Hybrid CNN-LSTM-Based Malware Detection System Fusing API Sequences with External Signals

Hamid Talib Zaidan[1], Jumana Waleed[1] and Reem Aljuaidi[2]
[1]*Department of Computer Science, College of Science, University of Diyala, 32001 Baqubah, Iraq*
[2]*Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, 16278 Al-Kharj, Saudi Arabia*
hamidtalib@uodiyala.edu.iq, jumanawaleed@uodiyala.edu.iq and r.aljuaidi@psau.edu.sa

Keywords: Hybrid System, Malware Detection, CNN, LSTM, API Call Sequences, External Features.

Abstract: The fast growth of malware and the continuous development of new techniques to hide malicious behavior have made traditional detection methods, such as signature-based and heuristic approaches, less effective. Today's cybersecurity needs systems that are accurate, scalable, and capable of dealing with new and unknown types of attacks. Deep learning models such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) have shown strong ability in analyzing API call sequences and detecting malware behavior. However, when these models are trained on small or unbalanced datasets, they often suffer from overfitting and weak generalization. In addition, many previous studies did not include external features from services like Hybrid Analysis, or they used costly transformations such as converting binaries into images, which reduced the practical value of their systems. To address these problems, we propose a hybrid CNN-LSTM framework that combines API sequence analysis with external features. API calls are integer-encoded to preserve their order, while extra attributes such as malicious ratios, detection counts, and threat scores were collected through the official API of Hybrid Analysis system. A variety of optimization methods were used such as dropout, token dropout, batch normalization, L2 regularization, adaptive learning-rate scheduling with AdamW and stratified dataset splitting. These techniques increased the generalization, training stability and decreased overfitting. It was tested on 3 datasets, Dataset 1 (MalBehavD-V1 + Oliveira 3,500 samples), Dataset 2 (the same dataset with external features with Hybrid Analysis API), and Dataset 3 (a large scale dataset with 2025 85,594 balanced samples). The proposed model achieved 95.43% accuracy on Dataset 1, 97.49% on Dataset 2, and 99.52% on Dataset 3, consistently surpassing standalone CNN and LSTM baselines in accuracy and AUC. These results demonstrate that combining behavioral API sequences with lightweight external features yields a precise, scalable malware detection system suitable for practical cybersecurity deployments.

1 INTRODUCTION

The rapid development of malware and the ever-evolving methods of hiding and code-changing has rendered the traditional signature based and heuristic methods of detection ineffective [1]. The contemporary cybersecurity demands not just the accuracy of systems but also scalability and resistance to new and unpredictable threats [2]. Deep learning in this regard has revolutionized the field of malware detection by making it possible to automatically extract features and model sophisticated nonlinear relationships previously inaccessible to handcrafted features [3]. The Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) are deep learning models that have been observed to demonstrate impressive performance in sequential and high-dimensional data modeling in various application tasks and have demonstrated robust capabilities at API call sequence modeling and malicious behavioral patterns recognition [4].

Despite the promising results, CNNs and LSTMs may show poor performance with small and/or unbalanced datasets, which causes overfitting and poor generalization [5]. Additionally, most of the existing literature uses an API sequence alone or expensive data manipulations including executable file to image converting, which contributes to higher cost of computation and reduces applicability to the real world [6]. Notably, most of the earlier research

failed to leverage the features that are external to the platforms such as Hybrid Analysis, as these may offer useful contextual information to malware detection.

Even though in the recent past, there have been some advancements, some significant gaps in the research on malware detection still exist [3]. Most of the systems use API sequences alone without incorporating extraneous contextual information hence constraining their capability to find more meaningful behaviors [1]. Some of them rely on the expensive preprocessing e.g. executable to image, transformer models which are expensive to run increasing the cost of computation and decreasing practical use in the real world [7], [6]. Scalability has not been assessed in many cases and most of the methods have been tested on small data sets and thus they are not reliable in large scale working conditions [8], [9]. In addition, deep learning models are typically poorly interpretable, which is a black-box system that hinders the trust and utilisation in the practice of cybersecurity [6]. There is also increased likelihood of overfitting as well as reduced generalization of a large variety of malware families due to the use of limited or imbalanced datasets. These gaps point out the necessity of having a viable framework, which is lightweight, scalable, readable, and able to integrate both behavioral and external characteristics [2].

To overcome these issues, we will assign a hybrid CNN-LSTM-based model, which allows us to learn the API sequences (reflecting the behavioral implementation patterns of programs) and external features, which are acquired with the help of the official API of Hybrid Analysis (offering contextual metadata like counts of detection and threat scores gathered in big malware libraries). The API calls are integers to maintain the sequence of execution whereas the external features like malicious ratios, the number of times it was detected and threat scores are used to supplement the model with complementary information. A number of optimization techniques, which include dropout, spatial dropout, token dropout, batch normalization, L2 regularity, adaptive learning-rate scheduling using AdamW, and stratified validation are used to enhance generalization and minimize overfitting. The framework is evaluated on three datasets:

- Dataset 1: MalBehavD-V1 + Oliveira (3,500 samples).
- Dataset 2: MalBehavD-V1 + Oliveira (3,500 samples) enriched with external features from Hybrid Analysis system API.
- Dataset 3: Smote Api Call Sequence Dataset (2025, 85,594 balanced samples).

The main contributions of this work are:

1) Hybrid CNN-LSTM framework: Combines behavioral API sequence modeling with external features from Hybrid Analysis API.
2) Dataset diversity and scalability: Evaluation on three datasets demonstrates the system's robustness across small, enriched, and large-scale scenarios.
3) Performance improvement with external features: Accuracy improved from 95.43% (API-only) to 97.49% with Hybrid Analysis.
4) Optimization for generalization: Applying dropout, token dropout, batch normalization, L2 regularization, adaptive scheduling, and stratified validation reduced overfitting and improved robustness.
5) Efficiency with reduced overhead: The system uses integer encoding and lightweight feature processing instead of heavy image or transformer-based methods.

The remainder of this paper is structured as follows. Section 2 reviews related work on malware detection and hybrid deep learning approaches. Section 3 presents the proposed methodology, including dataset preparation, preprocessing, feature selection, and model design. Section 4 explains the experiments and measures of evaluation. Section 5 provides the discussion of the results, limits and comparison with previous works. Lastly, Section 6 draws the conclusion of the paper and summarizes future research directions.

2 RELATED WORKS

Deep learning systems have become a critical part of malware detection as more sophisticated methods have been used that alter code behavior and structure and are hard to be detected by older signature-based and heuristic systems. Over the past few years, various hybrid models employing convolutional and recurrent models have been suggested to handle API call sequences, opcode characteristics, or even visual images of malware. Although these methods yield good performances, they also have limitations with regard to the size of databases, preprocessing cost, computations, and ability to generalize. Table 1 presents a summary of some of the representative studies.

Karat et al. [8] (2024) presented CNN-LSTM hybrid to call API analysis with the help of the data gathered with Rohitab API Monitor and Sysmon logs. They mined 308 distinct API calls and trained them

using Conv1D and LSTM (512 units and dropout) with 96% validation accuracy. Nonetheless, the dataset was smaller (approximately 2,500 samples), and the model was very resource-intensive (heavy use of GPUs) and reliant on particular monitoring tools to allow reproducibility.

The EarlyMalDetect by Maniriho et al. [1] (2024) is oriented on the initial 20-50 API calls to accomplish the early detection. With GPT-2 prediction and DistilBERT + BiGRU with attention classification, they reached an accuracy of about 96 percent. Although useful in the early stages of detection, the use of large language models complicated and raised the cost of the process and failed to track longer traces.

CAFTrans is a transformer-based system proposed by Qian and Cong [7] (2024) that involves the use of call frequencies and channel characteristics. It was tested on Mal-API-2019 and had moderate results (F1 = 0.65, AUC = 0.79). A limitation to frequency-based features was seen by weak temporal modeling and obsolete data lowering the generalization.

The study by Hussain et al. [2] (2024) took ransomware as the object of GN-BiLSTM on CIC-MalMem-2022 and 10,000 ransomware samples. Even though detection accuracy was very high, emphasis on ransomware made it less applicable to other malware families and the use of memory traces made it more susceptible to evasive samples.

Alshomrani et al. [6] (2025) introduced a hybrid model, which is explainable, using ConvNeXt-Tiny and Swin Transformer to classify visual malware. It was tested on Malimg, MaleVis, and VirusMNIST with an accuracy of 94 98% with Grad-CAM explanations. Nonetheless, it was less useful to API-sequence analysis and real-world implementation due to costly image conversion and processing intensive examinations using transformers.

Song et al. [3] (2025) conducted the review of 363 studies on deep learning-based malware detection. They noted such issues as the absence of standardized datasets, poor cross-domain generalization, and little explainability. But, being in the form of a review, the work did not give any concrete solutions.

Li et al. [9] (2025) developed a deep architecture that consists of three components, namely, API Phrase (1D-CNN), Semantic Chain, and Bi-LSTM. The model used on Cuckoo Sandbox on 43,000 samples yielded an accuracy of 97.31 (F1 = 0.97). On newer samples (~0.91), performance decreased, which is susceptible to concept drift. Besides, the intricate nature of semantic features engineering decreased scalability.

