Available online at www.qu.edu.iq/journalcm
JOURNAL OF AL-QADISIYAH FOR COMPUTER SCIENCE AND MATHEMATICS
ISSN:2521-3504(online) ISSN:2074-0204(print)

A Hybrid CNN–LSTM Framework for Behavioral Malware Detection and Dynamic YARA Rule Generation

Hamid Talib Zaidan[1], Jumana Waleed[2], Ruaa Azzah Suhail[3]

[1]*University of Diyala, Diyala, Iraq. hamidtalib@uodiyala.edu.iq*

[2]*University of Diyala, Diyala, Iraq. jumanawaleed@uodiyala.edu.iq*

[3]*University of Diyala, Diyala, Iraq. ruaaizzat@uodiyala.edu.iq*

ARTICLE INFO

Article history:
Received: 11 /02/2026
Rrevised form: 01 /04/2026
Accepted : 05 /04/2026
Available online: 30 /06/2026

Keywords:

Malware Detection

Cybersecurity

Deep Learning

CNN–LSTM

Dynamic YARA Rules

API Call Sequences

ABSTRACT

The rapid evolution of malware through the use of obfuscation techniques and continuous runtime behavior mutation has made traditional signature-based detection mechanisms much less effective, making there is a dire need for adaptive and deployable malware detection solutions. In response, behavioral analysis based on API call sequences has received more and more attention, especially with the use of deep learning models, such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTMs) networks. Although these models are shown to have a strong ability in modeling the sequential execution behavior, a lot of the existing approaches are limited to API level features and are not strongly linked to the practical detection tools used in real-world environments. This paper proposes a hybrid malware detection framework using CNN-LSTM-based behavioral modelling and contextual intelligence using the Hybrid Analysis platform. The system takes execution level API call sequences and augments them with light-weight external features such as threat score and antivirus detection counts to make the classification robust and reduce ambiguity in decision making. Furthermore, the learned behavioral patterns are then translated into dynamically generated YARA rules for interpretable and practical deployment, not limited to black box classification. The proposed framework is evaluated with a well-established academic data set created by combining the MalBehavD-V1 and Oliveira API call sequence datasets with 3500 samples. Experimental results show that the accuracy of hybrid CNN and LSTM reach 95.43% with only API sequences, and reach 97.49% when incorporating Hybrid Analysis features and combining the two sets of accuracy will be clearly improved, the discriminative effect will be improved as shown in the AUC metric. These results show that the fusion of deep learning-based behavioral analysis with external contextual intelligence is an effective and deployable malware detection solution which supports dynamic YARA rule generation.

MSC..

https://doi.org/10.29304/jqcsm.2026.18.22646

1. Introduction

The constant change in the malware, which is due to obfuscation techniques, polymorphism and the frequent mutation of the codes, has severely degraded the efficiency of the old signature-based and heuristic security

∗Corresponding author : *Hamid Talib Zaidan*

Email addresses: *hamidtalib@uodiyala.edu.iq*

Communicated by ‘sub etitor’

mechanisms. As contemporary malware behavior undergoes more modifications during execution, the reliability of static detection methods becomes limited, thus providing the need to develop more adaptive and behavioral aware cybersecurity solutions for real-world deployment [1]. In this context, behavioral malware analysis has become a promising alternative, in particular methods that model program execution in terms of the sequence of API calls. Recent development in the field of deep learning has only further reinforced in this direction where models like Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks have shown great capability in learning sequential patterns as well as learning the complex behavioral characteristics associated with malicious activities [2].

Despite their effectiveness, many systems used for malware detection based on deep learning are still restricted in their practical applicability. Several studies are based on computationally expensive preprocessing methods, such as images of executable files, that raise processing overhead, while losing important behavioral semantics. Other methods use complex architectures such as transformers-based models which are computationally expensive and take a long time to train, which makes them unsuitable for deployment in resource-constrained environments [3]. Furthermore, most deep learning models are black box systems and have limited interpretability and little integration with operational cybersecurity tools, which leads to low trust and low adoption in practice [3].

