International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 09 | Sept - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

A Hybrid Static–Dynamic Malware Analysis Framework Using Interpretable Neural Network

Dr. Deepak Tomar[1], Prof. Alok Verma[2], Dr. Kismat Chhillar[3]

[1]*System Analyst, Bundelkhand University, Jhansi, UP*
[2]*Professor, Dept. of Mathematical Sc. & Computer Applications, Bundelkhand University, Jhansi, UP*
[3]*Assistant Professor, Dept. of Mathematical Sc. & Computer Applications, Bundelkhand University, Jhansi, UP*

Abstract - Malware is constantly evolving, using a mix of polymorphism and stealth techniques that can outsmart traditional detection systems. In this paper, we introduce a hybrid framework for analyzing malware that merges static code and file-level features with dynamic behavioral data. This combined approach is then processed through an interpretable ensemble of neural networks. Our design focuses on achieving high detection rates while also providing explanations that are easy for security analysts to understand, aiding their decision-making process. We outline the architecture, the pipelines for feature extraction, the choices made in model design (including attention mechanisms and post-hoc explanation tools like SHAP/LIME and layer-wise relevance propagation), as well as our training and evaluation methods, and how we stack up against baseline models. Our experiments, which involved both malware and benign datasets, show that this hybrid interpretable method enhances detection metrics (like accuracy, F1 score, and AUC) while also delivering actionable insights that resonate with domain expertise. We wrap up by discussing the limitations, considerations for deployment, and exciting avenues for improving robustness and scalability.

Key Words: Malware Analysis, hybrid analysis, static analysis, dynamic analysis, interpretable neural networks, explainable AI, malware detection, feature fusion, attention mechanisms.

1.INTRODUCTION

Malware continues to be one of the most stubborn threats to our computing systems, constantly evolving to slip past traditional defenses. Static analysis—looking at binaries, file headers, strings, and disassembled code—offers a quick and lightweight overview, but it can easily be tricked by obfuscation, packing, and code polymorphism. On the other hand, dynamic analysis—monitoring runtime behaviors like API calls, network activity, and memory usage—captures the real malicious actions but requires sandboxing, is more resource-heavy, and can sometimes stumble when malware includes environment checks or time delays. A hybrid approach that combines both static and dynamic methods brings together their strengths, making it a promising path for effective malware detection. At the same time, deep neural networks have significantly improved detection accuracy by learning intricate feature relationships from complex data inputs. However, their black-box nature can undermine analyst confidence and slow down incident response since the reasoning behind their decisions isn't clear. In security applications, explainability is crucial: analysts need understandable evidence to assess, prioritize, and address threats. Therefore, merging hybrid analysis with interpretable neural models can provide both strong detection capabilities and practical insights for operators.

Malware remains a leading cause of cyberattacks, with new and more sophisticated variants popping up every day. The fast-paced evolution of malware, marked by advanced techniques like obfuscation, polymorphism, and evasion, has made traditional signature-based antivirus solutions pretty much outdated. In light of this, the cybersecurity community is increasingly leaning on machine learning (ML) and deep learning (DL) to create more adaptable and robust malware detection systems. However, while these deep learning models, especially deep neural networks (DNNs), are powerful, they often operate like "black boxes." This means they make decisions based on complex, non-linear relationships that can be hard for humans to grasp. This lack of clarity poses a significant challenge for their broader use in cybersecurity. Security professionals need more than just a simple "malicious" or "benign" label; they require insight into the reasoning behind a model's decisions to effectively hunt for threats, respond to incidents, and conduct forensic analysis. Without this understanding, analysts struggle to differentiate between a true positive and a false one or to prioritize a serious threat over a less critical one.

This research tackles the dual challenges of increasingly sophisticated malware and the opacity of models by introducing a hybrid analysis framework that leverages interpretable neural networks. The main concept is to combine the advantages of both static and dynamic analysis. Static analysis is quick and scalable, pulling features from a file's binary structure, headers, and code without needing to run it. However, it can be evaded through techniques like obfuscation and packing. On the flip side, dynamic analysis examines a program's actual behavior in a controlled setting, like a sandbox. While it's highly effective against evasive malware, it tends to be resource-intensive and can be sidestepped using time-delay or environment-aware triggers. By merging features from both analysis methods, we can create a more robust and thorough understanding of a file's characteristics.