Table 1: The evaluation of some applicable deep learning-based malware detection studies.

Authors, Year	Datasets	Methodology	Performance	Limitations / Key Insight
Karat et al. [8], 2024	Rohitab API Monitor, Sysmon (~2.5k)	CNN-LSTM (Conv1D + LSTM)	Acc = 96%	Small dataset, high GPU dependency, limited reproducibility across tools
Maniriho et al. [1], 2024	Custom traces (20–50 API calls)	GPT-2 + DistilBERT + BiGRU	Acc ≈ 95–96%	High complexity (LLMs); limited to short traces
Qian & Cong [7], 2024	Mal-API-2019	CAFTrans (frequency + transformer)	F1 = 0.65, AUC = 0.79	Outdated dataset; weak temporal modeling; imbalance issues
Hussain et al. [2], 2024	CIC-MalMem-2022 + ransomware (10k)	GN-BiLSTM	Near-perfect detection	Domain-specific (ransomware only); reduced robustness to evasive malware
Alshomrani et al. [6], 2025	Malimg, MaleVis, VirusMNIST	ConvNeXt-Tiny + Swin Transformer	Acc = 94–98%	Heavy image preprocessing; transformer GPU-intensive; unsuitable for API-sequence analysis
Song et al. [3], 2025	363 studies (72 analyzed)	Review	N/A	Identified issues: limited datasets, poor generalization, low explainability; no proposed model
Li et al. [9], 2025	Cuckoo Sandbox (~43k)	1D-CNN (API Phrase) + Bi-LSTM (Semantic Chain)	Acc = 97.31%, F1 = 0.97	Accuracy dropped on newer data (~0.91); complex semantic feature engineering
Bamber et al. [4], 2025	NSL-KDD	CNN-LSTM + RFE + Decision Tree feature ranking	Acc = 95%, F1 = 0.94	Focused on network intrusion; demonstrates CNN-LSTM scalability and robustness
Thakur et al. [5], 2024	Public malware binaries (grayscale image conversion)	CNN-LSTM + PCA-based feature reduction	N/A	Image-based preprocessing increases complexity; limited adaptability to non-visual datasets

The intelligent intrusion detection system (IDS) proposed by Bamber et al. [4] (2025) is based on CNN-LSTM and it uses the NSL-KDD dataset. They used Recursive Feature Elimination (RFE) and Decision Tree-based feature selection, and well-performed on the overall. The experiment has shown that CNN-LSTM hybrid is a powerful tool to differentiate between malicious and benign network traffic with a high level of efficiency and robustness, which proves that it can be useful in the context of cybersecurity.

Thakur et al. [5] (2024) suggested a hybrid deep learning model of CNN and LSTM to detect malware. Malware binaries were transformed into gray scale image and fed through CNN-LSTM networks with PCA based feature reduction. In spite of high precision, recall, and F1-score of the study. In addition, the image-based preprocessing added more complexity in terms of computation and reduced flexibility to behavioral or sequencing-based data.

Past studies affirm the fact that CNN LSTM, transformer based, and multimodel system can be used to generate a high level of detection accuracy. Nevertheless, there are still typical drawbacks: dependence on small or old datasets, expensive preprocessing (e.g. image conversion, semantic parsing), high computational demands (transformers, LLMs), or a narrow scope of malware types. Noteworthy, external features retrieved using APIs i.e. Hybrid Analysis are hardly ever directly combined in a study, yet they can provide contextual information about behavioral data.

In comparison, the suggested CNN-LSTM framework is capable of addressing these problems through the use of three datasets (a small academic dataset, the same dataset with API-based external features and a large-scale contemporary dataset), and optimization tools, i.e. dropout, token dropout, spatial dropout, batch normalization, L2 regularization, adaptive scheduling, stratified validation. Such a combination guarantees high accuracy, efficiency, scalability, and generalization in comparison with the previous works.

3 PROPOSED HYBRID CYBERSECURITY SYSTEM

The general structure of the hybrid cybersecurity system proposed s illustrated in Figure 1. It starts with the choice of one of the three datasets: Dataset 1 (Datasets 1 includes the original MalBehavD-V1 + Oliveira, 3,500 samples) [10], Dataset 2 (Datasets 2 contains the same dataset 1, but with external features added through Hybrid Analysis APIs: 3,500 samples), and Dataset 3 (contains a large-scale API call sequence dataset of 2025, with 85,594 balanced samples). These datasets have been fully described in Subsection 3.1.

After the selection of the dataset, the data is preprocessed to complete the data ready to be modeled. The general preprocessing actions include integer encoding API sequences, padding of the lengths of the sequences and missing values imputation and normalization of the external features where applicable. In Subsection 3.2, detailed operations are given.

The learning phase uses a CNN-LSTM model. In the case of datasets with sequences of API calls only (Datasets 1 and 3), the framework uses a CNN-LSTM model to learn both local patterns of API calls as well as long-term sequential relationships. In the case of Dataset 2 enriched with contextual information of Hybrid Analysis APIs, a CNN-LSTM fusion model will be applied to be trained to learn the sequence of behavioral APIs and external metadata simultaneously. This flexible design enables the system to use more intelligence where it is present and still be efficient when used on bigger data sets.

Lastly, the trained models are tested on held-out test sets with Accuracy, Precision, Recall, F1-score and AUC to ensure that they are well tested. The generalization capability of the flexible design facilitates the usage of the framework to a variety of scales of data sets and feature configurations, which offers scalability, robustness, and reliability in practical applications of cybersecurity.

3.1 Utilized Datasets

In this study, three datasets were employed to evaluate the proposed hybrid framework. Dataset 1 consists of API call sequences collected from academic corpora, Dataset 2 extends the same dataset with additional external features from Hybrid Analysis APIs, and Dataset 3 is a large-scale Smote-API sequence balanced dataset compiled in 2025 to assess scalability and generalization. A detailed description of each dataset is provided below.

Dataset 1. In malware detection research, the MalBehavD-V1 dataset [11] and Angelo Oliveira's API Call Sequences dataset [12] are widely used to evaluate and train machine learning and deep learning models that rely on behavioral features. In this study, the two corpora were merged to form a benchmark dataset consisting of 3,500 executable samples, including 1,762 malware and 1,738 benign programs.

Specifically, 2,474 samples were obtained from MalBehavD-V1, while 1,026 were sourced from Oliveira's dataset. Each sample is represented by API call sequences extracted from dynamic execution, which provide a consistent behavioral profile of the software. The nearly balanced distribution of Dataset 1 is illustrated in Figure 2.

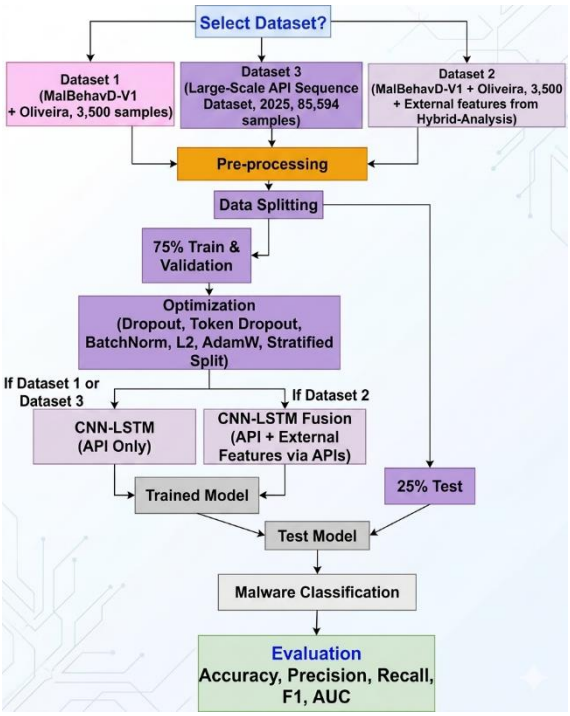

Figure 1: Framework of proposed system.

Dataset 2. To enrich behavioral data with external features, Dataset 1 was extended with attributes obtained from the Hybrid Analysis API. These enrichment features include threat scores, antivirus detection counts, verdicts, file type, and file size, which provide contextual metadata in addition to raw behavioral sequences.

The structure of Dataset 1 is presented in Figure 3 which shows the dataset composed only of hash identifiers, API sequences, and binary labels. In contrast, the enriched structure of Dataset 2 is illustrated in Figure 4 where the additional contextual attributes obtained from Hybrid Analysis are integrated with the behavioral sequences.

Dataset 3. The Large-Scale Smote-API call Sequence Dataset [13] is one of the most recent resources in malware detection. It comprises 85,594 balanced samples, including 42,797 malware and 42,797 benign executables. Each record contains a hash identifier, up to 100 API call tokens extracted from execution traces, and a binary malware label.

The class balance of Dataset 3 is illustrated in Figure 5, the feature structure is illustrated in Figure 6. No external enrichment features are included in the dataset and it only contains hash identifiers, API call sequences, and binary malware labels.

