the original image. In the insertion procedure, the process is reversed: the input is initialized as a baseline image with all patches set to 0, and the most relevant patches are progressively restored from the original image. This process is performed in 20 steps while recording the model's

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

predicted target-class probability. The area under the resulting probability curves is used as the final score. Higher Insertion AUC values indicate that restoring relevant regions recovers the model's prediction, whereas lower Deletion AUC values indicate that removing relevant regions substantially reduces the target-class probability. Faithfulness Correlation, Infidelity, and Sensitivity-n are not included, as they assess related aspects of faithfulness and would provide limited additional insight.

Consistency. Consistency measures whether similar inputs produce similar explanations and is evaluated using the variance-based formulation of Saqib et al. [SMFC24]. To operationalize this consistency criterion in the context of image-based malware family classification, each sample is combined with class-preserving perturbed variants generated using three perturbation types: additive Gaussian noise ($\sigma = 0.01$), global intensity scaling (random factor within $\pm 5\%$), and random patch masking (2% of image patches). The perturbation strengths were chosen to introduce small variations while preserving the overall structure of the input, as Saqib et al. [SMFC24] do not prescribe specific perturbation parameters for the consistency metric. To ensure that consistency is assessed across different forms of input variation, at least one accepted perturbation from each perturbation type is required. Only perturbations that preserve the target prediction are retained, as consistency is intended to compare explanations for similar inputs that lead to the same model decision. For every accepted variant, an explanation map is generated and converted into a normalized patch-level relevance vector. The explanation vectors of the original sample and all accepted perturbations are then combined into a single set and evaluated using the consistency metric, with a small constant ε added to the denominator to avoid numerical instability when the input variance is close to zero. Higher values indicate more consistent explanations.

Sparsity. Sparsity is evaluated using Effective Complexity [NM20, KBSA26]. Image patches are ranked by relevance, and the smallest number of top-ranked patches required to preserve at least 95% of the original target-class confidence is determined. This Effective Complexity value is normalized by the total number of image patches. Lower Effective Complexity values indicate that the explanation is concentrated on a smaller subset of salient regions. This metric is used because, unlike the sparsity formulation discussed by Saqib et al. [SMFC24], it does not require ground-truth explanation masks.

Stability. Stability assesses whether repeated executions of an explanation method produce similar explanations for the same input and is evaluated using the top-k overlap formulation of Tanha and Kafaie [TK25]. To apply this metric to Vision Transformer-based malware family classification, three independently trained model instances were created for each encoding using random seeds 42, 7, and 123. This follows the rationale of Fan et al. [FWX+20], who assess the stability of explanations across comparable model instances. For each test sample, explanations were generated by all three models, converted into patch-level relevance rankings, and the overlap of the top-k patches was computed according to Equation 3.3. Stability scores were then averaged across all

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

samples in the test set. Higher values indicate that an explanation method identifies similar image regions across independently trained model instances and is therefore less sensitive to random initialization.

Computational Efficiency. Computational efficiency is measured as the average runtime required to generate an explanation for a single test sample.

Excluded Evaluation Dimensions. Based on the applicability considerations discussed in Section 3.3.2, several evaluation dimensions are not included in the quantitative analysis. Interpretability-result metrics such as Intersection over Union (IoU) and Pointing Game require human-annotated ground-truth explanation masks, which are unavailable for malware images. Similarity between explanation outputs generated by different methods is not treated as a primary criterion, as the objective is to assess explanation quality rather than agreement between methods. Completeness is not considered because all selected explainability methods are deterministic gradient- or attention-based techniques that are expected to produce valid saliency maps whenever the forward and backward passes can be computed successfully.

Robustness is not included because the robustness formulations proposed in malware explainability research evaluate class-level explanation similarity across samples from the same and different malware families [SMFC24, TK25]. While this criterion is useful for assessing whether explanations capture family-specific patterns, it differs from the objective of the present evaluation, which focuses on the quality of explanations for individual predictions.

4.7.3 Feature-Space Visualization

While the explainability metrics evaluate the quality of individual explanation methods, they provide limited insight into the structure of the learned feature space. Therefore, feature-space visualizations are generated for each malware-to-image encoding using Principal Component Analysis (PCA) followed by t-distributed Stochastic Neighbor Embedding (t-SNE) [MH08] to investigate whether malware families form separable clusters.

For each test sample, the final representation produced by the Vision Transformer immediately before classification is extracted and used as the feature vector. PCA is first applied to reduce the feature vectors to 50 dimensions, following the preprocessing approach of van der Maaten [vdM14], after which t-SNE projects the feature vectors into two dimensions for visualization. While a direct two-dimensional PCA projection would primarily preserve directions of global variance, t-SNE is used for the final projection because it emphasizes local neighborhood structure, which is of particular interest when examining family-specific clustering [MH08].

While such visualizations provide useful insight into the organization of the learned representations, visual inspection alone is inherently subjective. Therefore, cluster

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

separability is additionally quantified using the Silhouette Score [Rou87]. The score is computed directly on the learned feature representations, using the ground-truth malware-family labels as the cluster assignments. For a sample i, the silhouette coefficient is defined as

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{4.2}$$

where $a(i)$ denotes the average distance to samples belonging to the same malware family and $b(i)$ denotes the average distance to samples of the nearest neighboring family. The overall silhouette score is obtained by averaging $s(i)$ across all samples. Values close to 1 indicate compact and well-separated clusters, whereas values close to 0 indicate overlapping clusters.

4.7.4 Evaluation of Reverse Traceability

The reverse-traceability framework is evaluated against the design requirements derived in Section 3.3.3: traceability, explanation interpretability, and computational efficiency.

Traceability. Traceability concerns whether explanations can be mapped from the image representation back to meaningful locations within the executable. This requirement is evaluated through mapping success at two levels of granularity. First, PE-level mapping success is assessed by measuring the proportion of highly attributed image patches that can be mapped to identifiable PE sections. Second, function-level mapping success is evaluated by measuring the proportion of traced regions that can be further resolved to individual functions. At both levels, mapping does not require a patch to contain an entire section or function; a mapping is considered successful when the source byte range associated with the patch overlaps the byte range of the corresponding executable artifact. A region remains unresolved when its source byte range cannot be associated with an artifact represented in the extracted metadata. The analysis considers mapping success across different Top-k explanation sets. For each sample, image patches are ranked according to their attribution scores, and the k highest-attributed patches are traced back to their source byte ranges. Each of the k patches is evaluated individually, and the Top-k mapping success for a sample is calculated as the proportion of these patches that overlap at least one identified PE section. Thus, the metric represents the mapping rate within the Top-k set rather than cumulative mapping coverage. The reported mapping success is then averaged across all evaluated samples. The analysis additionally examines variation across malware families. Together, these analyses assess both the robustness of the reverse-mapping process and the extent to which explanations can be linked to increasingly fine-grained executable artifacts. High traceability indicates that highlighted image regions can be associated with concrete executable locations, thereby supporting reverse-engineering workflows.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Explanation Interpretability. Explanation interpretability concerns whether the traced executable artifacts provide meaningful insight into the model's decision process. After mapping salient regions back to executable artifacts, the evaluation examines which PE sections and functions are associated with highly attributed explanation regions and whether consistent attribution patterns emerge across malware families. The analysis focuses on the distribution of traced evidence across executable artifacts, including dominant PE sections, recurring functions, and similarities or differences in attribution profiles between malware families.

To complement this dataset-level analysis, a small number of selected samples is examined through focused reverse-engineering case studies. These case studies examine whether the identified functions correspond to plausible malware-related behavior and whether they have a measurable influence on the model's predictions. Together, these analyses assess if the traced artifacts are also interpretable from a malware-analysis perspective. The design of the case studies is described in Section 4.7.5.

Efficiency. The potential of the framework to reduce analyst effort is discussed qualitatively. A quantitative assessment of this requirement would require a dedicated user study comparing analysis workflows with and without reverse traceability and is therefore beyond the scope of this thesis.

4.7.5 Reverse Engineering Case Studies

Because no ground-truth annotations exist that specify which executable artifacts are truly responsible for a malware family prediction, direct evaluation of explanation correctness is not possible. The reverse-engineering case studies, therefore, provide a complementary form of validation by assessing whether the traced artifacts correspond to plausible malware-related functionality and whether they influence the model's prediction. The case studies combine function-level perturbation experiments with manual reverse-engineering inspection.

The case-study analysis proceeds in four main steps:

1. **Sample Selection.** The reverse-engineering case studies are based on a small set of representative samples selected to cover different traceability outcomes, ranging from strong function-level mappings to weaker or only partially recoverable mappings. This enables assessment of whether the identified artifacts remain meaningful under varying levels of traceability quality. Additional samples are included where necessary to evaluate the consistency of the observed effects.

2. **Function Selection and Ranking.** For each selected sample, the analysis starts from the highest-attribution patch, or from a broader set of top-N patches. Functions overlapping the recovered source ranges are then ranked by an overlap-weighted contribution score derived from the traceability output. The score combines patch saliency with the extent of byte-level overlap between a function and the selected

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

patch evidence, so that functions supported by both highly salient patches and substantial overlap receive higher ranks. This makes the ranking an evidence-weighted ordering rather than a pure size-based or count-based heuristic. Functions with larger overlap contributions and support from multiple source ranges are ranked higher. In addition to the full ranking, an instruction-confirmed subset is constructed. Function and instruction metadata are recovered separately during traceability-database construction and linked through address-range overlap. Consequently, a function may overlap source ranges highlighted by the explanation without those same source ranges overlapping recovered instruction records. The instruction-confirmed subset, therefore, restricts the analysis to functions supported by direct instruction-level evidence, providing a stricter set of candidate functions for subsequent analysis.

3. **Function-Level Perturbation Analysis.** The ranked function sets are used to guide controlled function-level perturbation experiments. The byte ranges corresponding to selected functions are modified, the resulting binaries are re-encoded into the image representation used by the classifier, and the model is re-evaluated. The analysis compares perturbations of highly ranked functions, lower-ranked functions, and instruction-confirmed subsets. The experiments comprise three components: the applied perturbation modes, a comparison against a random-function baseline, and an analysis of the function ranking:

   a) **Perturbation Modes.** Two perturbation modes are used. In the random-byte mode, the selected function bytes are replaced by deterministic pseudorandom values generated from a seed derived from the sample and function identifier, ensuring repeatable perturbations across runs. In the zero-byte mode, the same byte range is replaced entirely by zeros.

   b) **Random-Function Baseline.** For baseline comparison, a pool of valid functions is constructed for each sample from the set of all recovered functions. Ten random function orders are then sampled from this pool, and cumulative ablation is performed under both perturbation modes. The resulting confidence values are averaged across trials to obtain baseline mean curves, providing a reference for assessing whether highly ranked functions influence the prediction more strongly than randomly selected functions.

   c) **Ranking-Order Analysis.** Additional experiments investigate different rank windows and alternative function orderings. To assess ranking effects independently of function-set membership, ten random permutations of selected ranked function sets are generated and evaluated under both perturbation modes. The resulting perturbation trajectories are then compared with those obtained from the original ranked order.

   After each perturbation step, the predicted class and the softmax confidence assigned to the true class are recorded. The perturbation effect is quantified by the first class-flip point, the number of ablation steps resulting in a class flip, the minimum

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

true-class confidence, and the area under the true-class confidence curve. Larger decreases in true-class confidence and earlier or more frequent class flips following perturbation of highly ranked functions provide evidence that the traced functions are relevant to the model's decision process.

4. **Manual Reverse-Engineering Inspection.** Based on the traceability results and the observed perturbation effects, selected functions are inspected manually in IDA. The analysis examines whether the identified functions correspond to plausible malware-related behavior and whether they provide a meaningful explanation of the model prediction. This qualitative inspection complements the quantitative perturbation analysis by assessing the interpretability of the traced artifacts from a malware-analysis perspective.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Results

This chapter presents the results of the experimental evaluation. The findings cover classification performance, explainability evaluation, reverse traceability analysis, and reverse-engineering case studies. Together, these results provide insights into the effectiveness of different malware-to-image encodings, the quality of the generated explanations, the extent to which explanation regions can be linked back to executable code and semantic program components, and the practical utility of the proposed reverse-traceability framework for malware analysis. Each section concludes with a brief summary of the key findings to highlight the principal observations and facilitate comparison across the evaluated aspects.

5.1 Classification Performance

In this section, the classification performance of the investigated malware-to-image encodings is evaluated on the BIG2015 and BODMAS-11 datasets. The analysis considers both classification effectiveness and computational efficiency to assess the performance-efficiency trade-offs of different encoding strategies. BIG2015 is a widely used benchmark dataset that enables comparison with prior malware-image classification studies, while BODMAS-11 provides a more recent benchmark for evaluating encoding performance on contemporary malware samples.

5.1.1 Classification Performance on BIG2015

BIG2015 is first used to evaluate how the investigated encodings perform in a standardized experimental setting and, where possible, to compare the results with prior malware-image classification studies. Table 5.1 reports the ViT classification results for the six encodings.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.1: Classification effectiveness and efficiency of the investigated encoding strategies on the BIG2015 dataset using the Vision Transformer classifier. Preprocessing times are reported per sample. The best result for each metric is highlighted in dark green.

Encoding	Acc.	F1	FPR	Epoch	Train (m)	Prep. Mean (ms)	Prep. Med. (ms)
markov_bytes	0.99213	0.96855	0.00101	34	381.5	399	367
rgb	0.99075	0.96845	0.00116	10	381.8	333	312
gray	0.98889	0.97938	0.00140	63	382.1	342	352
entropy_malfcs	0.98797	0.94822	0.00149	4	383.2	444	414
mctvd	0.99075	0.97732	0.00116	12	381.6	1469	478
simhash_opcode	0.93522	0.92255	0.00840	37	381.0	912	377

Overall, five of the six encodings achieved very high classification performance on BIG2015. Markov Bytes achieved the highest test accuracy at 99.21%, followed by MCTVD and RGB at 99.07%. Gray achieved 98.89% accuracy, followed by Entropy MALFCS at 98.80%. With the exception of SimHash Opcode, all investigated encodings reached approximately 98.8% accuracy or higher.

Comparison with Reported Literature Results

Table 5.2 compares the classification performance obtained in this study with the results reported in the respective original publications. Direct comparisons should be interpreted with caution, as the reviewed works employ different classifier architectures, preprocessing pipelines, image resolutions, and evaluation protocols. In addition, the exact train-test splits used in the original works are not available, making exact replication impossible. Consequently, identical performance values are not necessarily expected, even when the same dataset is used.

Table 5.2: Comparison of classification accuracy reported in the original studies and obtained by the proposed Vision Transformer (ViT) framework on BIG2015

Encoding	Original Paper	Model	Accuracy	This Work
Gray	Lu et al. [LZKN22]	ViT	96.35%	98.89%
RGB	Wang et al. [WZLZ19]	CNN	95.20%	99.07%
Markov Bytes	Yuan et al. [YWL+20]	CNN	99.26%	99.21%
MCTVD	Deng et al. [DGS+23]	CNN	99.44%	99.07%
Entropy MALFCS	Xiao et al. [XLCL20]	CNN	100.00%	98.80%
SimHash Opcode	Ni et al. [NQZ18]	CNN	98.86%	93.52%

In our evaluation, several encodings achieved performance levels that closely match those reported in the original studies. In particular, Markov Bytes achieved an accuracy of 99.21%, compared with 99.26% reported by Yuan et al. [YWL+20]. Similarly, MCTVD achieved 99.07%, remaining close to the 99.44% reported by Deng et al. [DGS+23].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

These small differences suggest that the discriminative information captured by both representations remains effective when transferred to a Vision Transformer architecture.

For Gray and RGB encodings, the present study achieved substantially higher accuracies than those reported in the original studies. Gray improved from 96.35% reported by Lu et al. [LZKN22] to 98.89%, while RGB improved from 95.20% reported by Wang et al. [WZLZ19] to 99.07%. These differences should be interpreted cautiously, as the original studies employed different classifier architectures and evaluation protocols. In particular, Lu et al. [LZKN22] evaluated grayscale malware images using their transformer-based ImgConvAttn architecture and partitioned BIG2015 into approximately 80% for training, 10% for validation, and 10% for testing. Wang et al. [WZLZ19] employed the ColCaps framework, which combines RGB malware visualization with a capsule network, for malware classification but did not report the exact dataset partitioning used for evaluation. Because the grayscale and RGB image generation procedures were reproduced from the original studies, the higher accuracies demonstrate that these relatively simple representations remain competitive when evaluated using the same ViT architecture and experimental setup as the other investigated encodings, rather than an encoding-specific optimized setup.

The entropy-based MALFCS representation achieved an accuracy of 98.80%, compared with the perfect classification accuracy reported by Xiao et al. [XLCL20]. The inability to reproduce the reported 100% accuracy suggests that the original result may be sensitive to the specific experimental conditions under which it was obtained. A more detailed assessment of this discrepancy is difficult, as Xiao et al. do not provide sufficient information to fully reproduce their experimental setup. Nevertheless, MALFCS remained among the strongest-performing encodings evaluated in this work.

The largest deviation from the literature was observed for SimHash Opcode, which achieved 93.52% accuracy compared with the 98.86% reported by Ni et al. [NQZ18]. Notably, a similar performance level was observed for both the CNN and ViT models evaluated in this study, suggesting that the discrepancy cannot be explained solely by classifier architecture. Furthermore, an additional experiment using the original 32×32 image size reported by Ni et al. yielded an even lower accuracy of 91.44%, indicating that the performance gap is not due to the standardized 256×256 image resolution used in our main evaluation. Reproducing the original SimHash pipeline is challenging because several preprocessing and evaluation details are only partially documented. The study reports experiments on a dataset containing 10,805 samples, whereas the explicit filtering procedure adopted in this work yields 10,801 samples. In addition, the exact criteria for identifying valid code regions in the BIG2015 assembly files are not fully specified, even though the SimHash representation depends directly on the extracted opcode sequences. The reported evaluation is based on an 80/20 train-test split, but the exact sample assignments and randomization procedure are not provided. The observed performance gap may therefore reflect limitations of the SimHash representation itself, differences in preprocessing and evaluation procedures, or a combination of both, making it difficult to attribute the discrepancy solely to the encoding. To avoid this ambiguity, the exact

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

sample assignments used in this work are provided in the accompanying repository, as described in Section 4.1.3.

Beyond comparisons with the original encoding publications, it is also useful to position the obtained results relative to more advanced Vision Transformer-based malware classification approaches reported in the literature. The best-performing configuration achieved 99.21% accuracy on BIG2015, comparable to HERL-ViT (99.20%) [CWQ+25] despite using a standard ViT-B/16 architecture. While B-ViT reports the highest accuracy among the reviewed Vision Transformer-based malware classification approaches on BIG2015 (99.49%) [BS23], this result should be interpreted cautiously, as discussed in Section 3.2.4. In particular, B-ViT applies data augmentation and class balancing, and it remains unclear whether the reported performance was obtained on a strictly held-out test set. Overall, these findings indicate that competitive performance can be achieved without relying on more complex hierarchical or hybrid transformer designs.

Detailed Analysis of Classification Performance

A closer examination of the classification results in Table 5.1 shows that accuracy and macro-F1 do not identify the same best-performing encoding. While Markov Bytes achieved the highest overall accuracy, Gray achieved the highest macro-F1 score with 97.94%, followed by MCTVD with 97.73%. This distinction is particularly relevant given the class imbalance of BIG2015, as macro-F1 assigns equal importance to all malware families regardless of their frequency. This effect is also reflected by Entropy MALFCS, which achieved an overall accuracy comparable to Gray but a substantially lower macro-F1 score (94.82%). The results, therefore, suggest that Gray and MCTVD provide more balanced performance across malware families, whereas Markov Bytes achieves the strongest overall classification accuracy.

To better understand the differences between accuracy and macro-F1, Figure 5.1 shows the row-normalized confusion matrices for the three best-performing encodings. All three matrices exhibit a strong diagonal structure, indicating highly accurate classification across malware families. The largest families, including Ramnit, Lollipop, and Kelihos_ver3, are classified almost perfectly by all three encodings, with only minor differences between them. The performance differences reflected in the macro-F1 scores originate primarily from smaller and medium-sized families such as Vundo and Simda, where Gray and MCTVD exhibit fewer misclassifications than Markov Bytes. In particular, Gray and MCTVD correctly classify 88% of Simda samples, compared with 75% for Markov Bytes, while 99% of Vundo samples are correctly classified by Gray and MCTVD compared with 95% by Markov Bytes. Overall, the confusion matrices suggest that the strongest encodings differ mainly in their treatment of minority classes, while their performance on the dominant families is nearly identical.