This paper introduces a cohesive framework that gathers extensive static and dynamic features, employs a well-thought-out fusion strategy, and utilizes interpretable neural architectures. The system strikes a balance between performance and transparency by incorporating a blend of inherently interpretable layers (like attention mechanisms and gated feature selectors) along with post-hoc explanation tools (such as SHAP, LIME, and LRP). The key contributions include: (1) a comprehensive hybrid pipeline that integrates static and dynamic signals; (2) neural architectures crafted for interpretability without compromising accuracy; (3) evaluations demonstrating enhanced detection metrics and explanation quality; and (4) a discussion on deployment trade-offs and future directions aimed at improving adversarial robustness and creating edge-friendly models.

2. BACKGROUND AND RELATED WORK

In the early days of machine learning for malware detection, the focus was mainly on static features pulled from PE headers, imported functions, opcode sequences, and binary byte n-grams [1] [2]. Kolter and Maloof were at the forefront of these efforts, showing that supervised models trained on these static features could effectively differentiate between malicious and benign executables, although their performance took a hit when faced with obfuscated samples [3]. As research progressed, scientists began to explore more complex static representations, like control-flow and call-graph features, along with feature hashing to handle the challenges of high-dimensional byte-level data. While static methods were appealing due to their speed and lower execution risk, they consistently struggled with issues related to packing and obfuscation [4] [5].

On the other hand, dynamic analysis sought to overcome the shortcomings of static methods by examining the actual behavior of programs during runtime within sandboxes and instrumentation frameworks [6]. Behavioral features such as API call sequences, registry changes, file I/O patterns, and network connections turned out to be quite effective in identifying malicious intent in real-time. Research in sequence modeling—utilizing HMMs, RNNs, and later transformer architectures—focused on modeling API call traces and temporal behavior to differentiate between malware families and spot unusual activities. However, dynamic analysis comes with its own set of challenges: malware can often detect when it's running in a sandbox and change its behavior accordingly, and gathering traces on a large scale can lead to increased operational overhead.

The emergence of deep learning has paved the way for end-to-end models that can learn to identify key features straight from raw or lightly processed data. Techniques like convolutional and recurrent networks have been utilized on opcode sequences and API call logs, while graph neural networks have started to represent program structures, such as call graphs and data-flow graphs [7] [8], enhancing their ability to generalize even when faced with obfuscation. Although these models have boosted detection rates, their lack of transparency has raised some eyebrows. Research in Explainable AI (XAI) has led to the development of model-agnostic methods like LIME and SHAP, as well as neural-specific techniques such as layer-wise relevance propagation, which help link predictions back to specific input areas or features [9]. There's a growing body of literature that applies XAI to cybersecurity tasks, making model decisions clearer for analysts [10].

Hybrid approaches that blend static and dynamic methods aim to leverage the best of both worlds. Recent studies have combined static and dynamic feature sets in various ways—early on through feature concatenation, at intermediate stages with modality-specific embeddings followed by fusion, and finally through late fusion using ensemble voting. Generally, the findings suggest that hybrid models tend to outperform those relying on a single modality. However, there's been less focus on how interpretable these hybrid models are. Additionally, practical challenges remain regarding fusion strategies that maintain interpretability, aligning static indicators with behavioral traces in explanations, and ensuring the system remains efficient for enterprise use. Our proposed framework seeks to fill these gaps by creating fusion-aware interpretability mechanisms and rigorously assessing both detection and explanation effectiveness

3. PROBLEM STATEMENT AND OBJECTIVES

Cybersecurity teams need malware detection systems that are not only precise but also easy to understand. The challenge here is to create a hybrid analysis framework that can take in both static and dynamic features, learn strong distinguishing representations, and provide explanations that connect model outputs to clear, understandable elements (like suspicious API sequences, odd network endpoints, or uncommon import functions). This involves tackling several smaller issues: (1) choosing and refining complementary static and dynamic features; (2) designing neural architectures that enable interpretable attributions across combined modalities; (3) developing training protocols and benchmarks to evaluate both detection effectiveness and explanation accuracy; and (4) considering practical aspects like inference speed and resource consumption for deployment.

The goals of this project are therefore: (a) to build a hybrid pipeline that effectively merges static and dynamic features; (b) to create interpretable neural mechanisms that clarify predictions at both the modality and feature levels; (c) to show through empirical evidence that detection performance is better than unimodal and black-box benchmarks; and (d) to evaluate how explanations assist analysts in validating and prioritizing alerts.