A review of the existing literature shows that a considerable part of the existing research addresses malware detection from a narrow perspective. Many works only apply the API call sequence modeling without introducing other external contextual information that would improve decision reliability and decrease ambiguity [1], [2]. On the other hand, rule-based detection mechanisms, e.g., YARA, are often manually built with static signatures; this restricts their adaptability to changes in the malware behavior and new forms of attacks [4], [5]. Although hybrid methods meeting the needs of both machine learning and rule-based detection have been approached, automatic and dynamic generation of YARA rules directly from the outputs of deep learning models has not been sufficiently studied [4].

To overcome these shortcomings, this paper proposes a hybrid malware detection framework based on CNN--LSTM-based behavior analysis and on dynamically generated YARA rules with the support of contextual intelligence provided by the Hybrid Analysis platform. The proposed system models call sequences of APIs to capture the runtime execution behavior and augments them with lightweight external features, called threat scores and antivirus detection counts, which have a complementary contextual understanding that also do not cause heavy computational overhead. By adding both behavioral and outside information, the framework improves classification robustness whilst keeping the architecture light in size and scalable enough for practical deployment.

The proposed approach is examined by using a well-established academic dataset created by combining the MalBehavD-V1 dataset with the Oliveira API call sequence dataset, consisting of 3,500 samples. Two experimental configurations are considered: an API-only configuration used to establish baseline behavioral performance, and an enriched configuration in which Hybrid Analysis features are incorporated to evaluate the effect of these features on detection accuracy and reliability. In addition, a series of optimization methods such as dropout, token-level dropout, batch normalization, L2 regularization, adaptive learning rate scheduling using AdamW and stratified data splitting are systematically used to enhance the training stability and prevent overfitting. The results of the trained deep learning model are then converted to dynamic and interpretable YARA rules to bridge the gap between high accuracy behavioral detection and deployable rule-based detection.

The major contributions of this work can be summarized as follows:

(i) a hybrid CNN-LSTM malware detection framework that used API call sequence modeling and external features from Hybrid Analysis, (ii) an empirical evaluation that showed the performance boost that came from the contextual feature integration, (iii) an automated pipeline for dynamic YARA rule generation based on the learned behavioral patterns, and (iv) a lightweight and efficient design that evades computationally expensive methods like image-based and transformers-heavy approaches while remaining suitable for real-world cybersecurity environments.

The rest of this paper is structured as follows. Related work on behavioral malware detection and hybrid approaches for malware detection is reviewed in Section 2. Section 3 explains the proposed methodology, from the construction of the dataset, processing of features and model architecture. Section 4 presents the experimental setup and results of the evaluation. The findings and their practical implications are discussed in Section 5. Finally, Section 6 concludes the paper and specifies the future research directions.