Although the strongest encodings all achieved very similar classification effectiveness, their training behavior was also broadly comparable, while larger efficiency differences emerged during preprocessing. As shown in Figure 5.2, most encodings reached stable

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

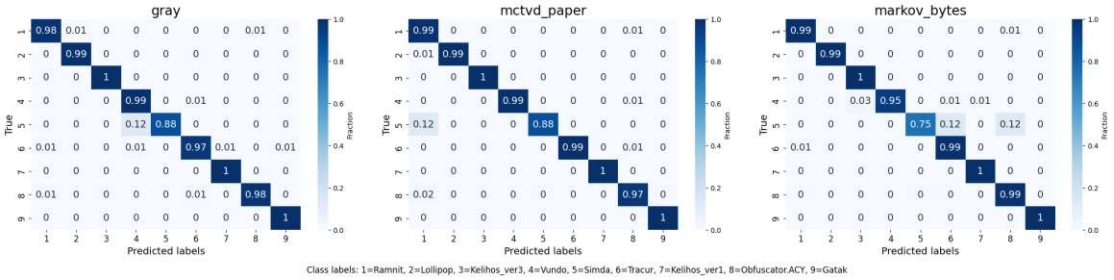


Figure 5.1: Row-normalized confusion matrices of the three best-performing encoding strategies on the BIG2015 dataset, where values indicate the fraction of samples within each true malware family assigned to each predicted class. Gray achieves 98.89% accuracy and 97.94% macro-F1, MCTVD 99.07% accuracy and 97.73% macro-F1, and Markov Bytes 99.21% accuracy and 96.86% macro-F1.

validation performance within approximately the first 10–20 epochs. Although the model with the highest validation accuracy was selected at different epochs (e.g., epoch 63 for Gray), these differences primarily reflected small fluctuations in validation accuracy rather than continued performance improvements. Training times remained nearly identical across all encodings. The largest efficiency differences occurred during preprocessing. RGB and Gray exhibited the lowest mean preprocessing costs, requiring approximately 333 ms and 342 ms per sample, respectively, whereas MCTVD incurred substantially higher preprocessing costs (1469 ms per sample) due to the additional disassembly and opcode-transition extraction steps required to construct the representation. Mean and median preprocessing times were similar for most encodings, whereas MCTVD and SimHash Opcode exhibited substantially larger differences, indicating greater variability in preprocessing costs across samples.

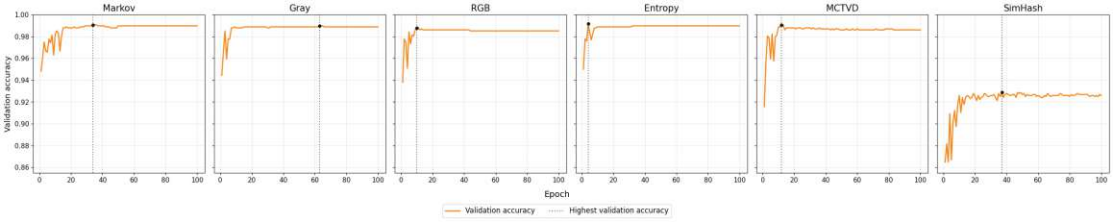


Figure 5.2: Validation accuracy across training epochs for the investigated encodings on BIG2015. The dotted line indicates the epoch at which the model with the highest validation accuracy was selected.

Comparison of CNN and Vision Transformer Architectures

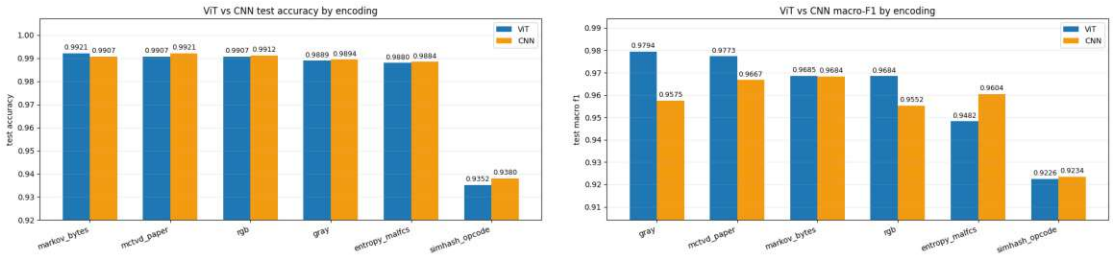
To evaluate the influence of the classifier architecture on the encodings, we compare the Vision Transformer results with those on a ResNet-18-based CNN. Table 5.3 reports the exact classification results, while Figure 5.3 provides a direct visual comparison. Overall,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

both architectures produced highly similar results, with classification accuracies differing by only 0.05–0.28 percentage points between the CNN and Vision Transformer across the six encodings. For example, MCTVD achieved 99.21% accuracy with the CNN and 99.07% with the Vision Transformer, a difference of only 0.14 percentage points. Larger differences emerged for macro-F1 (Figure 5.3b). The Vision Transformer achieved higher macro-F1 scores for Gray, RGB, and MCTVD, whereas the CNN performed better for Entropy MALFCS and SimHash Opcode. Markov Bytes resulted in nearly identical macro-F1 scores across both architectures. The largest difference was observed for Gray, where macro-F1 increased from 95.75% for the CNN to 97.94% for the Vision Transformer. In summary, these results suggest that classifier architecture has only a limited impact on the relative performance of the investigated encodings, although certain representations, particularly Gray, appear to benefit more strongly from the Vision Transformer.

Table 5.3: Comparison of classification performance between the Vision Transformer (ViT) and ResNet-18-based CNN across the investigated encoding strategies on BIG2015.

Encoding	ViT Acc.	CNN Acc.	ViT Macro-F1	CNN Macro-F1
markov_bytes	0.99213	0.99075	0.96855	0.96836
rgb	0.99075	0.99121	0.96845	0.95523
gray	0.98889	0.98936	0.97938	0.95749
entropy_malfcs	0.98797	0.98843	0.94822	0.96038
mctvd	0.99075	0.99213	0.97732	0.96672
simhash_opcode	0.93522	0.93799	0.92255	0.92345



(a) Accuracy.

(b) Macro F1-score.

Figure 5.3: Comparison of CNN and Vision Transformer performance across malware image encodings on BIG2015.

5.1.2 Classification Performance on BODMAS-11

The encodings were also evaluated on BODMAS-11, a larger and more contemporary malware dataset. Table 5.4 summarizes the resulting classification performance of the Vision Transformer across all evaluated encodings.

Overall, Markov Bytes achieved the strongest performance across all evaluation metrics, obtaining the highest accuracy (97.24%), the highest macro-F1 score (97.04%), and the lowest macro-FPR (0.0027). RGB and Gray followed closely, consistently ranking

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 5.4: Classification effectiveness and efficiency of the investigated encodings on BODMAS-11 (ViT) using the Vision Transformer classifier. Preprocessing times are reported per sample. The best result for each metric is highlighted in dark green.

Encoding	Acc.	F1	FPR	Epoch	Train (m)	Prep. Mean (ms)	Prep. Med. (ms)
markov_bytes	0.97238	0.97041	0.00275	25	1281.4	1248	308
rgb	0.97127	0.96850	0.00286	12	821.2	1124	245
gray	0.96943	0.96646	0.00308	63	922.9	1142	238
entropy_malfcs	0.96611	0.96322	0.00338	10	807.2	1425	348
mctvd	0.94512	0.93601	0.00542	25	593.8	126	37
simhash_opcode	0.84715	0.83347	0.01632	83	543.8	115	20

among the strongest-performing encodings. In contrast, MCTVD achieved noticeably lower performance than the remaining image-based encodings, which differs from the BIG2015 results, where it ranked among the strongest representations. Similar to the observations on BIG2015, SimHash Opcode again produced the weakest results by a substantial margin across all evaluation metrics.

For comparison, the paper by Lu et al. [LZKN22] was the only identified work reporting classification results specifically on the same 11-family subset of BODMAS. Lu et al. reported accuracies between 95.02% and 96.96% on the BODMAS-11 benchmark, depending on the image representation and architecture employed. The highest accuracy obtained in this work was 97.24% using the Markov Bytes encoding, while the grayscale Vision Transformer achieved 96.94%. Although direct comparison is limited by differences in image representations and architectures, the results indicate competitive performance on the BODMAS-11 benchmark.

Examination of the confusion matrices revealed largely similar class-level prediction patterns across the investigated encodings. The most notable exception was the *Wacatac* family, for which MCTVD misclassified 36.7% of samples, compared with 11.9% for Gray and 9.5% for Markov Bytes. This class-specific degradation accounts for a substantial portion of the performance gap between MCTVD and the stronger-performing encodings on BODMAS-11.

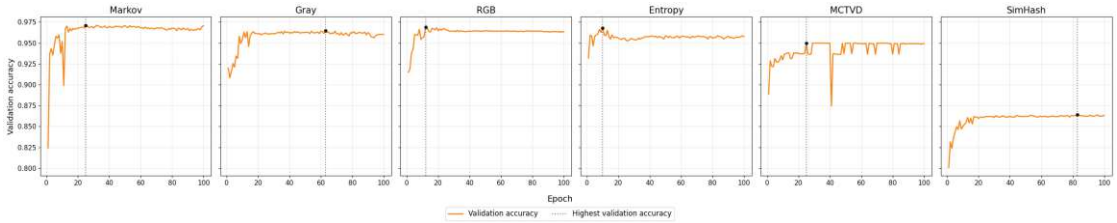


Figure 5.4: Validation accuracy trajectories for the investigated encodings on BODMAS-11. Dotted lines indicate the epoch at which the model with the highest validation accuracy was selected.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure 5.4 shows the validation accuracy curves of the investigated encodings on BODMAS-11. The observed validation trajectories largely mirror those on BIG2015. RGB exhibited the most stable validation trajectory, reaching a plateau within approximately the first 10–15 epochs with little subsequent variation. Markov Bytes again showed larger fluctuations during the initial training epochs before stabilizing, while Gray reached a stable validation plateau early, yet its highest validation accuracy was again obtained considerably later in training. In contrast to BIG2015, MCTVD exhibited noticeably greater instability after reaching its plateau and stabilized at a substantially lower validation accuracy, consistent with its reduced classification performance on BODMAS-11.

Although the validation trajectories were broadly similar to those observed on BIG2015, the effectiveness–efficiency trade-offs were more pronounced on BODMAS-11. Here, MCTVD exhibited one of the lowest mean preprocessing costs (126 ms per sample) but ranked among the weakest image-based encodings. This change coincided with a substantial reduction in assembly file size between the two datasets, with the median ASM file size decreasing from 3.34 MB in BIG2015 to 0.024 MB in BODMAS-11. SimHash Opcode showed a similar pattern, combining a low computational cost (115 ms per sample) with the overall weakest classification performance. Interestingly, the median preprocessing times for the byte-based encodings remained relatively similar across both datasets, whereas their mean preprocessing costs increased substantially on BODMAS-11. For example, Markov Bytes exhibited median preprocessing times of 367 ms on BIG2015 and 308 ms on BODMAS-11, while the corresponding mean preprocessing costs increased from 399 ms to 1248 ms. As shown in Figure 5.5, this difference is consistent with the more skewed byte-file size distribution of BODMAS-11, which exhibits a broader distribution and a longer tail toward large files. The corresponding statistics in Table 5.5 show that BODMAS-11 has a lower median file size but substantially larger upper-quartile and mean file sizes.

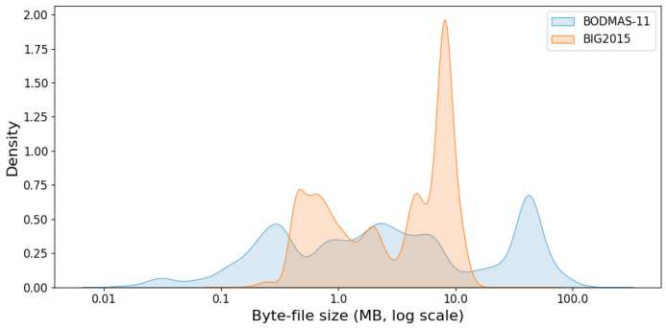


Figure 5.5: Density distributions of byte-file sizes for BIG2015 and BODMAS-11. The file-size axis is shown on a logarithmic scale.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.5: Byte-file size distribution statistics for BIG2015 and BODMAS-11. Q1 and Q3 denote the first (25th percentile) and third (75th percentile) quartiles, respectively. All values are reported in megabytes (MB).

Dataset	Q1	Median	Q3	Mean	Max
BIG2015	0.88	4.17	8.49	4.46	53.3
BODMAS-11	0.41	2.38	16.47	12.51	170.2

Comparison of CNN and Vision Transformer Architectures

Similar to BIG2015, both ViTs and the ResNet-18-based CNN architectures produced nearly identical rankings of the investigated encodings, with Markov Bytes remaining the strongest-performing representation and SimHash Opcode consistently achieving the weakest results. However, unlike BIG2015, where Gray and Entropy MALFCS exhibited noticeable macro-F1 differences between CNN and Vision Transformer models, architecture-dependent effects on BODMAS-11 were minimal: for all encodings, differences in accuracy and macro-F1 remained below 0.3 percentage points. This suggests that the architecture-specific effects observed on BIG2015 become less pronounced on the more balanced BODMAS-11 dataset.

A comparison of the class-level prediction patterns further supports this observation. The largest differences in per-family recall were 4 percentage points, observed for *CeeInject* with SimHash Opcode (93% for the CNN compared with 89% for the Vision Transformer) and *Small* with Entropy MALFCS (94% compared with 90%, respectively). However, the differences did not consistently favor the CNN. For example, for *Wacatac* with RGB, the Vision Transformer achieved a recall of 92% compared with 90% for the CNN. Overall, the two architectures exhibited broadly similar class-level error patterns, with no consistent tendency for either architecture to produce more errors for particular malware families. The remaining differences, therefore, appear to depend more on the specific encoding–family combination than on the classifier architecture itself.

Robustness Analysis

To assess the robustness of the obtained results, we conducted a series of additional experiments using different dataset splits, training seeds, and alternative disassembly backends for MCTVD. Given the scope of these additional experiments, the robustness analysis focuses on BODMAS-11 as the primary dataset of this study.

First, we extended the evaluation on the three BODMAS-11 dataset partitions described in Section 4.1.3. Figure 5.6 summarizes the resulting mean accuracy and macro-F1 scores across splits. Detailed per-partition classification results, together with the corresponding mean and standard deviation across partitions, are provided in Appendix 6.3. SimHash Opcode is omitted from the figure for readability due to its substantially lower classification performance; however, its complete results are reported in Appendix 6.3, and it is included in the subsequent robustness analysis. Across all partitions, the overall performance trends remained consistent, with Markov Bytes, RGB, and Gray forming the strongest-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

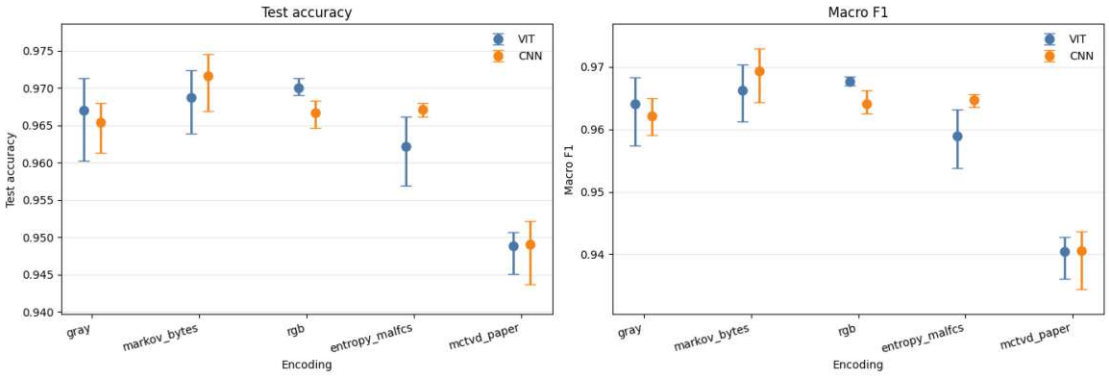


Figure 5.6: Mean test accuracy and macro-F1 across split seeds for ViT and CNN on BODMAS-11, with error bars showing the standard deviation across seeds.

performing group. Performance variability across dataset partitions was generally small. Depending on the encoding, the difference between the highest and lowest test accuracy ranged from 0.22 to 1.11 percentage points, while the corresponding difference in macro-F1 ranged from 0.14 to 1.09 percentage points. The additional splits also confirmed the architecture-specific trends observed in the primary evaluation. For example, for the Gray encoding, the Vision Transformer achieved a mean accuracy of 96.70% and a mean macro-F1 score of 96.40%, compared with 96.54% and 96.21%, respectively, for the CNN. While classification effectiveness remained highly stable across dataset partitions, the best-validation epoch of the Vision Transformer exhibited greater variability. RGB and Entropy MALFCS showed the smallest variation, with the selected best-validation epochs differing by only eight epochs between the earliest and latest selections across the three dataset partitions. Markov Bytes exhibited moderate variability (17 epochs), whereas Gray, MCTVD, and SimHash Opcode showed substantially larger ranges of 46, 51, and 50 epochs, respectively.

Second, to evaluate the influence of optimization randomness, the Vision Transformer models were trained using three different random seeds while keeping the dataset split fixed (see Table 4 in Appendix 6.3). Performance differences remained small, with accuracy variations below 0.2 percentage points for all encodings except Entropy MALFCS, which exhibited a maximum variation of 0.48 percentage points. Across all random seeds, the ranking of the investigated encodings remained unchanged. This indicates that optimization randomness had only a minor influence on the observed performance differences.

Finally, to investigate whether the reduced performance of MCTVD on BODMAS-11 could be attributed to limitations of the Capstone-based disassembly pipeline, an additional experiment was conducted using the Ghidra-based ASM representation described in Section 4.1.1. The resulting performance differences were small, with the Ghidra-based representation achieving slightly higher classification accuracy for the CNN (94.64% compared with 94.36% for Capstone), whereas the Capstone-based representation achieved

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

slightly higher accuracy for the Vision Transformer (94.51% compared with 94.15% for Ghidra). Although the Ghidra-generated assembly files were generally larger than the corresponding Capstone-based outputs (median 0.046 MB versus 0.024 MB), these differences led to only minor changes in classification performance. Overall, these findings suggest that the observed performance reduction in MCTVD on BODMAS-11 is unlikely to be caused solely by the choice of disassembly backend.

5.1.3 Classification Performance Analysis

A central finding is that the performance gap between malware-to-image encodings is substantially smaller than suggested by many individual encoding studies [YWL+20, XLCL20, DGS+23] when evaluated under identical experimental conditions. Nevertheless, the alternative encoding strategies proposed in the malware-image classification literature proved to be at least as effective as the grayscale and RGB representations that dominate current ViT-based malware classification research. In particular, Markov Bytes, Gray, and RGB consistently ranked among the best-performing representations across both BIG2015 and BODMAS-11. Entropy MALFCS also achieved high overall accuracy on both datasets, although its macro-F1 on BIG2015 was noticeably lower than that of these three encodings. Although Markov Bytes frequently achieved the highest classification effectiveness in the primary evaluation, no single encoding emerged as a clear overall winner across all evaluation criteria. Among the top-performing encodings, classification performance differed only marginally. Training behavior was also broadly comparable, whereas more pronounced differences emerged in preprocessing cost. On BIG2015, RGB and Gray had the lowest mean preprocessing costs, whereas on BODMAS-11 the lowest costs were achieved by SimHash Opcode and MCTVD, although both exhibited substantially lower classification effectiveness. Among the strongest-performing encodings on BODMAS-11, RGB provided the lowest mean preprocessing cost and exhibited the most stable validation trajectory.

A second key finding is that performance differences between CNN and Vision Transformer architectures appear smaller under a standardized evaluation framework than suggested by many individual studies. Across both datasets, CNN and Vision Transformer architectures achieved largely comparable classification performance, despite prior work reporting stronger claims of superiority [LZKN22, PC22, BaoEB23]. Although the Vision Transformer achieved higher macro-F1 scores for certain encodings, particularly Gray, the overall differences remained modest. This suggests that architectural choice alone may have a smaller impact on classification performance than often implied in the literature [PC22, BaoEB23, LZKN22]. Instead, differences in experimental design, preprocessing pipelines, dataset partitioning, and evaluation methodologies may contribute substantially to many of the performance differences reported across studies. Furthermore, the observed ranking patterns remained stable across alternative dataset partitions and training seeds, increasing confidence in the robustness of our findings.

Finally, the results suggest that the benefits of higher-level opcode-based image representations depend strongly on the availability of recoverable assembly information. While

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

MCTVD performed competitively on BIG2015, its effectiveness decreased substantially on BODMAS-11, coinciding with a large reduction in the amount of recovered assembly code. This trend was observed across two disassembly backends, suggesting that opcode-based representations may be more sensitive to code recoverability and dataset characteristics than byte-based approaches. Consequently, utilizing higher-level semantic information derived through disassembly alone does not guarantee superior classification performance. One possible explanation is that differences in packing, obfuscation, or other anti-analysis techniques between the datasets contribute to the substantially lower amount of recoverable assembly code observed for BODMAS-11. In contrast, byte-based encodings remain directly applicable regardless of how much higher-level program structure can be recovered.

5.2 Explainability Evaluation

The classification experiments demonstrated that multiple malware-to-image encodings achieve highly competitive classification performance, with no single representation emerging as a clear overall winner across all evaluation criteria. While these results provide insight into the effectiveness and efficiency of different encoding strategies, they do not reveal how the resulting Vision Transformer models arrive at their predictions. The explainability evaluation therefore investigates how the malware family classification methods arrive at their predictions.SimHash Opcode was excluded from the explainability evaluation because it consistently exhibited substantially lower classification performance than the remaining encodings across both datasets and therefore did not represent a competitive encoding alternative. Because BODMAS-11 provides a larger and more diverse evaluation set than BIG2015, the remainder of the explainability evaluation focuses exclusively on BODMAS-11. The evaluation compares Attention Rollout, Grad-CAM, and Grad-SAM across the remaining malware-to-image encodings. Both qualitative and quantitative analyses were conducted to evaluate characteristics of the explanations. In addition, a feature-space analysis was performed to investigate the structure of the learned representations.