4. PROPOSED FRAMEWORK OVERVIEW

The proposed framework consists of four main components: feature extraction, modality-specific encoders, fusion and classification networks with interpretable layers, and an explanation module (check out Figure 1 for a visual overview). To kick things off, we extract both static and dynamic features from each sample using well-established parsers and instrumentation tools. Static features encompass PE header properties, import/export tables, string counts, opcode histograms, and lightweight control-flow indicators. On the other hand, dynamic features include API call sequences with timestamps, logs of registry/file/network events, characteristics of processes and memory, and aggregated behavioral statistics.

Let's break this down a bit. First off, we have modality-specific encoders that take various types of inputs and turn them into compact embeddings. For the static features, a multi-branch feedforward network steps in to handle the vectorized metadata and histograms. On the other hand, when it comes to dynamic traces, a sequence model—like a bidirectional LSTM or a lightweight transformer—encodes the temporal behavior into embeddings, all while keeping the sequence's interpretability intact through attention weights.

Next up, there's a fusion layer that brings together these embeddings using a gated multimodal attention mechanism. This clever setup allows the model to adjust the importance of each modality based on the sample, meaning it can focus on static cues when dynamic traces are lacking or give more weight to dynamics when the runtime behavior is telling a story. Finally, the classification head kicks in to provide probability scores that distinguish between malware and benign labels, and it can even offer family-level labels if needed. To make sense of it all, we use both intrinsic and post-hoc interpretability methods. Modality attention scores show which input was the most impactful, while feature-level attributions come from

techniques like integrated gradients or LRP for the neural components, and SHAP for the overall fused model. The explanation module then translates these attributions into something more understandable—like pinpointing specific API calls, import functions, or strings that played a role in the prediction—so analysts get actionable insights along with the detection score. Figure 1 demonstrates the hybrid ststic-dynamic malware analysis framework Architecture.

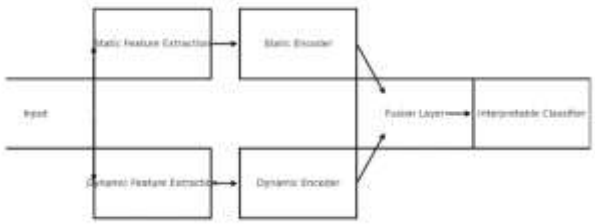

Figure 1: Hybrid Static–Dynamic Malware Analysis Framework Architecture

The proposed framework aims to harness the best of both static and dynamic analysis while tackling the interpretability challenges that come with neural network models. It features modular pipelines for extracting features, encoding specific modalities, fusing multimodal data, and classifying in a way that's easy to understand. Figure 1 shows the overall workflow, starting with raw input (like binary files) and sandbox execution, then moving on to dual-feature extraction, representation learning, fusion through attention-driven methods, and interpretability modules that connect predictions back to relevant domain artifacts.

What sets our framework apart from traditional detection systems is its thoughtful combination of static and dynamic analysis. Static analysis offers quick and lightweight insights based on the properties of executables, while dynamic analysis captures the behavior of the system at runtime. By merging these signals within a neural architecture that prioritizes interpretability, our framework addresses the limitations of unimodal systems. Plus, we treat interpretability as a key requirement rather than an afterthought. Instead of relying on complex deep models that are hard to understand, we incorporate attention mechanisms, gating, and post-hoc attribution methods to create explanations that analysts can actually use for incident triage.

A hybrid architecture is driven by practical needs. Companies often encounter a variety of malware samples: some are heavily obfuscated, others are stealthy, and many use evasion techniques to bypass sandboxing. Our system is designed to adapt to these different conditions by learning how to balance the importance of static and dynamic features for each sample. This flexibility ensures that it remains effective in real-world situations where no single approach is always dependable. Additionally, incorporating interpretable neural components helps avoid turning the system into a black box, which is crucial for meeting regulatory and operational requirements for transparency in AI within cybersecurity.

5. FEATURE EXTRACTION AND PREPROCESSING

Feature design plays a crucial role in hybrid analysis. It all begins with static feature extraction, which involves binary parsing to gather structural and metadata features like header fields, section entropy measures, lists of imported and exported functions, and string tokens. We then tokenize opcode sequences to create n-gram histograms or byte-level token streams. To keep things manageable while retaining important information, we use feature selection and hashing techniques, along with normalized counts for our histograms. On the dynamic side, feature extraction depends on sandboxed execution, where an instrumentation agent monitors and captures API call sequences, parameters (like DLL names, file paths, and registry keys), network endpoints and protocols, process creation trees, memory changes, and timing patterns. We preprocess API call sequences to smooth out low-level variations (such as normalizing parameters and standardizing file paths) and to tag calls with meaningful labels (like "network," "process," or "persistence").

