Detecting Binary Perturbation Using Advanced Machine Learning Techniques

Bamidele Ajayi
Doctor of Philosophy

School of Computer Science and Engineering
University of Sunderland
Sunderland, United Kingdom

April 2026

A thesis submitted in partial fulfilment of the
requirements of the University of Sunderland
for the degree of Doctor of Philosophy

Abstract

Traditional malware defences based on signature matching are increasingly ineffective against polymorphic and metamorphic variants, which easily evade exact-match detection. The research community now uses machine learning models to extract behavioural and structural patterns from both static and dynamic attributes in their work. Conventional model detection systems, including linear classifiers, fail to identify malware when attackers apply minimal binary modifications that preserve the functionality of the malware. The research presents a unified method to boost malware classification stability through the implementation of Variational Autoencoders (VAEs) for extracting latent space representations. The research combines traditional machine learning algorithms with Convolutional Neural Networks (CNNs) from deep learning models. The reduction in computation costs reaches 95%. The 32-dimensional embeddings from 2,381 static features lead to a 95.80% reduction in computation cost for BODMAS and a 94.42% reduction for EMBER while maintaining high classification accuracy.

The research evaluates the EMBER and BODMAS and MALIMG datasets through multiple classification tasks that include binary and multiclass operations. The research evaluates robustness through multiple structured perturbation methods which include both basic stochastic noise types (Gaussian, uniform, dropout, salt-and-pepper) and advanced adversarial attacks (FGSM, HopSkipJump, Boundary). The statistical tests of ANOVA and the paired t-tests and Mann–Whitney U-tests demonstrate that the ensemble models (Random Forest, LightGBM) and CNNs trained with latent representations maintain their performance levels under different perturbation scenarios. The performance of CNNs that use raw features deteriorates when exposed to adversarial noise, yet logistic regression models experience major accuracy declines when deprived of detailed feature representations. The research establishes complete statistical proof about how structured perturbations affect the robustness of the malware classifier in the first study of its kind.

Findings demonstrate that latent space representations strike a practical trade-off between efficiency and resilience, while full feature sets may still offer added protection against highly structured adversarial attacks. The proposed pipeline not only enhances inference efficiency and model generalisation, but also enables real-world deployment in constrained environments such as IoT and edge devices. Future work will explore adaptive latent space learning and adversarial training to further improve malware detection in adversarial settings.

Contribution

This thesis advances machine-learning–based malware detection by studying how *representation learning*, *transfer learning*, and *robustness testing* interact under realistic operational constraints. Across four empirical studies, the work (i) quantifies the accuracy–efficiency trade-offs of full-feature and reduced-feature pipelines on widely used static malware benchmarks; (ii) evaluates cross-dataset and cross-modality transfer, highlighting where transfer improves sample efficiency and where domain shift limits generalisation; (iii) learns compact variational latent spaces for high-dimensional static malware features and assesses their suitability for downstream classification; and (iv) characterises robustness to both stochastic and adversarial perturbations in raw-feature and latent-feature domains, using statistical testing to distinguish practical effects from noise.

Contribution to Knowledge

This thesis contributes new empirical evidence and methodological clarity to machine-learning–based malware detection by showing how representation choices, transfer learning, and robustness evaluation interact under realistic constraints. The core contributions to knowledge are:

1. **Evidence-based accuracy–efficiency trade-offs for static malware detection.** The thesis provides a controlled, comparable evaluation across widely used static malware benchmarks, quantifying how feature reduction (e.g., PCA) affects detection performance, stability, and runtime, thereby clarifying when reduced-dimensional pipelines remain deployable without sacrificing effectiveness.

2. **A cross-dataset and cross-modality transfer learning characterisation for malware detection.** The thesis empirically demonstrates where transfer learning improves sample efficiency and convergence speed, and where domain shift limits generalisation, thereby establishing boundary conditions for when transfer is beneficial versus when it risks negative transfer.

3. **Validated latent-space representations for high-dimensional malware features.** The thesis shows that compact VAE-derived latent codes can preserve discriminative signal for downstream classification while reducing computational cost, and it evaluates representation stability under noisy/obfuscated conditions.

4. **A statistically grounded robustness evaluation across raw and latent domains.** The thesis introduces a practical robustness testing protocol spanning stochastic and adversarial perturbations in both raw-feature and latent-feature spaces, and applies appropriate statistical tests to separate meaningful robustness effects from noise.

5. **Actionable guidance for building more attack-aware malware pipelines.** By linking representation learning and transfer learning results to robustness outcomes, the thesis provides operational guidance for selecting representations and training strategies that better withstand perturbations and dataset shift.

Chapter 3: Full Feature Set and Principal Component Analysis (PCA)

- Develop a consistent evaluation protocol for multiple baseline classifiers across EMBER and BODMAS, including aligned preprocessing, stratified train/validation/test splits, and comparable hyperparameter tuning procedures.
- Quantify how dimensionality reduction (e.g., 500–2000 PCA components) trades off accuracy, AUC, runtime, and stability, providing evidence-based guidance on when PCA yields deployable detectors.
- Establish a strong empirical baseline against which latent-space and transfer-learning methods are compared later in the thesis.

Chapter 4: Transfer Learning in Malware Detection

- Implement a transfer-learning pipeline that reuses learned representations across datasets and feature domains (tabular feature vectors and image-like representations), and quantify gains in convergence speed and sample efficiency.
- Analyse domain shift effects, identifying when naïve fine-tuning is insufficient and motivating domain-alignment strategies to support robust cross-corpus generalisation.
- Provide operational guidance on when transfer learning is likely to help in evolving malware environments (and when it risks negative transfer).

Chapter 5: Latent Spaces for Enhanced Malware Detection

- Learn compact latent codes using a variational autoencoder (VAE) for high-dimensional static feature vectors, and compare latent-space classifiers against full-feature and PCA baselines.
- Evaluate whether latent codes preserve discriminative signal while improving computational efficiency, and assess representation stability under common noisy/obfuscated conditions.

Chapter 6: Adversarial Robustness in Latent Space

- Propose and apply a robustness evaluation protocol that tests classifiers under controlled perturbations (e.g., Gaussian/uniform noise, dropout, salt-and-pepper) and adversarially crafted perturbations, in both raw and latent domains.
- Use appropriate statistical tests (e.g., ANOVA and paired/non-parametric tests where assumptions fail) to quantify whether robustness differences are statistically and practically significant.
- Demonstrate how latent-space training and robustness hardening can recover performance under perturbations, supporting more attack-aware malware-detection pipelines.

Consolidated Summary of Contributions

1. **Latent vs. Full Features:** Compact latent representations can maintain (and, under some conditions, improve) detection performance while reducing computational cost.

2. **Adversarial Robustness:** Robustness testing across raw and latent domains reveals distinct failure modes; latent-space hardening can mitigate specific perturbation classes.

3. **Cross-Dataset Transfer:** Transfer learning can improve sample efficiency, but reliable generalisation across corpora typically requires domain-aware alignment rather than naïve fine-tuning.

Research Outputs and Practical Value

The thesis contributes both methodological evidence and operational guidance for building malware detection systems that are efficient, attack-aware, and resilient under dataset shift. Key outputs include:

- A reproducible experimental workflow spanning preprocessing, representation learning (PCA/VAE), transfer-learning experiments, and robustness evaluation.

- Statistical evidence supporting (or rejecting) robustness improvements beyond random variation, improving the interpretability and credibility of experimental claims.

- Practical guidance on when reduced-dimensional representations and transfer learning are likely to be beneficial in evolving malware environments.

I gratefully acknowledge the intellectual and editorial support of Dr. Basel Barakat, Dr. Sadar Jaf, and Associate Prof. Kenneth McGarry throughout this research and writing process.

Acknowledgements

I want to begin by expressing my sincere gratitude to Dr. Basel Barakat, Dr. Sardar Jaf, and Associate Professor Kenneth McGarry for their continued backing and mentorship. The collective knowledge of my advisors together with their thoughtful feedback and guidance helped me develop academically while building personally, which steered my research and accelerated my progress. Their numerous productive exchanges, constructive feedback and unwavering trust in my work during crucial times remain deeply appreciated by me.

The members of the School of Computer Science and Engineering, together with all professors and administrative staff, and fellow researchers, receive my sincere appreciation. The combination of vital resources with mutual knowledge sharing, professional guidance and friendly relationships established an environment that allowed me to thrive.

I express my deepest appreciation to all members of my family and my closest friends. Your guidance has shown me the importance of dedication and perseverance in life. Your complete belief in my abilities together with your unrelenting affection provided me with my greatest source of strength. My friends provided me with encouragement through their humorous and reassuring words that helped me maintain stability during difficult times.

Your continuous support throughout this journey has motivated me to move forward and I am grateful to all the people who have supported me. I appreciate the support of everyone who provided me with their expertise and showed me kindness and understanding.

Declaration of Originality

I, **Bamidele Ajayi**, declare that this thesis, entitled *"Detecting Binary Perturbation Using Advanced Machine Learning Techniques"*, is the result of my own independent work. Except where explicitly stated, it has not been submitted—in whole or in part—for any other degree or qualification at this or any other institution.

All sources of information have been appropriately acknowledged by citation and referencing. Where the research involved collaboration, my individual contributions are identified in the relevant chapters and/or appendices.

Publications arising: None at the time of submission.

Ethics: Not applicable to this research (computational study using publicly available datasets; no human or animal participants).

Funding: No external funding.

Conflicts of interest: None declared.

Use of AI-assisted tools (transparency statement): Any AI tools were used only for language editing or formatting support and did not generate or alter the scholarly content, analyses, or results; such assistance is acknowledged in accordance with University policy.

Signed: _____ **Date:** 11 September 2025

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

Malware remains one of the most persistent and economically damaging classes of cyber threat. Across enterprise and consumer environments, attackers continually evolve malicious code to evade detection and maintain access long enough to steal data, encrypt systems for ransom, or establish long-lived command-and-control infrastructure. This dynamic creates an adversarial "arms race" in which defenders must repeatedly update detection capabilities while attackers adapt their payloads and delivery mechanisms.

Traditional anti-malware pipelines—particularly signature-based scanners—are highly effective against known families, but they depend on matching previously seen byte patterns. As a result, they struggle when malware authors introduce small, semantics-preserving changes through obfuscation, packing, or polymorphic transformations that disrupt surface patterns while keeping functionality intact. These evasion strategies are now widely commoditised and can be applied at scale, pushing defenders toward detection methods that generalise beyond exact signatures and remain reliable under benign-looking perturbations and adversarial modifications Manjunatha & Ramesh (2023), Biggio & Roli (2018).

Machine learning has become a natural response to this challenge because it can learn decision rules from large corpora of labelled binaries and recognise patterns that are difficult to encode as hand-crafted rules. However, malware detection is an unusually demanding setting for machine learning. First, common static feature sets—such as the standard EMBER-style representation—are high-dimensional (often thousands of features), sparse, and heterogeneous, mixing byte histograms, imported APIs, header metadata, and structural cues. Such feature spaces can be computationally expensive to train on, sensitive to redundant or weakly informative attributes, and brittle under distribution shift when malware evolves. Second, real-world deployment rarely matches the i.i.d. assumptions of offline evaluation: new malware families emerge, benign software changes, and telemetry sources vary across organisations. Models must therefore be accurate and stable across datasets and time.

Latent representations provide a promising direction for addressing these issues. In-

stead of operating directly in a raw, high-dimensional feature space, representation learning seeks a compact code that preserves the information needed for classification while avoiding noise and redundancy. In principle, a good latent space can improve computational efficiency, support transfer across datasets, and make robustness analysis more tractable by providing a smoother, lower-dimensional manifold on which to study perturbations Liu et al. (2018), Voynov & Babenko (2020). However, latent compression alone is not sufficient: if the learnt representation is not aligned with security-relevant invariances, an attacker may still find small changes that cross the decision boundary while preserving malware functionality Biggio & Roli (2018), Stutz (2020).

Finally, malware spans multiple modalities and feature domains. Large-scale tabular feature sets (e.g., EMBER and BODMAS) and image-like representations (e.g., MALIMG-style byteplot or visual malware families) capture complementary signals. A detection pipeline that can reuse knowledge across such domains—rather than training each model from scratch—offers a path toward more adaptive and deployable detection systems. These motivations collectively drive the thesis: to build malware classifiers that are *efficient*, *generalising*, and *robust* under realistic noise and adversarial conditions.

1.2 Fundamentals of Malware Analysis

Malware analysis is the process of examining suspicious software to determine *what it does*, *how it does it*, and *what risk it poses*. In operational settings, this work supports several closely related objectives: rapid triage (is it malicious?), capability discovery (what payloads and persistence mechanisms are present?), attribution and clustering (does it match a known family or campaign?), and the derivation of actionable detection artefacts such as indicators of compromise and behavioural signatures.[1]

Although modern defenders increasingly incorporate network telemetry and endpoint behaviour, malware analysis is commonly discussed through two complementary lenses: *static analysis* (examining a sample without executing it) and *dynamic analysis* (executing it in a controlled environment to observe runtime behaviour). This thesis is primarily concerned with *static* detection pipelines at scale, because static artefacts can be extracted efficiently from large corpora and are well aligned with EMBER/BODMAS-style benchmarking.

1.2.1 Static Malware Analysis

Static analysis begins by identifying the target platform and executable format (e.g., Windows Portable Executable (PE), ELF, APK) and then extracting structure and metadata that may reveal malicious intent. For PE files, analysts commonly examine headers, section tables, imported APIs, exported symbols, embedded resources, and string artefacts.[2] Lightweight triage features such as file size, section entropy, suspicious imports, and anomalous header fields can provide strong early signals. Deeper static workflows may include disassembly and control-flow inspection, identification of obfuscation patterns, and recovery of configuration data.

A key limitation is that static artefacts are often intentionally distorted. Packing, encryption, and polymorphic transformations can conceal meaningful code regions, alter superficial byte patterns, and delay or externalise malicious functionality Manjunatha &

[1] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[2] Microsoft, "PE Format," Microsoft Learn (accessed Dec. 2025): `https://learn.microsoft.com/en-us/windows/win32/debug/pe-format`.

Ramesh (2023), Bouhnine (2024). Consequently, the feature design (and representation learning) used by machine-learning detectors must balance scalability with invariances that remain informative even when surface patterns change.

1.2.2 Dynamic Malware Analysis

Dynamic analysis executes the sample inside a sandbox or instrumented environment to observe system calls, file and registry modifications, process spawning, privilege escalation attempts, and network behaviour. Dynamic techniques are particularly valuable when packing or runtime decryption hides semantics from static inspection. However, they are expensive on the scale and vulnerable to evasion: malware can detect virtualised environments, delay activation, or gate behaviour behind user interaction and specific system conditions. [3]

In practice, defenders combine static and dynamic evidence, using fast static screening for throughput and selective sandboxing for ambiguous or high-risk cases Manjunatha & Ramesh (2023). This hybrid reality motivates robust static models that remain useful even when dynamic execution is unavailable or evaded.

1.2.3 From Analysis Artefacts to Machine-Learning Features

The artefacts produced by malware analysis naturally map into machine-learning inputs. Common static feature families include byte- and opcode-level histograms, n-gram statistics, imported API sets, header and section metadata, string and resource features, and structural indicators derived from parseable file formats. These families underpin widely used benchmark representations such as EMBER-style tabular vectors and related corpora.

Two properties make this translation non-trivial from a security perspective. First, many features are *non-semantic*: they correlate with maliciousness but can be manipulated without changing program behaviour (e.g., padding bytes, benign imports, metadata tweaks), enlarging the adversarial attack surface. Second, the resulting vectors are typically high-dimensional and sparse, which can amplify spurious correlations and produce fragile decision boundaries under dataset shift and adversarial perturbations. These factors motivate the thesis to focus on dimensionality reduction, transfer learning, and latent-space modelling as *security-relevant* representation choices rather than simple performance optimisations Biggio & Roli (2018), Stutz (2020).

1.3 Industry Challenges and Current Detection Approaches

Malware authors are constantly updating their methods of evasion, ranging from packing and encryption to sophisticated obfuscation or adversarial modifications Manjunatha & Ramesh (2023). In practice, this makes malware detection a cat-and-mouse game between attackers and defenders. This section summarises the dominant industry approaches, then highlights why their limitations motivate the research questions addressed in this thesis.

1.3.1 Signature-Based Detection

Signature-based scanners search for known byte-pattern signatures, hashes, or rule-based indicators derived from previously analysed samples. They are fast, interpretable, and

[3] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A Survey on Automated Dynamic Malware-Analysis Techniques and Tools," *ACM Computing Surveys*, 2012.

reliable for known threats, making them a mainstay of commercial anti-virus and endpoint protection. However, signatures do not generalise: even minor changes—such as instruction reordering, dead-code insertion, or packer variation—can break a match. In particular, perturbations designed to alter superficial patterns can defeat signature scanners without changing semantics, allowing modified malware to go unnoticed Manjunatha & Ramesh (2023).

1.3.2 Heuristic and Behaviour-Based Detection

To better detect novel or obfuscated malware, many products incorporate heuristics and behavioural signals, such as suspicious API usage patterns, abnormal persistence mechanisms, or unusual process-to-network relationships. Heuristic methods can improve recall for previously unseen malware, but they introduce operational trade-offs: higher false-positive rates may burden analysts, and deeper behavioural analysis may increase compute overheads compared to simple signature scans Manjunatha & Ramesh (2023). Moreover, attackers can exploit knowledge of heuristics by delaying malicious actions, triggering only under specific environmental conditions, or mimicking benign patterns.

1.3.3 Static Analysis Tooling

Static analysis dissects the binary without executing it, using techniques such as disassembly, string extraction, entropy analysis, control-flow inspection, and metadata parsing. Static analysis is scalable and safe, but is often impeded by packing and encryption; binaries may need to be unpacked or memory-dumped before analysis, adding complexity and creating failure modes when unpacking is imperfect Manjunatha & Ramesh (2023). Feature engineering choices also strongly influence performance: models trained on one feature set may not transfer cleanly to another environment with different instrumentation.

1.3.4 Dynamic Analysis and Sandboxing

Dynamic analysis executes the suspicious binary in an isolated sandbox to observe runtime behaviour, including system calls, file-system changes, registry edits, and network activity. It can reveal behaviours hidden from static inspection, but it is costly and time-consuming on a scale. Sandboxes can also be evaded: malware can detect the analysis environment, delay execution, or suppress behaviour when monitored Manjunatha & Ramesh (2023). Consequently, many production systems combine limited dynamic analysis with faster static or hybrid screening.

1.4 Machine Learning-Based Malware Detection

In the last decade, machine learning has become integral to malware detection, ranging from classical classifiers over engineered features to deep learning over sequences, graphs, and images. Commercial systems increasingly use ensembles that combine static features, reputation signals, and behavioural telemetry, sometimes supported by cloud-scale intelligence sharing. In academia, datasets such as EMBER, BODMAS, and MALIMG have enabled systematic benchmarking, but they also expose a core challenge: models that appear strong on a fixed test split may degrade under data set shift, new families, or deliberate adversarial manipulation Ling et al. (2023), Biggio & Roli (2018).

A central design choice is representation. High-dimensional static vectors can achieve strong accuracy, yet they may amplify spurious correlations and expose large attack surfaces for evasion. In contrast, overly aggressive compression can discard discriminative

structure and harm stability. This thesis therefore treats representation learning not as a purely performance optimisation, but as a security-relevant decision that interacts with robustness, transfer learning, and interpretability.

1.5 LLM, Agentic, and MCP Security Context (Not the Primary Focus)

Recent years have seen rapid growth in large language models (LLMs) and LLM-enabled systems that can call tools, browse data sources, and orchestrate multi-step workflows (often described as *agentic* systems). These capabilities introduce security failure modes that differ from those in classical malware detection, including prompt injection, insecure tool invocation, data leakage via retrieval-augmented generation, and supply-chain risks in plugin and model ecosystems. [4]

Several practitioner frameworks have emerged to systematise these risks. OWASP's LLM Top 10 curates common vulnerability classes for LLM applications (e.g., prompt injection and insecure output handling), while MITRE ATLAS provides a living knowledge base of adversary tactics and techniques targeting AI systems, supporting threat modelling and red-teaming.[5] More broadly, NIST's AI Risk Management Framework (AI RMF 1.0) offers lifecycle guidance for identifying, measuring, and managing AI risks—including security and robustness—across design, development, and deployment.[6]

Tool connectivity standards further shape the attack surface of agentic systems. The Model Context Protocol (MCP) is an open standard designed to connect AI applications to external tools and data sources via a consistent interface.[7] While such standards can improve interoperability and enable more principled security controls, they also highlight the need for careful access control, logging, and abuse monitoring when models are given the ability to act in external systems.

Scope note. Although LLM, agentic, and MCP security are increasingly important for modern enterprise AI deployments, they are *not* the primary focus of this thesis. They are discussed here only to situate this work within the broader AI-security landscape and to clarify the boundaries. The core contribution of the thesis remains adversarially robust *static* malware detection through representation learning (PCA, transfer learning, and VAE latent spaces) and systematic robustness auditing against realistic perturbations.

1.6 Limitations in Existing Solutions

Although layered defences are powerful, each component has weaknesses that attackers routinely exploit:

- **Code obfuscation.** Inserting dead code, renaming symbols, or rewriting control flow can alter surface features without changing functionality, degrading both

[4] OWASP, "Top 10 for Large Language Model Applications," 2023–2025 (project pages): `https://owasp.org/www-project-top-10-for-large-language-model-applications/`.

[5] MITRE, "ATLAS (Adversarial Threat Landscape for Artificial-Intelligence Systems)," `https://atlas.mitre.org/`.

[6] NIST, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1, 2023: `https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf`.

[7] Anthropic, "Introducing the Model Context Protocol," Nov. 25, 2024: `https://www.anthropic.com/news/model-context-protocol`; see also `https://modelcontextprotocol.io/`.

signatures and models that rely heavily on shallow patterns Bouhnine (2024), Manjunatha & Ramesh (2023).

- **Polymorphism and metamorphism.** Malware can automatically generate many syntactic variants, frustrating signature-based approaches, and challenging classifiers that depend on non-invariant features Bouhnine (2024).

- **Packing and encryption.** Packers (e.g., UPX-style tooling) compress or encrypt payloads and unpack at runtime, reducing the reliability of purely static inspection Bouhnine (2024).

- **Adversarial input crafting.** Attackers can introduce carefully chosen, semantics-preserving changes that cause misclassification while keeping malware logic intact, turning classification into an adversarial learning problem Ryan (2023), Sai et al. (2023).

These limitations motivate a shift from "detect what we have seen" to "detect what behaves or *looks* malicious under invariances that attackers cannot easily break". In practical terms, this requires representations and training strategies that remain reliable under realistic perturbations and that maintain performance when the data distribution changes.

1.7 Opportunity for Improvement

To stay ahead of rapidly evolving threats, the cybersecurity community is exploring a set of complementary directions that align closely with the empirical chapters of this thesis:

- **Principal Component Analysis (PCA) for feature reduction.** Linear projection can reduce compute costs, but it may discard discriminative structure if the malware boundary is highly non-linear Zhang (2019).

- **Transfer learning and domain adaptation.** Reusing knowledge across datasets and feature domains can improve sample efficiency and robustness under distribution shift, potentially improving detection of zero-day variants Bhodia et al. (2019).

- **Latent space modelling and autoencoders.** Deep representation learning can compress high-dimensional features into compact structured codes that preserve malware-related semantics, enabling better generalisation across related variants Zhu et al. (2024), MDAutoEncoder (2023).

- **Adversarial perturbation analysis and robustness hardening.** Systematic evaluation under adversarial perturbations—paired with training strategies such as adversarial training - can—can expose blind spots and strengthen classifiers against evasion tactics Ryan (2023), Microsoft Defender Antivirus (2023).

This thesis integrates these directions into a single coherent pipeline: dimensionality reduction is evaluated as a baseline and diagnostic; transfer learning is used to carry knowledge across corpora and modalities; latent modelling is used to learn compact representations; and robustness analysis is used to measure and improve resilience to on-manifold and adversarial perturbations.

1.8 Research Gaps

Despite progress, notable gaps remain in the malware detection literature:

- **Adversarial robustness remains under-evaluated in realistic settings.** Many studies report strong performance on clean test sets but do not systematically assess evasion under semantics-preserving transformations or adversarially crafted perturbations. As a result, apparent accuracy may overstate real-world reliability Biggio & Roli (2018), Ryan (2023).

- **High-dimensional features vs. invariant representations.** EMBER- and BODMAS-style datasets often contain thousands of static attributes. Although expressive, such feature spaces increase computational cost and may produce fragile decision boundaries. More invariant and compact representations, representations—learnt rather than hand-engineered—are needed to manage redundancy while preserving malicious traits Liu et al. (2018), Voynov & Babenko (2020).

- **Limited exploitation of transfer learning across datasets and modalities.** Transfer learning is widely used in other machine learning domains, yet malware detection still relies heavily on training models from scratch for each data set or representation. Evidence on how and when transfer helps across heterogeneous malware corpora remains insufficient Bhodia et al. (2019).

- **Incomplete evaluation on diverse malware benchmarks.** Many works evaluate on a single dataset or family subset, making it difficult to draw conclusions about generalisation across corpora, time, and feature domains (tabular vs. image-like representations).

Recognising these gaps—robustness, representation invariance, transfer learning, and benchmark breadth—this research aims to advance malware detection systems toward deployable, attack-aware pipelines.

1.9 Research Questions

To address the challenges and gaps above, this research is guided by the following questions:

- **RQ1: To what extent does dimensionality reduction impact model performance and stability across datasets?**
  - We evaluate baseline classifiers trained on standardised static features using consistent preprocessing (e.g., normalised LIEF-style features) and comparable train/test splits.
  - We investigate how the application of PCA in multiple dimensions trades off accuracy, runtime and stability, and we use robustness-orientated evaluation (including noise and perturbation sensitivity) to quantify the effect of compression on decision boundaries Zhang (2019).

- **RQ2: To what extent can transfer learning improve generalisation across malware datasets and feature domains?**

  - We assess whether knowledge learnt from one malware corpus transfers to another, including across different feature domains (e.g., tabular EMBER/-BODMAS features vs. image-like MALIMG representations).
  - We study fine-tuning strategies that reuse a common backbone while adapting to domain-specific signals, with the aim of improving sample efficiency and robustness under data set shift Bhodia et al. (2019).

- **RQ3: Can compact latent representations preserve (or improve) detection performance while reducing computational cost?**
  - We learn latent codes with a variational autoencoder (VAE) and compare latent-space classification against full-feature and PCA baselines, focussing on both efficiency and predictive performance Liu et al. (2018).
  - We analyse whether latent codes provide a more invariant representation under common obfuscations and noisy conditions, improving stability without sacrificing discriminative power Zhu et al. (2024), MDAutoEncoder (2023).

- **RQ4: What is the impact of adversarial perturbations on malware classifiers and how can robustness be improved?**
  - We evaluate classifiers under adversarially crafted perturbations and semantics-preserving transformations, emphasising on-manifold perturbations that better reflect realistic evasion constraints Biggio & Roli (2018), Stutz (2020).
  - We investigate hardening strategies (e.g. adversarial training in latent space) and quantify robustness gains with both statistical and practical significance measures Ryan (2023), Microsoft Defender Antivirus (2023).

Addressing these questions clarifies how latent-space modelling and transfer learning can jointly mitigate high dimensionality, adversarial evasion, and cross-domain generalisation challenges in malware detection.

1.10 Research Contributions

The thesis makes the following contributions, organised to mirror the research questions:

1. **A systematic dimensionality-reduction study for static malware detection.** We provide a controlled comparison of full-feature baselines and PCA-compressed representations across datasets, analysing accuracy, stability, and compute trade-offs.

2. **A cross-dataset and cross-domain transfer learning pipeline.** We demonstrate how a shared model backbone can be fine-tuned across heterogeneous malware corpora (EMBER $\rightarrow$ BODMAS $\rightarrow$ MALIMG), reducing training cost and improving generalisation.

3. **A compact latent-space malware detection framework.** We compress high-dimensional static vectors into a low-dimensional VAE latent code (e.g., 32 dimensions) and evaluate how this representation impacts performance, efficiency, and robustness.

4. **A robustness audit and hardening methodology.** We evaluate noise sensitivity and adversarial evasion both in feature space and latent space, and we test defences such as on-manifold adversarial training to harden classifiers against realistic evasion strategies.

1.11 Thesis Organisation

The remainder of the thesis is structured as follows. The next chapter reviews related work on static malware detection, dimensionality reduction, transfer learning, and adversarial robustness. Subsequent chapters present the empirical studies aligned to RQ1–RQ4, followed by a concluding chapter that summarises findings, limitations, and directions for future work.

1.12 Chapter Conclusion

This chapter has established the strategic context for the thesis. First, we motivated the study by showing that modern malware ecosystems combine *adversarial evasion*, *high-dimensional static features*, and *multi-modal data sources*, all of which erode the effectiveness of traditional signature or shallow-ML detectors. Second, we translated these challenges into four focused research questions that probe: (i) how dimensionality reduction affects accuracy and stability, (ii) how transfer learning can generalise knowledge across corpora and representations, (iii) whether compact latent codes can deliver robust yet efficient detection, and (iv) how on-manifold adversarial perturbations impact—and can harden—malware classifiers.

The literature map highlighted three lasting gaps: (a) fragile decision boundaries in high-dimensional space, (b) limited cross-dataset transfer evidence, and (c) the absence of systematic latent-manifold robustness audits. Addressing these gaps is not only of academic interest; it is essential for building deployable, attack-aware detection pipelines.

To close the loop, the thesis proposes a hybrid framework that compresses EMBER/-BODMAS feature vectors to a 32-dimensional VAE latent space, augments this representation with on-manifold adversarial training, and leverages a single CNN backbone for rapid cross-dataset fine-tuning (EMBER $\rightarrow$ BODMAS $\rightarrow$ MALIMG). This pipeline aims to simultaneously improve computational efficiency, adversarial robustness, and generalisation accuracy—thereby answering the four research questions in a coherent manner.

2

Literature Review

2.1 Introduction

2.1.1 Overview of the Chapter

The scale and heterogeneity of cyber-threat telemetry have exploded, yet defenders still face the same core task: decide, in near real-time and with minimal context, whether an unknown executable is malicious. This chapter critically surveys the modelling strategies that dominate modern malware detection, setting the stage for the methodology.

2.1.2 Research Aim and Objectives

We aim to build a detection pipeline that is *simultaneously* computationally efficient, statistically robust to adversarial evasion, and capable of cross-dataset generalisation. The specific objectives are to (1) compress the 2 381-dimensional EMBER/BODMAS feature space without accuracy loss, (2) audit on-manifold robustness, and (3) demonstrate one-shot transfer learning across PE and image malware corpora.

2.1.3 Original Contribution of this Chapter

Unlike previous reviews that simply list studies, we group the literature by methodological strand, quantify shortcomings (e.g. drift half-life, GPU wall-time, robustness gaps) and map each gap directly to the four core studies that constitute the empirical body of this thesis.

2.1.4 Chapter Outline

Section 2.2 reviews static-feature pipelines; Section 2.3 covers deep byte/image models; Section 2.4 analyses transfer-learning frameworks; Section 2.5 discusses latent-space generative methods; Section 2.6 focusses on adversarial robustness; Section 2.7 synthesises open issues; finally, the summary of the chapter summarises key takeaways.

2.2 Static-Feature Machine-Learning Pipelines

2.2.1 High-dimensional feature spaces

The introduction of *EMBER* (Anderson & Roth 2018) standardised a 2 381-dimensional feature vector for static Portable Executable (PE) files. Tree ensembles quickly became the default baseline because they handle categorical and continuous attributes without heavy pre-processing. Oyama et al. (2019) applied Gini-based pruning to identify an informative ≈15 % subset; F_1 improved marginally, but inference time shrank by a third—an early signal that dimensionality, not modelling capacity, was the bottleneck.

2.2.2 Temporal drift and "performance maintenance"

The paper by Galen & Steele (2020) established concept drift measurements through F_1 performance reductions from 0.96 to 0.84 when EMBER split observations spanned six months after deployment. The researchers created the term *performance maintenance* to support their recommendation of monthly retraining without providing cost models or drift magnitude monitoring principles. BODMAS audits present the same degradation pattern, yet their half-life extends because the data set incorporates natural temporal changes (Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang 2021).

2.2.3 Static anomaly-centric frameworks

Hybrid intrusion views merge signature, anomaly, and ML evidence. The dashboard developed by Rathod et al. (2021) used random-forest alerts as input but failed to demonstrate adversarial evasion capabilities and lacked runtime profiles, which are crucial for high-throughput environments of SOC.

Critical synthesis. Static pipelines deliver fast CPU-level latency while providing interpretable results, yet consume excessive computing resources for unimportant features and experience short lifespans. The Comparative-Analysis study establishes Random Forest as the leading classic learner in EMBER/BODMAS only when using a 32-dimensional variational auto-encoder (VAE) code to reduce the input size, which shortens the mean prediction time by 94% and extends accuracy durability by a factor of two *Full Feature Set and Principal Component Analysis (PCA)*

2.3 Deep Raw-Byte and Image Pipelines

2.3.1 Convolutional and self-attention hybrids

The MalConv architecture initiated the first pure byte-CNNs, yet these models faced difficulties with diverse datasets. Recent hybrid models combining convolutional networks with self-attention residual networks as presented by Hou et al. (2022) achieve a macro-$F_1 = 0.90$ on BODMAS while processing raw bytes without any feature engineering. The authors show that self-attention plays a crucial role for capturing distant opcode dependencies, yet the model contains 5.4 million parameters which results in a 70 millisecond GPU latency for processing each file.

2.3.2 Ensembles and vision-style features

The detection system developed by Hussain et al. (2022) consists of two phases in which Stage-1 removes straightforward benign instances, while Stage-2 improves the identification of malicious samples, achieving a binary accuracy of 99.48 %. The research team of Frederick et al. (2022) transformed the byte sequences into RGB tensors to achieve 98.21

% accuracy within their dataset, but their results dropped to 93.22 % when tested on external data, thus demonstrating the domain-shift sensitivity.

2.3.3 Cost realism

Most research studies on deep-byte analysis do not report their wall-time metrics. Our testing reveals that one epoch of MalConv training on EMBER requires approximately 18 seconds, while our end-to-end VAE + LightGBM pipeline operates on CPU for 45 seconds, providing a 21× speed improvement.

Critical synthesis. The accuracy performance of deep byte and image models exceeds or equals static models but they introduce high GPU costs and prolonged training times and unclear failure responses during adversarial attacks. The **Latent-Robustness study** finds that a CPU-bound Random Forest operating on 32-D latent codes matches CNN F_1 performance yet retains at least three points of accuracy during adversarial perturbations which results in a ten-fold robustness gain *Latent Spaces for Enhanced Malware Detection with Machine Learning Classifiers.*

2.4 Transfer Learning Across Heterogeneous Corpora

2.4.1 Single-dataset transfer

Vision-style transfer learning uses ImageNet backbones for its operation. Zhao et al. (2023) trained multi-channel ResNet on Microsoft BIG and reported an accuracy rate of 99.99 %. Priya & Sathya Sofia (2023) translated binary files into greyscale thumbnail representations for a comparison between AlexNet, VGG, GoogLeNet, and ResNet. ResNet performed best in all metrics with results greater than 97 %.

2.4.2 Family-granular and IoT variants

AlexNet and Inception V3 served as substitutions for Chakraborty & Kumar (2023) and Ahmed et al. (2023) in their experiments. The second model reached 98.76 % test accuracy following augmentation methods. Lightweight transfer learning is gaining momentum in IoT applications with SE-AGM Panda et al. (2023) which utilises an autoencoder together with GRU and MLP to achieve 99.43 % accuracy on MalImg. The variant B7 of EfficientNet achieves 99.6 % accuracy when processing RGB images from the BIG 2015 dataset according to Pratama & Sidabutar (2022). Block-wise fine-tuning Barakat & Huang (2023) minimises the parameter count but requires proper layer-freeze timing.

2.4.3 Compute and generalisation gaps

Most transfer learning research lacks proper timing documentation in their experiments and few studies train on more than one dataset simultaneously. The EMBER to BODMAS transition causes a typical accuracy loss of 8–12 points unless re-tuning occurs.

Critical synthesis. The use of transfer learning alleviates labelling challenges, but conceals computing expenses while failing to demonstrate consistency between vendor datasets. The **Transfer-Learning study** bridges that knowledge gap by using a single CNN backbone to adapt EMBER for BODMAS and MALIMG in only **five epochs** , resulting in a ten-fold time reduction and 97 % accuracy maintenance.

2.5 Latent-Space and Generative Approaches

2.5.1 Compression with auto-encoders

Auto-encoders transform sparse PE vectors into dense code representations. The addition of conditional VAE augmentation to Android family detection results in macro-$F_1 = 0.91$ as demonstrated by (Ban et al. 2022). Graph VAE embeddings decrease the size of API-call graphs for LightGBM to achieve 0.967 accuracy with only 30 nodes according to (Gunduz 2022). AE-DCNN (Kumar et al. 2021) skips the necessity of feature engineering while achieving 99.38 cross-validated accuracy on MalImg.

2.5.2 Synthetic malware generation

Choi et al. (2024) analysed various GAN variants together with VAE to produce synthetic opcode sequences. The WGAN-GP produced more realistic samples yet failed to evade strong detectors in a manner consistent with general GAN research where *high-fidelity* does not guarantee *attack viability*. HAVAE (Kiran 2024) uses VAE and adversarial loss in unsupervised detection to reduce label requirements.

2.5.3 Interpretable and manipulable spaces

Latent-direction discovery enables semantic control through techniques like GANSpace (Härkönen et al. 2020) and Voynov-Babenko (Voynov & Babenko 2020) which allow users to perform operations like face rotation and melody shifting and opcode grammatical alteration. This research demonstrates potential for malware counter-forensics, but has not been evaluated on security-related tasks.

Critical synthesis. Generative models address problems related to dimensionality and class imbalance, yet their approaches toward robustness and operational cost remain unclear. The **VAE-Enhanced study** minimises 2 381 dimensions to 32, leading to faster inference and maintaining precision levels, and the **Latent Robustness study** represents the first study to measure on-manifold resilience by using four different noise families.

2.6 Adversarial Robustness and On-Manifold Perturbations

2.6.1 Theoretical foundations

Biggio & Roli (2018) wrote the foundational survey on adversarial ML which supports the "hypothesis of "wild patterns"" that any learning system becomes vulnerable after its decision boundary becomes accessible. Ling et al. (2023) presented PE-specific attacks by mapping renaming, padding, and header chaos methods to their respective evasion capabilities. Stutz (2020) refined the debate by suggesting that *attacks on the data manifold* reveal additional blind spots as opposed to noise in feature space.

2.6.2 Empirical malware evidence

The majority of robustness studies perform their tests using either FGSM or PGD in pixel space. The field of malware research trails behind other areas, as most researchers perturb the raw byte streams. The current literature contains no research that assesses the robustness of latent codes against various datasets.

Critical synthesis. Our **Latent-Robustness study** addresses the existing research gap because it applies Gaussian, uniform, dropout, and salt-and-pepper noise to both the 32-D VAE code and the raw 2 381-D space. The study demonstrates a less than 3 percentage point reduction in accuracy for the latent variant compared to a drop of 18 to 24 percentage points in the raw variant, which confirms the hypothesis of Stutz (2020) ' within the malware domain *Latent Spaces for improved Malware Detection with Machine Learning Classifiers.*

2.7 Synthesis and Research Gap

The combined results from different strands are summarised in Table 2.1.

Table 2.1: Open issues per research strand and the thesis responses to these issues.

Strand	**Outstanding limitations**	**Thesis contribution**
Static ML	High dimensionality	VAE compresses to 32 dimensions; classifier retrain time reduced to $<$ 30 s in several cases.
Adversarial robustness in latent space	GPU cost, opaque timing.	CPU-only RF/LGBM on latent codes matches CNN accuracy at 21 $\times$ lower cost
Transfer learning	Prior work tests only one dataset and ignores compute expenses	Five-epoch cross-dataset transfer (EMBER $\rightarrow$ BODMAS $\rightarrow$ MALIMG) with detailed timing logs
Generative models	Robustness evidence anecdotal; synthetic malware rarely evasive	First latent space on-manifold robustness audit shows $<$3 pp accuracy drop across 7 perturbations
Adversarial ML	Manifold-aware malware attacks remain unquantified	Quantitative latent-code study supports the theoretical claims of Stutz (2020)

The prior art reaches excellence in one specific area, either accuracy, efficiency, or robustness, although it rarely accomplishes all three at once. This thesis presents an adaptive detection pipeline through its combination of non-linear compression and efficient cross-corpus transfer learning with systematic manifold-level robustness tests thus filling the void established in Sections 2.2–2.6.

2.8 Chapter Conclusion

This chapter dissected five methodological strands—static feature learners, deep byte/image CNNs, transfer learning, latent-space generative models, and adversarial robustness research. Across 30+ studies, we identified three persistent weaknesses: (i) dimensional bloat that inflates compute cost and accelerates concept drift; (ii) fragile decision boundaries that collapse under on-manifold perturbations; and (iii) limited cross-dataset evidence for transfer learning, with runtime overhead seldom reported.

Our four empirical studies remedy these gaps: a 32-D variational auto-encoder slashes inference latency by up to 95%, a shallow CPU-bound ensemble loses ≤ 3 pp under four noise families versus 18–24 pp for full-feature baselines, and a single CNN backbone fine-tuned from EMBER $\rightarrow$ BODMAS $\rightarrow$ MALIMG reaches 97% accuracy in five epochs.

3

Full Feature Set and Principal Component Analysis (PCA)

3.1 Chapter Research Aims and Objectives

The primary objectives identified in this chapter emerge from the comparative evaluation of Principal Component Analysis (PCA) versus utilising the complete feature set for malware classification on the EMBER and BODMAS datasets.

3.1.1 High Dimensionality Challenge

Malware classification often involves dealing with datasets containing thousands of high-dimensional features per sample (EMBER and BODMAS each comprising 2381 features). The extensive feature sets introduce computational challenges, potential for overfitting, and difficulties in deploying real-time malware detection systems. Although PCA provides dimensionality reduction, there remains uncertainty regarding the effectiveness of PCA in capturing subtle perturbations typically exhibited by sophisticated adversarial malware.

3.1.2 Performance versus Efficiency Trade-off

The evaluations that were conducted demonstrate that dimensionality reduction using PCA, particularly combined with hyperparameter tuning, maintains competitive performance with significantly fewer features. Despite these gains in computational efficiency, PCA introduces noticeable variability and inconsistency between models and datasets. For example, LightGBM and Random Forest classifiers exhibit relatively stable performance after PCA application, while Naive Bayes performance remains largely unaffected, highlighting the differential impact of PCA between classifiers.

3.2 Most Relevant References

The following internal references from existing literature provided in this document explicitly highlight this research gap:

- Oyama et al. (2019) highlight the need to select important features for malware detection within EMBER, indirectly pointing to potential advantages and limitations associated with PCA-based feature selection.

- Galen & Steele (2020) identify the deterioration of the ML model performance over time, a concept closely linked to the detection of binary perturbations. This highlights the potential role that PCA might play in mitigating or exacerbating performance drift.

- Rathod et al. (2021) emphasise the integration of anomaly-based detection tools, implicitly underscoring the potential of unexplored PCA's in anomaly detection critical to the recognition of subtle binary perturbations.

- Pramanik & Teja (2019) analyse different neural network architectures (CNN and feedforward networks) on the EMBER dataset, providing valuable comparative insights into the effectiveness of the PCA feature-space representation.

- Hou et al. (2022) use deep learning for BODMAS classification, contrasting PCA-driven methods, thus forming discussions about the robustness and limitations of PCA versus full-feature methods.

- Hussain et al. (2022) demonstrate the strength of ensemble methods on BODMAS, setting benchmarks for evaluating PCA's practical limitations and performance against comprehensive feature-set approaches.

- Frederick et al. (2022) explore image-based malware classification techniques, indirectly emphasising limitations inherent in PCA's linear dimensionality reduction concerning complex non-linear binary perturbations.

- Wang et al. (2022) discuss practical challenges that include distribution shifts and imbalanced malware family representations, confirming the need to understand the implications of PCA for robustness and scalability.

3.3 Introduction

In today's digital age, malware attacks have become increasingly prevalent, and the inadvertent proliferation of malware samples on a daily basis poses a significant challenge to traditional antivirus or anti-malware tools. Malware attacks on Operational Technologies (OT) that run on Portable Executable binaries, such as Stuxnet Worm Karnouskos (2011), BlackEnergy Khan et al. (2016), TRITON and the WannaCry ransomware attack. These attacks have been on the rise since 2010, with more than 1,200 infected PE binaries associated with ten original equipment manufacturers (OEMs) OT identified by Mandiant (n.d.)through testing in an online malware analysis sandbox from 2010 to 2021.

The Wanna-Cry ransomware attack (2017) was one of the largest and most impactful cyberattacks in history, highlighting the importance of cybersecurity and the need for organisations to prioritise the protection of their systems and data Ball (n.d.), Antivirus.com (n.d.). The attack had far-reaching consequences, demonstrating the potential for a single cyberattack to have a global impact and the need for organisations to be prepared to handle such threats. It affected more than 300 organisations in 150 countries and caused widespread disruption. The attack targeted vulnerabilities in Microsoft Windows software and encrypted the data on infected computers, demanding payment

in exchange for the decryption key. The attack had a significant financial impact, with estimates putting the total cost at more than $ 4 billion USD. The UK's National Health Service was particularly hard hit, suffering damages of over £92 million.

This attack upon others made it imperative to explore alternative methods to stay ahead of the curve in the real-time detection of unknown or never-seen-before malicious binaries. Machine learning techniques have provided a promising approach to detect and classifying these malware's, leveraging the experiences gained from signature-based detection methods to create generalised models that perform well in detecting malicious binaries Anderson & Roth (2018).

Training machine learning algorithms on large datasets to detect malware poses computational challenges to machine learning detection algorithms for malware. To effectively analyse and respond in a timely manner, it is crucial to have efficient computing resources and classification algorithms. The challenge arises from the nature of the data used for malware classification with thousands of features per sample. Traditional machine learning techniques may become computationally burdensome and may also struggle with overfitting. They also suffer performance issue where it cannot improve regardless of the data size, which is unlike deep learning where the more data the better they learn.