5.2.1 Qualitative Analysis of Explanations

Because each malware-to-image encoding represents distinct characteristics of the underlying executable, explanation maps must be interpreted in the context of the corresponding representation. For Gray encoding, salient image regions correspond directly to contiguous byte regions within the executable. RGB preserves the original byte ordering but groups three consecutive bytes into color channels, such that salient regions correspond to byte triplets. In contrast, Byte-Level Markov encodings represent byte-transition statistics, meaning that salient regions correspond to frequently occurring byte transitions rather than to individual byte locations. Entropy MALFCS highlights entropy patterns across byte segments, while MCTVD explanations correspond to opcode-transition relationships derived from disassembled code.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

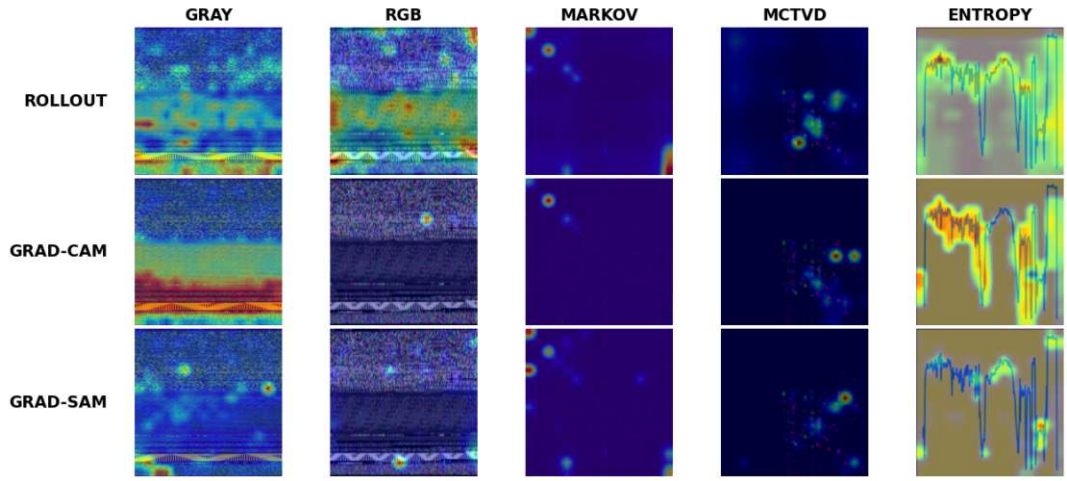


Figure 5.7: Comparison of explanation maps for the same correctly classified *Wabot* sample across malware-to-image encodings and explainability methods.

Figure 5.7 shows explanation maps generated for the same correctly classified malware sample from the *Wabot* family across all investigated encodings and explainability methods. The figure illustrates that the resulting explanations differ across both malware-to-image encodings and explainability methods. In particular, for the Gray encoding, Grad-CAM seems to highlight a larger spatial region of relevance, whereas Rollout and Grad-SAM tend to produce more localized attribution patterns. Consequently, qualitative inspection provides useful insight into how different representations influence explanation maps, but does not allow reliable assessment of explanation quality, as the visual appearance of an explanation does not necessarily reflect how faithfully or consistently it captures the model's decision process. Quantitative evaluation is therefore required to compare explainability methods with respect to faithfulness, sparsity, consistency, stability, and computational efficiency.

5.2.2 Quantitative Comparison of Explainability Methods

Table 5.6 summarizes the explainability results across all evaluated malware-to-image encodings and explainability methods. The following analysis examines each evaluation dimension individually.

Faithfulness Across all malware-to-image encodings, Attention Rollout and Grad-SAM consistently achieved higher mean Insertion AUC scores than Grad-CAM. The differences between the two methods incorporating attention information were generally small: Rollout achieved marginally higher insertion scores for Entropy (0.942 vs. 0.940) and Gray (0.902 vs. 0.899), whereas Grad-SAM achieved higher scores for Markov (0.856 vs. 0.816) and RGB (0.924 vs. 0.909). For MCTVD, both methods obtained identical mean Insertion AUC values of 0.903. In contrast, Grad-CAM consistently produced

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.6: Summary of explainability evaluation results across malware-to-image encodings and explainability methods. Higher values indicate better performance for Insertion AUC (Ins.), Consistency (Cons.), and Stability@20 (Stab.@20). Lower values indicate better performance for Deletion AUC (Del.), Effective Complexity (EC), and mean explanation generation time per sample in milliseconds (Time). Stability@20 denotes the average top-20 patch overlap across independently trained model instances, macro-averaged over malware families. The best result for each metric and encoding is highlighted in dark green.

Encoding	Method	Ins.↑	Del.↓	EC↓	Cons.↑	Stab.@20↑	Time↓
Entropy	Grad-CAM	0.760	0.487	68.431	0.981	0.209	32.117
	Grad-SAM	0.940	0.178	10.383	0.999	0.493	34.121
	Rollout	0.942	0.187	9.492	1.000	0.688	11.802
Gray	Grad-CAM	0.747	0.600	72.573	0.921	0.371	34.637
	Grad-SAM	0.899	0.580	26.486	0.977	0.421	35.217
	Rollout	0.902	0.583	25.555	0.994	0.559	12.149
Markov	Grad-CAM	0.567	0.552	117.659	0.962	0.315	31.504
	Grad-SAM	0.856	0.167	33.849	0.997	0.464	36.031
	Rollout	0.816	0.147	38.962	0.996	0.507	12.801
MCTVD	Grad-CAM	0.402	0.616	163.872	0.998	0.376	28.588
	Grad-SAM	0.903	0.099	25.898	1.000	0.575	35.329
	Rollout	0.903	0.098	25.508	1.000	0.377	11.868
RGB	Grad-CAM	0.701	0.585	110.203	0.966	0.354	33.351
	Grad-SAM	0.924	0.424	28.046	0.997	0.373	37.466
	Rollout	0.909	0.441	31.359	0.999	0.572	16.014

substantially lower insertion scores, most notably for MCTVD, where it achieved only 0.402, compared with 0.567 for Markov, 0.701 for RGB, and 0.760 for Entropy.

A similar pattern emerged for mean Deletion AUC. The differences between Rollout and Grad-SAM remained small across all encodings, and neither method consistently outperformed the other. Grad-SAM achieved lower deletion scores for Entropy (0.178 vs. 0.187) and RGB (0.424 vs. 0.441), whereas Rollout achieved lower scores for Markov (0.147 vs. 0.167). For Gray and MCTVD, both methods produced nearly identical results (0.580 vs. 0.583 and 0.098 vs. 0.099, respectively). Although Grad-CAM achieved a comparable mean deletion score for Gray (0.600), its interquartile range was substantially larger (0.468 vs. 0.320 for Rollout and 0.327 for Grad-SAM; see Table 5.7), indicating greater variability in deletion performance across samples despite similar average performance.

Sparsity Effective Complexity (EC) results indicate that Rollout and Grad-SAM generally produced similarly sparse explanations. For Entropy, Gray, and MCTVD, the differences between the two methods were negligible, with EC values differing by less than one point (0.891, 0.931, and 0.390, respectively). More pronounced differences emerged for Markov and RGB, where Grad-SAM achieved lower EC values than Rollout

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.7: Interquartile ranges (IQR) of Insertion and Deletion AUC across evaluated samples for each malware-to-image encoding and explainability method. Lower values indicate lower variability across samples. The best result for each metric and encoding is highlighted in dark green.

Encoding	Method	Insertion IQR↓	Deletion IQR↓
Entropy	Grad-CAM	0.149	0.510
	Grad-SAM	0.017	0.117
	Rollout	0.009	0.150
Gray	Grad-CAM	0.356	0.468
	Grad-SAM	0.102	0.327
	Rollout	0.100	0.320
Markov	Grad-CAM	0.662	0.514
	Grad-SAM	0.098	0.135
	Rollout	0.102	0.159
MCTVD	Grad-CAM	0.201	0.223
	Grad-SAM	0.058	0.006
	Rollout	0.061	0.002
RGB	Grad-CAM	0.403	0.538
	Grad-SAM	0.052	0.263
	Rollout	0.097	0.310

(33.849 vs. 38.962 and 28.046 vs. 31.359, respectively), indicating more concentrated relevance maps. Grad-CAM, however, consistently produced substantially larger EC values across all encodings, ranging from 68.431 to 163.872, suggesting considerably more diffuse explanations.

Consistency Consistency scores were generally high across all methods and encodings. Attention Rollout and Grad-SAM achieved near-perfect consistency throughout the evaluation, with only negligible differences between them. Grad-CAM also produced high consistency scores overall, but consistently trailed the attention-based methods. The largest difference was observed for Gray, where Grad-CAM achieved a score of 0.921 compared to 0.977 for Grad-SAM and 0.994 for Rollout. For MCTVD, all three methods achieved virtually identical consistency scores close to 1.0.

Stability Stability results exhibited greater variation across encodings than the previous evaluation dimensions. Stability@20 denotes the average top-20 patch overlap across independently trained model instances, macro-averaged over malware families. Rollout achieved the highest Stability@20 values for Entropy (0.688), Gray (0.559), RGB (0.572), and Markov (0.507), whereas Grad-SAM performed best for MCTVD (0.575 vs. 0.377 for Rollout). Grad-CAM consistently produced the lowest stability scores, ranging from 0.209 to 0.376. Figure 5.8 further shows that stability generally increased with larger values of k, indicating greater agreement in the relevant patches identified by independently trained model instances. While Rollout maintained the highest stability across most

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

encodings, the relative ranking of the methods was not always constant across values of k. For MCTVD, Rollout initially achieved higher stability, peaking at approximately $k = 4$, after which stability gradually declined as larger sets of patches were considered. As a result, Grad-SAM achieved the highest Stability@20 score (0.575 vs. 0.377 for Rollout). For Markov, Rollout achieved substantially higher stability at $k = 1$ (0.550 vs. 0.201 for Grad-SAM). At $k = 20$, however, the difference decreased to 0.043 (0.507 vs. 0.464).

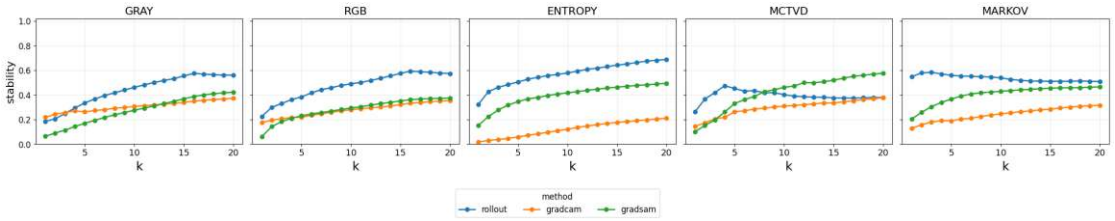


Figure 5.8: Stability across different values of k, averaged over malware families and three independently trained model instances.

Computational Efficiency Attention Rollout was consistently the most computationally efficient method, requiring 11.8–16.0 seconds per explanation compared to 28.6–37.5 seconds for Grad-CAM and Grad-SAM. While Grad-CAM and Grad-SAM exhibited overall similar runtimes, Grad-SAM was slightly slower for most encodings – despite not consistently outperforming Rollout on other evaluation metrics.

5.2.3 Family-Level Explainability Characteristics

We further investigated whether explainability characteristics vary across malware families by performing a family-level analysis for the Gray encoding. Gray was selected as a representative encoding because it achieved competitive classification and explainability performance, while the analysis was restricted to Rollout and Grad-SAM since both methods consistently outperformed Grad-CAM in the aggregate evaluation.

Table 5.8 reveals substantial differences in explainability across malware families. The families *Small* and *Wabot* generally achieved the strongest explainability characteristics for both Rollout and Grad-SAM, combining high insertion scores with low effective complexity and comparatively high stability. In contrast, *Berbew* exhibited the weakest explainability characteristics across both methods, achieving the lowest insertion scores, the highest effective complexity values, and the lowest stability scores. These results suggest that explanations for *Berbew* are less concentrated, less faithful, and less reproducible than for other families. *Upatre* and *CeeInject* showed similar tendencies to *Berbew*, although less pronounced.

While insertion, deletion, and effective complexity varied across families, neither Rollout nor Grad-SAM consistently outperformed the other in terms of faithfulness. Instead, the relative ranking of malware families remained largely consistent across both methods. Families such as *Sillyp2p*, *Small*, and *Sfone* achieved comparatively high stability, whereas

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 5.8: Per-family explainability results for the Gray encoding using Attention Rollout and Grad-SAM. Best and worst values for each metric are highlighted in bold and underlined, respectively. Cell shading indicates relative performance across malware families within each metric and explainability method using min–max normalization, with green indicating better and red indicating poorer relative performance.

Family	Rollout					Grad-SAM				
	Ins.↑	Del.↓	EC↓	Cons.↑	Stab.↑	Ins.↑	Del.↓	EC↓	Cons.↑	Stab.↑
Berbew	0.798	0.652	64.9	0.983	0.268	0.756	0.714	74.9	0.969	0.128
CeeInject	0.862	0.735	46.0	0.997	0.298	0.882	0.536	38.7	0.993	0.192
Dinwod	0.923	0.524	22.0	0.998	0.429	0.938	0.475	18.3	0.994	0.258
Ganelp	0.940	0.672	19.9	**0.999**	0.382	0.926	0.719	24.8	0.996	0.256
Mira	0.864	0.535	39.8	0.998	0.396	0.859	0.561	42.5	0.996	0.244
Sfone	0.883	0.517	35.4	**0.999**	0.756	0.925	0.485	22.9	0.998	0.647
Sillyp2p	0.854	**0.318**	9.0	0.973	**0.887**	0.852	**0.378**	12.1	0.913	**0.785**
Small	0.961	0.644	**3.2**	0.983	0.820	0.960	0.613	**5.4**	0.947	0.740
Upatre	0.904	0.636	29.5	**0.999**	0.277	0.839	0.657	48.6	**0.999**	0.214
Wabot	**0.973**	0.666	6.7	0.993	0.630	**0.967**	0.710	11.1	0.937	0.353
Wacatac	0.849	0.474	27.1	0.998	0.529	0.836	0.443	22.9	0.995	0.359

Berbew, *Upatre*, and *CeeInject* remained among the least stable. This suggests that explainability characteristics are influenced not only by the choice of explainability method but also by family-specific properties reflected in the Gray encoding.

We further analyzed whether explainability characteristics were associated with family-level classification performance. No clear relationship was observed. While some families, such as *Small* and *Wabot*, combined strong classification and explainability performance, others, including *Berbew* and *Upatre*, achieved high classification accuracy despite comparatively weak explainability characteristics. Therefore, similar classification performance did not necessarily correspond to similar explainability characteristics across malware families.

5.2.4 Feature-Space Visualization

While the explainability metrics evaluate the quality of individual explanations, they provide limited insight into how malware families are organized within the learned feature space. Figure 5.9 and Table 5.9 thus complement the previous analysis by visualizing and quantifying the learned feature representations of the Vision Transformer.

The visualizations reveal notable differences in the structure of the learned feature spaces. Gray produced the most compact and visually distinct family clusters, which is reflected by the highest silhouette score (0.626). Markov Bytes achieved a similarly structured feature space (0.607), although several families formed elongated cluster shapes. RGB obtained a slightly lower silhouette score (0.588) and exhibited more fragmented clusters, although family-specific regions remained visually distinguishable. Entropy MALFCS (0.561) produced more fragmented clusters and weaker boundaries between several malware

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

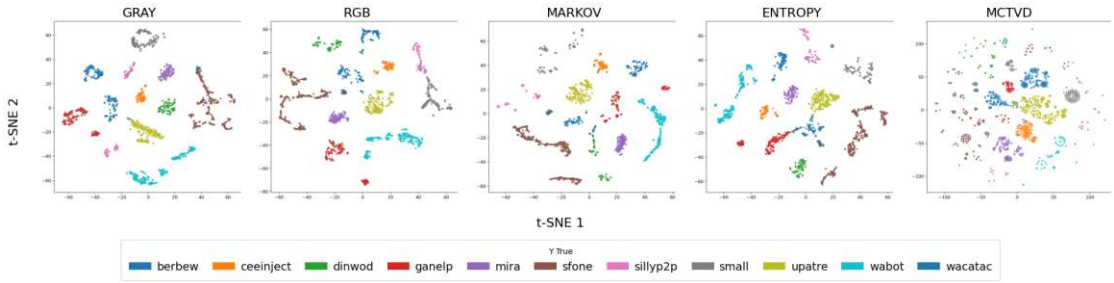


Figure 5.9: PCA+t-SNE projections of the learned feature representations for the five malware-to-image encodings included in the explainability evaluation. Each point corresponds to a test sample and is colored according to its malware family.

Encoding	**Silhouette Score**
Gray	0.626
Markov	0.607
RGB	0.588
Entropy	0.561
MCTVD	0.538

Table 5.9: Silhouette scores computed on the learned feature representations. Higher values indicate more compact and better separated malware-family clusters. The best result is highlighted in dark green.

families. MCTVD showed the least structured feature space, characterized by increased dispersion and reduced separation between families, resulting in the lowest silhouette score (0.538).

Interestingly, Gray and RGB, both derived directly from the raw byte stream and preserving the original byte ordering, exhibit noticeably different feature-space structures. While Gray produced more compact clusters and achieved a higher silhouette score, RGB generated more fragmented cluster shapes despite achieving comparable classification performance. As an important difference between both encodings is that RGB groups three consecutive bytes into color pixels, whereas Gray maps each byte to an individual grayscale pixel, the results suggest that seemingly minor design choices in the image construction process can influence the structure observed in the resulting t-SNE projections. However, these visual differences should be interpreted cautiously, as t-SNE projections can vary across initializations and do not preserve global geometry.

The visualizations provide complementary insight beyond classification metrics alone. While Gray, RGB, and Markov Bytes achieved similarly strong classification performance on BODMAS-11, their learned feature spaces exhibit noticeably different structures. In particular, Gray produced the most clearly separated clusters despite not achieving the strongest classification performance. This suggests that classification effectiveness and feature-space separability reflect related but distinct properties of the learned

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

representations. Encodings can therefore achieve similar predictive performance while exhibiting different family structures in their respective t-SNE projections.

5.2.5 Explainability Evaluation Analysis

A central finding is that the transformer-oriented Rollout and Grad-SAM methods consistently outperformed the ViT-adapted Grad-CAM baseline across nearly all evaluation dimensions. This observation is particularly noteworthy because existing Vision Transformer malware-classification studies that consider explainability primarily rely on Grad-CAM or qualitative attention visualizations [JLKC25, LZKN22]. While Grad-CAM produced substantially more diffuse explanations and achieved the lowest faithfulness and stability scores, Attention Rollout and Grad-SAM consistently generated more faithful, sparse, and stable explanations across all investigated encodings.

Although Attention Rollout and Grad-SAM achieved comparable performance for most evaluation dimensions, Attention Rollout provided the strongest overall trade-off between explanation quality and computational efficiency. Rollout generally achieved higher stability while generating explanations approximately two to three times faster than Grad-SAM. Consequently, Rollout emerged as the most practical explainability method among those evaluated.

The family-level analysis revealed substantial variation in explainability across malware families. The families *Small* and *Wabot* consistently achieved strong explainability characteristics, whereas *Berbew* and *Upatre* achieved comparatively weak results across multiple metrics. Interestingly, no clear relationship was observed between family-level classification performance and the quality of explainability. For example, *Berbew* achieved near-perfect classification performance despite exhibiting some of the weakest explainability characteristics.

The feature-space visualizations further showed that encodings with similar classification performance can produce noticeably different representation structures. In particular, Gray and RGB achieved comparable classification effectiveness despite differing in cluster compactness and separability. Furthermore, Gray achieved the highest silhouette score among all investigated encodings despite not achieving the strongest classification performance. These results indicate that feature-space structure and classification effectiveness represent related but distinct properties of the learned representations.

5.3 Reverse Traceability Analysis

While the previous section focused on the quality of explanations in image space, it did not address whether these explanations can be related to concrete program structures. This section, therefore, evaluates whether highly attributed image regions can be traced to executable artifacts in the original malware sample and interpreted within the context of malware analysis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The reverse-traceability evaluation was conducted using the grayscale encoding. Among the investigated representations, grayscale and RGB preserve a direct correspondence between image locations and byte offsets in the original executable, enabling reverse mapping from image-space explanations to executable artifacts. Grayscale was selected because it achieved competitive classification performance and remains the most widely used malware-image representation in the literature. As in the preceding analysis, the reverse-traceability evaluation was restricted to Attention Rollout and Grad-SAM due to their stronger overall explainability performance.

The remaining encodings were not included in the reverse-traceability analysis because they do not preserve the direct byte-level correspondence required by the proposed framework. Entropy aggregates information over byte windows, whereas Markov and MCTVD represent statistical relationships between bytes and opcodes rather than concrete executable locations. Consequently, highly attributed image regions cannot be directly resolved to specific instructions, functions, imported APIs, or PE sections.

5.3.1 Traceability Evaluation

The reverse-traceability evaluation investigated whether highly attributed image regions could be mapped back to executable artifacts at both the PE section and function levels. We further examined how mapping success varied across different Top-k explanation sets, where Top-k denotes the k highest-attributed image patches for each sample, and across malware families. Together, these analyses provide an assessment of the traceability capability of the proposed framework.