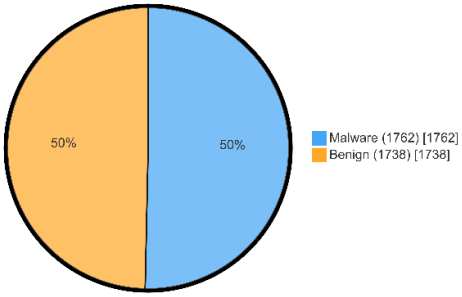

Figure 2: Class distribution of Dataset 1 (MalBehavD-V1 + Oliveira) and Dataset 2 (MalBehavD-V1 + Oliveira + external features from Hybrid Analysis system API).

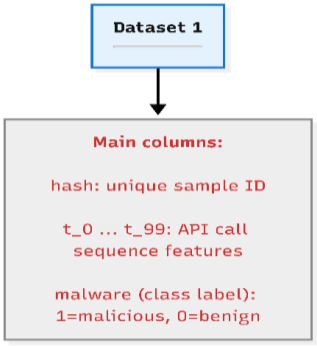

Figure 3: Features structure of Dataset 1.

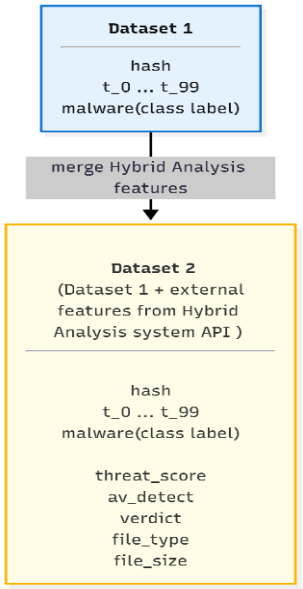

Figure 4: Features structure of Dataset 2.

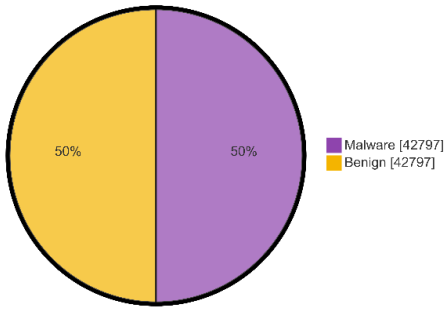

Figure 5: Class distribution of Dataset 3 (Smote API call Sequence Dataset, 2025).

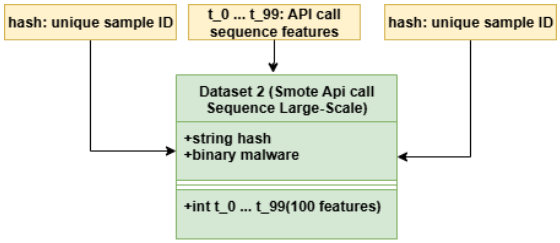

Figure 6: Features structure of Dataset 3.

These datasets can be used to assess the systems: Dataset 1 enables the assessment of systems that rely solely on behavioral API sequences, Dataset 2 can assess the system based on the additional contextual features of Hybrid Analysis, and Dataset 3 can indicate the scalability of the behavioral conditions. Such design emphasizes flexibility of the offered system to various situations of deployment.

3.2 Data Pre-Processing

The datasets should be preprocessed effectively to ensure the deep learning preparation and to ensure the leakage-free evaluation is consistent. Due to the differences in the representations of the three datasets, each of them was preprocessed in a specific way.

- Dataset 1. API call sequences were partly stored as text and partly as numbers. Textual API names were mapped to integer identifiers, while values that were already numeric were kept unchanged. Unseen or rare APIs were assigned to an unknown token (UNK = 1). Since the length of API call sequences varied across samples (some shorter than 100 calls and others longer), all sequences were standardized to a fixed length of 100 API calls. Sequences shorter than 100 were initially filled with −1 to indicate missing positions, and later these values were replaced with PAD = 0. Sequences longer than 100 were truncated to the first 100 calls. This ensured a uniform representation across all samples.
- Dataset 2. The same preprocessing steps as Dataset 1 were applied to the API sequences. In addition, this dataset included external features obtained from Hybrid Analysis.
  1) Numeric features (e.g., threat score, detection count, file size) were normalized in two stages: first, log(1+x) scaling was applied to reduce skewness in highly variable attributes such as file size, then z-score standardization was applied column-wise so that each feature had a mean of 0 and a standard deviation of 1. This prevented large-valued features from dominating the learning process.
  2) Categorical features (e.g., verdict, file type) were represented using Top-K one-hot encoding. The 20 most frequent values were given their own one-hot columns, while all other rare values were grouped together into a single "OTHER" column. For example, if EXE, DLL, and PDF are among the most common file types, they receive their own one-hot positions, while any other type (e.g., DOC, RAR) is mapped to the "OTHER" position.
  3) Missing values in external features were imputed with the median of the training split. To explicitly retain the information about missingness, binary indicators (e.g., threat_score_isna, file_size_isna) were added to mark whether a value was originally missing (1 = missing, 0 = present).
- Dataset 3. This large-scale dataset already contained integer-encoded API sequences, so no textual mapping was required. Preprocessing was therefore limited to handling missing values (−1 → PAD= 0), padding sequences to a fixed length, and applying token-level dropout during training to improve robustness and generalization.

Finally, all datasets were split using stratified sampling to preserve the balance between malware and benign classes. 75% of the samples were allocated for training and 25% for testing. Based on the training part, a subsample of internal validation (20 percent) was obtained to track the progress of learning and to control the decision threshold. This hierarchical process provided a balance in the overall assessments and stopped information leakage of validation or test sets into training. This workflow of preprocessing is summarized in Figure 7. The preprocessing pipeline, although a brief comparison between the three datasets is provided in Table 2.

3.3 API Call Features and Representation

Windows API call sequences were used as the main behavioral features in this work. The API call is central to dynamic malware analysis since it can give first hand information on the behavior of runnable code without having to access the source code. Through tracking of API sequences, one can get to capture actions like file manipulation, registry modification, process injection, and network communications that are a good indication of malicious intent.

Figure 8 shows that the procedure of gathering and describing API sequences was organized as a pipeline. Firstly malware and benign programs were run in a controlled environment, with monitoring tools like Sysmon, API Monitor or sandbox frameworks capturing their runtime behavior. The raw logs (XML, JSON or TXT) obtained were then processed to get API call sequences. All API names mapped to integer identifier and values that had been put in numerical form were retained. In order to be consistent across samples, all sequences were trimmed to the standard length of 100 calls: shorter sequences were padded and longer ones cut off. Missing values were first represented by the value -1, but this was substituted with a special padding token (PAD = 0). Unknown APIs that were rare or not observable were given a special token (UNK = 1).

This integer-based representation maintains the sequence of execution of API calls and offers a standard input format to subsequent modeling. The reason why API call sequences are selected as key features is due to the fact that they represent dynamic behavioral characteristics that cannot be easily obfuscated. Conversely, the code structure (non-runtime) features like opcodes or raw bytes are represented by static features and thus become more vulnerable to obfuscation and packing. The sequence-based features are stronger as API calls directly indicate runtime activities, such as file operations, registry changes, and network interactions, which are likely to remain consistent among multiple malware families. Furthermore, API-based representations are sparse but rich and do not need preprocessing as much as image-based and other high-dimensional static representations.

Windows API call sequences were used in this research as the primary behavioral features. The API calls are important in dynamic malware analysis since it provides a direct visibility of the runtime activity of a program without access to a source code. By tracing these sequences, one can trace some behavior like: file manipulation, editing of the registry, process injection and network activity all of which are highly probative of possible malicious intent.

The API names in text were coded into the integer identifiers and the numeric values were left in their original form. The unknown tokens (UNK = 1) were used to represent rare or previously unknown APIs and padding tokens (PAD = 0) were employed to fill gaps (−1) and make sure that all sequences had the same length.

Table 2: Summary of steps across datasets.

Dataset	API Sequences Processing	External Features Processing	Missing Value Handling	Notes
Dataset 1	Mixed input: text values mapped to integer IDs, numeric values kept as is, UNK=1 for rare APIs, standardized to fixed length = 100, PAD=0 used for padding	Not included	−1 → PAD=0 (for API calls)	Behavioral features only(no external features)
Dataset 2	Same as Dataset 1	Numeric features: log(1+x) + z-score standardization Categorical features: Top-20 one-hot (others → OTHER)	Median imputation (training split only) + binary missing indicators (e.g., threat_score_isna)	Combines behavioral and external features
Dataset 3	Already integer-encoded with fixed length = 100	Not included	−1 → PAD=0	Large-scale dataset (no external features)

Figure 7: Preprocessing workflow across the three datasets.

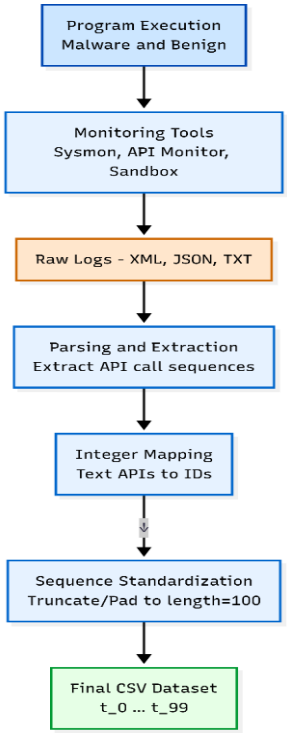

Figure 8: Workflow of API call collection and preprocessing for dataset representation.