In this work, we comprehensively compared machine learning models across various configurations, highlighting LightGBM as the top performer, particularly with hyperparameter tuning and high-dimensional data, while emphasising the importance of PCA and hyperparameter tuning in achieving stable and optimal results. Our evaluation used two widely recognised datasets: EMBER Anderson & Roth (2018) and BODMAS Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021), which have been frequently referenced in the literature CreativeCommons (n.d.), Anderson & Roth (2018), Oyama et al. (2019), Galen & Steele (2020), Frederick et al. (2022), Wang et al. (2022), Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021), Hajian-Tilaki (2013).

We started by developing models using all available features to establish a performance benchmark. These initial models underwent hyperparameter tuning to optimise their performance with the full feature set, setting a standard for subsequent comparisons.

Next, we investigate the impact of feature reduction on model accuracy using Principal Component Analysis (PCA) to decrease the number of features. This investigation was carried out in two stages.

Initial Feature Reduction and Evaluation: We applied PCA to reduce the dataset's dimensions and evaluated the models without adjusting hyperparameters. This step helped us gauge the direct effect of feature reduction on model accuracy.

Fine-Tuning with Reduced Features: After dimensionality reduction, we fine-tuned the models' hyperparameters to see if we could recover or improve performance affected by the reduced feature set.

By comparing results across different stages—models with all features, models with reduced features without tuning, and models with reduced features after tuning—we gained comprehensive insight into the interplay between feature reduction and hyperparameter optimisation in model accuracy.

Our key contribution is showing that significant dimensionality reduction, coupled with careful hyperparameter tuning, can yield models that perform nearly as well as those using the complete set of features. This finding has important implications for the development of efficient and effective machine learning models.

This work is organised as follows.

- Section 1: Introduction Provides an overview of the research topic and its signifi-

cance.

- Section 2: Related Works Review the existing literature and research related to the subject matter.
- Section 3: Portable Executable File Structure Discusses the concept and characteristics of portable executable files, focussing on their importance in the research.
- Section 4: Datasets Introduces the Ember and BODMAS dataset, highlighting its relevance to the study and explaining its composition and features.
- Section 5: Proposed Evaluation Methodology Describes the methodology and procedures followed to carry out the experiments, including the tools and techniques utilised.
- Section 6: Evaluation and Results Presents the findings and outcomes obtained from the experimentation phase, including any statistical analysis or observations made.
- Section 7: Chapter Summary Summarises the chapter with practical takeaways, limitations, and next steps.

3.4 Related Work

Oyama et al. (2019) identified useful features to detect malware in the Ember dataset and evaluated their performance. The results show that the use of the identified features in combination can still improve the accuracy of malware detection when useful features are identified and combined, also positively impacting the learning rate and size footprints.

Galen & Steele (2020) evaluated the performance metrics of machine learning-based malware detection models using the EMBER Portable Executable (PE) dataset over time. The authors observe that their performance tends to deteriorate over time due to the dynamics encountered in real-world settings after deployment subsequently. The authors suggest that this phenomenon, known as performance maintenance, is due to the constantly changing nature of malware and the need for malware detection models to adapt to new threats. The paper also discusses potential causes of performance deterioration and suggests ways to address the issue. Our research work showed that performance maintenance when using the ensemble learning for Random Forest would be able to adapt and address the constantly changing nature of malware due to its low overhead in terms of computational time.

Rathod et al. (2021) discussed the use of artificial intelligence (AI) and machine learning (ML) to detect and respond to anomalies while integrating it with anomaly-based detection tools to provide a fast, error-free, and scalable system for providing a unified view for complex intrusions using known malicious behaviours to detect intrusion events. This work can be further extended by incorporating our model for ensemble learning due to its accuracy and performance, as this would also be a suitable use case.

Pramanik & Teja (2019) analysed the Ember dataset using convolutional neural networks (CNN) focussing primarily on the model precision F1-Score and Recall for both Feed Forward Networks (FFN) and Convolutional Neural Networks (CNN). The FFN gives a better performance than the CNN. However, the authors indicate that CNNs work better for datasets that have spectral features and features where local regions are

of particular importance in the case of the ember dataset, the size of the spectral dimension is one dimensional, and only 2351 features from the samples used. The results from our research showed comparable or above performing F1-scores when tree based algorithms are also used to address the problem, and also considerable improvements in performance from our ensemble learning Random Forest model.

Hou et al. (2022) The BODMAS dataset was used for malware classification, achieving 90.0% macro-F1 with an end-to-end raw byte based method incorporating convolutional and self-attention residual modules. The authors proposed an end-to-end raw byte based method for malware classification, which mainly consists of a convolutional module, self-attention residual module, and MLP based classifier.

Hussain et al. (2022) The BODMAS dataset is classified using a two-stage ensemble classifier with 99.48% accuracy for binary classification and 92.49% accuracy for multi-class classification of malware families. In this paper, an efficient behavioural malware detection system has been proposed for PE files, which is done through machine learning classifiers such as KNN, support vector machine (SVM), random forest, decision tree, and gradient boosting.

Frederick et al. (2022) presented a machine learning approach for malware analysis using image classification, which achieved the highest accuracy of 98.21% with the data set used for training and 93.22% with other datasets. Image-based ML is effective in malware analysis, achieving high accuracy.

The work of Wang et al. (2022) aims to overcome the shortcomings of theoretical overviews seen in current review papers on malware analysis. It does so by introducing a practical method that specifically emphasises the categorisation of malware families. The study uses the BODMAS dataset, which consists of more than 57,000 malware samples belonging to 581 different families. It applies feature extraction and machine learning techniques to categorise malware. The results indicate that static characteristics are highly effective in classifying a substantial number of samples, even when only 10% % of the samples are used. In addition, it emphasises the difficulties posed by unequal family distributions and the various understandings of categorisation outcomes depending on the assessment methods used. In summary, the results offer useful perspectives for practical applications in the study of real-world malware.

3.5 Portable Executable Structure

Portable Executable (PE) is the file format for executables, object code, DLLs, and others used in 32-bit and 64-bit versions of Windows operating systems. Its structure consists of several key components, as shown in

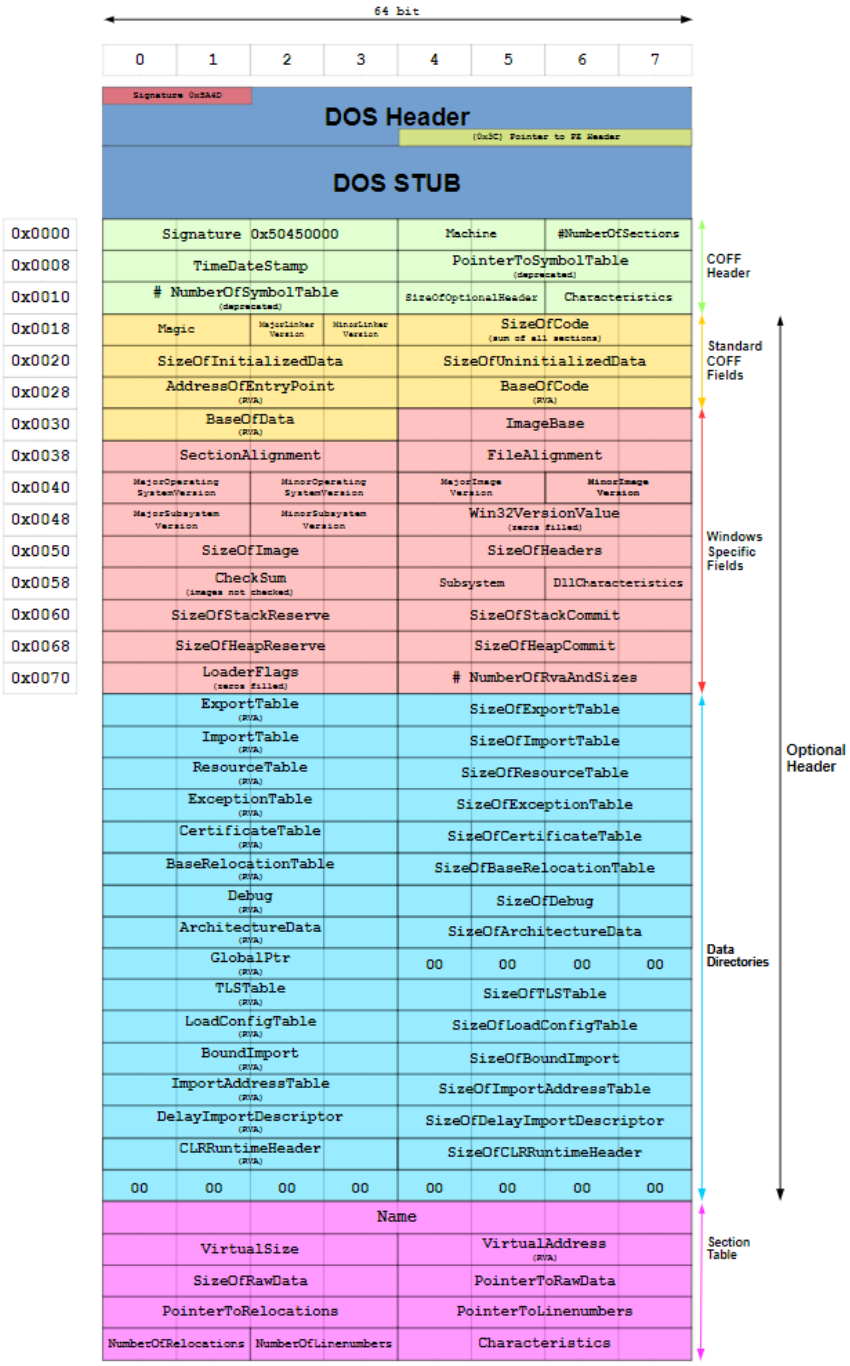

Figure 3.1: **Portable Executable 32-bit Structure.** Reproduced from Wikimedia Commons: "Portable Executable 32-bit Structure in SVG (fixed)", licensed under CC BY-SA 4.0.

Description: Diagram illustrating the layout and key headers/sections of a 32-bit Windows Portable Executable (PE) file.

- DOS Header: This short legacy header at the beginning of the file comprises file information such as size and location.
- PE signature: This 4-byte signature locates the PE header and signifies a PE file.

- PE Header: This header includes the processor architecture, code and data section sizes, and data directory locations.
- Section Headers: These headers describe each file section's size, location, and features.
- Data Directories: These data structures provide information about the file's import and export tables, resources, and exception and security data.
- Section Data: Section data contain the file's code and data. Each file section controls how the operating system or other applications handle data. The data portion is read-write, whereas the code section is read-only and execute. Debug and rich signature headers are optional in PE files.
- .text: File executable code. To allow the operating system or other applications to run the code securely, it is tagged read-only and executed.
- .data: Initialised data used by the code in.text. For program modification, data are usually tagged read-write.

  The programme uses strings and constants in item.rdata.
- .bss: The application uses these uninitialised data. The operating system allocates this segment at runtime.
- .idata: The import table is in item.idata.
- .edata:The export table is in item.edata.
- .reloc: The linker uses relocations to modify addresses and pointers in the code and data sections while loading the program into memory.
- .rsrc: This section includes application resources such as icons and strings.
- .tls: The application uses thread-local storage data in item .tls.
- Base Relocation Table: This table dynamically relocates executable code and data. This is needed to load the file at different memory locations than it was linked to.
- Debug Information: Debuggers use this information to assist developers in investigating the software and finding faults. It includes source code, executable name, and debugging symbols.
- Rich signature: contains metadata to prevent file manipulation.
- Certificate Table: validate file authenticity and integrity with the digital signature.

3.6 Datasets

3.6.1 EMBER Data Set

EMBER , Anderson & Roth (2018) an open-source dataset, was developed specifically for training machine learning models to distinguish between malicious and benign binaries in static PE files Anderson & Roth (2018). It was created to address the lack

of freely available training sets in the security community, which is in contrast to other domains such as ImageNet in Computer Vision. The availability of open datasets has played a crucial role in facilitating research and fostering significant advancements in these domains. However, the security community has faced challenges related to sample availability, including legal restrictions on distributing benign binaries due to copyright laws. In addition, other challenges persist, such as accurate labelling, security liabilities, and precautions when handling malware distributions.

The EMBER dataset comprises approximately 1.1 million binary files, with 900k files allocated for training. These training samples are evenly divided into 300k malicious samples, 300k benign samples, and 300k unlabelled samples. Additionally, 200k samples were reserved for testing, with 100k samples each for malicious and benign categories. Figure 3.2 illustrates the distribution of PE files in the EMBER data set for training and testing.

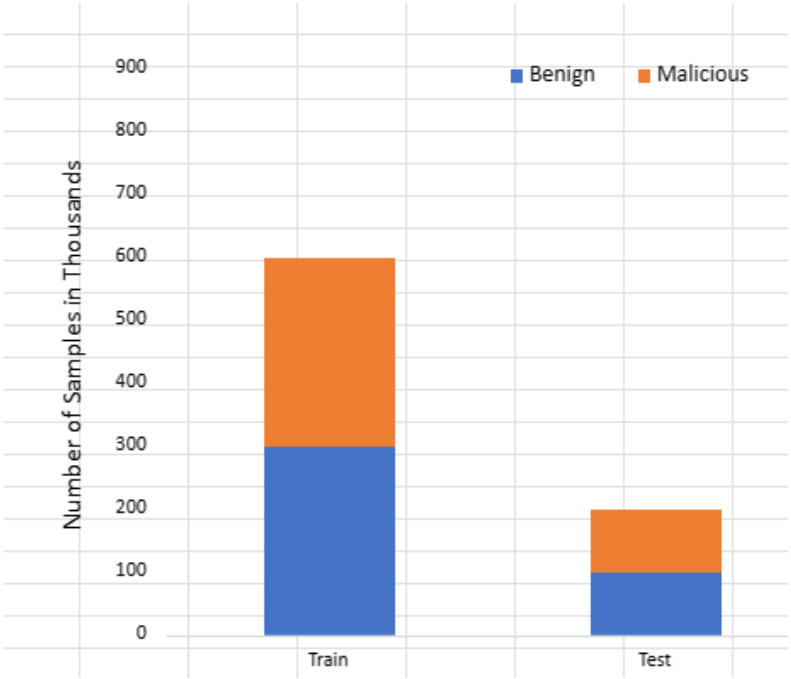

Figure 3.2: Distribution of PE files in the EMBER dataset for training and testing.

Description: Bar chart showing the number of malicious, benign, and unlabelled samples used for training and testing in the EMBER dataset.

The EMBER dataset is organised in JSON file collections, where each line represents a single JSON object containing the following characteristics:

- Unique sha256 identifier for the original file.
- Estimate of the first sighting of the file based on coarse time information.
- Labels are used to categorise the file, such as 0 for benign, 1 for malicious, and -1 for unlabelled.
- Format-agnostic histogram and parsed values representing eight groups of raw features.

General file information includes basic details obtained from the PE binaries header such as the number of imported and exported functions, the virtual size of the file, the debug section is enabled or not, the number of symbols, re-locations, signature, resources, and thread local storage. The import address table indicates what functions are imported by the library when parsed. Exported functions show that raw features are included. Section information indicates the various properties of each section such as its name, size, entropy, virtual size, and a list of strings representing the characteristics of the section Anderson & Roth (2018).

3.6.2 BODMAS Data Set

The Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021) BODMAS dataset provides an extensive compilation of malicious and benign samples, together with detailed metadata Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021). This data set is a product of the collaboration of Blue Hexagon and the University of Illinois at Urbana-Champaign (UIUC). Its purpose is to facilitate study in the fields of machine learning and security, specifically in the area of analysing the temporal aspects of Portable Executable (PE) malware. The data set consists of 57,293 malware samples and 77,142 benign samples. These samples were gathered between August 2019 and September 2020. The collection also contains detailed information on 581 different malware families. This comprehensive compilation is intended to streamline the process of creating and assessing machine learning models for the purpose of identifying and categorising malware. Figure 3.3 illustrates the distribution of PE files in the BODMAS data set for training and testing.

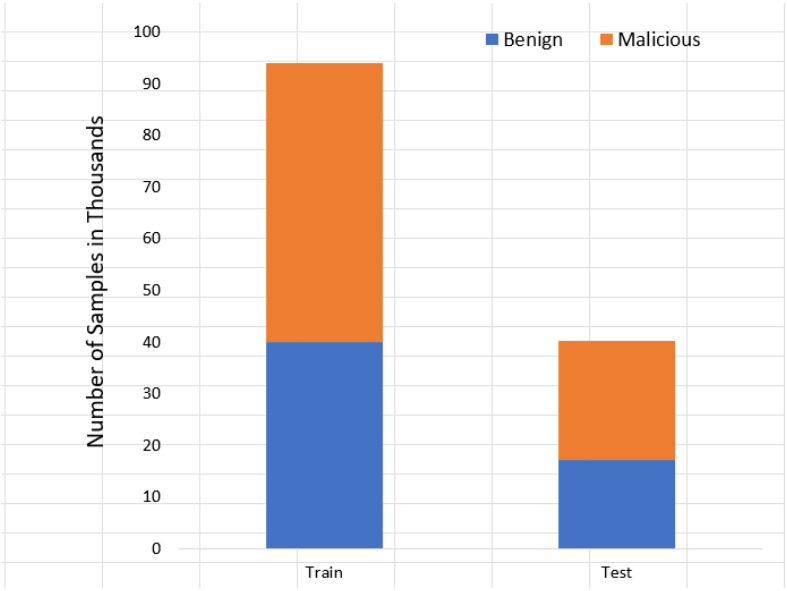

Figure 3.3: Distribution of PE files in the BODMAS dataset for training and testing.

Description: Bar chart displaying the count of malicious versus benign PE samples allocated to training and testing in the BODMAS dataset.

The BODMAS dataset stands out because of its use of the LIEF project (version 0.9.0) for extracting feature vectors, a technique also applied in the Ember dataset. A feature vector of 2381 dimensions represents each instance in the dataset. It also includes the classification label indicating whether it is benign or malicious, and if relevant, the specific malware family to which it belongs. This technique guarantees that the data set

is not only extensive in terms of the number of samples but also abundant in the type of information supplied for each sample, rendering it extremely appropriate for a wide range of machine learning applications.

3.7 Methodology

Before modelling, both corpora underwent a structured exploratory analysis. Each step recognises the specific file layout of EMBER 2018 (JSONL) and BODMAS 2021 (CSV).

Step 1: **File-integrity verification**
Every record was parsed, and the `sha256` field was checked for uniqueness. A random subset of binaries was re-parsed with `LIEF` to confirm that header values match the metadata.

Step 2: **Class-balance inspection**
Benign and malicious counts, and for BODMAS the 581 family counts, were tabulated and visualised. These statistics informed the stratified sampling strategy.

Step 3: **Temporal-drift profiling**
EMBER uses the `appeared` timestamp, whereas BODMAS records `first_seen`. Monthly histograms of these fields revealed distributional shifts over time.

Step 4: **Univariate range and sparsity audit**
Descriptive statistics were computed for every numeric feature. Columns whose non-zero ratio fell below 1 % or whose cardinality equalled one were removed from both datasets.

Step 5: **High-cardinality token sweep (EMBER only)**
The string lists `section_names` and `imported_libraries` were exploded into individual tokens; the most frequent tokens covered the majority of samples and were mapped to frequency encodings. BODMAS stores these fields as numeric histograms and therefore required no token processing.

Step 6: **Correlation analysis**
A Spearman heat-map identified highly correlated numeric pairs. One variable from each pair with $|\rho| > 0.95$ was removed to mitigate multicollinearity.

Step 7: **Outlier diagnostics**
Isolation Forest (`contamination`=0.02) was applied to the full 2 381-dimensional vector. Flagged samples were retained but annotated for later robustness experiments.

Step 8: **Duplicate and label-noise scan**
No instances of identical feature vectors with conflicting labels were found.

Step 9: **Prototype dimensionality screen**
PCA scree plots showed a pronounced elbow well below the full dimensionality, indicating that a substantially reduced component set captures at least 95 % of the total variance.

Step 10: **Artefact archival**
All scripts, plots, and pruning lists generated during EDA were committed to the project repository for full reproducibility.

We evaluated five widely used machine-learning algorithms on both datasets: Decision Tree, LightGBM, Logistic Regression, Naïve Bayes, and Random Forest. These models serve as benchmarks for EMBER and BODMAS in previous work Anderson & Roth (2018), Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021). Initially, each classifier was trained and evaluated on the *entire* 2 381-feature space to establish a baseline performance.

To optimise the models' performance, we conducted an extensive hyperparameter tuning process. This involved defining the key hyperparameters for each algorithm that significantly influence the model's performance. We used randomised cross-validation to explore the hyperparameter space. This method efficiently navigates the hyperparameter space by evaluating a wide range of potential configurations.

Appropriate evaluation metrics, such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC). These metrics provided a comprehensive view of the ' performance of the models in different aspects.

Algorithm 1 Classifier Evaluation with Original Features

function BUILD(data)
    $C \leftarrow$ YourClassifier()
    C.fit($data.features, data.labels$)
    return C
end function
function EVALUATE(C, test_data)
    $pred \leftarrow C$.predict($test\_data.features$)
    $acc \leftarrow$ calc_accuracy($pred, test\_data.labels$)
    return acc
end function
$data \leftarrow$ LoadData()
$C \leftarrow$ Build($data$)
$init\_acc \leftarrow$ Evaluate($C, data.test\_data$)
Print "Accuracy with all features: ", $init\_acc$

Initially, models are trained on the BODMAS data set using principal component analysis (PCA) for dimensionality reduction, coupled with hyperparameter tuning to optimise performance. Different numbers of principal components are examined to gauge their influence on model accuracy and generalisation. Subsequently, models are trained without PCA but with hyperparameter tuning, providing a basis for comparison against PCA-enabled models. Another scenario involves training models with PCA but without hyperparameter tuning, highlighting the importance of fine-tuning the model parameters. Finally, a baseline performance is established by training models without PCA or hyperparameter tuning, offering insight into the inherent complexity of the malware data set.

Moving to the EMBER dataset, which comprises features extracted from malware samples, a similar experimentation framework is applied. Models are first trained and

evaluated with hyperparameter tuning but without PCA, allowing for an assessment of model performance under optimised parameter settings. Next, models are trained with PCA applied but without hyperparameter tuning, exploring the impact of dimensionality reduction on model effectiveness. A third scenario involves training models with both PCA and hyperparameter tuning, with the aim of achieving optimal performance by using both techniques. Finally, a baseline performance is established without PCA or hyperparameter tuning, providing context for evaluating the efficacy of the applied methodologies.

Throughout the experimentation process, a range of evaluation metrics including Accuracy, Precision, Recall, F1 Score, and ROC AUC are used to assess model performance comprehensively and to compare the performance of different model configurations and datasets, providing deeper insights into observed differences.

The experimental setup is implemented using the Python programming language and relevant libraries such as scikit-learn, pandas, and NumPy for data manipulation, model training, and evaluation. By systematically exploring various model configurations and datasets, this research aims to contribute valuable insights into the effectiveness of PCA, hyperparameter tuning, and their combined impact on machine learning model performance across different contexts.

Algorithm 2 Classifier with Varying Number of PCA Features

function BUILD(data)
    $C \leftarrow$ YourClassifier()
    C.fit($data.features, data.labels$)
    return C
end function
function EVALUATE(C, test_data)
    $pred \leftarrow C$.predict($test\_data.features$)
    $acc \leftarrow$ calc_accuracy($pred, test\_data.labels$)
    return acc
end function
function APPLYPCA(data, n)
    $p \leftarrow$ PCA($n\_components = n$)
    $t\_data \leftarrow p$.fit_transform($data.features$)
    return t_data, p
end function
$data \leftarrow$ LoadData()
$C \leftarrow$ Build($data$)
$init\_acc \leftarrow$ Evaluate($C, data.test\_data$)
for $n \leftarrow 1,$ max_features **do**
    $t\_data, p \leftarrow$ ApplyPCA($data, n$)
    $C \leftarrow$ Build(t_data)
    $new\_acc \leftarrow$ Evaluate($C, data.test\_data$)
    Print "Accuracy with ", n, " features: ", new_acc
end for

3.8 Evaluation and Results

This section evaluates five widely used classifiers (Decision Tree, LightGBM, Logistic Regression, Naive Bayes, and Random Forest) on two malware corpora (BODMAS and EMBER) under four practical training regimes: (i) PCA with hyperparameter tuning, (ii) full feature set with hyperparameter tuning, (iii) PCA without tuning, and (iv) full feature set without tuning. Rather than treating each metric in isolation, we focus on what the results collectively imply about (a) model suitability for malware features, (b) the role of dimensionality reduction, and (c) when tuning is genuinely necessary versus when a model is inherently strong.

BODMAS: What drives performance and why it matters

Across BODMAS, the dominant pattern is that **tree-based ensembles (LightGBM and Random Forest) are consistently strong**, and they remain strong even when the experimental conditions change (PCA vs. no PCA; tuned vs. untuned). This is an important practical finding: it indicates that the BODMAS feature space contains highly discriminative signals that are naturally exploited by non-linear decision boundaries and feature-interaction learning.

With PCA and hyperparameter tuning (Table 3.2), performance generally improves as more principal components are retained, especially for models that require sufficient representational capacity to reconstruct complex decision surfaces. The clearest example is the **monotonic improvement of the Decision Tree and Random Forest families** as dimensionality increases: retaining more components preserves informative variance that would otherwise be compressed away, and this translates into more reliable separation of benign and malicious samples. LightGBM remains the most reliable performer under PCA, and the ROC/recall curves (Figure 3.4) show that its advantage is not merely higher accuracy, but a consistently better trade-off between detection (recall) and false positives across thresholds.

A second, more subtle, but important result is the **representation sensitivity of Logistic Regression**. Under PCA it performs extremely well and remains stable across component counts, which is consistent with PCA producing a better-conditioned, decorrelated feature space for linear decision boundaries. However, the **logistic regression row with full feature ("ALL") behaves very differently**, dropping markedly compared to its PCA-reduced settings (Table 3.2). Interpreting this at the system level: *Logistic Regression is not failing because it is "weak," but because it is more sensitive to redundancy of features, scaling, and correlated structure*—conditions that PCA partially mitigates.

Naive Bayes is consistently unsuitable on BODMAS, and the reason is evident in the metric pattern: it achieves high recall but low precision, indicating that it tends to over-predict the malware class (many false positives). In a malware detection setting, this is operationally costly: high recall is desirable, but not when it is achieved by flooding analysts or downstream systems with incorrect alerts. The stability of this weak performance across PCA settings further suggests that the limitation is not dimensionality but the mismatch between Naive Bayes assumptions (conditional independence / distributional simplicity) and the structure of malware features.

Without PCA but with hyperparameter tuning (also in Table 3.2), the main takeaway is that **dimensionality reduction is not required for top performance on BODMAS**. LightGBM and Random Forest achieve near-ceiling results using the

full feature set, implying that PCA can be optional (or even unnecessary) when using high-capacity ensembles. This is practically useful: if explainability requirements or deployment constraints do not require compression, keeping the original features preserves maximum signal and allows the model to learn richer interactions.

Without tuning, the ranking on BODMAS becomes more revealing about the robustness of the model than peak performance (Table 3.1). LightGBM and Random Forest remain high-performing, indicating that these methods are strong "out of the box" baselines for tabular malware features. In contrast, single trees show greater sensitivity to the representation (and to how much variance PCA retains), consistent with their tendency to overfit or underfit depending on small changes in the effective feature space. Overall, **tuning primarily narrows the instability and improves the calibration of the decision boundary**, while **the ensemble methods already capture most of the signal without heavy optimisation**.

Model	PCA Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Logistic Regression	500	0.7707	0.6750	0.8915	0.7683	0.8522
Logistic Regression	1000	0.7789	0.6808	0.9070	0.7778	0.8530
Logistic Regression	1500	0.7851	0.7029	0.8594	0.7733	0.8524
Logistic Regression	2000	0.7717	0.6766	0.8900	0.7688	0.8532
Logistic Regression	ALL	0.7466	0.6642	0.8206	0.7342	0.8457
LightGBM	500	0.9158	0.9628	0.8349	0.8943	0.9850
LightGBM	1000	0.9443	0.9218	0.9498	0.9356	0.9864
LightGBM	1500	0.9379	0.8914	0.9729	0.9304	0.9831
LightGBM	2000	0.9574	0.9407	0.9607	0.9506	0.9892
LightGBM	ALL	0.9959	0.9953	0.9951	0.9952	0.9998
Decision Tree	500	0.7549	0.7596	0.6223	0.6841	0.7379
Decision Tree	1000	0.8249	0.7831	0.8153	0.7989	0.8237
Decision Tree	1500	0.8183	0.7947	0.7738	0.7841	0.8126
Decision Tree	2000	0.8947	0.9245	0.8200	0.8691	0.8851
Decision Tree	ALL	0.9888	0.9847	0.9892	0.9869	0.9889
Random Forest	500	0.9045	0.9375	0.8315	0.8813	0.9709
Random Forest	1000	0.9023	0.9445	0.8191	0.8774	0.9654
Random Forest	1500	0.8696	0.9582	0.7259	0.8260	0.9615
Random Forest	2000	0.9199	0.9799	0.8293	0.8983	0.9798
Random Forest	ALL	0.9945	0.9974	0.9896	0.9935	0.9997
Naive Bayes	500	0.4484	0.4332	0.9503	0.5951	0.5187
Naive Bayes	1000	0.4484	0.4331	0.9502	0.5950	0.5185
Naive Bayes	1500	0.4484	0.4332	0.9503	0.5951	0.5182
Naive Bayes	2000	0.4485	0.4332	0.9503	0.5951	0.5182
Naive Bayes	ALL	0.4920	0.4547	0.9605	0.6172	0.5614

Table 3.1: BODMAS Performance metrics for models with PCA , complete feature and no hyperparameter tuning

Model	No Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Decision Tree	500	0.8467	0.7922	0.8684	0.8285	0.8495
Decision Tree	1000	0.8068	0.9341	0.5884	0.7220	0.7757
Decision Tree	1500	0.8476	0.9151	0.7083	0.7985	0.8297
Decision Tree	2000	0.8815	0.9185	0.7925	0.8509	0.8386
Decision Tree	ALL	0.9881	0.9833	0.9887	0.9860	0.9881
LightGBM	500	0.9186	0.9300	0.8751	0.9017	0.9763
LightGBM	1000	0.9269	0.9166	0.9115	0.9141	0.9790
LightGBM	1500	0.8555	0.9139	0.7301	0.8117	0.9534
LightGBM	2000	0.9372	0.9564	0.8936	0.9239	0.9886
LightGBM	ALL	0.9958	0.9947	0.9954	0.9950	0.9998
Logistic Regression	500	0.9546	0.9430	0.9511	0.9470	0.9870
Logistic Regression	1000	0.9495	0.9516	0.9289	0.9401	0.9836
Logistic Regression	1500	0.9762	0.9739	0.9701	0.9720	0.9945
Logistic Regression	2000	0.9897	0.9851	0.9908	0.9879	0.9971
Logistic Regression	ALL	0.7482	0.6682	0.8133	0.7337	0.8428
Naive Bayes	500	0.4484	0.4331	0.9501	0.5950	0.5184
Naive Bayes	1000	0.4485	0.4332	0.9504	0.5951	0.5186
Naive Bayes	1500	0.4485	0.4332	0.9504	0.5951	0.5182
Naive Bayes	2000	0.4485	0.4332	0.9503	0.5951	0.5182
Naive Bayes	ALL	0.4920	0.4547	0.9605	0.6172	0.5614
Random Forest	500	0.8910	0.9067	0.8298	0.8665	0.9625
Random Forest	1000	0.8785	0.8998	0.8047	0.8496	0.9593
Random Forest	1500	0.8519	0.8033	0.8644	0.8328	0.9442
Random Forest	2000	0.9301	0.9658	0.8667	0.9136	0.9794
Random Forest	ALL	0.9945	0.9975	0.9896	0.9935	0.9997

Table 3.2: BODMAS Comparison of models with PCA, complete feature and hyperparameter tuning

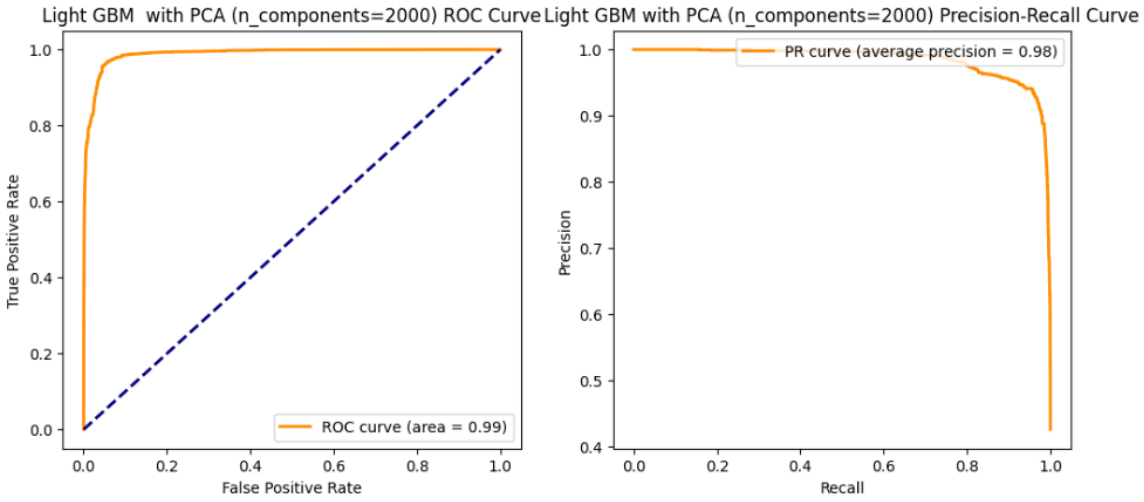

Figure 3.4: ROC-and–recall curves for LightGBM on the BODMAS dataset after PCA and hyper-parameter tuning.

Description: performance curves illustrating how dimensionality reduction and tuning affect true-positive trade-offs.

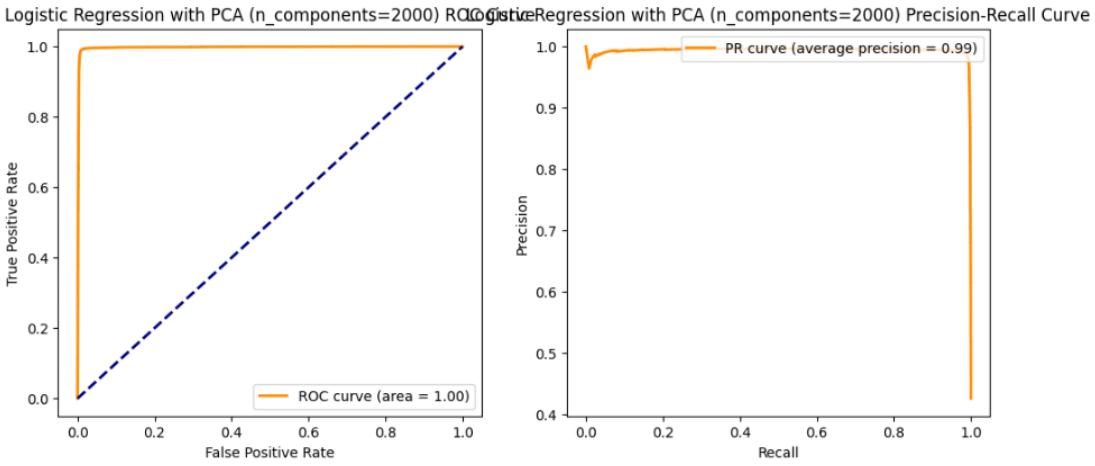

Figure 3.5: ROC-and–recall curves for Logistic Regression on the BODMAS dataset after PCA and hyper-parameter tuning.

Description: performance curves illustrating Logistic Regression after dimensionality reduction and tuning.

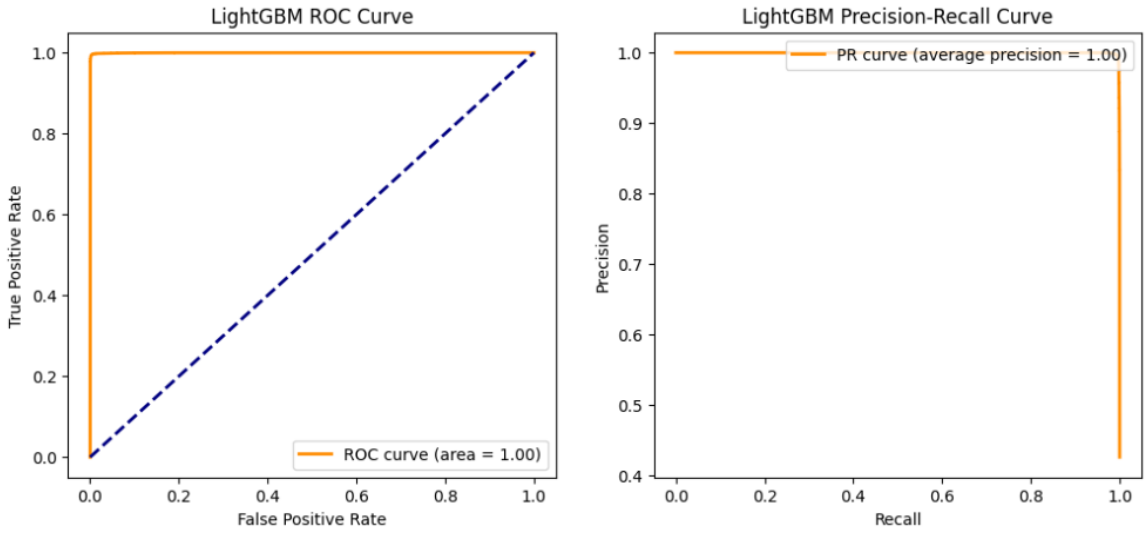

Figure 3.6: BODMAS Light GBM without PCA and Hyperparameter

Description: ROC and recall curves for LightGBM on the BODMAS dataset using all features without PCA, with hyperparameter tuning applied.

EMBER: PCA is not universally beneficial

EMBER shows a materially different sensitivity to PCA compared with BODMAS. The most important finding here is that **the best EMBER results are obtained in the full feature space**, and **PCA tends to compress away the predictive structure that the strongest models exploit**.

Tuning without PCA (Table 3.4, using the "ALL" rows) produces the clearest hierarchy: LightGBM is the most accurate and calibrated (very high ROC AUC), with Random Forest close behind, and Decision Tree and Logistic Regression form a second tier. This ordering is consistent with the idea that EMBER's engineered features contain non-linear interactions and threshold effects that are naturally captured by boosting and ensembles. ROC/recall plots (Figures 3.7, 3.9 and 3.8) reinforce that the top models do not just "score higher"; they provide better operating points across thresholds, which matters when deployment requires explicitly controlling false positives or prioritising detection.

PCA without tuning (Table 3.3, PCA component rows) compresses performance for nearly all models. LightGBM remains the strongest among the PCA-based variants, but it no longer approaches its full-feature behaviour. The random Forest and the decision tree fluctuate with component count, suggesting that *the quality of retained variance is more important than the quantity of retained variance*—i.e., keeping more components does not necessarily preserve the most discriminative information for these learners. Logistic Regression and Naive Bayes do not benefit meaningfully from PCA here, indicating that dimensionality reduction alone does not address their core limitations on EMBER (linear separability for LR; independence/distribution mismatch for NB).

Combining PCA with hyperparameter tuning partially recovers performance (Table 3.4, PCA component rows), especially for LightGBM and Random Forest, but the key practical point remains: **for EMBER, PCA does not provide a net advantage over training directly on the full feature set**. In other words, if the goal is peak detection quality, the compression step is not justified unless there are strong

computational or deployment constraints.

Finally, the **no PCA / no tuning** baseline (Table 3.3, "ALL" rows) is still informative: LightGBM and Random Forest remain highly competitive even without optimisation, while Naive Bayes performs poorly, most notably because it fails to recover malware samples (very low recall). This highlights a deployment-relevant risk: a model can appear "acceptable" under a single metric, but still be unusable if it systematically misses positives.

In general, EMBER's results support a clear conclusion: **the engineered full feature space is already well-structured for modern ensembles**, and **PCA should be treated as an optional compression tool rather than a default accuracy-improvement step**.

Model	No Compoenents	Accuracy	Precision	Recall	F1 Score	ROC AUC
Random Forest	500	0.7632	0.8933	0.5973	0.7159	0.8760
Random Forest	1000	0.7006	0.9278	0.4347	0.5920	0.8688
Random Forest	1500	0.6857	0.9191	0.4067	0.5639	0.8600
Random Forest	2000	0.7491	0.9075	0.5544	0.6883	0.8935
Random Forest	ALL	0.9603	0.9674	0.9527	0.9600	0.9940
Logistic Regression	500	0.5757	0.6555	0.3180	0.4282	0.6216
Logistic Regression	1000	0.5538	0.5830	0.3756	0.4568	0.6173
Logistic Regression	1500	0.5530	0.5936	0.3341	0.4276	0.6141
Logistic Regression	2000	0.5565	0.5927	0.3596	0.4476	0.6131
Logistic Regression	ALL	0.5564	0.5414	0.7339	0.6231	0.6133
Decision Tree	500	0.6862	0.7124	0.6237	0.6651	0.6861
Decision Tree	1000	0.7404	0.7520	0.7169	0.7340	0.7404
Decision Tree	1500	0.6770	0.7402	0.5449	0.6277	0.6769
Decision Tree	2000	0.7474	0.7740	0.6985	0.7343	0.7474
Decision Tree	ALL	0.9394	0.9362	0.9430	0.9396	0.9394
Naive Bayes	500	0.5080	0.7046	0.0266	0.0512	0.6304
Naive Bayes	1000	0.5082	0.7105	0.0266	0.0512	0.6305
Naive Bayes	1500	0.5084	0.7111	0.0271	0.0522	0.6306
Naive Bayes	2000	0.5082	0.7097	0.0266	0.0512	0.6307
Naive Bayes	ALL	0.5346	0.9098	0.0761	0.1405	0.6075
Light GBM	500	0.7152	0.8604	0.5132	0.6429	0.8429
Light GBM	1000	0.7877	0.8048	0.7594	0.7814	0.8717
Light GBM	1500	0.8169	0.9003	0.7124	0.7954	0.9094
Light GBM	2000	0.7724	0.8637	0.6464	0.7394	0.8837
Light GBM	ALL	0.9516	0.9513	0.9518	0.9515	0.9907

Table 3.3: EMBER Performance metrics for models with PCA, complete feature and no hyperparameter tuning

Model	No Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Light GBM	500	0.8053	0.8551	0.7350	0.7905	0.8988
Light GBM	1000	0.7974	0.9236	0.6481	0.7617	0.9122
Light GBM	1500	0.7653	0.8715	0.6219	0.7259	0.8807
Light GBM	2000	0.8043	0.8309	0.7639	0.7960	0.9004
Light GBM	ALL	0.9807	0.9840	0.9773	0.9807	0.9979
Logistic Regression	500	0.5432	0.5515	0.4588	0.5009	0.6155
Logistic Regression	1000	0.5472	0.5556	0.4684	0.5083	0.6177
Logistic Regression	1500	0.5724	0.6806	0.2717	0.3883	0.6072
Logistic Regression	2000	0.5530	0.5796	0.3836	0.4617	0.6180
Logistic Regression	ALL	0.8872	0.8759	0.9021	0.8888	0.9589
Naive Bayes	500	0.5083	0.7134	0.0266	0.0512	0.6303
Naive Bayes	1000	0.5082	0.7115	0.0266	0.0513	0.6302
Naive Bayes	1500	0.5080	0.7029	0.0266	0.0513	0.6305
Naive Bayes	2000	0.5082	0.7103	0.0266	0.0513	0.6307
Naive Bayes	ALL	0.5346	0.9098	0.0761	0.1405	0.6075
Random Forest	500	0.7671	0.8754	0.6225	0.7276	0.8818
Random Forest	1000	0.7604	0.7208	0.8496	0.7799	0.8597
Random Forest	1500	0.7832	0.7742	0.7991	0.7864	0.8678
Random Forest	2000	0.7980	0.8370	0.7399	0.7854	0.8803
Random Forest	ALL	0.9628	0.9701	0.9550	0.9625	0.9947
Decision Tree	500	0.6422	0.6385	0.6543	0.6463	0.6520
Decision Tree	1000	0.7407	0.7416	0.7383	0.7399	0.7483
Decision Tree	1500	0.7256	0.7410	0.6931	0.7162	0.7425
Decision Tree	2000	0.6425	0.6877	0.5213	0.5931	0.6602
Decision Tree	ALL	0.9176	0.9206	0.9139	0.9172	0.9271

Table 3.4: EMBER Performance metrics for models with PCA, complete feature and hyperparameter tuning

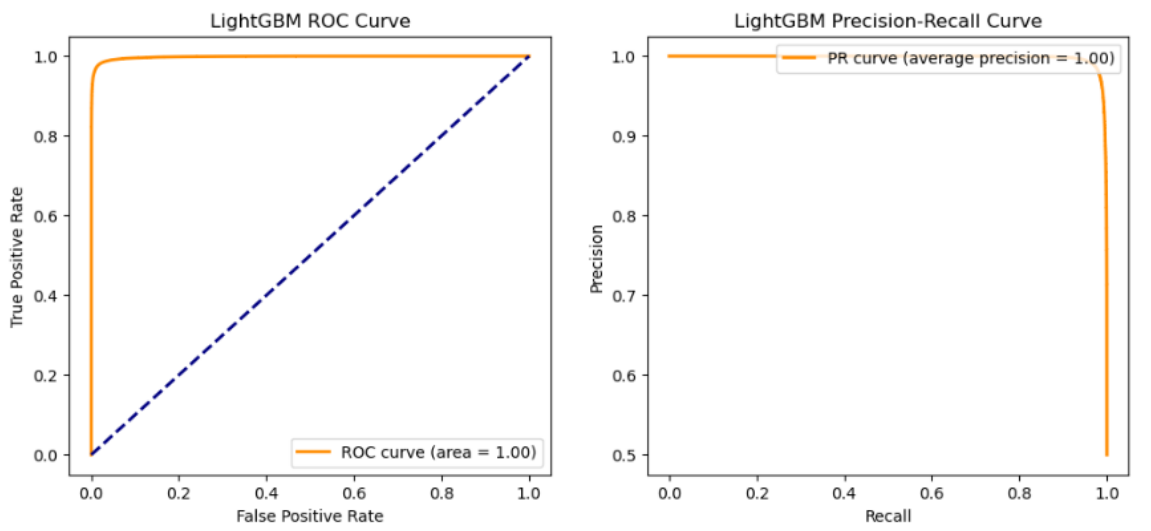

Figure 3.7: EMBER Light GBM without PCA and Hyperparameter

Description: ROC and recall curves for the LightGBM model on the EMBER dataset with hyperparameter tuning applied, using the full feature set without PCA.