1. Related works

Deep learning-based malware detection has been a hot topic in the research community in recent years, especially with the growing trend of modern malware to incorporate obfuscation, polymorphism, and runtime behavior manipulation, which seriously compromises the security of malware detection through traditional signature-based and heuristic methods [1], [2]. As a result, behavioral analysis through dynamic representations such as API call sequences has become an effective alternative, allowing models to capture execution level characteristics which are hard to evade. Among behavioral methods, hybrid architectures based on the Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks have demonstrated a good performance in modeling the sequential behavior of malware. Karat et al. [6] suggested CNN-LSTM-based framework for malware detection based on analysis of API call sequences obtained by dynamic monitoring tools. Their model resulted in an accuracy of about 96% which shows the power of combining convolution feature extraction with temporal modelling. However, the research was based on a relatively small dataset consisting of around 2,500 samples and was GPU-intensive to train, and its reliance on specific monitoring tools limited its reproducibility and large-scale deployment. Li et al. [7] proposed in-depth behavioral malware detection framework by combining the 1D-CNN-based representation of APIs with the semantic chain modeling and BiLSTM networks. The accuracy was 97.31% and F1-score was 0.97 on a dataset containing about 43,000 samples obtained using Cuckoo Sandbox. Despite these results, the model demonstrated significant degradation in performance on new malware samples, indicating that the model is sensitive to concept drift. In addition, the complexity of semantic feature engineering was reducing the scalability and posing challenges for practical deployments. Other research delved into deeper and more intricate architectures of deep learning. Maniriho et al. [1] proposed an early detection system for malware by analyzing short prefixes of API call sequences using a combination of GPT-2 based sequence prediction and DistilBert-BiGRU classifiers. Although the framework was able to achieve an accuracy close to 96% and was effective for early-stage detection, the use of large language models made computational cost a significant factor, as well as the fact that long-term behavioral dependencies could not be modeled very well. Similarly, CAFTrans [8] is a transformer-based malware detection system that uses API call frequency and channel-level features, the approach gained only moderate performance (F1-score = 0.65; AUC = 0.79), mostly because of poor temporal modeling and the use of an old dataset. Beyond behavioral sequence modeling, there have been works in which alternative representations of data are explored. The authors of the paper Alshomrani et al. [3] introduced the explainable hybrid model using ConvNeXt and Swin Transformer architectures to perform visual malware classification with image representations of binaries. Although the model achieved accuracy values of between 94% and 98% with various benchmarking datasets, the conversion of binaries to images imposed a large preprocessing overhead, while the transformer-based architecture imposed a high computational cost, restricting the use case to lightweight behavioral analysis contexts. In parallel with classification-based approaches automated generation of YARA rules has received attention as a way of improving rule-based malware detection systems. Coscia et al. [4] presented APIARY, an API-driven framework for the automatic generation of YARA rules, based on the identification of discriminative API patterns from static and dynamic analysis data, APIARY achieved strong performance in detection with best accuracy close to 97.9%. However, the approach uses a statistical feature selection and rule optimization primarily and does not use deep learning-based temporal modeling, which limits the ability to capture complex behavioral dependencies. Similarly, Zhang et al. [5] proposed RULELLM, an approach for automatically generating YARA and Semgrep rules from source code and metadata using large language models. While the results were promising, the approach has high computational overhead, a risk of hallucinations, and was primarily focused on static analysis and not dynamic API sequence modeling. A broader outlook on the subject of deep learning-based malware detection was presented by Song et al. [2], which is a thorough survey of 363 publications. Their analysis revealed more persistent difficulties, such as the lack of standardized datasets, poor cross-domain generalization, and poor model interpretability. However, as a survey work, no concrete solution at the system level and no framework for deployment were proposed. A consolidated comparison of most representative and relevant studies discussed above highlights the fact that existing CNN--LSTM based approaches exhibit strong capability in modeling the call sequence of API but often suffer from limitations in data sets, high computational cost, or poor combination with practical detection mechanisms. On the other hand, automated YARA-based systems do offer deployable rule-based detection, but usually lack deep learning-based temporal modeling of behavioral data. In contrast to previous projects, the new proposed framework combines CNN-LSTM-based behavioral modeling of API call sequences with contextual intelligence gathered from the Hybrid Analysis platform and the new concept of an automated pipeline of dynamic YARA rule generation from the learned behavioral patterns. This unified design bridges the gap between high accuracy deep learning-based behavioral analysis and practical and interpretable and deployable YARA-based malware detection.

2. Proposed hybrid cybersecurity system

The system architecture of the proposed hybrid cybersecurity system is shown in Figure 1. The system is designed as a modular and flexible system that combines deep learning based behavioral analysis with dynamic YARA rule generation to deliver accurate and deployable malware detection. Its design supports several experimental configurations with a clear separation of behavioral feature learning and the integration of contextual intelligence.

The workflow starts with the selection of the dataset depending on the objective of the experiment. In this work, the use of two closely related datasets is made. Dataset 1 is a small-scale academic benchmark dataset that is built by combining the MalBehavD-V1 dataset [9] with the API call sequence dataset proposed by Oliveira [10]. This dataset is built with 3500 samples with a nearly equal distribution of malware and benign software. It contains just behavioural API call sequences and is the baseline dataset for testing the effectiveness of deep learning models with API-based behavioural features only. Dataset 2 is directly based on Dataset 1 by enhancing the same samples with other contextual attributes obtained through the Hybrid Analysis platform. These attributes are threat-related scores, antivirus detection scores, verdict information, file type, and file size. Dataset 2 is specifically designed to evaluate the effect of combining external behavioral intelligence and API sequence modeling in a unified deep learning framework.

