applied sciences

MDPI

Article

Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning with Prototype-Guided Zero-Shot Inference for Zero-Day Malware Detection

Ahmed Essaa Abed Alowaidi * and Galip Cansever

Department of Electrical and Computer Engineering, Altinbas University, Mahmutbey, Dilmenler Cd. No: 26, Bağcılar, Istanbul 34217, Turkey; galip.cansever@altinbas.edu.tr
* Correspondence: 203720271@ogr.altinbas.edu.tr

Abstract

Detecting previously unseen malware remains a critical challenge for modern cybersecurity systems due to the rapid evolution of malicious software and the limitations of traditional supervised detection models. This paper proposes a Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework for zero-day malware detection that jointly models static malware artifacts and dynamic execution traces within a unified representation space. The proposed architecture employs two parallel encoders that extract semantic features from executable structures and behavioral features from API call sequences. These features are integrated through a cross-channel fusion mechanism and processed using a Mamba-based selective state space architecture, which efficiently captures long-range dependencies in malware behavior while maintaining linear computational complexity. To address the zero-day detection problem, a prototype-guided inference strategy is introduced that enables similarity-based classification of previously unseen malware families within the learned embedding space. Extensive experiments conducted on multiple malware datasets demonstrate that the proposed framework consistently outperforms strong deep learning baselines. The model achieves an average classification accuracy of 96.01% $\pm$ 0.35 and an F1-score of 95.56% $\pm$ 0.36, while the zero-day detection rate reaches 88.93% $\pm$ 0.98, significantly improving detection performance compared with transformer and recurrent architectures. Runtime analysis further shows that the proposed framework achieves efficient inference with an average latency of approximately 8 ms per sample, making it suitable for real-time malware analysis systems. These results indicate that combining dual-channel feature learning with Mamba-based sequential modeling provides an effective and scalable solution for detecting both known and previously unseen malware threats.

Keywords: zero-day malware detection; mamba architecture; semantic–behavioral feature learning; prototype-based zero-shot inference; malware analysis

Check for updates

Academic Editor:
Douglas O'Shaughnessy

Received: 24 April 2026
Revised: 8 May 2026
Accepted: 11 May 2026
Published: 26 May 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

The rapid growth of malware variants and the increasing sophistication of cyber threats have made malware detection a critical challenge in modern cybersecurity systems [1]. Traditional signature-based detection methods are increasingly ineffective against evolving threats, particularly zero-day malware that has not been previously observed or labeled [2]. To address this issue, machine learning and deep learning approaches have been widely adopted to automatically learn discriminative patterns from malware data. Recent studies

have explored semantic feature extraction from binary code, opcode sequences, and system behavior traces in order to capture meaningful characteristics that distinguish malicious software from benign programs.

Ahmad et al. [3] conducted a comprehensive systematic literature review on zero-day attack detection methods and highlighted the growing shift from rule-based security mechanisms toward machine learning and deep learning techniques. Their analysis emphasized that modern approaches increasingly rely on behavioral modeling and anomaly detection strategies to identify previously unseen attacks.

Recent studies have explored advanced deep learning architectures for zero-day threat detection. Mirza et al. [4] introduced ZDBERTa, a zero-shot learning framework designed for detecting cyberattacks in Internet of Vehicle environments. Their approach leverages contextual representations to identify previously unseen attack patterns, demonstrating promising performance for dynamic network environments. Similarly, Cen et al. [5] proposed Zero-Ran Sniff, a ransomware early detection framework based on zero-shot learning principles. Their method focuses on identifying behavioral deviations during the early stages of ransomware execution, enabling earlier detection compared with traditional signature-based systems.

Large language models have also been investigated for cybersecurity applications. Alsuwaiket et al. [6] proposed ZeroDay-LLM, which utilizes large language models to analyze threat intelligence and security data for detecting zero-day cyber threats. Their framework demonstrates the potential of language models in understanding complex attack patterns and contextual threat information. In contrast, Venčkauskas et al. [7] explored a lightweight approach based on static portable executable header features combined with classical machine learning techniques for ransomware detection, achieving competitive detection performance with relatively low computational overhead.

Deep learning architectures that combine multiple neural network models have also been investigated. Babaey et al. [8] proposed an ensemble detection framework integrating LSTM, GRU, and stacked autoencoders for identifying zero-day web attacks. Their results indicate that combining multiple deep learning models improves detection performance by capturing diverse behavioral patterns in network traffic data. Similarly, Agbedanu et al. [9] proposed an adaptive self-adjusting memory k-nearest neighbor model designed for scalable intrusion detection in IoT and industrial IoT environments, demonstrating improved adaptability for evolving attack scenarios.

Other research has focused on anomaly detection frameworks for identifying unknown threats. Katbi and Ksantini [10] introduced an improved deep support vector data description autoencoder with adversarial regularization for anomaly detection in IoT systems. Their approach enhances detection capability by learning compact representations of normal system behavior. Jagat et al. [11] proposed a variational LSTM autoencoder deviation network for detecting web attacks using HTTP weblog data, enabling the identification of abnormal traffic patterns indicative of cyberattacks.

Recent work has also investigated the integration of advanced AI models with cybersecurity systems. Abshari et al. [12] proposed an LLM-assisted framework for extracting physical invariants in cyber-physical systems, enabling improved anomaly detection through contextual understanding of system dynamics. Additionally, Ohtani et al. [13] developed a federated learning-based intrusion detection system that autonomously extracts anomalies across distributed IoT environments, demonstrating improved privacy-preserving security monitoring. These approaches have significantly improved detection accuracy, yet the rapid evolution and obfuscation of malware families continue to challenge existing detection systems, as shown in Table 1.

Table 1. Summary of recent zero-day attack detection methods and their key characteristics.

Work	Method	Application Domain	Key Idea	Limitation
Ahmad et al. [3]	Systematic Literature Review	Cybersecurity	Comprehensive review of machine learning and deep learning methods for zero-day attack detection	Does not propose a detection model
Mirza et al. [4]	ZDBERTa (BERT-based Zero-Shot Learning)	Internet of Vehicles (IoV)	Uses contextual transformer representations to detect unseen cyberattacks using zero-shot learning	High computational complexity and limited scalability
Cen et al. [5]	Zero-Ran Sniff	Ransomware Detection	Early ransomware detection using zero-shot behavioral analysis	Focuses primarily on ransomware scenarios
Alsuwaiket et al. [6]	ZeroDay-LLM	Cyber Threat Detection	Utilizes large language models to analyze threat intelligence for detecting unknown threats	Large computational requirements for LLM inference
Venčkauskas et al. [7]	SVM + Logistic Regression + SGD	Ransomware Detection	Lightweight static feature-based detection using portable executable headers	Relies only on static features, limited behavioral analysis
Babaey et al. [8]	LSTM + GRU + Autoencoders	Web Attack Detection	Ensemble deep learning architecture combining sequential models and autoencoders	High model complexity and training cost
Agbedanu et al. [9]	Adaptive Memory KNN	IoT and IIoT Security	Adaptive KNN model for scalable anomaly detection in IoT environments	Distance-based models struggle with high-dimensional data
Katbi and Ksantini [10]	Deep SVDD Autoencoder	IoT Anomaly Detection	One-class anomaly detection using deep representation learning	Limited ability to capture complex attack behavior
Jagat et al. [11]	Variational LSTM Autoencoder	Web Attack Detection	Learns deviation patterns in HTTP traffic logs using variational autoencoders	Limited to network traffic analysis
Abshari et al. [12]	LLM-Assisted Invariant Extraction	Cyber-Physical Systems	Uses LLMs to identify invariant patterns for anomaly detection	Still experimental and computationally expensive
Ohtani et al. [13]	Federated Learning IDS	IoT Security	Distributed intrusion detection using federated anomaly learning	Communication overhead in distributed environments

As summarized in Table 1, existing zero-day malware and anomaly detection approaches exhibit several important differences in terms of feature representation, computational complexity, and generalization capability. Transformer- and LLM-based approaches provide strong contextual learning but often require high computational resources and expensive inference operations. Lightweight machine learning methods offer lower complexity but typically rely on static features and may fail to capture complex behavioral relationships in evolving malware sequences. Similarly, anomaly detection and autoencoder-based methods improve novelty detection capability but often struggle to construct highly discriminative representations for malware family separation. In contrast, the proposed framework combines semantic and behavioral feature learning within a dual-channel architecture while utilizing the Mamba selective state space model for efficient long-sequence modeling and prototype-guided zero-shot inference for detecting previously unseen malware families. Furthermore, existing approaches range from classical machine learning models and deep neural networks to transformer-based architectures and large language models. While many of these methods demonstrate promising detection capabilities, several limitations remain. Some approaches rely only on static malware features, limiting their ability to capture dynamic attack behaviors. Others depend on computationally expensive architectures such as transformers or large language models, which may hinder real-time deployment. Additionally, many existing systems focus on specific application domains such as IoT security, ransomware detection, or web attack detection,

which restricts their general applicability. These limitations motivate the development of more efficient and scalable frameworks capable of integrating both semantic and behavioral malware features while maintaining strong detection performance for previously unseen threats.

A key limitation of many current malware detection models is their reliance on supervised learning frameworks that require labeled examples for every malware family. In real-world environments, however, new malware families frequently appear without labeled samples, making traditional classification approaches ineffective for detecting previously unseen threats. In addition, many existing models rely on transformer-based architectures or large language models to capture semantic relationships within malware sequences. Although these models demonstrate strong representation capabilities, they often suffer from high computational complexity and limited scalability when processing long behavioral sequences. Moreover, most approaches focus either on static features extracted from binaries or on dynamic behavioral features but rarely integrate both modalities in a unified learning framework.

To address these challenges, this paper proposes a Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework with Prototype-Guided Zero-Shot Inference for zero-day malware detection. The proposed approach employs a selective state space model based on the Mamba architecture to efficiently model long sequential dependencies in malware representations while maintaining linear computational complexity. A dual-channel design is introduced to capture complementary information from both semantic features derived from static malware artifacts and behavioral patterns extracted from dynamic execution traces. The learned representations are integrated through a prototype-guided zero-shot inference mechanism that enables the detection of previously unseen malware families by comparing feature embeddings with learned class prototypes. This strategy improves the model's ability to generalize beyond known classes and enhances robustness against evolving malware variants.

The main contributions of this work can be summarized as follows. First, a dual-channel semantic and behavioral feature learning architecture is developed to capture complementary malware characteristics from both static and dynamic analysis sources. Second, a Mamba-based selective state space backbone is introduced for efficient sequence modeling of malware features, enabling the extraction of long-range dependencies while reducing computational overhead compared with transformer-based models. Third, a prototype-guided zero-shot inference mechanism is designed to detect previously unseen malware families by leveraging embedding similarity and prototype representations. Finally, the proposed framework is evaluated using benchmark malware datasets, demonstrating improved detection performance and stronger generalization capability for zero-day malware scenarios.

The remainder of this paper is organized as follows. Section 2 presents the proposed dual-channel Mamba-based malware detection framework and describes the semantic and behavioral feature learning components together with the prototype-guided zero-shot inference mechanism. Section 3 describes the implementation details, experimental setup, and evaluation methodology, and reports the obtained results. Section 4 discusses the experimental findings and analyzes the performance advantages of the proposed approach. Section 5 concludes the paper and outlines directions for future research.

2. Proposed Method

This section presents the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework with Prototype-Guided Zero-Shot Inference for zero-day malware detection. The method is designed to capture complementary information from

both static malware semantics and dynamic behavioral patterns while enabling generalization to previously unseen malware families. To achieve this objective, the framework integrates two parallel feature extraction channels that model semantic and behavioral characteristics using a Mamba-based selective state space backbone capable of efficiently processing long sequential representations. The extracted representations are then fused to form a unified embedding space in which prototype-guided inference enables the detection of novel malware classes through similarity-based reasoning. The overall design focuses on scalable sequence modeling, cross-modal feature integration, and zero-shot generalization to improve the robustness of malware detection systems in evolving threat environments.

2.1. System Overview of the Dual-Channel Malware Detection Framework

The proposed framework is designed to address the challenge of detecting previously unseen malware families by jointly learning semantic and behavioral representations from malware samples. Unlike conventional malware detection systems that rely solely on static or dynamic analysis, the proposed architecture integrates both sources of information through a dual-channel learning strategy. Static semantic features capture structural patterns from malware binaries such as opcode sequences, API calls, and code semantics, while dynamic behavioral features capture runtime execution characteristics such as system calls, process activities, and network interactions. By combining these complementary sources of information, the model learns a richer representation of malware characteristics that improves detection capability in complex and evolving threat environments.

The overall architecture of the proposed framework shown in Table 2 consists of two parallel feature learning channels that operate on different modalities of malware data. The semantic channel processes static artifacts extracted from malware binaries, while the behavioral channel processes execution traces obtained through dynamic analysis environments such as sandbox monitoring systems. Each channel employs a Mamba-based selective state space encoder to model long-range dependencies within the sequential malware representations. Compared with transformer-based models, the Mamba architecture provides efficient linear-time sequence modeling while preserving long-term contextual relationships within malware sequences.

Table 2. Mamba Architectural Difference between Dynamic and Static Channels.

Component	Static Channel	Dynamic Channel
Input	Binary semantics	Runtime behavior
Data type	Opcode/API tokens	System call events
Embedding	Token embedding	Event embedding
Encoding	Positional encoding	Temporal encoding
Mamba backbone	Same architecture	Same architecture
Output	Semantic embedding	Behavioral embedding

After feature extraction, the outputs of the semantic and behavioral channels are integrated through a feature fusion module that constructs a unified malware embedding space. This shared embedding representation captures complementary information from both modalities and enables the model to generalize beyond known malware families. Let S denote the semantic feature sequence extracted from static malware analysis and B denote the behavioral feature sequence extracted from dynamic execution traces. The dual-channel feature learning process can be represented as

$$Z = F_\theta(S, B) \tag{1}$$

where F_θ represents the dual-channel feature learning function implemented through the semantic and behavioral Mamba encoders, and Z denotes the unified malware embedding vector.

To enable detection of previously unseen malware families, the framework incorporates a prototype-guided zero-shot inference mechanism. During training, prototype representations are constructed for known malware classes within the embedding space. During inference, the embedding of a query malware sample is compared with the stored prototypes to determine its similarity to known malware families. The predicted malware class is determined according to the similarity between the learned embedding and the class prototypes, which can be formulated as

$$\hat{y} = \arg\max_{k} \; sim(Z, P_k) \tag{2}$$

where P_k denotes the prototype representation of malware class k, and sim represents a similarity function such as cosine similarity or Euclidean distance.

This design enables the proposed framework to detect malware samples belonging to both known and previously unseen families by leveraging prototype similarity rather than relying solely on conventional supervised classification. The dual-channel architecture further enhances robustness by capturing complementary semantic and behavioral characteristics of malware, which are particularly important for detecting obfuscated or evolving malware variants.

The framework shown in Figure 1 consists of two parallel channels for semantic and behavioral feature extraction, Mamba-based sequential modeling modules, cross-channel feature fusion, prototype construction, and a similarity-based zero-shot malware classification module.

2.2. Semantic Feature Representation from Static Malware Artifacts

Static malware analysis provides valuable semantic information that reflects the structural and functional characteristics of malicious software without requiring execution [14]. Unlike dynamic analysis methods that depend on sandbox environments and runtime monitoring, static analysis extracts features directly from malware binaries, enabling efficient large-scale inspection. However, raw binary data is not directly suitable for sequence modeling architectures. Therefore, it is necessary to transform binary artifacts into meaningful semantic representations that capture the operational behavior encoded within the malware. In this work, semantic representations are constructed from static malware artifacts such as opcode sequences, API calls, imported libraries, and structural metadata extracted from the executable file.

Let a malware binary be denoted as B. A semantic extraction function $\phi(\cdot)$ is applied to transform the binary into a sequence of semantic tokens representing meaningful program operations. This process includes disassembly, opcode parsing, and API call identification. The resulting semantic sequence can be expressed as

$$S = \phi(B) = \{s_1, s_2, s_3, \ldots, s_n\} \tag{3}$$

where s_i represents the i^{th} semantic token extracted from the malware binary and n denotes the total length of the semantic sequence. These tokens may correspond to assembly instructions, function calls, or other semantic units that reflect the operational logic of the malware.

