Malware Detection Using NLP-based Hybrid Feature Analysis: A Comprehensive Review

Manoj D. Shelar [1], Monali R. Devkar [2], Neha S. Gadhave [3], Sayali S. Jadhav [4,*], Tanuja R. More [5]

[1]Assistant Professor, (Computer Engineering),
Vidya Pratishthan's Kamalnayan Bajaj Institute of Engineering and Technology, Baramati, SPPU Pune, Maharashtra, India.
manojshelar58@gmail.com

[2,3,4,5]Student,(Computer Engineering),
Vidya Pratishthan's Kamalnayan Bajaj Institute of Engineering and Technology, Baramati, SPPU Pune, Maharashtra, India.
monalidevkar05@gmail.com , gadhaveneha20@gmail.com ,
*ssjadhav715@gmail.com , tanujarmore2003@gmail.com

Abstract. The complex modern malware cannot be recognised by a typical detection system signature based detection and single layer analysis detection will not be able to work. This paper suggests a hybrid malware detection model that statically extracts and employs multiple types of features, such as printable strings, API-level structural metadata, and packing tool signatures. Each stream is encoded independently so as not to lose any of the features semantic information which is then combined through an attention-based LSTM. The model effectively addresses key challenges like the issue of feature dominance, sensitivity to sequence alignment, and the fusion of imbalanced features. Use of NLP based multi-feature approach for detection of malware and malware based attack in large scale dataset is shown empirically. This proves to be more robust against obfuscated and zero-day attacks as compared to model based on single feature.

Keywords: Malware Analysis; Cybersecurity; Deep Learning; NLP; Hybrid Model; Attention Mechanism.

1. Introduction

The progression of malware is getting stronger and spreading at a rapid pace. This may hinder cybersecurity. Because conventional approaches, such as searching for known malware signatures, can often miss new or altering malware [10]. Polymorphic and zero-day attacks are two sorts of malware specifically designed to conceal and evade detection [1].

In order to stop them we need more intelligent anti-virus software that will view malware from multiple perspectives. Malware analysis is like a detective solving a case: one clue is not enough, but bringing together different clues helps solve the case.

- **Static features:** such as file size or randomness can give signs of suspicious files [7].
- **Dynamic analysis:** of API calls can show harmful actions [2].
- **Cryptographic hashes:** can quickly detect known malware [10].

Each of these methods is useful, but they work best when combined. Relying on only one type of analysis leaves security gaps.

A hybrid framework that automates this process is suggested by us. It gathers static, dynamic, and hash-based evidence and links them using a cross attention method. This information is further pipelined into a deep learning model composed of Transformer Encoder, BiLSTM and Self-Attention that can learn complex malware signatures that are undetectable by other methods.

2. Literature Survey

Recent malware detection research has increasingly applied machine and deep learning models.

Mimura and Kanno [1] developed a hybrid NLP model combining n-grams, PE metadata, and strings on the FRR dataset (300k samples), achieving a 92.7% F-measure with effective feature fusion, though underperforming deeper neural models.

Sang et al. [4] proposed a GAN and Genetic Algorithm framework that evaded seven ML detectors with over 96% success on VirusShare and Ember datasets, but focused solely on evasion.

M. Kaushik et al. (eds.), *Proceedings of the International Conference on Responsible, Risk-aware, and Regulated AI (RRRAI 2026)*, Advances in Intelligent Systems Research 210,
https://doi.org/10.2991/978-94-6239-723-1_40

Panja et al. [6] proposed a resource-efficient static analysis framework with the use of multiple classifiers achieving 99.39% accuracy, making it quite suitable for IoT devices. However, the downside to this method is the lack of real-world validation.

Durabii [5] employed an image-based method with Snake Optimization, ShuffleNet, and Attention BLSTM on the Malimg dataset, reaching 98.42% accuracy with robustness to polymorphic malware, yet limited generalizability.

Alzahrani et al. [2] utilized dual VGG-19 networks for image-based transfer learning, achieving 99% detection and 98.2% classification accuracy, but on a small dataset of only five families.