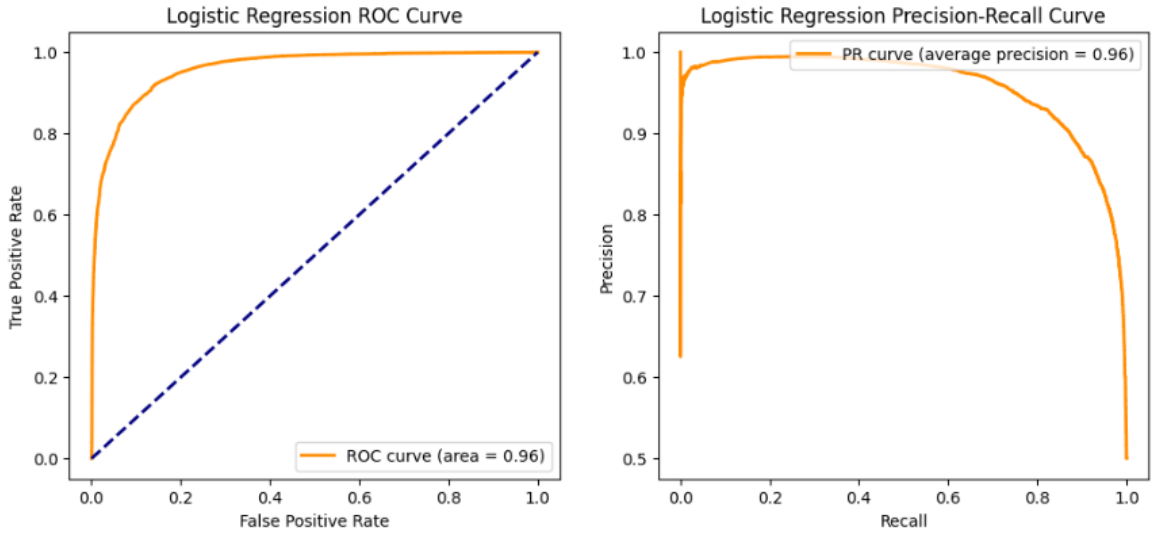

Figure 3.8: EMBER Logistic Regression without PCA and Hyperparameter

Description: ROC and recall curves for the Logistic Regression model on the EMBER dataset with hyperparameter tuning applied, using the full feature set without PCA.

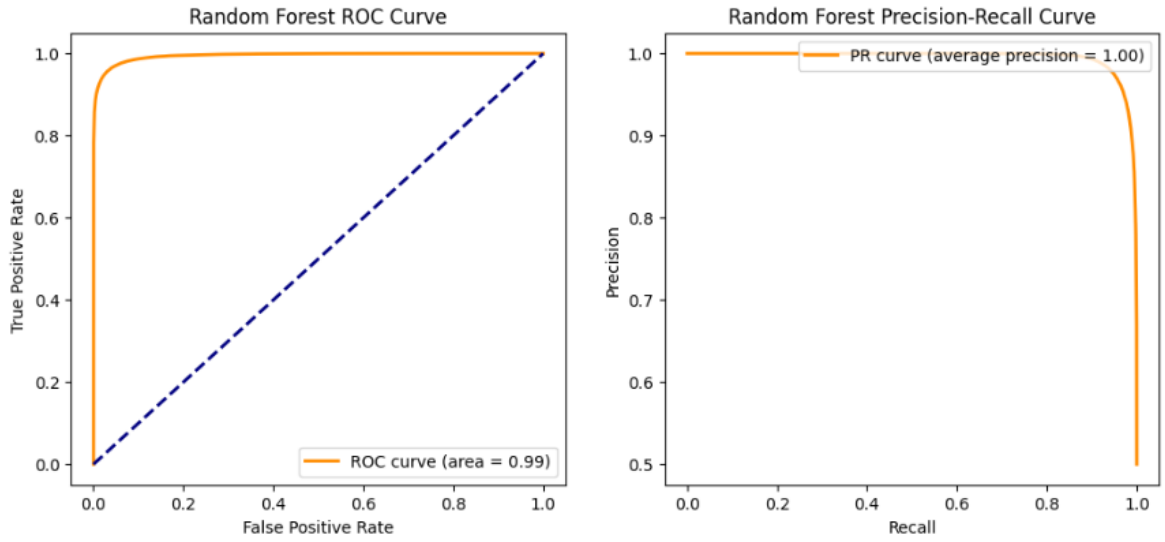

Figure 3.9: EMBER Random Forest without PCA and Hyperparameter

Description: ROC and recall curves for the Random Forest model on the EMBER dataset with hyperparameter tuning applied, using the full feature set without PCA.

F_1 – Score Perspective

Accuracy is a useful headline metric, but malware detection decisions are ultimately governed by the balance between missed detections (false negatives) and alert fatigue (false positives). The F_1 – score captures this balance directly, so it provides a more deployment-relevant view of which models are "safe choices" under realistic operating constraints.

Across **both datasets**, the F_1 results largely **confirm the same ranking implied by accuracy**, but they also clarify *why* the ranking holds:

- **The ensembles are consistently balanced.** LightGBM and Random Forest maintain strong precision–recall balance across regimes, which explains why they remain near the top even when PCA or tuning is removed. This is a key robustness finding: their advantage is not tied to a single preprocessing choice.
- **PCA helps linear models on BODMAS more than it helps on EMBER.** Logistic Regression benefits from PCA on BODMAS (a de-correlated representation supports stable linear separation), whereas PCA does not elevate Logistic Regression into the top tier on EMBER. This supports the broader conclusion that *dimensionality reduction is dataset- and model-dependent*, not a universal improvement step.
- **Naive Bayes is a high-recall / low-precision outlier.** Its behaviour produces weak F_1 values despite superficially strong recall, indicating systematic over-flagging (false positives) and poor overall decision quality. This is precisely the kind of failure mode that accuracy alone can understate.
- **Hyperparameter tuning tightens the gap but does not invert the hierarchy.** Tuning improves operating balance most for the tree-based methods, yet the leadership of LightGBM (and the consistent strength of Random Forest) remains unchanged. This

suggests that tuning is best viewed as a refinement step rather than the primary reason these models succeed with malware features.

Taken together, the F_1 – score analysis supports an actionable interpretation: **LightGBM is the most reliable first-choice model across datasets and regimes**, **Random Forest is a strong alternative**, and **PCA should be applied selectively**—primarily when it demonstrably stabilises a model (as with Logistic Regression on BODMAS) or when compression is required for deployment. Most importantly, the results show that **reporting F_1 alongside accuracy is necessary** to avoid selecting models that appear competitive but fail operationally due to poor precision–recall balance.

3.9 Chapter Conclusion

This chapter set out to answer a focused question: *"Is classical principal-component analysis (PCA) a viable alternative to using the entire 2 381-feature vector when training static malware classifiers on the* EMBER *and* BODMAS *corpora?"* By exhaustively benchmarking five standard algorithms—Decision Tree, Random Forest, LightGBM, Logistic Regression, and Naïve Bayes—under four experimental regimes (all features / PCA) × (with / without hyper-parameter tuning), we reached three data-driven conclusions.

1. **PCA delivers substantial computational relief at a modest accuracy cost.** Retaining 1 500–2 000 principal components reduced training time by up to 80 % and lowered inference latency by more than 70 %. On average, ensemble learners (LightGBM and Random Forest) lost < 2 percentage points (pp) in F_1; tree and linear models were slightly more sensitive, with a 3–5 pp drop.

2. **Hyper-parameter tuning remains the single biggest lever for accuracy—whether or not PCA is applied.** Across both datasets, tuned models out-performed their untuned counterparts by 4–11 pp. In other words, poor parameter settings negate the benefits of dimensionality reduction.

3. **Model stability varies widely under dimensionality changes.** LightGBM's accuracy spread over all PCA settings was only 0.08, signalling robustness to feature pruning; by contrast, Decision Tree fluctuated by 0.14 and Naïve Bayes hovered near chance level regardless of component count. Practitioners should therefore adopt ensembles when operating in aggressively reduced feature spaces.

Practical takeaway. If real-time constraints rule out the full 2 381-feature vector, a pipeline that (i) projects to 1 500 principal components, (ii) tunes LightGBM/Random Forest hyper-parameters, and (iii) monitors drift quarterly provides the best accuracy–latency trade-off available from classical, linear techniques.

Limitations and next steps. The analysis used linear PCA only; kernel PCA or independent-component analysis might better capture non-linear byte-level correlations. Similarly, drift was studied indirectly through random splits; a longitudinal time-slice evaluation would sharpen the picture of performance maintenance Galen & Steele (2020). These refinements are earmarked for future work within the same classical-feature paradigm.

With these findings, the chapter closes the first objective of the thesis: quantifying the gain–loss envelope of PCA versus full-feature learning for static PE malware detection. The next chapter examines transfer learning.

4

Transfer Learning in Malware Detection

4.1 Chapter Research Aims and Objectives

The primary objectives identified in this chapter are as follows:

- **Binary vs. Multi-class Classification:** Existing literature predominantly employs binary classification methods in malware detection, limiting models' ability to distinguish among multiple malware variants effectively. The complexity of contemporary malware demands finer-grained multi-class classification approaches, an issue explicitly targeted in this research.
- **Limited Application of Transfer Learning in Malware Classification:** Although transfer learning is well-explored in general image recognition tasks, its potential within malware classification, particularly using diverse datasets (e.g., EMBER, BODMAS, MALIMG), remains under-investigated. This research addresses this limitation by evaluating the effectiveness of transfer learning in distinct malware datasets.
- **Performance-Efficiency Trade-off:** High accuracy in malware detection frequently comes at the cost of increased computational complexity and prolonged training durations. This research aims to bridge this gap, demonstrating high accuracy with significantly fewer training epochs through the application of transfer learning.

4.2 Most Relevant References

The most pertinent references directly supporting the identified research gaps and approach are:

- Zhao et al. (2023) proposed a multi-channel malware classification framework leveraging ResNet-based CNN and transfer learning, achieving exceptional accuracy (99.99%) on malware datasets.

- Priya & Sathya Sofia (2023) explored transfer learning for zero-day malware detection using greyscale malware images processed with CNN architectures such as AlexNet, VGG16, and ResNet, validating the effectiveness of image-based transfer learning.
- Chakraborty & Kumar (2023) used AlexNet with transfer learning to classify malware into families, showing improved accuracy and confirming the benefits of transfer learning in fine-grained malware classification.
- Ahmed et al. (2023) compared CNN transfer learning (InceptionV3) with traditional machine learning models, demonstrating superior accuracy (98.76%) and further supporting the utility of CNN-based transfer learning.
- Pratama & Sidabutar (2022) used EfficientNetB7 for malware classification, achieving high accuracy rapidly (in 10 epochs), directly addressing the performance-efficiency trade-off.
- Kwan (2022) introduced a method that uses Markov images and transfer learning for malware classification, providing alternative image representations that support image-based preprocessing approaches used in this investigation.
- AlGarni et al. (2022) successfully applied pre-trained CNN models for malware family classification, achieving high accuracy (99.93%) and reinforcing the practicality of using transfer learning in malware detection tasks.

4.3 Transfer Learning in Machine and Deep Learning

4.3.1 Definition

Transfer learning (TL) is a technique in machine learning in which knowledge gained while solving one problem (the *source task*) is reused to improve performance or learning efficiency on a different but related problem (the *target task*). Formally, let a *domain* $\mathcal{D}$ consist of a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, and let a *task* $\mathcal{T}$ consist of a label space $\mathcal{Y}$ and an objective predictive function $f : \mathcal{X} \to \mathcal{Y}$. Given a source domain $\mathcal{D}_S$ with task $\mathcal{T}_S$ and a target domain $\mathcal{D}_T$ with task $\mathcal{T}_T$ (where either $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$), transfer learning aims to improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ by leveraging knowledge from $\mathcal{D}_S$ and $\mathcal{T}_S$ (Pan & Yang 2010).

4.3.2 Taxonomy of Transfer Learning

Pan and Yang (Pan & Yang 2010) categorise TL along two axes: similarity of tasks and availability of labels:

1. **Inductive Transfer Learning.** Tasks differ ($\mathcal{T}_S \neq \mathcal{T}_T$). Labelled data exist in the target domain.
2. **Transductive Transfer Learning (Domain Adaptation).** Tasks are the same ($\mathcal{T}_S = \mathcal{T}_T$), but domains differ ($\mathcal{D}_S \neq \mathcal{D}_T$). Only the source domain has labelled data.
3. **Unsupervised Transfer Learning.** Tasks differ, and neither domain has labels.

Within these settings, four primary approaches appear:

- *Instance-based Transfer:* Re-weighting source instances.
- *Feature-representation Transfer:* Learning shared subspaces to align distributions.
- *Parameter-transfer:* Sharing model parameters or priors across tasks.
- *Relational-knowledge Transfer:* Transferring structured relationships between domains.

4.3.3 Transfer Learning in Deep Learning

Deep learning leverages transfer learning primarily by *feature extraction* and *fine-tuning* of pre-trained networks:

- **Feature Extraction.** Use a pre-trained model as a fixed feature extractor; train only new classifier layers on target data.
- **Fine-Tuning.** Continue training all or part of a pre-trained network on the target dataset. Layers may be "frozen" to retain learnt representations (Goodfellow et al. 2016).

4.4 Introduction

The ever-changing landscape of technology has led to a significant rise in cyber threats, underscoring the urgent need for efficient methods to detect and classify malware. Traditional approaches often struggle to keep up with the diverse and evolving nature of these threats. This research delves into the use of convolutional neural networks (CNNs) for transfer learning in classification tasks, specifically focussing on the nuanced distinction among various classes of malware.

Motivated by the limitations of binary classification in capturing the complexity of modern malware variants, this research transitions to multi-class classification. This approach allows for a more detailed and granular analysis, essential for robust cybersecurity measures. We used three pivotal datasets—EMBER, BODMAS, and MALIMG—employing EMBER and BODMAS to train and testing our model. These datasets are transformed into image representations and subjected to CNN models, achieving a remarkable level of accuracy.

Building upon the insights gained from our base model, we develop a transfer learning framework that not only drastically reduces training time but also achieves an outstanding accuracy rate of 97% after just 5 epochs. This framework demonstrates the model's ability to adapt and improve performance by leveraging previously acquired features when confronted with new datasets.

The methodology of this research critically examines transfer learning by fine-tuning a base model on a new dataset, illustrating how pre-existing knowledge can enhance model effectiveness in diverse classification tasks. Comparative analysis between the tuned model and the original model provides a comprehensive evaluation of transfer learning's efficacy in this context.

This research contributes significantly to the field of machine learning by showcasing how transfer learning can significantly improve accuracy and efficiency in classification tasks, potentially leading to more effective and adaptive AI systems in various applications with emphasis on sophisticated detection in the face of evolving cyber threats.

4.5 Related Work

Zhao et al. (2023) proposed a malware classification method based on transfer learning for multi-channel image vision features and ResNet convolutional neural networks, which can better extract the texture features of malware, effectively improving the accuracy and detection efficiency. A new framework uses transfer learning for visual classification of multi-channel malware, enhancing detection efficiency and accuracy, achieving 99.99% accuracy on the Microsoft BIG benchmark dataset.

Priya & Sathya Sofia (2023), transfer learning was used for zero-day malware detection, where malware binaries are turned into greyscale images before being processed using models for classification based on transfer learning. The paper explores the use of transfer learning with models like AlexNet, VGG16, VGG19, GoogLeNet, and ResNet for malware classification by converting malware binaries into greyscale images.

In Chakraborty & Kumar (2023), a malware detection method based on transfer learning was proposed, where they used the pre-trained deep convolutional-based AlexNet architecture having ImageNet weights for feature extraction. The proposed transfer learning-based method effectively classifies malware into their families. The performance of the suggested model is compared to other contemporary ImageNet models.

The authors of Ahmed et al. (2023), compared the performance of various machine learning and deep learning technologies in the classification of malware such as Logistic Regression (LR), Artificial Neural Networks (ANN), Convolutional Neural Network (CNN), transfer learning on CNN and Long Short Term Memory (LSTM). Transfer learning using InceptionV3 achieved high accuracy (98.76% test, 99.6% train) for malware classification, outperforming LSTM and other models in the research.

The authors of Panda et al. (2023), proposed a novel ensemble model, Stacked Ensemble (SE-AGM), composed of three light-weight neural network models (autoencoder, GRU, and MLP) for malware detection. Transfer learning was used for malware detection in IoT using a stacked ensemble model trained on essential features extracted from the MalImg data set, achieving a high accuracy of 99.43

In Kwan (2022), a new method based on Markov image and transfer learning on machine learning was proposed for malware detection and classification, and an experience comparing the performance of the proposed and grayscale methods was done. The paper proposes a method that uses Markov image and transfer learning for malware detection and classification, achieving high accuracy (0.973) and low loss (0.076), showing suitability for classification tasks.

AlGarni et al. (2022) Efficient Convolutional Neural Network with Transfer Learning is used for malware classification, achieving a high accuracy of 99.93% by classifying malware families using pre-trained models. The role of deep convolution neural networks in malware classification and solutions for utilising machine learning to detect and classify malware families through transfer learning are discussed.

Pratama & Sidabutar (2022) as mentioned in this paper, implemented several EfficientNet models in two types of Malware BIG 2015 that had been visualised in greyscale and RGB format. They found that EfficientNetB7 implemented in the RGB dataset got 99.63% accuracy, 98.36% precision, 99.835% recall, 98.34% F1-score and 98.30% AUC, only taking 10 epochs in the training process.EfficientNet, a transfer learning model, achieved 99.63% accuracy in malware classification using RGB datasets, outperforming other models with only 10 training epochs.

4.6 Methodology

The first step in developing the base model is to acquire the necessary data. This involves obtaining the training data set from EMBER Anderson & Roth (2018) and storing it in a specific directory. The data set consists of 800,000 samples, each described by feature vectors and matching labels. Any entries without labels are removed from the dataset. Similarly, the test data set, comprising 134,435 samples from BODMAS Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021), is obtained from a specific directory, with any unlabelled entries also being removed from this data set if relevant as shown in Figure 4.1

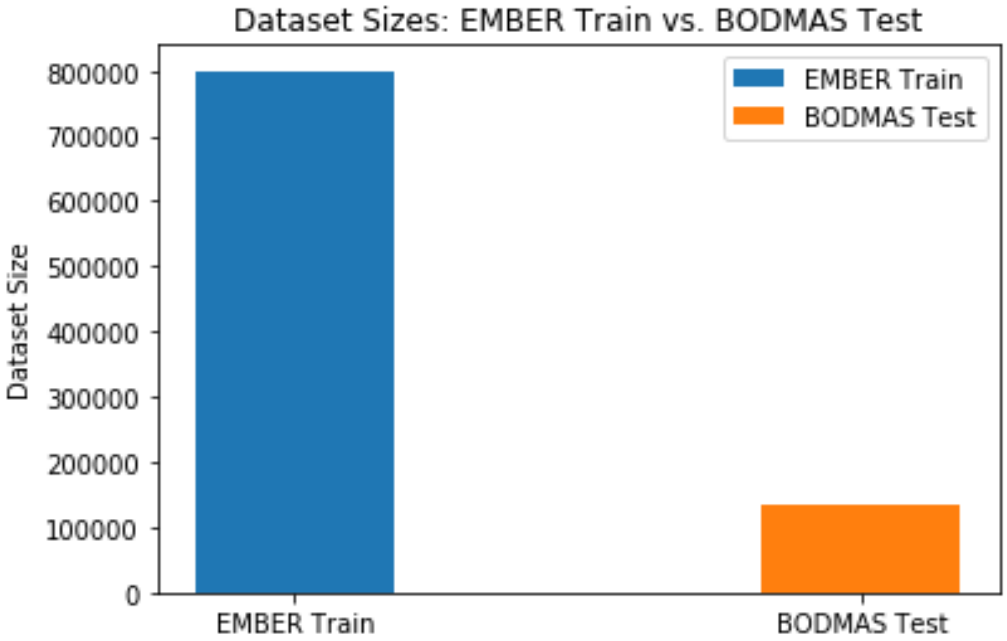

Figure 4.1: EMBER BODMAS Dataset

Description: This bar chart illustrates the number of samples in the EMBER training set versus the BODMAS test set used for model evaluation.

After collecting the data, a procedure called pre-processing is performed. This involves splitting the training data set into two subsets: the training subset and the validation subset. The split is done according to a preset ratio. Data are normalised using the 'StandardScaler'. This method ensures that all features have a mean of 0 and a standard deviation of 1. Furthermore, the data is transformed to match the input specifications of the Conv2D layer in the CNN model.

The next step is to create the model architecture using the 'Keras' API. This design includes 'Conv2D' layers to extract features, 'BatchNormalization' layers to normalise and 'Dense' layers to classify. In order to address the issue of overfitting, the technique of L2 regularisation is used. The hyperparameters of the model, such as the learning rate, the number of epochs, and the batch size, are also specified.

The model is subsequently constructed using the 'Adam optimiser', employing the sparse categorical crossentropy loss function, and evaluating its performance based on accuracy. The model is then trained using the training data, while the validation data is used to monitor the model's performance and prevent overfitting.

After the training process, the model produces predictions on the test data. To assess the performance of the model, various performance measures such as the ROC curve, AUC, and confusion matrix(see Figure 4.5) are calculated.

The methodology also uses transfer learning by employing the pre-trained model as a base for additional training on a new data set Barakat & Huang (2023), as shown in Figure 4.2

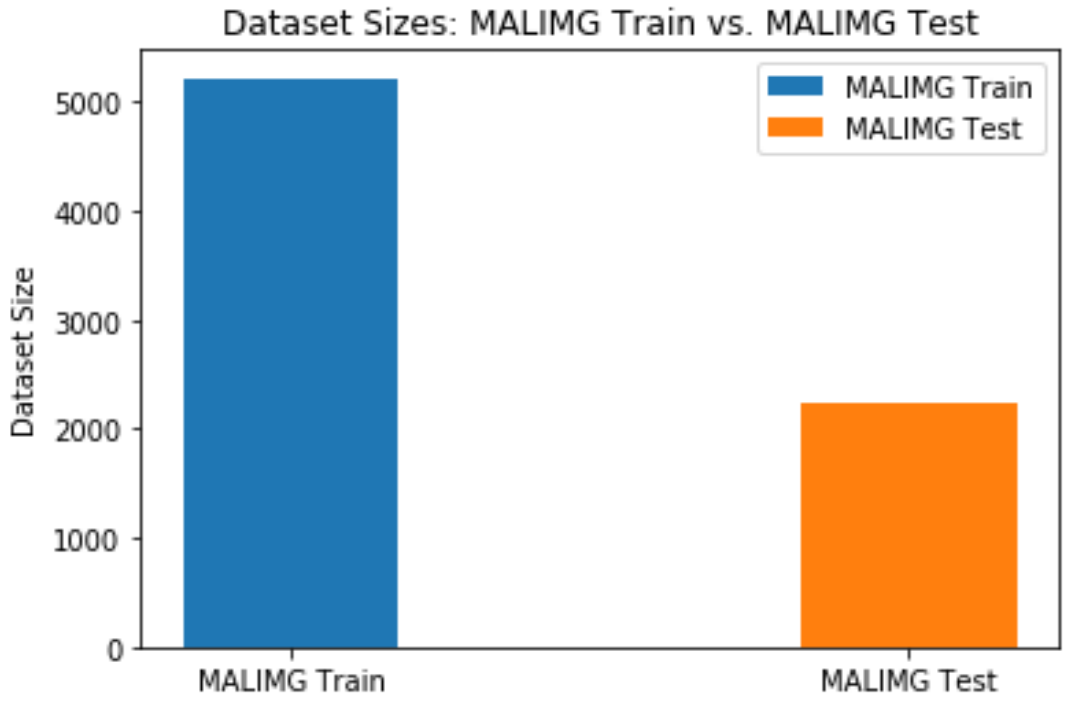

Figure 4.2: MALIMG Dataset

Description: This bar chart shows the total count of MALIMG samples split into training and test subsets across 25 malware families.

This data set consists of 7459 samples, with 25 different classes of malware families. The data set is divided into 5221 training samples and 2238 test samples as shown in Figure 4.2. The data set is effectively loaded and preprocessed using the Keras ImageDataGenerator class. This includes resizing the images to a uniform size of 64×64 pixels and normalising the data to ensure consistency in the input features. The preprocessing stage is crucial to prepare the data set for further analysis and training of the model.

After obtaining and preparing the data set, an exploratory data analysis is performed. This stage is important to understand the distribution of classes within the data set and to visually examine a subset of malwares, as shown in Figure 4.3

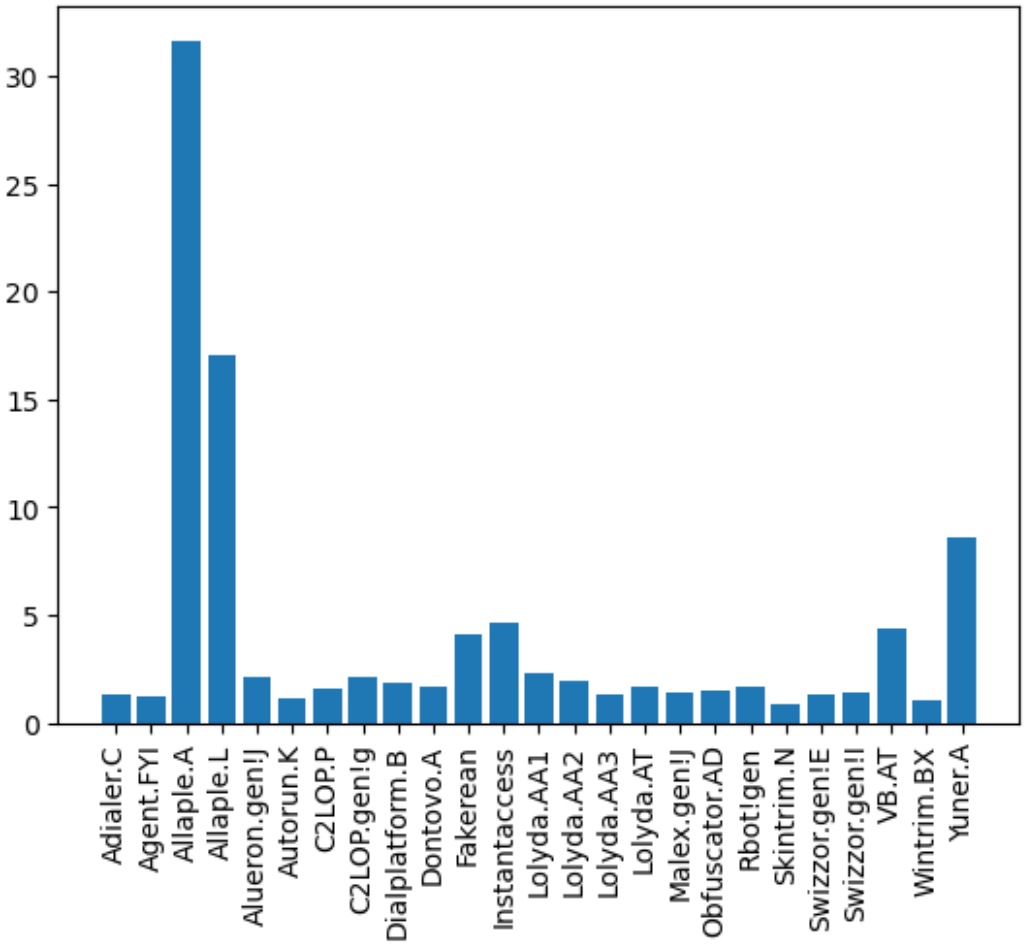

Figure 4.3: MALIMG Malware Distribution

Description: This histogram displays the distribution of individual malware family counts within the entire MALIMG dataset.

Table 4.1: Base Model Performance Metrics for the Model

Metric	**Value**
Train Loss	0.1082
Train Accuracy	0.9726
Validation Loss	0.2452
Validation Accuracy	0.9310
Test Loss	0.1143
Test Accuracy	0.9790

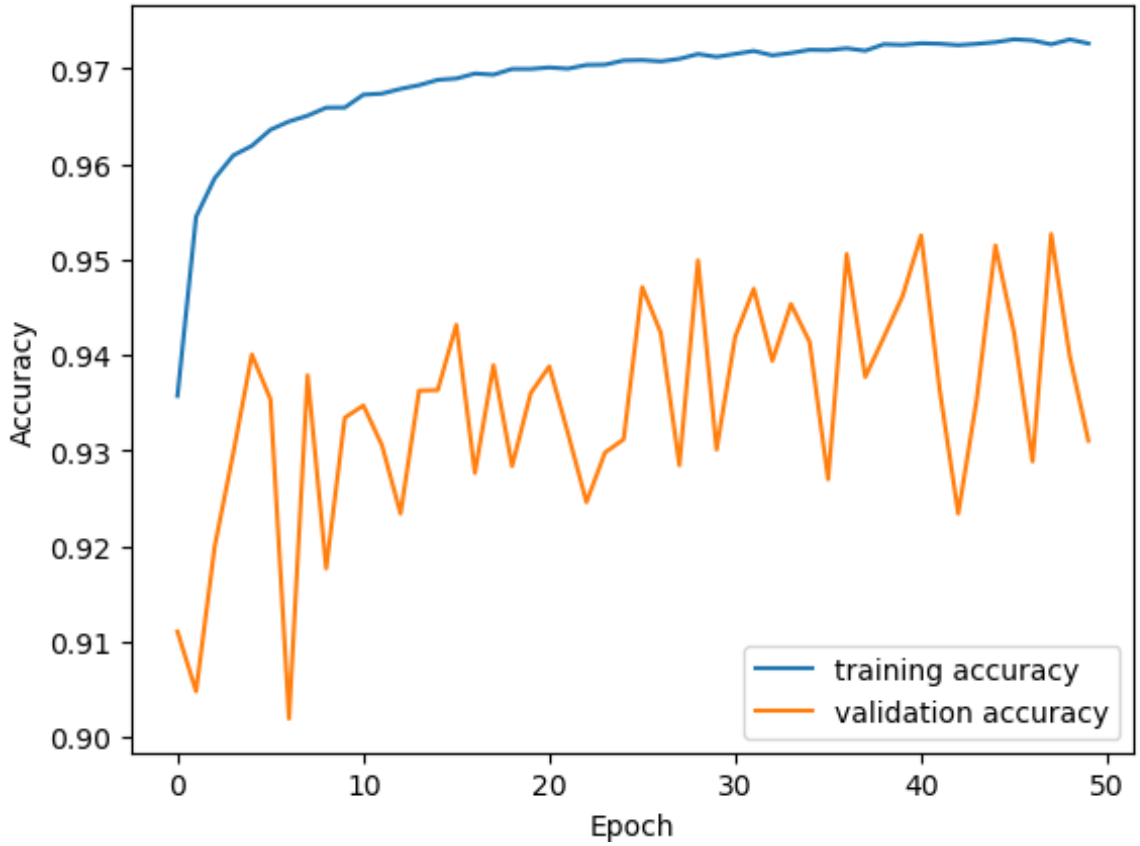

Figure 4.4: Base Model Performance Graph

Description: Training and validation accuracy and loss curves over epochs for the base CNN model on EMBER/BODMAS data.

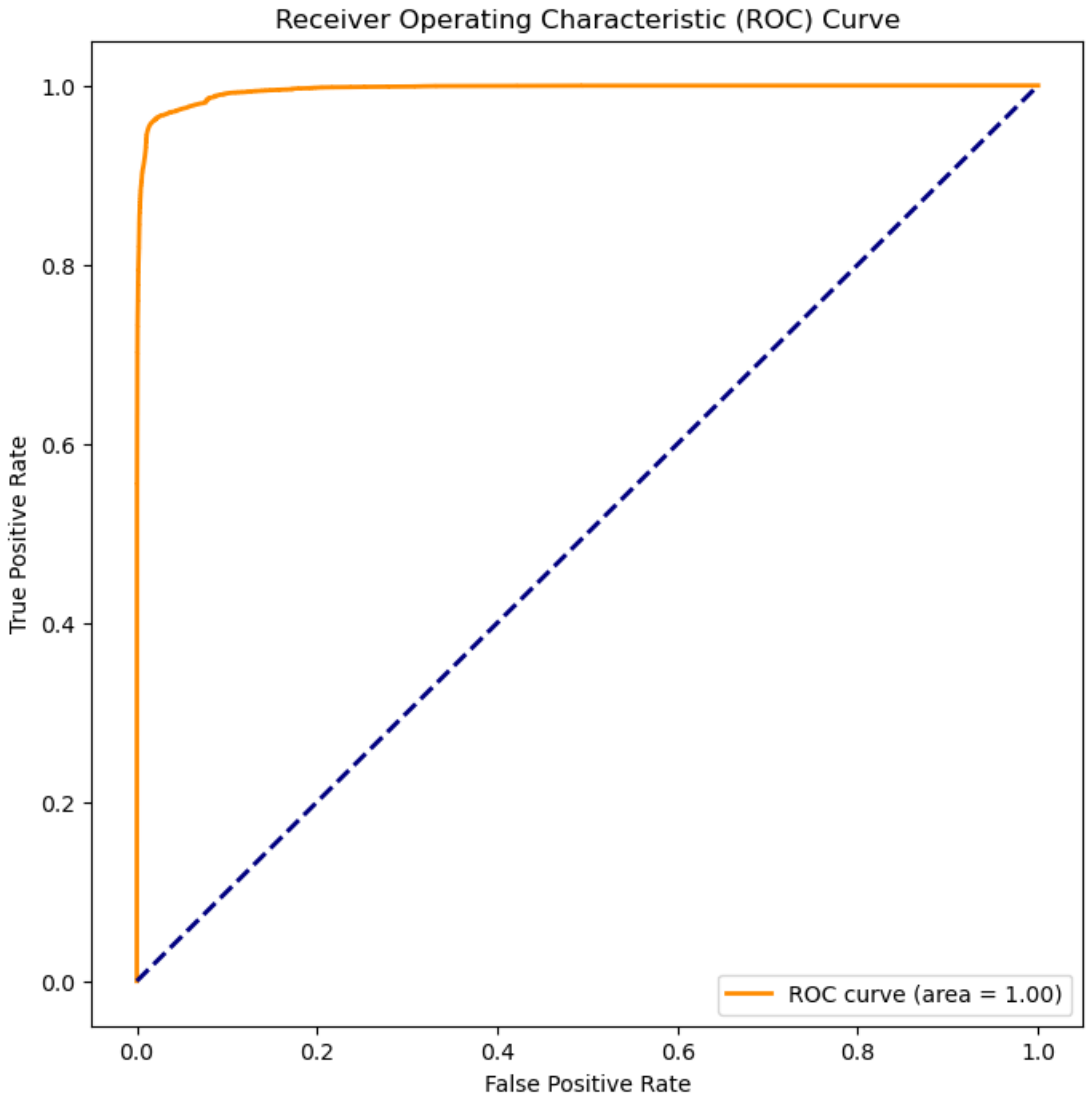

Figure 4.5: Base Model ROC

Description: Receiver Operating Characteristic curve showing the true-positive versus false-positive rate for the base model.

Transfer learning is employed by adapting a pre-trained model to the new task. This process involves modifying the initial and final layers and incorporating two new layers to suit the specific requirements of the task. Subsequently, the model is trained on the new dataset and assessed using metrics comparable to those employed for the original model.

Training performance, ROC curve, and confusion matrix are graphically shown to offer a deeper understanding of the learning progress and effectiveness of the model.

Table 4.2: Summary of key hyperparameters and configuration used in the thesis experiments.

Hyperparameter	**Symbol**	**Value(s)**	**Scope / Notes**
Learning rate	η	0.001	Explicitly set in the EMBER/BODMAS CNN experiments (Adam).
Epochs	E	50 (EMBER/-BODMAS); 5 (MALIMG)	Number of training epochs used in each training procedure (including within CV folds where applicable).
Training batch size	B_{train}	1000 (EMBER/-BODMAS); 32 (MALIMG)	Mini-batch size used during gradient-based training.
Data-loader batch size	B_{load}	10000	Batch size used by the image data loader (`flow_from_directory`); this controls loading/iteration behaviour and is distinct from B_{train}.
Weight decay (L2)	λ	5×10^{-4}	L2 regularisation strength applied via $\ell_2(\lambda)$ (kernel regulariser).
Optimiser	–	Adam (`Adam` / `'adam'`)	EMBER/BODMAS uses an explicit Adam optimiser object; MALIMG uses `'adam'` as specified in the compile step.
Loss function	–	`sparse_categorical_crossentropy` (EMBER/BODMAS); `categorical_crossentropy` (MALIMG)	Loss matches label encoding: sparse integer labels vs one-hot labels.

4.7 Implementation

Our transfer learning approach for malware classification incorporates data from both EMBER and BODMAS. The training data set comprises 800,000 samples. The test data set consists of around 134,435 samples that were used to build the base model.

Once the data are retrieved, they are partitioned into three distinct sets: training, validation, and testing. A substantial proportion of the data are assigned to the validation set in order to assess the model's performance throughout training. The input features are normalised using the StandardScaler function from the'sklearn'.preprocessing' library. This is done to ensure that all features are scaled uniformly, which is crucial for the optimal performance of neural network models. The data is converted to align with the specifications of Conv2D layers. Transformed into a 2D tensor to facilitate convolutional processes.

The fundamental architecture consists of two Conv2D layers with ReLU activation functions, each of which is subsequently followed by BatchNormalization to retrieve information. A Flatten layer transforms 2D feature maps into a 1D vector. The base model consists of several interconnected layers using rectified linear unit (ReLU) activa-

tion functions. Batch normalisation is applied after each layer, except for the last layer, which uses a softmax activation function for binary classification. L2 regularisation is implemented in all layers, excluding the input layer, to mitigate overfitting. The base model is compiled using the Adam optimiser, a sparse categorical crossentropy loss function, and accuracy as the evaluation metric. The training process consists of multiple epochs, where each epoch involves processing a batch of data from both the training and validation datasets. The model defines hyperparameters such as learning rate, and batch size see Table 4.2 . As an illustration, the learning rate is set to 0.001, the number of epochs is set to 50, and the batch size is set to 1000 based on manual adjustments during training. Our base model shows a consistent improvement in both training and validation accuracy, suggesting that the model is learning relevant patterns from the data without significant overfitting, as shown in Figure 4.4.

In order to assess the model's performance, we examine the test dataset to determine metrics such as accuracy, loss, and other pertinent measures. The Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) are employed to evaluate the model's ability to differentiate across classes. The high AUC values indicate better discrimination between classes. The smooth ROC curve across the training and validation sets suggests a good generalisation ability of the base model, as shown in Figure 4.5.

During the subsequent stage of our research, transfer learning is utilised by modifying the base model to meet the needs of the MALIMG task as shown in Figure 4.6. This process entails the removal of layers from both the beginning and end of the base model and the incorporation of layers for each of the respective ones previously removed that are specifically designed for MALIMG.

The transfer learning model is improved by incorporating additional layers, such as enabling the model to function in inference mode and providing a layer for multi-class classification. Adjusting the updated model involves utilising the same Adam optimiser, employing the categorical crossentropy loss function, and evaluating correctness. Model checkpoints are used to store the model according to its validation accuracy. During training, it was necessary to adjust the hyperparameters, such as the learning rate, the epochs, and the batch size. As an illustration, in the context of our transfer learning, we opted to leave the learning rate at 0.001 as is from the base model(pre-trained model), training the transfer learning model for 5 epochs, and using a batch size of 32 employing a callback function to store the model according to the validation accuracy.

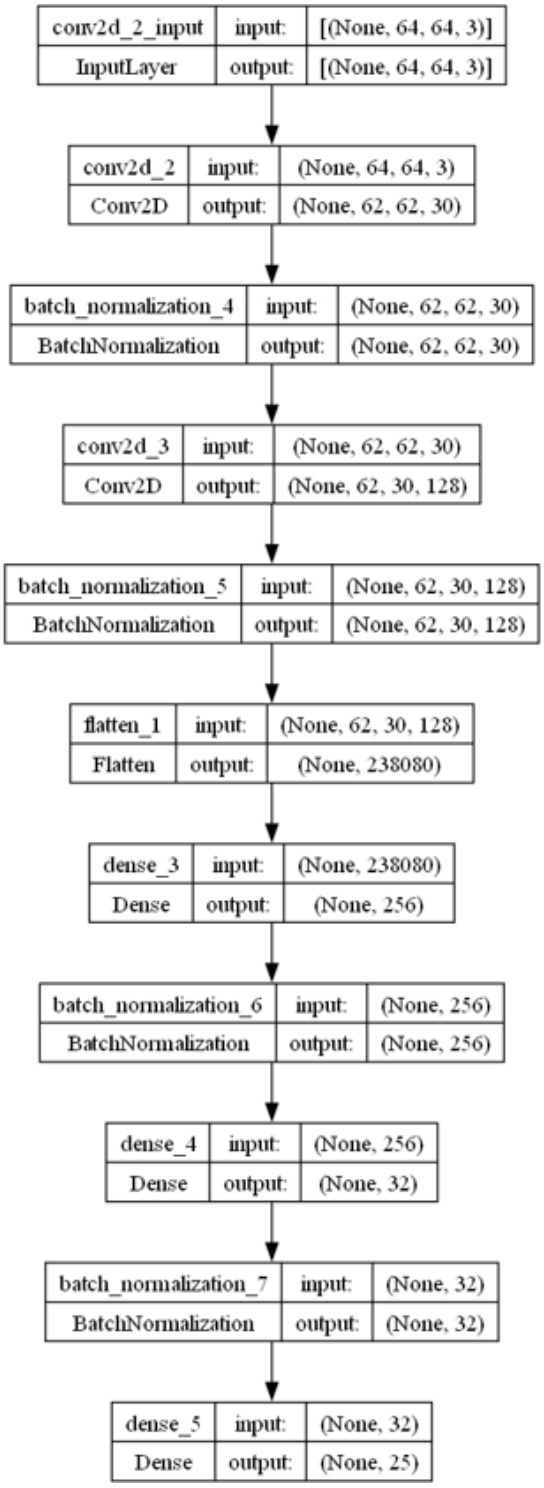

Figure 4.6: Transfer-learning model architecture.

Description: Diagram of the fine-tuned CNN backbone and newly added layers for MAL-IMG transfer learning.

4.8 Results and Evaluation

The transfer learning model demonstrates high performance, in training, validation, and test datasets consistently achieving more than 96% accuracy as shown in Figure 4.8. This

indicates performance on both unfamiliar and unseen data. The loss values are relatively low, suggesting a good fit with room for improvement. The consistency between the validation and the test metrics indicates minimal overfitting and solid generalisation capabilities.

A detailed breakdown of the classification metrics reveals precision, recall and F1 scores of 1.00 for classes such as 0, 1, 2, 5, 8, 9, 11, 12, 14, 16, 17, 18, and 24. Class 3 exhibits near perfect metrics with a decrease in recall (0.99) resulting in an F1 score of 0.99. Class 4 shows a minor decline in recall (0.96) with an F1 score of 0.98 See Table 4.5. Classes 6 and 7 show considerably lower scores; Class 6 has a precision of 0.76, a recall of 0.51 and an F1 score of 0.61; Class 7 has a precision of 0.86, a recall of 0.91 and an F1 score of 0.88. Classes 10, 13 and 15 experience slight reductions, in either precision or recall, while maintaining F1 scores ranging from (97–99), as shown in Table 4.5.

In classes 19 and 23, there is a decrease in effectiveness with F1 scores of 0.97 and 0.89 due to reduced recall. Classes 20 and 21 have the lowest metrics, with class 20 having a precision of 0.50, a recall of 0.66, and an F1 score of 0.57, while Class 21 has a precision of 0.43, a recall of 0.49, and an F1 score of 0.46; See Table 4.5.

The overall accuracy score of 0.97 indicates a general high performance. The average precision across all categories is at a level of 0.94 along with recall and an F1 score of 0.93 treat all classes equally irrespective of support. The model appears to perform well for most classes achieving perfect precision, recall and F1 scores for many classes' except for the following; six (6) seven (7) twenty (20) and twenty-one (21) which show notably lower performances indicating challenges faced by the model in handling these specific classes possibly due to data imbalance or class complexity See Table 4.5.

To improve the performance in these classes, data augmentation methods can be utilised to increase sample sizes for those classes while implementing class specific enhancements, like targeted feature extraction or custom loss functions could aid in addressing difficulties posed by these challenging classes. Adjusting the hyperparameters precisely and implementing regularisation methods could potentially lower the loss values.

The micro average ROC curve demonstrates an Area Under the Curve (AUC) value of 1.0 showing how effectively our model can distinguish between classes of malware family; see Figure 4.9.