PE-Level Mapping Success

Before evaluating the traceability of explanation regions, we first analyzed how much of each malware image could theoretically be mapped back to executable artifacts. Figure 5.10 shows the distribution of mappable image coverage across the evaluation dataset. A patch was considered mappable if it overlapped at least one identified PE section in the original executable. Consequently, mappable image coverage is the proportion of image patches that can be linked to at least one executable artifact via the reverse-mapping pipeline.

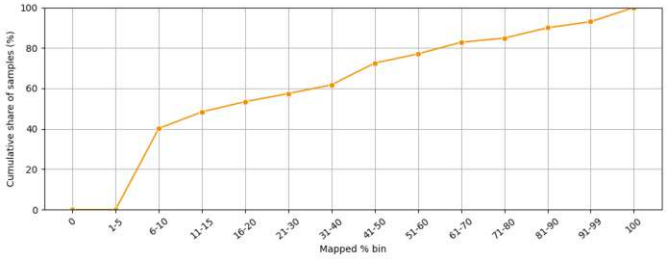


Figure 5.10: Cumulative distribution of mappable image coverage across the evaluation dataset.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The distribution is strongly skewed towards low coverage. The largest group of samples exhibits between 6% and 10% mappable coverage, and approximately 73% of samples have at most 50% of their image patches associated with executable artifacts. Only a small fraction of samples achieve complete coverage, accounting for approximately 7% of the dataset.

Despite the generally low proportion of mappable image regions, highly attributed explanation patches could be linked to executable artifacts in most cases. Table 5.10 summarizes the Top-1 mapping success rates for Rollout and GradSAM. Rollout consistently achieved higher mapping success (86.9% vs. 77.1%) and fewer unmapped regions (13.1% vs. 22.9%) than GradSAM. Notably, the high mapping success of explanation patches contrasts with the limited mappable coverage observed across many malware images. Furthermore, for both explanation methods, most mapped patches overlapped multiple PE sections (Rollout: 57.9%, GradSAM: 54.7%), indicating that highly attributed image regions often span multiple executable structures rather than isolated sections.

Table 5.10: Top-1 mapping success across all 2,767 test samples.

Method	**Mapped (%)**	**Unmapped (%)**
Rollout	86.9	13.1
GradSAM	77.1	22.9

We further examined how mapping success changed when larger sets of highly attributed patches were considered. Rather than evaluating only the single highest-ranked patch, the analysis was repeated for Top-k explanation patches with $k \in \{1, \ldots, 20\}$. For each k, mapping success represents the proportion of individually evaluated Top-k patches that could be mapped to at least one PE section, rather than cumulative mapping coverage. Figure 5.11 shows the resulting mapping success rates.Figure 5.11 shows the resulting mapping success rates.

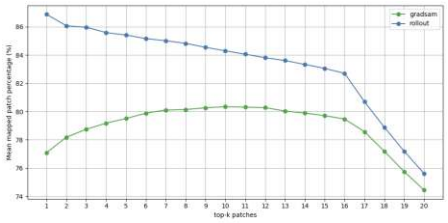


Figure 5.11: Top-k PE-level mapping success rates for Rollout and GradSAM.

Rollout achieved higher mapping success than GradSAM across all values of k. However, the gap between the methods decreased steadily as larger sets of explanation patches were included. While Rollout achieved a Top-1 mapping success rate of 86.9%, this value gradually declined as lower-ranked patches were considered. In contrast, GradSAM increased from 77.1% at Top-1 to approximately 80.3% around Top-10, then declined for larger values of k. For both methods, mapping success decreased more noticeably beyond

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Top-16, converging to similar rates at Top-20 (Rollout: 75.6%, GradSAM: 74.4%). These results show that mapping success remains relatively high beyond the single highest-ranked explanation patch, although it gradually decreases as lower-ranked patches are included.

Table 5.11: PE-level mapping success rates (%) by malware family for Rollout and GradSAM at Top-1 and Top-20.

	Top-1		Top-20	
Family	**Rollout**	**GradSAM**	**Rollout**	**GradSAM**
Berbew	41.3	16.8	33.3	33.4
CeeInject	76.0	64.8	74.0	69.7
Dinwod	71.1	39.2	57.6	52.3
Ganelp	65.3	35.9	49.6	49.4
Mira	100.0	84.0	85.5	80.0
Sfone	99.6	96.1	89.0	87.3
Sillyp2p	99.4	99.4	80.3	80.1
Small	100.0	99.7	81.9	81.8
Upatre	81.8	77.5	84.1	85.5
Wabot	100.0	93.9	82.4	81.5
Wacatac	80.8	76.4	76.2	75.8
Overall	**86.9**	**77.1**	**75.6**	**74.4**

To assess whether traceability is consistent across different malware types, we further analyzed mapping success at the family level. As shown in Table 5.11, mapping success varied substantially across malware families. At Top-1, several families, including *Mira*, *Small*, and *Wabot*, achieved perfect mapping success under Rollout, while *Sfone* and *Sillyp2p* exhibited near-perfect traceability. In contrast, *Berbew*, *Ganelp*, and *Dinwod* remained considerably more challenging to map. Across most families, Rollout achieved higher mapping success than GradSAM, although these differences became less pronounced at Top-20, where both methods converged to similar mapping rates for many families. Overall, the results indicate that traceability depends on both the explanation method and on characteristics of the underlying malware family. Across most families, Rollout achieved higher mapping success than Grad-SAM and was therefore selected for the remaining reverse-traceability analyses.

Function-level Mapping Success

While PE-level mapping establishes whether an explanation can be linked to any region of the executable, it does not indicate whether the highlighted region can be resolved to finer-grained executable artifacts. We therefore investigated the extent to which the traced regions could be associated with individual functions.

Because individual explanation patches often corresponded to large numbers of traced source ranges (median: 173, mean: 912), function-level traceability was evaluated using

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

patch-weighted statistics. Specifically, resolution categories were first computed for each explanation patch and subsequently averaged across patches.

Table 5.12 summarizes the resulting resolution outcomes for Rollout explanations. Across all Top-k settings, approximately 13-14% of traced ranges could be associated with a function, while around 40% could only be resolved to a PE section. The remaining 44-47% could not be linked to either. Interestingly, the highest-ranked explanation patches exhibited slightly higher function-level resolution rates and lower proportions of unresolved regions than the larger Top-10 and Top-20 patch sets. This suggests that the most highly attributed regions are more likely to correspond to recoverable executable artifacts.

Table 5.12: Patch-weighted function-level traceability for Rollout explanations.

Resolution Category	**Top-1 (%)**	**Top-10 (%)**	**Top-20 (%)**
Function match	14.13	12.76	12.76
PE section match only	41.45	40.03	40.03
Unresolved	44.42	47.21	47.21

Function-level traceability varied considerably across malware families (cf. Table 5.13). While nearly half of the traced regions in *Mira* could be resolved to specific functions (49.52%), function-level resolution remained below 6% for *Berbew*, *Sillyp2p*, and *Small*. In these families, traced regions could often be linked to PE sections but not to individual functions. Overall, the results indicate that, as expected given the finer granularity of function-level mapping, PE-level traceability does not necessarily translate into function-level traceability and that the recoverability of fine-grained executable artifacts differs substantially across malware families.

Table 5.13: Function-level traceability of Top-1 Rollout explanations across malware families (%).

Family	**Function match**	**PE Only**	**Unresolved**
Berbew	0.02	4.96	95.02
CeeInject	5.00	66.36	28.64
Dinwod	21.89	27.43	50.68
Ganelp	10.12	50.56	39.32
Mira	49.52	47.65	2.82
Sfone	24.29	55.84	19.87
Sillyp2p	2.65	8.41	88.94
Small	5.47	20.56	73.98
Upatre	13.82	61.86	24.32
Wabot	10.45	29.90	59.65
Wacatac	5.51	58.58	35.91
Overall	**14.13**	**41.45**	**44.42**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Traceability Evaluation Analysis

The traceability evaluation demonstrates that highly attributed image regions can be mapped back to executable artifacts. While many malware images contained only a limited proportion of mappable patches overall, the highest-ranked explanation patches were successfully mapped to PE sections in most cases. Mapping success was highest for the Top-1 explanation patch and remained comparatively high across larger Top-k explanation sets and most malware families, demonstrating that traceability extends beyond a single explanation region. This indicates that explanations are concentrated in executable regions that can be recovered through the proposed reverse-mapping framework. Importantly, the high mapping success of explanation patches contrasts with the generally low mappable coverage of complete malware images, suggesting that the framework identifies executable artifacts in a meaningful manner rather than recovering them by chance.

Function-level mapping proved substantially more challenging than PE-level mapping. Although more than half of the traced explanation regions could be resolved to at least a PE section, only 14.13% could be further associated with specific functions. These results demonstrate that the proposed framework supports reliable traceability at the PE section level while also enabling finer-grained function-level resolution for a subset of highly attributed regions.

Rollout consistently achieved higher mapping success than Grad-SAM. Interestingly, while the explainability evaluation indicated only modest differences between the two methods on grayscale images, the traceability evaluation revealed a substantially larger gap in mapping success. Consequently, the remaining analyses focus exclusively on Rollout explanations.

5.3.2 Explanation Interpretability

The traceability evaluation established that highly attributed explanation regions can be linked to executable artifacts. The next aspect investigated by the reverse-traceability evaluation was which executable structures are associated with these explanations. To answer this question, we analyzed the PE sections and functions associated with highly attributed regions and examined whether consistent attribution patterns emerged across malware families. Together, these analyses provide insight into the executable structures that contribute most strongly to the model's predictions.

Unless stated otherwise, the following analyses are based on the Top-1 Rollout explanation patch. For each sample, the proportion of traced bytes associated with a given executable artifact was computed relative to the total number of traced bytes in that sample. Family-level percentages were then obtained by averaging these proportions across all samples belonging to the respective malware family. Consequently, the reported values reflect the average share of traced bytes attributed to a particular executable artifact rather than the frequency with which the artifact occurs. Section-level percentages are computed only from mapped regions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

PE section Attribution Patterns

We first analyzed the PE sections associated with the Top-1 Rollout explanation patches. Figure 5.12 shows the distribution of traced bytes across the three most prominent PE sections for each malware family.

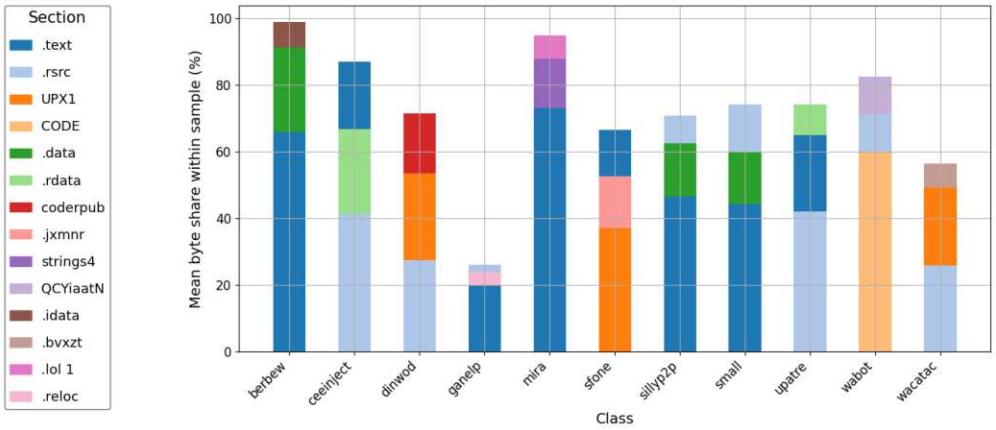


Figure 5.12: Top three PE sections receiving the largest share of traced bytes for each malware family based on Top-1 Rollout explanations.

Clear differences emerged across malware families. Several families were dominated by the `.text` section, including *Mira* (73.1%), *Berbew* (66.0%), *Sillyp2p* (46.4%), and *Small* (44.3%). In contrast, *CeeInject* (41.3%), *Upatre* (42.0%), *Dinwod* (27.5%) and *Wacatac* (25.9%) exhibited the strongest attribution to the `.rsrc` section. Distinct family-specific patterns were also observed. For example, Sfone was primarily associated with the packed section `UPX1` (37.1%), while *Wabot* was dominated by the `CODE` section (59.9%).

Several similarities between malware families are visible. In particular, *Small* and *Sillyp2p* exhibit highly similar section-level attribution profiles, with both families showing dominant attribution to `.text` followed by contributions from `.data` and `.rsrc`. Likewise, *CeeInject*, *Upatre*, and *Wacatac* place greater emphasis on resource-related sections. In contrast, *Dinwod* combines attribution to the `.rsrc` section with substantial contributions from packed and custom sections, namely `UPX1` and `coderpub`. Distinct family-specific patterns are further observed for *Sfone* and *Wabot*, which are dominated by the `UPX1` and `CODE` sections, respectively.

Function Attribution Patterns

We then analyzed which functions were most frequently associated with Top-1 Rollout explanation patches. For each sample, functions were ranked by how often they appeared among the traced source ranges in the top-1 patch scope. Family-level percentages, therefore, indicate the proportion of samples in which a function appeared at a given rank. Table 5.14 summarizes the three most frequently occurring top-ranked functions within each malware family.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 5.14: Three most frequently occurring top-ranked functions for each malware family based on Top-1 Rollout explanations. Percentages indicate the proportion of samples in which the respective function appeared at the given rank.

Family	Top-1 Function	Top-2 Function	Top-3 Function
Berbew	fcn.00430003 (29.4%)	fcn.00430004 (22.8%)	int.00430003 (0.7%)
CeeInject	entry0 (33.3%)	fcn.004020cd (29.6%)	fcn.004022e0 (29.6%)
Dinwod	fcn.00427005 (38.5%)	fcn.0042777a (33.8%)	fcn.0042b1cb (9.5%)
Ganelp	entry0 (23.1%)	fcn.0040caae (5.3%)	fcn.00408080 (2.4%)
Mira	fcn.00407ec0 (73.4%)	fcn.004015fe (72.7%)	fcn.00407480 (73.4%)
Sfone	entry0 (72.1%)	fcn.00408c30 (20.6%)	fcn.00407e10 (20.0%)
Sillyp2p	fcn.00403db4 (53.6%)	fcn.00401577 (53.6%)	fcn.004037b0 (43.0%)
Small	fcn.00403db4 (89.5%)	fcn.00401577 (89.5%)	fcn.00402936 (81.1%)
Upatre	entry0 (59.4%)	fcn.00401000 (8.1%)	main (1.3%)
Wabot	fcn.00405cdc (70.7%)	fcn.0040747c (70.1%)	fcn.00407c9c (32.7%)
Wacatac	entry0 (65.9%)	fcn.0040ca16 (5.3%)	int.0040af76 (4.1%)

Several recurring patterns emerged. The executable entry point (`entry0`) was the most frequently highlighted function for *Sfone* (72.1%), *Wacatac* (65.9%), *Upatre* (59.4%), and *CeeInject* (33.3%). In contrast, *Small* and *Sillyp2p* shared the functions `fcn.00403db4` and `fcn.00401577` as their most frequently highlighted functions, suggesting similarities in the executable structures emphasized by the model. Likewise, *Mira* and *Wabot* exhibited highly recurring family-specific functions that appeared in more than 70% of samples. The results indicate that explanations repeatedly map to a small set of recurring functions within each malware family.

To assess the robustness of these patterns, we additionally examined the number of unique functions associated with each explanation region. Three distinct patterns emerged. Families such as *Sfone*, *Upatre*, *Ganelp*, and *Wacatac* exhibited highly concentrated mappings, with a median of one recovered function per explanation region. *Berbew*, *CeeInject*, and *Dinwod* showed moderately richer mappings (median = 2). In contrast, *Mira*, *Small*, *Sillyp2p*, and *Wabot* exhibited substantially richer function-level mappings, with median values ranging from 45 to 81 unique functions per explanation region.

The concentrated mappings observed for some families do not necessarily imply that explanations always resolve to a single function. For example, although 64.5% of *Sfone* explanation regions contained only one recovered function, `fcn.00408c30` and `fcn.00407e10` still appeared as the second- and third-ranked functions in approximately 20% of samples. Conversely, *Mira* and *Wabot* exhibited recurring non-entry functions despite substantially richer function-level mappings. Together, these observations suggest that recurring function-level patterns are not solely driven by sparse mappings, although families dominated by `entry0` provide less specific insight into the executable structures highlighted by the model.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Interpretability Evaluation Findings

The interpretability evaluation revealed clear family-specific attribution patterns at the PE section level. Different malware families consistently emphasized different executable structures, ranging from standard sections such as `.text` and `.rsrc` to packed or family-specific sections such as `UPX1` and `CODE`. Similar section-level attribution profiles were also observed for certain families, such as *Small* and *Sillyp2p*, suggesting that the model captures common executable characteristics shared across malware families.

Function-level analyses provided a more nuanced picture. Recurring function-level attribution patterns were observed for several families, including recurring non-entry functions in *Mira*, *Wabot*, *Small*, and *Sillyp2p*. However, these findings must be interpreted in the context of the traceability results, which showed that only a subset of explanation regions could be resolved to individual functions. Consequently, the strongest interpretability evidence is provided at the PE section level, while function-level analyses demonstrate the potential for finer-grained interpretation when sufficient function-level traceability is available.

5.4 Reverse Engineering Case Studies

While the previous analyses examined interpretability at the dataset and malware-family levels by identifying recurring attribution patterns across executable artifacts, they do not establish whether these artifacts are actually relevant to individual model predictions. The final question addressed, therefore, was: *Are the recovered artifacts relevant to the model's decision process?* To answer this question, we conducted two reverse-engineering case studies combining function-level perturbation experiments with manual reverse-engineering analysis of the recovered functions.

5.4.1 Case Study 1: *Small*

The selected sample was correctly classified as *Small* with near-perfect confidence. Tables 5.15 and 5.16 summarize the sample and the highest-attribution explanation patch. Although only 0.79% of the whole executable could be mapped to PE sections, the Top-1 patch contained recoverable PE section and function-level artifacts that formed the basis for the subsequent analysis.

Function-Level Perturbation Analysis

Across all recovered artifacts, 115 unique functions were identified. Restricting the analysis to the Top-3 and Top-1 patches reduced the candidate set to 94 and 62 functions, respectively, while the instruction-confirmed subsets contained 42 and 27 functions. Functions overlapping the recovered source ranges were then ranked according to their overlap-weighted contribution score. Table 5.17 presents the highest-ranked functions recovered from the Top-1 patch. Several of the highest-ranked functions were supported

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.15: Sample-level summary.

Property	Value
Family	*Small*
SHA-256	`ea2f4b72...a225376`
Confidence	0.999998
Size	5.16 MB
PE bytes	40,960
Coverage	0.79%
PE sections	.text, .data, .rdata, .rsrc

Table 5.16: Top-1 patch traceability summary.

Property	Value
Source-range span	20,144 bytes
Function match	7.23%
PE section only	5.48%
Unresolved	87.29%

by recovered instruction records, while others were identified solely through source-range overlap.

Table 5.17: Highest-ranked functions recovered from the Top-1 explanation patch. Functions are ordered by overlap-weighted contribution score.

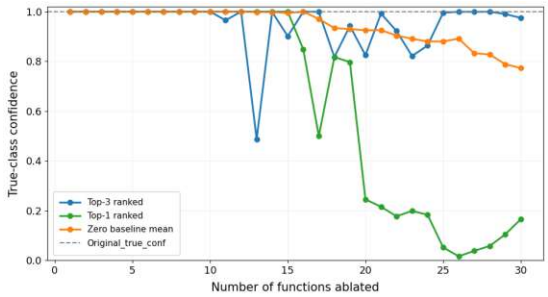
Function	**Overlap Bytes**	**Source Ranges**	**Instr. Count**
fcn.00403db4	112	7	0
fcn.00401577	64	4	18
fcn.00402936	58	4	21
fcn.00402c61	48	3	21
fcn.004037b0	48	3	11
fcn.004052c0	48	3	0
fcn.0040592f	48	3	0
fcn.00406612	38	3	0

The highest-ranked function (`fcn.00403db4`) was supported exclusively through overlap evidence, whereas four of the next five highest-ranked functions were instruction-confirmed. This distinction enabled subsequent comparison between perturbations based on the full ranking and perturbations restricted to instruction-confirmed functions.

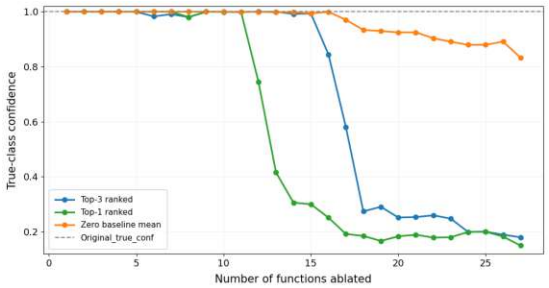
Perturbation of Ranked Functions As shown in Figures 5.13a and 5.13c, cumulative ablation of functions recovered from the highest-attribution patch (Top-1) produced substantially larger reductions in true-class confidence than either the broader Top-3 ranking or the random-function baselines. Under zero-byte perturbation, the predicted class first changed after 17 ablated functions, whereas under random-byte perturbation, the first class change occurred after 25 ablated functions. In both cases, the sample was reassigned from *Small* to *Sillyp2p*. Although the Top-3 ranking also produced noticeable reductions in confidence, these effects were less persistent, and confidence recovered substantially as additional functions were ablated. In contrast, confidence remained low following perturbation of the Top-1 ranking, suggesting that prediction-relevant evidence is concentrated within functions recovered from the single highest-attribution patch.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

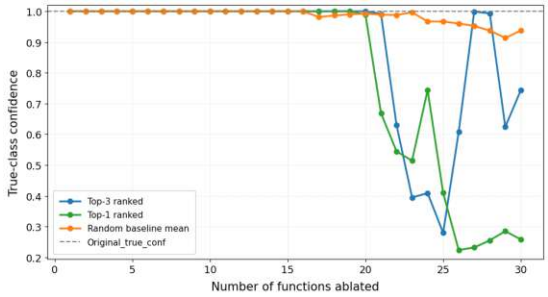
Figure 5.13: Function-level perturbation results. The left column compares Top-1 and Top-3 rankings against random-function baselines, while the right column restricts the analysis to instruction-confirmed functions. Rows correspond to zero-byte and random-byte perturbations, where the bytes associated with the selected functions are replaced with zeros or random byte values, respectively.