This integer-encoded representation maintains the original sequence of execution of API calls and provides a standardized input representation to do subsequent modeling that allows the capture of short and long-term behavioral dependencies respectively.

3.4 Hybrid CNN-LSTM

In order to be able to capture local and long-term dependencies in API call chains, the hybrid deep learning model utilizing Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) nets was designed. The CNN layers are highly applicable in detection of local n-gram based behaviour patterns whereas the LSTM layers map through time sequential dependencies which enable the system to comprehend complex behavioral structure and contextual association. This architecture is a combination of accuracy, scalability and generalization hence appropriate in real-life malware detection.

3.4.1 CNN-LSTM for API-Only Datasets (Dataset 1 and Dataset 3)

The framework proposed in the Dataset 1 and Dataset 3 uses API call sequences as the sole behavioral input

feature, but does not involve any external attributes. Each sample is represented as a fixed length sequence of a hundred integer-encoded API identifiers as shown in Figure 9 and Table 3. Short sequences get padded with tokens (PAD = 0), and tokens of rare or unknown calls are assigned with an unknown token (UNK = 1).

The sequence is then passed through an embedding layer that transforms discrete tokens into dense numerical vectors that capture semantic relationships among API calls and disregard padding positions. It is followed by the embedded representation being fed through one-dimensional convolutional layers (1D CNNs) which identify local patterns of short-range, e.g., common file or registry actions. The resulting feature maps are passed to LSTM layers, which model long-term temporal dependencies across the entire sequence, learning how individual actions evolve into extended behavioral chains.

The final hidden state of the LSTM is a learned API feature vector, which is a summary of dynamical behavior of the sample. This is then passed through layers of dense with dropout and batch normalization to enhance robustness and to avoid overfitting. The sigmoid output layer is used to give a binary probability of whether the sequence is a benign or a malicious sample. This simplified CNN-LSTM architecture is both highly accurate and computationally efficient, and useful on both small-scale data (Dataset 1) and large scale behavioral data (Dataset 3).

Table 3. Configuration details of CNN-LSTM model.

Layer (Type)	Output Shape	Param #	Corresponding Block (Aligned with Fig. 9)
tokens (InputLayer)	(None, 100)	0	API Sequences (IDs)
tokendrop (TokenDrop)	(None, 100)	0	API Sequences (Regularization)
embedding (Embedding)	(None, 100, 32)	9,856	Embedding Layer
emb_spdrop (SpatialDropout1D)	(None, 100, 32)	0	Embedding Regularization
rcb1_conv1 (Conv1D)	(None, 100, 64)	18,496	1D CNN (Block 1)
rcb1_bn1 (BatchNormalization)	(None, 100, 64)	256	1D CNN (Block 1)
rcb1_relu1 (Activation)	(None, 100, 64)	0	1D CNN (Block 1)
rcb1_drop1 (Dropout)	(None, 100, 64)	0	1D CNN (Block 1)
rcb1_conv2 (Conv1D)	(None, 100, 64)	36,928	1D CNN (Block 1)
rcb1_proj (Conv1D)	(None, 100, 64)	2,112	1D CNN (Residual Path 1)
rcb1_bn2 (BatchNormalization)	(None, 100, 64)	256	1D CNN (Block 1)
rcb1_add (Add)	(None, 100, 64)	0	1D CNN (Residual Merge 1)
rcb1_relu_out (Activation)	(None, 100, 64)	0	1D CNN (Block 1 Output)
rcb1_pool (MaxPooling1D)	(None, 50, 64)	0	1D CNN (Pooling 1)
rcb2_conv1 (Conv1D)	(None, 50, 96)	43,104	1D CNN (Block 2)
rcb2_bn1 (BatchNormalization)	(None, 50, 96)	384	1D CNN (Block 2)
rcb2_relu1 (Activation)	(None, 50, 96)	0	1D CNN (Block 2)
rcb2_drop1 (Dropout)	(None, 50, 96)	0	1D CNN (Block 2)
rcb2_conv2 (Conv1D)	(None, 50, 96)	64,608	1D CNN (Block 2)
rcb2_proj (Conv1D)	(None, 50, 96)	6,240	1D CNN (Residual Path 2)
rcb2_bn2 (BatchNormalization)	(None, 50, 96)	384	1D CNN (Block 2)
rcb2_add (Add)	(None, 50, 96)	0	1D CNN (Residual Merge 2)
rcb2_relu_out (Activation)	(None, 50, 96)	0	1D CNN (Block 2 Output)
rcb2_pool (MaxPooling1D)	(None, 25, 96)	0	1D CNN (Pooling 2)
lstm1 (LSTM)	(None, 25, 256)	361,472	LSTM Layer 1
lstm2 (LSTM)	(None, 25, 128)	197,120	LSTM Layer 2
gmp (GlobalMaxPooling1D)	(None, 128)	0	Learned API Feature Vector (Aggregation)
gap (GlobalAveragePooling1D)	(None, 128)	0	Learned API Feature Vector (Aggregation)
concat_feats (Concatenate)	(None, 256)	0	Learned API Feature Vector (Fusion)
feat_drop (Dropout)	(None, 256)	0	Learned API Feature Vector (Regularization)
fc1 (Dense)	(None, 256)	65,792	Dense Layers
fc1_drop (Dropout)	(None, 256)	0	Dense Layers
out (Dense)	(None, 1)	257	Sigmoid Output (Malware vs Benign)
Total Parameters: 807,265 (3.08 MB) Trainable Parameters: 806,625 (3.08 MB) Non-trainable Parameters: 640 (2.50 KB)			

3.4.2 Hybrid CNN-LSTM with Fusion (Dataset 2)

In Dataset 2, the system builds upon the same CNN-LSTM architecture by adding more external features which are acquired through the Hybrid Analysis system, and integrates behavioral and contextual information with each other through the hybrid model. Figure 10 and Table 4 illustrate the overall structure comprising of two parallel branches and a fusion layer.

The API-sequence branch is implemented in the same way as Dataset 1 and 3: embedding the integer-encoded API calls, convolutional filters are used to extract local patterns and long-range sequential dependencies are captured with LSTM layers. The output of the last LSTM is summarized as learned API feature vector which depicts the behavioral profile of the program.

Table 4: Configuration details of CNN-LSTM with fusion model.

Layer (Type)	Output Shape	Param	Corresponding Block (from Fig. 10)
tokens (InputLayer)	(None, 100)	0	API Sequences (IDs)
token_drop (TokenDrop)	(None, 100)	0	API Sequences (Regularization)
embedding (Embedding)	(None, 100, 32)	9,824	Embedding Layer
emb_spdrop (SpatialDropout1D)	(None, 100, 32)	0	Embedding Regularization
conv1d (Conv1D)	(None, 100, 64)	18,496	1D CNN (Block 1)
batch_normalization	(None, 100, 64)	256	1D CNN (Block 1)
re_lu (ReLU)	(None, 100, 64)	0	1D CNN (Block 1)
dropout (Dropout)	(None, 100, 64)	0	1D CNN (Block 1)
conv1d_1 (Conv1D)	(None, 100, 64)	36,928	1D CNN (Block 1)
conv1d_2 (Conv1D)	(None, 100, 64)	2,112	1D CNN (Residual Path 1)
batch_normalization_1	(None, 100, 64)	256	1D CNN (Block 1)
add (Add)	(None, 100, 64)	0	1D CNN (Residual Merge 1)
re_lu_1 (ReLU)	(None, 100, 64)	0	1D CNN (Block 1 Output)
max_pooling1d (MaxPooling1D)	(None, 50, 64)	0	1D CNN (Pooling 1)
conv1d_3 (Conv1D)	(None, 50, 96)	43,104	1D CNN (Block 2)
batch_normalization_2	(None, 50, 96)	384	1D CNN (Block 2)
re_lu_2 (ReLU)	(None, 50, 96)	0	1D CNN (Block 2)
dropout_1 (Dropout)	(None, 50, 96)	0	1D CNN (Block 2)
conv1d_4 (Conv1D)	(None, 50, 96)	64,608	1D CNN (Block 2)
conv1d_5 (Conv1D)	(None, 50, 96)	6,240	1D CNN (Residual Path 2)
batch_normalization_3	(None, 50, 96)	384	1D CNN (Block 2)
add_1 (Add)	(None, 50, 96)	0	1D CNN (Residual Merge 2)
re_lu_3 (ReLU)	(None, 50, 96)	0	1D CNN (Block 2 Output)
max_pooling1d_1 (MaxPooling1D)	(None, 25, 96)	0	1D CNN (Pooling 2)
lstm1 (LSTM)	(None, 25, 256)	361,472	LSTM Layer 1
lstm2 (LSTM)	(None, 25, 128)	197,120	LSTM Layer 2
global_max_pooling1d	(None, 128)	0	Learned API Feature Vector (Aggregation)
global_average_pooling1d	(None, 128)	0	Learned API Feature Vector (Aggregation)
seq_pool (Concatenate)	(None, 256)	0	Learned API Feature Vector (Fusion)
seq_drop (Dropout)	(None, 256)	0	Learned API Feature Vector (Regularization)
seq_fc (Dense)	(None, 256)	65,792	Learned API Feature Vector (Dense Transformation)
ha_feats (InputLayer)	(None, 19)	0	External Features
ha_fc1 (Dense)	(None, 64)	1,280	Normalization & Encoding
ha_drop1 (Dropout)	(None, 64)	0	External Features (Regularization)
ha_fc2 (Dense)	(None, 32)	2,080	Learned External Feature Vector
fusion_concat (Concatenate)	(None, 288)	0	Fusion (Concatenate)
fusion_fc1 (Dense)	(None, 256)	73,984	Dense Layers (Post-Fusion)
fusion_drop (Dropout)	(None, 256)	0	Dense Layers (Regularization)
out (Dense)	(None, 1)	257	Sigmoid Output (Malware vs Benign)
Total Parameters: 884,577 (3.37 MB) Trainable Parameters: 883,937 (3.37 MB) Non-trainable Parameters: 640 (2.50 KB)			