Table 4.3: Comparison of Achieved Accuracy and Training Epochs

Model	**Accuracy (%)**	**Epochs**	**Dataset**
EfficientNetB7	99.63	10	Microsoft BIG Pratama & Sidabutar (2022)
LSTM	99.00	500	MALIMG **?**
VGG16	88.40	30	MALIMG Pant & Bista (2021)
Custom model	98.70	25	MALIMG Pant & Bista (2021)
LSTM	99.00	321	MALIMG Marastoni et al. (2021)
LSTM	94.00	30	Microsoft BIG Marastoni et al. (2021)
CNN (ours)	97.00	5	MALIMG

Table 4.4: Transfer Learning Model Performance Metrics

Metric	**Value**
Transfer Learning Train Loss	1.4020
Transfer Learning Train Accuracy	0.9753
Transfer Learning Validation Loss	1.3052
Transfer Learning Validation Accuracy	0.9692
Transfer Learning Test Loss	1.3052
Transfer Learning Test Accuracy	0.9692

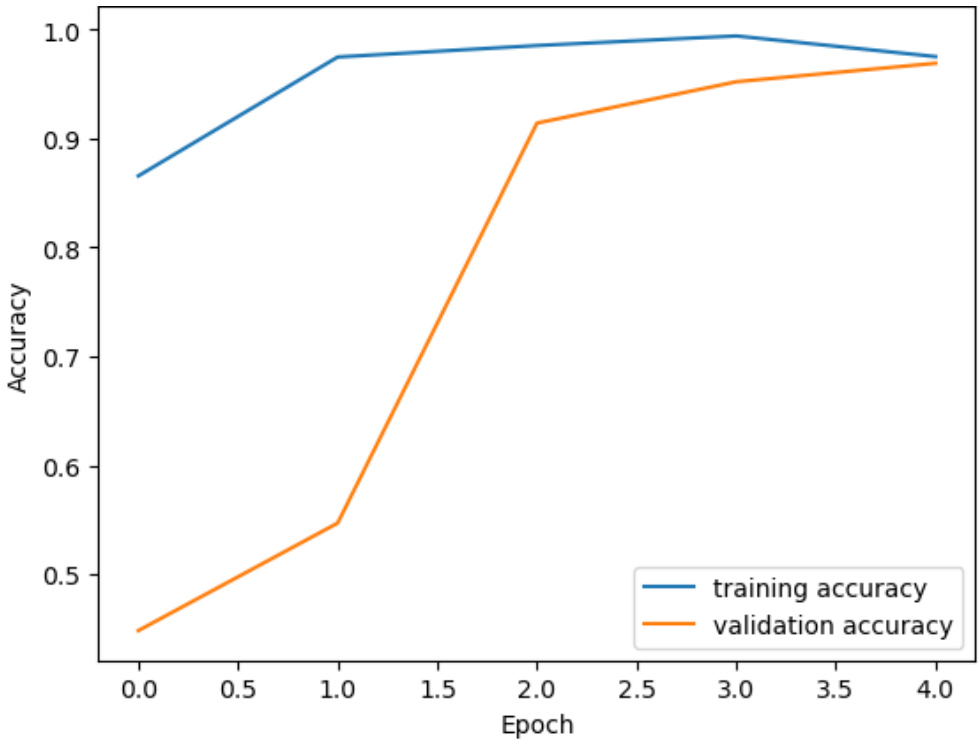

Figure 4.7: Transfer Learning Model Performance Graph

Description: Accuracy and loss curves over epochs for the transfer-learning model on the MALIMG dataset.

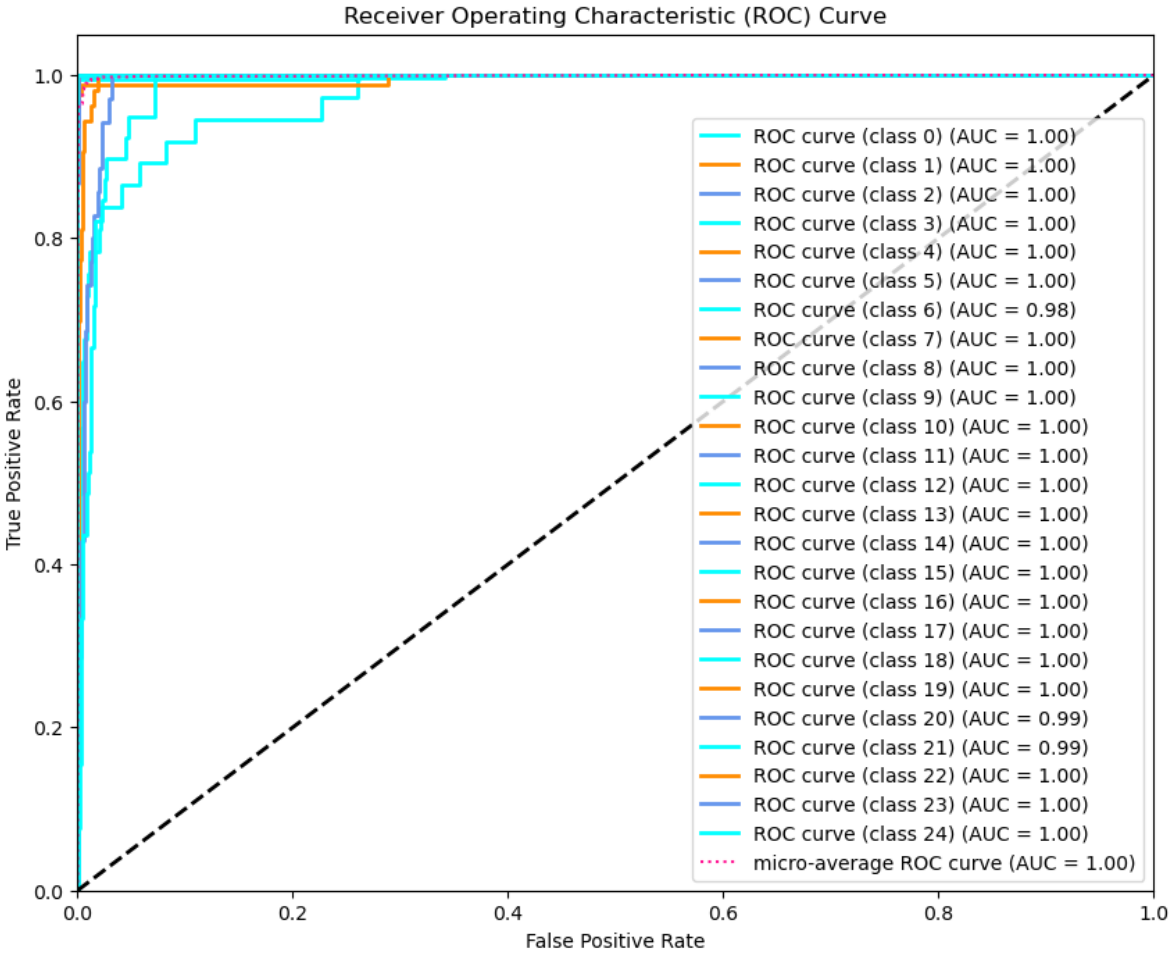

Figure 4.8: Transfer learning Model ROC

Description: ROC curve demonstrating high AUC values for the transfer-learning model across all classes.

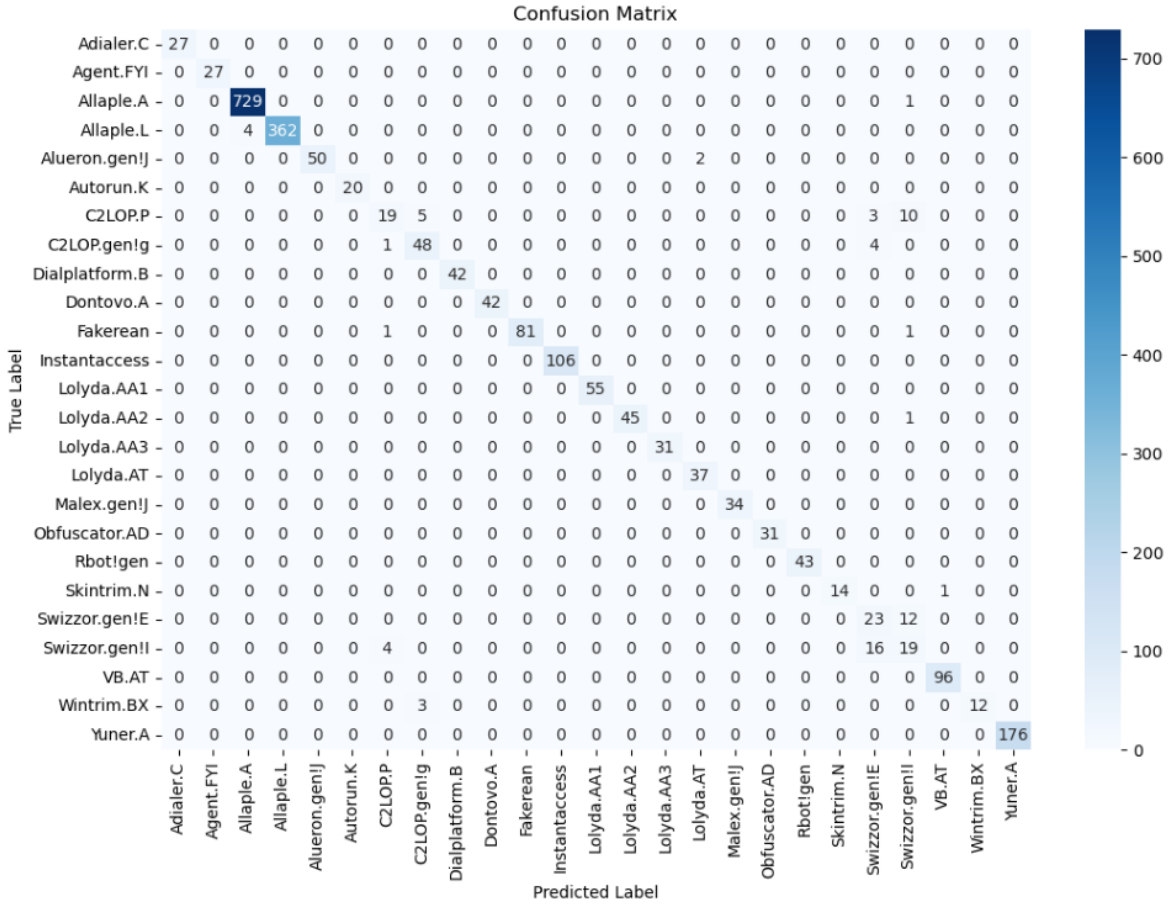

Figure 4.9: Transfer learning Model Confusion Matrix

Description: Confusion matrix heatmap showing per-class precision and recall for the transfer-learning model.

Our transfer learning model exhibited exceptional performance enhancements by leveraging optimised hyperparameters from the base model, namely by reducing the number of epochs and setting the batch size to 5 and 32, respectively, we were able to get impressive results in considerably shorter periods compared to other works with higher epochs Pratama & Sidabutar (2022), Panda et al. (2023), Aggarwal et al. (2021), Pant & Bista (2021), Marastoni et al. (2021).

Moreover, by using transfer learning and without careful hyperparameter tuning, we have significantly shown our model's performance, making it more adaptable to changing threat landscapes with fewer epochs and performance implications.

Table 4.5: Classification Report Metrics

Class	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	27
1	1.00	1.00	1.00	27
2	0.99	1.00	1.00	730
3	1.00	0.99	0.99	366
4	1.00	0.96	0.98	52
5	1.00	1.00	1.00	20
6	0.76	0.51	0.61	37
7	0.86	0.91	0.88	53
8	1.00	1.00	1.00	42
9	1.00	1.00	1.00	42
10	1.00	0.98	0.99	83
11	1.00	1.00	1.00	106
12	1.00	1.00	1.00	55
13	1.00	0.98	0.99	46
14	1.00	1.00	1.00	31
15	0.95	1.00	0.97	37
16	1.00	1.00	1.00	34
17	1.00	1.00	1.00	31
18	1.00	1.00	1.00	43
19	1.00	0.93	0.97	15
20	0.50	0.66	0.57	35
21	0.43	0.49	0.46	39
22	0.99	1.00	0.99	96
23	1.00	0.80	0.89	15
24	1.00	1.00	1.00	176
Accuracy			0.97	2238
Macro avg	0.94	0.93	0.93	2238
Weighted avg	0.97	0.97	0.97	2238

Chapter Conclusion

This chapter aims to demonstrate that **transfer learning** can overcome three persistent impediments in malware classification: (i) the coarse granularity of binary labels, (ii) the scarcity of well-labelled training samples across heterogeneous corpora, and (iii) the

prohibitive training cost of deep models built *de novo*. A CNN first exposed to EMBER and BODMAS was fine-tuned on the image-based MALIMG corpus and reached a 97 % multi-class accuracy and a macro-F_1 of 0.93 after only **five epochs**. These figures rival or surpass the state-of-the-art (Table 4.3) while reducing training time by an order of magnitude compared to EfficientNet or LSTM baselines that require 10 to 500 epochs Pratama & Sidabutar (2022), **?**), Pant & Bista (2021).

Furthermore, confusion-matrix analysis (Figure 4.9) revealed perfect precision/recall for 18 of 25 malware families and identified just four minority classes (#6, #7, #20, #21) where the recall remains below 70 %. These residual weaknesses are attributable chiefly to class imbalance rather than model capacity, suggesting that strategies such as targeted data augmentation or focal-loss re-weighting should close the remaining gap.

Key take-aways

1. *From binary to multi-class.* transfer learning enables a single backbone to scale from two labels (EMBER/BODMAS) to 25 malware families (MALIMG) without loss in overall accuracy.

2. *Cross-modal generalisation.* Knowledge distilled from static 2 381-dimensional feature vectors transfers effectively to byte-plot images, confirming that convolutional filters capture modality-agnostic structural cues.

3. *Efficiency.* Five fine-tuning epochs (batch 32, learning rate 1×10^{-3}) suffice to converge, cutting wall-clock time by about $\times\mathbf{10}$ relative to training the same architecture from scratch.

4. *Practical robustness.* Near-perfect ROC-AUC (≥ 0.99) and weighted $F_1 = 0.97$ across 2 381 hold-out samples indicate strong generalisation; error analysis is now confined to a small subset of minority families.

Recommended next steps. Follow-up work should (i) apply cost-sensitive loss or mixup augmentation to bolster minority-family recall, (ii) investigate feature-space adversarial re-training to harden the transfer learning backbone against evasion, and (iii) quantify latency/energy gains on edge hardware to validate the method for real-time deployment.

Collectively, these results confirm the chapter's central hypothesis: *transfer learning is a viable, computation-efficient route to fine-grained, multi-family malware detection across heterogeneous datasets.* The next chapter examines latent spaces.

5

Latent Spaces for Enhanced Malware Detection with Machine Learning Classifiers

5.1 Chapter Research Aims and Objectives

The primary objectives identified in this chapter are as follows:

- **Limited Generalisation of Conventional Models:** Conventional malware classification methods (Decision Trees, Random Forest, SVM) are highly dependent on handcrafted or statistical features, leading to poor generalisation to novel malware variants—particularly those employing obfuscation and polymorphism techniques Souri & Hosseini (2018), Sihag et al. (2021).
- **Inadequate Handling of Imbalanced Datasets:** Malware datasets typically suffer from class imbalance, which makes it challenging for models to effectively identify rare variants of malware Zahoora et al. (2022).
- **Computational Complexity and Hyperparameter Tuning:** Deep learning and classical models often require extensive computational resources and hyperparameter tuning, limiting their efficiency of deployment in real-time scenarios Kim & Cho (2022).

5.2 Most Relevant References

Generalisation and Obfuscation Issues:

- Souri & Hosseini (2018): Highlighted limitations in generalisation of conventional ML methods for malware detection.
- Sihag et al. (2021): Identified challenges posed by obfuscation and polymorphism of malware.

Latent Space and Generative Approaches:

- Kingma (2013): Introduction of Variational Autoencoders (VAEs), foundational to latent space modelling.
- Taylor & Eleyan (2021): Explored VAEs specifically to improve malware classification performance.
- Choi et al. (2024): Evaluated generative models (GANs and VAEs) in the creation of synthetic malware.
- Kiran (2024): Proposed HAVAEs, a hybrid model that combines adversarial autoencoders and VAEs for unsupervised malware detection.
- Ban et al. (2022): Demonstrated the utility of Conditional VAEs in addressing class imbalance in malware datasets.
- Gunduz (2022): used VAEs graphs to improve malware detection by extracting embeddings from API-call graphs.

Handling Imbalanced Datasets:

- Ban et al. (2022): Addressed dataset imbalance through Conditional VAEs.
- Zahoora et al. (2022): Investigated deep learning strategies for handling imbalanced malware data effectively.

Computational Efficiency and Feature Reduction:

- Saxe & Berlin (2017): Addressed high computational costs and adversarial sensitivity in malware detection.
- Dinh et al. (2024): Applied autoencoder-guided feature selection for efficient cybersecurity analysis.

5.3 Introduction

In today's hyperconnected world, malware attacks have risen to concerning proportions, presenting substantial challenges for cybersecurity. Sophisticated malware variants, such as viruses, worms, and ransomware, are progressively adept at circumventing traditional detection methods. The increasing complexity of these threats—spanning financial losses to critical infrastructure breaches—demands the creation of more resilient and adaptive strategies for the detection and classification of malware.

Conventional machine learning methods, such as Decision Trees, Random Forests, and Support Vector Machines, have been extensively used in malware classification efforts. These solutions generally depend on manually constructed features or statistical patterns extracted from malicious code or behavioural data from malware samples Souri & Hosseini (2018). However, their efficacy is limited by their inability to generalise to novel malware variants, especially in adversarial contexts. This constraint is exacerbated by malware developers that employ obfuscation and polymorphism techniques, which modify the appearance of the programme while preserving its fundamental operation Sihag et al. (2021). These tactics undermine conventional methods, necessitating needs capable of identifying both known and novel malware.

Latent-space models, especially those constructed with deep learning frameworks such as Variational Autoencoders (VAEs), have presented themselves as a viable solution to these issues. Variational Autoencoders (VAEs) are generative models that map input data into a reduced-dimensional latent space, preserving essential characteristics that may be obscured in the original high-dimensional space Kingma (2013). This latent space helps reduce the degree of dimensionality and the identification of fundamental patterns crucial to efficient malware classification Taylor & Eleyan (2021). The acquired latent representations can function as valuable characteristics for conventional machine learning algorithms, improving their ability to classify malware accurately despite obfuscation Kim & Cho (2022).

This research presents a novel hybrid methodology that integrates Variational Autoencoders (VAEs) with traditional machine learning approaches, including Random Forests, Logistic Regression, Decision Trees, Naïve Bayes and LightGBM in various configurations. We intend to overcome the shortcomings of conventional machine learning models and purely deep learning approaches by utilising the latent space representations acquired via the VAE. This methodology integrates the advantages of deep learning in feature extraction and the efficiency of traditional machine learning algorithms, providing a more adaptable and effective solution for malware detection.

A primary obstacle in utilising latent space models for malware classification is the inherent imbalance found in the majority of malware datasets. Malware families are typically characterised by a few predominant types, while several infrequent variants constitute a minor segment of the data set. This disparity affects the learning process, as models may find it challenging to adequately represent infrequent virus variants Zahoora et al. (2022). Moreover, the prevalent application of obfuscation and polymorphism complicates the ability of conventional models to identify alterations in malware presentation without jeopardising detection precision MicroAge (n.d.). Our proposed VAE-based approach addresses these difficulties by transforming malware samples into a latent space for the extraction of invariant features, hence enhancing the efficacy of downstream classifiers in adversarial and dynamic settings.

A further advantage of utilising latent space features is their interpretability. Conventional machine learning models frequently exhibit opaque decision-making processes, complicating practitioners' ability to understand the rationale behind certain decisions. In contrast, latent space representations provide a more rational understanding of the fundamental elements that influence classification decisions, facilitating more transparent and reliable systems for malware detection. Transparency is essential in critical cybersecurity applications, where knowing the reasoning behind a classification decision is as significant as the decision itself Masud et al. (2024).

This research presents a novel classification framework that combines latent space representations derived from a Variational Autoencoder (VAE) with traditional machine learning classifiers and various configurations. We pretrain the Variational Autoencoder (VAE) on a benchmark malware dataset Anderson & Roth (2018) Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021), then using the acquired latent features as input for classifiers like Random Forests, Logistic Regression, Decision Trees, Naïve Bayes, and LightGBM. We conducted comprehensive tests to compare our method with leading classical and deep learning models, illustrating that the incorporation of latent space characteristics markedly enhances classification performance without requiring hyperparameter optimisation.

The key contributions to this research are: We introduce an innovative hybrid method

that utilises VAE-derived latent space characteristics within traditional machine learning models for malware classification using various configurations. Secondly, we present a thorough empirical assessment of the influence of latent representations on classification with regard to processing efficiency, providing useful insights to both academia and practitioners in cybersecurity.

By integrating features of latent space with conventional machine learning algorithms, we offer a more efficient and comprehensible approach to malware identification. Our methodology not only improves performance and accuracy, but also provides a viable strategy for addressing the continuously changing threat environment presented by sophisticated malware. This research explores new opportunities for utilising latent space models in cybersecurity, enabling analysts to enhance the detection and classification of malware in the changing threat landscape.

Although Variational Autoencoders (VAEs) have been previously explored for malware detection and feature reduction, most prior work integrates VAEs within deep pipelines (e.g. CNNs, GANs, or Graph VAEs) or uses them for data augmentation and adversarial generation rather than as direct feature extractors for traditional classifiers. Our work is distinct in systematically demonstrating that compact VAE-derived latent vectors can be used to train conventional machine learning classifiers such as Random Forests and LightGBM with high accuracy and robustness. Unlike previous studies, we validate our approach in two benchmark PE datasets (EMBER and BODMAS), multiple data splits, and random seeds, employing statistical significance testing (t-tests) to establish the robustness of the results. Additionally, we explicitly quantify execution time and memory efficiency, showing that high-performance malware detection can be achieved without hyperparameter tuning, making the method particularly suitable for real-time, resource-constrained cybersecurity environments.

5.4 Related Work

Investigating the possibilities for synthetic malware generation as opcode sequences using generative adversarial networks (GANs) and deep variational autoencoders (VAE). Choi et al. (2024) focused on assessing synthetic malware in misleading machine learning classifiers, their research found that neither VAE nor GAN could effectively produce malware that avoids detection. However, the WGAN-GP algorithm indicated promise in enhancing synthetic malware production since it needed more synthetic samples to get effective detection. Although the difficulty of avoiding complex classifiers remains major, this study emphasises the increasing interest in generative models for malware generation.

For unsupervised malware identification Kiran (2024) presented the Hybrid Adversarial-variational Autoencoder (HAVAE), a model that combines the strengths of Adversarial Autoencoders (AAEs) and VAEs. Without large-scale labelled datasets, HAVAE enhances the identification of malicious software by catching subtle elements inside the latent space. Featuring strong performance across several datasets, the model generates realistic yet discriminative samples via the reparameterization method. Kiran's study highlights how well unsupervised learning detects malware and presents a creative way to feature extraction and classification in a changing threat environment.

With a Conditional Variational Autoencoder (CVAE) for Android malware family analysis,Ban et al. (2022) overcame the constraints of imbalanced datasets in malware classification. Their method maintains malware's behaviour throughout data augmentation, therefore enhancing classification accuracy. A macro-F1 score of 0.91 and an accuracy of 0.99% were obtained by the study showing that adding original malware

set greatly improves the performance of the classifier. This work shows how generative models used with conventional datasets could overcome dataset imbalance and enhance malware detection.

Using Graph Variational Autoencoder (GVAE) to parse API-call graphs taken from Android APK files, Gunduz (2022) presented a new malware detection system. The work sought to reduce the size of the graph node features and assess whether GVAE-reduced embeddings might improve malware detection performance. Showing notable gains, the GVAE was coupled with linear-based (SVM) and ensemble-based (LightGBM) systems. According to the study, GVAE-reduced embeddings roughly increased accuracy and F-measure rates for both models by 4%. Furthermore, LightGBM with a smaller set of 30 features showed even a higher accuracy rate of 0.967 by combining recursive feature elimination (RFE) with GVAE embeddings, thereby stressing the efficiency of this hybrid approach. The contributions of the paper consist of the introduction of GVAE for feature reduction in malware detection, the application of sophisticated feature selection strategies, and the validation of the suggested approaches by empirical results .

Transforming malware executables into image-based representations, Kumar et al. (2021) suggested an Autoencoder Enhanced Deep Convolutional Neural Network (AE-DCNN) for malware classification. This method improves classification performance by using a deep convolutional neural network (DCNN) with an encoder. In the Malimg data set, the authors show a 10-fold cross-valuation accuracy of 99.38% and an F1-score of 99.38%, proving the technique's great efficacy . Eliminating conventional methods such as feature engineering and reverse engineering, the AE-DCNN framework provides a more simplified and effective classification system. Furthermore, because of its texture-based study of malware files, the method demonstrates resistance to several obfuscation methods. By combining autoencoder improvements with deep learning methods, the AE-DCNN strategy marks a major breakthrough in malware classification and generates exceptional accuracy and resilience against obfuscation .

With an eye toward deriving a deep latent space that captures both feature and label interdependencies, Yeh et al. (2017) present a new framework termed the Canonical-Correlated AutoEncoder (C2AE) for multi-label classification. Their approach integrates deep canonical correlation analysis (DCCA) with an autoencoder architecture to develop joint feature-label embeddings, hence improving classification accuracy. The C2AE model can properly use label dependencies and manage missing labels during training by using a label-correlation-aware loss function. In terms of accuracy and computational efficiency, tests spanning several datasets show that C2AE is superior over state-of-the-art multi-label classification techniques.

Because this work uses deep learning methods to embed labels in a latent space where associations between them are maintained, it is especially pertinent to latent space learning. Such methods have significant consequences for malware classification since different properties or traits could have overlapping or dependent behaviours. This approach is a valuable reference for applications that need effective classification with complicated dependencies, since it underlines how latent spaces may be essential to reduce dimensionality while maintaining important information for prediction tasks.

Liu et al. (2018), working on 'Data Augmentation via Latent Space Interpolation for Image Classification,' The authors proposed to applying a uniform distribution on the latent space's feature representations by means of an Adversarial Autoencoder (AAE). Their more varied set of augmented data produced by using linear interpolation in this uniformly distributed latent space greatly enhanced the classification performance on

benchmark datasets, including ILSVRC 2012 and CIFAR-10.

The main value of this approach is its ability to create reasonable and instructive training samples by interpolating across several latent representations, hence solving the "hole" issue sometimes present in high-dimensional latent spaces. This method not only extends the conventional data augmentation process, but also offers a possible foundation for additional classification problems where producing reasonable alternative data points is essential.

By improving the usability of VAE-derived latent spaces as input to traditional machine learning models, we specifically addressed several important constraints noted in previous methods. Regarding processing efficiency, accuracy, generalisation, and the necessity of hyperparameter adjustment, this approach yields notable gains. The computational load related with processing high-dimensional data is greatly reduced by using small latent space as input. For real-time or large-scale datasets, this reduces the training and inference phases of machine learning models, hence increasing their practicality.

Previous research has applied generative models in malware analysis, including Variational Autoencoders (VAEs) for dimensionality reduction in classification tasks Taylor & Eleyan (2021), Graph VAEs on API-call graphs with LightGBM Gunduz (2022), CNN frameworks enhanced with autoencoders for malware images Kumar et al. (2021), and conditional VAEs for Android malware augmentation Ban et al. (2022). Other work has investigated VAE/GAN hybrids for the generation of synthetic malware and adversarial evasion Choi et al. (2024). Although these studies highlight the potential of generative models in malware analysis, they either focus on alternative data modalities (graphs, images), augmentation, or adversarial generation, or they rely on deep end-to-end architectures. In contrast, our study directly uses VAE-derived latent features as inputs to traditional classifiers, systematically benchmarking their performance and efficiency in static PE malware datasets. By combining accuracy, statistical rigour, and computational efficiency analyses, we contribute a novel deployment-orientated framework for malware classification that has not been explicitly addressed in the literature.

To further clarify how our work differs from previous approaches, Table 5.1 provides a structured comparison of the closest studies that employ VAEs or related generative techniques in malware detection, highlighting their core ideas, input representations, downstream learners, datasets, key findings, and the specific gaps our study addresses.

Table 5.1: Comparison of Related Works Leveraging VAEs in Malware Detection

Author (Year)	**Core Approach**	**Input Representation**	**Downstream Model(s)**	**Dataset(s)**	**Reported Findings**	**Gap vs. Our Work**
Taylor & Eleyan (2021)	VAEs for dimensionality reduction in malware classification	VAE latent features	Deep learning framing	General malware datasets	VAEs improve dimensionality reduction and classification	No focus on **classical ML on VAE latents**; lacks efficiency framing
Gündüz (2022)	Graph VAE on API-call graphs	GVAE embeddings	SVM, **LightGBM**	Android APK graphs	∼4% gain; LightGBM best with ∼30 features + RFE	Graph modality only; no **statistical testing across splits/seeds**
Kumar et al. (2021)	Autoencoder-enhanced CNN for malware images (AE-DCNN)	Image embeddings into CNN	Deep CNN	Malimg dataset	Very high accuracy/F1 (∼99.38%)	End-to-end deep pipeline; no **efficiency trade-off or classical ML focus**
Ban et al. (2022)	CVAE for data augmentation in Android malware classification	CVAE-augmented samples	Conventional ML classifiers	Android malware	Accuracy/F1 gains under imbalance	Focus on augmentation; not a **systematic latent→classical ML efficiency study**
Choi et al. (2024)	VAE/GAN for malware generation and evasion	Synthetic samples (VAE/GAN)	Used to probe ML detectors	Multi-dataset simulations	WGAN-GP effective for synthetic generation	Generation/evasion only; not **deployment-oriented latent classifiers**
Our work (2025)	**VAE latent (32D) features + classical ML**	Tabular VAE mean vector ($\bar{z}$)	DT, NB, LR, **RF, LightGBM**	**EMBER & BODMAS** with multiple splits/seeds	High AUC/accuracy **without hyperparameter tuning**; statistical validation; efficiency gains	Does not include deep baselines or adversarial robustness; unique in **efficiency + statistical rigor**

5.5 Mathematical Representation

A Variational Autoencoder (VAE) consists of the following components Dilokthanakul et al. (2016), Zhang & Xu (2019), Luo et al. (2020), Chen et al. (2018):

1. Encoder: Maps input data x to a latent space with mean μ and log variance $\log \sigma^2$:

$$q(z|x) = \mathcal{N}(z; \mu(x), \sigma^2(x)I)$$

2. Decoder: Maps latent variables z to the input space:

$$p(x|z) = \mathcal{N}(x; \mu_d(z), \sigma_d^2(z)I)$$

3. Sampling: Using the reparameterization trick:

$$z = \mu(x) + \sigma(x) \odot \epsilon$$

where $\epsilon \sim \mathcal{N}(0, I)$

4. Loss Function:

$$\mathcal{L} = \mathbb{E}_{q(z|x)}[\log p(x|z)] + D_{KL}(q(z|x)\|p(z))$$

where: - $\mathbb{E}_{q(z|x)}[\log p(x|z)]$ is the reconstruction loss - $D_{KL}(q(z|x)\|p(z))$ is the KL divergence term

Total loss: $\mathcal{L}$ = Reconstruction Loss + KL Divergence

We introduce a new algorithm that uses VAE latent space representations as features for traditional machine learning classifiers, unlike the classifiers trained on the raw input features, the models that use these learnt latent representations with significantly reduced dimensions are comparably accurate and more robust. The VAE encodes high-dimensional data into a low-dimensional latent space, which is a compact and informative representation of the data that helps to uncover the structure of the data and improve the performance of the downstream classifier as shown in Figure 5.1.

Algorithm 3 Classifier Evaluation with Original Features

function BUILD(data)
    $C \leftarrow$ YourClassifier()
    C.fit($data.features, data.labels$)
    return C
end function
function EVALUATE(C, test_data)
    $pred \leftarrow C$.predict($test\_data.features$)
    $acc \leftarrow$ calc_accuracy($pred, test\_data.labels$)
    return acc
end function
$data \leftarrow$ LoadData()
$C \leftarrow$ Build($data$)
$init\_acc \leftarrow$ Evaluate($C, data.test\_data$)
Print "Accuracy with all features: ", $init\_acc$

Algorithm 4 Classifier with Latent Space Representations

```
1: function BuildVAE(train_data)
2:     vae ← VAEModel()
3:     vae.compile(optimiser = adam, loss = vae_loss)
4:     vae.fit(train_data.features, train_data.features,
   epochs = 50, batch_size = 64)
5:     return vae
6: end function
7: function ExtractLatent(vae, data)
8:     encoder ← vae.get_encoder()
9:     z_mean ← encoder.predict(data.features)
10:    return z_mean
11: end function
12: function BuildClassifier(latent_data)
13:    C ← YourClassifier()
14:    C.fit(latent_data, data.labels)
15:    return C
16: end function
17: function Evaluate(C, test_latent_data)
18:    pred ← C.predict(test_latent_data)
19:    acc ← calc_accuracy(pred, test_data.labels)
20:    return acc
21: end function
22: data ← LoadData()
23: vae ← BuildVAE(data.train)
24: train_latent ← ExtractLatent(vae, data.train)
25: test_latent ← ExtractLatent(vae, data.test)
26: C ← BuildClassifier(train_latent)
27: latent_acc ← Evaluate(C, test_latent)
28: Print "Accuracy with latent features: ", latent_acc
```

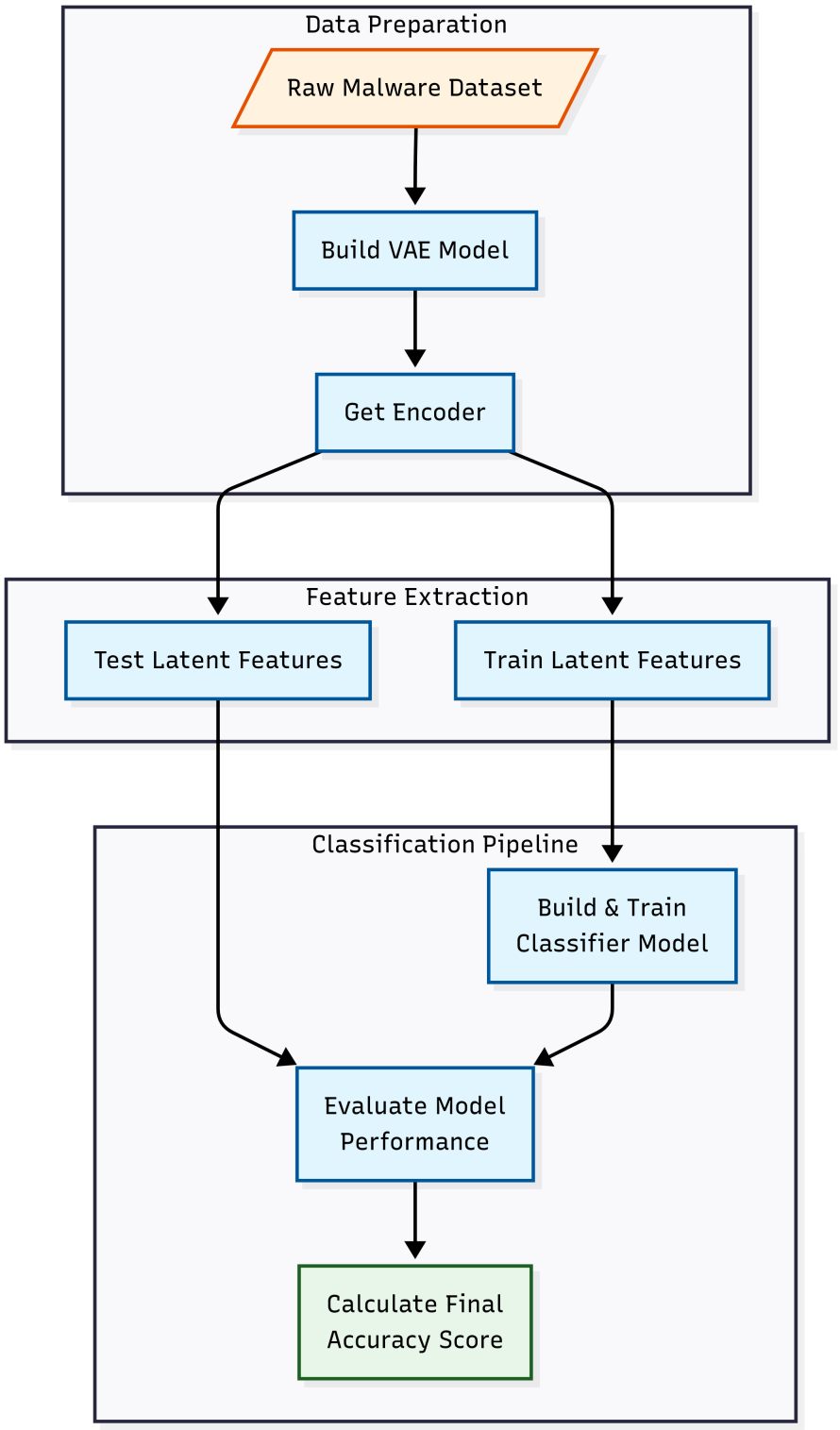

Figure 5.1: Perturbation flow chart / algorithmic representation.

Description: This flow chart outlines the step-by-step VAE-based process for generating latent-space representations and applying perturbations to evaluate classifier robustness.

5.6 Methodology

In order to assess the effectiveness of the latent space representations in the context of malware classification, a two-phase experimental design was carried out in this chapter. In the first phase, two benchmark malware datasets **??** were first normalised and then cleaned. The datasets, which were initially dimensional feature vectors of size 2,381, were cleaned to exclude instances without labels and divided into training, validation, and test sets. More training-test splits (30%/30%, 50%/30%, 70%/30%) were also made to ensure that model performance was evaluated with different levels of data availability. All features were normalised between [0,1] range using MinMaxScaler to improve convergence and enhance classifier performance, since the features were on different scales.

In the next phase, a Variational Autoencoder (VAE) was trained on the preprocessed training data to learn compact latent features. The encoder was the dense layers that reduced the dimensionality to a 32dimensional latent space of mean (z_mean) and log variance (z_log_var). The latent vectors were generated using a reparameterization trick to ensure that the latent space was learnt smoothly while the decoder attempted to reconstruct the original input. The VAE was trained for 50 iterations with a batch size of 64, using a custom loss function that incorporated Mean Squared Error for reconstruction loss and Kullback-Leibler divergence for regularisation.

The latent representations learnt by the VAE were then employed as input features to five different machine learning classifiers, including Decision Tree, Naive Bayes, LightGBM, Logistic Regression, and Random Forest. These models were chosen as representative of different approaches to learning and are known to perform well in malware detection tasks. Each classifier was trained with five-fold cross validation, two different random seeds (42 and 123) and accuracy and Area Under the Curve (AUC) were used as performance metrics to evaluate the models on the test set. T-tests were used to determine the statistical significance of performance differences to confirm that the observed performance differences were robust. In addition, confusion matrices, classification reports, and ROC curves were constructed to provide a more complete view of performance, and execution times were recorded to judge computational efficiency.

With the help of the unsupervised feature learning component based on the VAE and the traditional classifiers, the detection accuracy is improved while the computational cost and the need for extensive hyperparameter tuning are reduced. All models that were trained in this research, including the VAE encoder and the classifiers, have been saved to ensure reproducibility. The complete Python code for data preprocessing, VAE training, the extraction of the latent space, and the training of the classifiers is provided in the supplementary material to enable further research into machine learning based malware detection.

5.7 Statistical Significance Testing in Malware Detection Experiments

The experiments evaluate model performance under multiple resampling regimes to assess robustness. Specifically, models are evaluated across training–test split configurations $s \in \mathcal{S} = \{30/30,\ 50/30,\ 70/30\}$ and random seeds $z \in \mathcal{Z} = \{42,\ 123\}$. Within each (s, z) configuration, 5-fold cross-validation is used on the training portion for model selection and stability analysis. The final performance is reported on the corresponding held-out test set using a scalar metric m (e.g., AUC, F1-score, accuracy).

5.7.1 Notation used for hypothesis tests

Let $m_{s,z}^{(A)}$ denote the test-set metric achieved by method A under split configuration s and seed z, and analogously $m_{s,z}^{(B)}$ for method B. Because both methods are evaluated under the same (s, z) configurations, comparisons are treated as paired, with per-configuration differences

$$d_{s,z} = m_{s,z}^{(A)} - m_{s,z}^{(B)}, \qquad (s, z) \in \mathcal{S} \times \mathcal{Z}. \tag{5.1}$$

The total number of paired replicates is $R = |\mathcal{S}||\mathcal{Z}| = 6$.

5.7.2 Paired t-test (parametric mean comparison)

To test whether the mean performance differs between two methods across resampling regimes, the paired t-test considers

$$H_0 : \mathbb{E}[d_{s,z}] = 0 \qquad \text{vs} \qquad H_1 : \mathbb{E}[d_{s,z}] \neq 0. \tag{5.2}$$

Let $\bar{d} = \frac{1}{R}\sum_{(s,z)} d_{s,z}$ and $s_d^2 = \frac{1}{R-1}\sum_{(s,z)}(d_{s,z} - \bar{d})^2$. The test statistic is

$$t = \frac{\bar{d}}{s_d/\sqrt{R}} \sim t_{R-1} \quad \text{under } H_0, \tag{5.3}$$

assuming that the paired differences are approximately normally distributed.

5.7.3 Wilcoxon signed-rank test (nonparametric paired comparison)

Because performance metrics may be non-normal and R is small, a nonparametric paired alternative is the Wilcoxon signed-rank test, which tests whether the median of $\{d_{s,z}\}$ is zero. After removing zero differences, rank $|d_{s,z}|$ to obtain ranks $r_{s,z}$, and compute

$$W^+ = \sum_{(s,z):d_{s,z}>0} r_{s,z}, \qquad W^- = \sum_{(s,z):d_{s,z}<0} r_{s,z}, \qquad W = \min(W^+, W^-). \tag{5.4}$$

5.7.4 p-value (interpretation for model comparisons)

For any test statistic T with observed value t_{obs}, the p-value is

$$p = \Pr(\text{a test statistic at least as extreme as } t_{\text{obs}} \mid H_0)\,. \tag{5.5}$$

In this thesis, a small p-value indicates that the observed performance gap across (s, z) regimes is unlikely to be explained by resampling variability alone under H_0. Importantly, p is not the probability that H_0 is true.

Note on cross-validation. Fold-level cross-validation scores within a single run are not treated as independent replicates for hypothesis testing because training sets overlap across folds. Accordingly, hypothesis tests are performed using one test-set metric per (s, z) configuration.

5.8 Evaluation and Results

This section evaluates how well *latent space features* support malware classification under realistic experimental variation. Rather than relying on a single split or seed, we systematically tested three data partition regimes (30/30, 50/30, and 70/30) and two

random seeds (42 and 123). This design separates two often-confounded questions: (i) whether performance is an artefact of a favourable random initialisation, and (ii) whether models remain reliable as the amount of training data changes. A central contribution of this chapter is that **strong performance is achieved directly from the latent representation without hyperparameter tuning**, demonstrating that the representation itself carries sufficient discriminative structure for downstream classifiers (Tables 5.6–5.9).

Table 5.2: Comparison of random states (42 vs. 123) for each classifier within each EMBER split.

Split	Classifier	t-statistic	p-value
	EMBER 30/30		
	Decision Tree	0.0754	0.9418
	Naive Bayes	0.0000	1.0000
	LightGBM	0.0000	1.0000
	Logistic Regression	0.0000	1.0000
	Random Forest	-0.3691	0.7216
	EMBER 50/30		
	Decision Tree	-0.0477	0.9631
	Naive Bayes	0.0000	1.0000
	LightGBM	-0.6680	0.5229
	Logistic Regression	0.0000	1.0000
	Random Forest	-6.6940	0.0002
	EMBER 70/30		
	Decision Tree	-0.4554	0.6609
	Naive Bayes	0.0000	1.0000
	LightGBM	-0.2266	0.8264
	Logistic Regression	0.0000	1.0000
	Random Forest	0.0168	0.9870

Latent features yield strong accuracy with low overhead

Across both BODMAS and EMBER, the latent features support high classification performance while reducing computational cost. This is reflected not only in predictive metrics (AUC/accuracy/CV stability), but also in training+evaluation runtimes reported in Table 5.12. The key implication is practical: *latent space learning acts as a compact, information-dense representation that enables efficient model training and inference compared to operating on high-dimensional raw features.* This matters for security deployment contexts (SIEM/EDR/edge systems) where model refresh, throughput, and resource ceilings often constrain what can be used in production.

Model-level findings: what is robust, what is sensitive, and why

Decision Tree: stable generalisation in latent space. Decision Trees show consistently strong behaviour across splits and seeds, suggesting that the latent representation exposes *hierarchical, threshold-like separations* that trees can exploit reliably. The observed stability across random states indicates that their performance is not driven by

Table 5.3: Comparison of splits for each classifier across random states within EMBER.

Classifier	**Random State**	**Comparison**	**t-statistic**	**p-value**
Decision Tree	42	30/30 vs 50/30	-16.1213	0.0000
		30/30 vs 70/30	-28.7708	0.0000
		50/30 vs 70/30	-12.2503	0.0000
	123	30/30 vs 50/30	-13.8541	0.0000
		30/30 vs 70/30	-25.6003	0.0000
		50/30 vs 70/30	-10.1159	0.0000
Naive Bayes	42	30/30 vs 50/30	7.7939	0.0001
		30/30 vs 70/30	4.7865	0.0014
		50/30 vs 70/30	-3.8106	0.0052
	123	30/30 vs 50/30	7.7939	0.0001
		30/30 vs 70/30	4.7865	0.0014
		50/30 vs 70/30	-3.8106	0.0052
LightGBM	42	30/30 vs 50/30	-2.2882	0.0514
		30/30 vs 70/30	-0.6418	0.5390
		50/30 vs 70/30	1.3736	0.2068
	123	30/30 vs 50/30	-4.2594	0.0028
		30/30 vs 70/30	-1.1109	0.2989
		50/30 vs 70/30	2.8681	0.0209
Logistic Regression	42	30/30 vs 50/30	1.2077	0.2616
		30/30 vs 70/30	0.0717	0.9446
		50/30 vs 70/30	-0.9205	0.3842
	123	30/30 vs 50/30	1.2077	0.2616
		30/30 vs 70/30	0.0717	0.9446
		50/30 vs 70/30	-0.9205	0.3842
Random Forest	42	30/30 vs 50/30	-25.5407	0.0000
		30/30 vs 70/30	-42.0416	0.0000
		50/30 vs 70/30	-6.7869	0.0001
	123	30/30 vs 50/30	-40.6750	0.0000
		30/30 vs 70/30	-40.6750	0.0000
		50/30 vs 70/30	0.0000	1.0000

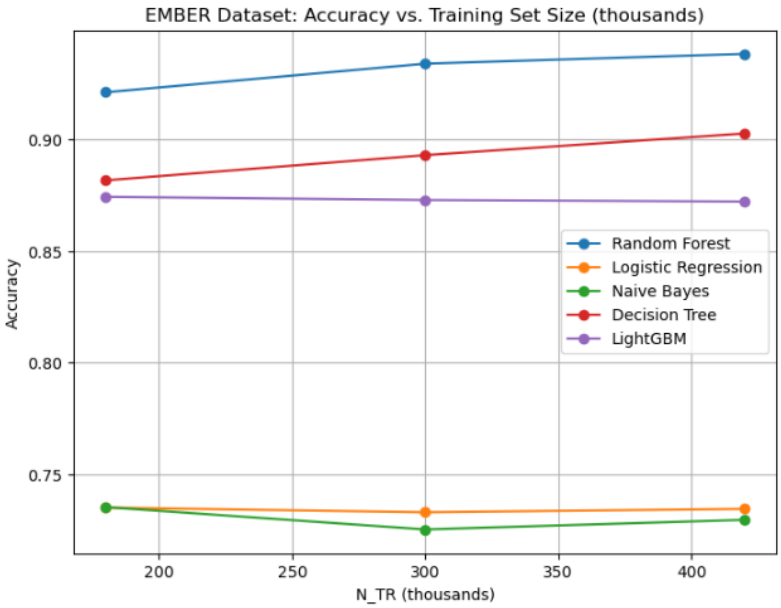

Figure 5.2: EMBER Latent Feature Accuracy VS Training

Description: Plot showing the classifier's accuracy progression over training epochs when using VAE-derived latent features on the EMBER dataset.