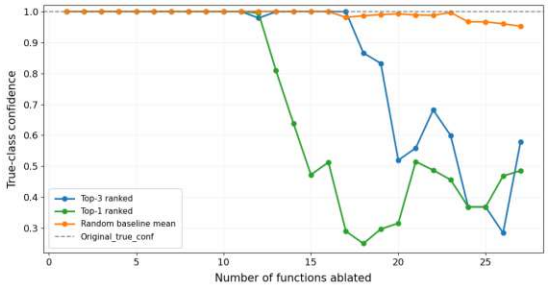
(a) Full ranking (zero-byte).



(b) Instruction-confirmed (zero-byte).



(c) Full ranking (random-byte).



(d) Instruction-confirmed (random-byte).

Restricting the analysis to instruction-confirmed functions further strengthened the perturbation effect (Figures 5.13b and 5.13d). For the Top-1 ranking, the first prediction change occurred after only 13 ablated functions under zero-byte perturbation and after 15 ablated functions under random-byte perturbation, compared with 17 and 25 functions, respectively, for the unrestricted Top-1 ranking. The confidence reduction also remained substantial throughout the remainder of the ablation process. In contrast, the instruction-confirmed Top-3 ranking required more perturbations before inducing a prediction change and generally produced weaker effects. These results suggest that the prediction-relevant evidence is concentrated in a small subset of functions supported by direct instruction-level traceability evidence and originating from the highest-attribution explanation patch.

Ranking Sensitivity As an additional control, the ordering of the top-27 ranked functions recovered from the Top-1 patch was randomized ten times. Under random-byte perturbation, the original ranking induced the first prediction change after 15 ablated functions, compared with an average of 23.5 functions for the randomized orderings. The ranked order also produced substantially more prediction changes overall (11 versus

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.8 on average). Under zero-byte perturbation, the first prediction change occurred after 13 ablated functions for the ranked order and after 13.5 functions on average for the randomized orderings. The ranked order nevertheless produced slightly more prediction changes overall (15 versus 13.6 on average). These findings suggest that the overlap-weighted ranking contains information beyond function-set membership alone, although the effect is considerably stronger under random-byte perturbation.

Table 5.18: Window-ablation experiments for the instruction-confirmed Top-1 ranking. Δ confidence denotes the change in true-class confidence relative to the original sample. Prediction flips indicate reassignment from *Small* to *Sillyp2p*.

Function subset	**Zero-byte**		**Random-byte**	
	Δ conf.	Flip	Δ conf.	Flip
Ranks [10–15]	-0.0001	No	-0.0453	No
Ranks [1–9]	-0.0000	No	-0.0000	No
Ranks [16–27]	-0.0001	No	0.0000	No
Ranks [1–15]	-0.7006	Yes	-0.5276	Yes
Ranks [10–15] + [16–27]	-0.5761	Yes	-0.0069	No

Window-Ablation Analysis To determine whether the sharp confidence reduction observed around ranks [10–15] was driven by the functions within this window alone, additional window-ablation evaluation was performed (Table 5.18). Perturbing ranks [10–15] in isolation produced only a modest confidence reduction under random-byte perturbation ($\Delta = -0.045$) and a negligible reduction under zero-byte perturbation, without changing the predicted class. Likewise, perturbing ranks [1–9] or [16–27] alone had almost no effect. In contrast, jointly perturbing ranks [1–15] reduced the true-class confidence by 0.53 and 0.70 under random-byte and zero-byte perturbation, respectively, causing the prediction to change from *Small* to *Sillyp2p* in both cases. Although perturbing ranks [10–15] together with ranks [16–27] also produced a substantial confidence reduction under zero-byte perturbation ($\Delta = -0.576$), the corresponding random-byte perturbation caused only a minor confidence change ($\Delta = -0.007$). Since random-byte perturbations are less intrusive, because they preserve byte-level variability rather than replacing entire function regions with uniform zero values, these results suggest that the confidence collapse observed during cumulative ablation is not driven by the ranks [10–15] window alone, but primarily emerges from its interaction with the higher-ranked functions contained in ranks [1–9].

Consistency of Recovered Functions Across Samples

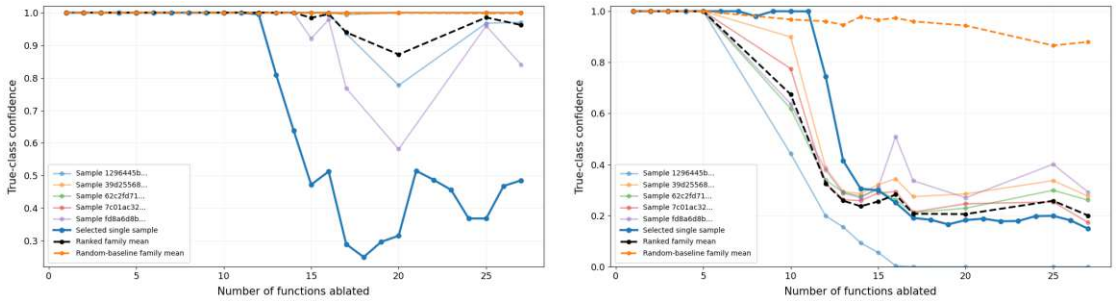
To assess whether the recovered functions represent sample-specific artifacts or recurring *Small*-family structures, the reverse-traceability analysis was repeated for five additional correctly classified *Small* samples. The recovered Top-1 function rankings were then compared across samples.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 5.19: Top-1 patch traceability composition across five additional *Small* samples. Values denote the percentage of Top-1 patch bytes assigned to each resolution category.

Sample	Span (bytes)	Function	PE only	Unresolved
1296445b...cc7b	12,736	11.43%	8.67%	79.90%
39d25568...2674	42,944	3.39%	2.57%	94.04%
62c2fd71...7542	46,544	3.13%	2.37%	94.50%
7c01ac32...9f6b	41,072	3.54%	2.69%	93.77%
fd8a6d8b...49ce	43,216	3.37%	2.55%	94.08%

As shown in Table 5.19, the Top-1 patch span varied substantially across the five additional *Small* samples, ranging from 12,736 to 46,544 bytes. Despite this variation, each sample contained the same amount of recoverable evidence at the function-level and PE section-level. Consequently, the proportion of function-resolved bytes ranged from 3.13% to 11.43%, whereas the majority of patch bytes remained unresolved. Despite these differences in traceability coverage, the instruction-confirmed Top-1 rankings were identical across all five samples, with no rank-order mismatches observed.



(a) Random-byte perturbation.

(b) Zero-byte perturbation.

Figure 5.14: Family-level perturbation results for five additional *Small* samples. Curves show cumulative ablation of the instruction-confirmed Top-1 function ranking. The dashed black line denotes the mean confidence trajectory across samples, while the dashed orange line denotes the corresponding random-function baseline mean.

The perturbation results in Figures 5.14a and 5.14b demonstrate that these recurring rankings were associated with consistent prediction effects across the family. Under zero-byte perturbation, all five samples exhibited substantial reductions in true-class confidence, with the first prediction change occurring after 10–12 ablated functions. In contrast, the corresponding random-function baselines produced considerably weaker effects, maintaining substantially higher confidence throughout most of the ablation process. The ranked zero-byte perturbations reduced the minimum true-class confidence to values between 0.00 and 0.27 across the five samples, whereas the corresponding random-function baselines under zero-byte perturbation maintained minimum confidence values between 0.70 and 0.97. Under random-byte perturbation, confidence reductions

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 5.20: Top-15 instruction-confirmed functions recovered from the highest-attribution patch.

Rank	Function	Category	Description
1	`fcn.00401577`	File propagation	Recursive filesystem traversal and target discovery
2	`fcn.00402936`	Runtime library	Heap memory deallocation routine
3	`fcn.00402C61`	Runtime library	Heap memory allocation routine
4	`fcn.004037B0`	Runtime library	`memcpy` implementation
5	`fcn.00403301`	File I/O	File opening and access handling
6	`fcn.004035FE`	File I/O	File write routine
7	`fcn.00401000`	Program orchestration	Main execution and propagation control
8	`fcn.004011EB`	File propagation	File replication and copying routine
9	`fcn.00401FC2`	P2P propagation	IMesh registry manipulation
10	`fcn.00401D27`	Data management	Linked-list sorting and record reordering
11	`fcn.00402200`	Runtime library	`strlen` implementation
12	`fcn.0040249E`	File I/O	`fread` implementation
13	`fcn.0040301B`	Runtime library	Heap allocation support routine
14	`fcn.0040322F`	File I/O	File buffer flushing routine
15	`fcn.0040196B`	File selection	File-extension filtering routine

were generally weaker but still exceeded those observed when the same number of functions was perturbed in randomly sampled function orders. Together, these results indicate that the recovered functions capture recurring family-level characteristics relevant to the model's predictions, supporting the interpretation that they reflect recurring family-level structure rather than only a single-sample artifact.

Manual Inspection of Recovered Functions

In this section, we present the results of our manual inspection of the highest-ranked instruction-confirmed functions. Table 5.20 summarizes the top-15 recovered functions and their inferred functionality. The analysis revealed several coherent groups of malware-related behavior.

- **File propagation and replication.** The most prominent malware-related group concerns file propagation and replication. The highest-ranked function, `fcn.00401577` (rank 1), performs recursive filesystem traversal using the functions `FindFirstFileA` and `FindNextFileA`, while `fcn.004011EB` implements file-copying logic. Function `fcn.0040196B` performs file-extension filtering and therefore appears to support the selection of target files during propagation. Detailed reverse-engineering evidence for the principal propagation routines is provided in Appendix 6.3.
- **Peer-to-peer manipulation.** Function `fcn.00401FC2` (rank 9) modifies IMesh registry settings, including entries associated with shared-content configuration.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This behavior is consistent with peer-to-peer propagation mechanisms commonly observed in file-sharing worms.

- **Control and data management.** Not all highly ranked functions directly implement malware actions. Function `fcn.00401000` (rank 7) acts as a central orchestration routine that coordinates traversal and propagation-related operations. Function `fcn.00401D27` (rank 10) traverses and reorders an internal linked-list structure, indicating a supporting role in organizing or prioritizing propagation-related records. Although these routines do not directly manipulate files or registry settings, they form part of the infrastructure required by the propagation workflow.

- **File-handling and runtime support.** Several other highly ranked functions correspond to file-handling operations, including file opening, reading, writing, and buffer flushing (`fcn.00403301`, `fcn.004035FE`, `fcn.0040249E`, `fcn.0040322F`). While these operations are not inherently malicious, their occurrence among the highly attributed functions places them within the broader propagation-related behavior identified for this sample. Finally, multiple recovered functions correspond to standard runtime-library components, such as memory allocation, deallocation, string processing, and memory-copy operations (`fcn.00402936`, `fcn.00402C61`, `fcn.0040301B`, `fcn.004037B0`, and `fcn.00402200`).

Overall, the recovered functions suggest a propagation strategy in which the malware identifies candidate files and directories, replicates itself under new filenames, and leverages peer-to-peer sharing mechanisms for distribution. The highest-ranked functions correspond to key stages of this process, including filesystem traversal, file selection, file copying, and IMesh configuration. This interpretation is consistent with the dynamic-analysis results reported by VirusTotal for the sample, which indicate file-replication and propagation-related activity [Vir20].

5.4.2 Case Study 2: *Ganelp*

While the *Small* family was selected as an example of strong and consistent function-level traceability, *Ganelp* was chosen as a contrasting case. The quantitative evaluation showed that explanations for *Ganelp* were generally less consistent across samples and more frequently remained unresolved at the function level. At the same time, recoverable artifacts were still present, making the family suitable for examining how our framework behaves when only partial function-level traceability is achievable. To illustrate this contrast, two representative *Ganelp* samples were selected. Sample A represents a case where substantial function-level recovery was achieved, whereas Sample B illustrates a case where no Top-1 traceability was achieved. Comparing these samples provides insight into how differences in traceability quality affect the resulting explanations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

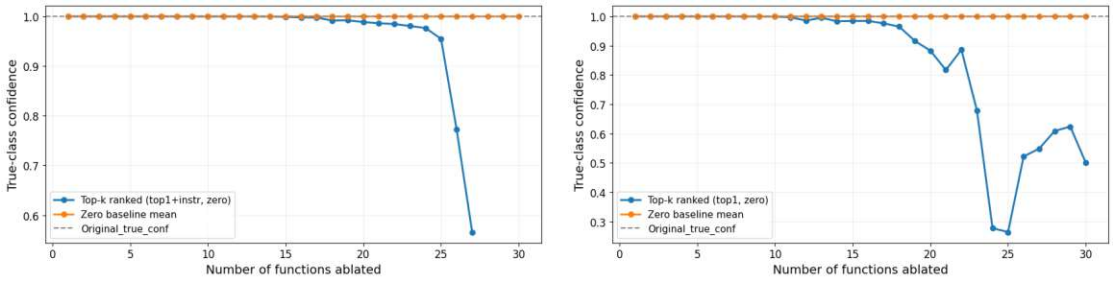
Table 5.21: Top-1 traceability comparison for two representative *Ganelp* samples.

Property	Sample A	Sample B
SHA-256	`7ec952de...8697c`	`03ca0fb0...228eb`
Top-1 PE mapping	100.0%	0.0%
Top-1 functions recovered	113	0
Function-resolved bytes	37.26%	0.0%
PE section-only bytes	48.74%	0.0%
Unresolved bytes	13.99%	100.0%
Dominant section	`.text` (100.0%)	–

Sample A: Partial Function-Level Recovery

Sample A represents a case where substantial recovery was achieved despite the overall lower traceability observed for the *Ganelp* family. As shown in Table 5.21, the highest-attribution regions could be linked to both PE-level structure and a large number of recovered functions, with only 13.99% of attributed bytes remaining unresolved. All section-level mappings were associated with the `.text` section, indicating that the model primarily focused on executable code regions.

To assess whether the recovered functions contribute to the model's decision, function-level ablation was performed on the Top-1 attribution using the same procedure as for the *Small* case study. As shown in Figure 5.15, in both the instruction-constrained and unrestricted rankings, true-class confidence decreased as progressively larger portions of the recovered function set were removed. The effect was more pronounced in the unrestricted setting, where the prediction changed after 24 recovered functions had been ablated. These results indicate that the recovered functions collectively capture information that is important for the classification decision.



(a) Instruction-confirmed Top-1 ranking. (b) Unrestricted Top-1 ranking.

Figure 5.15: Function-level perturbation results for *Ganelp* Sample A under zero-byte perturbation. The plots show the change in true-class confidence as progressively more ranked functions are ablated, compared with the random-function baseline.

Because the unrestricted Top-1 ranking produced the largest confidence reduction during ablation and was the only setting that eventually changed the prediction, its Top-25 re-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

covered functions were selected for manual analysis. Inspection of these functions revealed several recurring categories. Dynamic API-resolution and PE-processing functionality was particularly prominent, including `fcn.00402560` (unrestricted rank 8; instruction-confirmed rank 2), which resolves API addresses at runtime using custom lookup logic, and `fcn.004041a0` (ranks 9/3), which performs PE- and Component Object Model (COM)-related processing, a Windows framework for communication between software components. In addition, `fcn.00401c60` (ranks 17/4) implements a sliding-window decompression routine consistent with unpacking behavior, while `fcn.00402a80` (ranks 18/5) performs recursive filesystem traversal. Network-related functionality was also recovered through `fcn.00402d80` (ranks 19/6), which dynamically resolves and invokes FTP-related APIs. Several additional recovered functions corresponded to runtime support, including memory management, heap allocation, and program-initialization routines.

Comparison of the unrestricted and instruction-confirmed rankings revealed that all behaviorally relevant functions appeared in both rankings. However, in the unrestricted ranking, they were positioned substantially lower (ranks 8–19) and preceded by numerous CRT runtime, heap management, and debugging routines. Of the Top-25 unrestricted functions, 18 corresponded primarily to generic execution-support infrastructure. Instruction-level confirmation promoted the behaviorally relevant functions to ranks 2–6, resulting in a more interpretable ranking.

Together, these findings suggest that the model's decision is driven by functionality related to unpacking, API resolution, filesystem processing, and network communication, providing a meaningful explanation for why the sample was classified as *Ganelp*.

Sample B: Attribution Without Function-Level Mapping

Sample B represents a limited-traceability case. As shown in Table 5.21, the Top-1 patch could not be mapped to PE sections or recovered functions. The same was true for the Top-3 patches. Consequently, the highest-attribution regions remained entirely unresolved at a function level. However, expanding the attribution scope beyond the single highest-ranked patch revealed a small amount of recoverable structure. At the Top-10 level, one function became recoverable, and a small fraction of the recoverable section-level evidence (1.7%) was associated with the `petite` section, indicating that traceability was not completely absent but concentrated outside the strongest attribution regions.

One possible explanation for the limited traceability is the prevalence of packing within the *Ganelp* family. Previous work reported that approximately 76.6% of analyzed *Ganelp* samples in the BODMAS-11 dataset were packed using the Petite packer [GOMdR25]. Consistent with this observation, Sample B exhibited evidence of Petite packing: the small amount of recoverable section-level evidence was associated with a section named `petite`, consistent with the use of this packer. Moreover, the single function recovered at the Top-6 patch (`fcn.0040CA16`) appears to implement an unpacking routine that reconstructs

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

code or data from a packed region of the executable into dynamically allocated memory. Together, these observations suggest that packing contributes to the limited amount of statically recoverable program structure available for analysis. However, establishing whether packing is systematically associated with reduced traceability would require a dataset-wide comparison of packed and unpacked samples. Such an evaluation is further complicated by the variety of packers and the absence of a single unpacking approach that can reliably handle all packing schemes, placing a systematic packed-versus-unpacked comparison beyond the scope of this analysis. Nevertheless, the traceability analysis still provides insight into the model's behavior. The highest-attribution patches contained no padding-like bytes, and even when the analysis was expanded to the Top-10 patches, only 2.9% of unresolved bytes corresponded to common padding tokens (`0x00`, `0x90`, or `0xCC`). In contrast, approximately 6.2% of the complete binary consists of such padding-like bytes. Although the attributed regions could not be fully mapped to functions or PE sections, the framework shows that the model's attention is concentrated on non-trivial byte patterns rather than padding or empty regions of the binary. Importantly, the absence of function-level recovery does not imply an absence of explanation. Although semantic interpretation remains limited, the framework still identifies the regions underlying the model's prediction, revealing that the model focuses primarily on non-padding regions and providing packing-related evidence through the `petite` section and the recovered unpacking routine, which offers substantially more insight than a black-box classification alone.

5.4.3 Reverse Engineering Case Studies Analysis

The perturbation experiments in both case studies showed that ablating recovered functions reduced model confidence and, in several cases, changed the predicted class. This suggests that the recovered artifacts are not merely correlated with attribution regions but contribute to the model's decision process. In the *Small* case study, perturbing instruction-confirmed functions consistently produced the strongest effects, indicating that instruction-level traceability helps isolate prediction-relevant functionality.

The two families exhibited substantially different levels of traceability. For *Small*, highly consistent function-level recovery was achieved across multiple samples, and the same instruction-confirmed functions repeatedly appeared among the highest-ranked artifacts. In contrast, *Ganelp* frequently contained attribution regions that could not be mapped to recovered functions, likely due to packing and obfuscation. These results indicate that the effectiveness of reverse traceability depends on the amount of recoverable program structure present in the analyzed binaries.

The comparison between unrestricted and instruction-confirmed rankings demonstrated that instruction-level confirmation can improve the quality of explanations. In the *Ganelp* sample, behaviorally relevant functions related to API resolution, unpacking, filesystem processing, and network communication appeared in both rankings, but instruction-level confirmation promoted these functions above generic runtime-library routines. This

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

resulted in rankings that were more directly interpretable from a reverse-engineering perspective.

When sufficient traceability is available, manual inspection of the recovered functions demonstrated that the explanations correspond to recognizable malware behavior. For *Small*, the recovered functions directly described propagation and peer-to-peer distribution mechanisms. For *Ganelp*, the recovered functions emphasized unpacking, API resolution, filesystem processing, and communication-related functionality. Even when complete payload behavior could not be recovered, the framework still identified execution mechanisms that plausibly contributed to the model's classification decision.