3. Machine Learning in Malware Analysis

The research stream on malware detection with machine learning draws on various technical fields [12], notably, Natural Language Processing and Deep Learning contribute different analytical power to make malware detection protocols amongst the most robust in today's analytical world.

3.1. Static Feature Extraction

Static analysis refers to the checks done on the file contents without executing them. The model collects several kinds of features to develop a holistic threat profile.

- All text contents that are readable including Unicode and ASCII strings are extracted from the file. [9]. At times, malware hides commands and URLs in non-English scripting languages to avoid detection. The following methods are used for this process:

  i) **Python libraries:** pefile and lief, along with custom parsers, can be employed to accurately extract Unicode strings encoded in UTF-16 or UTF-32 [7].

  ii) **strings (CLI):** Used for fast extraction of ASCII/UTF-8 text:

  ```
  strings sample.exe > extracted.txt
  ```

- It analyzes sequences of characters or words (N-grams); this helps to find patterns including risky file types (.exe, .dll) and frequently observed attack-related phrases [14]. The following methods perform this process:

  i) **N-grams generation:** Find structural patterns in strings, also identify malicious phrases [1].

  ii) **YARA:** Rule-based utility, operates as a malware scanner, using rules defined by researchers to match specific textual and hexadecimal patterns within files [11].

- Identifies and categorizes extracted strings into types such as URLs, file paths, registry keys, IP addresses, or commands [3]. The following methods perform this process:

  i) **Bayes' Theorem:** Calculate the probability that a string belongs to a specific category, given its features.

  ii) **Mathematical formula:**

$$P(\text{Category} \mid \text{Features}) \propto P(\text{Features} \mid \text{Category}) \cdot P(\text{Category})$$

The static feature extraction pipeline is illustrated in Fig. 1, which shows how raw executable content is processed through Unicode/ASCII extraction, N-gram generation, vectorization, and string categorization stages.

3.2. NLP for Feature Processing

NLP provides methods for the numerical encoding of extracted text [13], enabling computational analysis while preserving semantic and contextual information.

- **Advanced Tokenization:** The text is segmented into subword [9] units, a process that moves beyond simple space-delimited tokenization. This is accomplished using the following methods:

  i) **WordPiece or BPE (Byte-Pair Encoding):** Splits words into sub-components [1].
  Example: "subword" → "sub" + "word"

$$BPE(x, y) \quad \Rightarrow \quad V_{t+1} = V_t \cup \{x + y\}$$

  where V_t represents the vocabulary at iteration t, and $x + y$ denotes the pair of symbols that occurs most frequently in the corpus.

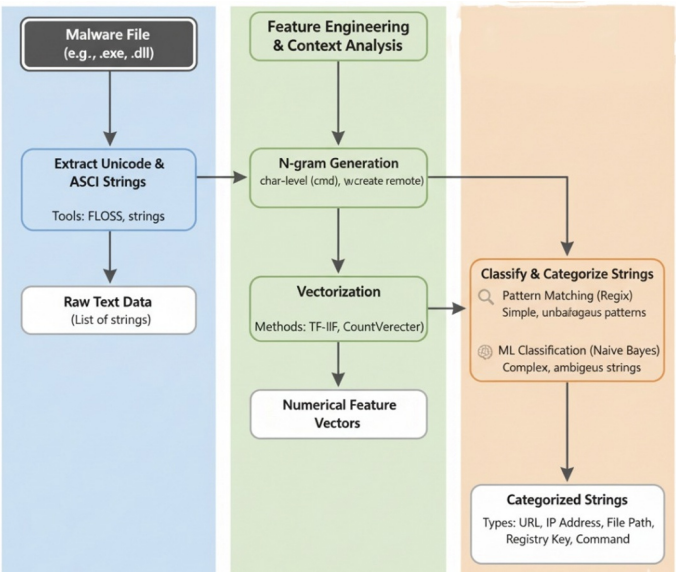

Figure 1: Static Feature Extraction

Advantage: This facilitates the interpretation of unseen word variants through the recognition of their constituent subword units.