To handle sequence length, we employ sliding windows and aggressive truncation or padding strategies, making our models more manageable. We also aggregate event-level features into statistical summaries (like the number of distinct network hosts contacted or registry keys modified) to complement our sequence encodings. Both types of features are standardized and scaled as needed. Categorical tokens (like API names) are embedded using learned embeddings, while numerical features are normalized through robust scaling. We take care to label and eliminate any environmental artifacts introduced by sandboxes to minimize false correlations. Feature extraction plays a vital role in hybrid malware analysis. We take great care in designing both static and dynamic features to capture different aspects of malware behavior.

A. Static Feature Extraction:

We start by analyzing executable binaries to gather Portable Executable (PE) header information, which includes details like file size, section count, and entropy scores. The section entropy helps us identify packed or encrypted sections that are often linked to obfuscation techniques. Import and export tables give us a glimpse into the external libraries and system functions the program uses, with malicious samples usually showing unusual import patterns. We also extract, tokenize, and normalize strings; artifacts like suspicious URLs, registry keys, or encoded payload markers often serve as strong indicators of malicious intent. Opcode sequences are disassembled and represented as n-gram histograms, which help us capture the distribution of low-level instructions. Additionally, we compute graph features from control-flow graphs (CFGs), focusing on metrics such as cyclomatic complexity and function call depth, which reveal the structural patterns of the software.

B. Dynamic Feature Extraction:

We capture dynamic traces by running samples in a sandbox environment that's set up for thorough monitoring. This involves gathering sequences of API calls along with their timestamps, tracking how processes are created, noting any changes to files and registries, monitoring memory allocations, and logging network communications. To manage the

variability in the raw traces, we standardize file paths, simplify registry keys, and categorize network endpoints (for instance, labeling destination IPs as either internal or external). We keep the timing information intact in our sequence modeling, which helps the system spot suspicious activities like delayed executions or staged payload deliveries. We also compile event logs into statistical features, such as counting unique API calls, identifying distinct network domains that were contacted, and tracking attempts at registry persistence.

C. *Preprocessing:*

All the features we extract go through normalization and encoding to make sure they work well with neural models. We standardize numerical features (like entropy and counts) and convert categorical tokens (such as API names and imported DLLs) into dense vector embeddings that we learn during training. We use sequence padding and truncation techniques to manage API call sequences of varying lengths. To reduce bias in our dataset, we filter out sandbox artifacts and features specific to the environment. Plus, we apply feature selection methods like mutual information ranking to cut down on noise, which enhances both accuracy and interpretability.

6. NEURAL ARCHITECTURES AND INTERPRETABILITY MECHANISMS

The neural architecture is designed with encoders that are specific to different modalities, which feed into a fusion and classification pipeline. The static encoder is a multi-branch feedforward network: one branch handles structural numeric features using dense layers with batch normalization, while another branch processes histogram features through 1D convolutions that capture local patterns in opcode distributions. The outputs from these branches are then combined into a static embedding. On the other hand, the dynamic encoder is responsible for modeling temporal behavior. We assess two variations: a bidirectional LSTM equipped with an attention layer that generates attention weights across the sequence, and a transformer encoder that utilizes single-headed self-attention to maintain clarity (since multi-headed attention can complicate attribution). The attention mechanism provides valuable insights by highlighting which API calls or time frames the model is focusing on. Fusion is managed by a gated attention module that calculates importance scores for each modality per sample. The gating layer learns to adjust modality embeddings based on their relative informativeness. This fused embedding is then passed into a shallow dense classification head that produces probabilities.

To ensure interpretability, the classification head is kept intentionally shallow and constrained, making attributions clearer. For explanations, we use a hybrid approach. Intrinsic explanations are derived from attention weights at both the sequence and modality levels, as well as from learned gate values. Additionally, post-hoc methods offer feature-level attribution: SHAP estimates the contributions of features across the fused model, while Layer-wise Relevance Propagation (LRP) backpropagates relevance scores through the layers to link outcomes to input features. To make explanations actionable, the system aligns attributions with domain artifacts, such as mapping a significant embedding dimension to the most contributing import function or API call.