In parallel, numerical and categorical attributes inputted by the Hybrid Analysis system are processed by the external-features branch. Numerical variables (e.g., threat score, detection count, file size) are normalized using log(1 + x) and then z-score scaled, and categorical variables (e.g., verdict, file type) are one-hot encoded with top-K groups and grouping infrequent categories into an (OTHER) bucket. The median of the training split is put in where the values are missing, and binary indicators are introduced to maintain missingness information. The processed features are then run over dense layers including dropout and batch normalization to produce a learned external-feature vector.

The results of the two branches are fused in a fusion layer, which constitutes a combination of behavioral and external features. This merged vector is then narrowed down with extra layers of dense, dropout, batch normalization and L2 regularization. The last sigmoid unit will give the likelihood of the sample being malicious or benign.

In the hybrid CNN-LSTM model, two feature vectors are produced independently: $F_{seq} \in \mathbb{R}^{d_1}$ from the API-sequence branch and $F_{ext} \in \mathbb{R}^{d_2}$ from the external-feature branch. These two representations are fused by concatenation at the feature level to preserve complementary information:

$$F_{fusion} = [\, F_{seq} \;||\; F_{ext} \,], \tag{1}$$

where ||denotes the concatenation operator. The fused vector F_{fusion} is then passed through fully connected layers with dropout and L2 regularization to produce the final output probability $\hat{y}$through a sigmoid activation:

$$\hat{y} = \sigma(W \cdot F_{fusion} + b). \tag{2}$$

This formulation allows the model to jointly learn behavioral (sequence-based) and contextual (external) patterns while maintaining computational efficiency.

The proposed hybrid structure allows the two runtime behavior (provided by API sequences) and analytical metadata (provided by external threat-analysis sources) to be used by this framework to enhance detection accuracy, scalability, and generalization over sequence-only models.

3.5 Training Hyperparameters

The training configuration was identical across all experiments except for the presence of external features in the hybrid variant.

Table 5 summarizes the key hyperparameters used to train both models.

Both networks employed the AdamW optimizer with cosine learning-rate scheduling, a batch size of 512, and dropout layers to reduce overfitting.

Binary Cross-Entropy was used as the loss function, and data were split using a stratified 75/20/25 ratio for training, validation, and testing, respectively.

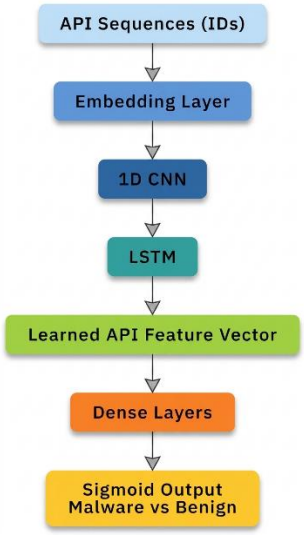

Figure 9: CNN-LSTM architecture for API-only datasets (Dataset 1 and Dataset 3).

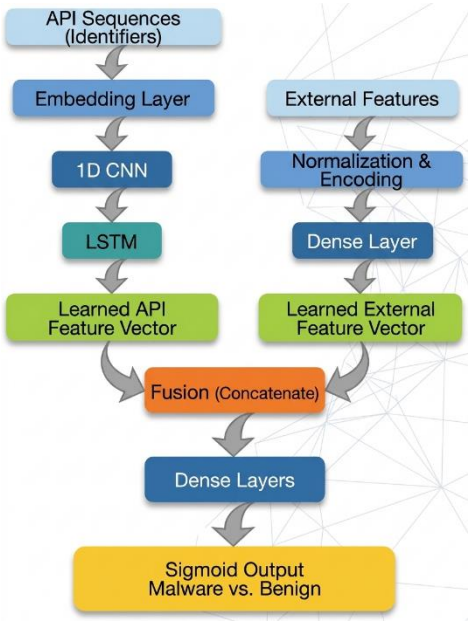

Figure 10: Hybrid CNN-LSTM architecture with fusion of API-sequence and external-features branches (Dataset 2).

Table 5: Training hyperparameters for CNN-LSTM and hybrid CNN-LSTM models.

Parameter	CNN-LSTM	Hybrid CNN-LSTM
API sequence length	100	100
Learning rate	0.001 (1×10^{-3})	0.001 (1×10^{-3})
Batch size	512	512
Optimizer	AdamW (with CosineDecayRestarts)	AdamW (with CosineDecayRestarts)
Number of epochs	150	150
Dropout rates	0.05 (token), 0.15 (spatial), 0.30–0.35 (LSTM), 0.25 (dense)	0.05 (token), 0.15 (spatial), 0.30–0.35 (LSTM), 0.25 (dense)
Loss function	Binary Cross-Entropy	Binary Cross-Entropy
Data split (train/val/test)	75% / 20% / 25% (stratified)	75% / 20% / 25% (stratified)

4 EXPERIMENTAL RESULTS

In this section, the results of the evaluation of the proposed CNN-LSTM framework on the three datasets. The experiments have provided validation and test results on various metrics, which would provide fair and consistent comparisons.

4.1 Performance Measures

Our research uses malware detection models as the evaluation criteria, which are described in this subsection and are critical in assessing the usefulness of models used to detect malware. These measures are a measurement of the quality of a classifier that separates malicious software and benign programs. Base measures of malware detection research continue to make use of accuracy and AUC [12]. Validation Accuracy also comes in to portray the model performance in the training process and avoid overfitting [12]. Moreover, Precision, Recall, F1-Score, and Specificity also give a more specific picture of the classification balance and make sure that all false positives and false negatives are both represented correctly [14], [15].

The recent studies have solidified the importance of evaluation measures in evaluation of malware detection systems. As Miller et al. (2024) note, the Accuracy, Precision, Recall, F1, and AUC metrics are broadly applied, but they should be interpreted with care when presented with an imbalanced cybersecurity datasets [16]. To build on the concept, Beddar-Wiesing et al. (2025) propose absolute evaluation measures to allow a more fair comparing of models and datasets [14]. On the same note, a predictive performance benchmarking framework offered by Cardoso et al. (2025) should also evaluate predictive performance and robustness jointly, which is why the measures of Specificity and ROC-AUC remain relevant in identifying malware [4].

4.2 Experimental Setup

All experiments were coded in Python 3.12.5 using TensorFlow 2.x and auxiliary libraries (NumPy, Pandas, and Scikit-learn). The proposed system was trained and evaluated on a consumer-grade machine, demonstrating that the framework can be replicated without requiring expensive hardware or high-end GPUs.

The hardware and software configuration used in this study is summarized below:

- CPU: Intel® Core™ i5, 11th Generation.
- GPU: Intel® Iris® Xe Graphics (4 GB total memory).
- Memory: 8 GB RAM.
- Operating System: Windows 10 (64-bit).
- Frameworks: Python 3.12.5, TensorFlow 2.x, Keras, NumPy, Pandas, Scikit-learn.

4.3 Results of the Proposed System

In this subsection, we provide a summarized overview of the findings obtained across the three evaluated datasets and highlight the performance differences between the baseline models and the proposed hybrid CNN-LSTM architecture. This comparison illustrates how each model behaves under different dataset scales and feature configurations.
The evaluation includes Dataset 1 (MalBehavD-V1 + Oliveira), Dataset 2 (the same dataset enriched with external features extracted from Hybrid-Analysis APIs), and Dataset 3 (the Large-Scale SMOTE-balanced API call sequence dataset, 2025). Performance metrics such as validation accuracy, test accuracy, precision, recall, F1-score, specificity, and AUC were used to provide a complete picture of the classification capability and the reliability of the models in distinguishing malicious and benign samples.

Figure 11 presents the confusion matrices for all evaluated models, showing that although standalone CNN and LSTM models achieved stable

performance, they produced more misclassification errors, particularly with more complex API sequences. In contrast, the hybrid CNN-LSTM architecture demonstrated fewer errors due to its ability to learn local API features through CNN layers while capturing long-range temporal relationships using LSTM components.

Additionally, the inclusion of external features in Dataset 2 further improved detection accuracy and reduced false positives and false negatives, confirming the value of these features. The strong results observed in Dataset 3 also highlight the system's consistency, scalability, and its high generalization ability, since Dataset 3 was specifically used to evaluate performance on a larger and more diverse API-sequence corpus.