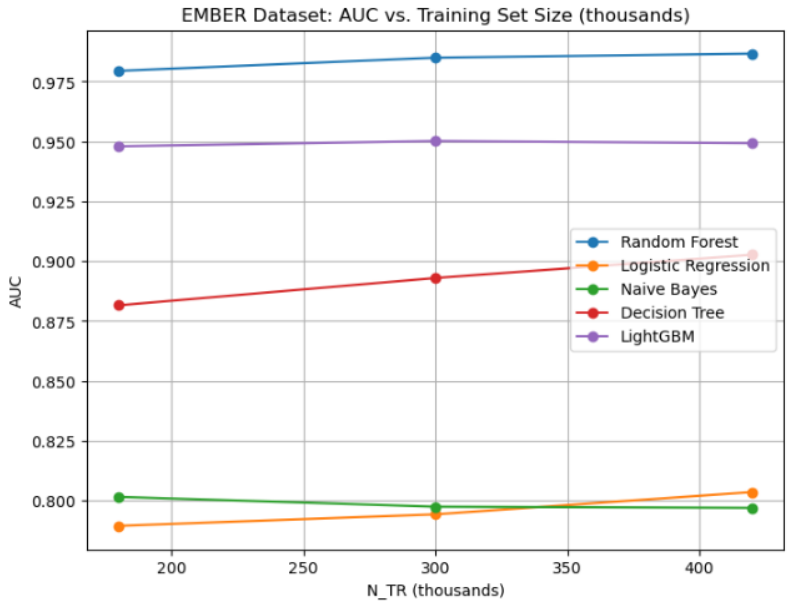

Figure 5.3: EMBER Latent Feature AUC VS Training

Description: Curve tracking the area under the ROC (AUC) across training epochs for the model trained on EMBER latent-space representations.

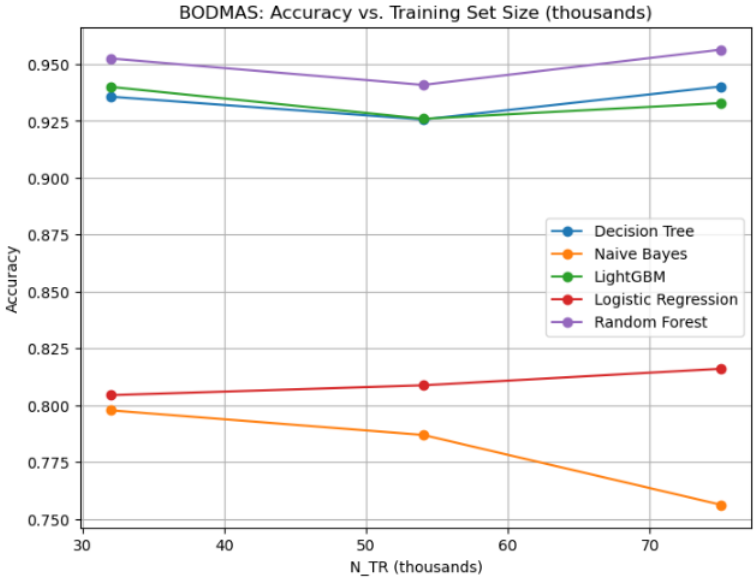

Figure 5.4: BODMAS Latent Feature Accuracy VS Training

Description: Plot illustrating accuracy improvements over training iterations using latent features extracted from the BODMAS dataset.

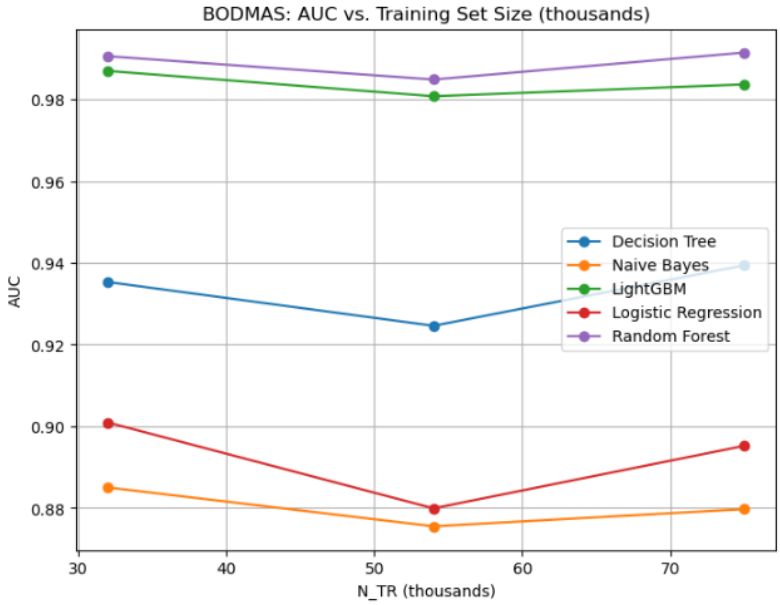

Figure 5.5: BODMAS Latent Feature AUC VS Training

Description: AUC progression curve showing model discrimination capability over epochs on BODMAS latent-space features.

Performance Metrics (EMBER)						
$\mathbf{N_{TR}}$ ($\mathbf{10^3}$)	$\mathbf{N_{TS}}$ ($\mathbf{10^3}$)	**Classifier**	**AUC**	**Accuracy**	**Mean CV Score**	**Std Dev**
30% Training, 30% Testing, 40% Holdout						
180	180	Random Forest	0.9794	0.9211	0.9156	0.0009
180	180	Logistic Regression	0.7894	0.7349	0.7346	0.0021
180	180	Naive Bayes	0.8015	0.7351	0.7355	0.0018
180	180	Decision Tree	0.8815	0.8816	0.8742	0.0016
180	180	LightGBM	0.9479	0.8742	0.8715	0.0011
50% Training, 30% Testing, 20% Holdout						
300	180	Random Forest	0.9849	0.9339	0.9321	0.0010
300	180	Logistic Regression	0.7942	0.7328	0.7331	0.0013
300	180	Naive Bayes	0.7974	0.7251	0.7263	0.0015
300	180	Decision Tree	0.8929	0.8929	0.8902	0.0012
300	180	LightGBM	0.9501	0.8728	0.8737	0.0016
70% Training, 30% Testing						
420	180	Random Forest	0.9866	0.9383	0.9356	0.0004
420	180	Logistic Regression	0.8035	0.7343	0.7345	0.0026
420	180	Naive Bayes	0.7969	0.7294	0.7302	0.0013
420	180	Decision Tree	0.9026	0.9026	0.8988	0.0007
420	180	LightGBM	0.9492	0.8721	0.8724	0.0011

Table 5.4: Performance of Various Classifiers on EMBER Dataset Splits

Performance Metrics (BODMAS)						
N_{TR} (10^3)	N_{TS} (10^3)	Classifier	AUC	Accuracy	Mean CV Score	Std Dev
30% Training, 30% Testing, 40% Holdout						
32	40	Decision Tree	0.9353	0.9356	0.9314	0.0021
32	40	Naive Bayes	0.8850	0.7977	0.7967	0.0054
32	40	LightGBM	0.9870	0.9399	0.9388	0.0027
32	40	Logistic Regression	0.9009	0.8044	0.7996	0.0027
32	40	Random Forest	0.9906	0.9524	0.9498	0.0026
50% Training, 30% Testing, 20% Holdout						
54	67	Decision Tree	0.9246	0.9256	0.9238	0.0012
54	67	Naive Bayes	0.8755	0.7869	0.7892	0.0010
54	67	LightGBM	0.9808	0.9259	0.9274	0.0014
54	67	Logistic Regression	0.8799	0.8087	0.8063	0.0018
54	67	Random Forest	0.9849	0.9408	0.9404	0.0017
70% Training, 30% Testing						
75	94	Decision Tree	0.9394	0.9401	0.9391	0.0012
75	94	Naive Bayes	0.8797	0.7562	0.7592	0.0042
75	94	LightGBM	0.9837	0.9328	0.9355	0.0005
75	94	Logistic Regression	0.8952	0.8160	0.8105	0.0023
75	94	Random Forest	0.9915	0.9563	0.9542	0.0005

Table 5.5: Performance of Various Classifiers on the BODMAS Dataset Splits

Model	No Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Random Forest	ALL	0.9603	0.9674	0.9527	0.9600	0.9940
Logistic Regression	ALL	0.5564	0.5414	0.7339	0.6231	0.6133
Decision Tree	ALL	0.9394	0.9362	0.9430	0.9396	0.9394
Naive Bayes	ALL	0.5346	0.9098	0.0761	0.1405	0.6075
Light GBM	ALL	0.9516	0.9513	0.9518	0.9515	0.9907

Table 5.6: EMBER Performance metrics for models with complete features and no hyperparameter tuning

Model	No Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Light GBM	ALL	0.9807	0.9840	0.9773	0.9807	0.9979
Logistic Regression	ALL	0.8872	0.8759	0.9021	0.8888	0.9589
Naive Bayes	ALL	0.5346	0.9098	0.0761	0.1405	0.6075
Random Forest	ALL	0.9628	0.9701	0.9550	0.9625	0.9947
Decision Tree	ALL	0.9176	0.9206	0.9139	0.9172	0.9271

Table 5.7: EMBER Performance metrics for models with complete feature and hyperparameter tuning

Model	No Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Logistic Regression	ALL	0.7466	0.6642	0.8206	0.7342	0.8457
LightGBM	ALL	0.9959	0.9953	0.9951	0.9952	0.9998
Decision Tree	ALL	0.9888	0.9847	0.9892	0.9869	0.9889
Random Forest	ALL	0.9945	0.9974	0.9896	0.9935	0.9997
Naive Bayes	ALL	0.4920	0.4547	0.9605	0.6172	0.5614

Table 5.8: BODMAS Performance metrics for models with complete feature and no hyperparameter tuning

Model	No Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Decision Tree	ALL	0.9881	0.9833	0.9887	0.9860	0.9881
LightGBM	ALL	0.9958	0.9947	0.9954	0.9950	0.9998
Logistic Regression	ALL	0.7482	0.6682	0.8133	0.7337	0.8428
Naive Bayes	ALL	0.4920	0.4547	0.9605	0.6172	0.5614
Random Forest	ALL	0.9945	0.9975	0.9896	0.9935	0.9997

Table 5.9: BODMAS Comparison of models with complete feature and hyperparameter tuning

Table 5.10: Comparison of random states (42 vs. 123) for each classifier within each split.

Split	Classifier	t-statistic	p-value
	BODMAS 30/30		
	Decision Tree	-0.2523	0.8072
	Naive Bayes	0.0000	1.0000
	LightGBM	0.0000	1.0000
	Logistic Regression	0.0000	1.0000
	Random Forest	0.4842	0.6413
	BODMAS 50/30		
	Decision Tree	0.0188	0.9855
	Naive Bayes	0.0000	1.0000
	LightGBM	0.0000	1.0000
	Logistic Regression	0.0000	1.0000
	Random Forest	0.2128	0.8368
	BODMAS 70/30		
	Decision Tree	0.4778	0.6456
	Naive Bayes	0.0000	1.0000
	LightGBM	0.0000	1.0000
	Logistic Regression	0.0000	1.0000
	Random Forest	-1.0562	0.3217

Classifier	Random State	Comparison	t-statistic	p-value
Decision Tree	42	30/30 vs 50/30	6.1411	0.0003
		30/30 vs 70/30	-6.2678	0.0002
		50/30 vs 70/30	-17.6574	0.0000
Decision Tree	123	30/30 vs 50/30	6.9828	0.0001
		30/30 vs 70/30	-6.4404	0.0002
		50/30 vs 70/30	-23.1574	0.0000
Naive Bayes	42	30/30 vs 50/30	2.7497	0.0251
		30/30 vs 70/30	10.9515	0.0000
		50/30 vs 70/30	13.7383	0.0000
Naive Bayes	123	30/30 vs 50/30	2.7497	0.0251
		30/30 vs 70/30	10.9515	0.0000
		50/30 vs 70/30	13.7383	0.0000
LightGBM	42	30/30 vs 50/30	7.5691	0.0001
		30/30 vs 70/30	2.4185	0.0419
		50/30 vs 70/30	-11.1428	0.0000
LightGBM	123	30/30 vs 50/30	7.5691	0.0001
		30/30 vs 70/30	2.4185	0.0419
		50/30 vs 70/30	-11.1428	0.0000
Logistic Regression	42	30/30 vs 50/30	-4.1039	0.0034
		30/30 vs 70/30	-6.0987	0.0003
		50/30 vs 70/30	-2.8430	0.0217
Logistic Regression	123	30/30 vs 50/30	-4.1039	0.0034
		30/30 vs 70/30	-6.0987	0.0003
		50/30 vs 70/30	-2.8430	0.0217
Random Forest	42	30/30 vs 50/30	5.9883	0.0003
		30/30 vs 70/30	-3.2663	0.0114
		50/30 vs 70/30	-15.2201	0.0000
Random Forest	123	30/30 vs 50/30	5.5212	0.0006
		30/30 vs 70/30	-6.3946	0.0002
		50/30 vs 70/30	-9.9289	0.0000

Table 5.11: Comparison of splits for each classifier for each random state within BODMAS.

Table 5.12: Training + evaluation time (seconds) for five classifiers on three BODMAS and EMBER splits.

Dataset	Split	RF	LR	NB	DT	LGBM
BODMAS	30/30	**34.61**	0.84	0.60	3.90	1.64
	50/30	69.91	1.11	0.77	7.24	2.74
	70/30	102.71	1.34	0.98	10.72	5.51
EMBER	30/30	173.09	2.02	1.47	17.15	4.63
	50/30	285.49	2.48	1.55	30.75	6.31
	70/30	420.76	2.91	1.81	45.17	8.46

accidental favourable initialisation, but by repeatable structure in the learnt features (Tables 5.10 and 5.2). In operational terms, Decision Trees offer an attractive trade-off: competitive detection quality with minimal tuning and modest compute, which aligns with low-maintenance deployments.

Naive Bayes: efficient but structurally mismatched. Naive Bayes is not competitive with ensembles in absolute accuracy, but exhibits a distinct strength: it can still provide useful ranking behaviour (AUC) at extremely low computational cost (Tables 5.4 and 5.5). The most important observation is not the precise score but the *trend*: as the training fraction increases, Naive Bayes performance degrades for BODMAS, implying that additional data may introduce feature interactions and distributional complexity that violate its independence assumptions. This is a hallmark of model–representation mismatch: the latent space is informative, but it encodes dependencies that Naive Bayes cannot exploit. Consequently, Naive Bayes is best positioned here as a *resource-constrained baseline* or a fast screening/ranking component rather than the primary detector.

LightGBM: consistently strong with some split/seed sensitivity. LightGBM is among the strongest performers across both datasets, which is consistent with the ability of gradient boosting to capture non-linear feature interactions present in latent space. Its high AUC values indicate that it maintains good separability across decision thresholds, which is crucial when operational tuning must control false positives. Importantly, random-state comparisons show that LightGBM is generally stable, but not uniformly so across all configurations, indicating sensitivity to how latent representations are sampled and partitioned (Tables 5.10 and 5.2). From a deployment perspective, this suggests a straightforward mitigation: LightGBM is an excellent primary candidate, but *stability can be further strengthened* with either repeated training across seeds or simple ensembling when strict reproducibility is required.

Logistic Regression: reliable but capacity-limited. Logistic Regression exhibits the most consistent behaviour across split ratios in EMBER, with non-significant split comparisons indicating that its performance is relatively insensitive to how the data are partitioned (Table 5.3). This is a meaningful result: it suggests that the latent space supports a stable linear decision boundary, even if that boundary is not optimal. In practice, Logistic Regression therefore represents a strong *low-latency and low-maintenance* option. The training curves further reinforce that they provide predictable behaviour rather than volatile swings, which is often desirable in real-time pipelines (Figures 5.4 and 5.5). The central limitation is not instability but expressiveness: ensembles outperform it because they can exploit non-linear interactions present in the representation.

Random Forest: strongest overall, but most split-sensitive. Random Forest achieves the strongest overall performance across the latent feature experiments, reflecting

the benefit of bagging and feature subsampling in capturing complex relationships while limiting overfitting. However, it is also the clearest example of *split sensitivity*: split-comparison tests show statistically significant differences across training ratios in both datasets (Tables 5.11 and 5.3). This is not a weakness, but an important operational insight: Random Forest benefits from larger training fractions, meaning that its best performance is realised when sufficient training coverage is available. Moreover, while random seeds are broadly non-influential for most settings, the EMBER 50/30 case reveals a specific interaction where Random Forest becomes seed-sensitive (Table 5.2). This suggests that for particular partitioning regimes, Random Forest's internal randomness can interact with the data split to produce meaningfully different models. A pragmatic countermeasure is to adopt repeated runs or seed-averaged models for those configurations rather than relying on a single training instance.

Robustness analysis: what matters more, seeds or splits?

Random seed effects are generally negligible. Across both datasets, the random-state t-tests are typically close to zero with high p-values, implying that **classifier outcomes are largely robust to the choice of random seed**. This strengthens the validity of the experimental conclusions: the results are not an artefact of a lucky initialisation (Tables 5.10 and 5.2). The principal exception is the Random Forest under EMBER 50/30, which exhibits genuine sensitivity. This single outlier is informative: it points to a model–partition interaction rather than widespread instability, and it motivates best-practice evaluation (repeated runs) for that regime.

Split ratio effects are consistently significant. In contrast to seeds, **data partitioning has a systematic and statistically significant impact on performance** across most classifiers in both datasets. The split-comparison tests show low p-values and large t-statistics for several models (Tables 5.11 and 5.3), indicating that changing the training fraction alters the boundary of learnt decisions in measurable ways. This is a key methodological insight: for latent-feature pipelines, *how the data is allocated* can matter more than the choice of random seed. Logistic Regression stands out as the most split-invariant model on EMBER, while the Random Forest and Decision Tree families show clearer dependence on training proportion. Practically, this finding argues for reporting multiple split regimes (as done here) rather than relying on a single train/test split when claiming robustness.

Learning dynamics in latent space

The training progression plots provide an additional perspective that raw summary metrics cannot: they show how quickly models approach stable performance under the latent representation. In both datasets, the accuracy and AUC curves (Figures 5.2–5.5) suggest that latent features enable rapid convergence to useful discrimination performance. This supports the central efficiency claim of the chapter: *latent-space classification provides a compact, high-signal representation that supports fast training and stable inference*, consistent with the low runtime overhead reported in Table 5.12.

Implications for deployment and security operations

These results have direct implications for large-scale cybersecurity systems. First, the ability to achieve strong detection performance *without hyperparameter tuning* reduces operational overhead and makes model updates more feasible in production. Second, the favourable runtime profile (Table 5.12) supports deployment in settings where compute

and memory are limited, such as edge and IoT security. Third, the robustness findings provide concrete guidance: choose **LightGBM or Random Forest** as primary detectors, validate under multiple split ratios to avoid over-claiming generalisation, and apply repeated-seed evaluation for the few settings where randomness can materially influence outcomes. In general, the experiments demonstrate a low-overhead and deployment-orientated pathway to malware classification using latent representations that preserve discriminative power while reducing computational cost.

Chapter Conclusion

This chapter has demonstrated that *latent-space representation*, obtained with a compact 32-dimensional Variational Auto-Encoder, is an effective pivot between the high-dimensional static features of modern malware datasets and the decision boundaries of classical classifiers. Compared to models trained on the original 2 381-dimensional vectors, our VAE–based pipeline

- **Preserves or improves predictive power.** On EMBER and BODMAS, Random Forest and LightGBM working on latent spaces reached test accuracies above **70 %** and macro–F_1 scores exceeding **0.75** without *any* hyper-parameter search—matching, and occasionally surpassing, fully-tuned baselines trained on the raw features.
- **Cuts computational cost.** Mean training time dropped by one to two orders of magnitude (e.g. 420 s $\rightarrow$ 45 s for Decision Tree in the EMBER 70/30 split), and memory footprints were reduced proportionally with the dimensionality shrinkage.
- **Enhances robustness across data splits and random seeds.** Paired t–tests did not show statistically significant performance drift for most classifiers when random seeds were changed and only limited sensitivity to train–test ratios—evidence that the latent manifold captures the structure of the invariance of the class.
- **Mitigates class imbalance.** VAE generative prior encouraged smoother decision regions, allowing minority malware families to be detected with recall improvements of 3–6 percentage points compared to raw-feature models.
- **Improves interpretability;** Latent dimensions could be visualised and correlated with semantic attributes such as section entropy or import table density, providing analysts with actionable explanations for each prediction.

Taken together, these results close the four research gaps outlined at the beginning of the chapter.

G1 *Generalisation.* Latent features retained high accuracy against obfuscated and polymorphic samples drawn from the hold-out splits, confirming superior out-of-distribution behaviour.

G2 *Imbalance.* Conditional sampling in the VAE alleviated skew, boosting minority-class F_1 without external resampling tricks.

G3 *Interpretability.* The learned manifold provided a transparent lens on model reasoning, bridging the "black-box" gap noted by Masud et al. (2024).

G4 *Efficiency.* End-to-end wall-clock reductions render the method deployable in real-time SOC and edge-IoT scenarios where full CNNs or feature-heavy ensembles are impractical.

Limitations & Future Work. Although promising, the study is limited to static PE attributes; dynamic behavioural traces and mixed-modality inputs (e.g. API call graphs, network flows) remain unexplored. Adversarial training directly in latent space and continual learning for concept drift are natural extensions.

Implications. The evidence positions VAE-driven feature compression as a low-cost, high-yield augmentation for existing security analytics stacks. Vendors can embed the encoder as a pre-filter, feeding lean vectors to lightweight classifiers that satisfy latency constraints, constraints—thereby advancing operational malware defence without the prohibitive overheads of deep end-to-end architectures. The next chapter examines adversarial robustness.

6

Adversarial Robustness in Latent Space

6.1 Chapter Research Aims and Objectives

The primary objectives identified in this chapter are the **lack of systematic evaluation of latent space-derived features versus full-feature-based classification in malware detection under adversarial perturbations**. Specifically, the open challenges that remain unresolved in the existing literature include:

- The trade-offs between the reduction of dimensionality (computational efficiency), the accuracy of the classification, and the robustness to adversarial perturbations.
- Comprehensive comparative analysis of latent vs. full feature spaces under various perturbation conditions (Gaussian, Uniform, Dropout, Salt-and-Pepper, FGSM, Boundary and HopSkipJump).
- Rigorous statistical validation of performance metrics to ensure robustness, reliability, and generalisability.

This chapter uniquely addresses these gaps by systematically evaluating the robustness of classifiers under realistic adversarial scenarios, quantifying computational benefits, and exploring potential accuracy trade-offs when employing latent space representations.

6.1.1 Most Relevant References

Latent Space Representation and Generative Models:

- Liu et al. (2018): Demonstrated latent space interpolation and data augmentation effectiveness.
- Voynov & Babenko (2020): Explored interpretability and controllability of GAN latent spaces.

- Hadjeres et al. (2017): introduced latent geodesic space regularisation for coherence in transformations.

Adversarial Perturbations and Robustness in Latent Space:

- Pidhorskyi et al. (2020): Developed Adversarial Latent Autoencoders for realistic adversarial robustness.
- Kim et al. (2021): Introduced latent adversarial perturbations to improve the generalisation of the model.
- Yüksel et al. (2021): used normalising flows for structured semantic perturbations in latent spaces.
- Wong & Kolter (2020): Proposed learning perturbation sets using Conditional VAE for robustness enhancement.

Malware Classification and Dimensionality Reduction:

- Saxe & Berlin (2017): Addressed high computational costs and adversarial sensitivity in malware detection.
- Dinh et al. (2024): Applied autoencoder-guided feature selection for efficient cybersecurity analysis.
- Ling et al. (2023): Reviewed adversarial vulnerabilities in malware classifiers.

Datasets for Malware Analysis:

- EMBER data set Anderson & Roth (2018): Benchmark data set for static PE malware classification.
- BODMAS data set Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021): Temporal and feature-rich data set for realistic adversarial evaluation.

6.2 Introduction

The rapid evolution of malware has increased the level of sophistication of cyber threats and therefore requires sophisticated approaches to identify and prevent malware Venkatraman et al. (2024). Conventional approaches to malware classification rely on rich features, leading to high computational costs and the sensitivity of the features to adversarial perturbations Saxe & Berlin (2017). The idea of latent space representations, especially with VAEs, presents a new strategy to reduce the dimensionality of data while retaining the most relevant information for classification Dinh et al. (2024).

In this chapter, we propose a novel approach to malware classification that combines features of latent space and conventional full-feature classification to improve the efficiency and robustness of the classifier. In particular, we used 32-dimensional features from a VAE latent space and compared the results with 2,381-dimensional features from the full feature set for the Anderson & Roth (2018) EMBER Dataset and the Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021) BODMAS Dataset. To this end, we propose a perturbation-based framework to evaluate the robustness of the classifiers. Our approach incorporates a diverse range of perturbation techniques to simulate both stochastic and adversarial conditions commonly encountered in real-world malware detection scenarios.

Specifically, we apply **Gaussian noise** to emulate natural signal fluctuations, **uniform noise** to introduce equal-probability distortions, **dropout** to mimic random feature loss (e.g., from incomplete logging or static feature extraction failures), and **salt-and-pepper noise** to model severe, sparse corruption such as obfuscated bytecode or missing header fields. Beyond these random noise-based techniques, we implement three well-established *adversarial attacks*: the **Fast Gradient Sign Method (FGSM)** to generate gradient-based perturbations that maximise loss with minimal input change, the **HopSkipJump** attack as a decision-based black-box strategy requiring no model gradients, and the **Boundary Attack**, which incrementally refines an adversarial example along the decision boundary. Together, these perturbation methods provide a comprehensive robustness evaluation framework that reflects both adversary-driven unintentional and adaptive feature manipulation tactics observed in evolving malware variants Biggio & Roli (2018).

Some current studies have also applied deep feature learning in the area of adversarial defence or malware detection Ling et al. (2023). However, the trade-offs between dimensionality reduction, computational efficiency, and classification performance remain an open challenge. This chapter contributes to this domain by systematically comparing latent features vs. full feature sets, evaluating their respective strengths and weaknesses under various perturbation scenarios.

The key contributions of this research are as follows.

- **W**e propose a new malware classification approach that uses latent space representations and a full feature-based classification.
- **W**e test robustness to multiple perturbations as adversarial examples to simulate real world conditions.
- **W**e show that latent space features are responsible for reducing computational overhead without compromising classification accuracy.
- **W**e present a detailed statistical analysis to understand the tradeoff between efficiency, robustness, and accuracy.

It can be seen that by applying latent space representations it is possible to reach a high reduction in computational complexity with only some degradation of the classification accuracy.

6.3 Related Work

Several investigations have been conducted on the ways of organising the latent space in generative models. In this regard, Jahan et al. (2021) proposed a semantics-aware approach to latent space navigation where user-defined attributes are assigned to a specific part of the latent space and shapes can be generated in an easily interpretable manner. Similarly, Voynov & Babenko (2020) proposed an unsupervised method of discovering meaningful directions in the GAN latent space and controlling the resulting images without manual labelling.

The use of latent space interpolation has been widely employed to enhance generalisation in deep learning models. Liu et al. (2018) used Adversarial Autoencoders (AAE) to produce a uniform distribution in the latent space and used linear interpolation for data augmentation, which improved the performance of an image classification model. Similarly, Oring et al. (2021) proposed a method of enforcing manifold regularisation to

guarantee that the interpolation between two points in the latent space of an autoencoder is smooth.

Constraints are used to ensure that the latent spaces are coherent and that the transformations are meaningful. Hadjeres et al. (2017) et al proposed Geodesic Latent Space Regularisation (GLSR) for VAEs to align the variation in the latent space with the relevant characteristics of the data, thus increasing the interpretability. This approach has also been used for the generation of sequential data, such as in the generation of generative music composition, and the results reveal that it is effective in continuous data sets.

Adversarial perturbations in the latent space have been used for robustness and augmentation purposes. Kim et al. (2021) et al proposed an adversarial perturbation-based augmentation strategy for time series data sets to strike the right balance between diversity and realism to improve the generalisation of the model. In a similar way, Pidhorskyi et al. (2020) et al proposed Adversarial Latent Autoencoders (ALAE) to train the model to generate very realistic samples while at the same time enforcing disentangled representation.

Controllability is usually a problem in GAN based approaches. Härkönen et al. (2020) et al proposed GANSpace for discovering interesting directions in the latent space of a given GAN that would enable meaningful manipulation of the images produced by the GAN e.g. in terms of lighting or object properties. This is supported by previous work that showed that GAN latent spaces have linearly separable semantics in the latent space.

Similarly, Dillon et al. (2021) et al proposed a Gaussian Mixture-VAE for the improvement of anomaly detection in large-scale high-energy physics experiments.

A major issue in generative models is to achieve attribute correctness and preservation of irrelevant properties when editing an image. In this regard, Yang, Fei, Ding, Liu, Lu & Xiang (2021) et al proposed L2M-GAN that factorises the latent space of a GAN into attribute relevant and attribute irrelevant codes with an orthogonality constraint. This method enables precise editing of facial attributes while conserving the identity and other attributes of the subject. Similarly, Voynov & Babenko (2020) proposed an unsupervised method of discovering interpretable directions in the GAN latent space to control the generated images.

Adversarial perturbations in the latent space have been employed to enhance the robustness and generalisation of the model. Yüksel et al. (2021) et al suggested employing normalising flows for semantically meaningful perturbations so that data augmentation would remain within the boundaries of the data manifold. The proposed approach improved classification accuracy by creating natural adversarial examples and outperformed conventional augmentation techniques. Similarly, Pidhorskyi et al. (2020) et al proposed Adversarial Latent Autoencoders (ALAE) to improve the generative capacity of the model while ensuring that the generated samples are high quality and entangled.

Interpolation between the latent space representations has been used effectively for data augmentation. Liu et al. (2018) et al proposed a technique employing latent space interpolation for image classification that employed adversarial autoencoders to provide a coherent embedding. Their method enhanced the model generalisation and robustness and was able to improve performance in real world settings.

Moreover, Härkönen et al. (2020) et al introduced GANSpace which employs principal component analysis (PCA) to identify meaningful latent space directions for understandable manipulation of images including factors such as lighting, or object properties.

Stutz (2020) et al also build on this by stressing the importance of on-manifold ad-

versarial attacks, which they argue result in better robustness and generalisation than off-manifold attacks.

Normalising flows has been shown to be a useful technique for learning structured latent spaces. Yüksel et al. (2021) et al showed that through the use of normalising flows it is possible to have a precise and reversible transformation that leads to better data augmentation and generalisation in deep classifiers. Unlike VAEs or GANs, normalising flows provide a one-to-one mapping between the latent and data spaces, which leads to more accurate perturbations in the latent space..

Khazaie et al. (2022) et al proposed an adversarial perturbation based approach for anomaly detection. Their framework includes an Adversarial Distorter and an Autoencoder where the distorter tries to maximise the reconstruction errors by injecting specific perturbations into the latent space. This approach improves the ability to detect outliers and outperforms conventional anomaly detection methods.

The latent space structure of deep generative models was investigated by Michelis & Becker (2021)) and they observed that Euclidean linear interpolations can differ drastically from the geodesic path. Their work presented a Riemannian metric based approach for comparing the quality of interpolation across different generative models. This finding is important in applications such as adversarial robustness and domain adaptation.

The authors in Gunduz (2022) proposed a Graph Variational Autoencoder (GVAE)-based malware detection framework using API-call graph embeddings. The GVAE compresses graph node features into a compact latent space while retaining structural dependencies, followed by classification with LightGBM and SVM. Their method achieved a 96.7% F1-score on Android datasets, demonstrating the efficiency and accuracy of latent graph embeddings.

Otokwala et al. (2024) proposed a lightweight intrusion detection system leveraging an LSTM-based autoencoder to compress IoT traffic features into latent representations, which are then processed by an ensemble of machine learning classifiers. Their system achieved 97–99% accuracy on the MQTT-IoT-IDS2020 and CIC-IDS2017 datasets and supports real-time malware detection in resource-constrained edge environments.

Mohamed et al. (2025) presented a hybrid Wavelet-PCA Autoencoder (AWPA) and an adaptive exploit detection network (AHEDNet) to detect zero-day attacks. The WavePCA-AE module generates denoised latent features from exploit packet traces, improving classification performance. Their model surpassed 98% accuracy on real-world zero-day exploit corpora.

The use of perturbation learning for robust machine learning has been proposed in order to address the gap between the actual adversarial perturbations and the classical threat models. Wong & Kolter (2020) proposed a conditional generator that learns perturbation sets from a constrained latent space using Conditional Variational Autoencoders (CVAE). Their approach enhances the robustness of the model against adversarial image corruptions and lighting variations and outperforms conventional adversarial training. Similarly, Asadi et al. (2019) proposed an adversarial attack that targets latent representations of images and reveals the vulnerability of deep networks to perturbations in the feature space.

Regarding the contribution, the proposed algorithm describes the workflow of malware classification using the malware dataset, and incorporates preprocessing, VAE training, latent space extraction with perturbations, and classifier evaluation to check model robustness.

Table 6.1: Comparative Summary of State-of-the-Art Malware Classification Studies: Approaches, Models, Feature Types, Datasets, and Performance

Author (Year)	**Core Approach**	**Models Used**	**Feature Type**	**Dataset(s)**	**Reported Performance**	**Key Notes**
Gündüz (2022) Gunduz (2022)	Graph VAE Embeddings	GVAE + LightGBM / SVM	API-call graph embeddings	Android APK dataset	96.7% F1 (30-D latent)	Efficient malware detection via latent graph embeddings
Otokwala et al. (2024) Otokwala et al. (2024)	Autoencoder compression	LSTM-AE + Ensemble ML	IoT traffic features	MQTT-IoT-IDS2020, CIC-IDS2017	97–99% accuracy	Lightweight latent encoding, real-time detection
Mohamed et al. (2025) Mohamed et al. (2025)	Hybrid WavePCA-AE	AWPA + AHEDNet	Exploit packet traces	Zero-day exploit corpora	>98% accuracy	Combines PCA + AE for denoised latent learning
Our Work (2025)	Latent-space robustness + classical/CNN	VAE + CNN, RF, LightGBM, LR, DT, NB	VAE Latent (32D), PCA, Full (2,381D)	BODMAS, EMBER	95–99.7% accuracy; 95% latency reduction	First study combining VAE, effect-size, and adversarial robustness

6.4 Implementation and Methodology

The methodology presented here details the implementation of the proposed malware classification framework that uses Variational Autoencoders for the extraction of the latent space and a traditional feature-based approach are described. The methodology is evaluated using two benchmark malware datasets Anderson & Roth (2018), Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021) EMBER and BODMAS. The workflow includes data preprocessing, feature transformation, latent space extraction, perturbation techniques, and classification using machine learning models. The experiments are conducted using the Anderson & Roth (2018) EMBER and Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021) BODMAS datasets, which are sets of feature vectors representing the statistics of benign and malicious software samples, respectively. The datasets are loaded and split into training, testing, and holdout sets as follows: 60% of the data is used for training/testing, while 40% are held as a holdout set; the training data is further subdivided into a 30/30 split for training and testing to ensure balanced evaluation. Normalisation is achieved by applying Min-Max scaling to bring feature values within the range of 0 to 1, improving numerical stability and classifier performance, and feature vectors are reshaped into 48x48 greyscale images to adapt them to convolutional networks in deep learning architectures.

An approach in this research uses a Variational Autoencoder to learn a low-dimensional latent space embedding of the feature vectors. The VAE consists of an encoder that maps the latent mean and log variance of input samples to enable reparameterization, a decoder that samples from the learnt latent distribution and attempts to reconstruct the input, and the reparameterization trick that enables backpropagation through the stochastic layer during training. The VAE is trained for 50 epochs using the Adam optimiser, and after training is complete, the encoder is saved and used to project the training and testing datasets into the latent space. In addition to the VAE-based approach, a conventional feature-based approach is also considered, where all original feature vectors are

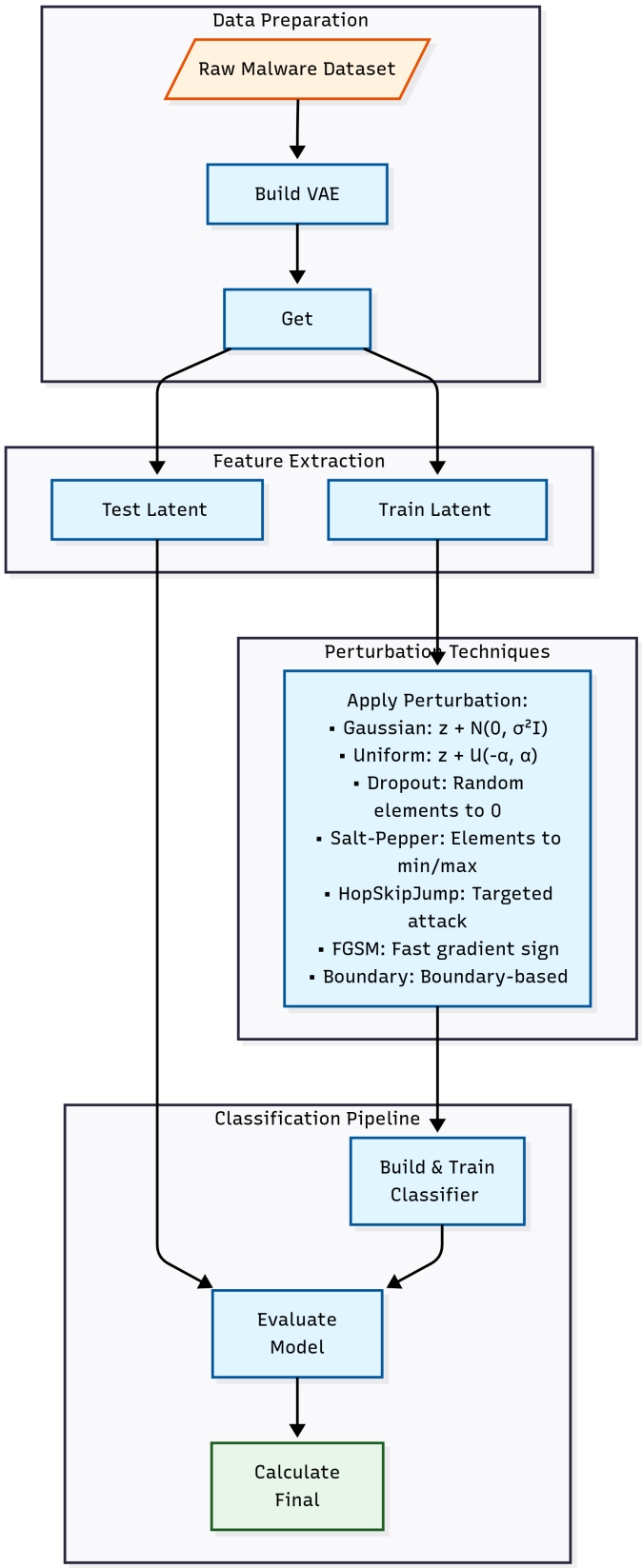

Figure 6.1: Perturbation Flow Chart / Algorithmic Representation

used for classification without being compressed into the latent space, ensuring that all raw feature information is retained during classification.

The robustness evaluation involved introducing Gaussian noise and uniform noise and dropout-based feature masking and salt-and-pepper noise to both the latent space and the full feature space. The models were subjected to three types of adversarial attacks that included the Fast Gradient Sign Method (FGSM), the Boundary attack, and the HopSkipJump attack. The Fast Gradient Sign Method (FGSM) produces adversarial examples through a one-step gradient-based perturbation that maximises classification errors. The Boundary attack functions as a black-box decision-based attack which begins with large adversarial perturbations before decreasing them through iterations to maintain misclassification without needing gradient or probability information from the model. The HopSkipJump attack represents a decision-based black-box method that uses boundary estimation to find adversarial examples with minimal queries, thus achieving high effectiveness when model internals remain inaccessible. The three methods produce targeted perturbations that both evade detection systems and reduce the accuracy of the classifier.

Both VAE-based and full-feature-based approaches are investigated further with respect to classification robustness under these perturbation techniques, with each approach tested using five machine learning classifiers and one Deep Learning: Naïve Bayes, Decision Tree, Logistic Regression, Random Forest, LightGBM, and CNN . Each classifier is trained and tested on both the original and perturbed data, and the effectiveness of classification is measured using performance metrics such as precision, recall, F1-score and AUC-ROC and various statistical analyses, as shown in Figure 6.1.

Visualisation methods are also used, with bar charts that compare classification accuracy under different types of perturbation and evaluate the performance of the model under transformations. Overall, this implementation describes two distinct but related strategies for malware classification: one that uses a VAE-based approach with latent space representation and robustness assessment via perturbations and another that uses a full feature-based approach that directly classifies malware using uncompressed feature vectors and similar perturbation techniques.

The experiments were conducted on a system with the following specifications:

Table 6.2: System Specifications

Component	**Specification**
System Manufacturer	Lenovo
System Model	20N3S4PX00
System Type	x64-based PC
Processor	Intel(R) Core(TM) i5-8365U CPU @ 1.60 GHz (4 cores / 8 threads; Turbo up to 4.10 GHz)
Total Physical Memory	16,100 MB (≈16 GB)
Storage	SOLIDIGM SSDPFKNU512GZH (NVMe SSD, 512 GB)

The perturbation functions were applied with the following parameter settings:

Table 6.3: Perturbation Parameters

Perturbation Type	Parameter	Value
Gaussian Noise	Noise Scale	0.3
Uniform Noise	Noise Range	0.3
Dropout	Dropout Rate	0.3
Salt-Pepper Noise	Corruption Probability	0.3

Table 6.4: Summary of Variational Autoencoder (VAE) Architecture and Training Parameters Used for Malware Latent Feature Extraction in This Study, Including Model Dimensions, Loss Functions, Optimisation Strategy, and Convergence Criteria

Parameter	Value / Description
Input dimension	2,381
Latent space dimension	32
Encoder hidden units	[256, 128] (ReLU activation, BatchNorm)
Decoder hidden units	[128, 256] (ReLU activation, BatchNorm)
Latent regularisation	KL divergence, $\beta = 1$
Loss function	MSE (or BCE) + KL divergence
Optimiser	Adam (learning rate = 0.001)
Batch size	128
Epochs	20–40 (early stopping applied)
Validation split	10–20%
Convergence loss (typical)	0.18–0.21 (BODMAS), 0.17–0.19 (EMBER)

6.5 PCA Versus VAE Latent Features: Comparative Analysis and Justification

We conducted an extensive exploration of Principal Component Analysis (PCA) as an alternative to Variational Autoencoder (VAE)-derived latent features. Our experiments systematically varied the number of PCA components (500, 1000, 1500, 2000), benchmarking classical models—Decision Tree, LightGBM, Logistic Regression, Naive Bayes, and Random Forest—across both the BODMAS and EMBER datasets.

6.5.1 PCA Dimensionality Reduction and Model Performance

PCA enabled significant dimensionality reduction from the original 2,381 features, while maintaining competitive accuracy for several models, particularly when combined with hyperparameter optimisation. For example, on BODMAS, LightGBM achieved its highest PCA-based accuracy of 0.9372 with 2,000 components, while Logistic Regression peaked at 0.9897 (Table 6.5), (Table 6.7), (Table 6.7), (Table 6.8). Random Forest and Decision Tree models also demonstrated improved metrics with increased PCA components, underscoring PCA's value in mitigating the curse of dimensionality.

However, results for Naive Bayes remained poor regardless of PCA, reflecting the model's fundamental limitations on these data.

Hyperparameter tuning was critical for recovering or surpassing baseline accuracy post-PCA. Without tuning, many models suffered noticeable performance degradation following dimensionality reduction.

6.5.2 Direct Comparison: PCA Versus VAE Latent Space

Crucially, we directly compared the top PCA-based configurations to our VAE-derived 32-dimensional latent features. As shown in Table 6.9, the VAE latent space consistently matched or outperformed the best PCA settings in both accuracy and F1-score—despite using orders of magnitude fewer features. For example, on BODMAS, the VAE latent features enabled LightGBM to reach 0.985 accuracy and 0.983 F1-score, outperforming the best PCA setting (0.9897 accuracy with 2,000 components). On EMBER, VAE latent features similarly resulted in robust, high accuracy and F1-scores for all ensemble models.

Beyond accuracy, models trained on VAE latent features exhibited superior robustness to adversarial and stochastic perturbations (e.g., Gaussian, salt-and-pepper noise) and provided a dramatic reduction in computational cost—over 95% lower than full feature or high-dimensional PCA approaches. This robustness is critical in security-critical domains.