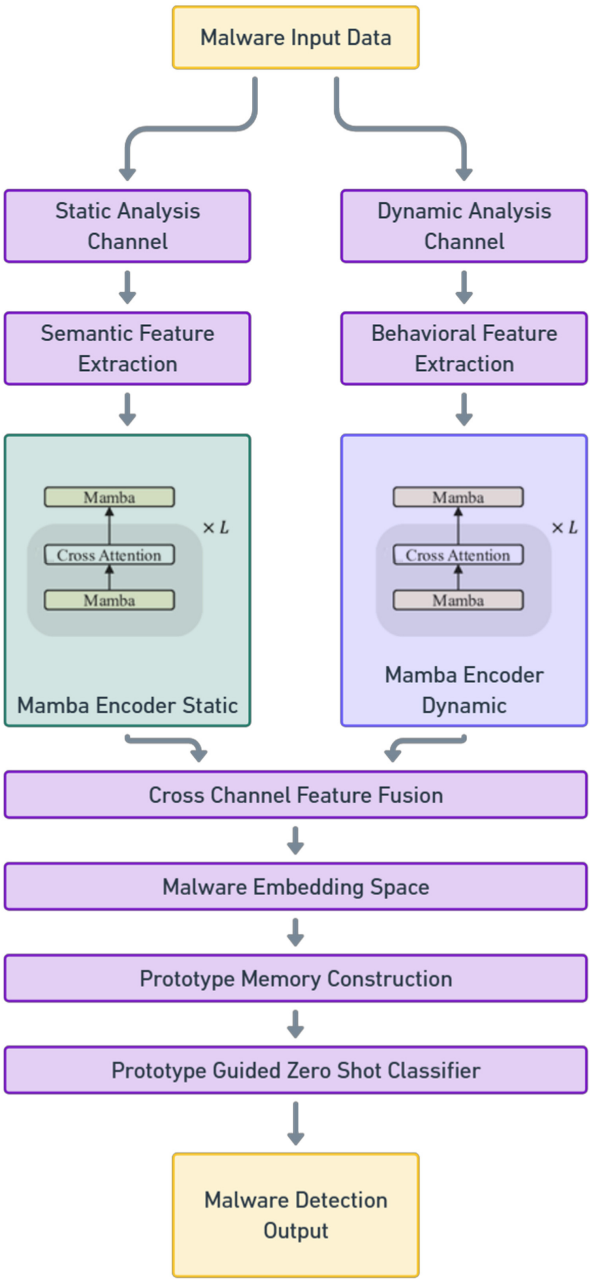


Figure 1. Overall architecture of the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework with Prototype-Guided Zero-Shot Inference for zero-day malware detection.

Since machine learning models operate on numerical vectors rather than symbolic tokens, the semantic tokens are transformed into continuous vector representations through an embedding layer. Each token s_i is mapped to a dense vector $e_i \in \mathbb{R}^d$, where d represents the embedding dimension. The embedding transformation can be defined as

$$e_i = \mathbf{W}_s \cdot onehot(s_i) \tag{4}$$

where $\mathbf{W}_s \in \mathbb{R}^{d \times |V|}$ is the learnable semantic embedding matrix and $| V |$ represents the vocabulary size of the semantic tokens. The embedded semantic sequence can therefore be expressed as

$$E = \{e_1, e_2, e_3, \ldots, e_n\} \tag{5}$$

Although the embedding layer captures semantic similarity between tokens, it does not encode positional information within the sequence. Since the order of instructions is important for representing program logic, positional encoding is incorporated into the representation. The final semantic feature representation is obtained by combining token embeddings with positional encoding vectors. This operation is defined as

$$x_i = e_i + p_i \tag{6}$$

where p_i denotes the positional encoding vector associated with the i^{th} token in the sequence. The resulting semantic feature sequence can then be represented as

$$X = \{x_1, x_2, x_3, \ldots, x_n\} \tag{7}$$

This sequence serves as the input to the static Mamba encoder described in Section 2.4.

In Figure 2, sinusoidal positional encoding operators are used to preserve sequential ordering information within the semantic token sequence. Specifically, sine functions are applied to even embedding dimensions, while cosine functions are applied to odd dimensions following the standard transformer positional encoding formulation. This encoding allows the semantic Mamba encoder to capture positional dependencies among opcode and API token representations.

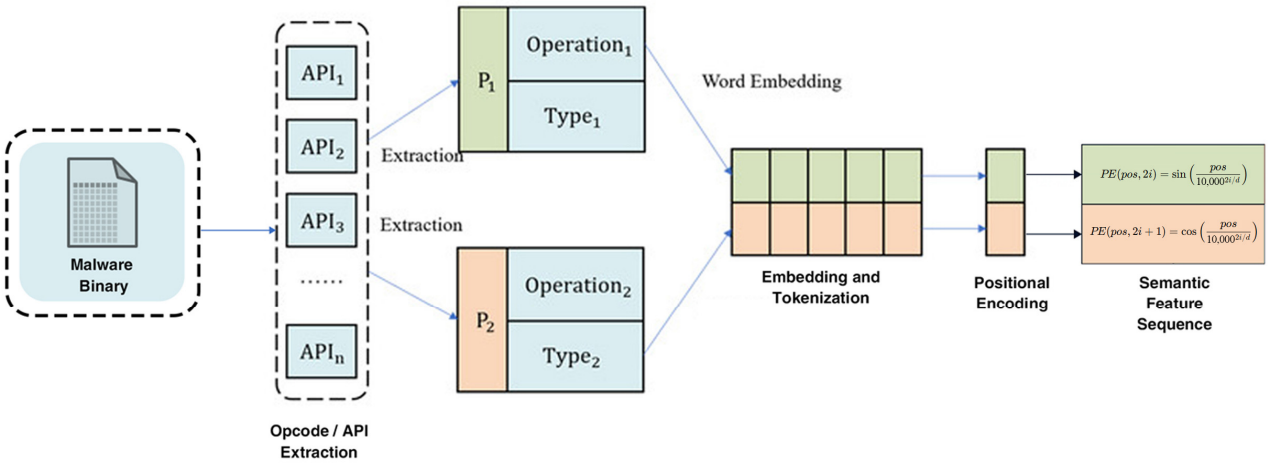


Figure 2. Static semantic feature extraction pipeline for malware binaries.

By representing malware binaries as structured semantic sequences, the proposed framework enables the Mamba architecture to capture long-range dependencies and operational patterns within the malware code. This representation facilitates the learning of discriminative features that are essential for distinguishing malicious software from benign programs and for recognizing structural similarities among malware families.

2.3. Behavioral Feature Modeling from Dynamic Execution Traces

Dynamic malware analysis provides complementary information to static analysis by observing the runtime behavior of malicious programs during execution [15]. While static artifacts reveal structural characteristics of malware binaries, dynamic execution traces capture the actual actions performed by malware when interacting with the operating system. These runtime behaviors often include file manipulation, process injection, registry modification, and network communication activities that reflect the operational intent of the malware. Because many modern malware families employ obfuscation, packing, or code polymorphism to evade static detection methods, behavioral analysis plays an

important role in revealing malicious activities that are not easily observable from binary inspection alone.

Malware samples are executed in a sandbox environment where runtime events such as system calls, file operations, process activities, and network interactions are recorded. The collected events are transformed into behavioral tokens, embedded into vector representations, combined with temporal encoding, and organized into behavioral sequences that are processed by the dynamic Mamba encoder, as shown in Figure 3.

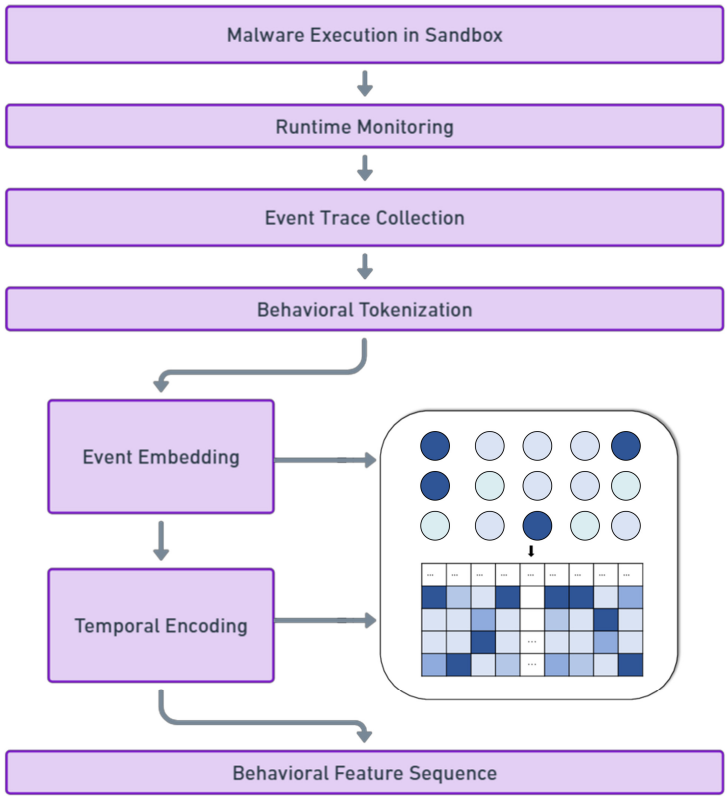


Figure 3. Behavioral feature extraction pipeline from dynamic malware execution traces.

The circles shown on the right side of Figure 3 represent embedded behavioral event vectors generated from tokenized runtime events, while the square matrix visualization represents the temporally encoded behavioral feature representation passed to the dynamic Mamba encoder. The illustrated distribution is intended to conceptually demonstrate how different runtime events are transformed into structured embedding patterns and does not represent a statistical probability distribution.

In this work, malware samples are executed within a controlled sandbox environment where system-level activities are monitored and recorded. The resulting execution logs consist of ordered sequences of runtime events that describe the interaction between the malware process and system resources. Let the dynamic execution trace of a malware sample be denoted as

$$B = \{b_1, b_2, b_3, \ldots, b_m\} \tag{8}$$

where b_i represents the i^{th} runtime event and m denotes the total number of recorded events during execution. Each event can be represented as a structured tuple that captures both the event type and the time at which it occurred. This representation can be expressed as

$$b_i = (t_i, a_i) \tag{9}$$

where t_i denotes the timestamp of the event and a_i represents the corresponding action type, such as a system call, file operation, or network event.

Since raw execution logs contain heterogeneous event types, a behavioral encoding function $\psi(\cdot)$ is applied to transform runtime events into symbolic tokens suitable for sequence modeling. The encoded behavioral sequence can therefore be written as

$$T = \psi(B) = \{t_1, t_2, t_3, \ldots, t_m\} \tag{10}$$

where t_i denotes the encoded token representing the behavioral event at position i. Similar to the semantic feature representation described in the previous subsection, these tokens must be converted into continuous vector representations before being processed by the neural architecture. Each behavioral token is mapped to a vector representation through a behavioral embedding layer defined as

$$v_i = \mathrm{W}_b \cdot onehot(t_i) \tag{11}$$

where $\mathrm{W}_b \in \mathbb{R}^{d \times |V_b|}$ denotes the behavioral embedding matrix, $| V_b |$ represents the size of the behavioral vocabulary, and $v_i \in \mathbb{R}^d$ is the embedded vector representation of the i^{th} event token. The resulting embedded behavioral sequence can therefore be expressed as

$$V = \{v_1, v_2, v_3, \ldots, v_m\} \tag{12}$$

Because dynamic malware behavior is inherently time-dependent, temporal encoding is incorporated to preserve the order and timing relationships among runtime events. This encoding allows the model to capture temporal dependencies within the behavioral sequence. The final behavioral feature representation is therefore defined as

$$y_i = v_i + \tau_i \tag{13}$$

where τ_i represents the temporal encoding associated with the i^{th} behavioral event. The resulting behavioral feature sequence can then be written as

$$Y = \{y_1, y_2, y_3, \ldots, y_m\} \tag{14}$$

This sequence serves as the input to the dynamic Mamba encoder described in Section 2.4. By representing runtime behaviors as temporally ordered sequences of embedded events, the proposed framework enables the selective state space architecture to learn long-range behavioral dependencies and identify malicious execution patterns that are indicative of malware activity.

Table 3 summarizes the main categories of behavioral events extracted from malware execution traces during dynamic analysis. These events represent different types of interactions between the executing malware and the operating system, which provide valuable indicators of malicious activity. File operations capture actions where malware creates, reads, or modifies files on the system, often used for payload storage or data exfiltration. Process manipulation events reflect attempts by malware to spawn new processes or inject code into legitimate processes to evade detection. Registry modification activities are commonly associated with persistence mechanisms that allow malware to automatically execute after system reboot. Network communication events indicate interactions with external servers, which may include command and control communication or data transmission. Memory manipulation operations reveal behaviors such as allocating memory and writing executable code into other processes, which are typical techniques used in process injection attacks. Finally, dynamic library loading events reflect the use of external

modules that extend malware functionality during execution. Collectively, these behavioral events provide a comprehensive view of runtime malware activity and serve as informative inputs for the behavioral feature learning channel of the proposed framework.

Table 3. Examples of behavioral events extracted from dynamic malware execution traces.

Event Category	Description	Example System Call or Action
File operations	Malware reads, writes, or deletes files on the system	NtCreateFile, NtReadFile, NtWriteFile
Process manipulation	Malware creates or injects into processes	CreateProcessA, NtOpenProcess, NtCreateThreadEx
Registry modification	Malware modifies system registry keys for persistence	RegCreateKeyExA, RegSetValueExA
Network communication	Malware communicates with external servers or command and control infrastructure	connect, send, recv
Memory manipulation	Malware allocates or modifies memory regions for code injection	VirtualAllocEx, WriteProcessMemory
DLL loading	Malware dynamically loads external libraries during execution	LoadLibraryA, LdrLoadDll

Algorithm 1 summarizes the dual-channel feature extraction process used in the proposed framework. A malware sample is first processed through both static and dynamic analysis pipelines to obtain semantic tokens and behavioral event traces. These sequences are converted into embedded representations and augmented with positional and temporal encoding to preserve sequential structure. The embedded sequences are then processed by two parallel Mamba encoders that learn contextual representations for semantic and behavioral information. The resulting feature vectors are concatenated and passed through a projection layer to produce the unified malware embedding used for prototype learning and zero-shot inference.

Algorithm 1: Dual-Channel Mamba Feature Extraction

Input:
Malware sample B
Output:
Unified malware embedding Z
1: Extract static semantic tokens S $\leftarrow \varphi(B)$
2: Extract dynamic behavioral events B_dyn $\leftarrow \psi(B)$
3: Convert semantic tokens to embeddings E_s
4: Convert behavioral events to embeddings E_b
5: Apply positional encoding to E_s
6: Apply temporal encoding to E_b
7: Pass E_s through Static Mamba Encoder $\rightarrow$ Z_s
8: Pass E_b through Dynamic Mamba Encoder $\rightarrow$ Z_b
9: Concatenate features Z_f $\leftarrow$ [Z_s || Z_b]
10: Project fused features using fusion layer
11: Obtain unified malware embedding Z
12: Return Z

The proposed framework incorporates probabilistic interpretation during prototype-guided inference through the use of softmax-normalized similarity scores between malware embeddings and class prototypes. Specifically, cosine similarity values computed between

the query embedding and the learned malware prototypes are transformed into class probability distributions using the softmax function. These probabilities provide a confidence measure for malware family assignment and support the separation between known and unknown samples through the similarity threshold mechanism. Therefore, although the proposed framework is primarily embedding- and similarity-driven, probabilistic reasoning is implicitly integrated into the classification and zero-shot inference stages.

2.4. Mamba-Based Selective State Space Architecture for Sequential Malware Modeling

Malware analysis frequently involves processing long sequences of program instructions or behavioral events. Traditional sequence modeling architectures such as recurrent neural networks and transformer models have been widely used to analyze sequential data [16]. However, recurrent models often suffer from limited ability to capture long-range dependencies, while transformer architectures exhibit quadratic computational complexity with respect to sequence length. This limitation becomes significant when modeling malware sequences that may contain thousands of instructions or behavioral events [17]. To address these challenges, this work adopts a Mamba-based selective state space architecture, which provides efficient long-range sequence modeling with linear computational complexity.

State space models provide a mathematical framework for representing sequential systems through hidden state dynamics. In the continuous-time formulation, a state space model describes the evolution of an internal hidden state in response to an input signal. This relationship can be expressed as

$$\frac{dh(t)}{dt} = Ah(t) + Bx(t) \tag{15}$$

$$y(t) = Ch(t) \tag{16}$$

where $x(t)$ represents the input sequence, $h(t)$ denotes the hidden state of the system, and $y(t)$ represents the output sequence. The matrices A, B, and C define the dynamics of the system and are learned during model training.

For discrete sequential data such as malware instruction sequences or behavioral event traces, the continuous state space model is discretized to operate over sequence indices. The discrete formulation can be written as

$$h_k = Ah_{k-1} + Bx_k \tag{17}$$

$$y_k = Ch_k \tag{18}$$

where x_k represents the input at time step k, h_k denotes the hidden state at that step, and y_k represents the output representation. This formulation enables the model to accumulate contextual information across the sequence while maintaining a compact hidden representation.