Following the selection of a dataset, the system moves to the data preprocessing stage which involves making the inputs ready for deep learning-based analysis. This stage includes the conversion of sequence of API calls into numerical form, normalizing of sequence length through padding or truncating, and handling of missing values. For Dataset 2, external Hybrid Analysis features are further normalized so that they are compatible with the learning process. The preprocessing strategy is dataset-specific and is explained in detail in Subsection 3.2.

The learning phase is based on a hybrid deep learning architecture based on Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. For Dataset 1, a CNN-LSTM model is used to learn both the local patterns of API interactions, as well as long-range temporal dependencies in the program execution behavior. For Dataset 2, a CNN--LSTM fusion architecture is used, such that behavioral API sequences and external Hybrid Analysis features are handled as parallel branches and merged at some later point. This design allows this model to learn the intrinsic execution behavior and the contextual intelligence simultaneously without introducing too much computational overhead.

In the final stage the trained models are evaluated on independent test sets which ensures reliable and unbiased performance assessment. Standard evaluation metrics are used, such as Accuracy, Precision, Recall, F1-score and Area Under the Receiver Operating Characteristic Curve (AUC). Furthermore, the superior model trained based on the Hybrid Analysis - enriched dataset is used to produce dynamic YARA rules. These rules are evaluated individually to prove that the learned behavioral knowledge can be turned into interpretable and practical deployable detection signatures. The close correlation between detection performance based on deep learning and the effectiveness of YARA rules emphasizes the feasibility of the proposed hybrid cybersecurity system.

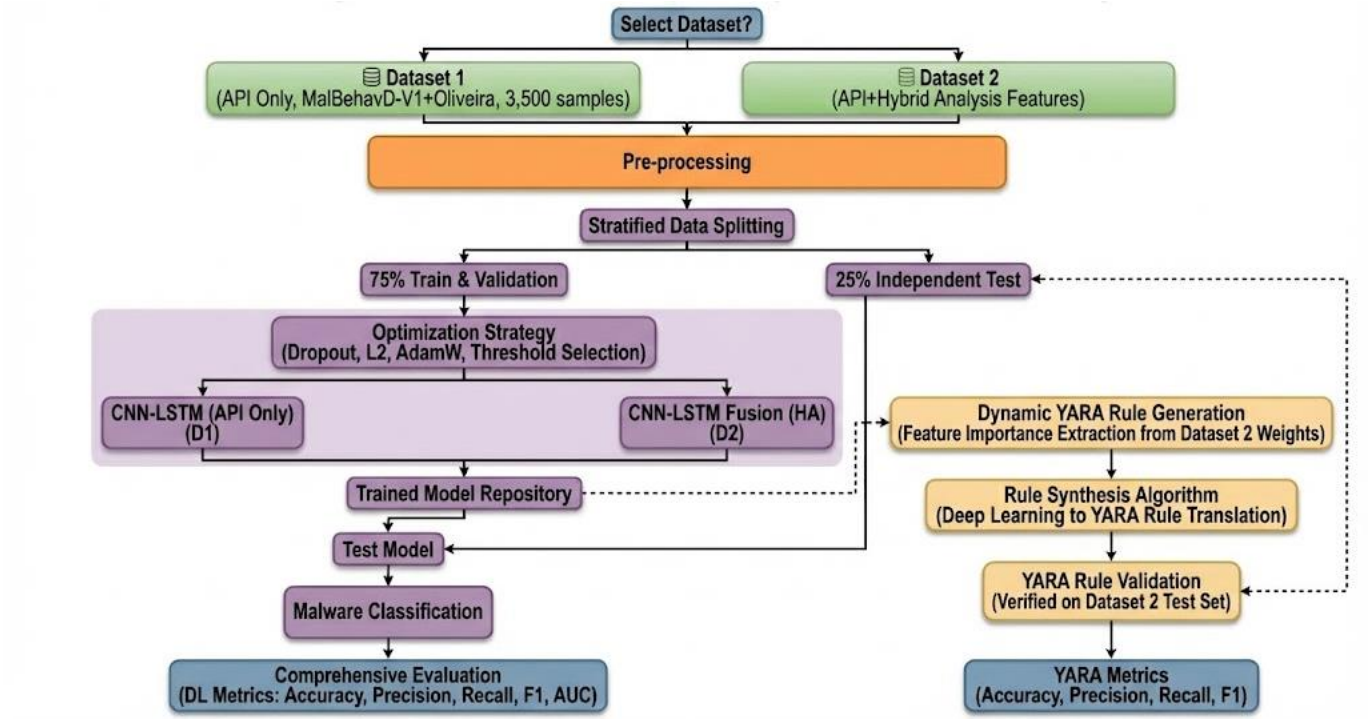

Fig. 1 - Overall framework of the proposed hybrid malware detection system integrating deep learning and dynamic YARA rules.