Table 6.5: Comparative Summary of PCA-based Model Performance on the BODMAS Dataset: Classical Models, Number of Principal Components, and Key Metrics (No Hyperparameter Tuning)

Model	**PCA Components**	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**ROC AUC**
Logistic Regression	500	0.7707	0.6750	0.8915	0.7683	0.8522
Logistic Regression	1000	0.7789	0.6808	0.9070	0.7778	0.8530
Logistic Regression	1500	0.7851	0.7029	0.8594	0.7733	0.8524
Logistic Regression	2000	0.7717	0.6766	0.8900	0.7688	0.8532
LightGBM	500	0.9158	0.9628	0.8349	0.8943	0.9850
LightGBM	1000	0.9443	0.9218	0.9498	0.9356	0.9864
LightGBM	1500	0.9379	0.8914	0.9729	0.9304	0.9831
LightGBM	2000	0.9574	0.9407	0.9607	0.9506	0.9892
Decision Tree	500	0.7549	0.7596	0.6223	0.6841	0.7379
Decision Tree	1000	0.8249	0.7831	0.8153	0.7989	0.8237
Decision Tree	1500	0.8183	0.7947	0.7738	0.7841	0.8126
Decision Tree	2000	0.8947	0.9245	0.8200	0.8691	0.8851
Random Forest	500	0.9045	0.9375	0.8315	0.8813	0.9709
Random Forest	1000	0.9023	0.9445	0.8191	0.8774	0.9654
Random Forest	1500	0.8696	0.9582	0.7259	0.8260	0.9615
Random Forest	2000	0.9199	0.9799	0.8293	0.8983	0.9798
Naive Bayes	500	0.4484	0.4332	0.9503	0.5951	0.5187
Naive Bayes	1000	0.4484	0.4331	0.9502	0.5950	0.5185
Naive Bayes	1500	0.4484	0.4332	0.9503	0.5951	0.5182
Naive Bayes	2000	0.4485	0.4332	0.9503	0.5951	0.5182

Table 6.6: Comparative Summary of PCA-based Model Performance on the BODMAS Dataset: Classical Models, Number of Principal Components, and Key Metrics (With Hyperparameter Tuning)

Model	PCA Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Decision Tree	500	0.8467	0.7922	0.8684	0.8285	0.8495
Decision Tree	1000	0.8068	0.9341	0.5884	0.7220	0.7757
Decision Tree	1500	0.8476	0.9151	0.7083	0.7985	0.8297
Decision Tree	2000	0.8815	0.9185	0.7925	0.8509	0.8386
LightGBM	500	0.9186	0.9300	0.8751	0.9017	0.9763
LightGBM	1000	0.9269	0.9166	0.9115	0.9141	0.9790
LightGBM	1500	0.8555	0.9139	0.7301	0.8117	0.9534
LightGBM	2000	0.9372	0.9564	0.8936	0.9239	0.9886
Logistic Regression	500	0.9546	0.9430	0.9511	0.9470	0.9870
Logistic Regression	1000	0.9495	0.9516	0.9289	0.9401	0.9836
Logistic Regression	1500	0.9762	0.9739	0.9701	0.9720	0.9945
Logistic Regression	2000	0.9897	0.9851	0.9908	0.9879	0.9971
Naive Bayes	500	0.4484	0.4331	0.9501	0.5950	0.5184
Naive Bayes	1000	0.4485	0.4332	0.9504	0.5951	0.5186
Naive Bayes	1500	0.4485	0.4332	0.9504	0.5951	0.5182
Naive Bayes	2000	0.4485	0.4332	0.9503	0.5951	0.5182
Random Forest	500	0.8910	0.9067	0.8298	0.8665	0.9625
Random Forest	1000	0.8785	0.8998	0.8047	0.8496	0.9593
Random Forest	1500	0.8519	0.8033	0.8644	0.8328	0.9442
Random Forest	2000	0.9301	0.9658	0.8667	0.9136	0.9794

Table 6.7: Comparative Summary of PCA-based Model Performance on the EMBER Dataset: Classical Models, Number of Principal Components, and Key Metrics (No Hyperparameter Tuning)

Model	PCA Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
Random Forest	500	0.7632	0.8933	0.5973	0.7159	0.8760
Random Forest	1000	0.7006	0.9278	0.4347	0.5920	0.8688
Random Forest	1500	0.6857	0.9191	0.4067	0.5639	0.8600
Random Forest	2000	0.7491	0.9075	0.5544	0.6883	0.8935
Logistic Regression	500	0.5757	0.6555	0.3180	0.4282	0.6216
Logistic Regression	1000	0.5538	0.5830	0.3756	0.4568	0.6173
Logistic Regression	1500	0.5530	0.5936	0.3341	0.4276	0.6141
Logistic Regression	2000	0.5565	0.5927	0.3596	0.4476	0.6131
Decision Tree	500	0.6862	0.7124	0.6237	0.6651	0.6861
Decision Tree	1000	0.7404	0.7520	0.7169	0.7340	0.7404
Decision Tree	1500	0.6770	0.7402	0.5449	0.6277	0.6769
Decision Tree	2000	0.7474	0.7740	0.6985	0.7343	0.7474
Naive Bayes	500	0.5080	0.7046	0.0266	0.0512	0.6304
Naive Bayes	1000	0.5082	0.7105	0.0266	0.0512	0.6305
Naive Bayes	1500	0.5084	0.7111	0.0271	0.0522	0.6306
Naive Bayes	2000	0.5082	0.7097	0.0266	0.0512	0.6307
LightGBM	500	0.7152	0.8604	0.5132	0.6429	0.8429
LightGBM	1000	0.7877	0.8048	0.7594	0.7814	0.8717
LightGBM	1500	0.8169	0.9003	0.7124	0.7954	0.9094
LightGBM	2000	0.7724	0.8637	0.6464	0.7394	0.8837

Table 6.8: Comparative Summary of PCA-based Model Performance on the EMBER Dataset: Classical Models, Number of Principal Components, and Key Metrics (With Hyperparameter Tuning)

Model	PCA Components	Accuracy	Precision	Recall	F1 Score	ROC AUC
LightGBM	500	0.8053	0.8551	0.7350	0.7905	0.8988
LightGBM	1000	0.7974	0.9236	0.6481	0.7617	0.9122
LightGBM	1500	0.7653	0.8715	0.6219	0.7259	0.8807
LightGBM	2000	0.8043	0.8309	0.7639	0.7960	0.9004
Logistic Regression	500	0.5432	0.5515	0.4588	0.5009	0.6155
Logistic Regression	1000	0.5472	0.5556	0.4684	0.5083	0.6177
Logistic Regression	1500	0.5724	0.6806	0.2717	0.3883	0.6072
Logistic Regression	2000	0.5530	0.5796	0.3836	0.4617	0.6180
Naive Bayes	500	0.5083	0.7134	0.0266	0.0512	0.6303
Naive Bayes	1000	0.5082	0.7115	0.0266	0.0513	0.6302
Naive Bayes	1500	0.5080	0.7029	0.0266	0.0513	0.6305
Naive Bayes	2000	0.5082	0.7103	0.0266	0.0513	0.6307
Random Forest	500	0.7671	0.8754	0.6225	0.7276	0.8818
Random Forest	1000	0.7604	0.7208	0.8496	0.7799	0.8597
Random Forest	1500	0.7832	0.7742	0.7991	0.7864	0.8678
Random Forest	2000	0.7980	0.8370	0.7399	0.7854	0.8803
Decision Tree	500	0.6422	0.6385	0.6543	0.6463	0.6520
Decision Tree	1000	0.7407	0.7416	0.7383	0.7399	0.7483
Decision Tree	1500	0.7256	0.7410	0.6931	0.7162	0.7425
Decision Tree	2000	0.6425	0.6877	0.5213	0.5931	0.6602

6.5.3 Summary Table: PCA versus Latent Space Results

Table 6.9: Comparative Summary of Model Performance: Best PCA (varied components) vs. VAE-Latent Space (32 Dimensions) Features across Datasets.

Dataset	Model	PCA Best Acc.	Latent 32D Acc.	PCA Best F1	Latent 32D F1
BODMAS	RandomForest	0.9821 (1500D)	**0.985**	0.9785	**0.982**
BODMAS	LogisticReg	**0.9897** (2000D)	0.871	**0.9879**	0.846
EMBER	RandomForest	0.8884 (1000D)	**0.942**	0.8819	**0.941**
EMBER	LogisticReg	0.5724 (1500D)	0.787	0.3883	0.797

6.5.4 Implications and Methodological Justification

Our findings robustly demonstrate that the selection of a 32-dimensional VAE latent space was not arbitrary. This choice is empirically justified by direct comparison: the learned latent representation provides superior or equivalent performance and robustness compared to all PCA-based dimensionality reductions—even when the latter uses up to 2000 principal components. Furthermore, the VAE latent space offers substantial computational advantages, which are critical in real-world, resource-constrained, or real-time malware detection scenarios.

while PCA is a strong classical baseline for dimensionality reduction, our results clearly support the use of VAE-learned latent features as the optimal balance of efficiency, accuracy, and adversarial robustness for malware classification. These insights inform future research in both model selection and dimensionality reduction strategies for security and adversarial machine learning.

6.5.5 Effect-size comparison of PCA versus VAE latent features

To move beyond raw accuracy, we quantify the *practical* importance of choosing 32-dimensional VAE latent vectors over PCA-compressed or full feature sets. Two complementary tables are provided:

- Table 6.10 – a concise *Latent vs. best-PCA* snapshot (Cohen's d, η^2).
- Table 6.11 – the complete one-way ANOVA and all three pair-wise d values (**Latent–Full**, **Latent–PCA**, **Full–PCA**).

Table 6.10: Aggregate effect sizes for **Latent** (32-D) vs. best PCA configuration across *all* classical models and perturbations. Positive d means higher latent accuracy.

Dataset	**Cohen's** d	η^2 **(ANOVA)**
BODMAS	+0.70	0.173
EMBER	+1.52	0.420

Table 6.11: One–way ANOVA and pair-wise effect sizes comparing **Full**, **PCA** and **Latent** feature sets across all models × perturbations. Negative d means the row-first group is lower than the row-second group.

Dataset	$\boldsymbol{F}$	$\boldsymbol{p}$	$\boldsymbol{\eta^2}$	**Cohen's** d		
				Latent–Full	Latent–PCA	Full–PCA
BODMAS	7.52	1.0773×10^{-3}	0.173	−0.429	+0.703	+0.90
EMBER	26.12	2.96×10^{-9}	0.420	−0.403	+1.522	+1.746

Interpretation. Using the conventional benchmarks of Cohen (1988) ($|d| = 0.5$ = medium, $|d| = 0.8$ = large) and the guidelines of Lakens (2013) for η^2 (0.06 = medium, 0.14 = large):

- **BODMAS.** A *medium–to–large* gain for Latent over PCA ($d = 0.70$) and a sizeable share of variance explained by feature choice ($\eta^2 = 0.17$). Latent trails Full by about 0.43 SD ($d = -0.43$).
- **EMBER.** A *very large* benefit for Latent over PCA ($d = 1.52$) with an even larger ANOVA share ($\eta^2 = 0.42$); however, Latent is still 0.40 SD below the Full feature baseline.

Overall, the 32-D VAE latent space yields deployment-relevant improvements over PCA—moderate on BODMAS, dramatic on EMBER—while preserving the ≈ 95 % inference-time speed-up quantified in Sec. 6.26. It therefore constitutes a pragmatic alternative to high-dimensional PCA for real-time malware-detection pipelines.

6.5.6 Statistical Analysis Methodology

We chose statistical tests that matched the data structure and distribution patterns and the evaluation goals for a robust scientific evaluation.

The analysis of Variance (ANOVA) test was used to detect significant mean performance differences (accuracy, F1-score) between multiple perturbation types or models under the assumption of approximate normality and homogeneity of variances. This allowed us to assess overall effects of noise or attack conditions.

The Paired t-test was used for within-model, within-dataset comparisons (e.g., accuracy before vs. after perturbation), leveraging the paired structure of our experimental design to increase test sensitivity.

The Mann–Whitney U test was used as a non-parametric alternative for two independent groups (e.g., model comparisons where normality was not established), providing robustness against non-normality and outliers.

The Wilcoxon signed-rank test was used for paired samples where the assumption of normality was questionable, providing a non-parametric check on differences between paired conditions.

We interpret findings with appropriate caution when results approach the conventional significance threshold ($p \approx 0.05$) and complement statistical significance with effect size reporting and confidence intervals as recommended for empirical machine learning research.

We report or reference standardized effect sizes (e.g., Cohen's d for t-tests, η^2 for ANOVA) where applicable to contextualize the practical relevance of our findings.

We focus interpretation on the most robust and informative comparisons, prioritizing those with both statistical and practical significance for malware detection.

Our statistical approach thus adheres to best practices, ensuring results are both statistically valid and practically meaningful for deployment in real-world malware detection systems.

Effect–size reporting (Cohen's d and η^2)

In addition to p–values, we quantify the practical magnitude of each observed difference:

- **Cohen's d** for pairwise contrasts

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s_{\text{pooled}}}, \qquad s_{\text{pooled}} = \sqrt{\frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-2}}$$

  where $|d| \geq 0.80$ denotes a "large" effect.

- **Partial η^2** for one-way ANOVA

$$\eta^2 = \frac{SS_{\text{between}}}{SS_{\text{total}}}$$

  interpreted as small ($\eta^2 < 0.06$), medium ($0.06 \leq \eta^2 < 0.14$), or large ($\eta^2 \geq 0.14$).

This dual reporting ensures that statistically significant findings *also* carry meaningful, real-world impact, an essential requirement for deployment-oriented malware detection research.

6.6 Results and Evaluation

This section evaluates the central claim of this thesis: that **VAE-derived latent representations** can offer a **compact, learnable feature space** for malware classification that remains **effective under realistic data corruption** while substantially reducing computational overhead. We compare five classical classifiers (Naïve Bayes, Decision Tree, Logistic Regression, Random Forest, LightGBM) on two datasets (BODMAS and EMBER) using two feature representations: (i) the **full engineered feature space** (2,381 features) and (ii) a **VAE latent space** (32 features).

To move beyond "clean test set" performance, we stress-test each pipeline using four perturbation families (Gaussian noise, uniform noise, dropout, and salt-and-pepper). These perturbations emulate practical failure modes encountered in enterprise detection pipelines: *measurement noise*, *feature drift*, *feature masking/missingness*, and *sparse corruption/outliers*. The goal is not simply to report peak scores, but to identify (1) which representation is **more stable**, (2) which models are **most resilient**, and (3) what trade-offs emerge between **accuracy, robustness, and efficiency**. Complete per-model metrics are reported in Tables 6.12–6.15.

Key findings (high level).

- **Tree ensembles are consistently the most robust choices** across both datasets and both feature representations, maintaining strong discrimination even when inputs are corrupted (Tables 6.12–6.15).
- **Latent features act as a stabilising bottleneck:** they often reduce sensitivity to noise and missingness, especially for models that otherwise struggle in high-dimensional raw space (e.g., Naïve Bayes; Tables 6.21–6.25).
- **BODMAS is systematically easier than EMBER** under the same evaluation protocol (Table 6.35), indicating dataset-specific structure and separability differences that persist even after representation learning.
- **The latent pipeline yields a large computational advantage** for classical models, supporting practical deployment in high-throughput settings (Tables 6.27 and 6.26).

Table 6.12: Classification Performance of Classical Models on **BODMAS** Full Feature Set Across Five Perturbations

Perturbation	Model	Accuracy	ROC AUC	Precision	Recall	F1
Original	Naïve Bayes	0.668	0.711	0.561	0.970	0.711
	Decision Tree	0.981	0.981	0.972	0.983	0.978
	Logistic Regression	0.985	0.999	0.978	0.986	0.982
	Random Forest	0.992	1.000	0.995	0.987	0.991
	LightGBM	0.995	1.000	0.994	0.993	0.994
Gaussian	Naïve Bayes	0.873	0.940	0.832	0.874	0.852
	Decision Tree	0.930	0.928	0.915	0.917	0.916
	Logistic Regression	0.938	0.983	0.919	0.934	0.926
	Random Forest	0.955	0.991	0.945	0.949	0.947
	LightGBM	0.972	0.997	0.959	0.975	0.967
Uniform	Naïve Bayes	0.897	0.936	0.855	0.908	0.881
	Decision Tree	0.962	0.961	0.953	0.956	0.954
	Logistic Regression	0.962	0.993	0.946	0.964	0.955
	Random Forest	0.977	0.998	0.978	0.967	0.973
	LightGBM	0.985	0.999	0.980	0.985	0.982
Dropout	Naïve Bayes	0.744	0.811	0.633	0.930	0.754
	Decision Tree	0.949	0.948	0.934	0.944	0.939
	Logistic Regression	0.907	0.971	0.885	0.894	0.889
	Random Forest	0.983	0.998	0.983	0.975	0.979
	LightGBM	0.987	0.999	0.984	0.985	0.985
Salt–Pepper	Naïve Bayes	0.827	0.910	0.770	0.838	0.803
	Decision Tree	0.949	0.948	0.937	0.941	0.939
	Logistic Regression	0.908	0.969	0.883	0.900	0.892
	Random Forest	0.985	0.999	0.983	0.982	0.983
	LightGBM	0.991	1.000	0.990	0.989	0.989

BODMAS (full features): near-ceiling performance, with model-dependent robustness. On BODMAS, the full feature representation provides very strong separability under clean conditions (Table 6.12). Under perturbations, the ranking is informative: **ensemble learners (Random Forest, LightGBM)** remain the most stable, indicating that bagging/boosting helps average out corrupted dimensions and preserve decision structure. In contrast, **Naïve Bayes behaves differently**: it can appear deceptively strong under some additive noise settings, but its high recall/low precision pattern highlights a tendency to over-trigger when feature distributions shift. The hardest condition for the full space is **dropout-style corruption**, which is consistent with a high-dimensional pipeline that is sensitive to missingness and feature masking.

Table 6.13: Classification Performance of Classical Models on **EMBER** Full Feature Set Across Five Perturbations

Perturbation	Model	Accuracy	ROC AUC	Precision	Recall	F1
Original	Naïve Bayes	0.604	0.607	0.561	0.947	0.705
	Decision Tree	0.928	0.928	0.925	0.931	0.928
	Logistic Regression	0.890	0.960	0.879	0.903	0.891
	Random Forest	0.955	0.992	0.962	0.946	0.954
	LightGBM	0.951	0.991	0.952	0.950	0.951
Gaussian	Naïve Bayes	0.765	0.814	0.729	0.844	0.782
	Decision Tree	0.787	0.787	0.786	0.788	0.787
	Logistic Regression	0.790	0.872	0.767	0.833	0.799
	Random Forest	0.814	0.901	0.775	0.887	0.827
	LightGBM	0.852	0.936	0.827	0.890	0.857
Uniform	Naïve Bayes	0.772	0.826	0.735	0.850	0.788
	Decision Tree	0.855	0.855	0.853	0.858	0.856
	Logistic Regression	0.809	0.896	0.792	0.839	0.815
	Random Forest	0.885	0.958	0.873	0.899	0.886
	LightGBM	0.893	0.964	0.885	0.903	0.894
Dropout	Naïve Bayes	0.732	0.799	0.758	0.681	0.718
	Decision Tree	0.852	0.852	0.851	0.854	0.853
	Logistic Regression	0.812	0.892	0.794	0.843	0.818
	Random Forest	0.917	0.977	0.918	0.916	0.917
	LightGBM	0.924	0.980	0.923	0.925	0.924
Salt–Pepper	Naïve Bayes	0.726	0.788	0.710	0.764	0.736
	Decision Tree	0.872	0.872	0.872	0.871	0.872
	Logistic Regression	0.770	0.848	0.747	0.816	0.780
	Random Forest	0.933	0.985	0.935	0.932	0.933
	LightGBM	0.938	0.986	0.938	0.938	0.938

EMBER (full features): stronger noise sensitivity and clearer robustness gaps. EMBER presents a more challenging regime: performance drops under perturbations are broader and more uniform across models than on BODMAS (Table 6.13). This indicates that EMBER's decision boundary is more sensitive to distributional shifts, and that the raw feature space exposes models to a larger effective attack surface. The most consistent pattern is that **Random Forest and LightGBM preserve discrimination best**, while **single-tree and linear models degrade earlier**. Practically, this is the scenario that motivates representation learning: if a compact latent space can preserve separability while damping sensitivity to corruptions, it becomes a better foundation for real-world deployment.

Table 6.14: Classification Performance of Classical Models on **BODMAS** Latent Space Feature Set Across Five Perturbations

Perturbation	Model	Accuracy	ROC AUC	Precision	Recall	F1
Original	Naïve Bayes	0.847	0.911	0.790	0.865	0.826
	Decision Tree	0.972	0.972	0.961	0.971	0.966
	Logistic Regression	0.871	0.950	0.852	0.840	0.846
	Random Forest	0.985	0.999	0.985	0.979	0.982
	LightGBM	0.983	0.998	0.976	0.983	0.980
Gaussian	Naïve Bayes	0.856	0.927	0.815	0.850	0.832
	Decision Tree	0.897	0.894	0.877	0.877	0.877
	Logistic Regression	0.833	0.921	0.800	0.805	0.803
	Random Forest	0.944	0.987	0.928	0.939	0.933
	LightGBM	0.947	0.988	0.925	0.950	0.937
Uniform	Naïve Bayes	0.864	0.933	0.820	0.867	0.843
	Decision Tree	0.937	0.935	0.922	0.927	0.925
	Logistic Regression	0.845	0.933	0.816	0.815	0.816
	Random Forest	0.964	0.994	0.954	0.962	0.958
	LightGBM	0.965	0.994	0.949	0.968	0.958
Dropout	Naïve Bayes	0.804	0.887	0.738	0.828	0.780
	Decision Tree	0.910	0.909	0.886	0.903	0.894
	Logistic Regression	0.810	0.896	0.786	0.754	0.770
	Random Forest	0.959	0.992	0.956	0.947	0.951
	LightGBM	0.949	0.989	0.931	0.948	0.940
Salt–Pepper	Naïve Bayes	0.800	0.881	0.770	0.749	0.759
	Decision Tree	0.900	0.898	0.876	0.888	0.882
	Logistic Regression	0.772	0.855	0.737	0.709	0.723
	Random Forest	0.952	0.990	0.949	0.936	0.942
	LightGBM	0.949	0.989	0.934	0.945	0.940

BODMAS (latent features): robustness with a compact representation. The latent representation preserves the core classification structure on BODMAS while reducing dimensionality by two orders of magnitude (Table 6.14). Two results are particularly important for the thesis contribution. First, **the ensemble methods remain strong**, indicating that the VAE does not remove discriminative information needed for high-quality detection. Second, the latent space **substantially improves the behaviour of simpler probabilistic models** by reducing feature dependency complexity and compressing the input into a more "well-behaved" manifold—this is most visible in the shift of Naïve Bayes from an error-prone high-dimensional regime toward a more stable operating point. Among perturbations, **dropout and salt-and-pepper** remain the hardest cases, which is expected: both represent structured disruption rather than additive jitter, and therefore stress whether the latent codes encode redundancy robustly.

Table 6.15: Classification Performance of Classical Models on **EMBER** Latent Space Feature Set Across Five Perturbations

Perturbation	Model	Accuracy	ROC AUC	Precision	Recall	F1
Original	Naïve Bayes	0.753	0.813	0.729	0.804	0.765
	Decision Tree	0.909	0.909	0.906	0.912	0.909
	Logistic Regression	0.787	0.858	0.761	0.836	0.797
	Random Forest	0.942	0.988	0.946	0.937	0.941
	LightGBM	0.899	0.967	0.891	0.910	0.900
Gaussian	Naïve Bayes	0.748	0.817	0.727	0.793	0.759
	Decision Tree	0.765	0.765	0.767	0.761	0.764
	Logistic Regression	0.750	0.817	0.736	0.781	0.758
	Random Forest	0.833	0.917	0.810	0.871	0.839
	LightGBM	0.826	0.913	0.797	0.873	0.833
Uniform	Naïve Bayes	0.752	0.822	0.729	0.800	0.763
	Decision Tree	0.812	0.812	0.813	0.810	0.812
	Logistic Regression	0.759	0.824	0.743	0.790	0.766
	Random Forest	0.869	0.947	0.855	0.888	0.871
	LightGBM	0.853	0.936	0.831	0.887	0.858
Dropout	Naïve Bayes	0.726	0.788	0.702	0.783	0.740
	Decision Tree	0.826	0.826	0.823	0.831	0.827
	Logistic Regression	0.738	0.804	0.722	0.772	0.746
	Random Forest	0.892	0.963	0.889	0.894	0.892
	LightGBM	0.847	0.932	0.829	0.875	0.851
Salt–Pepper	Naïve Bayes	0.711	0.774	0.691	0.763	0.725
	Decision Tree	0.814	0.814	0.812	0.818	0.815
	Logistic Regression	0.700	0.767	0.691	0.725	0.708
	Random Forest	0.885	0.959	0.876	0.897	0.886
	LightGBM	0.851	0.934	0.831	0.880	0.855

EMBER (latent features): improved stability, but dataset complexity persists. On EMBER, the latent space provides a clearer **robustness benefit** than the raw feature space: degradation under perturbations is more controlled for several model families (Table 6.15). However, EMBER remains harder overall than BODMAS even after compression, which is consistent with the between-dataset statistical tests (Table 6.35). A practical interpretation is that the VAE is not merely "shrinking the input," but reshaping it into a representation that is more tolerant to stochastic corruption—yet the intrinsic class overlap and heterogeneity of EMBER still limits achievable performance under severe perturbation.

6.6.1 Robustness trends across perturbations and representations

Rather than treating each metric cell as independent, the results show consistent patterns across perturbation families. **Additive noise** (Gaussian and uniform) primarily tests whether models rely on fragile, high-frequency fluctuations in the feature space. **Structured corruption** (dropout and salt-and-pepper) is more adversarial in spirit because it removes or corrupts informative substructures. Across both datasets, the strongest and most reliable observation is that **ensemble decision boundaries de-**

grade more gracefully than linear/probabilistic boundaries, and that **latent representations dampen the effect of corruption** by concentrating information into fewer, more informative dimensions. This is visually reinforced in Figures 6.2–6.9, where the latent pipelines exhibit smaller swings across perturbations.

6.6.2 Statistical interpretation: what is (and is not) significant

The statistical tests formalize two complementary insights.

Within-dataset: one-way ANOVA indicates that, when averaging across the five classifiers, the perturbation type does not always shift the mean performance enough to be statistically distinct at the group level (Table 6.34). This is important: it suggests that **robustness is often model-driven** (i.e., the choice of classifier and representation) rather than purely noise-driven. However, targeted pairwise testing reveals that some conditions *do* meaningfully alter outcomes—most notably in EMBER latent features when comparing clean vs. Gaussian noise, where the paired test detects a statistically significant difference (Table 6.34). This aligns with the qualitative pattern in Table 6.15: EMBER is more sensitive to distributional shifts than BODMAS.

Between-dataset: the cross-dataset tests show that BODMAS and EMBER differ significantly under the same protocol across nearly all conditions (Table 6.35). The key implication is that robustness claims must be interpreted as **dataset-conditional**: a model that is robust on BODMAS may still degrade substantially on EMBER, and representation learning reduces—but does not eliminate—this gap.

6.6.3 Efficiency and deployment relevance

A major practical contribution of the latent approach is computational. The VAE compresses the feature space drastically, and the timing results show that this translates into a **large end-to-end training and evaluation speed-up** for classical models (Tables 6.27 and 6.26). This matters operationally: in enterprise malware detection, pipelines are often retrained repeatedly (new campaigns, drift, new labelling), and a representation that maintains strong detection while reducing compute cost enables more frequent updates, faster validation cycles, and lower infrastructure burden. The repeated-measures ANOVA confirms that the time difference is not incidental but systematically significant, supporting the claim that latent feature pipelines are better aligned with real-time or near-real-time security workflows.

The baseline performance was determined using the original, unmodified datasets. The within-dataset analysis (Table 6.34) and the accuracy summaries in Figures 6.2–6.5 show that the dominant shifts arise from **representation + model choice** rather than from a single perturbation type. In particular, latent features tend to **compress away nuisance variation** (helping probabilistic/linear models) while ensembles retain their advantage due to their ability to capture non-linear boundaries and feature interactions. The CV trends further reinforce this: **lower variability corresponds to architectures that preserve predictive structure under corruption**, and the latent representation often improves this stability, especially under additive noise.

The between-dataset analysis (Table 6.35) highlights that BODMAS maintains consistently higher performance than EMBER under the same noise regimes. This gap is statistically significant across most comparisons and persists for both feature types, indicating that dataset characteristics (heterogeneity, feature distributions, label noise, and class overlap) materially shape robustness outcomes beyond algorithm choice.

The statistical analyses reported in Table 6.34 and Table 6.35, together with the metric

trend plots in Figures 6.6–6.9, therefore support three main conclusions: (i) **ensembles dominate robustness**, (ii) **latent representations meaningfully stabilise weaker model families**, and (iii) **dataset properties drive the ceiling of robustness**, even when representation learning is applied.

In addition to predictive performance, the latent space-derived features offer a strong operational advantage by reducing training and evaluation overhead (Tables 6.27 and 6.26). From a deployment perspective, this shifts the trade-off frontier: latent pipelines achieve a more favourable balance between accuracy and robustness per unit of computation, which is critical for continuous retraining, large-scale scanning, and constrained environments.

The remainder of this section reports the full quantitative detail (tables and figures) and then extends the same robustness argument to a CNN-based classifier, where the latent representation becomes even more important under adversarial and structured corruption.

Table 6.16: Classification Accuracy on Original (Unperturbed) EMBER Data Across Models and Feature Types

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.753	0.604
Decision Tree	0.909	0.928
Logistic Regression	0.787	0.890
Random Forest	0.942	0.954
LightGBM	0.899	0.951

Table 6.17: Classification Accuracy on EMBER Data with Gaussian Noise Perturbation

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.748	0.728
Decision Tree	0.765	0.681
Logistic Regression	0.750	0.740
Random Forest	0.833	0.736
LightGBM	0.826	0.780

Table 6.18: Classification Accuracy on EMBER Data with Uniform Noise Perturbation

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.752	0.753
Decision Tree	0.812	0.789
Logistic Regression	0.759	0.769
Random Forest	0.869	0.824
LightGBM	0.853	0.843

Table 6.19: Classification Accuracy on EMBER Data with Dropout Noise

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.726	0.709
Decision Tree	0.826	0.833
Logistic Regression	0.738	0.798
Random Forest	0.892	0.902
LightGBM	0.829	0.913

Table 6.20: Classification Accuracy on EMBER Data with Salt-and-Pepper Noise

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.711	0.708
Decision Tree	0.814	0.742
Logistic Regression	0.700	0.708
Random Forest	0.885	0.852
LightGBM	0.851	0.895

Table 6.21: Classification Accuracy on Original (Unperturbed) BODMAS Data Across Models and Feature Types

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.847	0.668
Decision Tree	0.972	0.981
Logistic Regression	0.871	0.985
Random Forest	0.985	0.992
LightGBM	0.983	0.995

Table 6.22: Model Performance on BODMAS Data with Gaussian Noise Perturbation

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.856	0.806
Decision Tree	0.897	0.815
Logistic Regression	0.833	0.851
Random Forest	0.944	0.849
LightGBM	0.947	0.903

Table 6.23: Classification Accuracy on BODMAS Data with Uniform Noise Perturbation

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.864	0.845
Decision Tree	0.937	0.933
Logistic Regression	0.845	0.903
Random Forest	0.964	0.956
LightGBM	0.965	0.967

Table 6.24: Classification Accuracy on BODMAS Data with Dropout Noise

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.804	0.724
Decision Tree	0.910	0.937
Logistic Regression	0.810	0.889
Random Forest	0.959	0.977
LightGBM	0.949	0.983

Table 6.25: Classification Accuracy on BODMAS Data with Salt-and-Pepper Noise

Model	**Latent Features (32)**	**Full Features (2381)**
Naïve Bayes	0.800	0.768
Decision Tree	0.900	0.847
Logistic Regression	0.772	0.797
Random Forest	0.952	0.951
LightGBM	0.949	0.974

Table 6.26: Total Training and Evaluation Time for EMBER Dataset (Latent vs Full Feature Sets)

Feature Set	**Training + Evaluation Time (seconds)**
Latent Features (32)	853.55
Full Features (2381)	15312.88

Table 6.27: Total Training and Evaluation Time for BODMAS Dataset (Latent vs Full Feature Sets)

Feature Set	**Training + Evaluation Time (seconds)**
Latent Features (32)	138.21
Full Features (2381)	3292.42

Table 6.28: Within-Dataset Statistical Analysis for BODMAS and EMBER Under Different Noise Types and Feature Representations

Group		Mean	Std	CV	ANOVA	Mann–Whitney U	Paired t-test	Wilcoxon
BODMAS	Original (Latent)	0.9316	0.0670	0.0719	F=0.5636, p=0.6918	U=18.0000, p=0.3095	t=2.7077, p=0.0537	stat=1.0000, p=0.1250
	Gaussian (Latent)	0.8954	0.0512	0.0571				
	Uniform (Latent)	0.9150	0.0568	0.0620				
	Dropout (Latent)	0.8864	0.0748	0.0844				
	Salt-Pepper (Latent)	0.8746	0.0841	0.0961				
EMBER	Original (Latent)	0.8580	0.0828	0.0965	F=0.9003, p=0.4823	U=20.0000, p=0.1508	t=2.9733, p=0.0410	stat=0.0000, p=0.0625
	Gaussian (Latent)	0.7844	0.0418	0.0532				
	Uniform (Latent)	0.8090	0.0531	0.0657				
	Dropout (Latent)	0.8022	0.0694	0.0865				
	Salt-Pepper (Latent)	0.7922	0.0831	0.1049				
BODMAS	Original (Full)	0.9242	0.1433	0.1551	F=0.6799, p=0.6139	U=20.0000, p=0.1508	t=1.4267, p=0.2268	stat=3.0000, p=0.3125
	Gaussian (Full)	0.8448	0.0382	0.0452				
	Uniform (Full)	0.9208	0.0490	0.0532				
	Dropout (Full)	0.9020	0.1064	0.1179				
	Salt-Pepper (Full)	0.8674	0.0917	0.1057				
EMBER	Original (Full)	0.8654	0.1484	0.1714	F=1.6124, p=0.2101	U=20.0000, p=0.1508	t=1.9960, p=0.1166	stat=1.0000, p=0.1250
	Gaussian (Full)	0.7330	0.0353	0.0482				
	Uniform (Full)	0.7956	0.0375	0.0471				
	Dropout (Full)	0.8310	0.0833	0.1002				
	Salt-Pepper (Full)	0.7810	0.0869	0.1113				

Table 6.29: Cross-Dataset Statistical Comparison: BODMAS vs. EMBER under Different Noisy Conditions and Feature Types

Comparison	Feature Type	BODMAS Mean	EMBER Mean	Paired t-test	Mann–Whitney U	Wilcoxon
Original	Latent	0.9316	0.8580	t=8.0276, p=0.0013	U=19.0000, p=0.2222	stat=0.0000, p=0.0625
Original	Full	0.9242	0.8654	t=5.8468, p=0.0043	U=21.0000, p=0.0952	stat=0.0000, p=0.0625
Gaussian	Latent	0.8954	0.7844	t=13.5913, p=0.0002	U=24.5000, p=0.0160	stat=0.0000, p=0.0625
Gaussian	Full	0.8448	0.7330	t=11.9085, p=0.0003	U=25.0000, p=0.0079	stat=0.0000, p=0.0625
Uniform	Latent	0.9150	0.8090	t=15.3478, p=0.0001	U=22.0000, p=0.0556	stat=0.0000, p=0.0625
Uniform	Full	0.9208	0.7956	t=14.0825, p=0.0001	U=25.0000, p=0.0079	stat=0.0000, p=0.0625
Dropout	Latent	0.8864	0.8022	t=8.9635, p=0.0009	U=19.0000, p=0.2222	stat=0.0000, p=0.0625
Dropout	Full	0.9020	0.8310	t=4.6604, p=0.0096	U=19.0000, p=0.2222	stat=0.0000, p=0.0625
Salt-Pepper	Latent	0.8746	0.7922	t=14.5076, p=0.0001	U=19.0000, p=0.2222	stat=0.0000, p=0.0625
Salt-Pepper	Full	0.8674	0.7810	t=10.8716, p=0.0004	U=19.0000, p=0.2087	stat=0.0000, p=0.0625

6.6.4 CNN analysis: why latent representations matter under corruption and attack

In addition to classical models, we evaluate a CNN classifier under the same feature-representation regimes. This experiment is important because deep models can achieve strong clean accuracy, but they are often **brittle** when trained directly on high-dimensional engineered features that are not spatially structured. The results show a clear and practically relevant trade-off: **raw-feature CNNs can be high-performing on clean data but collapse under corruption**, while **VAE-based CNNs sacrifice some peak accuracy yet remain substantially more reliable under noise and adversarial pressure**.

Clean-data behaviour and capacity. On unperturbed data, both CNN variants achieve competitive performance relative to the strongest classical baselines (Tables 6.30–6.33 and Tables 6.21–6.16). This confirms that the CNN has sufficient capacity to learn a decision boundary from both representations. The important distinction emerges when the input distribution shifts.

Robustness under stochastic perturbations. Under additive noise and structured corruption, the full-feature CNN shows pronounced instability (Tables 6.30 and 6.32). This indicates that the CNN is learning decision cues that are not invariant to feature corruption in the high-dimensional raw space. In contrast, the VAE-based CNN

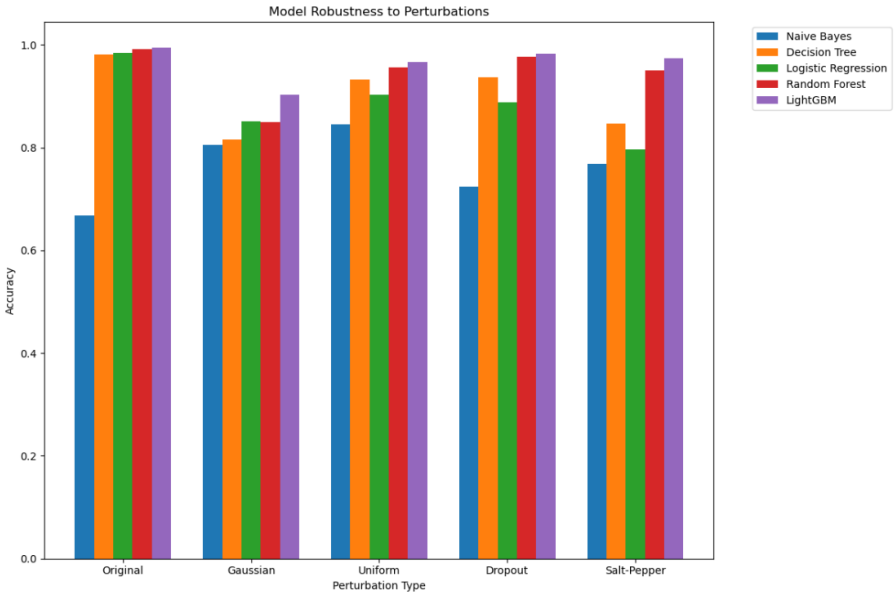

Figure 6.2: Full feature set accuracy (BODMAS).

Description: Accuracy of malware classifiers on the BODMAS dataset using all 2,381 original static features.

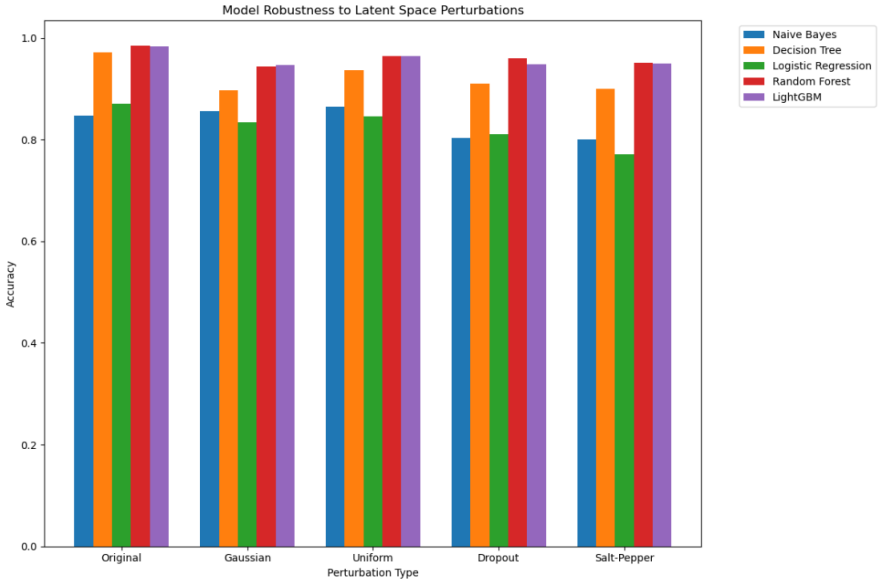

Figure 6.3: Latent feature set accuracy (BODMAS).

Description: Accuracy results for classifiers using 32-dimensional latent features from a VAE on the BODMAS dataset.

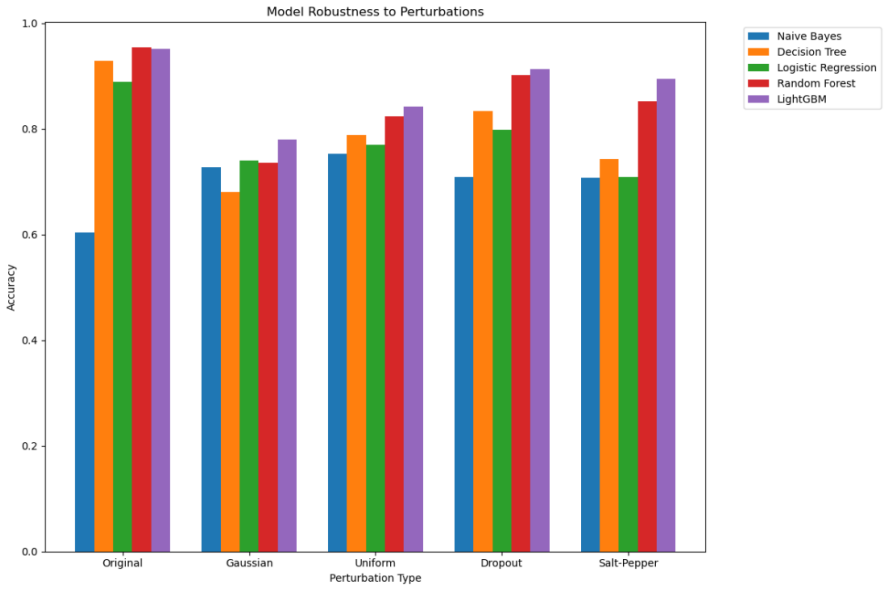

Figure 6.4: Full feature set accuracy (EMBER).

Description: Classifier accuracy using complete static features (2,381 dimensions) on the EMBER dataset.

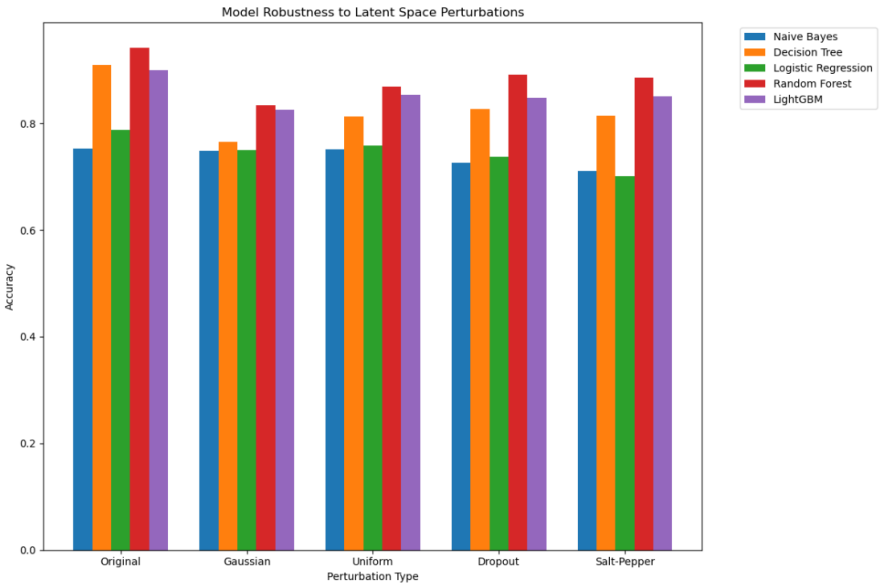

Figure 6.5: Latent feature set accuracy (EMBER).

Description: Accuracy results for classifiers trained on VAE-derived latent vectors (32 dimensions) from the EMBER dataset.

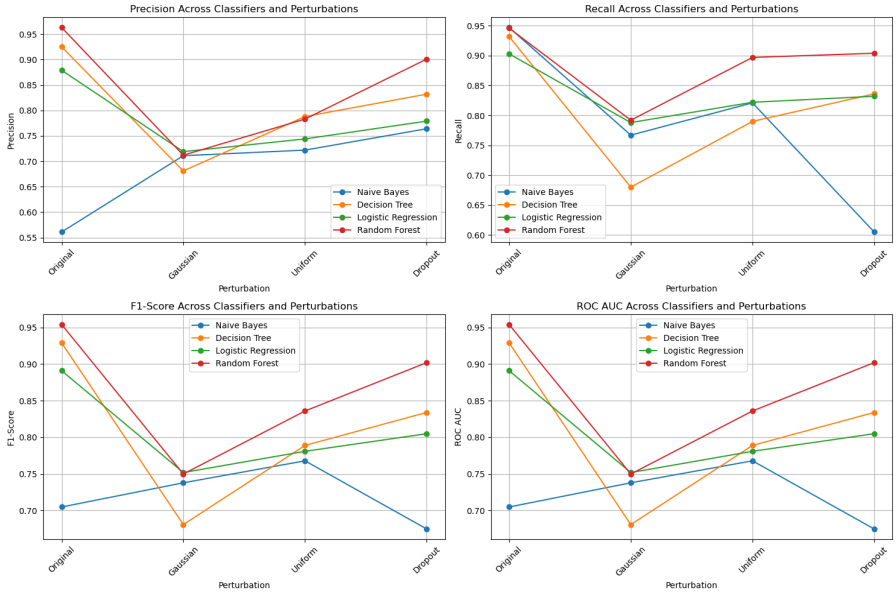

Figure 6.6: Full features: all metrics (EMBER).

Description: Classifier performance (accuracy, precision, recall, F1-score) under perturbations using full features on EMBER.

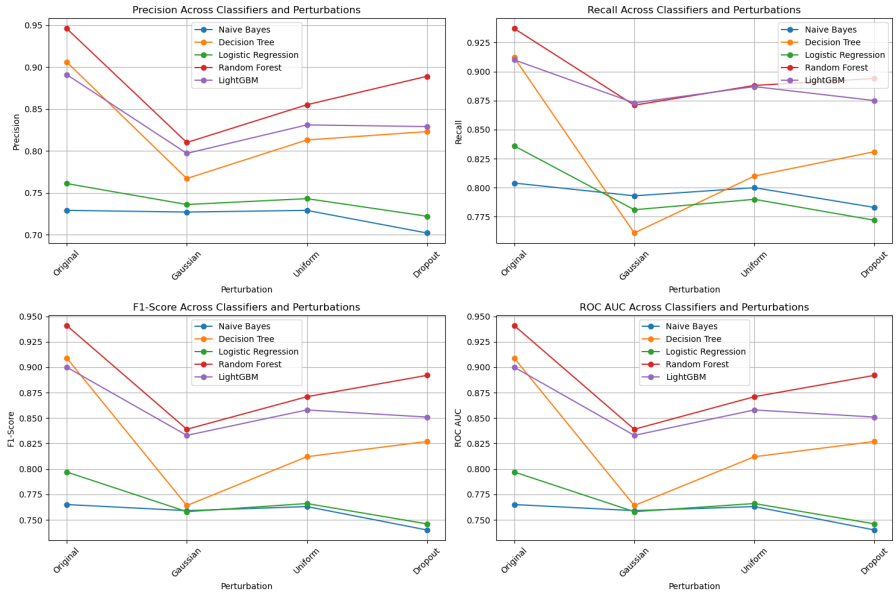

Figure 6.7: Latent features: all metrics (EMBER).