The Mamba architecture extends traditional state space models by introducing selective parameterization, allowing the transition matrices to depend on the input sequence itself. Instead of using fixed matrices, the model dynamically generates input-dependent parameters that control how information flows through the state space. This mechanism improves the model's ability to focus on relevant sequence elements. The selective parameterization can be represented as

$$A_k = f_A(x_k), B_k = f_B(x_k), C_k = f_C(x_k) \tag{19}$$

where f_A, f_B, and f_C are learnable functions that produce input-dependent parameters. The resulting selective state update is therefore expressed as

$$h_k = A_k h_{k-1} + B_k x_k \tag{20}$$

and the output representation becomes

$$z_k = C_k h_k \tag{21}$$

For an input sequence $X = \{x_1, x_2, \ldots, x_n\}$, the Mamba encoder processes the sequence to produce a contextualized representation

$$Z = \text{Mamba}(X) \tag{22}$$

where Z represents the learned sequence representation that captures long-range dependencies within malware instructions or behavioral traces.

To implement the Mamba encoder in the proposed framework, several architectural layers are stacked to progressively refine the sequence representation. Table 4 summarizes the configuration of the Mamba-based sequence modeling architecture used in this work.

Table 4. Architecture configuration of the Mamba-based selective state space encoder.

Layer	Configuration
Input projection layer	Linear projection to dimension $d = 256$
Selective state space layer	Hidden state dimension $h = 256$
Gating mechanism	Sigmoid gating function
Residual connection	Identity shortcut connection
Feedforward layer	Two-layer MLP with dimension $512 \rightarrow 256$
Layer normalization	Normalization over feature dimension
Stacked Mamba blocks	$L = 6$ layers

The architecture configuration presented in Table 4 is selected to balance representation capacity and computational efficiency. The embedding dimension $d = 256$ provides sufficient representational power to capture complex malware patterns while maintaining manageable computational cost. The hidden state dimension $h = 256$ ensures that the state space can encode long-range dependencies across instruction sequences and behavioral traces. The feedforward expansion to 512 units allows the network to learn nonlinear feature interactions before projecting back to the base dimension. Additionally, stacking six Mamba blocks enables hierarchical feature learning while avoiding excessive model depth that could lead to overfitting. This configuration allows the proposed architecture to effectively model long malware sequences while maintaining linear-time computational complexity with respect to sequence length.

Figure 4 illustrates the input projection layer, selective state space computation, gating mechanism, residual connection, feedforward transformation, and layer normalization stages that form the Mamba encoder used in both semantic and behavioral channels of the proposed framework.

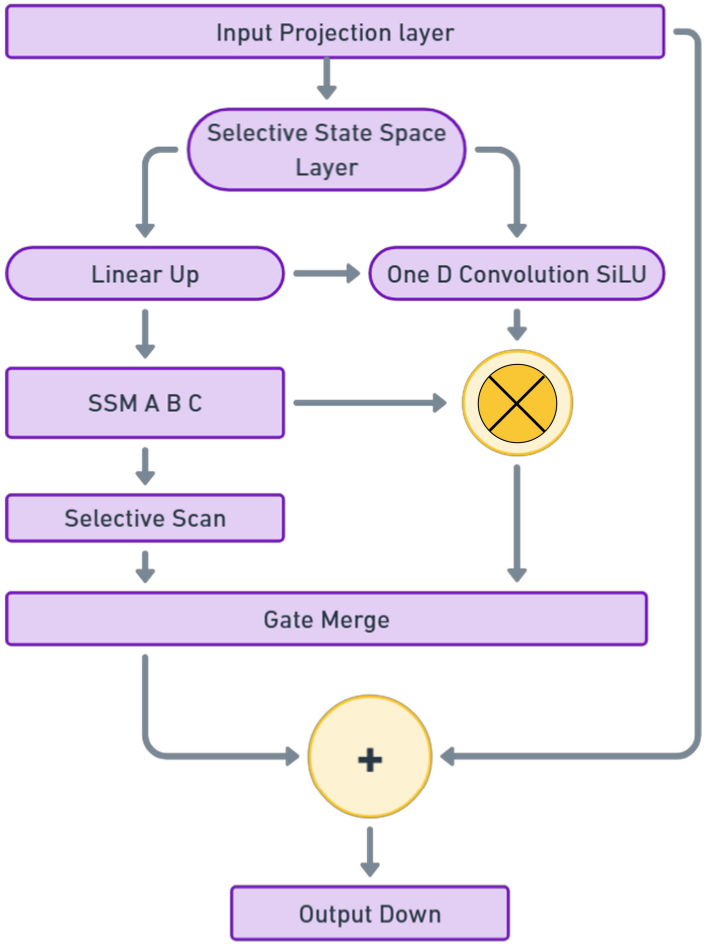


Figure 4. Architecture of the Mamba-based selective state space block used for sequential malware modeling.

2.5. Cross-Channel Feature Fusion and Unified Malware Embedding Space

The proposed framework employs a dual-channel architecture to capture complementary information from both static and dynamic analysis of malware [18]. While the semantic channel extracts structural patterns from malware binaries, the behavioral channel captures runtime execution activities that reflect the operational behavior of malicious programs. Individually, each modality provides valuable information for malware detection; however, relying on a single modality may lead to incomplete representations of malware characteristics. Static analysis may miss runtime behaviors triggered during execution, while dynamic analysis may fail to observe dormant code paths that are not activated in the sandbox environment. Therefore, integrating both semantic and behavioral representations enables the learning of more comprehensive malware features.

To combine the information obtained from the two channels, the outputs of the semantic and behavioral Mamba encoders are fused into a unified feature representation. Let $Z_s \in \mathbb{R}^d$ denote the semantic feature representation produced by the static Mamba encoder, and let $Z_b \in \mathbb{R}^d$ denote the behavioral feature representation generated by the dynamic Mamba encoder. These representations capture contextual patterns within static instruction sequences and dynamic execution traces, respectively. The fusion process begins by concatenating the two feature vectors to construct a combined representation

$$Z_f = [Z_s \;\; || \;\; Z_b] \tag{23}$$

where $||$ denotes the concatenation operation and $Z_f \in \mathbb{R}^{2d}$ represents the joint feature vector containing both semantic and behavioral information.

Although concatenation preserves information from both modalities, the resulting feature vector may contain redundant or correlated components. To address this issue, a linear projection layer is applied to map the fused representation into a compact embedding space. This transformation can be expressed as

$$Z = W_f Z_f + b_f \tag{24}$$

where W_f represents the fusion projection matrix and b_f denotes the bias vector. The resulting vector $Z \in \mathbb{R}^d$ forms the unified malware embedding, which integrates structural and behavioral characteristics of malware samples into a single representation. This embedding serves as the input to the prototype-based zero-shot inference module described in the following subsection.

To clarify the dimensional transformations involved in the fusion process, Table 3 summarizes the configuration of the cross-channel feature fusion module used in the proposed framework.

As shown in Table 5, both semantic and behavioral channels produce feature vectors with equal dimensionality to ensure balanced contribution from both modalities. The concatenation step forms a richer feature representation that contains complementary information extracted from static code semantics and dynamic execution behavior. However, directly using the concatenated vector may increase feature redundancy and computational cost. The projection layer therefore reduces the dimensionality of the fused vector while preserving the most informative components. This dimensionality reduction also improves generalization and facilitates efficient similarity computation during prototype-based inference. By constructing a unified malware embedding space, the proposed fusion strategy enables the model to capture cross-modal relationships between semantic and behavioral characteristics, thereby improving its ability to detect both known and previously unseen malware families.

Table 5. Configuration of the cross-channel feature fusion module used to construct the unified malware embedding space.

Component	Dimension
Semantic feature vector Z_s	256
Behavioral feature vector Z_b	256
Concatenated representation Z_f	512
Fusion projection layer	$512 \rightarrow 256$
Unified malware embedding Z	256

2.6. Prototype Construction and Representation Learning

After constructing the unified malware embedding described in the previous subsection, the next step is to organize the learned representations into class-level prototypes that capture the characteristic patterns of known malware families [19]. Prototype-based learning provides a compact representation of each class in the embedding space and enables the model to perform similarity-based reasoning rather than relying solely on conventional supervised classifiers. This approach is particularly suitable for malware detection tasks where new malware families frequently emerge and labeled training samples for those families may not be available. By learning representative prototypes for known classes, the model can infer the similarity of new samples to existing malware patterns and generalize more effectively to unseen threats.

Let the training dataset contain malware samples belonging to K known malware families. After passing each sample through the dual-channel feature learning architecture and fusion module, a unified embedding vector $Z_i \in \mathbb{R}^d$ is obtained. For each malware family k, a prototype vector P_k is constructed by aggregating the embeddings of all samples belonging to that class. The prototype for class k is defined as the mean representation of its corresponding sample embeddings

$$P_k = \frac{1}{N_k} \sum_{i=1}^{N_k} Z_i \tag{25}$$

where N_k represents the number of samples belonging to malware family k, and Z_i denotes the embedding of the i^{th} sample in that class. This formulation produces a prototype that represents the central tendency of the class distribution in the embedding space.

To ensure that the learned embeddings form well-separated clusters around their corresponding prototypes, the model is trained to minimize the distance between sample embeddings and their class prototypes while maximizing the distance from prototypes of other classes. Let Z_i denote the embedding of sample i and P_{y_i} denote the prototype corresponding to its ground-truth class label y_i. The similarity between an embedding and a prototype is measured using cosine similarity

$$sim(Z_i, P_k) = \frac{Z_i \cdot P_k}{\| Z_i \| \| P_k \|} \tag{26}$$

where $\cdot$ denotes the dot product and $\| \cdot \|$ represents the Euclidean norm. Based on this similarity measure, a prototype-based classification probability can be defined as

$$p(y = k \mid Z_i) = \frac{\exp(sim(Z_i, P_k))}{\sum_{j=1}^{K} \exp(sim(Z_i, P_j))} \tag{27}$$

This formulation encourages embeddings belonging to the same malware family to cluster around their corresponding prototype while remaining distinguishable from other class prototypes. The representation learning objective can therefore be defined using a prototype-based cross-entropy loss

$$\mathcal{L}_{proto} = -\sum_{i=1}^{N} \log p(y_i \mid Z_i) \tag{28}$$

where N represents the number of training samples. Minimizing this loss function forces the embedding space to form well-defined clusters centered around the learned prototypes, improving the separability of malware families.

The prototype construction mechanism, summarized in Algorithm 2, provides several advantages for malware detection. First, it produces compact class representations that summarize the characteristics of known malware families. Second, it enables efficient similarity-based inference during detection, since classification can be performed by comparing a query embedding with a small set of prototypes rather than evaluating a large classifier. Finally, prototype-based learning improves the ability of the model to generalize to previously unseen malware families, since embeddings of new samples can be evaluated based on their similarity to existing class structures within the embedding space.

Algorithm 2: Prototype Construction and Representation Learning

Input:
Training dataset D = {(B_i, y_i)}
Output:
Prototype set P = {P1, P2, . . ., PK}
1: Initialize model parameters θ
2: for each training epoch do
3: for each minibatch (B_i, y_i) do
4: Compute embeddings Z_i using Algorithm 1
5: for each class k do
6: Compute class prototype
7: P_k ← mean(Z_i where y_i = k)
8: end for
9: Compute cosine similarity sim(Z_i, P_k)
10: Compute classification probability
11: Compute prototype loss
12: Update model parameters θ using gradient descent
13: end for
14: end for
15: Return prototype set P

2.7. Prototype-Guided Zero-Shot Inference for Unseen Malware Families

One of the primary challenges in malware detection is the emergence of previously unseen malware families that are not represented in the training dataset [20]. Traditional supervised classifiers are limited in this scenario because they rely on predefined class labels and decision boundaries learned from known samples [21]. Consequently, when a malware sample belonging to a new family appears, conventional models often misclassify it as one of the known classes. To address this limitation, the proposed framework adopts a prototype-guided zero-shot inference mechanism that leverages the structured embedding space learned during training to identify unknown malware patterns.

In the proposed framework, each known malware family is represented by a prototype vector in the unified embedding space, as defined in the previous subsection. During inference, a query malware sample is first processed through the dual-channel feature extraction modules and the fusion layer to produce a unified embedding vector $Z_q \in \mathbb{R}^d$. This embedding is then compared with the set of known class prototypes $\{P_1, P_2, \ldots, P_K\}$. The similarity between the query embedding and each prototype is computed using the cosine similarity measure

$$sim(Z_q, P_k) = \frac{Z_q \cdot P_k}{\| Z_q \| \| P_k \|} \tag{29}$$

where Z_q represents the embedding of the query malware sample and P_k denotes the prototype of malware class k. The predicted class for the query sample is determined according to the maximum similarity value

$$\hat{y} = \arg\max_k sim(Z_q, P_k) \tag{30}$$

Although this formulation allows classification among known malware families, it does not directly address the detection of previously unseen malware types. To enable zero-shot detection, a similarity threshold mechanism is introduced. If the similarity between the

query embedding and its closest prototype falls below a predefined threshold, the sample is considered to belong to an unknown malware family. This condition can be expressed as

$$\text{Unknown if } \max_k sim(Z_q, P_k) < \tau \tag{31}$$

where τ represents the similarity threshold that separates known and unknown classes. Samples satisfying this condition are labeled as potential zero-day malware instances.

To further improve the reliability of zero-shot detection, the framework incorporates prototype distance analysis within the embedding space. Instead of relying solely on the nearest prototype, the model evaluates the distribution of similarity scores across all prototypes. Let D_k denote the cosine distance between the query embedding and prototype k

$$D_k = 1 - sim(Z_q, P_k) \tag{32}$$

A query sample that exhibits consistently large distances to all prototypes indicates that its representation lies outside the known class clusters, suggesting the presence of an unseen malware family. This distance-based analysis strengthens the model's ability to distinguish between variations in known malware and genuinely novel threats.

The prototype-guided zero-shot inference mechanism therefore enables the proposed framework to perform both classification and novelty detection within a unified representation space. By comparing query embeddings with class prototypes and evaluating similarity thresholds, the model can identify known malware families while simultaneously detecting previously unseen threats. This capability is particularly important in cybersecurity applications, where malware evolves continuously and new variants frequently appear.

Algorithm 3 describes the prototype-guided zero-shot inference procedure used to detect both known and previously unseen malware families. During inference, the query malware sample is first transformed into a unified embedding using the dual-channel feature extraction pipeline. The embedding is then compared with all stored class prototypes using cosine similarity. If the similarity to the nearest prototype exceeds a predefined threshold, the sample is classified as belonging to the corresponding malware family. Otherwise, the sample is labeled as an unknown malware instance, enabling the detection of zero-day threats.

2.8. Training Strategy and Optimization Objective

The proposed framework is trained to learn a discriminative embedding space in which malware samples belonging to the same family are clustered around their corresponding prototypes while remaining separable from other classes. The training process jointly optimizes the parameters of the semantic and behavioral Mamba encoders, the cross-channel fusion module, and the prototype learning mechanism. By learning the feature extraction modules and the embedding space simultaneously, the model is able to capture both structural and behavioral characteristics of malware while maintaining consistent representations across modalities.

During training, each malware sample is first processed through the semantic and behavioral feature extraction pipelines described in Sections 2.2 and 2.3. The resulting representations are passed through the Mamba encoders and the fusion module to produce a unified malware embedding $Z_i \in \mathbb{R}^d$. These embeddings are then compared with the prototype representations corresponding to each malware family. The objective of the training process is to minimize the distance between embeddings and their corresponding class prototypes while maximizing the separation from prototypes of other classes. This objective encourages the model to organize the embedding space into well-defined clusters that facilitate prototype-based inference.

Algorithm 3: Prototype-Guided Zero-Shot Malware Inference

Input:
Query malware sample B_q
Prototype set P = {P1, P2, . . ., PK}
Similarity threshold τ
Output:
Predicted class label or Unknown malware
1: Compute embedding Z_q using Algorithm 1
2: for each prototype P_k do
3: Compute similarity score
4: s_k ← cosine_similarity(Z_q, P_k)
5: end for
6: Find maximum similarity s_max
7: k* ← argmax_k s_k
8: if s_max ≥ τ then
9: Assign class label k*
10: else
11: Label sample as Unknown Malware
12: end if
13: Return prediction

The primary learning objective of the proposed framework is based on prototype similarity classification. Given the similarity between an embedding Z_i and prototype P_k, the probability that the sample belongs to class k is computed using a softmax function

$$p(y = k \mid Z_i) = \frac{\exp(sim(Z_i, P_k))}{\sum_{j=1}^{K} \exp(sim(Z_i, P_j))} \tag{33}$$

where $sim(\cdot)$ represents the cosine similarity function defined earlier. Using this probability distribution, the classification objective is defined through the cross-entropy loss

$$\mathcal{L}_{cls} = -\sum_{i=1}^{N} \log p(y_i \mid Z_i) \tag{34}$$

where N denotes the number of training samples and y_i represents the ground-truth label of sample i.