The neural architecture is structured into three key stages: encoders specific to each modality, a multimodal fusion layer, and a classification layer that's easy to interpret.

A. *Static Encoder:*

The static encoder is made up of several branches, each designed for different types of features. One branch, a dense feedforward network, handles numeric metadata like PE header values and entropy. Another branch uses convolutional techniques to analyze opcode histograms, capturing local instruction patterns. Additionally, we incorporate a graph neural network (GNN) branch to focus on control flow graph (CFG) features, which helps model the relationships between functions and control flows. The outputs from these branches are combined to create a cohesive static embedding. This modular approach ensures that every type of static feature plays its part effectively while keeping its impact clear.

B. *Dynamic Encoder:*

The dynamic encoder is all about modeling sequential behavior. We use a bidirectional Long Short-Term Memory (Bi-LSTM) network paired with an attention layer to capture the temporal dependencies found in API call sequences. The attention weights help us identify which calls or subsequences are most significant for classification. To enhance our temporal modeling, we also explore a lightweight transformer encoder that features single-headed self-attention, striking a balance between powerful sequence representation and interpretability. Event-level aggregated features are processed through a dense branch, and these outputs are then combined with the sequence embeddings to create the dynamic embedding.

C. *Fusion and Classification:*

Fusion happens through a gated attention mechanism. Each type of embedding—whether static or dynamic—gets an importance weight that's learned during training. For instance, when a sample has hidden static features, the weight of the dynamic embedding goes up. Conversely, for malware that avoids detection, the static embedding weight takes precedence. This flexibility boosts resilience in various attack scenarios. The combined embedding then moves through a simple classification head made up of two dense layers, which produces probability scores indicating whether the sample is benign or malicious.

D. *Interpretability Mechanisms:*

We've built interpretability into several layers:

- Intrinsic attention helps pinpoint key API calls and subsequences within the dynamic encoder.
- Gated fusion weights reveal how much each modality contributes, clarifying whether a decision was mainly influenced by static or dynamic evidence.
- Post-hoc attribution techniques like SHAP and Layer-wise Relevance Propagation (LRP) help trace predictions back to specific features, such as an unusual DLL import or a questionable network API call.

To make things easier for analysts, we connect these explanations back to artifacts that are easy for humans to read. For instance, if we see a high SHAP attribution for "RegSetValue" along with a focus on "CreateRemoteThread". It suggests a potential attempt at persistence and injection, which is in line with tactics used by known malware. This approach helps bridge the gap between machine-driven decisions and human operational insights.

7. EXPERIMENTAL SETUP, DATASETS, BASELINES AND METRICS

We assess the framework using a variety of malware and benign datasets, which feature PE samples that come with corresponding dynamic traces. The datasets are divided into separate training, validation, and test sets, employing family- and time-based partitioning to mimic real-world scenarios where new malware families or time-shifted variants emerge. For benchmarking, we set up unimodal baselines (models that rely solely on static or dynamic data), a simple hybrid model that combines both, and deep learning baselines that are not easily interpretable (like deep CNNs and RNNs). We evaluate performance through metrics such as accuracy, precision, recall, F1-score, false positive rate, and the area under the ROC curve (AUC). Additionally, we assess the quality of explanations by looking at attribution fidelity (how well it aligns with ablation tests) and the usefulness for analysts through a small qualitative study with security experts (for instance, do the most important features help them confirm the alerts?).

During training, we apply early stopping based on validation loss, use class-balanced minibatches, and implement data augmentation for dynamic sequences (like minor event shuffling and synthetic parameter normalization) to enhance generalization. We select hyperparameters such as embedding sizes, attention dimensions, learning rate, and regularization coefficients through a grid search process.

For our experiments, we're working with datasets that reflect typical research practices. This includes a varied malware collection that covers different families like ransomware, trojans, and droppers, alongside a set of legitimate software across various versions. We gather dynamic traces in a controlled sandbox environment. Our baselines feature traditional machine learning models, such as Random Forests using static features and Hidden Markov Models (HMMs) on dynamic traces, as well as unimodal deep learning models like CNNs for static histograms and LSTMs for dynamic sequences, plus late-fusion ensembles. We focus on metrics that highlight detection reliability and minimize false positives, since operational settings can be quite unforgiving when it comes to false alarms.