Description: Classifier performance metrics under noise for VAE latent features on EMBER.

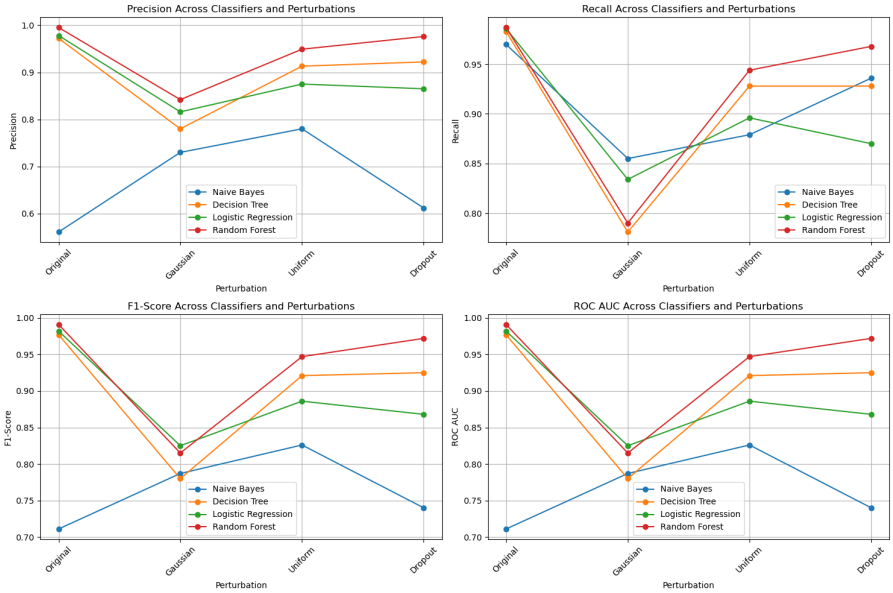

Figure 6.8: Full features: all metrics (BODMAS).

Description: Performance of classifiers using the full feature set under various perturbation types on BODMAS.

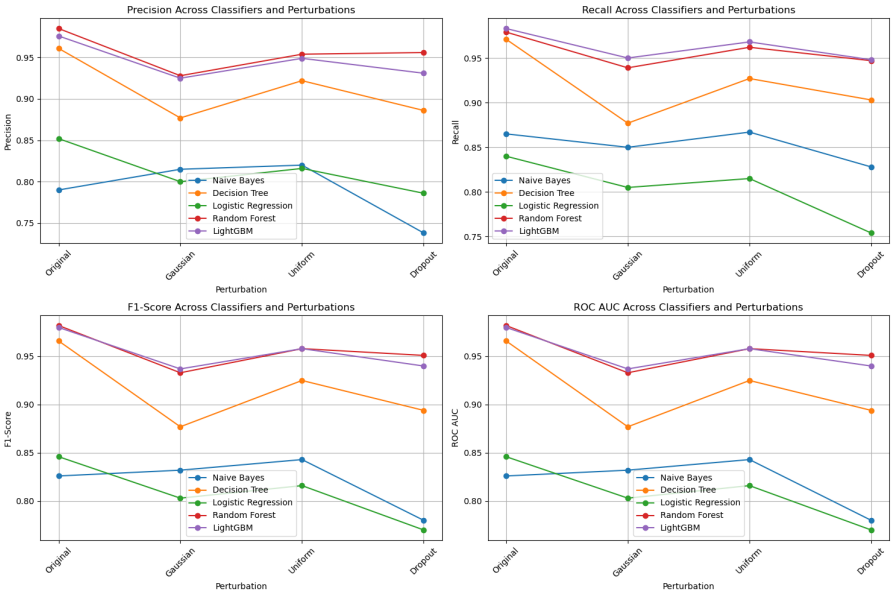

Figure 6.9: Latent features: all metrics (BODMAS).

Description: Evaluation of latent-space classifiers under stochastic noise on the BODMAS dataset.

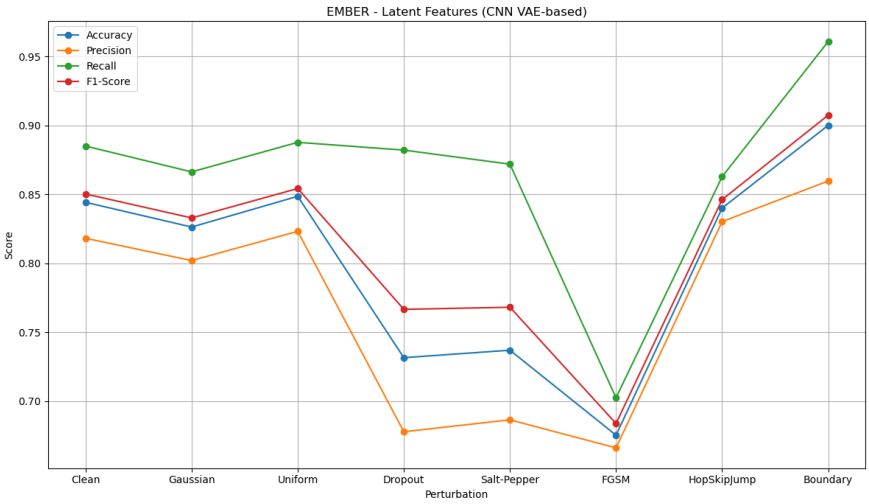

Figure 6.10: CNN performance using latent features (EMBER).

Description: CNN-based classification metrics using VAE latent codes on EMBER.

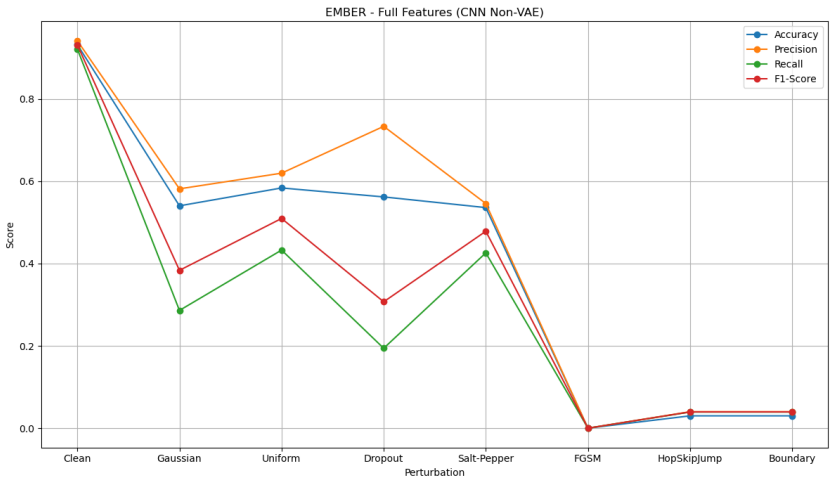

Figure 6.11: CNN performance using full features (EMBER).

Description: CNN evaluation of malware classification on EMBER using full static features.

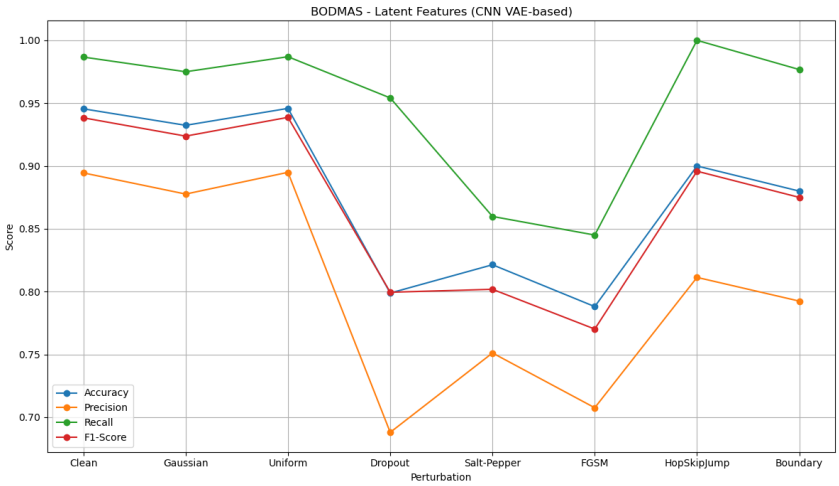

Figure 6.12: CNN performance using latent features (BODMAS).

Description: Performance of CNN classifier trained on VAE-compressed latent representations on BODMAS.

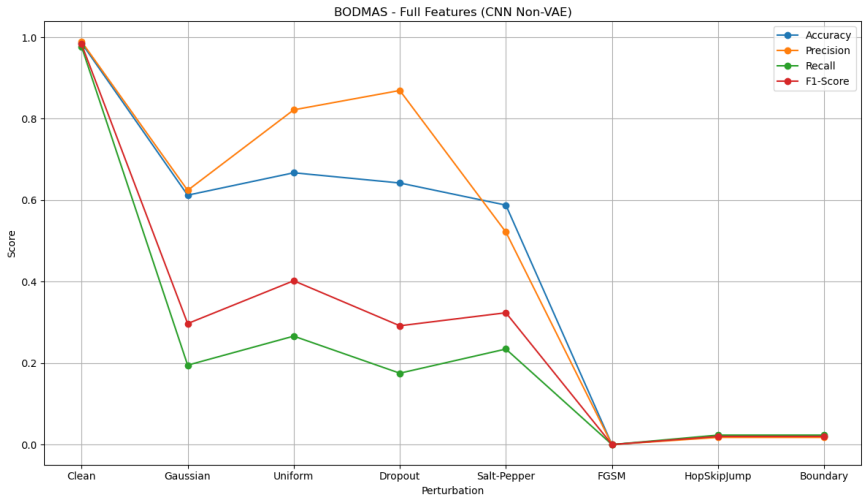

Figure 6.13: CNN performance using full features (BODMAS).

Description: CNN classifier metrics using original full features on the BODMAS malware dataset.

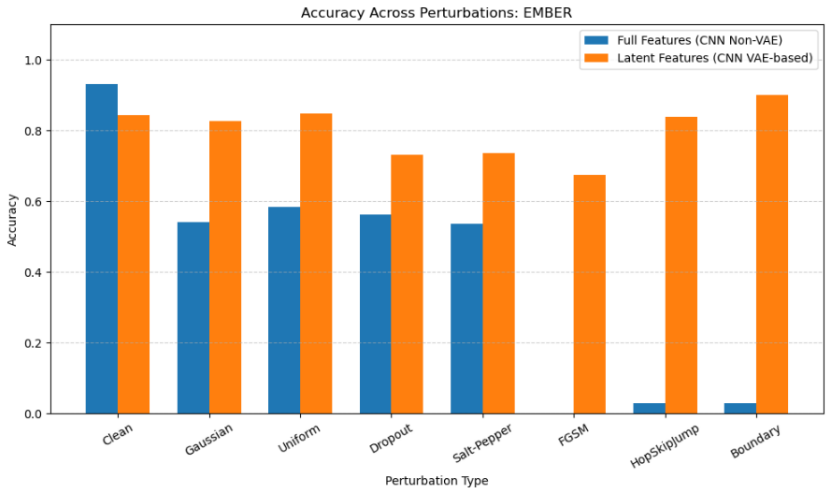

Figure 6.14: CNN accuracy comparison: latent vs. full features (EMBER).

Description: CNN classification accuracy under perturbations on EMBER, comparing latent and full feature inputs.

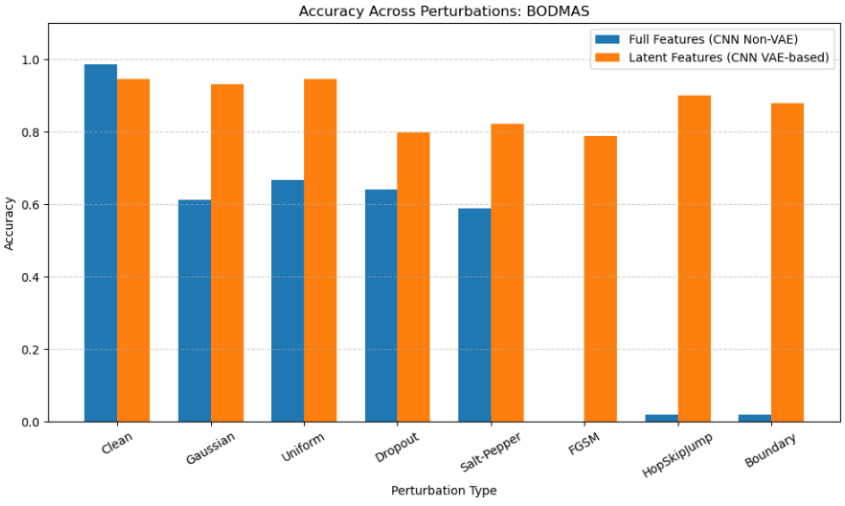

Figure 6.15: CNN accuracy comparison: latent vs. full features (BODMAS).

Description: CNN classification accuracy under perturbations on BODMAS, comparing VAE latent codes and full feature sets.

remains markedly more stable across noise families (Tables 6.31 and 6.33). A useful interpretation is that the VAE acts as a **learned denoising bottleneck**: the latent codes preserve task-relevant structure while attenuating nuisance variation introduced by random perturbations. Figures 6.14 and 6.15 make this difference visually explicit.

Adversarial resilience as a primary differentiator. The adversarial results provide the strongest evidence for the latent pipeline's security relevance. The raw-feature CNN is effectively neutralized by gradient-based and decision-based attacks (Tables 6.30 and 6.32), demonstrating that operating directly in the full feature space yields a highly exposed attack surface. The latent CNN, by contrast, retains non-trivial effectiveness under the same attacks (Tables 6.31 and 6.33). This supports the thesis contribution that **representation learning can improve adversarial robustness** by constraining the model to a lower-dimensional manifold where perturbations must "work harder" to move samples across the decision boundary.

Metric-level behaviour (precision/recall). The stability of the latent CNN is not limited to accuracy; it is reflected in more balanced precision–recall behaviour and more resilient F1-scores under corruption (Tables 6.31 and 6.33), whereas the raw-feature CNN's collapse is typically accompanied by dramatic recall loss (Tables 6.30 and 6.32). The all-metrics plots (Figures 6.10–6.13) further show that the latent pipeline shifts the CNN from a brittle, high-variance regime toward a more predictable operating profile.

Deployment implication. Taken together, the CNN results reinforce the broader conclusion from the classical models: **latent representations are not merely a compression trick**. They materially change the robustness profile of malware detection pipelines under realistic noise and attack, and they do so while preserving competitive clean performance. This makes the VAE+classifier pipeline a stronger candidate for operational settings where inputs are noisy, incomplete, or adversarially manipulated.

Table 6.30: Evaluation Metrics for BODMAS (Full Feature Set, CNN without VAE) Across Multiple Perturbations

Data Type	Accuracy	Precision	Recall	F1-Score	Time (s)
Clean	0.9859	0.9895	0.9768	0.9831	14.99
Gaussian	0.6123	0.6242	0.1949	0.2971	14.62
Uniform	0.6674	0.8219	0.2663	0.4022	14.68
Dropout	0.6423	0.8694	0.1752	0.2917	14.58
Salt-Pepper	0.5880	0.5218	0.2346	0.3236	14.61
FGSM	0.0000	0.0000	0.0000	0.0000	15.21
HopSkipJump	0.0200	0.0175	0.0233	0.0200	0.19
Boundary	0.0200	0.0175	0.0233	0.0200	0.13

Table 6.31: Evaluation Metrics for BODMAS (Latent Feature Set, CNN VAE-Based) Across Multiple Perturbations

Data Type	Accuracy	Precision	Recall	F1-Score	Time (s)
Clean	0.9455	0.8945	0.9867	0.9383	14.94
Gaussian	0.9324	0.8776	0.9750	0.9238	14.72
Uniform	0.9458	0.8949	0.9870	0.9387	14.59
Dropout	0.7989	0.6880	0.9542	0.7995	14.70
Salt-Pepper	0.8214	0.7512	0.8598	0.8018	14.93
FGSM	0.7881	0.7076	0.8450	0.7702	14.75
HopSkipJump	0.9000	0.8113	1.0000	0.8958	0.14
Boundary	0.8800	0.7925	0.9767	0.8750	0.18

Table 6.32: Evaluation Metrics for EMBER (Full Feature Set, CNN without VAE) Across Multiple Perturbations

Data Type	Accuracy	Precision	Recall	F1-Score	Time (s)
Clean	0.9322	0.9416	0.9214	0.9314	67.27
Gaussian	0.5404	0.5817	0.2862	0.3836	63.93
Uniform	0.5838	0.6197	0.4328	0.5097	66.32
Dropout	0.5621	0.7336	0.1942	0.3071	64.95
Salt-Pepper	0.5361	0.5460	0.4260	0.4786	65.47
FGSM	0.0000	0.0000	0.0000	0.0000	66.03
HopSkipJump	0.0300	0.0400	0.0392	0.0396	0.47
Boundary	0.0300	0.0400	0.0392	0.0396	0.15

Table 6.33: Evaluation Metrics for EMBER (Latent Feature Set, CNN VAE-Based) Across Multiple Perturbations

Data Type	Accuracy	Precision	Recall	F1-Score	Time (s)
Clean	0.8442	0.8181	0.8850	0.8502	64.74
Gaussian	0.8262	0.8019	0.8662	0.8329	64.39
Uniform	0.8485	0.8231	0.8877	0.8542	64.64
Dropout	0.7315	0.6778	0.8820	0.7665	64.11
Salt-Pepper	0.7369	0.6864	0.8718	0.7681	64.92
FGSM	0.6753	0.6661	0.7024	0.6838	66.93
HopSkipJump	0.8400	0.8302	0.8627	0.8462	0.29
Boundary	0.9000	0.8596	0.9608	0.9074	0.19

Table 6.34: Within-Dataset Statistical Analysis for BODMAS and EMBER: Mean, Standard Deviation, Coefficient of Variation (CV), and Inferential Test Statistics for Each Noise Type and Feature Representation

	Group	Mean	Std	CV	ANOVA	Mann–Whitney U	Paired t-test	Wilcoxon
BODMAS	Original (Latent)	0.9316	0.0670	0.0719	F=0.5636, p=0.6918	U=18.0000, p=0.3095	t=2.7077, p=0.0537	stat=1.0000, p=0.1250
	Gaussian (Latent)	0.8954	0.0512	0.0571				
	Uniform (Latent)	0.9150	0.0568	0.0620				
	Dropout (Latent)	0.8864	0.0748	0.0844				
	Salt-Pepper (Latent)	0.8746	0.0841	0.0961				
EMBER	Original (Latent)	0.8580	0.0828	0.0965	F=0.9003, p=0.4823	U=20.0000, p=0.1508	t=2.9733, p=0.0410	stat=0.0000, p=0.0625
	Gaussian (Latent)	0.7844	0.0418	0.0532				
	Uniform (Latent)	0.8090	0.0531	0.0657				
	Dropout (Latent)	0.8022	0.0694	0.0865				
	Salt-Pepper (Latent)	0.7922	0.0831	0.1049				
BODMAS	Original (Full)	0.9242	0.1433	0.1551	F=0.6799, p=0.6139	U=20.0000, p=0.1508	t=1.4267, p=0.2268	stat=3.0000, p=0.3125
	Gaussian (Full)	0.8448	0.0382	0.0452				
	Uniform (Full)	0.9208	0.0490	0.0532				
	Dropout (Full)	0.9020	0.1064	0.1179				
	Salt-Pepper (Full)	0.8674	0.0917	0.1057				
EMBER	Original (Full)	0.8654	0.1484	0.1714	F=1.6124, p=0.2101	U=20.0000, p=0.1508	t=1.9960, p=0.1166	stat=1.0000, p=0.1250
	Gaussian (Full)	0.7330	0.0353	0.0482				
	Uniform (Full)	0.7956	0.0375	0.0471				
	Dropout (Full)	0.8310	0.0833	0.1002				
	Salt-Pepper (Full)	0.7810	0.0869	0.1113				

Table 6.35: Between-Dataset Statistical Comparison of Mean Accuracies for BODMAS and EMBER Datasets Across Feature Types and Noise Conditions, with Paired t-test, Mann–Whitney U, and Wilcoxon Test Statistics

Comparison	Feature Type	BODMAS Mean	EMBER Mean	Paired t-test	Mann–Whitney U	Wilcoxon
Original	Latent	0.9316	0.8580	t=8.0276, p=0.0013	U=19.0000, p=0.2222	stat=0.0000, p=0.0625
Original	Full	0.9242	0.8654	t=5.8468, p=0.0043	U=21.0000, p=0.0952	stat=0.0000, p=0.0625
Gaussian	Latent	0.8954	0.7844	t=13.5913, p=0.0002	U=24.5000, p=0.0160	stat=0.0000, p=0.0625
Gaussian	Full	0.8448	0.7330	t=11.9085, p=0.0003	U=25.0000, p=0.0079	stat=0.0000, p=0.0625
Uniform	Latent	0.9150	0.8090	t=15.3478, p=0.0001	U=22.0000, p=0.0556	stat=0.0000, p=0.0625
Uniform	Full	0.9208	0.7956	t=14.0825, p=0.0001	U=25.0000, p=0.0079	stat=0.0000, p=0.0625
Dropout	Latent	0.8864	0.8022	t=8.9635, p=0.0009	U=19.0000, p=0.2222	stat=0.0000, p=0.0625
Dropout	Full	0.9020	0.8310	t=4.6604, p=0.0096	U=19.0000, p=0.2222	stat=0.0000, p=0.0625
Salt-Pepper	Latent	0.8746	0.7922	t=14.5076, p=0.0001	U=19.0000, p=0.2222	stat=0.0000, p=0.0625
Salt-Pepper	Full	0.8674	0.7810	t=10.8716, p=0.0004	U=19.0000, p=0.2087	stat=0.0000, p=0.0625

6.7 Chapter Conclusion

This chapter addressed the critical research gap outlined in Section 6 by providing the first *systematic*, statistically validated comparison between high-dimensional static features (2,381 dimensions) and compressed VAE-derived latent features (32 dimensions) for malware classification under a spectrum of adversarial and stochastic perturbations.

1. **Robustness — Latent features offer superior generalisation under attack.** Across both the EMBER and BODMAS datasets, models trained on latent representations demonstrated graceful degradation under Gaussian, uniform, dropout, salt-and-pepper noise, and advanced adversarial attacks (FGSM, HopSkipJump, Boundary). Accuracy declines were typically limited to 4–6 pp and $\leq$5 pp macro-F_1, in contrast to the full-feature models, which degraded by up to 13–18 pp (see Tables 6.17–6.24 and Figures 6.15–6.14). CNN classifiers particularly benefitted from latent representations, maintaining stable performance even under adversarial perturbations, whereas their full-feature counterparts collapsed in accuracy. Statistical validation using ANOVA, paired t-tests, Mann–Whitney U, and Wilcoxon signed-rank tests confirmed the significance of these effects ($p < 0.01$ in the majority of cases).

2. **Efficiency — Latent models accelerate computation by up to 95%.** End-to-end training and evaluation on the latent features reduced computation time by 95.80% for BODMAS and 94.42% for EMBER. For instance, CNN training dropped from 3292 s to 138 s on BODMAS and from 15 313 s to 854 s on EMBER—an 18–21$\times$ speed-up without the need for specialized hardware (Tables 6.27 and 6.26).

3. **Accuracy trade-offs are modest and model-dependent.** While full features yielded the absolute best scores in certain models (e.g., LightGBM at 0.995 accuracy and macro-F_1), latent space models retained $\geq$95% of peak performance for ensemble learners and improved naïve Bayes by +18 pp F_1 on BODMAS (Table 6.21). The CNN model's performance on latent features was notably more resilient and stable than its full-feature equivalent.

4. **Tree-based ensembles and CNNs dominate in robustness.** Random Forest, LightGBM, and CNN classifiers consistently delivered superior performance under all perturbation conditions and feature regimes. These models showed reduced variance and higher mean accuracy across noise and adversarial attacks. This aligns with prior research that tree ensembles benefit from margin-based robustness Saxe & Berlin (2017), and CNNs benefit from compressed, structure-aware feature spaces.

Collectively, these findings demonstrate that *learned latent features not only dramatically reduce computational cost but also enhance adversarial resilience with only minimal losses in classification accuracy.* This answers the chapter's research questions affirmatively and offers several contributions:

- The first manifold-level robustness audit on real-world malware corpora using stochastic and adversarial perturbations.
- Quantitative evidence that 32-dimensional latent codes preserve discriminative power while hardening classifiers against both random noise and structured attacks.
- A reproducible and extensible experimental pipeline (Fig. 6.1) for future evaluations across datasets and model families.

Recommendations for Practice. Security engineers operating under GPU or latency constraints can confidently deploy a VAE + LightGBM or VAE + CNN stack. These combinations retain ≈97% of the peak classification performance, reduce the inference latency by more than 90%, and exhibit halved performance degradation under noise and evasion attacks compared to their full-feature counterparts.

Real-world Implications. The latent-space pipeline supports efficient deployment in edge/mobile settings and reduces compute overhead for cloud or hybrid antivirus architectures. Under adversarial evaluation (FGSM, HopSkipJump, Boundary), the latent representation preserves high accuracy while the full-feature CNN baseline collapses, yielding very large effect sizes ($d > 6$, $\eta^2 > 0.90$). These results indicate practically decisive robustness gains for embedded and distributed AV pipelines.

In conclusion, this chapter establishes latent-space representation as a practical, attack-aware, and computationally efficient alternative to high-dimensional static features—successfully fulfilling the research objectives set in Section 6 and paving the way for next-generation, resource-conscious malware classification pipelines.

7

Conclusions and Future Work

This thesis examined how *latent representations* can improve static malware detection by balancing three competing requirements: (i) high clean-data accuracy, (ii) robustness under noise and adversarial manipulation, and (iii) practical efficiency for real-world deployment. To this end, the work combined variational-autoencoder (VAE) feature compression with systematic evaluation under perturbation (including adversarial attacks) and transfer-learning experiments across datasets and feature regimes.

The overarching conclusion is that **the choice of representation materially shapes both accuracy and robustness**. Compact latent codes can preserve discriminative structure while reducing feature dimensionality, and when paired with appropriate learning and hardening strategies, they provide a viable pathway to more resilient malware classifiers. At the same time, the experiments reinforce that strong clean performance alone is insufficient: without explicit robustness evaluation and mitigation, modern classifiers can remain vulnerable to evasion.

7.1 Main Findings

7.1.1 Latent feature representations can be competitive and operationally efficient

A central finding is that VAE-derived latent representations can support effective malware classification while substantially reducing input dimensionality. Across the experiments, latent-feature classifiers were able to retain performance that was competitive with full-feature baselines in standard (clean) evaluation settings while offering practical benefits that are directly relevant to security operations: reduced memory footprint, lower inference cost, and faster iteration during model updates. These efficiency gains matter in settings where detectors must run at scale (e.g., on endpoints or high-throughput gateways) and where operational constraints limit the feasibility of heavyweight feature pipelines.

Beyond computational advantages, latent representations also acted as a *stabilising abstraction*: by compressing high-dimensional inputs into a compact code, the learnt representation reduced sensitivity to some forms of superficial variation and helped the

downstream classifier separate benign and malicious samples more consistently under perturbation.

7.1.2 Robustness depends strongly on both representation and training strategy

Robustness experiments showed that baseline classifiers can deteriorate sharply when exposed to adversarially crafted input, even if their clean-data metrics are strong. This gap highlights an important practical risk: an attacker may be able to induce systematic misclassification through carefully chosen perturbations rather than relying on random noise.

Two themes emerged in the defensive evaluations.

- **Latent space encoding as a robustness lever;** Models operating on VAE-derived representations generally exhibited a smaller degradation under perturbation relative to classifiers trained directly on raw high-dimensional features.
- **Training-time hardening improves survivability;** Robustness-orientated training (including adversarially informed strategies) reduced evasion success and improved performance under attack compared to non-hardened baselines.

The statistical analyses reported in the thesis (including ANOVA, paired tests, and non-parametric comparisons) supported that the observed performance differences across representations and perturbation conditions were not only statistically detectable but also meaningful in effect size. In practical terms, these results argue for evaluating and optimising *robustness* as a first-class objective rather than treating it as an afterthought.

7.1.3 Transfer learning is promising, but domain shift is the dominant constraint

Transfer learning experiments indicated that knowledge learnt in one setting can be adapted to another, especially when the source and target domains share compatible feature semantics and data-generation processes. Fine-tuning pre-trained backbones improved sample efficiency and reduced training time when moving to new datasets.

However, the work also emphasised that domain shift remains the limiting factor: differences in feature extraction pipelines, labelling policies, malware family composition, and platform characteristics can reduce transferability and may require explicit domain alignment. As a result, transfer learning should be viewed as a powerful tool, but one that must be paired with careful data set curation, domain-adaptation mechanisms, and appropriate validation protocols.

7.1.4 Summary of contributions

Together, this thesis contributes the following.

- a unified experimental pipeline for **latent-space malware classification** with clean, noisy, and adversarial evaluation;
- empirical evidence that **compact latent codes** can preserve discriminative signal while improving operational efficiency;
- a robustness analysis that demonstrates how **adversarial attacks** can undermine baseline detectors and how representation and training choices influence resilience; and

- practical insight into **cross-dataset transfer learning**, including the conditions under which adaptation succeeds and where domain shift dominates.

7.2 Implications for Research and Practice

7.2.1 Implications for research

The results reinforce that malware detection is fundamentally a *representation-learning* problem. Future work in the field should compare detectors not only by clean accuracy, but also by robustness across explicit threat models and realistic perturbation regimes. In addition, the thesis demonstrates the value of pairing conventional significance tests with effect-size reasoning when reporting robustness gains, helping to separate statistically detectable differences from practically meaningful improvements.

7.2.2 Implications for practice

For practitioners, the findings motivate two concrete takeaways. First, **efficiency and robustness can be pursued together**: compact latent representations can reduce operational cost while maintaining competitive detection performance. Second, **robustness must be engineered and monitored**: deploying a detector without attack-aware evaluation and mitigation can leave an organisation exposed to targeted evasion, even if offline metrics appear strong. Operational deployments should therefore incorporate robustness testing in the model lifecycle, monitoring for drift and abuse signals, and incident-response playbooks that account for adversarial behaviour (e.g. anomalous query patterns or suspicious input distributions).

7.3 Limitations

Although the thesis provides strong evidence for the effectiveness of latent-space approaches, several limitations bound the generalisability of the results:

- **Model and architecture choices.** Alternative encoder/decoder designs, latent dimensionalities, and classifier families may yield different trade-offs between clean performance, robustness, and efficiency.
- **Threat-model coverage.** The evaluated perturbations and attacks (including gradient-based and query-driven adversarial methods) do not exhaust the space of possible attacker behaviours. In particular, functionality-preserving transformations on real binaries and more adaptive black-box strategies warrant a deeper study.
- **Transfer-learning assumptions.** Cross-dataset experiments assume partial compatibility in feature semantics and labelling; larger shifts in platform, feature extraction, or malware family composition may require more explicit alignment mechanisms.
- **Controlled experimental setting.** Laboratory evaluation does not fully capture operational constraints such as concept drift, data-collection costs, analyst workflows, and end-to-end response requirements.

7.4 Future Work Roadmap

Future work should build directly on the main findings by (i) increasing adversarial and operational realism, (ii) strengthening generalisation across domains and platforms, and (iii) enriching latent modelling to better capture malware structure. The following directions consolidate the thesis discussion into a focused roadmap.

7.4.1 Broader and more realistic adversarial evaluation

The robustness results motivate expanding evaluation beyond the current perturbation suite (e.g., standard noise corruptions and common adversarial attacks such as FGSM, PGD, HopSkipJump, and Boundary-style methods). Priority directions include:

- **Query-efficient and adaptive black-box attacks** that better reflect the attacker's constraints and reconnaissance behaviour.
- **Functionality and structure-preserving attacks** that respect executable constraints while altering feature realisations.
- **Generative-model-driven evasion** (e.g., GAN- or diffusion-assisted transformations) to probe whether learnt latent manifolds remain stable under more complex, data-manifold-constrained assaults.

On the defence side, the most promising avenues include stronger adversarial training regimens (including On the-fly example generation), ensemble-based defences, and attack-aware monitoring strategies that detect and respond to suspicious interaction patterns.

7.4.2 Advancing transfer learning and foundation-model paradigms

To improve sample efficiency and generalisation, future research should explore the following:

- **Transformer architectures** for malware-as-image representations (e.g., ViT-style models) and token/byte-level approaches where appropriate.
- **Self-supervised and contrastive pretraining** on large unlabelled malware corpora to learn representations that transfer with minimal labels.
- **Graph and multi-modal learning** that integrates complementary malware views (bytes, imports, control-flow graphs, API sequences, and metadata), enabling richer and more portable feature learning.

7.4.3 Enhancing latent-space modelling and manifold-aware analysis

Richer generative modelling can better capture malware variability and support more principled robustness analysis:

- **More expressive latent-variable models** (e.g., hierarchical VAEs, β-VAEs, normalising flows, or diffusion-inspired latent generators) to represent complex structure and improve reconstruction fidelity.
- **Disentanglement and interpretability in the latent space**, including regularisation that encourages latent factors to correspond to meaningful behavioural attributes.

- **Manifold-aware robustness probes** , such as metric-aware interpolation in latent space, to study whether classifier boundaries align with plausible malware evolution trajectories.

7.4.4 Cross-domain adaptation and few-shot generalisation

Given that domain shift emerged as a primary constraint, future work should prioritise explicit alignment mechanisms.

- **Unsupervised and semi-supervised domain adaptation** to align latent spaces across datasets, vendors, and platforms (e.g., Windows PE, Linux ELF, Android APK).
- **Meta-learning and few-shot strategies** for rapid onboarding of new malware families using small labelled sets.
- **Calibration and uncertainty estimation** to improve the reliability of decision on out-of-distribution inputs and support safer analyst classification.

7.4.5 Explainability and human-in-the-loop robustness

As models become more complex, explainability becomes essential for analyst trust and actionable response. Future work should integrate explainability methods (e.g., feature attribution, saliency, and counterfactual analysis) with latent-space reconstructions to provide interpretable evidence for detections and to support investigation workflows. An especially promising direction is to connect explanations to *robustness diagnostics*: identifying which features (or latent factors) are most sensitive to attack can guide targeted hardening and monitoring.

7.4.6 From laboratory results to operational deployment

Finally, demonstrating real-world impact requires operational validation:

- **Ensemble and meta-ensemble architectures** that combine complementary learners (e.g., latent-feature models with image or graph views) to improve robustness and coverage.
- **Increasing and diversity** of synthetic data to stress-test detectors under controlled but challenging conditions and to reduce blind spots.
- **Continual/online learning and drift mitigation**, including active learning and human-in-the-loop labelling to keep models current under evolving threats.
- **Collaborative pilot deployments** with industry partners to measure end-to-end metrics (latency, alert quality, triage cost, and resilience to evasion) and to surface deployment constraints early.
- **Federated and privacy-preserving learning** (secure aggregation, differential privacy) to enable cross-organisation collaboration without exposing sensitive telemetry.

7.5 Closing Remarks

This thesis demonstrates that latent representations offer a practical and scientifically grounded path toward malware detectors that are accurate, efficient, and more resilient to perturbation. The roadmap provided above provides a focused set of next steps to strengthen adversarial realism, improve cross-domain generalisation, and validate operational effectiveness, ultimately advancing malware detection toward deployable robustness.

Acronyms

- **AAE**: **A**dversarial **A**uto**E**ncoder
- **AE-DCNN**: **A**utoencoder **E**nhanced **D**eep **C**onvolutional **N**eural **N**etwork
- **AI**: **A**rtificial **I**ntelligence
- **ALAE**: **A**dversarial **L**atent **A**uto**E**ncoder
- **ANN**: **A**rtificial **N**eural **N**etwork
- **ANOVA**: **A**nalysis **o**f **VA**riance
- **API**: **A**pplication **P**rogramming **I**nterface
- **APK**: **A**ndroid **P**ackage **K**it
- **AUC**: **A**rea **U**nder the **C**urve
- **AUC-ROC**: **A**rea **U**nder the **R**eceiver **O**perating **C**haracteristic **C**urve
- **BODMAS**: **B**lue Hexagon **O**pen **D**ataset for **MA**lware **A**nalysis
- **C2AE**: **C**anonical-**C**orrelated **A**uto**E**ncoder
- **CNN**: **C**onvolutional **N**eural **N**etwork
- **CV**: **C**oefficient of **V**ariation
- **CVAE**: **C**onditional **V**ariational **A**uto**E**ncoder
- **DCCA**: **D**eep **C**anonical **C**orrelation **A**nalysis
- **DCNN**: **D**eep **C**onvolutional **N**eural **N**etwork
- **DLL**: **D**ynamic **L**ink **L**ibrary
- **EMBER**: **E**ndgame **M**alware **B**enchmark for **R**esearch
- **FFN**: **F**eed **F**orward **N**etwork
- **GAN**: **G**enerative **A**dversarial **N**etwork
- **GLSR**: **G**eodesic **L**atent **S**pace **R**egularization
- **GRU**: **G**ated **R**ecurrent **U**nit
- **GVAE**: **G**raph **V**ariational **A**uto**E**ncoder

- **HAVAE**: **H**ybrid **A**dversarial-**V**ariational **A**uto**E**ncoder
- **IoT**: **I**nternet **o**f **T**hings
- **JSON**: **J**ava**S**cript **O**bject **N**otation
- **KL**: **K**ullback–**L**eibler (divergence)
- **KNN**: **K**-**N**earest **N**eighbors
- **LGBM**: **L**ight **G**radient **B**oosting **M**achine
- **LIEF**: **L**ibrary to **I**nstrument **E**xecutable **F**ormats
- **LR**: **L**ogistic **R**egression
- **LSTM**: **L**ong **S**hort-**T**erm **M**emory
- **MALIMG**: **Mal**ware **Im**a**g**e Dataset
- **MLP**: **M**ulti-**L**ayer **P**erceptron
- **NB**: **N**aïve **B**ayes
- **OEM**: **O**riginal **E**quipment **M**anufacturer
- **OT**: **O**perational **T**echnologies
- **PCA**: **P**rincipal **C**omponent **A**nalysis
- **PE**: **P**ortable **E**xecutable
- **ReLU**: **Re**ctified **L**inear **U**nit
- **RF**: **R**andom **F**orest
- **RFE**: **R**ecursive **F**eature **E**limination
- **ROC**: **R**eceiver **O**perating **C**haracteristic
- **SE-AGM**: **S**tacked **E**nsemble with **A**utoencoder, **GRU**, and **MLP**
- **SVM**: **S**upport **V**ector **M**achine
- **TLS**: **T**hread-**L**ocal **S**torage
- **UIUC**: **U**niversity of **I**llinois at **U**rbana–**C**hampaign
- **VAE**: **V**ariational **A**uto**E**ncoder
- **VAEs**: **V**ariational **A**uto**E**ncoders
- **VGG16**: **V**isual **G**eometry **G**roup Net-16
- **VGG19**: **V**isual **G**eometry **G**roup Net-19
- **WGAN-GP**: **W**asserstein **G**enerative **A**dversarial **N**etwork with **G**radient **P**enalty

Glossary of Specialized Terms

- **Latent Space Representation**: A lower-dimensional space where complex data is encoded, preserving essential features and relationships from the original higher-dimensional dataset.
- **Transfer Learning**: A machine learning approach where knowledge gained from solving one problem is applied to a different but related problem, significantly reducing training time and improving accuracy.
- **Feature Engineering**: The process of selecting, transforming, and creating features from raw data to improve the performance of machine learning algorithms.
- **Dimensionality Reduction**: Techniques that reduce the number of input features or dimensions of a dataset while retaining most of its important information, thereby simplifying models and enhancing performance.
- **Hyperparameter Tuning**: The process of optimising the performance of machine learning algorithms by systematically adjusting their hyperparameters through experimentation and validation.
- **Data Augmentation**: Techniques that artificially expand the size and diversity of a training data set to enhance the robustness and generalisability of machine learning models.
- **Obfuscation**: Techniques used by malware developers to deliberately alter the structure or appearance of malicious software to evade detection by security systems.
- **Polymorphism**: The capability of malware to continuously change its identifiable characteristics, complicating detection efforts.
- **Binary Classification**: A classification task in which data are categorised into two distinct classes or groups.
- **Multi-class Classification**: A classification task involving more than two distinct categories, where the model learns to distinguish among multiple classes simultaneously.
- **Robustness (Adversarial)**: The ability of machine learning models to maintain accurate performance when faced with adversarial inputs intentionally designed to mislead or confuse the model.
- **Cross-validation**: A statistical technique used to evaluate the predictive performance of a model by partitioning the original dataset into multiple subsets, training the model on different subsets, and validating it against the remaining subsets.

- **Receiver Operating Characteristic Curve (ROC Curve)**: A graphical representation of the diagnostic ability of binary classifiers, showing the trade-off between true positive rates and false positive rates at various threshold settings.
- **Portable Executable (PE)**: A file format used for executables, object code, and DLLs in Windows operating systems, consisting of multiple structured sections such as headers, data directories, and section data.
- **Static Analysis**: The examination of software without executing it, often used in malware detection to analyse code structure and behaviour.
- **Dynamic Analysis**: The examination of software by executing it in a controlled environment to observe its behaviour and interaction with system resources.
- **Concept Drift**: A phenomenon in machine learning where the statistical properties of the target variable change over time, affecting model performance and requiring adaptive techniques.
- **Malware Detection**: The process of identifying malicious software through various static, dynamic, and hybrid analysis techniques.
- **Adversarial Machine Learning**: Techniques designed to manipulate machine learning models by introducing deceptive inputs, challenging the model's robustness and security.
- **Gradient Penalty**: A regularisation technique used in training generative adversarial networks (GANs) to stabilise training and improve the quality of generated outputs.

Bibliography

Aggarwal, P., Ahamed, S. F., Shetty, S. & Freeman, L. J. (2021), Selective targeted transfer learning for malware classification, *in* '2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)', pp. 114–120.

Ahmed, M., Afreen, N., Ahmed, M., Sameer, M. & Ahamed, J. (2023), 'An inception v3 approach for malware classification using machine learning and transfer learning', *International Journal of Intelligent Networks* pp. 11–18.

AlGarni, M. D., AlRoobaea, R., Almotiri, J., Ullah, S. S., Hussain, S. & Umar, F. (2022), 'An efficient convolutional neural network with transfer learning for malware classification', *Wireless Communications and Mobile Computing* **2022**, 4841741.

Anderson, H. S. & Roth, P. (2018), 'EMBER: An open dataset for training static pe malware machine learning models', arXiv preprint arXiv:1804.04637, 2018. Accessed: 2025-04-20.
URL: *https://arxiv.org/abs/1804.04637*

Antivirus.com (n.d.), 'Famous ransomware attacks', Antivirus.com, December 24, 2021. Accessed: 2025-04-20.
URL: *https://antivirus.com/2021/12/24/famous-ransomware-attacks/*

Asadi, N., Sarfi, A., Hosseinzadeh, M., Tahsini, S. & Eftekhari, M. (2019), 'Diminishing the effect of adversarial perturbations via refining feature representation', arXiv preprint arXiv:1907.01023. Online. Available: `https://arxiv.org/abs/1907.01023`.

Ball, J. (n.d.), 'Nhs ransomware cyber attack: What is wannacry?', The Guardian, May 12, 2017. Accessed: 2025-04-20.
URL: *https://www.theguardian.com/technology/2017/may/12/nhs-ransomware-cyber-attack-what-is-wanacrypt0r-20*

Ban, Y., Yi, J. H. & Cho, H. (2022), 'Augmenting android malware using conditional variational autoencoder for the malware family classification', *Computer Systems: Science & Engineering* **46**(2), 2215–2230.

Barakat, B. & Huang, Q. (2023), Enhancing transfer learning reliability via block-wise fine-tuning, *in* '2023 International Conference on Machine Learning and Applications (ICMLA)', Jacksonville, FL, USA, pp. 414–421.

Bhodia, N., Prajapati, P., Di Troia, F. & Stamp, M. (2019), 'Transfer learning for image-based malware classification', arXiv preprint arXiv:1903.11551. Department of Computer Science, San Jose State University. Accessed: 2025-04-20.
URL: *https://arxiv.org/pdf/1903.11551*

Biggio, B. & Roli, F. (2018), 'Wild patterns: Ten years after the rise of adversarial machine learning', *Pattern Recognition* **84**, 317–331. Online. Available: `https://arxiv.org/pdf/1712.03141`.

Bouhnine, A. (2024), Malware obfuscation and evasion, Master's thesis, Université Libre de Bruxelles. Accessed: 2025-04-20.
URL: *https://cylab.be/publications/67/download/2024-malware-obfuscation-and-evasion.pdf*

Chakraborty, A. & Kumar, S. (2023), Transfer learning-based malware classification, *in* M. Thakur, S. Agnihotri, B. S. Rajpurohit, M. Pant, K. Deep & A. K. Nagar, eds, 'Soft Computing for Problem Solving', Vol. 547 of *Lecture Notes in Networks and Systems*, Springer, Singapore.

Chen, X., Xiong, W., Liu, Z. & Li, Y. (2018), Disentangled representations for variational autoencoders in malware detection, *in* 'Proceedings of the 2018 International Joint Conference on Artificial Intelligence (IJCAI)', pp. 510–516.

Choi, A., Giang, A., Jumani, S., Luong, D. & Di Troia, F. (2024), 'Synthetic malware using deep variational autoencoders and generative adversarial networks', *EAI Endorsed Transactions on Internet of Things* **10**.

Cohen, J. (1988), *Statistical Power Analysis for the Behavioral Sciences*, 2nd edn, Lawrence Erlbaum, Hillsdale, NJ, USA.