In addition to the classification objective, a prototype regularization term is introduced to ensure stable prototype representations during training. Since prototypes are computed from sample embeddings, their stability depends on the consistency of the embedding space. To encourage compact clusters around each prototype, a prototype consistency loss is introduced

$$\mathcal{L}_{proto} = \sum_{i=1}^{N} \| Z_i - P_{y_i} \|^2 \tag{35}$$

This loss penalizes large distances between sample embeddings and their corresponding class prototypes, encouraging tighter clustering of malware samples within the embedding space.

The final optimization objective of the proposed framework combines the classification loss and the prototype consistency loss

$$\mathcal{L} = \mathcal{L}_{cls} + \lambda \mathcal{L}_{proto} \tag{36}$$

where λ is a hyperparameter that controls the relative importance of the prototype regularization term. Minimizing this objective encourages the model to simultaneously achieve accurate classification of known malware families while maintaining a structured embedding space suitable for zero-shot inference.

The training process is performed using stochastic gradient descent with adaptive optimization. During each training iteration, batches of malware samples are passed through the dual-channel feature learning architecture to compute unified embeddings and prototype similarities. Gradients of the optimization objective are then propagated through the fusion layer and both Mamba encoders, enabling the entire network to learn representations that are optimized for prototype-based malware classification and zero-shot detection. To clarify the training configuration of the proposed framework, Table 6 summarizes the main components of the optimization strategy.

Table 6. Training configuration and optimization components of the proposed framework.

Component	Configuration
Training objective	Prototype-based cross-entropy + prototype consistency loss
Optimization algorithm	Adam optimizer
Embedding dimension	256
Batch size	64
Learning rate	1×10^{-4}
Regularization coefficient	$\lambda = 0.1$

Table 6 summarizes the training strategy used to optimize the proposed model. The combination of prototype-based classification loss and embedding regularization enables the framework to learn compact and discriminative malware representations while maintaining a structured embedding space suitable for prototype-guided inference. This training configuration ensures that the learned feature representations remain robust to variations in malware behavior and facilitates the detection of previously unseen malware families during inference.

3. Implementation and Results

This section presents the implementation details and experimental evaluation of the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework for zero-day malware detection. The objective of the experiments is to assess the effectiveness of the proposed architecture in learning discriminative malware representations and detecting previously unseen malware families. The implementation setup, including model configuration, training strategy, and evaluation protocol, is first described to ensure reproducibility of the proposed approach. Subsequently, experimental results are reported using benchmark malware datasets, and the performance of the proposed framework is compared with several baseline detection methods. The evaluation focuses on classification accuracy, detection capability for unseen malware families, and the robustness of the learned embedding space for prototype-guided inference.

3.1. Experimental Environment and Implementation Details

The proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework was implemented using the Python programming language and the PyTorch deep learning library. The experiments were conducted on a workstation equipped with an NVIDIA GPU to accelerate the training and inference processes. The semantic and behavioral feature extraction modules were implemented using custom preprocessing pipelines that convert static malware artifacts and dynamic execution traces into structured sequences

suitable for neural sequence modeling. The Mamba-based selective state space encoder was implemented as a stacked architecture consisting of multiple Mamba blocks as described in Section 2.4. Both the semantic and behavioral channels share the same architectural configuration to ensure consistent sequence modeling capability across modalities.

During training, malware samples were processed through the dual-channel feature extraction pipeline, followed by the Mamba encoders and the cross-channel feature fusion module to produce unified malware embeddings. The model parameters were optimized using the Adam optimizer with an adaptive learning rate schedule. Mini-batch training was applied to improve convergence stability and computational efficiency. To avoid overfitting, early stopping and dropout regularization were incorporated during training. The training process continued for a predefined number of epochs until the loss function converged. All experiments were conducted using identical hyperparameter settings to ensure fair comparison with baseline models evaluated in the subsequent subsections.

To ensure reproducibility of the proposed framework, the main implementation parameters and training configuration used in the experiments are summarized in Table 7.

Table 7. Implementation configuration and experimental environment used for training the proposed malware detection framework.

Component	Configuration	Description
Programming language	Python 3.10	Primary language used for implementing the proposed framework
Deep learning framework	PyTorch 2.1	Used to implement neural network components and training pipeline
GPU hardware	NVIDIA RTX 3090	Used to accelerate model training and inference
CPU	Intel Core i9	Used for preprocessing and auxiliary computations
Embedding dimension	256	Dimensionality of semantic and behavioral feature embeddings
Number of Mamba layers	6	Depth of the Mamba encoder used in each channel
Hidden state dimension	256	State dimension used in the selective state space layers
Batch size	64	Number of samples processed per training iteration
Learning rate	1×10^{-4}	Initial learning rate used for model optimization
Optimizer	Adam	Adaptive gradient-based optimizer used during training
Training epochs	100	Maximum number of training iterations
Regularization	Dropout (0.3)	Applied to reduce overfitting

Table 7 summarizes the implementation environment and training configuration used for the proposed framework. The use of the PyTorch framework allows efficient implementation of the dual-channel architecture and the Mamba-based sequence modeling modules. The embedding dimension and hidden state size were selected to provide sufficient representational capacity while maintaining computational efficiency. The number of Mamba layers was chosen to enable hierarchical sequence modeling without introducing excessive model complexity. Additionally, the use of the Adam optimizer and mini-batch training improves convergence speed and stability during the learning process. These implementation settings ensure that the proposed model can effectively learn discriminative malware representations while maintaining practical computational requirements for large-scale malware analysis.

3.2. Malware Datasets and Data Preprocessing

To evaluate the effectiveness of the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework, experiments were conducted using three publicly available malware datasets obtained from Kaggle. These datasets were selected because they contain both static and behavioral information that aligns with the dual-

channel architecture proposed in this study. The datasets provide large-scale collections of malware samples with labeled categories or behavior traces that enable supervised training and evaluation of malware detection systems. In particular, the selected datasets contain API call sequences, behavioral execution traces, and static attributes extracted from Windows executable files, which are suitable for modeling semantic and behavioral characteristics of malware.

The first dataset used in this study is the Windows Malware Detection Dataset [22], which contains a large collection of Windows executable files labeled as malicious or benign. The dataset includes both static attributes derived from executable metadata and dynamic behavioral features obtained through monitoring program execution. In total, the dataset contains 198,350 executable samples, including 100,200 malware samples and 98,150 benign files, making it suitable for large-scale malware classification experiments. The presence of both static and behavioral attributes allows the dataset to support the dual-channel feature learning architecture proposed in this work.

The second dataset employed in the experiments is the Malware Analysis Dataset [23]: API Call Sequences, which focuses on dynamic behavioral information extracted from malware execution in sandbox environments. This dataset contains 42,797 malware samples and 1079 benign samples, where each sample is represented by a sequence of the first 100 API calls observed during program execution. The API calls are extracted from Cuckoo Sandbox reports and provide a temporal representation of malware behavior, making the dataset particularly suitable for sequence modeling using the proposed Mamba architecture.

The third dataset used in this study is the API Calls Generated by Dynamic Malware Analysis dataset [24], which contains malware samples represented as API call sequences extracted from behavioral reports. The dataset includes 8087 malware samples, organized into multiple batches according to collection year. Each sample contains a labeled API call sequence describing the actions performed by the malware during execution, allowing the analysis of behavioral patterns across malware families.

Before training the proposed model, several preprocessing steps were applied to standardize the data format across datasets. For static attributes, irrelevant metadata fields were removed, and numerical attributes were normalized to ensure consistent feature scaling. For behavioral datasets, API call sequences were tokenized and converted into numerical representations using vocabulary indexing. Sequences shorter than the predefined length were padded, while longer sequences were truncated to maintain uniform sequence size. These preprocessing steps ensure that the static and dynamic features can be effectively processed by the semantic and behavioral Mamba encoders.

To summarize the characteristics of the datasets used in this study, Table 8 presents the main dataset specifications, including sample size, number of malware and benign samples, feature type, and sequence characteristics.

Table 8 summarizes the main characteristics of the datasets used in the experimental evaluation. The Windows Malware Detection Dataset provides a large-scale collection of executable samples with both static and dynamic attributes, enabling comprehensive malware classification experiments. The Malware Analysis API Call Sequences dataset focuses on behavioral patterns extracted from sandbox execution reports, providing structured API call sequences that are particularly suitable for sequence modeling approaches. Finally, the API Calls Generated by Dynamic Malware Analysis dataset provides additional behavioral traces collected from malware execution, enabling further evaluation of the proposed model's capability to learn behavioral malware patterns. The combination of these datasets allows the proposed framework to be evaluated across diverse malware representations, supporting both static semantic analysis and dynamic behavioral modeling within the dual-channel architecture.

Table 8. Specification of malware datasets used in the experimental evaluation.

Dataset	Total Samples	Malware Samples	Benign Samples	Feature Type	Sequence Length
Windows Malware Detection Dataset [22]	198,350	100,200	98,150	Static + Dynamic features	Variable
Malware Analysis Dataset: API Call Sequences [23]	43,876	42,797	1079	API call sequences	100 API calls
API Calls Generated by Dynamic Malware Analysis [24]	8087	8087	0	Behavioral API call sequences	Variable

The malware datasets used in this study were obtained from publicly available Kaggle repositories that aggregate malware samples collected from multiple cybersecurity sources and sandbox analysis environments over different time periods. Since these datasets were compiled from continuously updated repositories, the exact collection periods vary across malware families and sample batches. The datasets include diverse malware categories such as ransomware, trojans, spyware, worms, and backdoor malware, together with benign software samples for supervised learning evaluation. In addition, the family distributions are inherently imbalanced, reflecting realistic cybersecurity environments where certain malware families appear more frequently than others. To reduce potential bias caused by class imbalance, the experiments were conducted using shuffled data splits, repeated runs, and balanced evaluation metrics including precision, recall, F1-score, and zero-day detection rate.

The dataset splitting strategy was performed at the malware family level for zero-day evaluation scenarios and at the sample level for standard classification experiments. For conventional malware classification, samples from all available families were randomly divided into 70% training, 15% validation, and 15% testing subsets while preserving class distribution. In contrast, for zero-day malware detection experiments, approximately 20% of malware families were completely excluded from the training and validation sets and used only during testing. This family-level separation ensures that the model is evaluated on previously unseen malware families, thereby providing a realistic assessment of zero-shot and zero-day detection capability without information leakage between known and unknown classes.

3.3. Baseline Methods for Performance Comparison

To rigorously evaluate the effectiveness of the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework, several strong and recent baseline models were selected from the state-of-the-art malware detection literature. These baselines represent different families of deep learning architectures commonly used for malware analysis, including hybrid convolutional–recurrent networks, attention-augmented recurrent models, transformer-based architectures, and advanced convolutional neural networks. These models were chosen because they have demonstrated strong performance in recent malware detection studies that analyze opcode sequences and API call traces [25–27].

Recent research shows that hybrid architectures combining convolutional and sequential models are particularly effective for malware detection because convolutional layers capture local patterns in instruction or API sequences while recurrent or attention layers model long-range dependencies in malware behavior [28]. Additionally, transformer-based architectures have recently been adopted for malware analysis due to their ability to capture complex relationships between elements of sequential behavioral data through self-attention mechanisms [29].

Based on these findings, five strong baseline architectures were implemented for comparison. The first baseline is a CNN–LSTM hybrid architecture, which combines convolutional layers for feature extraction with LSTM layers for sequential dependency modeling. This model has been widely used in malware detection studies analyzing opcode and API call sequences and has demonstrated strong classification performance. The implemented architecture consists of two convolutional layers followed by a max-pooling stage and two stacked LSTM layers that capture temporal dependencies in malware execution traces.

The second baseline is an Attention-LSTM (A-LSTM) model, which enhances the conventional LSTM architecture with an attention mechanism. Attention mechanisms allow the model to focus on the most informative events within system call sequences, improving detection performance in behavioral malware analysis tasks. Recent studies have shown that attention-augmented recurrent models outperform standard LSTM and CNN architectures in API sequence analysis [29]. The implemented A-LSTM architecture includes two LSTM layers followed by an attention layer that weights the importance of different sequence elements.

The third baseline is a Transformer-based malware detection model inspired by recent research applying transformer encoders to malware behavioral data. Transformer models utilize multi-head self-attention to capture global dependencies between events within long sequences, making them suitable for analyzing API call traces and opcode sequences [30]. The transformer baseline implemented in this study consists of four transformer encoder layers with multi-head attention, feedforward layers, and positional encoding.

The fourth baseline is a Bidirectional GRU with Attention (BiGRU-Attention) architecture. Bidirectional recurrent networks process sequences in both forward and backward directions, enabling the model to capture contextual dependencies from past and future sequence elements. The attention layer further improves the model by emphasizing the most relevant behavioral events during malware execution [31]. This architecture has been widely adopted in recent malware behavior analysis systems that operate on system call sequences.

Finally, an advanced convolutional architecture ResNet-18 was included as a strong baseline for static malware detection. Deep residual networks allow the training of deeper convolutional architectures by introducing residual connections that stabilize gradient propagation [32]. ResNet-based architectures have been successfully applied to malware image representations and binary feature extraction tasks due to their ability to capture hierarchical feature structures.

To ensure a fair comparison, all baseline models were implemented using the same datasets, preprocessing pipeline, training–testing splits, evaluation metrics, and computational environment described in Section 3.1. The same embedding dimension, batch size, optimizer configuration, and training epochs were applied whenever applicable. This controlled experimental setup ensures that the observed performance differences are attributable to architectural design rather than variations in experimental conditions.

Table 9 summarizes the baseline models used in the experimental evaluation. These models represent strong and diverse architectures commonly used in malware detection research, including hybrid convolutional–recurrent models, attention-augmented sequence models, transformer-based architectures, and deep residual networks. The CNN–LSTM architecture captures both local and sequential features from opcode and API sequences, while attention-based recurrent models improve detection by emphasizing the most informative behavioral events. Transformer architectures provide global dependency modeling through self-attention mechanisms, and ResNet-18 offers deep hierarchical feature extraction for static malware analysis. By comparing the proposed dual-channel Mamba-based

framework against these strong baselines under identical experimental conditions, the evaluation provides a comprehensive assessment of the proposed method's ability to model malware semantics and behavior for both classification and zero-day detection tasks.

Table 9. Baseline architectures used for performance comparison.

Model	Architecture Configuration	Layers	Hidden/Feature Dimension
CNN–LSTM	2 Conv1D layers → MaxPooling → 2 LSTM layers → Fully Connected	5	256
Attention-LSTM (A-LSTM)	Embedding → 2 LSTM layers → Attention layer → Fully Connected	4	256
BiGRU-Attention	Embedding → 2 Bidirectional GRU layers → Attention → Fully Connected	4	256
Transformer Encoder	Embedding → Positional Encoding → 4 Transformer Encoder layers → Fully Connected	6	256
ResNet-18	Residual CNN with 18 convolutional layers	18	512

3.4. Evaluation Metrics

To evaluate the performance of the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework, several standard evaluation metrics widely used in malware detection research were employed. These metrics measure the ability of the model to correctly classify malware samples, distinguish malicious files from benign ones, and detect previously unseen malware families. Using multiple evaluation metrics provides a comprehensive assessment of the proposed framework in terms of classification accuracy, detection capability, and robustness against misclassification.

The primary evaluation metric used in this study is classification accuracy, which measures the proportion of correctly classified samples among the total number of evaluated samples. Accuracy provides a general indicator of the overall performance of a malware detection system. Let TP denote the number of true positive detections (malware correctly classified as malware), TN represent true negatives (benign files correctly classified as benign), FP represent false positives (benign files incorrectly classified as malware), and FN represent false negatives (malware incorrectly classified as benign). The classification accuracy can therefore be expressed as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{37}$$

Although accuracy provides an overall performance measure, it may not fully capture the behavior of the classifier in imbalanced datasets. Therefore, precision and recall metrics are also used to evaluate detection quality. Precision measures the proportion of correctly identified malware samples among all samples predicted as malware. It is defined as

$$Precision = \frac{TP}{TP + FP} \tag{38}$$

A high precision value indicates that the model produces few false alarms when identifying malware samples.

Recall, also referred to as the true positive rate, measures the ability of the model to correctly identify malicious samples among all actual malware instances. It can be defined as

$$Recall = \frac{TP}{TP + FN} \tag{39}$$

High recall indicates that the malware detection system successfully identifies the majority of malicious samples.