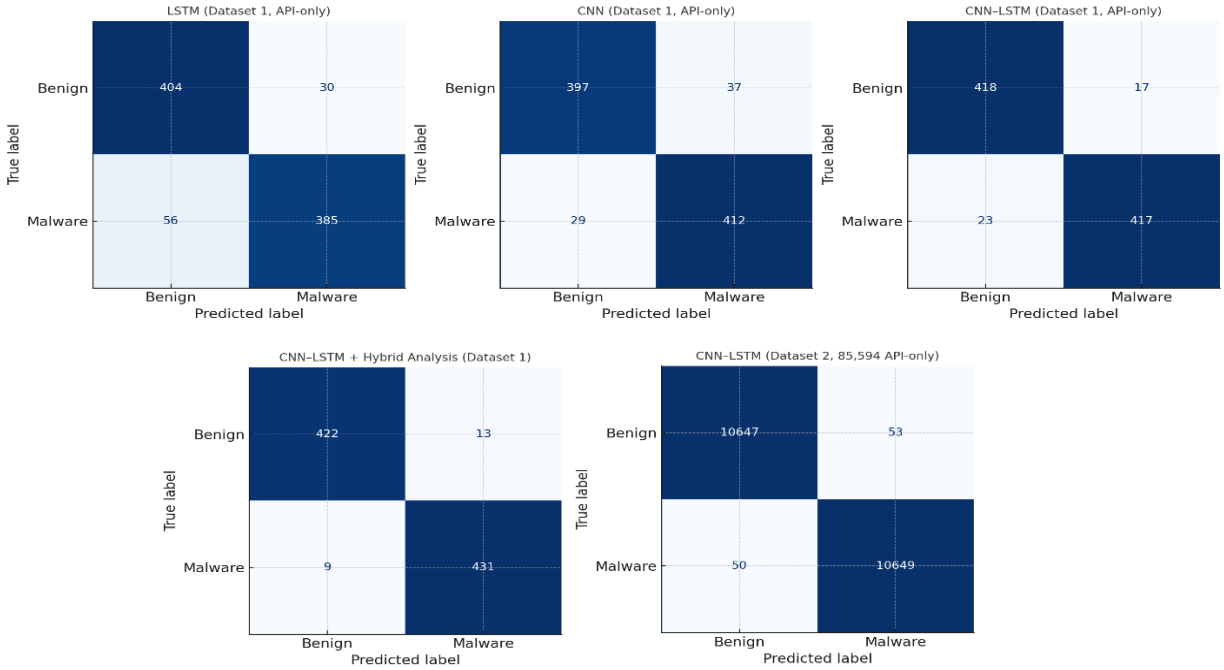

Figure 11: Confusion matrices of the evaluated models across both datasets.

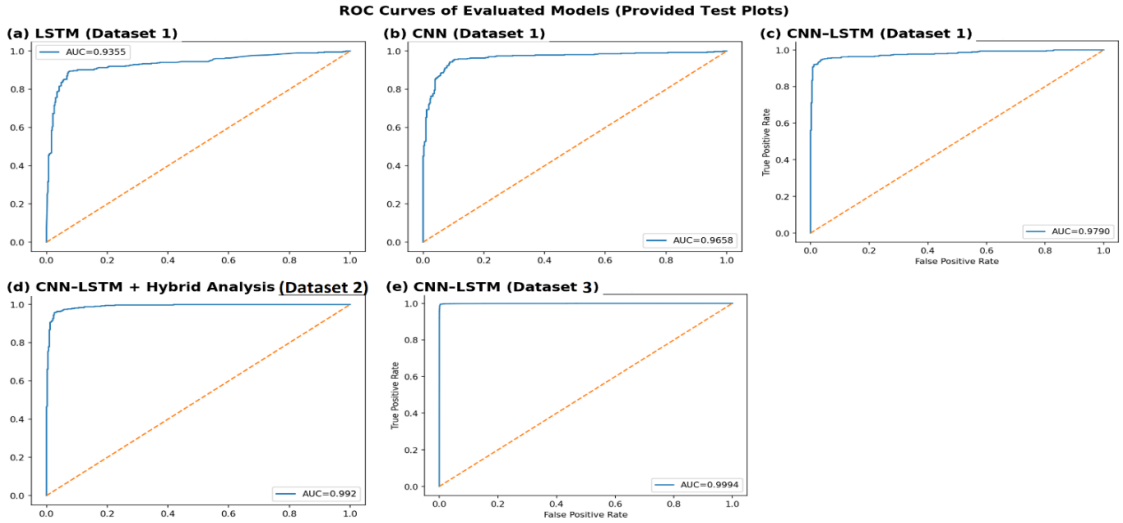

Figure 12: ROC curves of the evaluated models (test results provided): a) LSTM (Dataset 1), b) CNN (Dataset 1), c) CNN-LSTM (Dataset 1), d) CNN-LSTM + Hybrid_Analysis (Dataset 2), and e) CNN-LSTM (Dataset 3).

The discriminative ability of the models using ROC curves illustrated in Figure 12, which determine the ability of each model to distinguish between benign and malicious samples in each of the three datasets.

In Dataset 1 (MalBehavD-V1 + Oliveira, API-only), the LSTM alone and CNN alone gave AUC scores of 0.9355 and 0.9658 respectively, whereas the CNN-LSTM (API-only) gave 0.9790, which is higher than the single models.

In Dataset 2 (Dataset 1 with Hybrid Analysis features, e.g. threat scores, AV detections, verdict, file type, and file size), CNN-LSTM (Hybrid) achieved AUC 0.992, indicating that the classification of behavioral sequences integrated with external features gives more confident and reliable results.

In the case of Dataset 3 (Large-Scale Smote-API Sequence Dataset, 2025; balanced, API-only no external features), the CNN-LSTM had the highest AUC of 0.9994, which verifies the good generalization and scalability to large-scale behavioral data.

It should be noted that the general pattern of the ROC curves is as follows: the introduction of hybrid characteristics (Dataset 2) and scaling to large, balanced corpus (Dataset 3) boost discriminative performance compared to single-model baselines and the smaller, API-only environment (Dataset 1).

The consolidated report of all performance measures is given in Table 6 and reports on validation accuracy and test accuracy, precision, recall, F1-score, specificity and AUC. The table indicates that though the baseline CNN and LSTM models on the MalBehavD-V1 + Oliveira dataset (Dataset 1, API-only) record reasonable performance, the CNN-LSTM frameworks are steadily effective in improving the results, which indicates that the frameworks can capture the local and sequential dependencies of API behavior.

More improvements are seen with Dataset 2, where Hybrid Analysis features are added to the same behavioural structure. The CNN-LSTM (Hybrid) model had the best performance and displayed the highest test accuracy of 99.20 and AUC of 0.9993, which validates the usefulness of uniting the behavioral and Hybrid Analysis features.

On the Large-Scale Smote API Call Sequence Dataset (Dataset 3), the CNN-LSTM maintained excellent generalization, reaching 99.52% test accuracy and an AUC of 0.9994, which demonstrates the scalability and robustness of the proposed approach across large and balanced datasets.

The evaluation was conducted on three test sets: 875 samples for Dataset 1 (435 benign and 440 malware), the same number of samples (875) for Dataset 2 (Hybrid-enriched version), and 21,399 samples for Dataset 3 (Large-Scale Smote API Sequence Dataset, 2025), ensuring a comprehensive and fair assessment across different dataset configurations.

We summarize validation/test metrics across all models to highlight relative gains, see Figure 13.

Figures 14-17 represent the performance of the models in six core metrics presented as radar charts. These graphs indicate the performance parity obtained with CNN_LSTM model on the MalBehavD-V1 + Oliveira dataset (Dataset 1), and why more pronounced outcomes are realized when Hybrid Analysis features are incorporated in Dataset 2. The almost perfect and extremely consistent results obtained on the Large-Scale Smote API Call Sequence Dataset (2025, Dataset 3) are also presented in the charts.

Table 6: Performance comparison of baseline and hybrid models on both datasets.