2.1. Utilized datasets

To evaluate the proposed hybrid malware detection framework, two datasets are close to each other are utilized in this study. These datasets aim to evaluate behavioral malware detection based on API call sequences and analyze the effect of incorporating external contextual intelligence acquired from the Hybrid Analysis platform in a unified deep learning architecture. Both data sets have the same core executable samples, which allows to control and fairly compare API-only modeling and enriched behavioral analysis.

Dataset 1 is used as the baseline behavioral benchmark and is built by combining two popular academic corpora used in research on malware detection, namely MalBehavD-V1 dataset [9] and the API call sequence dataset proposed by Oliveira [10]. The resultant merged data set contains a total of 3500 executable samples, 1762 samples of malware and 1738 samples of benign programs. Specifically, 2474 samples are taken from the MalBehavD-V1 dataset and 1026 samples are taken from Oliveira's dataset.

Each sample in Dataset 1 is only represented by API call sequences extracted from dynamic execution traces that represent a consistent and interpretable behavioral profile of software execution. No external contextual or threat intelligence features are included in this dataset, so it can be used as a baseline for the evaluation of deep learning models based only on the behavioral information.

Since Dataset 2 is an enriched version of Dataset 1, where the composition of samples remains the same, both datasets have the same class distribution. The distribution of malware and benign samples for Dataset 1 and the enriched version, Dataset 2, is shown in Figure 2.

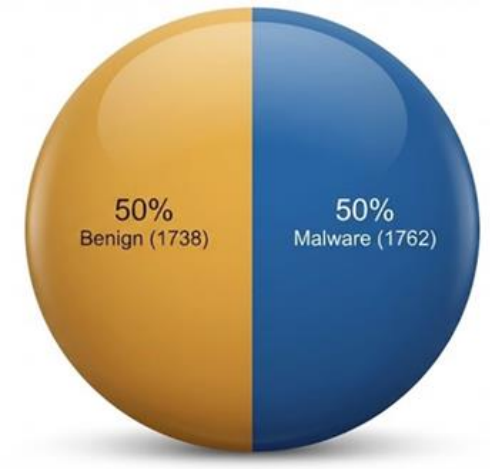

Fig. 2 - Class distribution of Dataset 1 and its enriched variant (Dataset 2 with Hybrid Analysis features), showing the balanced proportion of malware and benign samples.

The feature representation that is followed for Dataset 1 is based on sequential API call tokens taken from dynamic execution traces. This representation maintains the time order of the API invocations, and allows the learning models to register the execution-level behavioral patterns. The structural composition of these API calls sequence is shown on the Figure 3.

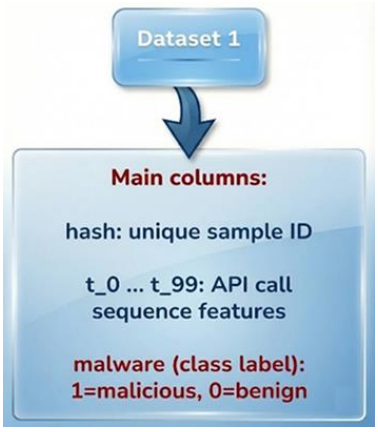

Fig. 3 - Feature structure of Dataset 1 based on API call sequences extracted from dynamic execution traces.

Dataset 2 is directly based on Dataset 1 by enriching the same executable samples with external contextual features retrieved from the Hybrid Analysis system. Along with the API call sequences, Dataset 2 includes sandbox-based attributes like threat scores, antivirus detection counts, verdict information, file type, and file size. This dataset is meant to test the impact of combining external behavioral intelligence with API sequence modeling in the context of a fusion-based CNN-LSTM architecture.