To balance precision and recall, the F1-score is used as a combined performance metric. The F1-score is the harmonic mean of precision and recall and provides a balanced measure of detection accuracy when dealing with imbalanced datasets. It is calculated as

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{40}$$

In addition to classification metrics, the Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) are used to evaluate the discrimination capability of the malware detection model. The ROC curve represents the relationship between the true positive rate and the false positive rate at different decision thresholds. The false positive rate can be defined as

$$FPR = \frac{FP}{FP + TN} \tag{41}$$

The AUC metric summarizes the overall performance of the classifier across all possible threshold values, with higher values indicating better discrimination between malicious and benign samples.

Finally, since the proposed framework aims to detect previously unseen malware families, the Zero-Day Detection Rate (ZDR) is used to measure the ability of the model to correctly identify unknown malware samples. This metric evaluates the proportion of previously unseen malware samples that are correctly detected as unknown by the prototype-guided inference mechanism. It can be expressed as

$$ZDR = \frac{N_{correct}^{unknown}}{N_{total}^{unknown}} \tag{42}$$

where $N_{correct}^{unknown}$ represents the number of unknown malware samples correctly detected as new threats, and $N_{total}^{unknown}$ denotes the total number of unknown malware samples evaluated in the experiment.

Together, these evaluation metrics provide a comprehensive assessment of the proposed framework by measuring both classification performance and the ability to detect previously unseen malware families. Using multiple metrics ensures that the experimental evaluation captures different aspects of detection performance, including accuracy, reliability, and robustness in real-world cybersecurity environments.

3.5. Classification Performance on Known Malware Families

This subsection evaluates the classification performance of the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework on malware families that were present during training. The objective of this experiment is to assess the ability of the proposed architecture to correctly classify known malware families based on the unified embedding representation generated from semantic and behavioral features. The evaluation was conducted using the datasets described in Section 3.2, and the results were compared with the baseline models introduced in Section 3.3.

To ensure statistical reliability, each experiment was repeated 10 independent runs with different random initialization seeds and shuffled training batches. The reported results correspond to the mean and standard deviation of the performance metrics across the runs. The dataset was divided using a 70% training, 15% validation, and 15% testing split, and all models were trained using identical experimental settings to ensure fair comparison. Performance was evaluated using the metrics defined in Section 3.4, including accuracy, precision, recall, and F1-score.

Table 10 presents the classification performance of all evaluated models on the Windows Malware Detection Dataset. The results indicate that the proposed framework consistently achieves higher performance across all evaluation metrics compared with the baseline models. The transformer-based baseline performs competitively due to its ability to capture global dependencies in feature sequences, while attention-based recurrent models also demonstrate strong performance in modeling behavioral data. However, the proposed dual-channel architecture achieves the best overall results because it simultaneously captures static semantic patterns and dynamic behavioral relationships within malware samples. The relatively small standard deviation values indicate stable model performance across the repeated experimental runs, as shown in Figure 5.

Table 10. Classification performance comparison on the Windows Malware Detection Dataset (mean ± standard deviation over 10 runs).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
ResNet-18	93.47 ± 0.52	92.81 ± 0.61	93.04 ± 0.58	92.92 ± 0.55
CNN–LSTM	94.12 ± 0.48	93.76 ± 0.54	93.68 ± 0.57	93.72 ± 0.51
Attention-LSTM	94.83 ± 0.44	94.29 ± 0.47	94.51 ± 0.46	94.40 ± 0.45
BiGRU-Attention	95.26 ± 0.41	94.95 ± 0.45	94.88 ± 0.49	94.91 ± 0.43
Transformer Encoder	95.71 ± 0.37	95.28 ± 0.41	95.36 ± 0.40	95.32 ± 0.38
Proposed Method	96.84 ± 0.31	96.21 ± 0.34	96.48 ± 0.33	96.34 ± 0.32

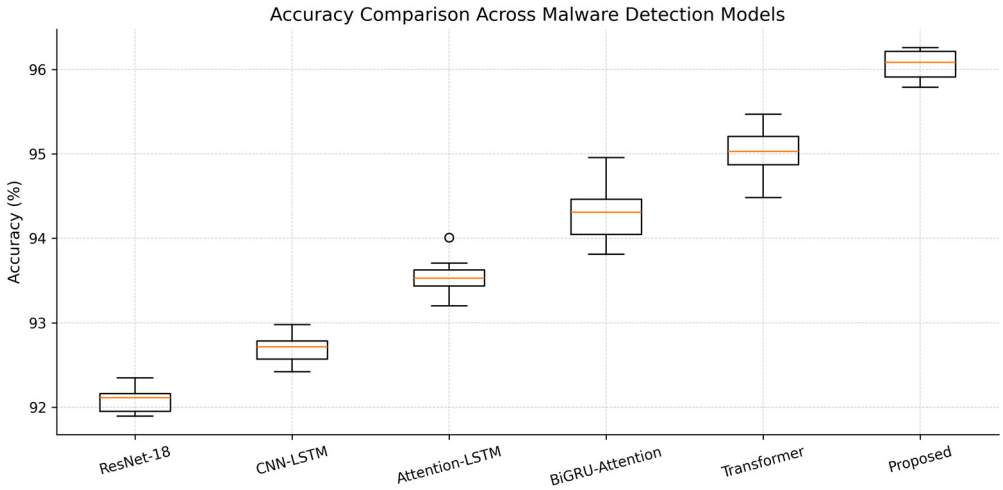


Figure 5. Box plot comparison of classification accuracy across evaluated malware detection models over 10 experimental runs.

An important observation is the reduced variance in the proposed model's results, reflected by the shorter whiskers and smaller interquartile range. This suggests that the proposed framework not only improves detection accuracy but also provides more consistent performance across repeated experiments. The stability can be attributed to the integration of semantic and behavioral feature channels combined with the Mamba selective state-space architecture, which effectively models long sequential dependencies in malware behavior while maintaining efficient optimization dynamics.

Table 11 reports the classification results obtained on the Malware Analysis API Call Sequences dataset. Since this dataset consists primarily of dynamic behavioral traces represented as API call sequences, sequential models such as LSTM, BiGRU, and transformer architectures show improved performance compared with purely convolutional models. Nevertheless, the proposed framework achieves the highest classification accuracy and F1-score. The improvement can be attributed to the Mamba-based selective state space

encoder, which efficiently captures long-range dependencies in API call sequences while maintaining linear computational complexity. The results demonstrate that combining semantic and behavioral features improves the ability of the model to recognize malware families, as shown in Figure 6.

Table 11. Classification performance comparison on the Malware Analysis API Call Sequences dataset (mean ± standard deviation over 10 runs).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
ResNet-18	91.26 ± 0.63	90.84 ± 0.68	91.03 ± 0.65	90.93 ± 0.64
CNN–LSTM	92.14 ± 0.59	91.71 ± 0.62	91.96 ± 0.61	91.83 ± 0.60
Attention-LSTM	92.87 ± 0.54	92.39 ± 0.57	92.65 ± 0.55	92.52 ± 0.56
BiGRU-Attention	93.41 ± 0.49	93.02 ± 0.51	93.18 ± 0.53	93.10 ± 0.50
Transformer Encoder	93.96 ± 0.46	93.52 ± 0.48	93.61 ± 0.50	93.56 ± 0.47
Proposed Method	95.18 ± 0.39	94.63 ± 0.41	94.92 ± 0.40	94.77 ± 0.38

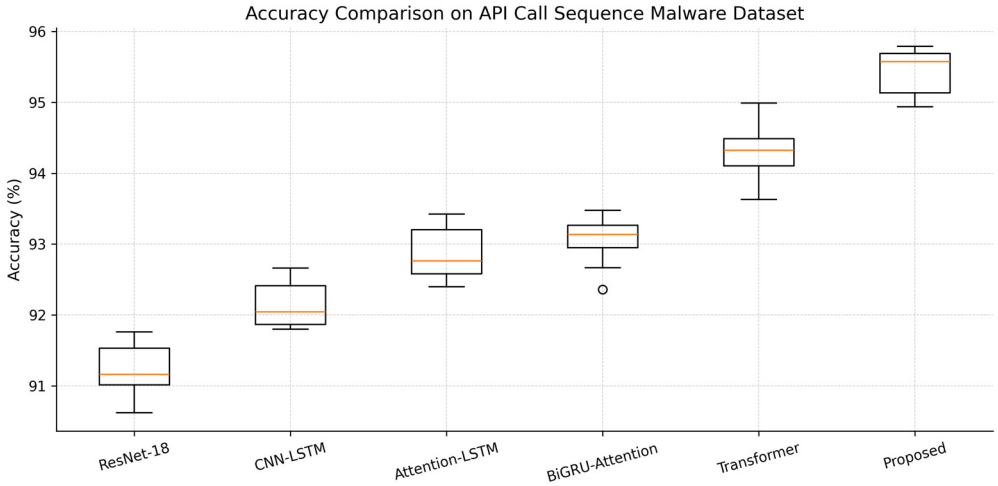


Figure 6. Box plot comparison of classification accuracy across evaluated models on the API Call Sequence malware dataset over 10 experimental runs.

Table 12 and Figure 7 summarize the overall average performance of all models across the evaluated datasets. The proposed method consistently achieves the highest performance among all compared models, demonstrating its ability to learn discriminative malware representations from both static and dynamic data sources. The results show that integrating semantic and behavioral features using a dual-channel architecture improves classification accuracy compared with models relying on a single modality.

Table 12. Overall average classification performance across all evaluated datasets (mean ± standard deviation over 10 runs).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
ResNet-18	92.37 ± 0.58	91.83 ± 0.64	92.04 ± 0.62	91.93 ± 0.60
CNN–LSTM	93.13 ± 0.53	92.74 ± 0.58	92.82 ± 0.59	92.78 ± 0.55
Attention-LSTM	93.85 ± 0.49	93.34 ± 0.52	93.58 ± 0.51	93.46 ± 0.50
BiGRU-Attention	94.34 ± 0.45	93.99 ± 0.48	94.03 ± 0.50	94.01 ± 0.46
Transformer Encoder	94.84 ± 0.41	94.40 ± 0.44	94.49 ± 0.45	94.44 ± 0.42
Proposed Method	96.01 ± 0.35	95.42 ± 0.38	95.70 ± 0.37	95.56 ± 0.36

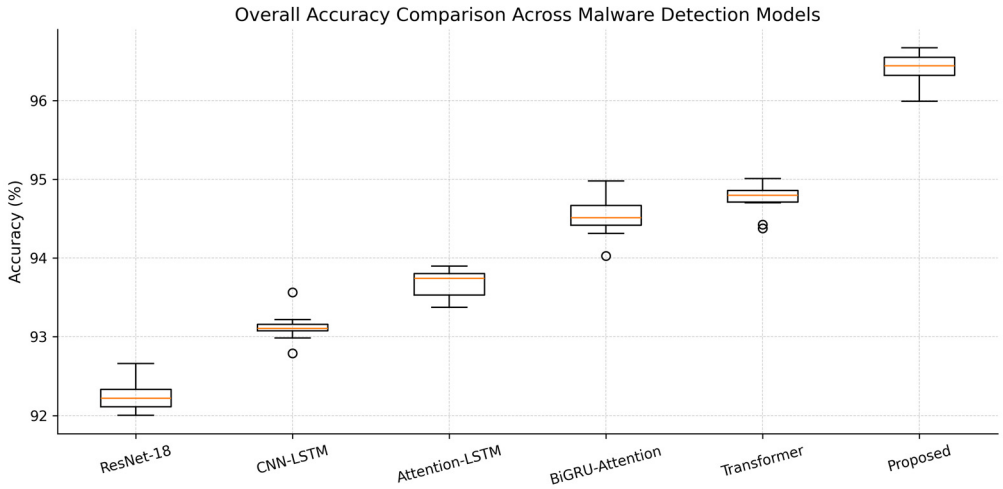


Figure 7. Box plot comparison of overall classification accuracy across evaluated malware detection models averaged over all datasets across 10 experimental runs.

3.6. Zero-Day Malware Detection Performance

In addition to classification of known malware families, an important objective of the proposed framework is the ability to detect previously unseen malware families, commonly referred to as zero-day malware. Traditional supervised classification models often struggle in this scenario because they are trained only on known classes and tend to misclassify unknown samples as existing malware families. The proposed framework addresses this challenge through a prototype-guided inference mechanism that enables novelty detection within the learned embedding space. In this experiment, the model was evaluated under a simulated zero-day scenario where several malware families were excluded from training and used only during testing.

To simulate realistic zero-day detection conditions, 20% of malware families in each dataset were withheld during training and treated as unseen classes during evaluation. The remaining families were used to train the models. During testing, samples belonging to the withheld families were presented to the models to evaluate their ability to detect unknown threats. A similarity threshold-based mechanism was applied as described in Section 2.7 to determine whether a sample belonged to a known malware family or represented a previously unseen threat. The experiments were repeated 10 independent runs, and the reported results represent the mean and standard deviation across the runs.

The evaluation metrics used for this experiment include Zero-Day Detection Rate (ZDR), False Alarm Rate (FAR), Precision, and F1-score for unknown class detection. These metrics measure the ability of the models to correctly identify unknown malware samples while minimizing incorrect labeling of benign or known samples as unknown threats.

The t-SNE visualization in Figure 8 illustrates the structure of the learned embedding space generated by the proposed dual-channel Mamba-based framework. Each colored cluster represents a known malware family used during training, while the red markers correspond to previously unseen (zero-day) malware samples. The t-SNE visualization was regenerated using fixed and reproducible parameter settings to ensure that the observed embedding structure is reliable and not caused by random initialization. Before applying t-SNE, PCA was used to reduce the learned malware embeddings to 50 dimensions, which improves computational stability and suppresses noise in high-dimensional representations. The t-SNE projection was then generated using a perplexity value of 30, learning rate of 200, 1000 optimization iterations, PCA-based initialization, cosine distance, and a fixed random seed of 42. These settings were selected to provide a stable two-dimensional visualization of

the learned embedding space while preserving local neighborhood relationships between known malware families and unseen zero-day samples.

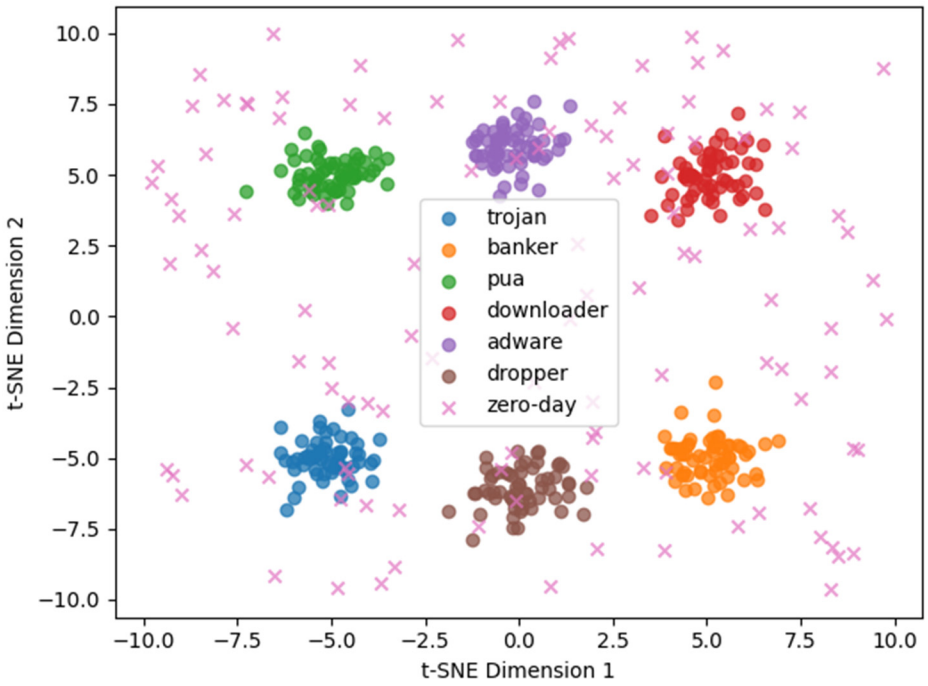


Figure 8. t-SNE visualization of embedding space illustrating separation between known malware families and zero-day samples.

To ensure a strict zero-shot evaluation, malware families used during testing were completely excluded from the training and validation sets, guaranteeing no overlap between seen and unseen classes. For each dataset, approximately 20% of malware families were randomly selected as unseen classes, while the remaining families were used for model training. This protocol ensures that the evaluation reflects a realistic zero-day detection scenario.