5.4.4 Efficiency

The proposed reverse-traceability framework has the potential to improve the efficiency of malware analysis by automatically prioritizing executable artifacts for further inspection. Rather than requiring analysts to examine all recovered functions in a sample, the framework identifies and ranks the functions most strongly associated with the model's prediction. For example, in the *Small* case study, the framework reduced the search space from 115 recovered functions to 27 instruction-confirmed candidate functions, enabling the subsequent manual reverse-engineering analysis to focus on the highest-ranked candidates. Similarly, in the *Ganelp* case study, 190 recovered functions were reduced to a ranked set of candidate functions, with the detailed manual analysis focusing on the Top-25 highest-ranked functions. Beyond reducing the search space, the recovered PE sections and functions provide concrete evidence to support malware family attribution and to document analysis results. However, the practical efficiency gains of the proposed framework were not evaluated quantitatively – assessing whether the framework reduces analysis time would require a controlled user study and is therefore beyond the scope of this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Discussion

In this chapter, we highlight the main contributions and insights of this thesis, followed by a discussion of the research questions introduced in Section 1.2. Finally, we outline possible directions for future work.

6.1 Insights and Contributions

In this thesis, we investigated how different malware-to-image encoding strategies influence malware family classification with Vision Transformers and the explainability of their predictions. We systematically compared representative malware-to-image encodings under identical experimental conditions. Beyond predictive performance, we evaluated transformer-specific explainability methods using quantitative and qualitative analyses. Finally, we proposed a reverse-traceability framework that links image-based explanations back to executable artifacts and validated the recovered functionality through perturbation experiments and reverse-engineering case studies.

In particular, this thesis makes the following contributions:

- We provide a systematic comparison of six representative malware-to-image encoding strategies that preserve different characteristics of the underlying executable, including direct byte-based, statistical, entropy-based, and opcode-based representations. The encodings are evaluated under identical experimental conditions on the BIG2015 and BODMAS-11 datasets using a Vision Transformer (ViT-B/16), enabling comparison of their influence on classification performance and computational efficiency.
- We evaluate the robustness of the investigated encoding strategies across multiple random seeds and dataset splits. The evaluation further includes a CNN baseline

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

(ResNet-18) to contextualize the performance of the encodings and assess whether their relative behavior is architecture-dependent.

- We perform a comprehensive evaluation of explainability methods for Vision Transformers by comparing Attention Rollout, Grad-CAM, and Grad-SAM using complementary quantitative metrics, including faithfulness, consistency, stability, sparsity, and computational efficiency, together with qualitative visualizations, malware-family analyses, and feature-space visualizations.
- We propose a reverse-traceability framework that combines executable metadata extraction, structured metadata storage, and patch-to-source mappings to relate attributed image patches to executable artifacts, including PE sections, recovered functions, and assembly instructions. This enables Vision Transformer explanations to be interpreted directly within the analyzed executable.
- We validate the proposed reverse-traceability framework through quantitative traceability analyses, perturbation experiments, and manual reverse-engineering case studies, demonstrating how recovered executable artifacts correspond to prediction-relevant malware functionality.

6.2 Research Questions

The findings presented in Chapter 5 are synthesized below to answer the research questions introduced in Chapter 1.2.

1. **To what extent do different malware-to-image encoding strategies affect the performance of a Vision Transformer model in malware family classification?** Overall, our results suggest that the investigated malware-to-image encoding strategies achieve highly comparable classification performance when evaluated under identical experimental conditions. Although only small differences in classification effectiveness were observed between most investigated encodings, more pronounced differences emerged in computational efficiency. Nevertheless, no single encoding consistently outperformed all others across both datasets and evaluation criteria.

   a) **To what extent does the choice of encoding influence effectiveness measured by metrics such as accuracy, F1-score, and false positive rate?**

   The experimental results show that the investigated malware-to-image encoding strategies achieved broadly comparable classification performance under identical experimental conditions, in contrast to the substantially larger performance differences reported by many individual encoding works [YWL+20, XLCL20, DGS+23]. On BIG2015, five of the six investigated encodings achieved competitive classification performance, although Entropy

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

MALFCS exhibited a noticeably lower macro-F1 score despite maintaining high overall accuracy. On BODMAS-11, four encodings (Markov Bytes, Gray, RGB, and Entropy MALFCS) remained competitive, whereas MCTVD showed a more pronounced performance decrease. Markov Bytes achieved the highest accuracy and the lowest false positive rate in the primary evaluation on both datasets (99.21% on BIG2015 and 97.24% on BODMAS-11). However, its advantage over the strongest competing encodings remained below 0.5 percentage points on BIG2015 and below 0.7 percentage points on BODMAS-11, and the robustness analyses further confirmed that the strongest-performing encodings remained closely matched across different dataset partitions and random seeds. SimHash Opcode was the only encoding that consistently achieved substantially lower classification performance than the other approaches, in contrast to the higher performance reported in the original study by Ni et al. [NQZ18]. When the competitive encoding strategies were evaluated under identical preprocessing procedures, image resolution, model architecture, training protocol, and evaluation methodology, the observed performance differences were considerably smaller than those reported by many works on individual encodings. The similar ranking of the investigated encodings for both the Vision Transformer and the CNN baseline further supports this finding, highlighting the importance of standardized evaluation when comparing malware-to-image representations.

The results further indicate that preserving higher-level semantic information alone does not guarantee superior classification performance. While the opcode-based MCTVD representation performed competitively on BIG2015, its performance decreased substantially on BODMAS-11, whereas byte-based encodings remained consistently strong across both datasets. This suggests that opcode-based representations are more sensitive to dataset characteristics and the availability of recoverable assembly information.

b) **To what extent does the choice of encoding impact efficiency?**

The choice of encoding had a greater impact on computational efficiency than on classification effectiveness. While the strongest encodings achieved highly comparable predictive performance, they exhibited different optimization dynamics during training. Across both datasets, RGB showed the most stable validation accuracy curves, whereas the remaining competitive encodings also reached stable validation performance within approximately the first 10–20 training epochs. Such stable validation trajectories could enable early stopping of training, thereby reducing training time. The largest efficiency differences, however, were observed during preprocessing. In particular, preprocessing costs depended strongly on the type of representation. Byte-based encodings exhibited consistently moderate preprocessing costs, whereas the computational cost of opcode-based representations depended heavily on the amount of recoverable assembly code, with less recoverable code requiring less information to process and encode. Consequently, MCTVD incurred the highest

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

preprocessing cost on BIG2015 but one of the lowest on BODMAS-11.

2. **To what extent can explainability methods for Vision Transformers (ViT) provide faithful, sparse, consistent, stable, and computationally efficient explanations of malware family classification decisions?**

   Overall, the results demonstrate that transformer-specific explainability methods provide more faithful, sparse, consistent, and stable explanations than Grad-CAM for Vision Transformer-based malware family classification. Across all investigated malware-to-image encodings, Attention Rollout and Grad-SAM consistently outperformed Grad-CAM across these evaluation dimensions. Among the evaluated methods, Attention Rollout provided the best overall trade-off by combining high faithfulness and sparsity with high stability and substantially lower computational cost.

   a) **To what extent do different explainability methods for Vision Transformers provide faithful and sparse explanations?**

      The results demonstrate that transformer-specific explainability methods provide substantially more faithful and sparse explanations than Grad-CAM. Attention Rollout and Grad-SAM consistently achieved higher Insertion AUC, lower Deletion AUC, and lower Effective Complexity across the malware-to-image encodings investigated. For example, on the Gray encoding, the Insertion AUC increased from 0.747 for Grad-CAM to 0.902 for Attention Rollout, while the Effective Complexity decreased from 72.6 to 25.6, indicating more focused explanations. Although neither Rollout nor Grad-SAM consistently outperformed the other, both methods clearly surpassed Grad-CAM regardless of the encoding strategy. This finding contrasts with existing Vision Transformer malware-classification studies, which primarily rely on Grad-CAM or qualitative attention visualizations.

   b) **To what extent do different explainability methods for Vision Transformers provide consistent, stable, and computationally efficient explanations?**

      Attention Rollout and Grad-SAM achieved similarly high consistency across all investigated encodings. Attention Rollout achieved the highest Stability@20 scores for four of the five investigated encodings, with MCTVD representing the only exception. Furthermore, Rollout generated explanations approximately two to three times faster (11.8–16.0 s compared with 28.6–37.5 s). Combined with its competitive faithfulness and sparsity, Attention Rollout provided the strongest overall trade-off across all evaluated explainability dimensions.

   c) **To what extent can visualization techniques provide additional insight into the model's learned feature space?**

      The qualitative explanation maps and feature-space visualizations provided complementary insights beyond the quantitative explainability metrics. While the explanation maps illustrated how different encodings and explainability

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

methods highlighted different regions of the executable, the feature-space visualizations revealed structural differences not reflected in classification performance alone. In particular, Gray achieved the highest silhouette score despite not obtaining the highest classification performance, demonstrating that feature-space organization provides complementary insight into the learned representations beyond predictive performance alone.

3. **To what extent can highly influential regions identified by explainability methods for Vision Transformers be mapped back to underlying program structures and meaningfully interpreted through reverse engineering?**

   The proposed reverse-traceability framework demonstrates that highly influential image regions can be systematically mapped back to executable artifacts and interpreted within the original malware binary. Despite many samples having only a limited proportion of image patches mappable, the highest-ranked explanation patches could be successfully linked to PE sections in most cases. Rollout achieved a substantially higher Top-1 mapping success than Grad-SAM (86.9% vs. 77.1%), demonstrating that explainability methods with similar quantitative explainability performance can differ considerably in their suitability for reverse traceability. Function-level traceability proved more challenging, with only 14.1% of traced regions being resolved to individual functions. Nevertheless, recurring PE-section and function-level attribution patterns emerged across malware families, indicating that the recovered artifacts reflect meaningful executable structures rather than random image regions.

   The reverse-engineering case studies further demonstrated that the recovered artifacts are relevant to the model's decision process. Perturbation experiments showed that ablating highly ranked recovered functions consistently reduced model confidence and, in several cases, changed the predicted malware family. Manual reverse engineering further linked the recovered functions to recognizable malware behavior, including propagation, unpacking, API resolution, and network communication. Although function-level interpretation depended on the recoverability of executable artifacts produced by the reverse-engineering pipeline and was therefore more limited for samples with little recoverable program structure, the framework consistently enabled interpretation at the PE-section level and provided function-level explanations whenever sufficient program structure was recoverable. In addition, by automatically ranking executable artifacts by estimated relevance, the framework can support more efficient malware-analysis workflows by narrowing the search space and guiding analysts toward the most relevant program structures for further investigation. Together, these findings demonstrate that Vision Transformer explanations can be systematically translated into executable artifacts that support practical malware analysis and reverse engineering.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.3 Future Work

The findings of this thesis suggest several directions for future research. First, this thesis compared three representative explainability methods for Vision Transformers, representing gradient-based, attention-based, and hybrid approaches. Evaluating additional explainability methods for Vision Transformers, such as Beyond Attention, Transition Attention Maps, and PartialLRP, would help determine whether alternative methods achieve improved explanation quality, reverse traceability, or both.

Second, the evaluation of the proposed framework showed that function-level traceability depends strongly on the recoverability of executable artifacts. Investigating the influence of packing and obfuscation by comparing reverse-traceability results for packed and unpacked malware samples would therefore be a valuable next step. Such an analysis could provide insight into how packing and obfuscation affect explainability, reverse traceability, and malware family classification. Furthermore, integrating automated unpacking techniques into the reverse-traceability pipeline could improve function recovery and extend the framework's applicability to a broader range of malware samples.

Third, function-level traceability remained challenging because each Vision Transformer patch represents a comparatively large byte region, which frequently spans multiple functions and executable artifacts. Investigating Vision Transformer architectures with smaller patch sizes could produce more fine-grained explanation maps and potentially improve function-level traceability. However, smaller patch sizes also increase the number of input tokens, which may affect classification performance and computational efficiency. Evaluating these trade-offs would provide a better understanding of how architectural design choices influence both classification and reverse traceability. Additionally, extending the reverse-traceability evaluation to RGB-based representations could investigate whether grouping multiple bytes into individual pixels affects function-level traceability, and whether the competitive classification performance of RGB encodings comes at the cost of reduced mapping granularity.

Fourth, this thesis evaluated the proposed framework in the context of malware family classification. Extending the evaluation to malware detection tasks that include both malicious and benign executables would assess whether reverse traceability provides meaningful evidence for malware detection in addition to malware family classification.

Finally, a controlled user study with malware analysts could evaluate the practical value of the proposed reverse-traceability framework. The study could compare malware-analysis workflows in which analysts receive only the model prediction and explanation map, with workflows that additionally support reverse traceability. The study could quantitatively assess whether reverse traceability reduces analysis time and search effort, improves the identification of relevant malware functionality, facilitates malware family attribution and report generation, and increases analyst confidence. Such a study would provide a quantitative evaluation of the efficiency requirement, which was assessed qualitatively in this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Overview of Generative AI Tools Used

OpenAI's ChatGPT was used throughout the thesis to improve language and phrasing. OpenAI's Codex was used during the implementation to support code development and debugging.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

2.1 Portable executable file format [GMP20]. 10
2.2 Relationship between high-level source code, machine code, and disassembly used in static malware analysis [SH12]. 11
2.3 Illustration of the visualization-based feature generation process, transforming a malware binary into an image by mapping byte values to pixel intensities [KRM+18]. 12
2.4 Example of a Vision Transformer architecture [Lon24]. 19
2.5 Example of a Vision Transformer architecture for malware classification [KY24]. 22
2.6 Workflow of an explainable malware analysis pipeline, integrating model training, performance evaluation, and explainability generation and assessment [SMFC24]. 25
2.7 Example of Grad-CAM relevance maps for image-based malware classification [SMFC24]. 25

3.1 Transformation of malware binaries into grayscale images [NKJM11] . . . 28

4.1 Example of the data preparation process for a Windows PE file. 56
4.2 Overview of the proposed reverse-traceability framework, including the hybrid metadata storage architecture and the artifact-resolution pipeline. 70

5.1 Row-normalized confusion matrices of the three best-performing encoding strategies on the BIG2015 dataset, where values indicate the fraction of samples within each true malware family assigned to each predicted class. Gray achieves 98.89% accuracy and 97.94% macro-F1, MCTVD 99.07% accuracy and 97.73% macro-F1, and Markov Bytes 99.21% accuracy and 96.86% macro-F1. . . 83
5.2 Validation accuracy across training epochs for the investigated encodings on BIG2015. The dotted line indicates the epoch at which the model with the highest validation accuracy was selected. 83
5.3 Comparison of CNN and Vision Transformer performance across malware image encodings on BIG2015. 84
5.4 Validation accuracy trajectories for the investigated encodings on BODMAS-11. Dotted lines indicate the epoch at which the model with the highest validation accuracy was selected. 85

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.5 Density distributions of byte-file sizes for BIG2015 and BODMAS-11. The file-size axis is shown on a logarithmic scale. 86
5.6 Mean test accuracy and macro-F1 across split seeds for ViT and CNN on BODMAS-11, with error bars showing the standard deviation across seeds. 88
5.7 Comparison of explanation maps for the same correctly classified *Wabot* sample across malware-to-image encodings and explainability methods. 91
5.8 Stability across different values of k, averaged over malware families and three independently trained model instances. 94
5.9 PCA+t-SNE projections of the learned feature representations for the five malware-to-image encodings included in the explainability evaluation. Each point corresponds to a test sample and is colored according to its malware family. 96
5.10 Cumulative distribution of mappable image coverage across the evaluation dataset. 98
5.11 Top-k PE-level mapping success rates for Rollout and GradSAM. 99
5.12 Top three PE sections receiving the largest share of traced bytes for each malware family based on Top-1 Rollout explanations. 103
5.13 Function-level perturbation results. The left column compares Top-1 and Top-3 rankings against random-function baselines, while the right column restricts the analysis to instruction-confirmed functions. Rows correspond to zero-byte and random-byte perturbations, where the bytes associated with the selected functions are replaced with zeros or random byte values, respectively. 107
5.14 Family-level perturbation results for five additional *Small* samples. Curves show cumulative ablation of the instruction-confirmed Top-1 function ranking. The dashed black line denotes the mean confidence trajectory across samples, while the dashed orange line denotes the corresponding random-function baseline mean. 109
5.15 Function-level perturbation results for *Ganelp* Sample A under zero-byte perturbation. The plots show the change in true-class confidence as progressively more ranked functions are ablated, compared with the random-function baseline. 112

1 Relational schema of the SQLite metadata used by the reverse-traceability framework. 144

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

3.1 Comparison of malware-to-image encoding strategies across key dimensions 35
3.2 Overview of transformer-based malware classification approaches using image representations 40
3.3 Comparison of evaluation practices, dataset splits, and efficiency considerations in transformer-based malware classification approaches 41
3.4 Comparison of evaluation criteria discussed in recent malware explainability surveys. 47

4.1 BODMAS family sample distribution. 55
4.2 BODMAS-11 class counts before and after filtering samples without a valid ASM representation for both disassembly backends. All samples were successfully converted to byte-level (.bytes) format. 57
4.3 Class-wise distributions of the reference dataset splits. 60
4.4 Summary of BODMAS-11 dataset variants used in the experiments. . . . 60
4.5 Vision Transformer (ViT-B/16) configuration used in this work. 66
4.6 CNN baseline configuration (ResNet-18) used in this work. 66
4.7 Training configuration used for ViT and ResNet experiments, with selected hyperparameters determined through the searches described below. 67

5.1 Classification effectiveness and efficiency of the investigated encoding strategies on the BIG2015 dataset using the Vision Transformer classifier. Preprocessing times are reported per sample. The best result for each metric is highlighted in dark green. 80
5.2 Comparison of classification accuracy reported in the original studies and obtained by the proposed Vision Transformer (ViT) framework on BIG2015 80
5.3 Comparison of classification performance between the Vision Transformer (ViT) and ResNet-18-based CNN across the investigated encoding strategies on BIG2015. 84
5.4 Classification effectiveness and efficiency of the investigated encodings on BODMAS-11 (ViT) using the Vision Transformer classifier. Preprocessing times are reported per sample. The best result for each metric is highlighted in dark green. 85

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.5 Byte-file size distribution statistics for BIG2015 and BODMAS-11. Q1 and Q3 denote the first (25th percentile) and third (75th percentile) quartiles, respectively. All values are reported in megabytes (MB). 87

5.6 Summary of explainability evaluation results across malware-to-image encodings and explainability methods. Higher values indicate better performance for Insertion AUC (Ins.), Consistency (Cons.), and Stability@20 (Stab.@20). Lower values indicate better performance for Deletion AUC (Del.), Effective Complexity (EC), and mean explanation generation time per sample in milliseconds (Time). Stability@20 denotes the average top-20 patch overlap across independently trained model instances, macro-averaged over malware families. The best result for each metric and encoding is highlighted in dark green. 92

5.7 Interquartile ranges (IQR) of Insertion and Deletion AUC across evaluated samples for each malware-to-image encoding and explainability method. Lower values indicate lower variability across samples. The best result for each metric and encoding is highlighted in dark green. 93

5.8 Per-family explainability results for the Gray encoding using Attention Rollout and Grad-SAM. Best and worst values for each metric are highlighted in bold and underlined, respectively. Cell shading indicates relative performance across malware families within each metric and explainability method using min–max normalization, with green indicating better and red indicating poorer relative performance. 95

5.9 Silhouette scores computed on the learned feature representations. Higher values indicate more compact and better separated malware-family clusters. The best result is highlighted in dark green. 96

5.10 Top-1 mapping success across all 2,767 test samples. 99

5.11 PE-level mapping success rates (%) by malware family for Rollout and Grad-SAM at Top-1 and Top-20. 100

5.12 Patch-weighted function-level traceability for Rollout explanations. 101

5.13 Function-level traceability of Top-1 Rollout explanations across malware families (%). 101

5.14 Three most frequently occurring top-ranked functions for each malware family based on Top-1 Rollout explanations. Percentages indicate the proportion of samples in which the respective function appeared at the given rank. . . . 104

5.15 Sample-level summary. 106

5.16 Top-1 patch traceability summary. 106

5.17 Highest-ranked functions recovered from the Top-1 explanation patch. Functions are ordered by overlap-weighted contribution score. 106

5.18 Window-ablation experiments for the instruction-confirmed Top-1 ranking. Δ confidence denotes the change in true-class confidence relative to the original sample. Prediction flips indicate reassignment from *Small* to *Sillyp2p*. . . 108

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.19 Top-1 patch traceability composition across five additional *Small* samples. Values denote the percentage of Top-1 patch bytes assigned to each resolution category. 109
5.20 Top-15 instruction-confirmed functions recovered from the highest-attribution patch. 110
5.21 Top-1 traceability comparison for two representative *Ganelp* samples. . . . 112

1 Test accuracy across the three BODMAS-11 dataset partitions. Results are reported as percentages. Mean and standard deviation are calculated across the three dataset partitions. 145
2 Macro-F1 scores across the three BODMAS-11 dataset partitions. Results are reported as percentages. Mean and standard deviation are calculated across the three dataset partitions. 145
3 Additional robustness metrics across the three BODMAS-11 dataset partitions. Macro-FPR is reported as mean ± standard deviation. The selected epochs correspond to the checkpoints achieving the highest validation accuracy for each dataset partition. 146
4 Vision Transformer classification performance across three training seeds on a fixed BODMAS-11 dataset partition. The dataset split is held constant while only the training seed is varied. Test accuracy, macro-F1, and macro-FPR are reported for seeds 42, 7, and 123. 146

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[AB18] Amina Adadi and Mohammed Berrada. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.

[AM25] Mohammad Alshoulie and Abid Mehmood. Deep Learning Approaches for Malware Detection: A Comprehensive Review of Techniques, Challenges, and Future Directions. *IEEE Access*, 13:118652–118677, 2025.