By fusing the results of intrinsic execution behavior with the analysis results of the context, Dataset 2 offers a richer and more informative representation of malware behavior while retaining the temporal structure of API call sequences. This enrichment means the proposed enrichment framework can jointly use behavioral and contextual cues without adding too much preprocessing complexity. The enriched feature structure and its combination with Call sequence in API are shown in Figure 4.

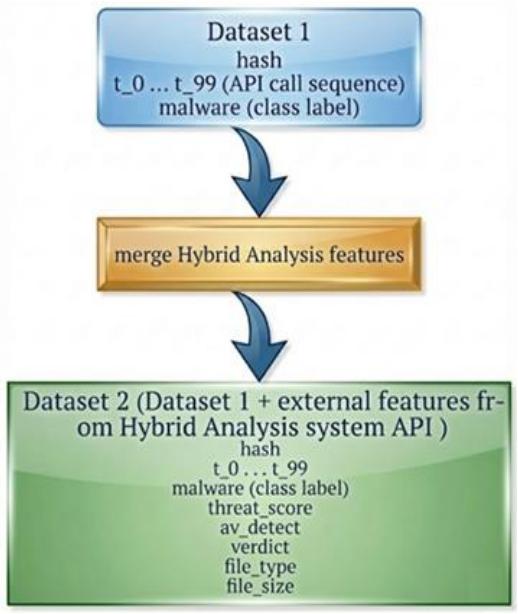

Fig. 4 - Structural representation of Dataset 2 combining API call sequences with Hybrid Analysis contextual features.

The dataset design that is adopted allows for a focused and systematic evaluation of the proposed hybrid system. Dataset 1 is used to evaluate how effective behavioral malware detection based on API call sequences only is, whereas dataset 2 checks the contribution of Hybrid Analysis contextual features and supports dynamic YARA rule generation in the proposed framework.

2.2. Data pre-processing

Effective data preprocessing is an important step to ensure a reliable training of deep learning models and to guarantee a leakage-free evaluation. Given the diversity in the composition of features between the datasets that were utilized, a customized but common preprocessing strategy was implemented. The overall objective was to maintain the semantics of behavior, impose uniform representations of inputs and strictly prevent information leakage from training, validation, to testing phases.

For Dataset 1 (API-only, 3,500 samples), API call sequences obtained from dynamic execution traces have been represented using a mixed encoding strategy. Textual API names were converted to unique integers; numeric values that originally appeared in the sequences were not changed. Calls to rare or previously unseen API calls were assigned a special unknown token (UNK = 1). Since API sequences varied in length, all samples were made into a fixed length of 100 API calls. Sequences of shorter length were first padded with a placeholder value (minus 1) to indicate position values and later replaced with the padding token (PAD = 0). Sequences with more than 100 API calls were truncated as first 100 API calls. This procedure guaranteed that a homogeneous and sequential representation is guaranteed for batch-based deep learning training.

For Dataset 2 (API + Hybrid Analysis features), the same API sequence preprocessing steps applied for Dataset 1 were kept in order to keep comparability. In addition, a set of external contextual features, retrieved by calling the Hybrid Analysis API, was added. Numeric attributes, like threat scores, antivirus detection number, file size, etc., were normalized by a two-stage process. First, a logarithmic scaling $\log(1 + x)$ was used in order to reduce skewness for highly variable features. Second, z-score standardization was applied in each feature to obtain the zero mean and unit variance for each feature.

Categorical attributes, such as verdict information and type of file, were encoded with a Top-K one hot encoding scheme. The 20 most frequent categories were explicitly represented while all other categories were grouped under a single "OTHER" category in order to control dimensionality. Missing values in the external features were imputed from median values calculated from the training split only to avoid data leakage. To retain missingness information,

binary indicator variables for each external feature were appended in order to allow the model to discriminate between observed and imputed values.

After preprocessing, all the datasets were partitioned using stratified sampling to preserve the original malware - benign class distribution. A total of 75% of the data was dedicated to training and validation, and 25% of the data was set aside as a separate testing dataset. From the training part an internal validation set of 20% was further separated which was used for monitoring convergence and decision threshold optimization. This hierarchical splitting strategy ensured a robust evaluation, and prevention of information leakage in a strict manner.