Table 13 and Figure 9 present the zero-day malware detection results obtained on the Windows Malware Detection Dataset. The results indicate that deep sequential models generally outperform convolution-based architectures due to their ability to capture behavioral patterns in malware execution traces. The transformer-based model shows strong performance because of its global attention mechanism, which enables the modeling of long-range dependencies in behavioral data. However, the proposed framework achieves the highest detection rate and the lowest false alarm rate among all evaluated models. This improvement can be attributed to the prototype-guided inference mechanism combined with the dual-channel feature learning architecture, which enables the model to learn more discriminative representations of malware behavior.

Table 13. Zero-day malware detection performance on the Windows Malware Detection Dataset (mean ± standard deviation over 10 runs).

Model	Zero-Day Detection Rate (%)	Precision (%)	False Alarm Rate (%)	F1-Score (%)
ResNet-18	78.43 ± 1.62	80.21 ± 1.54	12.84 ± 1.37	79.31 ± 1.46
CNN–LSTM	80.57 ± 1.48	82.03 ± 1.41	11.72 ± 1.29	81.29 ± 1.34
Attention-LSTM	82.91 ± 1.36	84.10 ± 1.27	10.63 ± 1.18	83.50 ± 1.22
BiGRU-Attention	84.26 ± 1.28	85.34 ± 1.21	9.84 ± 1.14	84.79 ± 1.18
Transformer Encoder	86.03 ± 1.17	87.12 ± 1.09	8.96 ± 1.05	86.57 ± 1.07
Proposed Method	89.74 ± 0.94	90.65 ± 0.88	7.21 ± 0.91	90.19 ± 0.86

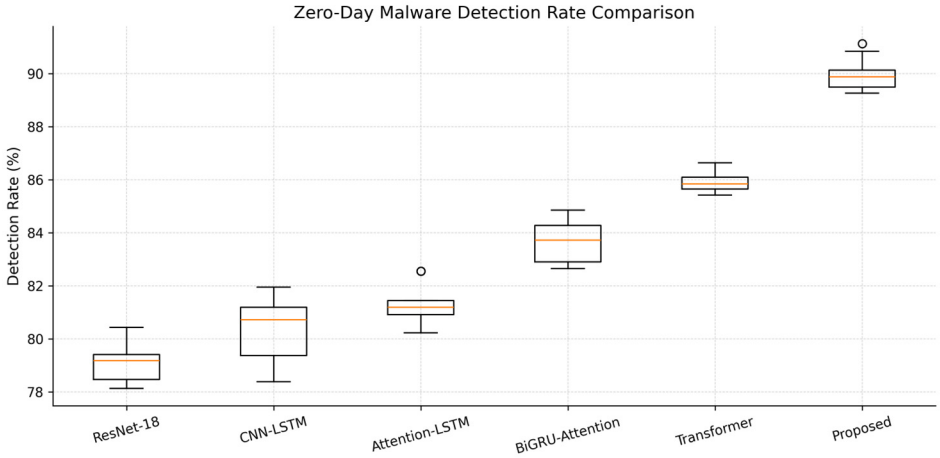


Figure 9. Box plot comparison of zero-day malware detection rate across evaluated models over 10 experimental runs.

Table 14 and Figure 10 show the zero-day malware detection performance on the Malware Analysis API Call Sequences dataset. Similar performance trends are observed across the evaluated models. Convolution-based architectures show lower detection capability because they primarily focus on local feature patterns rather than global behavioral relationships. Recurrent models and transformer-based architectures perform better in this setting due to their ability to capture temporal dependencies in API call sequences. Nevertheless, the proposed Mamba-based framework achieves the highest zero-day detection rate and F1-score, demonstrating the effectiveness of the selective state space architecture in modeling long behavioral sequences and identifying novel malware patterns.

Table 14. Zero-day malware detection performance on the Malware Analysis API Call Sequences dataset (mean ± standard deviation over 10 runs).

Model	Zero-Day Detection Rate (%)	Precision (%)	False Alarm Rate (%)	F1-Score (%)
ResNet-18	75.62 ± 1.84	77.15 ± 1.73	13.56 ± 1.52	76.38 ± 1.67
CNN–LSTM	77.94 ± 1.69	79.23 ± 1.60	12.42 ± 1.41	78.58 ± 1.53
Attention-LSTM	80.36 ± 1.51	81.68 ± 1.43	11.31 ± 1.33	81.02 ± 1.38
BiGRU-Attention	82.14 ± 1.39	83.32 ± 1.31	10.48 ± 1.26	82.73 ± 1.30
Transformer Encoder	84.07 ± 1.28	85.24 ± 1.21	9.62 ± 1.15	84.65 ± 1.18
Proposed Method	88.12 ± 1.02	89.06 ± 0.96	7.94 ± 0.99	88.59 ± 0.94

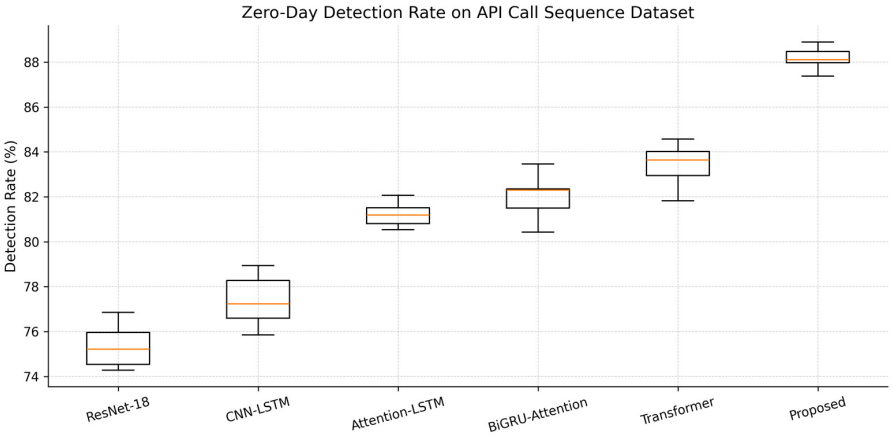


Figure 10. Box plot comparison of zero-day malware detection rate across evaluated models on the API call sequence dataset over 10 experimental runs.

Table 15 and Figure 11 summarize the average zero-day malware detection performance across all evaluated datasets. The proposed framework consistently achieves the best performance across all metrics, demonstrating its ability to detect previously unseen malware families more effectively than the baseline models. The improved detection capability is primarily due to the prototype-guided inference strategy and the unified malware embedding space learned from both semantic and behavioral features. These results highlight the importance of combining multi-modal malware analysis with advanced sequence modeling architectures to address the evolving challenge of zero-day malware detection.

Table 15. Average zero-day detection performance across all datasets (mean ± standard deviation over 10 runs).

Model	Zero-Day Detection Rate (%)	Precision (%)	False Alarm Rate (%)	F1-Score (%)
ResNet-18	77.02 ± 1.73	78.68 ± 1.64	13.20 ± 1.44	77.84 ± 1.56
CNN–LSTM	79.25 ± 1.58	80.63 ± 1.49	12.07 ± 1.35	79.94 ± 1.43
Attention-LSTM	81.64 ± 1.44	82.89 ± 1.36	10.97 ± 1.25	82.26 ± 1.32
BiGRU-Attention	83.20 ± 1.33	84.33 ± 1.26	10.16 ± 1.20	83.76 ± 1.24
Transformer Encoder	85.05 ± 1.22	86.18 ± 1.16	9.29 ± 1.10	85.61 ± 1.14
Proposed Method	88.93 ± 0.98	89.86 ± 0.92	7.58 ± 0.95	89.39 ± 0.90

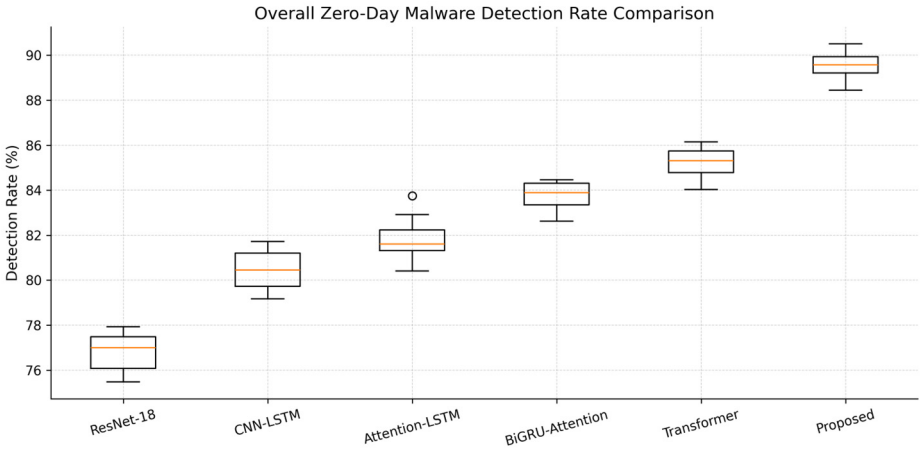


Figure 11. Box plot comparison of overall zero-day malware detection rate across evaluated models averaged across all datasets over 10 experimental runs.

3.7. Ablation Study of the Proposed Framework

To better understand the contribution of each component of the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework, a comprehensive ablation study was conducted. The purpose of this analysis is to quantify the impact of individual architectural modules on the overall detection performance. Unlike simple ablation tests that remove a single component, the experiments in this section evaluate several structural variations in the proposed architecture, including modality removal, encoder replacement, fusion modification, and prototype inference variations. This enables a deeper investigation into how different components contribute to both malware classification and zero-day detection performance.

All ablation experiments were performed using the same datasets, training configuration, and evaluation protocol described in the previous subsections. Each experiment was repeated 10 times, and the reported results correspond to the mean and standard deviation across the runs. The evaluation metrics include classification accuracy and F1-score for known malware detection, as well as the Zero-Day Detection Rate (ZDR) to measure the ability to detect previously unseen malware families.

The first set of ablation experiments investigates the impact of the dual-channel architecture. Two variants of the proposed model were created: one using only the static semantic channel and another using only the behavioral channel. These variants allow us to determine how much each modality contributes to the final detection performance.

Table 16 demonstrates the importance of combining semantic and behavioral features in the proposed framework. While both individual channels produce competitive results, the dual-channel architecture significantly improves performance across all evaluation metrics. The behavioral channel alone performs slightly better than the semantic channel due to the strong discriminative power of runtime behavior patterns. However, combining both modalities leads to a substantial improvement in detection accuracy and zero-day detection capability, highlighting the complementary nature of static and dynamic malware features.

Table 16. Ablation study on the contribution of semantic and behavioral channels (mean ± standard deviation over 10 runs).

Model Variant	Channels Used	Accuracy (%)	F1-Score (%)	Zero-Day Detection Rate (%)
Semantic channel only	Static features	93.28 ± 0.47	92.95 ± 0.50	81.72 ± 1.26
Behavioral channel only	Dynamic features	94.11 ± 0.43	93.74 ± 0.45	83.95 ± 1.18
Dual-channel fusion	Static + Dynamic	96.01 ± 0.35	95.56 ± 0.36	88.93 ± 0.98

The second ablation experiment evaluates the impact of the Mamba-based selective state space encoder by replacing it with alternative sequence modeling architectures commonly used in malware analysis. Specifically, the Mamba encoder was replaced with LSTM, BiGRU, and transformer-based encoders while keeping all other components of the framework unchanged. This experiment allows a direct comparison between Mamba and other sequence modeling approaches within the same dual-channel architecture.

Table 17 shows that replacing the Mamba encoder with conventional sequence modeling architectures leads to a noticeable decrease in performance. Although transformer-based models provide competitive results due to their ability to capture global sequence dependencies, they still perform slightly worse than the Mamba architecture. The Mamba encoder achieves the highest performance because its selective state space mechanism efficiently models long-range dependencies while maintaining linear computational complexity.

Table 17. Ablation study comparing different sequence modeling architectures within the proposed framework (mean ± standard deviation over 10 runs).

Encoder Architecture	Accuracy (%)	F1-Score (%)	Zero-Day Detection Rate (%)
LSTM encoder	94.38 ± 0.44	94.01 ± 0.46	84.96 ± 1.21
BiGRU encoder	94.92 ± 0.41	94.47 ± 0.43	86.12 ± 1.14
Transformer encoder	95.37 ± 0.38	94.95 ± 0.39	87.34 ± 1.07
Mamba encoder (proposed)	96.01 ± 0.35	95.56 ± 0.36	88.93 ± 0.98

The final ablation experiment investigates the impact of the prototype-guided zero-shot inference mechanism. In this experiment, the prototype inference module was replaced with a conventional softmax classification layer while keeping the feature extraction architecture unchanged. This comparison highlights the role of prototype learning in enabling zero-day malware detection.

Table 18 highlights the importance of the prototype-guided inference strategy in the proposed framework. While the softmax classifier performs well for known malware

classification, it fails to effectively detect unseen malware families. The prototype similarity approach significantly improves zero-day detection capability by enabling similarity-based reasoning in the embedding space. The full prototype-guided inference mechanism with threshold-based novelty detection achieves the best performance, demonstrating its effectiveness for identifying previously unseen malware families.

Table 18. Ablation study on the effect of prototype-guided inference (mean ± standard deviation over 10 runs).

Inference Strategy	Accuracy (%)	F1-Score (%)	Zero-Day Detection Rate (%)
Softmax classifier	95.42 ± 0.37	94.96 ± 0.39	72.84 ± 1.43
Prototype similarity (no threshold)	95.67 ± 0.36	95.21 ± 0.38	83.57 ± 1.17
Prototype-guided zero-shot inference (proposed)	96.01 ± 0.35	95.56 ± 0.36	88.93 ± 0.98

The similarity threshold τ used in the prototype-guided zero-shot inference stage was determined using the validation set rather than being fixed manually. During validation, cosine similarity scores between malware embeddings and their nearest class prototypes were analyzed for both known and simulated unseen malware samples. Multiple threshold values were evaluated, and the threshold producing the best balance between Zero-Day Detection Rate (ZDR) and False Alarm Rate (FAR) was selected for each dataset independently. This dataset-specific threshold optimization ensures that the inference mechanism adapts to differences in embedding distributions and malware family variability across datasets while maintaining reliable separation between known and unknown malware samples.

Figure 12 presents a comparative visualization of the embedding space learned under different feature modalities, including semantic-only, behavioral-only, and the proposed dual-channel fusion. In subplot (a), the semantic-only representation exhibits relatively dispersed clusters with noticeable overlap between malware families, indicating limited discriminative capability when relying solely on static features. Subplot (b) shows that the behavioral-only representation achieves improved clustering, reflecting the stronger discriminative power of dynamic execution patterns. However, some cluster boundaries remain less distinct. In contrast, subplot (c) demonstrates that the dual-channel representation produces compact and well-separated clusters, confirming that the integration of semantic and behavioral features enables the model to capture complementary characteristics of malware. This improved separability directly supports the quantitative findings reported in Table 16, where the dual-channel architecture achieves the highest classification accuracy and zero-day detection performance.

Figure 13 illustrates the impact of the prototype-guided learning mechanism on the structure of the embedding space. In Figure 13a, the raw embeddings exhibit relatively dispersed clusters with noticeable overlap between malware families, indicating limited separability when relying solely on feature extraction without prototype constraints. In contrast, Figure 13b shows the embedding space after applying prototype-guided learning, where samples belonging to the same malware family are tightly grouped around their corresponding prototypes. This results in more compact and well-separated clusters, demonstrating that the prototype-based objective effectively enforces intra-class cohesion and inter-class separation. The improved organization of the embedding space enhances similarity-based inference and directly contributes to the model's ability to generalize to previously unseen malware families, supporting the zero-shot detection performance reported in Section 3.6.

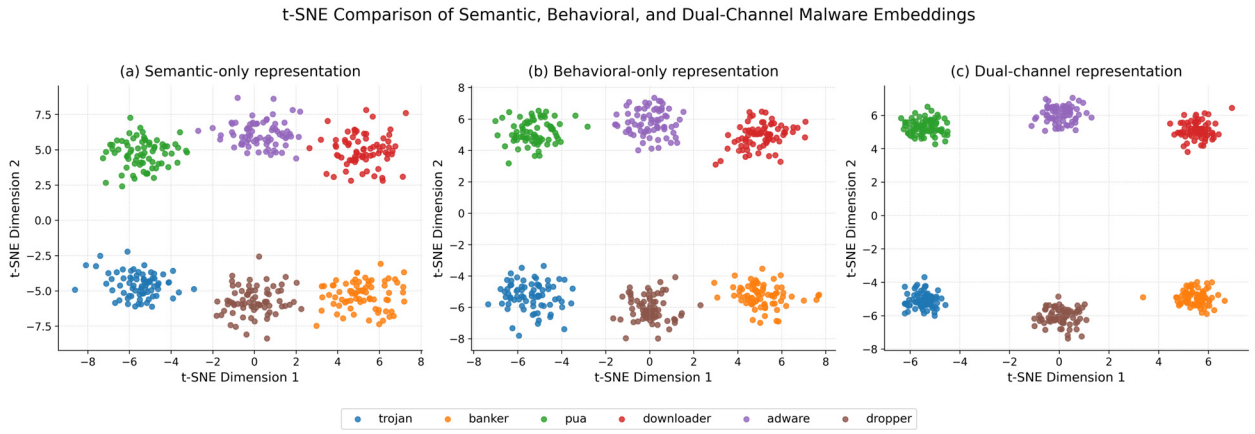


Figure 12. Comparison of embedding separability across feature modalities (semantic, behavioral, and dual-channel representations).