[AZ20] Samira Abnar and Willem Zuidema. Quantifying Attention Flow in Transformers. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4190–4197, Online, July 2020. Association for Computational Linguistics.

[BAC+26] Matteo Brosolo, Asmitha K. A., Mauro Conti, Rafidha Rehiman K. A., Muhammed Shafi K. P., Serena Nicolazzo, Antonino Nocera, and Vinod P. Security through the eyes of AI: How visualization is shaping malware detection. *Computer Science Review*, 61:100914, August 2026.

[BaoEB23] Ikram Ben abdel ouahab, Lotfi Elaachak, and Mohammed Bouhorma. Enhancing Malware Classification with Vision Transformers: A Comparative Study with Traditional CNN Models. In *Proceedings of the 6th International Conference on Networking, Intelligent Systems & Security*, NISS '23, pages 1–5, Larache, Morocco, November 2023. Association for Computing Machinery.

[BBM+15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLoS ONE*, 10(7):e0130140, July 2015.

[BC17] Or Biran and Courtenay V. Cotton. Explanation and Justification in Machine Learning: A Survey. In *Proceedings of the IJCAI-17 Workshop on Explainable Artificial Intelligence (XAI)*, pages 8–13, Melbourne, Australia, August 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[BD18] Sergii Banin and Geir Olav Dyrkolbotn. Multinomial malware classification via low-level features. *Digit. Investig.*, 26(S):S107–S117, July 2018.

[BHC+21] Oren Barkan, Edan Hauon, Avi Caciularu, Ori Katz, Itzik Malkiel, Omri Armstrong, and Noam Koenigstein. Grad-SAM: Explaining Transformers via Gradient Self-Attention Maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, pages 2882–2887, New York, NY, USA, October 2021. Association for Computing Machinery.

[BJS25] David Bálik, Martin Jureček, and Mark Stamp. RawMal-TF: Raw Malware Dataset Labeled by Type and Family, June 2025. arXiv:2506.23909 [cs].

[BM24] Shrey Bavishi and Shrey Modi. Accelerating Malware Classification: A Vision Transformer Solution, September 2024. arXiv:2409.19461 [cs].

[BPA+24] Matteo Brosolo, Vinod Puthuvath, Asmitha K. A., Rafidha Rehiman K. A., and Mauro Conti. SoK: Visualization-based Malware Detection Techniques. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ARES '24, pages 1–13, New York, NY, USA, July 2024. Association for Computing Machinery.

[Bro97] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, Salerno, Italy, June 1997. IEEE.

[BS23] Mohamad Mulham Belal and Divya Meena Sundaram. Global-Local Attention-Based Butterfly Vision Transformer for Visualization-Based Malware Classification. *IEEE Access*, 11:69337–69355, 2023.

[Bun24] Bundesamt für Sicherheit in der Informationstechnik (BSI). The State of IT Security in Germany 2024. Technical report, Federal Office for Information Security (BSI), November 2024. Available at: `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Securitysituation/IT-Security-Situation-in-Germany-2024.html?nn=1021082`.

[CGW21] Hila Chefer, Shir Gur, and Lior Wolf. Transformer Interpretability Beyond Attention Visualization. In *Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 782–791. Institute of Electrical and Electronics Engineers, June 2021.

[Cha02] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 380–388, New York, NY, USA, May 2002. Association for Computing Machinery.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[CJKK17] Sunoh Choi, Sungwook Jang, Youngsoo Kim, and Jonghyun Kim. Malware detection using malware image and deep learning. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1193–1195, Jeju Island, Republic of Korea, October 2017. Institute of Electrical and Electronics Engineers Inc.

[CLH+22] Shi Chen, Ying Liu, Wei Hu, Jianyi Liu, Yating Gao, and Bingjie Lin. Malicious Code Family Classification Method Based on Vision Transformer. In *2022 IEEE 10th International Conference on Information, Communication and Networks (ICICN)*, pages 704–709, Zhangye, China, August 2022. IEEE.

[CLY+23] Jiamin Chen, Xuhong Li, Lei Yu, Dejing Dou, and Haoyi Xiong. Beyond Intuition: Rethinking Token Attributions inside Transformers. *Transactions on Machine Learning Research*, 2023.

[CWQ+25] Boyan Cui, Huijuan Wang, Yongjun Qi, Hongce Chen, Quanbo Yuan, Dongran Liu, and Xuehua Zhou. HERL-ViT: A Hybrid Enhanced Vision Transformer Based on Regional-Local Attention for Malware Detection. *Computers, Materials & Continua*, 85(3):5531–5553, 2025.

[DA23] Fatemeh Deldar and Mahdi Abadi. Deep Learning for Zero-day Malware Detection and Classification: A Survey. *ACM Comput. Surv.*, 56(2):36:1–36:37, September 2023.

[DAM+21] Abdulbasit Darem, Jemal Abawajy, Aaisha Makkar, Asma Alhashmi, and Sultan Alanazi. Visualization and deep-learning-based malware variant detection using OpCode-level features. *Future Generation Computer Systems*, 125:314–323, December 2021.

[DBH18] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0210–0215, Opatija, Croatia, May 2018. IEEE.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. arXiv:2010.11929 [cs].

[DGS+23] Huaxin Deng, Chun Guo, Guowei Shen, Yunhe Cui, and Yuan Ping. MCTVD: A malware classification method based on three-channel visualization and deep learning. *Computers & Security*, 126:103084, March 2023.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[DJT+22] Nitul Dutta, Nilesh Jadav, Sudeep Tanwar, Hiren Kumar Deva Sarma, and Emil Pricop. Introduction to Malware Analysis. In Nitul Dutta, Nilesh Jadav, Sudeep Tanwar, Hiren Kumar Deva Sarma, and Emil Pricop, editors, *Cyber Security: Issues and Current Trends*, pages 129–141. Springer, Singapore, 2022.

[EH20] Ahmet Efe and Saleh Hussin S. Hussin. Malware Visualization Techniques. *International Journal of Applied Mathematics Electronics and Computers*, 8(1):7–20, March 2020.

[FWX+20] Ming Fan, Wenying Wei, Xiaofei Xie, Yang Liu, Xiaohong Guan, and Ting Liu. Can We Trust Your Explanations? Sanity Checks for Interpreters in Android Malware Analysis, August 2020. arXiv:2008.05895 [cs.CR].

[FXW+18] Jianwen Fu, Jingfeng Xue, Yong Wang, Zhenyan Liu, and Chun Shan. Malware Visualization for Fine-Grained Classification. *IEEE Access*, 6:14510–14523, 2018.

[GBS14] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. Malware Analysis and Classification: A Survey. *Journal of Information Security*, 5(2):56–64, April 2014.

[GMP20] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, March 2020.

[GMPV19] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15(1):15–28, March 2019.

[GOMdR25] Numan Halit Guldemir, Oluwafemi Olukoya, and Jesús Martínez-del Rincón. Navigating Concept Drift and Packing Complexity in Malware Family Classification. In *Conference on Applied Machine Learning for Information Security (CAMLIS 2024): Proceedings*, volume 3920 of *CEUR Workshop Proceedings*, pages 129–144, Arlington, United States, February 2025. CEUR-WS.

[HR20] Richard Harang and Ethan M. Rudd. SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection, December 2020. arXiv:2012.07634 [cs].

[HWB+23] Anna Hedström, Leander Weber, Dilyara Bareeva, Daniel Krakowczyk, Franz Motzkus, Wojciech Samek, Sebastian Lapuschkin, and Marina M.-C. Höhne. Quantus: An explainable ai toolkit for responsible evaluation of neural network explanations and beyond. *J. Mach. Learn. Res.*, 24(1):34:1339–34:1349, January 2023.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE.

[ICM+21] Giacomo Iadarola, Rosangela Casolare, Fabio Martinelli, Francesco Mercaldo, Christian Peluso, and Antonella Santone. A Semi-Automated Explainability-Driven Approach for Malware Analysis through Deep Learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Shenzhen, China, July 2021. IEEE.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA, May 1998. Association for Computing Machinery.

[JAV+24] Jahez Abraham Johny, K. A. Asmitha, P. Vinod, G. Radhamani, K. A. Rafidha Rehiman, and Mauro Conti. Deep learning fusion for effective malware detection: leveraging visual features. *Cluster Computing*, 28(2):135, November 2024.

[JG20] Alon Jacovi and Yoav Goldberg. Towards Faithfully Interpretable NLP Systems: How Should We Define and Evaluate Faithfulness? In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4198–4205, Online, July 2020. Association for Computational Linguistics.

[JKR+21] Yifei Jian, Hongbo Kuang, Chenglong Ren, Zicheng Ma, and Haizhou Wang. A novel framework for image-based malware detection with a deep neural network. *Computers & Security*, 109:102400, October 2021.

[JLKC25] In-Woong Jeong, Han-Ju Lee, Gwang-Nam Kim, and Seok-Hwan Choi. MalFormer: A Novel Vision Transformer Model for Robust Malware Analysis. *IEEE Access*, 13:122671–122683, 2025.

[KA18] Monnappa K. A. *Learning malware analysis: explore the concepts, tools, and techniques to analyze and investigate Windows malware.* Packt Publishing, Birmingham, UK, 2018.

[Kas24] Kaspersky Lab. The cyber surge: Kaspersky detected 467,000 malicious files daily in 2024, December 2024. Available at: `https://www.kaspersky.com/about/press-releases/the-cyber-surge-kaspersky-detected-467000-malicious-files-daily-in-2024`.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[KBSA26] Rojina Kashefi, Leili Barekatain, Mohammad Sabokrou, and Fatemeh Aghaeipoor. Explainability of vision transformers: a comprehensive review and new perspectives. *Multimedia Tools and Applications*, 85(2):115, February 2026.

[Khu20] Vala Khushali. A Review on Fileless Malware Analysis Techniques. *International Journal of Engineering Research & Technology*, 9(5), May 2020.

[KNH+22] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in Vision: A Survey. *ACM Computing Surveys*, 54(10s):1–41, January 2022.

[KRM+18] Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil D. B. Bruce, Yang Wang, and Farkhund Iqbal. Malware Classification with Deep Convolutional Neural Networks. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, Paris, France, February 2018. IEEE.

[KY24] Oğuzhan Katar and Özal Yıldırım. Classification of Malware Images Using Fine-Tunned ViT. *Sakarya University Journal of Computer and Information Sciences*, 7(1):22–35, April 2024.

[KZK19] Riaz Ullah Khan, Xiaosong Zhang, and Rajesh Kumar. Analysis of ResNet and GoogleNet models for malware detection. *Journal of Computer Virology and Hacking Techniques*, 15(1):29–37, March 2019.

[LACB24] Qing Lyu, Marianna Apidianaki, and Chris Callison-Burch. Towards Faithful Model Explanation in NLP: A Survey. *Computational Linguistics*, 50(2):657–723, June 2024.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[Lip18] Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, September 2018.

[LLLZ20] Xinbo Liu, Yaping Lin, He Li, and Jiliang Zhang. A novel method for malware detection on ML-based visualization technique. *Computers & Security*, 89:101682, February 2020.

[Lon24] Hanhua Long. Hybrid Design of CNN and Vision Transformer: A Review. In *Proceedings of the 2024 7th International Conference on Computer Information Science and Artificial Intelligence*, CISAI '24, pages 121–127, New York, NY, USA, December 2024. Association for Computing Machinery.

[LWSW24] Sicong Li, Jian Wang, Yafei Song, and Shuo Wang. Tri-channel visualised malicious code classification based on improved ResNet. *Applied Intelligence*, 54(23):12453–12475, December 2024.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[LWZ+23] Xiang Ling, Lingfei Wu, Jiangyu Zhang, Zhenqing Qu, Wei Deng, Xiang Chen, Yaguan Qian, Chunming Wu, Shouling Ji, Tianyue Luo, Jingzheng Wu, and Yanjun Wu. Adversarial Attacks against Windows PE Malware Detection: A Survey of the State-of-the-Art. *Computers & Security*, 128:103134, May 2023.

[LZKN22] Qikai Lu, Hongwen Zhang, Husam Kinawi, and Di Niu. Self-Attentive Models for Real-Time Malware Classification. *IEEE Access*, 10:95970–95985, 2022.

[MCS+26] Fadi Makarem, Rida El Chall, Abed Ellatif Samhat, Khouloud Samrouth, and Nader Bakir. Exploiting Vision Transformer and Ensemble Learning for Advanced Malware Classification. *Engineering Reports*, 8(1):e70558, 2026.

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[Mil19] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, February 2019.

[MLW+24] Melkamu Mersha, Khang Lam, Joseph Wood, Ali AlShami, and Jugal Kalita. Explainable Artificial Intelligence: A Survey of Needs, Techniques, Applications, and Future Direction. *Neurocomputing*, 599:128111, September 2024.

[MMC24] Pascal Maniriho, Abdun Naser Mahmood, and Mohammad Jabed Morshed Chowdhury. A systematic literature review on Windows malware detection: Techniques, research issues, and future directions. *Journal of Systems and Software*, 209:111921, March 2024.

[NKJM11] 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*, VizSec '11, pages 1–7, New York, NY, USA, July 2011. Association for Computing Machinery.

[NKM15] Lakshmanan Nataraj, S. Karthikeyan, and B.S. Manjunath. SATTVA: SpArsiTy inspired classificaTion of malware VAriants. In *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '15, pages 135–140, New York, NY, USA, June 2015. Association for Computing Machinery.

[NM20] An-phi Nguyen and María Rodríguez Martínez. On quantitative aspects of model interpretability, July 2020. arXiv:2007.07584 [cs.LG].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[NQZ18] Sang Ni, Quan Qian, and Rui Zhang. Malware identification using visualization images and deep learning. *Computers & Security*, 77:871–885, August 2018.

[NYPZ11] Lakshmanan Nataraj, Vinod Yegneswaran, Phillip Porras, and Jian Zhang. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, pages 21–30, New York, NY, USA, October 2011. Association for Computing Machinery.

[OS22] Stephen O'Shaughnessy and Stephen Sheridan. Image-based malware classification hybrid framework based on space-filling curves. *Computers & Security*, 116:102660, May 2022.

[PC22] Kyoung-Won Park and Sung-Bae Cho. A Vision Transformer Enhanced with Patch Encoding for Malware Classification. In Hujun Yin, David Camacho, and Peter Tino, editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2022*, pages 289–299, Cham, 2022. Springer International Publishing.

[PDS18] Vitali Petsiuk, Abir Das, and Kate Saenko. RISE: Randomized Input Sampling for Explanation of Black-box Models. In *British Machine Vision Conference 2018 (BMVC 2018)*, page 151, Newcastle, UK, 2018. BMVA Press.

[PLP+21] Anson Pinhero, Anupama M. L., Vinod P., C. A. Visaggio, Aneesh N., Abhijith S., and AnanthaKrishnan S. Malware detection employed by visualization and deep neural network. *Computers & Security*, 105:102247, June 2021.

[RCS23] Akshara Ravi, Vivek Chaturvedi, and Muhammad Shafique. ViT4Mal: Lightweight Vision Transformer for Malware Detection on Edge Devices. *ACM Trans. Embed. Comput. Syst.*, 22(5s):117:1–117:26, September 2023.

[Rou87] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, November 1987.

[RRF+18] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft Malware Classification Challenge, February 2018. arXiv:1802.10135 [cs].

[RZ24] Tianyu Ruan and Shihua Zhang. Towards understanding how attention mechanism works in deep learning, December 2024. arXiv:2412.18288 [cs].

[SCD+17] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

from Deep Networks via Gradient-Based Localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, Venice, Italy, October 2017. IEEE.

[SCMS23] Sédrick Stassin, Valentin Corduant, Sidi Ahmed Mahmoudi, and Xavier Siebert. Explainability and Evaluation of Vision Transformers: An In-Depth Experimental Study. *Electronics*, 13(1), December 2023.

[SDG23] Marcelo Invert Palma Salas and Paulo De Geus. Static Analysis for Malware Classification Using Machine and Deep Learning. In *2023 XLIX Latin American Computer Conference (CLEI)*, pages 1–10, La Paz, Bolivia, October 2023. IEEE.

[SH12] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, February 2012.

[SHKS19] Ajay Singh, Anand Handa, Nitesh Kumar, and Sandeep Kumar Shukla. Malware Classification Using Image Representation. In Shlomi Dolev, Danny Hendler, Sachin Lodha, and Moti Yung, editors, *Cyber Security Cryptography and Machine Learning*, pages 75–92, Cham, 2019. Springer International Publishing.

[SMFC24] Mohd Saqib, Samaneh Mahdavifar, Benjamin C. M. Fung, and Philippe Charland. A Comprehensive Analysis of Explainable AI for Malware Hunting. *ACM Comput. Surv.*, 56(12):314:1–314:40, October 2024.

[SQ21] Guosong Sun and Quan Qian. Deep Learning and Visualization for Identifying Malware Families. *IEEE Transactions on Dependable and Secure Computing*, 18(1):283–295, January 2021.

[SSRK22] Sachith Seneviratne, Ridwan Shariffdeen, Sanka Rasnayaka, and Nuran Kasthuriarachchi. Self-Supervised Vision Transformers for Malware Detection. *IEEE Access*, 10:103121–103135, 2022.

[TK25] Maryam Tanha and Somayeh Kafaie. A Review of Explainable AI for Android Malware Detection and Analysis. *IEEE Access*, 13:141958–141974, 2025.

[UAB19] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, March 2019.

[vdM14] Laurens van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15:3221–3245, 2014.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Vir20] VirusTotal. VirusTotal analysis report for SHA-256 ea2f4b7253770cb59d41388fad43e02f0f89773f32d7aa7783a85ffb8a225376, 2020. First submitted and analyzed on 25 February 2020. Accessed on 20 August 2025. Available at: `https://www.virustotal.com/`.

[VNN+20] Duc-Ly Vu, Trong-Kha Nguyen, Tam V. Nguyen, Tu N. Nguyen, Fabio Massacci, and Phu H. Phung. HIT4Mal: Hybrid image transformation for malware classification. *Transactions on Emerging Telecommunications Technologies*, 31(11):e3789, 2020.

[VRM+19] Daniel Votipka, Seth Rabin, Kristopher Micinski, Jeffrey S. Foster, and Michelle L. Mazurek. An Observational Investigation of Reverse Engineers' Process and Mental Models. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI EA '19, pages 1–6, New York, NY, USA, May 2019. Association for Computing Machinery.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is all You need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30. Curran Associates, Inc., 2017.

[VTM+19] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics.

[Wat05] M. Wattenberg. A note on space-filling visualizations and space-filling curves. In *IEEE Symposium on Information Visualization (InfoVis)*, pages 181–186, Minneapolis, MN, USA, October 2005. IEEE.

[WAWR20] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. Evaluating Explanation Methods for Deep Learning in Security. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 158–174, Genoa, Italy, September 2020. IEEE.

[WG15] Haibing Wu and Xiaodong Gu. Towards dropout training for convolutional neural networks. *Neural Networks*, 71:1–10, November 2015.

[WZLZ19] Shu-wei Wang, Gang Zhou, Ji-cang Lu, and Feng-juan Zhang. A Novel Malware Detection and Classification Method Based on Capsule Network. In Xingming Sun, Zhaoqing Pan, and Elisa Bertino, editors, *Artificial Intelligence and Security*, pages 573–584, Cham, 2019. Springer International Publishing.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[XLCL20] Guoqing Xiao, Jingning Li, Yuedan Chen, and Kenli Li. MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. *Journal of Parallel and Distributed Computing*, 141:49–58, July 2020.

[YCA+25] Yaoxiang Yu, Bo Cai, Kamran Aziz, Xinyan Wang, Jian Luo, Muhammad Shahid Iqbal, Prasun Chakrabarti, and Tulika Chakrabarti. Semantic lossless encoded image representation for malware classification. *Scientific Reports*, 15(1):7997, March 2025.

[YCL+21] 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, San Francisco, CA, USA, May 2021. IEEE.

[YLAI17] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.*, 50(3):41:1–41:40, June 2017.

[YLX+21] Tingyi Yuan, Xuhong Li, Haoyi Xiong, Hui Cao, and Dejing Dou. Explaining information flow inside vision transformers using markov chain. In *1st Workshop on eXplainable AI Approaches for Debugging and Diagnosis (XAI4Debugging@NeurIPS 2021)*, 2021.

[YSN+19] Hiromu Yakura, Shinnosuke Shinozaki, Reon Nishimura, Yoshihiro Oyama, and Jun Sakuma. Neural malware analysis with attention mechanism. *Computers & Security*, 87:101592, November 2019.

[YT21] Balram Yadav and Sanjiv Tokekar. Deep Learning in Malware Identification and Classification. In Mark Stamp, Mamoun Alazab, and Andrii Shalaginov, editors, *Malware Analysis Using Artificial Intelligence and Deep Learning*, pages 163–205. Springer International Publishing, Cham, 2021.

[YWL+20] Baoguo Yuan, Junfeng Wang, Dong Liu, Wen Guo, Peng Wu, and Xuhua Bao. Byte-level malware classification based on markov images and deep learning. *Computers & Security*, 92:101740, May 2020.

[YWLA+21] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M. Blough, Elissa M. Redmiles, and Mustaque Ahamad. An Inside Look into the Practice of Malware Analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 3053–3069, New York, NY, USA, November 2021. Association for Computing Machinery.