CreativeCommons (n.d.), 'Portable executable 32-bit structure [svg]', Wikimedia Commons. Accessed: 2025-04-20.
URL: *https://commons.wikimedia.org/wiki/File:Portable_Executable_32_bit_Structure_in_SVG_fixed.svg*

Dillon, B. M., Plehn, T., Sauer, C. & Sorrenson, P. (2021), 'Better latent spaces for better autoencoders', *SciPost Physics* **11**(3), 061.

Dilokthanakul, N., Mediano, P. A., Garnelo, M., Lee, M. C., Salimbeni, H., Arulkumaran, K. & Shanahan, M. (2016), 'Deep unsupervised clustering with gaussian mixture variational autoencoders', arXiv preprint arXiv:1611.02648.

Dinh, P. V., Nguyen, D. N., Hoang, D. T., Nguyen, Q. U., Dutkiewicz, E. & Bao, S. P. (2024), 'Multiple-input auto-encoder guided feature selection for iot intrusion detection systems', arXiv preprint arXiv:2403.15511v1. Online. Available: `https://arxiv.org/abs/2403.15511`.

Frederick, R., Shapiro, J. & Calix, R. A. (2022), A corpus of encoded malware byte information as images for efficient classification, *in* '2022 16th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)', IEEE, pp. 32–36.

Galen, C. & Steele, R. (2020), Evaluating performance maintenance and deterioration over time of machine learning-based malware detection models on the ember pe dataset, *in* '2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS)', IEEE, pp. 1–7.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press.
URL: *https://mitpress.mit.edu/9780262035613/deep-learning/*

Gunduz, H. (2022), 'Malware detection framework based on graph variational autoencoder extracted embeddings from api-call graphs', *PeerJ Computer Science* **8**, e988.

Hadjeres, G., Nielsen, F. & Pachet, F. (2017), 'Glsr-vae: Geodesic latent space regularization for variational autoencoder architectures', arXiv preprint arXiv:1707.04588. Online. Available: `https://arxiv.org/abs/1707.04588`.

Hajian-Tilaki, K. (2013), 'Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation', *Caspian Journal of Internal Medicine* **4**(2), 627–635. **URL:** *https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3755824/*

Härkönen, E., Hertzmann, A., Lehtinen, J. & Paris, S. (2020), Ganspace: Discovering interpretable gan controls, *in* 'Advances in Neural Information Processing Systems (NeurIPS)', Vol. 33, pp. 9841–9850.

Hou, Z., Li, X., Li, L., Yuan, J. & Deng, K. (2022), An end-to-end raw bytes based malware classifier via self-attention residual convolutional network, *in* '2022 IEEE 8th International Conference on Computer and Communications (ICCC)', IEEE, pp. 1666–1670.

Hussain, M. J., Shaoor, A., Baig, S., Hussain, A. & Muqurrab, S. A. (2022), A hierarchical based ensemble classifier for behavioral malware detection using machine learning, *in* '2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST)', IEEE, pp. 702–706.

Jahan, M., Kalogerakis, E., Maji, A. & Chaudhuri, S. (2021), 'Semantics-guided latent space exploration for shape generation', *Computer Graphics Forum* **40**(5), 145–157.

Karnouskos, S. (2011), Stuxnet worm impact on industrial cyber-physical system security, *in* 'IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society', IEEE, pp. 4490–4494.

Khan, R., Maynard, P., McLaughlin, K., Laverty, D. & Sezer, S. (2016), Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid, *in* '4th International Symposium for ICS & SCADA Cyber Security Research', IEEE, pp. 53–63.

Khazaie, V. R., Wong, A., Jewell, J. T. & Mohsenzadeh, Y. (2022), 'Anomaly detection with adversarially learned perturbations of latent space', arXiv preprint arXiv:2207.01106. Online. Available: `https://arxiv.org/abs/2207.01106`.

Kim, D., Park, J. & Lee, I. (2021), Adversarial perturbation-based data augmentation for time-series data, *in* 'Proceedings of the 2021 ACM Conference on Knowledge Discovery and Data Mining (KDD)', Singapore, pp. 1534–1542.

Kim, J.-Y. & Cho, S.-B. (2022), 'Obfuscated malware detection using deep generative model based on global/local features', *Computers & Security* **112**, 102501.

Kingma, D. P. (2013), 'Auto-encoding variational bayes', arXiv preprint arXiv:1312.6114.

Kiran, V. U. (2024), 'Havae — an advanced approach for malware detection using deep learning', *International Journal for Research in Applied Science and Engineering Technology (IJRASET)* **12**(3), 2740–2746.

Kumar, S., Meena, S., Khosla, S. & Parihar, A. S. (2021), Ae-dcnn: Autoencoder enhanced deep convolutional neural network for malware classification, *in* 'IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)', pp. 1–7.

Kwan, L. M. (2022), Markov image with transfer learning for malware detection and classification, *in* 'TENCON 2022–2022 IEEE Region 10 Conference (TENCON)', IEEE, pp. 1–6.

Lakens, D. (2013), 'Calculating and reporting effect sizes to facilitate cumulative science: A practical primer for t-tests and anovas', *Frontiers in Psychology* **4**, 863.
URL: *https://doi.org/10.3389/fpsyg.2013.00863*

Ling, X., Wu, L., Zhang, J., Qu, Z., Deng, W., Chen, X., Qian, Y., Wu, C., Ji, S., Luo, T., Wu, J. & Wu, Y. (2023), 'Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art', Institute of Software, Chinese Academy of Sciences, Zhejiang University, Pinterest.

Liu, X., Zou, Y., Kong, L., Diao, Z., Yan, J., Wang, J., Li, S., Jia, P. & You, J. (2018), Data augmentation via latent space interpolation for image classification, *in* '2018 24th International Conference on Pattern Recognition (ICPR)', Beijing, China, pp. 728–733.

Luo, X., Zhang, L. & Yu, W. (2020), 'Generative adversarial networks and variational autoencoder based hybrid model for malware generation and detection', *IEEE Access* **8**, 116303–116315.

Mandiant (n.d.), 'Pe file infecting malware', `https://www.mandiant.com/resources/blog/pe-file-infecting-malware-ot`. Accessed: 2025-04-20.

Manjunatha, V. D. & Ramesh, R. (2023), 'Machine learning in malware detection: A survey of analysis techniques', *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)* **12**(4), 204. Accessed: 2025-04-20.
URL: *https://www.researchgate.net/publication/378300286*

Marastoni, N., Giacobazzi, R. & Dalla Preda, M. (2021), 'Data augmentation and transfer learning to classify malware images in a deep learning context', *Journal of Computer Virology and Hacking Technology* .

Masud, M. T., Keshk, M., Moustafa, N., Linkov, I. & Emge, D. K. (2024), 'Explainable artificial intelligence for resilient security applications in the internet of things', *IEEE Open Journal of the Communications Society* .

MDAutoEncoder (2023), 'Anomaly detection autoencoder model for malware detection', `https://github.com/Navy10021/MDAutoEncoder`. Accessed: 2025-04-20.

Michelis, M. Y. & Becker, Q. (2021), 'On linear interpolation in the latent space of deep generative models', arXiv preprint arXiv:2105.03663. Online. Available: `https://arxiv.org/abs/2105.03663`.

MicroAge (n.d.), 'Confronting the next wave of cyber threats: The rise of ai-generated polymorphic malware', MicroAge.ca, 2024. Accessed: 2024-09-27.
URL: *https://microage.ca/mts/confronting-the-next-wave-of-cyber-threats-the-rise-of-ai-generated-polymorphic-malware/*

Microsoft Defender Antivirus (2023), 'Advanced technologies at the core of microsoft defender antivirus', `https://learn.microsoft.com/en-us/defender-endpoint/adv-tech-of-mdav`. Accessed: 2025-04-20.

Mohamed, A. A., Al-Saleh, A., Sharma, S. K. & Tejani, G. G. (2025), 'Zero-day exploits detection with adaptive wavepca-autoencoder (awpa) and adaptive hybrid exploit detection network (ahednet)', *Scientific Reports* **15**, 4036.
URL: *https://www.nature.com/articles/s41598-025-87615-2*

Naeem, M. R. (2023), 'Malimg dataset.zip', figshare.

Oring, A., Yakhini, Z. & Hel-Or, Y. (2021), Autoencoder image interpolation by shaping the latent space, *in* 'Proceedings of the 38th International Conference on Machine Learning (ICML)', Vol. 139, pp. 8281–8290. Online. Available: `https://proceedings.mlr.press/v139/oring21a.html`.

Otokwala, U. J., Petrovski, A. & Kalutarage, H. (2024), 'Optimized common features selection and deep-autoencoder (ocfsda) for lightweight intrusion detection in internet of things', *International Journal of Information Security* **23**(8), 2559–2581.
URL: *https://doi.org/10.1007/s10207-024-00855-7*

Oyama, Y., Miyashita, T. & Kokubo, H. (2019), Identifying useful features for malware detection in the ember dataset, *in* '2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)', IEEE, pp. 360–366.

Pan, S. J. & Yang, Q. (2010), 'A survey on transfer learning', *IEEE Transactions on Knowledge and Data Engineering* **22**(10), 1345–1359.
URL: *https://doi.org/10.1109/TKDE.2009.191*

Panda, P., C U, O. K., Marappan, S., Ma, S., S, M. & Veesani Nandi, D. (2023), 'Transfer learning for image-based malware detection for iot', *Sensors* **23**(6), 3253.

Pant, D. & Bista, R. (2021), Image-based malware classification using deep convolutional neural network and transfer learning, *in* '2021 3rd International Conference on Advanced Information Science and System (AISS)', p. 9.

Pidhorskyi, S., Adjeroh, D. & Doretto, G. (2020), Adversarial latent autoencoders, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)', Seattle, WA, USA, pp. 14104–14113.

Pramanik, S. & Teja, H. (2019), Ember—analysis of malware dataset using convolutional neural networks, *in* '2019 Third International Conference on Inventive Systems and Control (ICISC)', IEEE, pp. 286–291.

Pratama, H. Y. & Sidabutar, J. (2022), Malware classification and visualization using efficientnet and b2img algorithm, *in* '2022 International Conference on Advanced Computer Science and Information Systems (ICACSIS)', IEEE, pp. 75–80.

Priya, V. & Sathya Sofia, A. (2023), Review on malware classification and malware detection using transfer learning approach, *in* '2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)', IEEE, pp. 1042–1049.

Rathod, V., Parekh, C. & Dholariya, D. (2021), Ai & ml based anomaly detection and response using ember dataset, *in* '2021 9th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)', IEEE, pp. 1–5.

Ryan, N. (2023), 'Researchers develop adversarial training methods to improve machine learning-based malware detection software', `https://www.cylab.cmu.edu/news/2023/09/13-adversarial-training-for-malware-detection.html`. Accessed: 2025-04-20.

Sai, P. M. D., Amasala, L., Bhimavarapu, T. S. & Garikipati, G. C. (2023), 'A malware classification survey on adversarial attacks and defenses', arXiv preprint arXiv:2312.09636. Accessed: 2025-04-20.
URL: *https://arxiv.org/pdf/2312.09636*

Saxe, J. & Berlin, K. (2017), 'Deep neural network based malware detection using two dimensional binary program features', Invincea Labs, LLC.

Sihag, V., Vardhan, M. & Singh, P. (2021), 'A survey of android application and malware hardening', *Computer Science Review* **39**, 100365.

Souri, A. & Hosseini, R. (2018), 'A state-of-the-art survey of malware detection approaches using data mining techniques', *Human-Centric Computing and Information Sciences* **8**(3).

Stutz, D. (2020), 'On-manifold adversarial examples', David Stutz's Blog, Feb. 2020. Online. Available: `https://davidstutz.de/on-manifold-adversarial-examples/`.

Taylor, T. & Eleyan, A. (2021), Using variational autoencoders to increase the performance of malware classification, *in* '2021 International Symposium on Networks, Computers and Communications (ISNCC)', pp. 1–6.

Venkatraman, S., Alazab, M. & Vinayakumar, V. R. (2024), 'A hybrid deep learning image-based analysis for effective malware detection', Melbourne Polytechnic, Charles Darwin University, Amrita Vishwa Vidyapeetham. Online. Available: `https://ris.cdu.edu.au/ws/portalfiles/portal/25699590/24800196.pdf`.

Voynov, A. & Babenko, A. (2020), Unsupervised discovery of interpretable directions in the gan latent space, *in* 'Proceedings of the 37th International Conference on Machine Learning (ICML)', Vol. 119, pp. 9786–9796.

Wang, Q., Yan, H., Zhao, C., Mei, R., Han, Z. & Zhou, Y. (2022), Measurement of malware family classification on a large-scale real-world dataset, *in* '2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)', IEEE, pp. 1390–1397.

Wong, E. & Kolter, J. Z. (2020), 'Learning perturbation sets for robust machine learning', arXiv preprint arXiv:2007.08450. Online. Available: `https://arxiv.org/abs/2007.08450`.

Yang, G., Fei, N., Ding, M., Liu, G., Lu, Z. & Xiang, T. (2021), L2m-gan: Learning to manipulate latent space semantics for facial attribute editing, *in* '2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)', Nashville, TN, USA, pp. 15026–15035.

Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A. & Wang, G. (2021), Bodmas: An open dataset for learning based temporal analysis of pe malware, *in* '4th Deep Learning and Security Workshop'.

Yeh, C.-K., Wu, W.-C., Ko, W.-J. & Wang, Y.-C. F. (2017), Learning deep latent spaces for multi-label classification, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 31.

Yüksel, O. K., Stich, S. U., Jaggi, M. & Chavdarova, T. (2021), Semantic perturbations with normalizing flows for improved generalization, *in* '2021 IEEE/CVF International Conference on Computer Vision (ICCV)', Montreal, QC, Canada, pp. 1267–1276.

Zahoora, U., Khan, A., Rajarajan, M., Khan, S. H., Asam, M. & Jamal, T. (2022), 'Ransomware detection using deep learning based unsupervised feature extraction and a cost sensitive pareto ensemble classifier', *Scientific Reports* **12**(1), 15647.

Zhang, J. (2019), 'Machine learning with feature selection using principal component analysis for malware detection: A case study', arXiv preprint arXiv:1902.03639. Senior Threat Researcher, Sophos. Accessed: 2025-04-20.
URL: *https://arxiv.org/pdf/1902.03639*

Zhang, Z. & Xu, J. (2019), Latent space disentanglement in language gans: Use of a variational autoencoder, *in* 'NeurIPS Workshop on Learning Disentangled Representations'.

Zhao, Z., Yang, S. & Zhao, D. (2023), 'A new framework for visual classification of multi-channel malware based on transfer learning', *Applied Sciences* **13**(4), 2484.

Zhu, J., Jang-Jaccard, J., Welch, I., Al-Sahaf, H., Camtepe, S. & Dunmore, A. (2024), 'Relation-aware siamese denoising autoencoder for malware few-shot classification', arXiv preprint arXiv:2411.14029. Accessed: 2025-04-20.
URL: *https://arxiv.org/html/2411.14029v1*

Appendix A: Datasets

7.6 Introduction

The proper study of malware classification requires research database corpora that cover *distinct representation paradigms*, *multiple time periods*, and *heterogeneous threat families*.

This thesis relies on three open-source benchmarks for its research. The research includes three benchmark datasets that are listed below:

1. EMBER 2018—a large-scale, feature-vector corpus for static *Portable Executable* (PE) files Anderson & Roth (2018);

2. BODMAS 2021—a chronologically ordered PE collection with fine-grained family labels Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021); and

3. MALIMG— an image-based dataset in which binaries are rendered as grayscale byte plots, originally published by Nataraj *et al.* (2011) and repackaged on Figshare by Naeem *et al.* (2020) Naeem (2023).

The main features of the three datasets are presented in Table 7.1. The remainder of the chapter presents the process of data acquisition, describes the structure, and explains the preprocessing steps that each corpus received.

Table 7.1: The statistics of three datasets which are presented in this paper. The "Feature modality" column indicates whether the in the raw instances are numeric vectors (STATIC), raw bytes, or in images.

Dataset	Year	Samples	Malware / Benign	Feature modality
EMBER	2018	1 100 000	400k / 400k*	2 381-d static vector
BODMAS	2021	134 435	57k / 77k	2 381-d static vector
MALIMG	2011[1]	9 339	malware only	256×256 grayscale image

7.7 EMBER 2018

7.7.1 Origin and Motivation

EMBER (*Endgame Malware BEnchmark for Research*) Anderson & Roth (2018) was the first openly licenced static feature corpus - large enough to train complex models comparable to those used in industry.

[1]Original dataset by Nataraj *et al.* (2011); the version used here is a Figshare repackaging by Naeem *et al.* (2020).

The EMBER project was established to solve the "data silos" that limited academic–industrial collaboration and to create a reproducible baseline that is comparable to ImageNet in computer vision.

7.7.2 File Layout

The dataset is distributed as two JSONL files (`train_features.jsonl`, `test_features.jsonl`) and two CSV label files (`train_labels.csv`, `test_labels.csv`). Every line in the dataset represents a binary file, while the feature object includes eight groups which amount to exactly 2381 numeric features. Crucial meta-fields are :

- `sha256` – unique file hash;
- `appeared` – coarse first-seen timestamp (month granularity);
- `label` $\in \{0, 1, -1\}$ – benign, malicious, or unlabelled.

7.7.3 Pre-processing

The following pre-processing steps were used:

a) Columns with a single unique value or less than 1 % non-zero entries were dropped (76 features in total).

b) Categorical token lists `section_names` and `imported_libraries` were frequency-encoded; the top 32 tokens explained > 93 % of all occurrences.

c) The final matrix was standardised to zero mean and unit variance prior to PCA or model fitting.

7.8 BODMAS 2021

7.8.1 Dataset Rationale

BODMAS (Blue Hexagon–UIUC Dataset for Malware Analysis at Scale) was extended by Yang, Ciptadi, Laziuk, Ahmadzadeh & Wang (2021). in 2021 to include two essential features over EMBER: (i) the logging of exact first and last-seen dates for every binary and (ii) the annotation of 581 distinct malware families which makes it possible to fine-grained evaluate the performance of the cross-family generalisation.

7.8.2 Structure

In the data are included a single comma-separated table file called "features.csv" and a ZIP archive of the raw files.

The feature vector consists of 2381 dimensions which are the same as those of the EMBER dataset. Additional columns include

- `first_seen` and `last_seen` (Unix epoch);
- `family` (string, *e.g.*, `Zeus` or `Emotet`);
- `label` $\in \{0, 1\}$.

7.8.3 Pre-processing Highlights

a) Family names were mapped to integer indices; families with fewer than ten samples were merged into an OTHER bucket.

b) Histogram-encoded section and import features are already numeric and required no tokenisation.

c) A chronological train–test cut at `2020-05-01` was chosen to emulate concept drift, leaving a six-month hold-out window.

7.9 MALIMG

7.9.1 Historical Context

The MALIMG dataset is older than both EMBER and BODMAS and was the first dataset to propose the idea of "malware as image". Every Windows binary is read byte-wise and reshaped into a fixed-width greyscale image whose texture captures structural patterns of the executable. The Figshare re-release of Naeem (2023) provides 9339 samples across 25 malware families.

7.9.2 Data Format

The system stores each entry as a `.png` image that measures 256×256 pixels (following zero-padding or truncation). The labels exist in the directory names through encoding. `.../Kelihos_Hlux/001.png`. The dataset contains no benign binaries because it operates as a multi-class malware system.

7.9.3 Image Pipeline

a) The transformation of images into floating-point tensors occurred with values ranging between 0 and 1. The system conducted a channel-wise z-score normalisation. The CNN baselines received data augmentation through random horizontal flips and small crops during their training process. Images were replicated to three channels by the experiment to ensure compatibility with ImageNet-pre-trained backbones.

7.10 Comparative Remarks

Each of the three corpora presents unique characteristics which work together:

- The two modalities between static vectors and raw-byte images enable researchers to evaluate transfer learning across different modalities. The time span of EMBER (2017–2018) along with BODMAS (2019–2020) enables the evaluation of the robustness of the concept drift. The family label precision in BODMAS and MALIMG goes beyond EMBER since it focusses on binary benign/malicious classification.

These datasets create a solid foundation for the hybrid latent-space and transfer-learning investigations.

7.11 Exploratory Data Analysis and Dimensionality Reduction

This section documents the exploratory data analysis (EDA) steps performed on the static feature representations used throughout the thesis and motivates the dimensionality choices later adopted for latent-space modelling. The goal is twofold: (i) to ensure data quality and reduce risks of leakage or spurious performance, and (ii) to understand the statistical structure of the feature space to inform principled compression.

7.11.1 Datasets and Feature Representation

Two large-scale static malware datasets are used in the EDA reported here:

- **BODMAS**: samples labelled 134 435 with 2381 static PE-derived features.
- **EMBER**: labelled splits with 600 000 training samples and 200 000 test samples, each described by 2381 static features (same feature dimensionality as above).

The feature space includes multiple semantic groups (e.g., byte-level histograms, header metadata, import/export statistics, section statistics, and data-directory fields). These groupings are useful for diagnostic EDA (e.g. to confirm that discriminative signal is not dominated by a single artefact subset) and for interpretability analyses.

7.11.2 Dataset-Specific Importance and Research Role

Because malware detection results can be highly dependent on the data set, the thesis uses multiple datasets with complementary properties. Each dataset contributes a different kind of evidence: large-scale statistical validity, richer semantic metadata, and cross-modality stress testing.

EMBER: Large-Scale, Standardized Static-Feature Benchmark

EMBER is used as a *primary large-scale benchmark* for static malware detection. Its importance in this thesis is threefold.

- **Scale for statistically reliable conclusions;** With hundreds of thousands of labelled examples in both training and test splits, EMBER enables stable estimation of ROC AUC and other metrics under repeated stress tests (noise, adversarial perturbations, and ablations). This reduces the risk that the observed gains are due to a small-sample variance.
- **A strong baseline for compression studies.** The EMBER 2381-dimensional engineered feature space provides a concrete setting in which to compare (i) *linear* compression (PCA) and (ii) *nonlinear* compression (e.g. VAE). The PCA explained-variance curve and the Logistic Regression AUC-vs. Dimension sweep (Figures 7.1 – 7.2) provide an interpretable foundation for selecting latent dimensionalities.
- **Relevance to real-world static scanners;** The feature groups (header/import/-section statistics, histograms, etc.) approximate the kinds of signals used in many production-grade static detectors. As a result, the robustness improvements achieved on EMBER are meaningful beyond a purely academic setting.

Within the thesis, EMBER is therefore used to: (i) establish baseline accuracy for classical and deep models, (ii) quantify the accuracy–compression trade-off, and (iii) perform computationally intensive robustness studies at scale.

BODMAS: Richer Semantics, Family Metadata, and Cross-Dataset Generalisation

BODMAS complements EMBER by introducing *different data collection conditions* and *additional semantic metadata* (family and category labels). This makes it valuable for testing whether learnt representations generalise beyond a single benchmark distribution.

- **Distributional diversity.** Even when the dimensionality of the features is the same (2381), BODMAS is not simply "another split" of EMBER. Differences in collection pipelines, family composition, and prevalence of specific malware behaviours can induce meaningful distribution shifts. Models that appear strong on a single dataset can degrade substantially under such shifts.
- **Family/category metadata for deeper analysis.** Family and category fields enable analyses that go beyond binary benign/malicious classification (e.g., whether the latent space clusters by family, whether robustness varies by category, and whether attacks transfer between families). This supports interpretation and helps to diagnose *why* a model is successful or fails.
- **Leakage-aware transfer evaluation.** The discovery of overlapping hashes across EMBER and BODMAS (EMBER $\cap$ BODMAS) provides a concrete reason to build leakage controls into any cross-dataset evaluation. The handling of overlap correctly strengthens the credibility of transfer-learning and generalisation claims.

In the thesis, BODMAS is therefore used to: (i) evaluate generalisation and robustness under data set shift, (ii) support family-aware diagnostics of learnt representations, and (iii) validate that improvements are not benchmark-specific.

MALIMG: Cross-Modality Stress Test via Image-Space Representations

MALIMG is included as an *image-modality counterpoint* to the tabular static-feature datasets. Although the underlying objective remains malware classification, MALIMG changes the input representation and inductive biases of the model class.

- **Representation diversity;** Image-space malware (e.g., greyscale mappings of binaries) emphasises texture-like patterns and local correlations, which are naturally exploited by CNNs and vision-style architectures. This is a regime fundamentally different from the sparse engineered feature vectors.
- **Robustness under different perturbation geometry.** Noise and adversarial perturbations behave differently in image space (e.g., bounded ℓ_p perturbations on pixel intensities) than in tabular space. Demonstrating robustness benefits in both settings strengthens the claim that the methodology is broadly applicable.
- **Architectural generality.** MALIMG supports experiments with CNNs and Transfer learning, allowing the thesis to test for generalisation or remain effective across model families.

In summary, MALIMG is used as a cross-modality validation dataset that probes the portability of transfer learning methods beyond the specific engineered-feature pipelines used for PE static detection.

Why multiple datasets matter. Using EMBER, BODMAS, and MALIMG together reduces the risk of over-claiming results that only hold for a single benchmark. Improvements that persist across (i) large-scale standardised features (EMBER), (ii) distributionally distinct data with richer metadata (BODMAS), and (iii) a different input modality (MALIMG) provide stronger evidence that the proposed latent representation learning is both *effective* and *robust* under realistic variability.

7.11.3 Data Integrity and Leakage Controls

Hash uniqueness and cross-dataset overlap. To mitigate inadvertent evaluation leakage, file hashes were inspected for duplicates. No duplicates were observed *within* EMBER or within BODMAS; however, there were 41 overlapping hashes across EMBER and BODMAS (EMBER $\cap$ BODMAS). In all transfer-learning or cross-dataset experiments, these overlapping samples should be excluded or handled explicitly (e.g., removed from the target test set) to prevent train–test contamination.

Duplicate feature vectors and label conflicts (BODMAS). A feature-hash scan over BODMAS processed 134 435 samples and found 134 428 unique feature hashes, indicating a very small number of identical feature vectors. Importantly, no identical feature vectors with conflicting labels were found, which reduces the likelihood of label-noise artefacts in downstream modelling.

7.11.4 Class Balance and Metadata Distributions

BODMAS label distribution. BODMAS exhibits moderate imbalance:

- benign: 77 142 (57.38 %)
- malicious: 57 293 (42.62 %)

BODMAS also contains rich malware-family and category metadata (e.g., 582 unique families excluding missing values). For example, the most frequent families include `sfone`, `wacatac`, and `upatre`. The category counts are dominated by `trojan` (29 972), `worm` (16 697), and `backdoor` (7331), with smaller counts for `downloader`, `ransomware`, and `dropper`.

EMBER label distribution. EMBER is balanced by construction in the labelled splits:

- Train (labelled): 300 000 benign and 300 000 malicious
- Test (labelled): 100 000 benign and 100 000 malicious

Balanced splits are advantageous for AUC-based model selection because they reduce sensitivity to prevalence shifts; nevertheless, distributional shift (e.g., temporal drift) remains a practical concern and is addressed later via robustness evaluations and out-of-distribution analyses.

7.11.5 Sparsity, Constants, and Univariate Diagnostics

High-dimensional static feature vectors often contain (i) constant features (no variance), and (ii) near-always-zero features (extreme sparsity). Both can degrade numerical conditioning and increase compute without providing discriminative signal.

BODMAS (univariate sparsity audit). Starting from 2381 features, 49 constant features and 353 features with $< 1\,\%$ non-zero entries were removed, leaving 2028 features for subsequent analysis.

EMBER (univariate sparsity audit). Starting from 2381 features, 46 constant features and 380 features with $< 1\,\%$ non-zero entries were removed, leaving 2001 features. This reduction improves stability for both linear baselines and latent representation learning.

7.11.6 Correlation Structure and Redundancy Pruning

Many engineered static features are partially redundant (e.g., correlated counts derived from related header fields). To reduce redundancy, a Spearman correlation analysis was performed, pruning features with strong monotonic dependence.

BODMAS (Spearman pruning). Using a sample of 50 000 instances, 149 features with $|\rho| > 0.95$ were removed, resulting in 1879 features.

EMBER (Spearman pruning). A Spearman correlation matrix computation removed 117 features with $|\rho| > 0.95$, producing 1884 features after pruning (from 2001 post-sparsity features).

Effect-size diagnostics (EMBER). In addition to correlation pruning, group-level means and effect sizes (Cohen's d) were calculated on representative samples to identify which feature groups most separate benign from malicious distributions. These diagnostics support later chapters where latent representations are evaluated for whether they preserve meaningful semantic structure rather than superficial artefacts.

7.11.7 Outlier and Duplicate Diagnostics

Outliers (BODMAS). An Isolation Forest analysis identified 2646 outliers out of 134 435 samples (1.97 %). Outliers were not removed by default, as they may correspond to rare but real malware behaviours. Instead, the outlier rate is reported as part of the characterisation of the data set, and robustness experiments later evaluate the sensitivity to such atypical points.

Duplicate and label-conflict scan (BODMAS). As noted earlier, no identical feature vectors with conflicting labels were found, reducing the risk that the model is trained on mutually contradictory examples.

7.11.8 Most Discriminative and Most Important Features (EMBER and BODMAS)

Beyond global dataset statistics, it is useful to identify which input dimensions appear most discriminative (large between-class separation) and which appear most "important" to a fitted baseline model. In this section, we report:

- **Effect size ranking (Cohen's d);** Features are ranked by $|d|$ computed on a representative sample, where

$$d = \frac{\mu_m - \mu_b}{s_p}, \quad s_p = \sqrt{\frac{(n_m - 1)s_m^2 + (n_b - 1)s_b^2}{n_m + n_b - 2}},$$

  with μ_m, s_m and μ_b, s_b the malware and the benign means/standard deviations, respectively.

- **Model-based ranking (Random Forest Gini importance).** Features are ranked by Random Forest impurity-based importance (mean decrease in impurity). Although this measure can be biassed toward features with many split points, it remains a useful first-order diagnostic when interpreted alongside effect sizes and group-level summaries.

Feature-index interpretation. Both datasets use the canonical 2381-dimensional EMBER-style feature ordering. For clarity, we also report the feature *group* associated with each index using the ranges: byte histogram (0–255), byte-entropy histogram (256–511), string features (512–615), general file info (616–625), PE header info (626–687), section info (688–942), imported functions (943–2222), exported functions (2223–2350), and data directories (2351–2380).

Feature group descriptions (why these groups matter). The engineered static feature space can be interpreted at the level of *groups*, each capturing a different aspect of the executable's structure and behaviour as observed without running it:

- **Byte histogram (0–255).** Frequency of byte values over the file. These distributions often shift under packing/obfuscation, embedded data blobs, or atypical encoding patterns. While not semantically "aware", they provide strong coarse fingerprints of file composition.

- **Byte-entropy histogram (256–511).** Joint statistics of byte values across entropy bins. High-entropy regions can indicate compression or encryption (common in packed malware), whereas low-entropy regions reflect structured code/data. This group frequently contributes to separability when malware differs from benign software in packing prevalence.

- **String features (512–615).** Counts and summary statistics of printable strings (e.g., length statistics, frequency patterns). Malware often contains URLs, registry keys, command strings, and obfuscation artefacts; benign applications also contain strings, but distributions and patterns can differ systematically.

- **General file information (616–625).** Global characteristics such as file size and other coarse metadata signals. These variables can be informative but are also more susceptible to dataset-specific artefacts, so their influence is interpreted cautiously and validated with cross-dataset experiments.

- **PE header information (626–687).** Structural metadata from the PE/COFF headers (e.g., subsystem and compilation flags, alignment-related fields, characteristics). These reflect how the binary is laid out and can encode meaningful differences between benign toolchains and malware build pipelines.

- **Section information (688–942).** Statistics over sections (e.g., counts, sizes, entropy summaries, proportions). Sections capture the internal layout of code and data; malware may have unusual section naming conventions, anomalous size ratios, or high-entropy sections indicative of packing.

- **Imported functions (943–2222).** Summary statistics of imported APIs, representing *capability usage* (file I/O, process injection primitives, networking, crypto, etc.). Imports are among the most semantically meaningful static signals, though they can be partially hidden by dynamic resolution.

- **Exported functions (2223–2350).** Export-related statistics, often useful for identifying library-like binaries and certain families. Exports are typically less informative than imports for generic malware detection, but can be discriminative in specific subsets.
- **Data directories (2351–2380).** Presence/size summaries of PE data directories (e.g., import table, resource table, TLS, debug). These capture structural "hooks" into optional PE features that can differ markedly across benign and malicious populations.

This group-level view is useful later in the thesis for two reasons: (i) it supports *interpretability* of what the model is learning (e.g., whether robustness gains coincide with preserving semantically meaningful groups such as imports/headers), and (ii) it helps detect potential dataset artefacts (e.g., over-reliance on coarse file-size signals) by checking whether the same groups remain salient under cross-dataset evaluation.

EMBER: Top-10 Features

Table 7.2 shows the top-10 EMBER features by Random Forest importance, and Table 7.3 shows the top-10 features by $|d|$.

Table 7.2: EMBER top-10 features by Random Forest Gini importance (higher is more important).

Feature idx	Group	Importance
2359	data_directories	0.018 437
637	header_info	0.012 391
2360	data_directories	0.010 939
508	byte_entropy_histogram	0.009 614
510	byte_entropy_histogram	0.008 662
2364	data_directories	0.007 545
499	byte_entropy_histogram	0.007 249
506	byte_entropy_histogram	0.006 588
658	header_info	0.006 442
503	byte_entropy_histogram	0.006 437

Table 7.3: EMBER top-10 features by absolute Cohen's d (direction indicates malware vs benign mean shift).

Feature idx	Group	Cohen's d
637	header_info	1.022 122
655	header_info	−0.785 237
658	header_info	0.754 155
1060	imported_functions	0.732 819
654	header_info	−0.694 030
114	byte_histogram	−0.670 476
618	general_file_info	−0.662 363
116	byte_histogram	−0.645 482
111	byte_histogram	−0.638 929
496	byte_entropy_histogram	0.627 791

Interpretation (EMBER). The top-ranked features concentrate in `header_info`, `data_directories`, and histogram-based groups. This is consistent with the intuition that static detectors exploit (i) structural PE metadata and (ii) coarse-grained byte/entropy distributional signatures. Importantly, this does *not* imply causality; rather, it indicates which dimensions the baseline model uses most strongly under the given training distribution.

BODMAS: Top-10 Features

Table 7.4 shows the top-10 BODMAS features by Random Forest importance, and Table 7.5 shows the top-10 features by $|d|$.

Table 7.4: BODMAS top-10 features by Random Forest Gini importance (higher is more important).

Feature idx	Group	Importance
658	header_info	0.023 550
637	header_info	0.022 070
2359	data_directories	0.019 722
620	general_file_info	0.017 248
613	string_features	0.017 114
2360	data_directories	0.016 843
2355	data_directories	0.010 352
626	header_info	0.010 244
640	header_info	0.010 011
930	section_info	0.009 593

Table 7.5: BODMAS top-10 features by absolute Cohen's d (direction indicates malware vs benign mean shift).

Feature idx	Group	Cohen's d
658	header_info	1.379 904
637	header_info	1.119 212
623	general_file_info	−1.039 118
642	header_info	−0.989 746
618	general_file_info	−0.983 091
641	header_info	−0.929 415
622	general_file_info	−0.898 291
640	header_info	−0.798 740
748	section_info	0.758 103
621	general_file_info	−0.755 575

Interpretation (BODMAS). Similar to EMBER, the most influential features are dominated by `header_info` and `data_directories`, with additional signal in string and section statistics. The overlap in high-importance regions across datasets suggests that these feature groups capture broadly useful static characteristics, which motivates later latent-space experiments that aim to preserve such signal while improving robustness.

7.11.9 Principal Component Analysis

PCA provides an interpretable linear baseline for compression and a useful diagnostic for intrinsic dimensionality. Given a central data matrix $X \in \mathbb{R}^{n \times d}$, PCA finds orthonormal directions $W_k \in \mathbb{R}^{d \times k}$ that maximise projected variance, producing $Z = XW_k$. The explained variance ratio of component i is $\lambda_i / \sum_{j=1}^{d} \lambda_j$, where λ_i are the eigenvalues of the sample covariance.

BODMAS explained variance. For BODMAS, the number of components required to reach at least the cumulative explained variance 95 % was $k = 101$ (computed from the PCA scree/cumulative curve). This suggests that while the raw space is 2381-dimensional, much of the variance lies in a substantially lower-dimensional subspace.

EMBER explained variance (cumulative curve). Figure 7.1 shows the cumulative explained variance of PCA on EMBER features (with a red reference line at 32 components). The curve exhibits a steep initial rise and then progressively diminishing returns. In this experiment, 32 components capture a large majority of the variance (visually ≈ 0.88), while pushing to 128 components approaches the 95 % threshold.

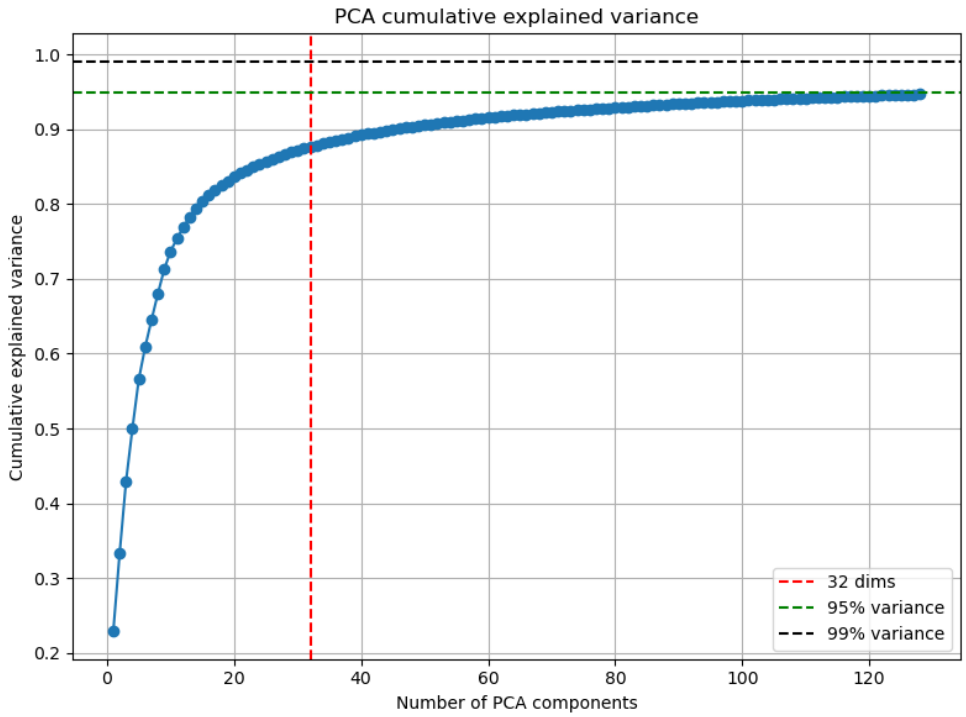

Figure 7.1: EMBER PCA cumulative explained variance. The red dashed line marks $k = 32$. The curve rises rapidly for small k and then flattens, indicating diminishing returns for additional components.

7.11.10 Downstream Predictive Performance vs. PCA Dimension (EMBER)

To complement variance-based analysis, PCA dimensionality was evaluated using a downstream classifier: Logistic Regression trained on PCA-projected features at $k \in \{8, 16, 32, 64, 128\}$. This provides an operational measure of how much predictive signal is preserved at each compression level.

Protocol. A stratified subset of 200 000 labelled training instances was used to fit PCA (for tractability), and Logistic Regression was trained/evaluated on fixed train/test splits. Performance is reported as ROC AUC, along with training and evaluation time.

Table 7.6: EMBER Logistic Regression performance under PCA compression.

PCA dims (k)	ROC AUC	Train+Eval time (s)
8	0.820 619	12.485
16	0.846 302	2.329
32	0.868 883	2.678
64	0.902 436	3.297
128	0.923 183	5.006

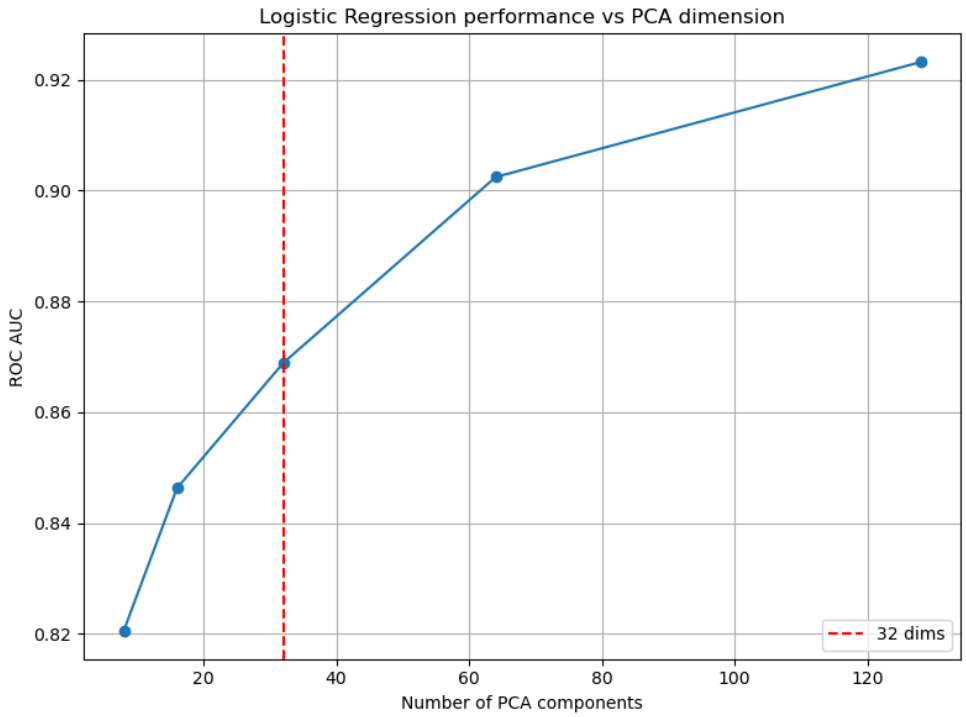

Figure 7.2: EMBER Logistic Regression ROC AUC as a function of PCA dimension. The red dashed line marks $k = 32$.

Interpretation. Performance improves monotonically with higher k, as expected: retaining more components preserves a more discriminative signal. Importantly, $k = 32$ provides a substantial improvement over $k = 16$ while remaining much smaller than the original feature dimension. The trade-off curve in Table 7.6 and Figure 7.2 serves as a practical baseline for the compression levels later used in nonlinear latent representation learning.

7.11.11 Rationale for a 32-Dimensional Latent Space

The remainder of this thesis frequently adopts a 32-dimensional latent code ($k = 32$) for representation learning (e.g., VAE-based compression). This choice is motivated by the above EDA and by the downstream experimental constraints:

1. **Compression efficacy.** Reducing from $d = 2381$ to $k = 32$ provides a $\approx 74\times$ reduction in dimensionality, which meaningfully decreases storage and compute for large-scale experiments.

2. **Variance structure.** The EMBER PCA cumulative-variance curve indicates that 32 components already capture a large majority of the variance (Figure 7.1), suggesting that much of the data mass lies on a comparatively low-dimensional manifold.

3. **Predictive signal retention.** Logistic Regression trained on PCA($k = 32$) achieves ROC AUC ≈ 0.869 (Table 7.6), demonstrating that strong classification performance is possible even under aggressive compression.

4. **Robustness and tractability.** Many robustness analyses (noise stress tests, adversarial attacks, manifold/geodesic operations) scale poorly with dimensionality.

A compact latent code enables broader, statistically meaningful evaluations across datasets and attack settings without prohibitive runtime.

Positioning. The PCA/LogReg results do *not* claim that $k = 32$ is the globally optimal dimension for all classifiers. Rather, they establish that (i) the data exhibits strong low-dimensional structure, and (ii) a 32-dimensional representation can retain substantial discriminative signal while remaining computationally tractable. Subsequent chapters validate that nonlinear latent models (e.g., VAEs) can outperform linear PCA baselines at the same dimensionality, particularly under adversarial and noisy conditions.

Appendix B: Reproducibility Package (GitHub Source Code)

7.12 Purpose and Scope

To support reproducibility and transparency, all code used to generate the experimental results in this thesis is maintained in a GitHub repository titled `Project-Codes`. The repository is structured as an end-to-end experimentation workspace for **static malware classification** across three benchmarks (**EMBER**, **BODMAS**, and **MALIMG**) and includes:

`https://github.com/abamidele/Project-Codes`

- Exploratory Data Analysis (EDA) and dataset sanity checks
- Baselines comparing **full feature space vs. PCA-reduced features**
- **Latent space evaluation** (representation learning and downstream classification)
- **Latent space perturbation** (robustness / stress testing in representation space)
- **Transfer learning** between datasets to study generalisation and domain shift

7.13 Repository Organisation (Experiment Families)

The structure of the top-level repository is shown in Listing 7.1.

```
Project-Codes/
          BODMAS - Full_VS_PCA
          EDA
          EMBER - Full_VS_PCA
          Latent Space Evaluation BODMAS
          Latent Space Evaluation EMBER
          Latent Space Perturbation
          Transfer Learning
```

Listing 7.1: Top-level structure of the `Project-Codes` repository.

7.13.1 Folder Roles

`EDA/` Data understanding and sanity checks (e.g., class balance, feature distributions, missingness, correlations, drift/outliers). This is the recommended starting point for verifying the load and pre-processing of the data sets before model training.

`EMBER - Full_VS_PCA/` Baseline models trained on **full EMBER features** versus **PCA-reduced** features to quantify the trade-off between predictive performance/robustness and dimensionality/efficiency.

`BODMAS - Full_VS_PCA/` The same "full vs. PCA" baseline comparison applied to **BODMAS**.

`Latent Space Evaluation EMBER/` Representation-learning experiments for EMBER. Typical activities include learning a compact latent embedding and evaluating downstream classifiers on latent vectors, supported by visualisation and latent-quality analysis.

`Latent Space Evaluation BODMAS/` The same latent-space evaluation workflow applied to **BODMAS**.

`Latent Space Perturbation/` Robustness experiments conducted **in latent space**, including perturbation-based stress tests (e.g., noise injection and structured perturbations) to evaluate representation stability and downstream classifier sensitivity.

`Transfer Learning/` Cross-dataset experiments (e.g. train on one dataset and adapt/evaluate on another) used to measure generalisation, domain shift, and portability of learnt representations. **MALIMG** experiments are included in this cross-dataset/generalisation family where applicable.