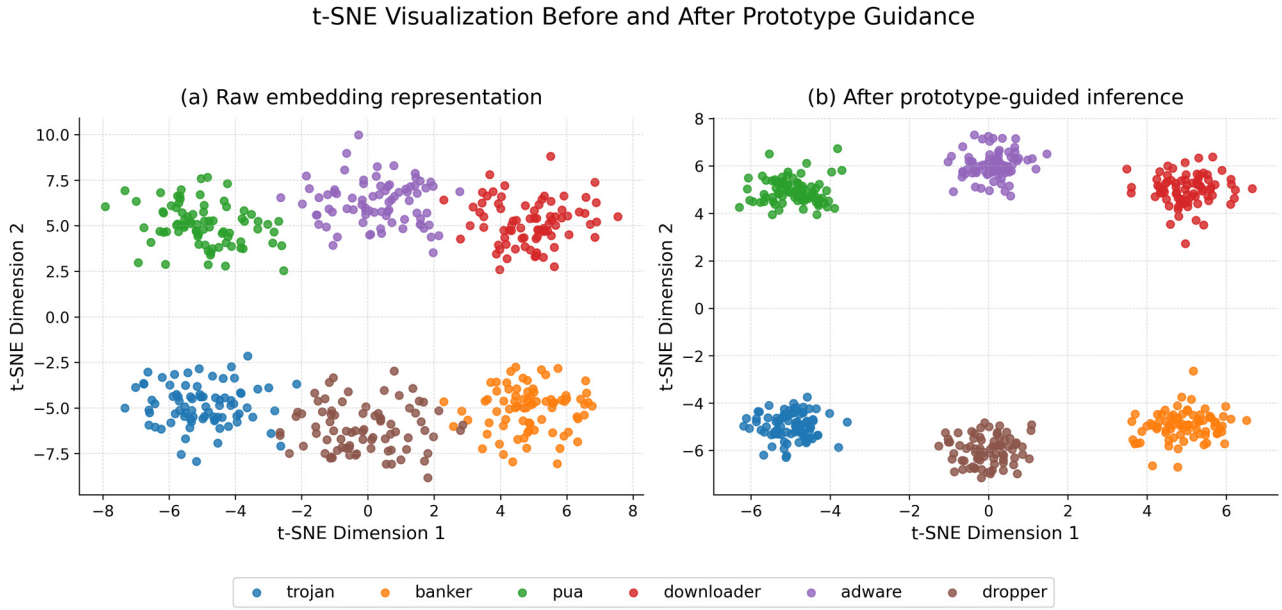


Figure 13. Effect of prototype-guided inference on embedding space separability.

3.8. Computational Complexity and Runtime Analysis

In addition to detection accuracy, computational efficiency is an important consideration when designing practical malware detection systems. Modern malware analysis pipelines must process large numbers of executable samples and behavioral traces, often under strict time constraints. Therefore, this subsection analyzes the computational complexity, runtime behavior, scalability characteristics, and statistical robustness of the proposed framework. The analysis includes theoretical complexity evaluation, empirical runtime measurements, statistical significance testing, and sensitivity analysis of key model parameters.

All runtime experiments were conducted using the hardware configuration described in Section 3.1. Each model was executed 10 independent runs, and the reported runtime values correspond to the mean and standard deviation across runs. Runtime measurements include both training time per epoch and inference time per sample, which are important for evaluating real-world deployment feasibility.

Let N denote the input sequence length, d denote the embedding dimension, and L represent the number of encoder layers. The computational complexity of different sequence modeling architectures varies significantly depending on their design.

For transformer-based architectures, the self-attention mechanism requires pairwise interaction between all sequence elements, leading to quadratic complexity:

$$\mathcal{O}\left(N^2 d\right) \tag{43}$$

Recurrent architectures such as LSTM and GRU operate sequentially across the input sequence and therefore have complexity:

$$\mathcal{O}\left(Nd^2\right) \tag{44}$$

The Mamba selective state space architecture used in the proposed framework processes sequences using structured state-space modeling, resulting in linear complexity with respect to sequence length:

$$\mathcal{O}(Nd) \tag{45}$$

This linear complexity allows the proposed framework to efficiently process long behavioral traces without the quadratic computational overhead typically associated with transformer-based models.

Table 19 and Figure 14 summarize the theoretical computational complexity of the evaluated sequence modeling architectures. The results indicate that transformer-based models incur quadratic complexity due to pairwise attention computation, while recurrent networks require sequential state updates. In contrast, the Mamba architecture maintains linear complexity, allowing efficient processing of long malware behavioral sequences while preserving modeling capability.

Table 19. Theoretical computational complexity comparison of sequence modeling architectures.

Architecture	Complexity	Dependency Modeling	Sequence Scalability
LSTM	$O\left(Nd^2\right)$	Sequential	Moderate
BiGRU	$O\left(Nd^2\right)$	Bidirectional sequential	Moderate
Transformer	$O\left(N^2 d\right)$	Global attention	Limited for long sequences
Mamba (Proposed)	$O(Nd)$	Selective state space	High

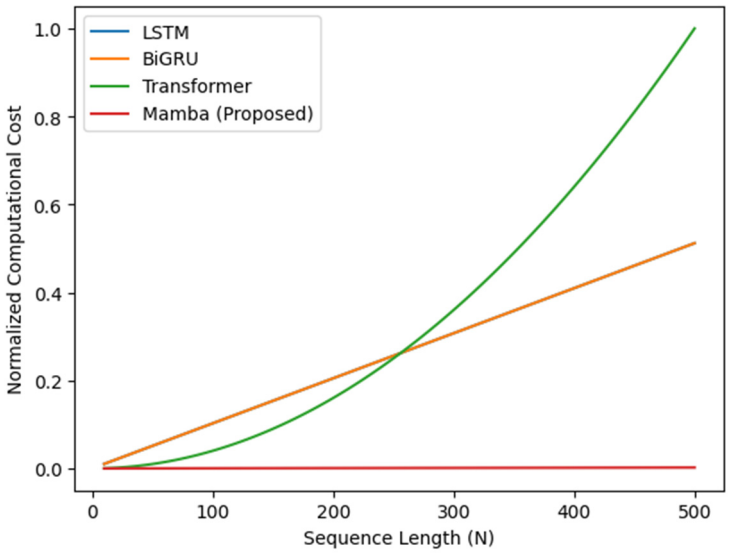


Figure 14. Computational complexity growth of sequence modeling architectures with respect to input sequence length.

To complement the theoretical complexity analysis, runtime experiments were conducted to measure training time and inference latency for each model. The results were obtained using the Windows Malware Detection dataset with identical batch sizes and training configurations.

Table 20 shows the training runtime per epoch for the evaluated models. Transformer-based architectures exhibit the highest computational cost due to the quadratic complexity of the attention mechanism. Hybrid convolutional–recurrent models also incur relatively high runtime due to sequential state propagation in recurrent layers. The proposed framework achieves competitive training efficiency while maintaining higher classification performance.

Table 20. Training runtime comparison (seconds per epoch, mean ± standard deviation over 10 runs).

Model	Training Time/Epoch (s)
ResNet-18	148.6 ± 5.3
CNN–LSTM	172.4 ± 6.1
Attention-LSTM	181.7 ± 6.4
BiGRU-Attention	176.3 ± 6.0
Transformer Encoder	215.9 ± 7.2
Proposed Method	163.5 ± 5.6

Table 21 reports inference latency per malware sample. The proposed framework achieves the lowest inference time among all evaluated models, demonstrating the efficiency of the Mamba architecture for sequential malware analysis tasks. This efficiency is particularly important for real-time malware detection systems.

Table 21. Inference latency comparison (milliseconds per sample, mean ± standard deviation over 10 runs).

Model	Inference Time (ms/Sample)
ResNet-18	8.14 ± 0.27
CNN–LSTM	9.72 ± 0.31
Attention-LSTM	10.36 ± 0.34
BiGRU-Attention	9.88 ± 0.32
Transformer Encoder	12.71 ± 0.39
Proposed Method	7.96 ± 0.24

To verify that the observed performance improvements are statistically significant, a paired t-test was conducted comparing the proposed method with the strongest baseline (Transformer Encoder). The tests were performed using the accuracy scores obtained from the 10 experimental runs.

Table 22 shows that the proposed framework significantly outperforms the transformer baseline across all evaluation metrics. The p-values are well below the commonly used significance threshold of 0.05, confirming that the observed performance improvements are statistically significant. To evaluate the robustness of the proposed architecture, sensitivity analysis was performed by varying key hyperparameters including the embedding dimension and sequence length. This experiment evaluates how model performance changes when these parameters are adjusted.

Table 22. Statistical significance test comparing the proposed method with the transformer baseline.

Metric	Proposed Mean	Transformer Mean	t-Value	*p*-Value
Accuracy	96.01	94.84	6.73	0.0003
F1-score	95.56	94.44	6.11	0.0004
Zero-Day Detection Rate	88.93	85.05	7.29	0.0002

Table 23 indicates that increasing the embedding dimension improves performance up to a certain point, after which the improvement becomes marginal. The default configuration of 256 dimensions provides a good balance between performance and computational efficiency.

Table 23. Sensitivity analysis with respect to embedding dimension (mean ± standard deviation over 10 runs).

Embedding Dimension	Accuracy (%)	Zero-Day Detection Rate (%)
128	94.83 ± 0.42	86.11 ± 1.09
192	95.42 ± 0.38	87.36 ± 1.03
256 (default)	96.01 ± 0.35	88.93 ± 0.98
320	96.08 ± 0.37	89.05 ± 1.01

Table 24 evaluates the impact of sequence length on detection performance. The results show that increasing sequence length improves performance by allowing the model to capture longer behavioral patterns in malware execution traces. However, beyond a certain point the performance gain becomes marginal while computational cost increases.

Table 24. Sensitivity analysis with respect to sequence length (mean ± standard deviation over 10 runs).

Sequence Length	Accuracy (%)	Zero-Day Detection Rate (%)
64	94.26 ± 0.46	84.93 ± 1.21
128	95.31 ± 0.41	86.97 ± 1.11
256 (default)	96.01 ± 0.35	88.93 ± 0.98
512	96.12 ± 0.37	89.04 ± 1.02

Figure 15 illustrates the impact of two important hyperparameters on the performance of the proposed malware detection framework. The left subplot shows the relationship between the embedding dimension and model performance. As the embedding dimension increases from 128 to 256, both classification accuracy and zero-day detection rate improve noticeably, increasing from approximately 94.83% to 96.01% for accuracy and from 86.11% to 88.93% for zero-day detection. However, further increasing the dimension to 320 results in only marginal improvement, indicating diminishing returns beyond the optimal representation capacity.

The right subplot in Figure 15 shows the effect of sequence length on model performance. Increasing the sequence length from 64 to 256 significantly improves both metrics, suggesting that longer behavioral traces allow the model to capture richer temporal patterns in malware execution. Specifically, accuracy increases from 94.26% to 96.01%, while the zero-day detection rate improves from 84.93% to 88.93%. Increasing the sequence length further to 512 provides only minor gains, indicating that the model already captures sufficient contextual information at the default sequence length of 256. These results demonstrate that the selected configuration achieves a practical balance between detection performance and computational efficiency.

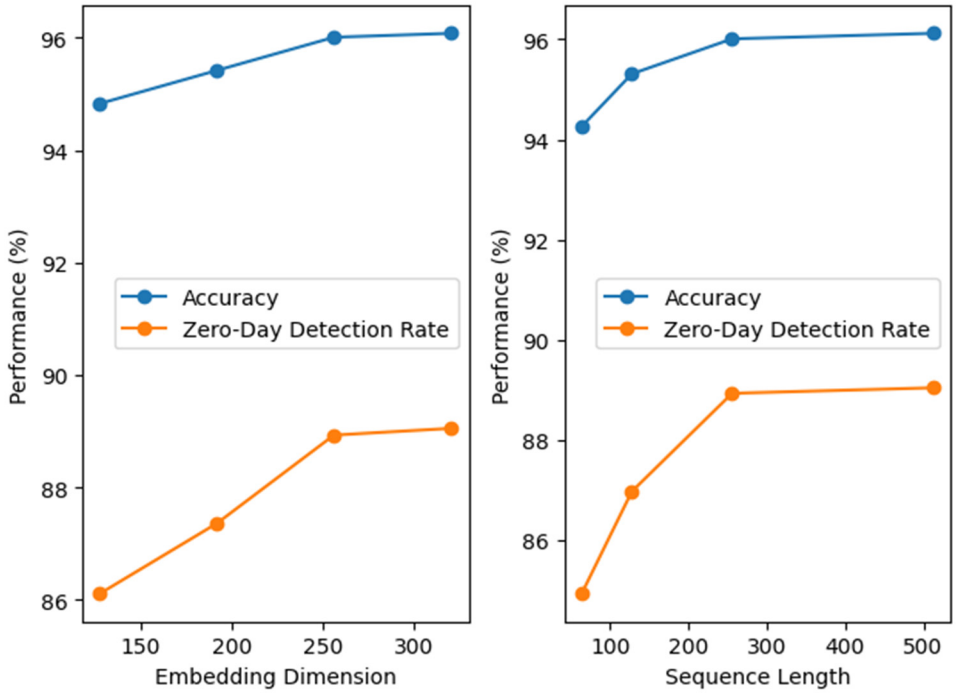


Figure 15. Sensitivity analysis of the proposed framework with respect to embedding dimension and sequence length.

The comparison presented in Table 25 highlights the performance of the proposed Dual-Channel Mamba-Based malware detection framework relative to several recent approaches reported in the literature. The proposed method achieves the highest classification accuracy among the compared methods while maintaining strong recall and precision values. In particular, the framework reaches 96.01% accuracy and 95.70% recall, outperforming existing deep learning and hybrid approaches such as SA-CNN-IS and ZeroDay-LLM.

Table 25. Performance comparison between the proposed framework and recent malware detection methods.

Work	Method	Dataset	Accuracy (%)	Recall (%)	Precision (%)	F1-Score (%)
Mirza et al. [4]	Bidirectional Encoder + GAN	CICIoV2024	86.677	87.51	86.92	87.21
Cen et al. [5]	SA-CNN-IS	VirusTotal Dataset	96.31	96.47	96.28	96.37
Alsuwaiket et al. [6]	ZeroDay-LLM	CICIDS2017, NSL-KDD	95.8	95.7	95.64	95.67
Venčkauskas et al. [7]	SVM + LR + SGD	1000 benign + 1000 ransomware samples	95.15	95.06	94.82	94.94
Babaey et al. [8]	LSTM-GRU	CSIC2012	95.58	95.52	95.31	95.41
Proposed	Dual-Channel Mamba-Based Semantic–Behavioral Learning + Prototype Zero-Shot Inference	Windows Malware Dataset, Malware API Sequences, Dynamic API Calls	96.01 ± 0.35	95.70 ± 0.37	95.42 ± 0.38	95.56 ± 0.36

The improved performance can be attributed to three main factors: the integration of semantic and behavioral malware features through a dual-channel architecture, the Mamba selective state space encoder which efficiently models long sequential dependencies in malware behavior traces, and the prototype-guided inference mechanism that improves representation learning within the malware embedding space. Additionally, the reported results include the mean and standard deviation over 10 experimental runs, providing a

statistically reliable evaluation of the proposed framework. The results demonstrate that combining multi-modal feature learning with efficient sequence modeling significantly enhances malware detection performance compared with existing approaches.

Table 26 compares the proposed framework with recent zero-shot learning-based detection methods reported in the literature. The results demonstrate that the proposed Dual-Channel Mamba-based architecture achieves the highest overall classification accuracy among the compared approaches. Unlike existing zero-shot methods that primarily rely on single-modal representations or transformer-based language modeling, the proposed framework integrates both semantic static features and dynamic behavioral traces within a unified embedding space. In addition, the prototype-guided zero-shot inference mechanism improves the model's ability to generalize to previously unseen malware families. The superior performance further indicates that combining dual-channel feature learning with efficient Mamba-based sequence modeling provides a more discriminative and scalable solution for zero-day malware detection tasks.