- **Contextual Embedding:** Each token is mapped to a high-dimensional vector that reflects its surrounding context, providing finer semantic information than traditional embeddings [7].

$$h_i = f(x_1, x_2, \ldots, x_n)$$

where h_i is the embedding of the i^{th} token, $x_1, x_2, \ldots, x_n$ represent all tokens in the input sequence, and f denotes a transformer-based function (self-attention + feed-forward layers).

The methods/tools:

i) **Pre-trained models:** These models leverage representations learned from large-scale code corpora [9] through pre-training, rather than being trained from random initialization.

ii) **Context-aware vector generation:** It generates context-aware embeddings that capture token semantics [7] based on surrounding linguistic context. For instance, the interpretation of the term "close" varies between phrases such as "close handle" and "close socket".

3.3. Advanced Code Analysis

Advanced code analysis examines program behavior during execution [2], providing runtime insights that complement static analysis of code structure or function imports.

- **Monitoring API Call Sequences:**Runtime behaviour is captured through monitored API calls [4].Program functionality is characterized by operations like file creation, registry modification, and attempts to connect to the network.

Example:

createFile() → modifyRegistry() → sendNetworkRequest()

Advantage: It provides information about how the program works, which is not clear just from a static analysis.

- **Detecting Concealed Functions:**APIs that are loaded dynamically at runtime are conducted to reveal relevant information that cannot be obtained through static analysis. This is particularly useful in spotting malicious routines that do not activate until runtime.

  Advantage: Reveals malicious behavior that static analysis might overlook.

- **Analyzing Program Structure:** ControlControl Flow Graphs (CFGs) represent all possible execution paths within a program, capturing the essence of program flow. By conducting a structural analysis of the graphs, we can find out compound or malicious patterns which would not be visible when observing individual operations in isolation.

  Notation:

$$\text{CFG} = (N, E)$$

  where N represents code blocks (nodes) and E represents execution paths (edges).

  Advantage: Provides a structural understanding of the program, enabling detection of sophisticated malicious behavior.

4. Deep Learning Models for Malware Analysis

- **Bidirectional LSTM (BiLSTM):** A Bidirectional Long Short-Term Memory (BiLSTM) network captures API call sequences [5] to build contextual dependencies for sequences in both directions. Both the above architectures suggest the existence of those programs whose initial behaviour looks benign but eventually becomes malicious [13]. By combining past and future temporal context, the BiLSTM model creates a more comprehensive representation of program behaviour which leads to improved accuracy of the detection.

  i) **Forward pass:** Captures dependencies from the start of the sequence to the end.

  ii) **Backward pass:** Captures dependencies from the end of the sequence to the start.

  iii) **Mathematical representation:**

$$h_t = [\overrightarrow{h_t} \parallel \overleftarrow{h_t}]$$

  The BiLSTM hidden state at time t, h_t, is defined as the sum of the forward hidden state h_t and the backward hidden state h_t.

  Advantage: It provides a better representation by utilizing information from both the past and future tokens.

  The BiLSTM architecture, showing the forward and backward processing layers and their concatenated output, is illustrated in Fig. 2.

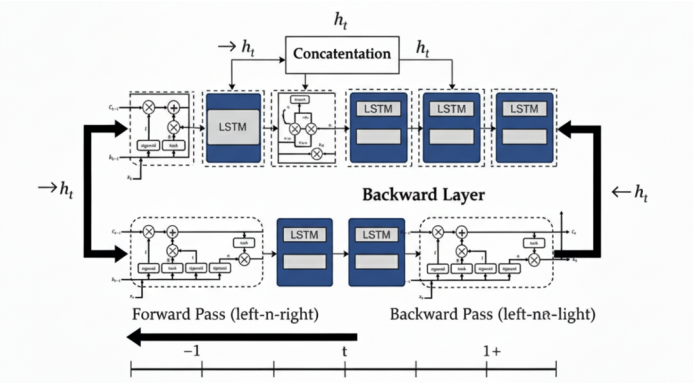

Figure 2: Bidirectional LSTM (BiLSTM)

- **Attention Mechanisms:** Attention allows the model to focus on the most important tokens [9] in a sequence instead of giving equal importance to all.

  i) **Self-attention:** Assigns different weights to tokens within the same sequence [7], highlighting the most important elements.

ii) **Mathematical formula:**

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q, K, and V are the query, key, and value matrices, and d_k is the dimension of the key.