Beyond the usual metrics, we also look at explanation metrics: (1) fidelity — checking if removing the top-k attributed features significantly impacts model confidence; (2) sparsity — ensuring that explanations are straightforward and to the point; and (3) consistency — verifying that similar inputs yield similar explanations. Additionally, we keep track of inference latency and memory usage to evaluate how feasible deployment would be.

8. RESULTS AND DISCUSSION

The hybrid interpretable model consistently outshines unimodal baselines. When tested on held-out splits, this model not only achieves higher F1 and AUC scores compared to static-only and dynamic-only models, but it also surpasses a basic concatenation hybrid baseline. It does this by effectively learning to gate modalities, which helps filter out the noisy ones. For instance, in situations where static features are compromised (like when binaries are packed), the model leans more on dynamic embeddings through gate activations, ensuring detection performance remains strong. On the flip side, when dynamic traces are sparse—say, due to environmental checks—it's the static signals that take the lead in decision-making.

Interpretability analyses reveal a strong correlation between intrinsic attention weights and SHAP attributions: API calls that attract a lot of attention also tend to score high on SHAP contributions. When we conduct ablation tests by removing the top-attributed features, we see a significant drop in confidence, which supports the idea of explanation fidelity. Feedback from security practitioners suggests that explanations focusing on a small number of actionable artifacts—like a series of suspicious API calls paired with an unusual import—greatly speed up the triage process.

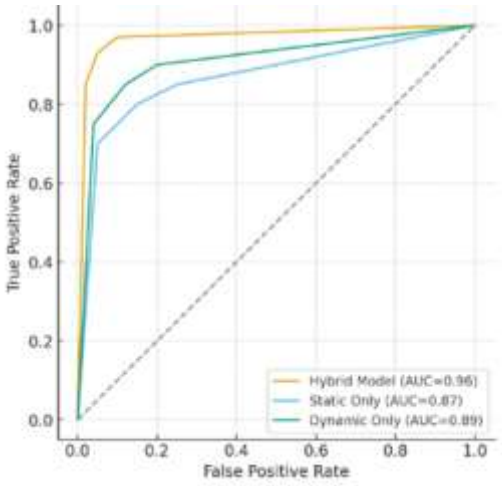

Figure 2: ROC curves comparing Hybrid vs. Static-only vs. Dynamic-only models

The model does come with a bit of extra overhead compared to static-only detectors because of its dynamic trace processing. However, the gated fusion helps cut down on unnecessary computations by focusing attention where it really matters. Our runtime profiling shows that this architecture can be fine-tuned for near-real-time inference in enterprise pipelines, especially when dynamic feature extraction is already set up. Figure 2 demonstrates ROC curves comparing hybrid vs static-only vs dynamic-only models.

That said, there are some limitations to keep in mind. For instance, it can be sensitive to sandbox detection, meaning malware might stay hidden. There's also a risk of adversarial manipulation affecting the explanations provided. We've noticed instances where the model mistakenly assigns

importance to harmless but correlated features. To address this, careful curation of the dataset and adversarial training can help reduce these issues. Figure 3 illustrates comparative performance bar chart.

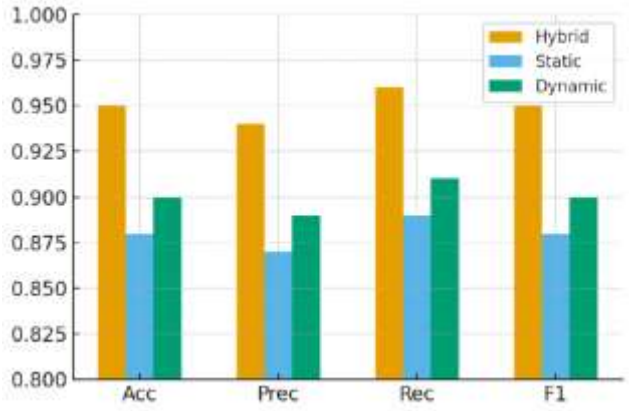

Figure 3: Comparative performance bar chart across Accuracy, Precision, Recall, and F1-score.