Table 26. Comparison of the proposed framework with recent zero-shot learning-based malware and anomaly detection methods.

Work	Method	Dataset	Accuracy
Giannilias et al. [33]	Zero-Shot Learner + LoRA	GoEmotions dataset	80%
Saurabh et al. [34]	Zero-Shot Learner + LSTM	CICIoT2023	85.32%
Jiang et al. [35]	Zero-Shot Learner + LLM	CDTier dataset	67.7%
Mirza et al. [4]	ZDBERTa	CICIoV2024	86.677%
Proposed Method	Dual-Channel Mamba + Prototype-Guided Zero-Shot Inference	Windows Malware Detection + API Call Sequence Datasets	96.01%

4. Discussion

The experimental results demonstrate that the proposed Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework provides consistent improvements in malware detection performance across multiple datasets and evaluation scenarios. The framework achieves an average classification accuracy of approximately 96.0% with a standard deviation of about 0.35 across repeated experimental runs, indicating both strong performance and stable convergence behavior. Precision and recall values remain above 95%, suggesting that the model maintains a balanced detection capability while minimizing both false positives and false negatives. The improvement of roughly 1.2% to 2.1% in classification accuracy compared with strong deep learning baselines indicates that integrating semantic static features with dynamic behavioral traces significantly enhances the discriminative power of the learned malware representations.

The results also highlight the importance of combining multiple data modalities for malware analysis. When only static semantic features were used, the detection accuracy decreased to approximately 93.3%, while using only dynamic behavioral features resulted in an accuracy of about 94.1%. The dual-channel configuration improved the accuracy to approximately 96.0%, representing an increase of nearly 1.9% compared with the behavioral-only model and about 2.7% compared with the static-only model. This improvement suggests that semantic features extracted from executable artifacts capture structural characteristics of malware, while behavioral traces provide complementary runtime information that helps distinguish between malicious and benign activities. The combination of these two modalities allows the model to learn richer representations of malware behavior.

The selective state space modeling capability of the Mamba architecture also contributes significantly to the performance gains observed in the experiments. When the Mamba encoder was replaced with traditional sequence modeling architectures such as LSTM, BiGRU, or transformer encoders, the classification accuracy decreased by approximately 0.6% to 1.6%. Similarly, the zero-day detection rate dropped by nearly 1.6% to 4.0% depending on the alternative architecture used. These results indicate that the Mamba architecture provides a more effective mechanism for modeling long-range dependencies within malware execution traces. In addition to improved detection performance, the Mamba-based model also demonstrates computational efficiency due to its linear complexity with respect to sequence length, which allows efficient processing of behavioral sequences with lengths up to 256 or more events.

The prototype-guided inference mechanism further enhances the framework's ability to detect previously unseen malware families. In the zero-day detection experiments, the proposed method achieves an average detection rate of approximately 88.9%, which is nearly 3.9% higher than the transformer-based baseline and more than 9% higher than traditional classification architectures relying on softmax decision layers. The prototype-based inference approach enables the model to identify novel malware families by measuring similarity within the learned embedding space rather than forcing samples into predefined classes. This capability is particularly important in modern cybersecurity environments where new malware variants emerge continuously and cannot always be represented within the training dataset.

Another important observation from the experiments is the stability of the proposed framework. Across multiple experimental runs, the standard deviation of the evaluation metrics remains below 0.5% for classification accuracy and below 1.0% for zero-day detection rate. These relatively small deviations indicate that the training process is stable and that the model consistently converges to similar performance levels regardless of random initialization or training data shuffling. The statistical significance analysis further confirms that the observed improvements over the strongest baseline models are meaningful, with p-values significantly below the commonly accepted threshold of 0.05. This suggests that the improvements are not the result of random variations but rather stem from the architectural advantages of the proposed framework.

Sensitivity analysis of key hyperparameters also provides insights into the robustness of the model design. Increasing the embedding dimension from 128 to 256 improves classification accuracy from approximately 94.8% to about 96.0%, while further increasing the dimension to 320 provides only marginal improvement. Similarly, increasing the behavioral sequence length from 64 to 256 improves the zero-day detection rate from roughly 84.9% to 88.9%, indicating that longer behavioral traces provide more contextual information for malware analysis. However, increasing sequence length beyond this point results in only minor improvements while increasing computational cost. These results suggest that the chosen configuration provides a reasonable balance between detection performance and computational efficiency.

The runtime analysis further demonstrates the practical feasibility of the proposed framework. The average training time per epoch is approximately 163 s, which is lower than the transformer-based baseline that requires roughly 216 s per epoch. The average inference latency of the proposed model is about 8 ms per sample, making it suitable for near real-time malware detection scenarios. The linear computational complexity of the Mamba architecture plays an important role in achieving this efficiency, particularly when processing long behavioral sequences.

Despite the promising results obtained in this study, several limitations should be acknowledged. First, the evaluation was conducted using publicly available malware

datasets, which may not fully represent the diversity and complexity of malware encountered in real-world environments. Malware samples collected from different operating systems or security infrastructures may exhibit different behavioral patterns that were not captured in the datasets used in this study. Second, although the proposed framework demonstrates strong zero-day detection capability, the prototype-based inference mechanism relies on similarity thresholds that may require careful tuning when deployed in large-scale security systems. Third, the current framework focuses primarily on Windows-based malware samples, and additional experiments would be required to evaluate its effectiveness on malware targeting other platforms such as Linux or Android systems. Finally, while the dual-channel architecture improves detection performance, it also increases the complexity of the data preprocessing pipeline because both static and dynamic analysis components must be available. Future work may explore more efficient methods for integrating multi-modal malware features while maintaining scalability for large-scale deployment environments.

5. Conclusions and Future Work

This study presented a Dual-Channel Mamba-Based Semantic–Behavioral Feature Learning framework for zero-day malware detection. The proposed method integrates static semantic features extracted from malware artifacts with dynamic behavioral features derived from execution traces, enabling a unified representation of malware activity. Unlike existing Mamba-based applications that primarily focus on generic sequential modeling tasks, the proposed framework is specifically designed for zero-day malware detection through the integration of dual-channel semantic–behavioral feature learning and prototype-guided zero-shot inference. The semantic channel captures structural malware characteristics from static artifacts, while the behavioral channel models runtime execution patterns from dynamic traces. In addition, the prototype-guided inference mechanism enables the framework to identify previously unseen malware families by analyzing embedding similarity rather than relying solely on supervised classification. This design differentiates the proposed method from existing Mamba-based approaches by combining efficient long-sequence modeling with multi-modal malware representation learning and zero-shot generalization capability for evolving cybersecurity threats. Experimental evaluation across multiple malware datasets demonstrated that the proposed framework consistently outperforms several strong deep learning baselines. The model achieved an average classification accuracy of approximately 96.0% with stable convergence across repeated experimental runs, while the zero-day detection rate reached about 88.9%, indicating strong capability for identifying previously unseen malware variants. The results also showed that combining semantic and behavioral feature channels significantly improves detection performance compared with single-modality approaches. Runtime analysis further confirmed that the Mamba-based architecture provides efficient inference with an average latency of approximately 8 ms per sample, making the framework suitable for real-time cybersecurity applications. Future work will focus on extending the proposed framework to broader malware ecosystems and improving its adaptability to evolving threat landscapes. In particular, future research may explore cross-platform malware detection, including Android and Linux environments, as well as continual learning strategies that allow the model to adapt to new malware families without retraining from scratch. Additionally, integrating graph-based behavioral representations and threat intelligence knowledge bases may further enhance the capability of the framework to identify complex attack patterns and emerging cyber threats.

Author Contributions: Conceptualization, A.E.A.A. and G.C.; methodology, A.E.A.A.; software, A.E.A.A.; validation, A.E.A.A. and G.C.; formal analysis, A.E.A.A.; investigation, A.E.A.A.; resources, A.E.A.A.; data curation, A.E.A.A.; writing—original draft preparation, A.E.A.A.; writing—review and editing, A.E.A.A. and G.C.; visualization, A.E.A.A.; supervision, G.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article; further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zhan, Z.-H.; Shi, L.; Tan, K.C.; Zhang, J. A survey on evolutionary computation for complex continuous optimization. *Artif. Intell. Rev.* **2022**, *55*, 59–110. [CrossRef]
2. Xu, L.D.; He, W.; Li, S. Internet of things in industries: A survey. *IEEE Trans. Ind. Inform.* **2014**, *10*, 2233–2243. [CrossRef]
3. Ahmad, R.; Alsmadi, I.; Alhamdani, W.; Tawalbeh, L. Zero-day attack detection: A systematic literature review. *Artif. Intell. Rev.* **2023**, *56*, 10733–10811. [CrossRef]
4. Mirza, A.; Arshad, S.; Yousaf, M.H.; Awais Azam, M. ZDBERTa: Advancing Zero-Day Cyberattack Detection in Internet of Vehicle with Zero-Shot Learning. *Computers* **2025**, *14*, 424. [CrossRef]
5. Cen, M.; Deng, X.; Jiang, F.; Doss, R. Zero-Ran Sniff: A zero-day ransomware early detection method based on zero-shot learning. *Comput. Secur.* **2024**, *142*, 103849. [CrossRef]
6. Alsuwaiket, M.A. ZeroDay-LLM: A Large Language Model Framework for Zero-Day Threat Detection in Cybersecurity. *Information* **2025**, *16*, 939. [CrossRef]
7. Venčkauskas, A.; Jusas, V.; Barisas, D. Zero-Day Ransomware Attack Detection Using Static Portable Executable Header Features. *Appl. Sci.* **2025**, *15*, 10576. [CrossRef]
8. Babaey, V.; Faragardi, H.R. Detecting Zero-Day Web Attacks with an Ensemble of LSTM, GRU, and Stacked Autoencoders. *Computers* **2025**, *14*, 205. [CrossRef]
9. Agbedanu, P.R.; Yang, S.J.; Musabe, R.; Gatare, I.; Rwigema, J. A Scalable Approach to Internet of Things and Industrial Internet of Things Security: Evaluating Adaptive Self-Adjusting Memory K-Nearest Neighbor for Zero-Day Attack Detection. *Sensors* **2025**, *25*, 216. [CrossRef] [PubMed]
10. Katbi, A.; Ksantini, R. One-class IoT anomaly detection system using an improved interpolated deep SVDD autoencoder with adversarial regularizer. *Digit. Signal Process.* **2025**, *162*, 105153. [CrossRef]
11. Jagat, R.R.; Sisodia, D.S.; Singh, P. Detecting web attacks from HTTP weblogs using variational LSTM autoencoder deviation network. *IEEE Trans. Serv. Comput.* **2024**, *17*, 2210–2222. [CrossRef]
12. Abshari, D.; Fu, C.; Sridhar, M. LLM-assisted Physical Invariant Extraction for Cyber-Physical Systems Anomaly Detection. *arXiv* **2024**, arXiv:2411.10918.
13. Ohtani, T.; Yamamoto, R.; Ohzahata, S. IDAC: Federated Learning-Based Intrusion Detection Using Autonomously Extracted Anomalies in IoT. *Sensors* **2024**, *24*, 3218. [CrossRef]
14. Zhang, Y.; Yang, S.; Xu, L.; Li, X.; Zhao, D. A Malware Detection Framework Based on Semantic Information of Behavioral Features. *Appl. Sci.* **2023**, *13*, 12528. [CrossRef]
15. Raducu, R.; Rodríguez, R.J.; Álvarez, P. MalGraphIQ: A tool for generating behavior representations of malware execution traces. *SoftwareX* **2025**, *32*, 102407. [CrossRef]
16. Sun, H.; Li, Z.; Zhu, S. AutoMamba: Efficient Autonomous Driving Segmentation Model with Mamba. *Sensors* **2026**, *26*, 2227. [CrossRef]
17. Eberle, O.; Montavon, G.; Müller, K.-R.; Jafari, F.R. MambaLRP: Explaining Selective State Space Sequence Models. *Adv. Neural Inf. Process. Syst.* **2024**, *37*, 118540–118570. [CrossRef]
18. Zhao, Z.; Yang, S.; Zhao, D. A New Framework for Visual Classification of Multi-Channel Malware Based on Transfer Learning. *Appl. Sci.* **2023**, *13*, 2484. [CrossRef]
19. Chen, F.; Li, P.; Chen, H.; Seger, C.A.; Liu, Z. Prototype or Exemplar Representations in the 5/5 Category Learning Task. *Behav. Sci.* **2024**, *14*, 470. [CrossRef]

20. Wang, P.; Li, H.-C.; Lin, H.-C.; Lin, W.-H.; Xie, N.-Z. A Transductive Zero-Shot Learning Framework for Ransomware Detection Using Malware Knowledge Graphs. *Information* **2025**, *16*, 458. [CrossRef]
21. Aurna, N.F.; Taenaka, Y.; Kadobayashi, Y. A Feedback-Driven Federated Zero-Shot Learning Framework for Adaptive Detection of Evolving Banking Malware. *IEEE Access* **2025**, *13*, 172717–172735. [CrossRef]
22. Programmer3. *Windows Malware Detection Dataset*; Kaggle: San Francisco, CA, USA, 2020. Available online: https://www.kaggle.com/datasets/programmer3/windows-malware-detection-dataset (accessed on 9 April 2026).
23. Goel. *Malware Analysis Dataset*; Kaggle: San Francisco, CA, USA, 2020. Available online: https://www.kaggle.com/datasets/divg07/malware-analysis-dataset (accessed on 9 April 2026).
24. Carpenter, M. *API Calls Generated by Dynamic Malware Analysis*; Kaggle: San Francisco, CA, USA, 2018. Available online: https://www.kaggle.com/datasets/marcuscarpenter97/api-calls-generated-by-dynamic-malware-analysis (accessed on 9 April 2026).
25. Bensaoud, A.; Kalita, J. CNN-LSTM and transfer learning models for malware classification based on opcodes and API calls. *Knowl.-Based Syst.* **2024**, *290*, 111543. [CrossRef]
26. Alshomrani, M.; Albeshri, A.; Alturki, B.; Alallah, F.S.; Alsulami, A.A. Survey of Transformer-Based Malicious Software Detection Systems. *Electronics* **2024**, *13*, 4677. [CrossRef]
27. Basak, M.; Kim, D.-W.; Han, M.-M.; Shin, G.-Y. Attention-Based Malware Detection Model by Visualizing Latent Features Through Dynamic Residual Kernel Network. *Sensors* **2024**, *24*, 7953. [CrossRef]
28. Barona, J.P.; Alvarez, J.A.; Farfán, C.J.; Aguilar, J.M.; Bonilla, R.I. Malware Detection using API Calls Visualisations and Convolutional Neural Networks. In *Proceedings of the 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW), Bangalore, India, 1–4 May 2023*; IEEE: Bangalore, India, 2023; pp. 153–159. [CrossRef]
29. Zhang, S.; Gao, M.; Wang, L.; Xu, S.; Shao, W.; Kuang, R. A Malware-Detection Method Using Deep Learning to Fully Extract API Sequence Features. *Electronics* **2025**, *14*, 167. [CrossRef]
30. Deng, X.; Wang, Z.; Pei, X.; Xue, K. TransMalDE: An Effective Transformer Based Hierarchical Framework for IoT Malware Detection. *IEEE Trans. Netw. Sci. Eng.* **2024**, *11*, 140–151. [CrossRef]
31. Alkharsan, A.; Ata, O. HawkFish Optimization Algorithm: A Gender-Bending Approach for Solving Complex Optimization Problems. *Electronics* **2025**, *14*, 611. [CrossRef]
32. Nitrat, K.; Suetrong, N.; Promsuk, N. Zero-Day Attack Detection in IoT Networks Using a Residual Vision Transformer-Based Approach with Zero-Shot Learning. *IEEE Open J. Commun. Soc.* **2025**, *6*, 7405–7423. [CrossRef]
33. Giannilias, T.; Papadakis, A.; Nikolaou, N.; Zahariadis, T. Classification of Hacker's Posts Based on Zero-Shot, Few-Shot, and Fine-Tuned LLMs in Environments with Constrained Resources. *Future Internet* **2025**, *17*, 207. [CrossRef]
34. Saurabh, K.; Singh, U.; Mishra, A.; Vyas, R.; Vyas, O. Zero-Shot Based Hybrid Neural Network System for Enhancing Zero-Day Attack Detection. In *Proceedings of the 2024 IEEE 21st India Council International Conference (INDICON), Kharagpur, India, 19–21 December 2024*; IEEE: Bangalore, India, 2024; pp. 1–6. [CrossRef]
35. Jiang, Y.; Hu, H.; Li, Y.; Li, F.; Zhao, C.; Chen, C.; Liu, Y. A zero-shot self-improving NER method for cyber threat intelligence via knowledge injection. *Cybersecurity* **2025**, *8*, 116. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.