Advantage: Model focuses on the most important features, which improves both understanding and detection accuracy.

5. Proposed Detection and Analysis Framework

An attention augmented LSTM architecture for malware detection is proposed in [7],It is capable of capturing sequential dependencies as well as salient features through the entire malware detection pipeline. The framework is presented on theoretical grounds but needs to be empirically validated [13].

The overall architecture of the proposed framework is presented in Fig. 3.

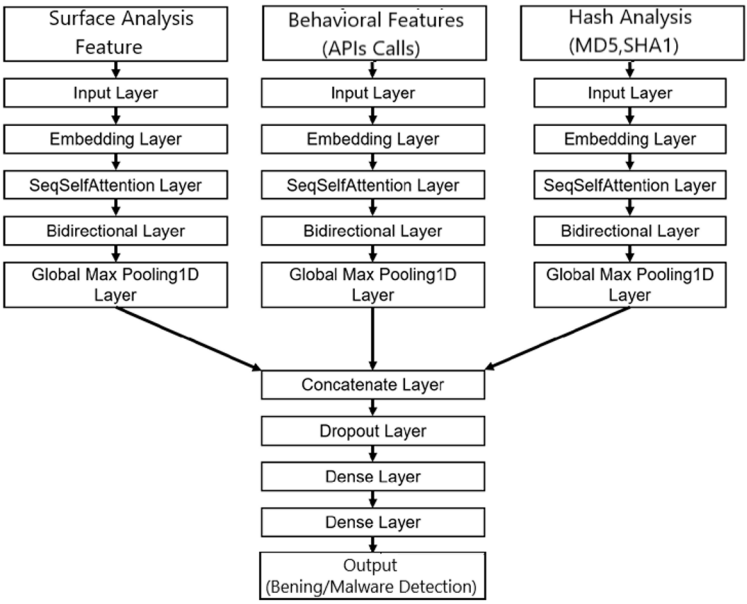

Figure 3: Proposed Model

5.1. Feature Extraction Phase

- **Input:** Malware file (e.g., `.exe` file).
- Extracts features through three branches [7]:
  - **String Analysis** – Finds text strings, N-grams, URLs, file paths [1].
  - **Code & API Analysis** – Tracks API calls, dynamic loading, control flow (CFG) [2].
  - **Obfuscation Analysis** – Detects packing/hiding, entropy changes, suspicious PE sections [5].
- Results are combined (**Feature Aggregation**) and converted into numerical vectors (**Vectorization**) for machine learning [7].

5.2. NLP-based Hybrid Feature Analysis

- Many features are text-based (API names, strings, packer info) [1].
- Uses NLP methods:
  - **Tokenization** – Splits text into smaller pieces [9].
  - **Contextual Embedding** – Uses models like CodeBERT to create meaningful vectors [7].
- Helps system understand the context of features, not just their presence.

5.3. Core Model and Classification

- Main engine: Hybrid LSTM + Self-Attention.
- Attention balances strong and weak features.
- Explainable AI (XAI) shows which features (API calls/strings) influenced the decision.
- Classification process:
  - Malware or benign?
  - If malware → identify type (Trojan, Ransomware, Worm).

5.4. System Impact and Explainability

- Goes beyond detection → also checks impact on system:
  - **Detection Metrics** – Accuracy & confidence score.
  - **Process Impact** – Effect on CPU, memory, threads.
  - **System Resources** – Performance and stability issues.
  - **Security Impact** – Possible integrity violations or unauthorized access.

5.5. Mathematical Formulation and Algorithm

Algorithm: Hybrid Malware Detection
Input: A malware file (for example, `.exe`).