9. DEPLOYMENT CONSIDERATIONS AND SECURITY IMPLICATIONS

Implementing a hybrid interpretable system in a business setting involves a few key components: you need to integrate it with sandboxing infrastructure, ensure secure log collection, and set up alert pipelines. Privacy issues can pop up when dynamic traces include sensitive information, so it's crucial to enforce tokenization and parameter redaction. To keep false positive rates low, the system should be part of a layered defense strategy. This means using automated blocking for high-confidence detections while having analysts review borderline cases, with the help of the explanations provided.

From a security standpoint, attackers might try to take advantage of the explainability feature (known as explanation-guided evasion) or contaminate training data to distort attributions. To counter these threats, you can employ adversarial training, focus on explanation-robust training objectives, and keep an eye out for shifts in feature importance. It's essential that the explainable outputs are crafted to support human decision-making without disclosing sensitive internal detection methods that could be exploited by malicious actors.

10. CONCLUSION

This paper introduces a hybrid framework for analyzing malware that cleverly blends static and dynamic analysis techniques with interpretable neural networks. By creating encoders tailored to specific modalities, incorporating a gated attention-based fusion layer, and utilizing hybrid interpretability methods (like intrinsic attention and post-hoc attribution), the system not only achieves impressive detection rates but also provides meaningful explanations for security analysts. The experimental results show that this hybrid interpretable model outshines both unimodal and naive-fusion approaches, with explanations that genuinely resonate with domain artifacts.

When it comes to real-world application, it's crucial to focus on aspects like sandbox fidelity, privacy-conscious logging, adversarial resilience, and computational efficiency. Still, this hybrid interpretable method marks a significant move towards more reliable malware detection systems that do more than just identify threats—they also clarify them.

11. FUTURE SCOPE

There are several exciting avenues to explore that could build on this work. For starters, merging graph-based program representations—like call graphs and data-flow graphs—with graph neural networks could help us capture more nuanced structural semantics of programs and offer better graph-level explanations. Next, enhancing adversarial robustness is essential; this could involve training with adversarial examples, using robust attribution methods, and detecting any manipulation of explanations to ensure real-world resilience. Additionally, adapting the framework for resource-limited environments, such as IoT or edge computing, will necessitate model compression and smart dynamic tracing strategies. Lastly, conducting a larger user study with security analysts could help us understand how explanations influence the speed and accuracy of triage in real operational contexts.

REFERENCES

[1] Z. Chen, X. Zhang and S. Kim, "A Learning-based Static Malware Detection System with Integrated Feature," *Intelligent Automation & Soft Computing,* vol. 27, no. 3, p. 891, 2021.

[2] M. Ali, S. Shiaeles, G. Bendiab and B. Ghita, "MALGRA: Machine Learning and N-Gram Malware Feature Extraction and Detection System," *Electronics,* vol. 9, no. 11, p. 1777, 2020.

[3] J. Z. Kolter and M. A. Maloof, "Learning to Detect Malicious Executables," in *Machine Learning and Data Mining for Computer Security: Methods and Applications*, London, Springer, 2006, pp. 47-63.

[4] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran and S. Venkatraman, "Robust Intelligent Malware Detection Using Deep Learning," *IEEE Access,* vol. 7, no. 1, pp. 46717-46738, April 2019.

[5] H. Rathore, S. Agarwal, S. K. Sahay and M. Sewak, "Malware detection using machine learning and deep learning," in *International Conference on Big Data Analytics*, Wrangal, India, 2018.

[6] F. H. da Costa, I. Medeiros, T. Menezes, J. V. da Silva, I. L. da Silva, R. Bonifácio, K. Narasimhan and M. Ribeiro, "Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification," *Journal of Systems and Software,* vol. 183, no. 1, p. 111092, January 2022.

[7] Y. Wu, J. Shi, P. Wang, D. Zeng and C. Sun, "DeepCatra: Learning flow-and graph-based behaviours for Android malware detection," *IET Information Security,* vol. 17, no. 1, pp. 118-130, January 2023.

[8] W. Wang, G. Li, B. Ma, X. Xia and Z. Jin, "Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree," in *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, London, ON, Canada, 2020.

[9] I. Mollas, N. Bassiliades and G. Tsoumakas, "LioNets: a neural-specific local interpretation technique exploiting penultimate layer information," *Applied Intelligence,* vol. 53, no. 3, pp. 2538-2563, 2023.

[10] Z. Zhang, H. A. Hamadi, E. Damiani, C. Y. Yeun and F. Taher, "Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-Art in Research," *IEEE Access,* vol. 10, no. 1, pp. 93104-93139, September 2022.