Model	Validation Accuracy	Test Accuracy	Precision	Recall	F1-score	Specificity	AUC	Threshold
LSTM-only (Dataset 1)	0.9048	0.9017	0.9030	0.9019	0.9017	0.9309	0.9355	0.50
CNN-only (Dataset 1)	0.9143	0.9246	0.9248	0.9245	0.9245	0.9147	0.9658	0.50
CNN-LSTM (Dataset 1)	0.9524	0.9543	0.9543	0.9543	0.9543	0.9609	0.9790	0.070
CNN-LSTM + Hybrid Analysis (Dataset 2)	0.9771	0.9749	0.9707	0.9795	0.9751	0.9701	0.9931	0.985
CNN-LSTM (Dataset 3)	0.9970	0.9952	0.9952	0.9952	0.9952	0.9950	0.9994	0.280

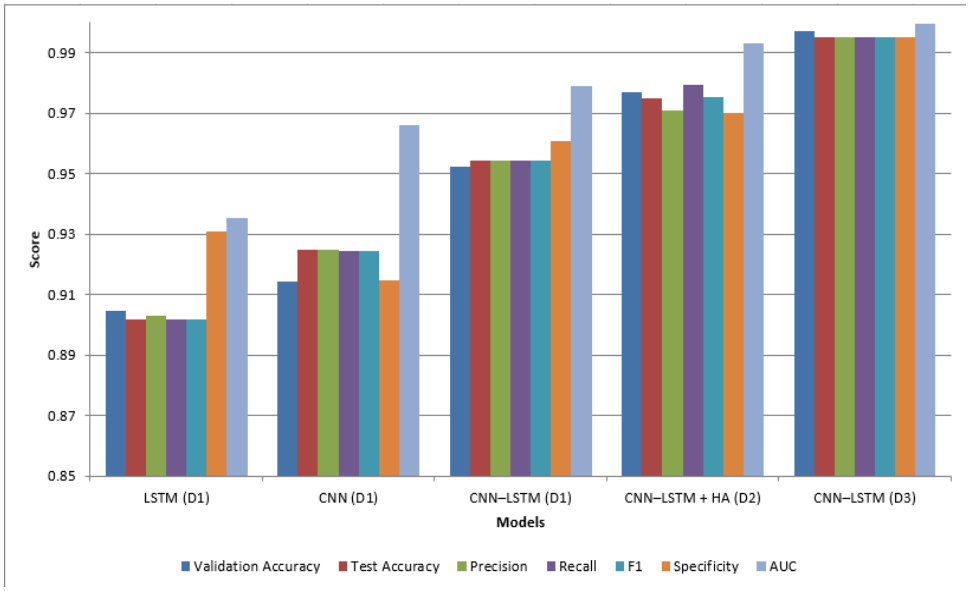

Figure 13: Comparative performance metrics of the evaluated models across the datasets.

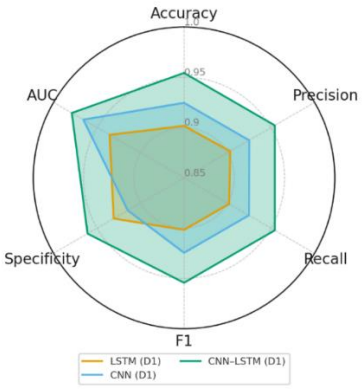

Figure 14: Radar chart of Dataset 1 (MalBehavD-V1 + Oliveira) models.

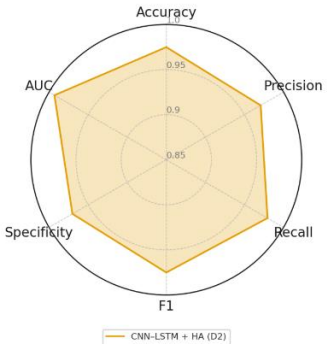

Figure 15: Radar chart of Dataset 2 (MalBehavD-V1 + Oliveira+ Hybrid Analysis Features) models.

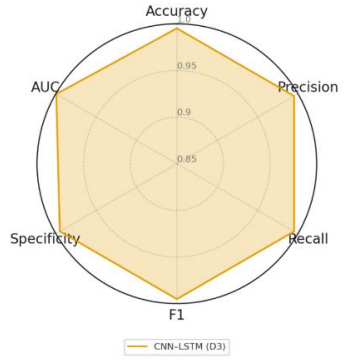

Figure 16: Radar chart of Dataset 3 (Large-Scale API Sequence Dataset, 2025) model.

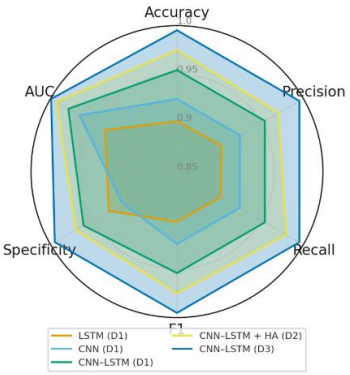

Figure 17: Radar chart representation of the evaluated models across six performance metrics (Accuracy, Precision, Recall, F1-score, Specificity, and AUC) on all datasets.

The radar charts presented by showing all metrics at the same time clearly indicate the fact that the proposed hybrid system provides high, stable, and scalable performance under varied datasets and evaluation metrics.

5 DISCUSSION

As shown by the experimental findings, the hybrid CNN-LSTM model can be successfully used to overcome most of the issues that have been noted in previous malware detection studies. On Dataset 1 (MalBehavD-V1 + Oliveira), the standalone CNN had a higher accuracy of 92.46 but had more false positives, whereas the LSTM-only model had a lower accuracy of 90.17 with reduced false negatives. When the two architectures were put together the CNN-LSTM topped 95.43% accuracy, which means that hybrid architectures do learn local and sequential dependencies of API behavior.

The existing enhancements were made on Dataset 2, where other Hybrid-Analysis characteristics were incorporated into the behavioral dataset. The resulting CNN-LSTM + Hybrid Analysis model was found to achieve 97.49 percent accuracy with AUC at 0.9931, indicating that the addition of external features (e.g. detection counts, verdicts, file type, and metadata) to the API-based behavioral sequences can enhance reliability as well as the reduction of false positive and false negative outcomes.

With no external features used, the CNN-LSTM recorded an accuracy of 99.52 and an AUC of 0.9994 on Dataset 3 (Large-Scale API Sequence Dataset, 2025). This shows that the scalability and stability of the proposed framework is justified since large, balanced datasets have the inherent capacity to promote better generalization.

Table 7: Comparative summary of representative studies and the proposed system.

Authors, Year	Datasets	Methodology	Performance	Limitations / Key Insight
Karat et al. [8], 2024	Rohitab API Monitor, Sysmon (~2.5k)	CNN-LSTM (Conv1D + LSTM)	Acc = 96%	Small dataset; high GPU dependency; limited reproducibility
Thakur et al. [5], 2024	Public malware binaries (image conversion)	CNN-LSTM + PCA-based feature reduction	Acc = N/A	High preprocessing complexity; limited to visual datasets
Bamber et al. [4], 2025	NSL-KDD	CNN-LSTM + RFE + Decision Tree feature ranking	Acc = 95%, F1 = 0.94	Moderate accuracy and F1; focused on network intrusion, not behavioral malware
Maniriho et al. [1], 2024	Custom traces (20–50 API calls)	GPT-2 + DistilBERT + BiGRU	Acc ≈ 95–96%	High model complexity; limited to short API traces
Qian & Cong [7], 2024	Mal-API-2019	CAFTrans (frequency + transformer)	F1 = 0.65, AUC = 0.79	Weak temporal modeling; outdated dataset; imbalance issues
Alshomrani et al. [6], 2025	Malimg, MaleVis, VirusMNIST	ConvNeXt-Tiny + Swin Transformer	Acc = 94–98%	Heavy image preprocessing; transformer GPU-intensive
Li et al. [9], 2025	Cuckoo Sandbox (~43k)	1D-CNN (API Phrase) + Bi-LSTM (Semantic Chain)	Acc = 97.31%, F1 = 0.97	Accuracy drops on newer data; complex feature engineering
Proposed System (Dataset 1)	MalBehavD-V1 + Oliveira (API-only, 3,500 samples)	CNN-LSTM (Conv1D + LSTM, integer encoding, dropout)	Acc = 95.43%, AUC = 0.9790	Balanced results; small-scale behavioral dataset; baseline hybrid performance
Proposed System (Dataset 2)	Dataset 1 + external Hybrid-Analysis features	CNN-LSTM + feature-level fusion (API + contextual attributes)	Acc = 97.49%, AUC = 0.9931	Stronger accuracy from feature enrichment; requires external data access
Proposed System (Dataset 3)	Large-Scale API Sequence Dataset (2025, 85,594 samples)	CNN-LSTM (optimized, balanced large-scale behavioral data)	Acc = 99.52%, AUC = 0.9994	Demonstrates scalability and generalization; no external enrichment needed

Compared to the current research that can be summarized as Table 7, there is a clear uniform positive result of the proposed system in terms of accuracy and design efficiency. As an example, Karat et al. [8] obtained 96% validation accuracy with the identical dataset combination but with a greater computation demand on the GPUs and low reproducibility. Although Bamber et al. [6] also employed a CNN-LSTM to detect intrusions, they got 95% and F1 = 0.94, which are lower than the scores of our framework. Their work validates the potential of hybrid deep learning, but also points towards the fact that generalization is a problem when models are based on network-level data only.

Similarly, a CNN-LSTM hybrid with a dimensionality reduction based on PCA was employed by Thakur et al. [5] to detect malware based on images. Even though they claimed high qualitative performance (high precision and recall), their method involved complex image preprocessing, which is not as practical when dealing with behavioral based sequence data. Conversely, our model encodes using integers and preprocesses in a structured manner, greatly decreasing the computational cost and maintaining very high predictive performance.

Additionally, transformer-based models including CAFTrans [7] and early-detection models including EarlyMalDetect [1] provided encouraging results but incorporated significant computational costs because of big-scale transformers or attention process. The proposed CNN-LSTM method obtains similar or even better performance at a reduced complexity which guarantees efficiency and practicability in the operational cybersecurity systems.