[ZG24] Pin Zhao and Gang Gan. Malware Detector Based On Enhanced Vision Transformer. In *2024 2nd International Conference On Mobile Internet,*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Cloud Computing and Information Security (MICCIS), pages 163–167, Changsha City, China, April 2024. IEEE.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Appendix

Hybrid Metadata Storage

The reverse-traceability framework uses a hybrid storage architecture consisting of SQLite metadata, per-sample Parquet mappings, and local explainability artifacts. Figure 1 illustrates the relational schema of the SQLite metadata used in the evaluation. Large patch-to-source mappings are stored separately as Parquet files, while prediction outputs and relevance maps are maintained in the corresponding experiment directories.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

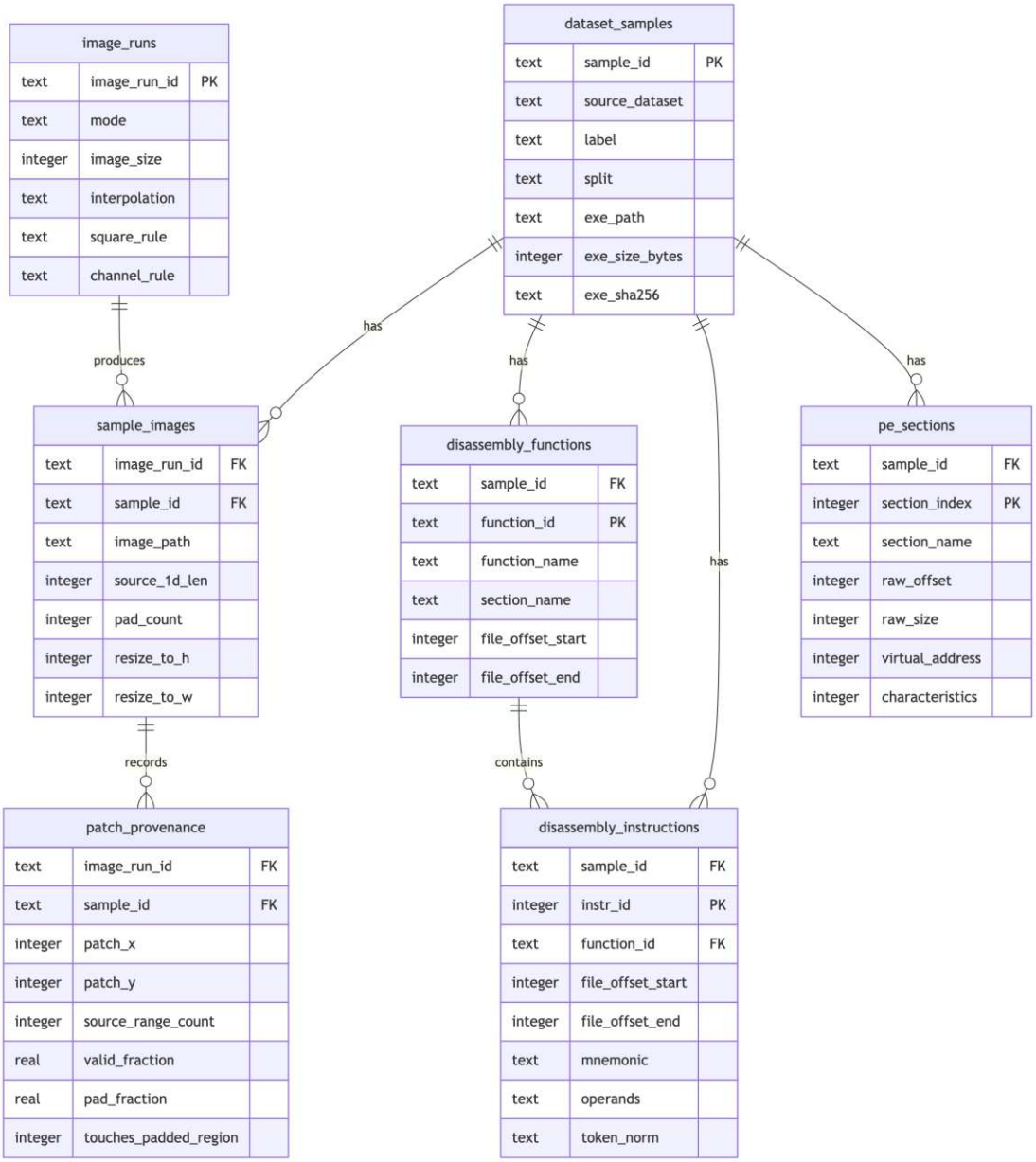


Figure 1: Relational schema of the SQLite metadata used by the reverse-traceability framework.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Classification Robustness

This section reports the detailed results underlying the robustness analysis presented in Section 5.1.2. The first three tables summarize classification performance across the three BODMAS-11 dataset partitions, while the final table reports the results of the training-seed robustness experiment using a fixed dataset partition.

Table 1: Test accuracy across the three BODMAS-11 dataset partitions. Results are reported as percentages. Mean and standard deviation are calculated across the three dataset partitions.

Model	Encoding	Partition 42	Partition 7	Partition 123	Mean ± SD
ViT	Gray	96.94	96.02	97.13	96.70 ± 0.59
ViT	Markov Bytes	97.24	96.39	96.98	96.87 ± 0.43
ViT	RGB	97.13	96.91	96.98	97.00 ± 0.11
ViT	Entropy MALFCS	96.61	95.69	96.35	96.22 ± 0.48
ViT	MCTVD	94.51	95.06	95.06	94.88 ± 0.32
ViT	SimHash Opcode	84.71	85.60	85.30	85.21 ± 0.45
CNN	Gray	96.80	96.13	96.69	96.54 ± 0.34
CNN	Markov Bytes	97.46	96.69	97.35	97.16 ± 0.42
CNN	RGB	96.83	96.46	96.72	96.67 ± 0.19
CNN	Entropy MALFCS	96.72	96.61	96.80	96.71 ± 0.09
CNN	MCTVD	94.36	95.21	95.14	94.90 ± 0.47
CNN	SimHash Opcode	84.94	85.01	85.41	85.12 ± 0.26

Table 2: Macro-F1 scores across the three BODMAS-11 dataset partitions. Results are reported as percentages. Mean and standard deviation are calculated across the three dataset partitions.

Model	Encoding	Partition 42	Partition 7	Partition 123	Mean ± SD
ViT	Gray	96.65	95.74	96.83	96.40 ± 0.58
ViT	Markov Bytes	97.04	96.13	96.72	96.63 ± 0.46
ViT	RGB	96.85	96.73	96.71	96.76 ± 0.08
ViT	Entropy MALFCS	96.32	95.37	95.99	95.90 ± 0.48
ViT	MCTVD	93.60	94.27	94.27	94.05 ± 0.39
ViT	SimHash Opcode	83.35	84.49	84.23	84.02 ± 0.60
CNN	Gray	96.50	95.91	96.24	96.21 ± 0.30
CNN	Markov Bytes	97.29	96.44	97.09	96.94 ± 0.45
CNN	RGB	96.62	96.25	96.34	96.40 ± 0.19
CNN	Entropy MALFCS	96.56	96.36	96.51	96.48 ± 0.11
CNN	MCTVD	93.43	94.36	94.37	94.05 ± 0.54
CNN	SimHash Opcode	83.57	83.62	84.29	83.83 ± 0.40

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 3: Additional robustness metrics across the three BODMAS-11 dataset partitions. Macro-FPR is reported as mean ± standard deviation. The selected epochs correspond to the checkpoints achieving the highest validation accuracy for each dataset partition.

Model	Encoding	Macro-FPR Mean ± SD	Partition 42	Partition 7	Partition 123	Range (epochs)
			Best-validation epoch			
ViT	Gray	0.00333 ± 0.00061	63	18	17	46
ViT	Markov Bytes	0.00317 ± 0.00049	25	42	30	17
ViT	RGB	0.00301 ± 0.00014	12	15	20	8
ViT	Entropy MALFCS	0.00380 ± 0.00053	10	18	11	8
ViT	MCTVD	0.00506 ± 0.00031	25	76	38	51
ViT	SimHash Opcode	0.01588 ± 0.00041	83	40	90	50
CNN	Gray	0.00348 ± 0.00038	12	22	62	50
CNN	Markov Bytes	0.00286 ± 0.00045	29	22	65	43
CNN	RGB	0.00336 ± 0.00023	76	28	69	48
CNN	Entropy MALFCS	0.00332 ± 0.00008	76	13	16	63
CNN	MCTVD	0.00503 ± 0.00046	15	14	8	7
CNN	SimHash Opcode	0.01599 ± 0.00028	10	5	74	69

Table 4: Vision Transformer classification performance across three training seeds on a fixed BODMAS-11 dataset partition. The dataset split is held constant while only the training seed is varied. Test accuracy, macro-F1, and macro-FPR are reported for seeds 42, 7, and 123.

Encoding	Test Accuracy			Macro-F1			Macro-FPR		
	Seed 42	Seed 7	Seed 123	Seed 42	Seed 7	Seed 123	Seed 42	Seed 7	Seed 123
Entropy MALFCS	96.61	96.80	96.32	96.32	96.70	95.99	0.00338	0.00324	0.00366
Gray	96.94	97.05	96.91	96.65	96.82	96.59	0.00308	0.00294	0.00309
Markov Bytes	97.24	97.31	97.35	97.04	97.09	97.19	0.00275	0.00269	0.00265
MCTVD	94.51	94.48	94.55	93.60	93.59	93.67	0.00542	0.00545	0.00538
RGB	97.13	96.98	97.02	96.85	96.71	96.73	0.00286	0.00301	0.00299
SimHash Opcode	84.71	84.68	84.57	83.35	83.33	83.09	0.01632	0.01636	0.01647

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Reverse Engineering Evidence of Malicious Behavior

This appendix provides representative disassembly excerpts from the functions discussed in the reverse-engineering case study. The excerpts illustrate the malware-related functionality identified during the manual analysis and support the interpretation of the recovered executable artifacts identified in the analyzed malware sample [1]. To aid readability, a brief overview of the most common assembly instructions is provided first.

Basic Assembly Instructions

Instruction	Meaning
`push X`	Place a value on the stack (typically for function arguments).
`call Func`	Call a function.
`mov A, B`	Copy the value from `B` to `A`.
`lea reg, [mem]`	Load the address of a memory location.
`cmp A, B`	Compare two values.
`jz, jnz, jg`	Conditional jumps.
`and reg, value`	Bitwise AND (commonly used for flag checks).
`test reg, reg`	Test whether a value is zero.

Function 00401577 – File Propagation

Code Snippet 1

```
push    offset asc_4094EC          ; push "*" wildcard string
lea     edx, [ebp+FileName]        ; load address of search-path buffer
push    edx                        ; push destination buffer
call    _strcat                    ; append wildcard to path
...
mov     eax, [ebp+lpFindFileData] ; load pointer to find-data structure
push    eax                        ; push output buffer for enumeration results
lea     ecx, [ebp+FileName]        ; load completed search pattern
push    ecx                        ; push file search pattern
call    ds:FindFirstFileA          ; begin directory enumeration
```

This snippet indicates directory enumeration behavior. The function appends the wildcard character "*" to a file path and invokes `FindFirstFileA`, which initiates a search for matching files and directories. This pattern is commonly used to iterate over filesystem entries and suggests that the program systematically processes files within a directory.

Code Snippet 2

```
push    offset Str2                ; "." current directory entry
mov     ecx, [ebp+lpFindFileData] ; load file info structure
add     ecx, 2Ch                   ; point to filename field
```

[1] SHA-256: `ea2f4b7253770cb59d41388fad43e02f0f89773f32d7aa7783a85ffb8a225376`.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
push    ecx                         ; push found filename
call    _strcmp                     ; compare with "."
...
push    offset asc_4094F4           ; ".." parent directory entry
mov     edx, [ebp+lpFindFileData]   ; load file info structure
add     edx, 2Ch                    ; point to filename field
push    edx                         ; push found filename
call    _strcmp                     ; compare with ".."
...
mov     eax, [ebp+lpFindFileData]   ; load file info structure
mov     ecx, [eax]                  ; load file attributes
and     ecx, 10h                    ; isolate directory flag
test    ecx, ecx                    ; check if entry is a directory
jz      loc_40178B                  ; jump if not a directory
```

This snippet refines the enumeration logic by filtering out the special directory entries "." and "..", which prevents redundant traversal. Additionally, the bitwise check using `and ecx, 10h` isolates the directory attribute flag, enabling the program to distinguish between files and directories. This pattern is consistent with recursive directory traversal routines.

Code Snippet 3

```
lea     eax, [ebp+FileName]         ; load address of filename buffer
push    eax                         ; push filename string
call    __strupr                    ; convert filename to uppercase
...
push    offset aEmuleExe            ; "EMULE.EXE" target name
lea     ecx, [ebp+FileName]         ; load normalized filename
push    ecx                         ; push filename for comparison
call    _strcmp                     ; compare with "EMULE.EXE"
```

This snippet shows normalization of filenames followed by comparison against a specific target string (`EMULE.EXE`). The explicit matching of a known peer-to-peer application [2] suggests targeted interaction with specific software, which may indicate propagation strategies that leverage shared directories or commonly distributed executables.

Code Snippet 4

```
push    eax                         ; push file info buffer
mov     ecx, [ebp+hFindFile]        ; load search handle
push    ecx                         ; push search handle
call    ds:FindNextFileA            ; continue directory enumeration
```

This instruction continues the enumeration process using `FindNextFileA`, confirming that the function iterates over multiple directory entries. Together with the previous snippets, this supports the interpretation of a loop-based traversal over filesystem contents.

[2] `https://www.emule-project.com/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Code Snippet 5

```
mov     ecx, [ebp+Block]          ; load current linked-list node
mov     edx, [ecx]                ; load stored child-directory path
push    edx                       ; push child path as argument
call    sub_401577                ; recursively process subdirectory
add     esp, 4                    ; clean argument from stack
```

This snippet provides direct evidence of recursion. After collecting subdirectory paths during enumeration, the function invokes itself on each stored child path. This confirms that the routine performs recursive descent through the filesystem rather than merely inspecting a single directory level.

Explanation Function 00401577 provides strong evidence of automated filesystem traversal behavior. The combination of wildcard-based enumeration, filtering of special directory entries, and directory attribute checks is consistent with recursive traversal mechanisms. Furthermore, the comparison against specific filenames such as `EMULE.EXE` suggests targeted interaction with particular applications. Overall, the observed behavior is consistent with patterns commonly associated with worm-like propagation mechanisms.

Function 00401FC2 – Peer-to-Peer Propagation

Code Snippet 1

```
lea     eax, [ebp+phkResult]      ; address for returned registry handle
push    eax                       ; push output handle pointer
push    offset aSoftwareImeshC    ; "SOFTWARE\\IMesh\\Client\\LocalContent"
push    80000001h                 ; HKEY_CURRENT_USER
call    ds:RegOpenKeyA            ; open IMesh registry key
```

This snippet shows access to a registry key associated with the IMesh client. The use of `RegOpenKeyA` with a specific path indicates interaction with application-specific configuration data stored in the Windows registry.

Code Snippet 2

```
push    eax                       ; push data size
lea     edx, [ebp+Data]           ; load address of data buffer
push    edx                       ; push data to write
push    1                         ; REG_SZ string type
push    0                         ; reserved
push    offset aDir0_0            ; "Dir0" registry value name
mov     eax, [ebp+phkResult]      ; load opened registry key
push    eax                       ; push registry handle
call    ds:RegSetValueExA         ; write shared directory entry
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This snippet writes a string value to the registry using `RegSetValueExA`. The value name `"Dir0"` suggests modification of directory-related configuration parameters. This behavior is consistent with altering application settings that define shared or accessible directories.

Code Snippet 3

```
push    4                           ; push data size (4 bytes)
lea     ecx, [ebp+var_3FC]          ; load address of DWORD value
push    ecx                         ; push data pointer
push    4                           ; REG_DWORD type
push    0                           ; reserved
push    offset aDisablesharing_0    ; "DisableSharing" value name
mov     edx, [ebp+phkResult]        ; load opened registry key
push    edx                         ; push registry handle
call    ds:RegSetValueExA           ; modify sharing setting
```

This snippet writes a `REG_DWORD` value named `"DisableSharing"`. Modifying such a parameter suggests that the program alters sharing-related configuration, potentially affecting how files are exposed or accessed within the application.

Code Snippet 4

```
movsx   ecx, [ebp+var_3F8]          ; load loop variable / drive index
cmp     ecx, 5Ah                    ; compare against 'Z'
...
push    offset aDirD_0              ; "Dir%d" formatted value name
lea     ecx, [ebp+Data]             ; load output buffer
push    ecx                         ; push buffer for formatted string
call    _sprintf                    ; build value name like "Dir0"
...
call    ds:RegSetValueExA           ; write generated directory entry
```

This snippet indicates iterative construction of registry value names (e.g., `"Dir0"`, `"Dir1"`, etc.) using `sprintf`. The loop suggests that multiple directory entries are written to the registry, which is consistent with systematic modification of application configuration across multiple entries.

Code Snippet 5

```
push    offset aSoftwareImeshC_0  ; IMesh user details key
push    80000001h                 ; HKEY_CURRENT_USER
call    ds:RegOpenKeyA            ; open user details registry key
...
push    16h                       ; push string size
push    offset aQuebonitoCompa_0  ; "QueBonito@Compartir.es"
push    1                         ; REG_SZ string type
push    0                         ; reserved
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
push    offset aEmail_0          ; "Email" registry value name
mov     edx, [ebp+phkResult]     ; load opened registry key
push    edx                      ; push registry handle
call    ds:RegSetValueExA        ; overwrite email setting
```

This snippet writes a string value to a registry field associated with user identity (e.g., `"Email"`). The modification of such fields suggests that the program alters application-specific user information stored in the registry.

Explanation Function 00401FC2 performs multiple registry operations targeting IMesh-specific configuration keys. The observed behavior includes writing directory-related values, modifying sharing parameters, and updating user-related information. This combination of actions is consistent with systematic reconfiguration of a peer-to-peer application's settings. In the context of the overall program behavior, these modifications may facilitate automated interaction with shared directories and application-level data.

Function 00401000 – Program Orchestration

Code Snippet 1

```
push    offset Mode              ; "rb" read mode
push    offset FileName          ; "C:\\Gusanillo QueBonito@Compartir.es"
call    _fopen                   ; open file if exists
...
push    offset Caption           ; message box title
push    offset Text              ; "Hola tio! soy el gusanillo\n" -- message
push    0                        ; no parent window
call    ds:MessageBoxA           ; display message
```

This snippet performs a file existence check using `fopen` on a specific path. The presence of a hardcoded filename suggests the use of a marker file, which may be used to indicate a prior execution state. If the file is found, a message box is displayed, indicating an alternative execution path depending on this condition.

Code Snippet 2

```
call    sub_401E43               ; perform initialization step
call    sub_401FC2               ; modify IMesh registry settings
mov     [ebp+var_14], 43h        ; start with drive letter 'C'
...
movsx   eax, [ebp+var_14]        ; load current drive letter
cmp     eax, 5Ah
          ; compare against 'Z'
jg      short loc_401112         ; stop after last drive
...
lea     edx, [ebp+Str]           ; load current drive string
push    edx                      ; push drive path
call    sub_401577               ; scan this drive
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This snippet illustrates a loop over drive letters, starting from `'C'` and continuing up to `'Z'`. For each drive, a helper function is invoked (`sub_401577`), which performs filesystem traversal. The prior invocation of `sub_401FC2` indicates that registry-related configuration is performed before initiating the drive-wide scanning process.

Explanation Function 00401000 coordinates the overall execution flow of the program. It includes an initial condition based on the presence of a specific file, followed by initialization steps and registry interaction. Subsequently, it iterates over available drives and invokes filesystem traversal routines. This combination of control flow, configuration modification, and systematic scanning is consistent with automated propagation behavior across multiple storage locations.

Function 004011EB – File Propagation

Code Snippet 1

```
push    offset Format           ; "%s/%s" path format string
lea     edx, [ebp+Str2]         ; load output path buffer
push    edx                     ; push destination buffer
call    _sprintf                ; build combined file path
...
lea     eax, [ebp+Str2]         ; load constructed path
push    eax                     ; push generated path
mov     ecx, [ebp+Str1]         ; load original path
push    ecx                     ; push original path
call    _strcmp                 ; compare source and target paths
```

This snippet constructs a new file path using a formatted string and compares it to an existing path. The comparison suggests that the program avoids processing identical source and destination paths, which is consistent with preventing redundant operations such as self-copying.

Code Snippet 2

```
push    offset aRb              ; "rb" read-binary mode
mov     edx, [ebp+Str1]         ; load source file path
push    edx                     ; push source file
call    _fopen                  ; open source for reading
...
push    offset aRb_0            ; "rb" read-binary mode
lea     eax, [ebp+Str2]         ; load target file path
push    eax                     ; push target file
call    _fopen                  ; test whether target exists
...
push    offset aWb              ; "wb" write-binary mode
lea     eax, [ebp+Str2]         ; load target file path
push    eax                     ; push target file
call    _fopen                  ; open target for writing
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This snippet demonstrates file handling logic involving both source and destination files. The program first opens a source file for reading, then checks whether the target file already exists. If necessary, it opens the target file in write mode, indicating conditional file creation or overwriting behavior.

Explanation Function 004011EB implements file processing logic involving path construction and conditional file handling. The combination of constructing destination paths, checking for existing target files, and opening source and destination files is consistent with file replication behavior. In the broader context of the program, this functionality may support automated propagation by copying files across different locations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.