Step 1: Extract Features

- **Text Check:** Look for text inside the file (words, URLs, file paths).
- **Code Check:** See how the program runs and the order of its actions.
- **Hidden Code Check:** Look for hidden or packed parts in the file.

Step 2: Convert to Numbers

Convert text, code, and hidden code data to computer-understandable numbers.

Step 3: Combine Features

Combine all numerical information so that key sections are highlighted.

Step 4: Analyze

- Use a smart model (Transformer + LSTM) to look for patterns.
- Focus attention on the most important details.

Step 5: Decide

- Classify the file as Trojan, Ransomware, Worm, or Benign.
- Pick the type with the highest score.

Output

- Result: "Malicious" or "Benign."
- A report that is accurate, confident, and system impact (CPU, memory, security).

6. Comparative Analysis

As presented in Table 1, a comprehensive comparative analysis of the approaches that were discussed reveals their strengths and limitations along with application scenarios.

Table 1: Comprehensive Comparative Analysis of Malware Detection Approaches

Approach / Method	**Core Models / Algorithms**	**Strengths**	**Limitations**	**Suitable Scenarios**
Hybrid NLP-based Model (Mimura & Kanno, 2024)	n-grams + PE info + string analysis (Hybrid NLP)	92.7% F-measure; robust against obfuscation; effective hybrid feature integration	Lower accuracy than image-based DL; depends on feature extraction	Enterprise malware detection; hybrid feature studies
Adversarial Generation (Sang et al., 2025)	GAN + Genetic Algorithm	>96% evasion success; malware functionality preserved	Focus on evasion, no mitigation strategies proposed	Red-teaming; security model stress testing; vulnerability research
Static ML Classifiers (Panja et al., 2025)	RF, ETC, XGB, SVM (with feature selection)	99.39% accuracy; efficient for IoT/low-resource devices	Limited to static analysis; deployment challenges remain	IoT/embedded systems; endpoint protection
Image-Based DL (Duraibi, 2024)	Snake Optimization + CNN (ShuffleNet + BiLSTM)	98.42% accuracy; strong against polymorphic malware	Tested only on Malimg dataset; generalization limited	Polymorphic malware detection; academic experiments
Image-Based DL (Alzahrani et al., 2022)	VGG-19 + Transfer Learning	99% detection; 98.2% family classification	Focused on 5 malware families; dataset size limited	Family classification; targeted malware detection
Baseline DL Models	RNN, LSTM, DNN	Simple to implement; captures sequential dependencies	Weak against long sequences; limited robustness to obfuscation	Benchmarking; educational research; baseline performance
Proposed Hybrid (Self-Attention + LSTM)	Cross-attention fusion + multi-feature embeddings	Handles zero-day malware; explainable AI (XAI); scalable to families	Computationally intensive; focused on Windows PE files	Enterprise security; critical infrastructure; cloud systems

7. Discussion and Research Gaps

The proposed model achieves promising results through multi-feature fusion [7], improving detection of obfuscated threats. However, several limitations remain:

i) **Works for Windows Only:** Most models mainly check Windows files [10]. We need ones that can also check files on phones (Android) and small devices (IoT) [2].

ii) **Stop Malware Tricks:** Some malware tricks AI to avoid detection [4]. We need to make models smarter so they don't get fooled.

iii) **Runs on Small Devices:** Many smart AI models need big, fast computers [6]. We need models that work fast on small gadgets with low power.

iv) **Test with Real Malware:** Most tests use old data [1]. We need to test models on real, new malware to know how well they work.

v) **Explain How Decisions Are Made:** Some models are hard to understand [7]. If models explain their decisions better, experts can trust and use them more easily.

Conclusion

For detecting several types of malware, this system makes use of some processes, one of which is an advanced AI model called LSTM. It is able to find previously unknown malware that older software wouldn't detect. It provides easy-to-understand results that show which features helped reach the conclusion, whenever it finds threats. The solution is safe for business, cloud services, and critical infrastructure, improving overall security. The tool enhances cybersecurity by helping to catch malware easily and enabling the quick reaction to threats. With further enhancement in the future by working in real time and learning about new malware types, it can further interconnect with threat intelligence systems to stay updated.