The entire preprocessing workflow that was followed for the proposed system is summarized in Figure 5, while a comparative view of preprocessing steps involved for the used datasets is presented in Table 1.

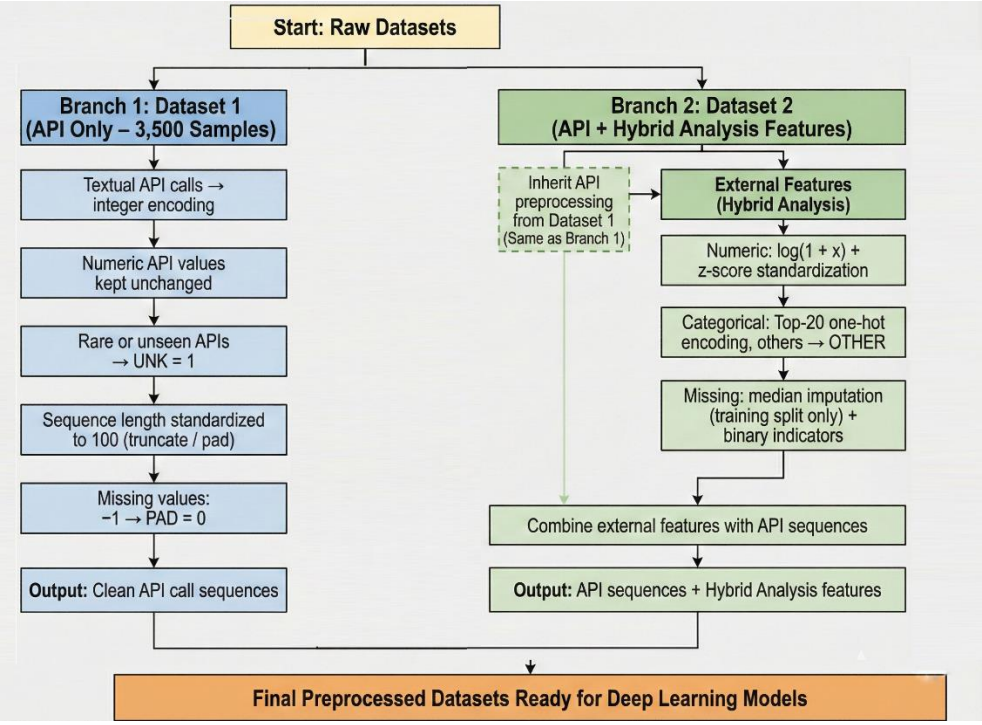

Fig. 5 - Overview of the data preprocessing workflow for Dataset 1 (API-only) and Dataset 2 (API sequences enriched with Hybrid Analysis contextual features).

Table 1 - Summary of data preprocessing steps across the utilized datasets

Dataset	API Sequence Processing	External Feature Processing	Missing Value Handling	Notes
Dataset 1	Textual API names mapped to integer IDs; numeric values retained; rare or unseen APIs mapped to UNK = 1; sequences standardized to length 100 using truncation and padding (PAD = 0).	Not included	Missing API positions initially marked as −1 and replaced with PAD = 0.	Behavioral API features only (baseline dataset).
Dataset 2	Same API sequence preprocessing as Dataset 1 to ensure comparability.	Hybrid Analysis features: numeric attributes normalized using log(1 + x) followed by z-score standardization; categorical attributes encoded using Top-20 one-hot encoding (others grouped as OTHER); missingness indicators added.	Median imputation applied to external features using training split only; binary missing-value indicators included.	Combines behavioral API sequences with Hybrid Analysis contextual intelligence.

2.3. API call features and representation

Windows API call sequences are used as primary behavioral representation in this work, because they allow one to have direct insight into the runtime behavior of executable programs without access to their source codes. API calls record security-related operations such as file operations, registry access, process manipulation, and network communication, so they are an effective indicator of malicious activity.

As shown in Figure 6, the calls of an API are obtained by dynamic execution in controlled environments and converted into a uniform representation. Textual API names are transformed into unique integer identifiers and calls not seen are assigned an unknown token (UNK = 1). To keep them consistent across the samples, all sequences are fixed to a length of 100 API calls using truncation and padding, where missing positions are represented by a padding token (PAD = 0).