The hybrid CNN-LSTM model remains lightweight, with about 0.88 M parameters (3.37 MB as shown in Table 4), although slightly more complex than a standalone CNN, it is still much simpler and faster than Transformer-based models, this balance between efficiency and accuracy makes it suitable for real-world cybersecurity applications.

In general, the addition of external Hybrid-Analysis features (Dataset 2) and optimization methods including dropout, token-level dropout and adaptive learning scheduling are the two key contributors to the achieved performance improvement. All these design decisions create a robust, powerful, and scalable malware detection model, which is able to perform better than previous CNN-LSTM models on various data sets and evaluation measures.

6 CONCLUSIONS

This research presented a hybrid CNN-LSTM-based malware detection framework that integrates behavioral API sequence analysis with external features obtained from the Hybrid Analysis platform. The motivation behind the proposed system stems from limitations observed in traditional detection approaches that rely heavily on static signatures or handcrafted heuristics, which often fail when dealing with obfuscated or rapidly evolving malware. In contrast, the proposed hybrid model benefits from combining Convolutional Neural Networks (CNNs), which learn local API-call features, with Long Short-Term Memory (LSTM) layers, which capture long-range temporal relationships across execution sequences. This combination allows the system to extract fine-grained structural patterns while also modeling temporal dependencies that unfold during program behavior. Additionally, incorporating external features - such as detection counts, analysis verdicts, file attributes, and associated metadata - provides supplementary information that enhances the model's ability to resolve ambiguous cases and reduce the likelihood of misclassification. Altogether, these elements form a unified deep learning framework designed to increase accuracy, improve robustness, and maintain practical efficiency for real-world cybersecurity environments.

The effectiveness of the proposed system was validated through extensive experiments conducted on three datasets of increasing size and complexity. Dataset 1 (MalBehavD-V1 + Oliveira), consisting of 3,500 samples, enabled the evaluation of the hybrid model under moderate data conditions. The CNN-LSTM achieved an accuracy of 95.43%, outperforming both the standalone CNN and LSTM baselines. Dataset 2 enriched the same behavioral sequences with external features obtained from Hybrid Analysis, increasing accuracy to 97.49% and reaching an AUC of 0.9931, demonstrating the added value of external features in reducing false positives and false negatives. The third dataset - an extensive balanced corpus of 85,594 API-sequence samples - was used to examine scalability and broader applicability, where the hybrid model achieved 99.52% accuracy and an AUC of 0.9994. These results collectively demonstrate not only the model's strong predictive performance but also its high generalization ability, as it consistently maintained superior accuracy across datasets that differed in scale, composition, and feature configuration. When compared to prior works summarized in Table 7, the proposed system further showed competitive or

superior performance while maintaining lower computational overhead, reinforcing its suitability for practical, resource-efficient cybersecurity environments.

Several meaningful directions emerge for future work. One important direction involves improving the interpretability of the model. Since cybersecurity analysts often need to understand why a file was classified as malicious, integrating explainability techniques - such as highlighting influential API tokens or identifying which external features contributed most to the final decision - would increase trust and support forensic analysis. Another promising direction is enabling real-time or near-real-time detection. Currently, the system operates on fixed-length sequences; however, many operational settings involve continuous streams of API calls generated during live execution. Adapting the model to analyze sequences dynamically could allow earlier detection of malicious behavior and enable more proactive defensive measures. Furthermore, incorporating additional behavioral sources - including registry operations, file-system activities, process creation events, or network-related traces - could enrich the overall representation of program behavior. Such multi-source feature integration aligns naturally with the system's modular design, which already combines internal behavioral features with external features. Introducing lightweight optimizations, reducing inference latency, and exploring more compact model variants would further enhance operational feasibility, especially for endpoint protection systems and cloud-managed security solutions.

Overall, the findings of this research demonstrate that combining API-sequence behavior with external features inside a unified hybrid deep learning architecture can substantially improve malware detection accuracy, scalability, and resilience. The proposed CNN-LSTM framework offers a balanced combination of predictive strength and computational efficiency, distinguishing it from approaches that rely on costly preprocessing or transformer-based architectures with heavy resource requirements. Through integer encoding of API sequences, dropout regularization, batch normalization, and adaptive learning-rate scheduling, the framework achieves high performance without imposing excessive computational overhead. The consistency of the results across three datasets of different scales confirms the model's robustness and its capability to generalize well under varying data conditions. As cybersecurity threats continue to evolve, practical detection systems must maintain both strong performance and resource efficiency; this research contributes toward that goal by presenting a hybrid system capable of bridging the gap between academic experimentation and real operational needs.

Despite these strengths, the framework presents several limitations that should be addressed in future work. The inclusion of LSTM layers introduces higher memory usage and longer convergence time compared to a purely convolutional model. The reliance on fixed-length API sequences may also limit flexibility when analyzing variable-length traces or monitoring behavior in real time. Although Dataset 2 demonstrated the usefulness of external features, such information may not always be accessible in all deployment environments due to network restrictions or missing metadata. Future research could explore more efficient sequence modeling architectures, such as lightweight attention mechanisms or optimized recurrent units, to reduce training time and enhance scalability. Expanding the framework to support multi-class classification of malware families would also increase its analytical value by enabling not just detection but also behavior-based categorization. Evaluating the model in real operational settings, including environments with noisy or adversarial behavior logs, would provide further insight into its practical resilience. Addressing these directions would strengthen the proposed system and extend its utility for modern, evolving cybersecurity landscapes.

REFERENCES

[1] P. Maniriho, A. N. Mahmood, and M. Kim, "EarlyMalDetect: A Novel Approach for Early Windows Malware Detection Based on Sequences of API Calls," arXiv preprint arXiv:2407.13355, 2024.

[2] A. Hussain, A. Saadia, M. Alhussein, A. Gul, and K. Aurangzeb, "Enhancing Ransomware Defense: Deep Learning-Based Detection and Family-Wise Classification of Evolving Threats," 2025.

[3] Y. Song, D. Zhang, J. Wang, Y. Wang, Y. Wang, and P. Ding, "Application of Deep Learning in Malware Detection: A Review," Journal of Big Data, vol. 12, no. 1, p. 75, 2025.

[4] S. S. Bamber, A. V. R. Katkuri, S. Sharma, and M. Angurala, "A hybrid CNN-LSTM approach for intelligent cyber intrusion detection system," Computers & Security, vol. 148, p. 104146, Oct. 2025.

[5] P. Thakur, V. Kansal, and V. Rishiwal, "Hybrid deep learning approach based on LSTM and CNN for malware detection," Wireless Personal Communications, vol. 136, pp. 1879-1901, Jun. 2024.

[6] M. Alshomrani, A. Albeshri, A. A. Alsulami, and B. Alturki, "An Explainable Hybrid CNN–Transformer Architecture for Visual Malware Classification," Sensors, vol. 25, no. 15, p. 4581, 2025.

[7] L. Qian and L. Cong, "Channel Features and API-Frequency-Based Transformer (CAFTrans) for Malware Identification," Sensors, vol. 24, no. 2, p. 580, 2024.

[8] G. Karat, J. M. Kannimoola, N. Nair, A. Vazhayil, S. V. G., and P. Poornachandran, "CNN-LSTM Hybrid Model for Enhanced Malware Analysis and Detection," Procedia Computer Science, vol. 233, pp. 492-503, 2024.

[9] C. Li, Q. Lv, N. Li, Y. Wang, D. Sun, and Y. Qiao, "A Novel Deep Framework for Dynamic Malware Detection Based on API Sequence Intrinsic Features," Computers & Security, vol. 116, p. 102686, 2022.

[10] 10 API-Call Sequences Dataset by Angelo Oliveira, GitHub repository, zarganaut/API-call-sequences, 2019. , [Online]. Available: https://github.com/zarganaut/API-call-sequences.

[11] MalBehavD-V1 Dataset, GitHub repository, mpasco/MalBehavD-V1, 2020. , [Online]. Available: https://github.com/mpasco/MalBehavD-V1.

[12] C. Miller, T. Portlock, D. M. Nyaga, and J. M. O'Sullivan, "A Review of Model Evaluation Metrics for Machine Learning in Genetics and Genomics," Frontiers in Bioinformatics, vol. 4, p. 1457619, 2024.

[13] M. D. Rahman, SMOTE API Call Sequence Dataset, Kaggle repository, 2025. , [Online]. Available: https://www.kaggle.com/datasets/marahmanju/smote-api-call-sequence-dataset.

[14] L. Cardoso, V. Santos, J. Ribeiro, R. Kawasaki, R. Prudêncio, and R. Alves, "Enhancing Classifier Evaluation: A Fairer Benchmarking Strategy Based on Ability and Robustness," arXiv preprint arXiv:2504.09759, 2025.

[15] R. J. Kolaib and J. Waleed, "Crime Activity Detection in Surveillance Videos Based on Developed Deep Learning Approach," Diyala Journal of Engineering Sciences, vol. 17, no. 3, pp. 98-114, Sep. 2024.

[16] S. Beddar-Wiesing, A. Moallemy-Oureh, M. Kempkes, and J. M. Thomas, "Absolute Evaluation Measures for Machine Learning: A Survey," arXiv preprint arXiv:2507.03392, 2025.