References

[1] Mimura, M., & Kanno, S. (2024). Hybrid Input Model Using Multiple Features From Surface Analysis for Malware Detection. *IEEE Access*, 12. DOI:10.1109/ACCESS.2024.3452675.

[2] H. Naeem, X. Cheng, F. Ullah, S. Jabbar, and S. Dong, "A deep convolutional neural network stacked ensemble for malware threat classification in Internet of Things," *J. Circuits, Syst. Comput.*, vol. 31, no. 17, 2022, Art. no. 2250302.

[3] F. Ullah, X. Cheng, L. Mostarda, and S. Jabbar, "Android-IoT malware classification and detection approach using deep URL features analysis," *J. Database Manage.*, vol. 34, no. 2, pp. 1–26, 2023.

[4] A. Sang, Z. Wang, L. Yang, L. Zhou, J. Jia, and H. Yang, "Generating Adversarial Malware Examples Against Multiple Machine Learning Detectors," *IEEE Transactions on Industrial Informatics*, vol. 21, no. 7, pp. 5655–5665, July 2025. doi: 10.1109/TII.2025.3556075.

[5] S. Duraibi, "Enhanced Image-Based Malware Classification Using Snake Optimization Algorithm With Deep Convolutional Neural Network," *IEEE Access*, vol. 12, pp. 95047–95057, Jul. 2024. doi: 10.1109/ACCESS.2024.3425593.

[6] S. Panja, S. Mondal, A. Nag, J. P. Singh, M. J. Saikia, and A. K. Barman, "An Efficient Malware Detection Approach Based on Machine Learning Feature Influence Techniques for Resource-Constrained Devices," *IEEE Access*, vol. 13, pp. 12647–12665, Jan. 2025. doi: 10.1109/ACCESS.2025.3526878.

[7] M. D. Shelar and S. Srinivasa Rao, "A Hybrid DNN-DKN Model With MapReduce Framework for Detecting Malicious Threats in Large Executable Files," *Biomedical Engineering: Applications, Basis and Communications*, World Scientific, ISSN: 1016-2372, May 2025.

[8] M. D. Shelar and S. Srinivasa Rao, "Enhanced Capsule Network based Executable Files Malware Detection and Classification – Deep Learning Approach," *Concurrency and Computation: Practice and Experience*, Wiley, ISSN: 1532-0634, Vol. 36, Issue 4, Feb. 2024.

[9] M. D. Shelar and S. Srinivasa Rao, "Deep malware hunter based unrivaled malware detection schema thru cache retrospective empiricism," *Concurrency and Computation: Practice and Experience*, Wiley, ISSN: 1532-0634, Vol. 34, Issue 19, Aug. 2022.

[10] M. D. Shelar and S. Srinivasa Rao, "Malware detection of executable file," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, ISSN: 2278-3075, Vol. 9, Issue 3, Jan. 2020.

[11] N. D. Birajdar, M. N. Dhuppe, T. M. Hegade, N. S. Jadhav, and M. D. Shelar, "Review Paper On Adware Detection Using Instruction Sequence Generation," *International Journal of Engineering and Techniques*, Vol. 1, Issue 6, Dec. 2015.

[12] Abd, M. S., & Behadili, S. (2025). Malware Detection Using Machine Learning Techniques: A Review. *Basra Journal of Science*, 42.

[13] Song, Y., Zhang, D., Wang, J. *et al.* (2025). Application of deep learning in malware detection: a review. *Journal of Big Data*, 12, 99.

[14] Addanki, S. R., Jahnavi, C., Amrutha, A., & Pachhala, N. (2025). Enhanced Malware Detection in Windows Applications Using Ensemble Learning Techniques. *Journal of Engineering Research and Reports*, 27(4), 94–100.

[15] Abedin, S., & Mehdi, S. A. (2025). Malware Detection Using Machine Learning Techniques. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 13(4). ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538.