This integer encoded sequential representation helps to maintain the order of API invocations and is also suitable for sequential deep learning models. Compared to static features, the API call sequences are more robust against common obfuscation techniques and offer a small but expressive characterization of program behavior.

Within the proposed framework, the call sequence of API calls describes the behavioral basis for two data sets (Dataset 1 and Dataset 2). Dataset 1 uses only API sequences while dataset 2 maintains the same representation and adds contextual features derived from Hybrid Analysis which allows for enriched behavioral modeling without changing the structure of the sequence itself.

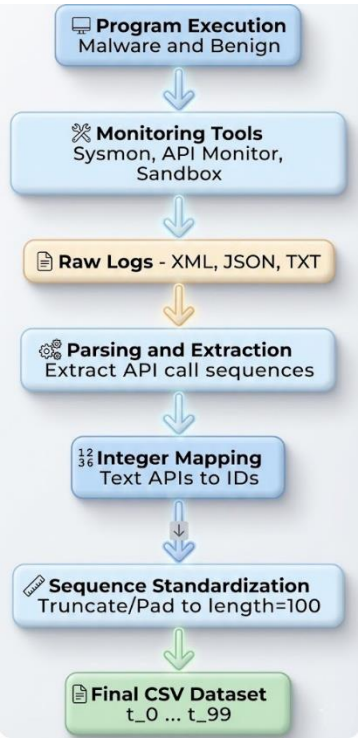

Fig. 6 - API call sequence extraction and representation pipeline used in the proposed framework.

2.4. Hybrid CNN–LSTM architecture

To successfully model both local and long-range dependencies of the sequences of API calls, the proposed system has a hybrid deep learning architecture, which merges Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. CNN layers are used to learn local interaction patterns between API calls in temporal sequence and LSTM to model temporal dependencies in the execution trace. This combination allows for accurate modeling of behavior without compromising the scalability and practicality of the model.

3.4.1 Hybrid CNN–LSTM for API-only datasets (dataset 1)

For Dataset 1, which is based solely on the sequences of API calls, samples are represented as fixed-length sequences of 100 integer-encoded API identifiers. Shorter sequences are padded with a special padding token (PAD = 0) and rarely or unseen API calls are converted to an unknown token (UNK = 1). The model processes these sequences through an embedding layer that converts discrete tokens of API that are used to calculate the vector representation into dense vectors while masking the padded positions.

The embedded sequences are then fed through stacked one-dimensional convolutional ones to extract local patterns in the behavior, followed by LSTM layers to extract long-term temporal relationships from the execution trace. The obtained sequence-level representation is then refined with fully connected layers with dropout and batch normalization after which a sigmoid activated output layer generates the final probability of malware classification. The architectural setup of this CNN-LSTM model in summarized in Table 2 and shown in Figure 7.

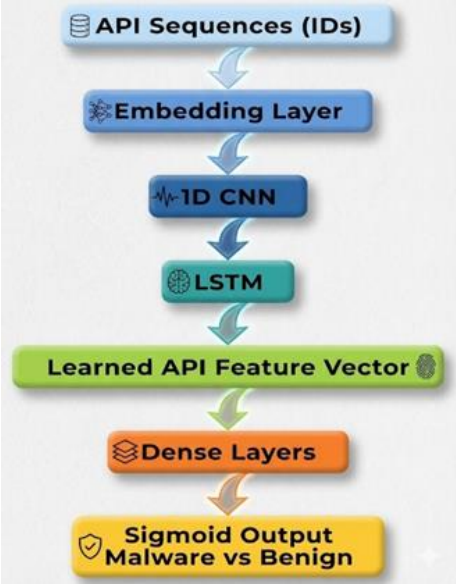

Fig. 7 - CNN–LSTM architecture for API-only dataset (dataset 1).

Table 2 - Configuration details of the CNN–LSTM model for API-only datasets (dataset 1).

Layer (Type)	*Output Shape*	*Param #*	*Corresponding Block (Aligned with Figure 7)*
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 feature fusion (dataset 2)

For Dataset 2, where the API call sequences are augmented with some contextual features gathered from Hybrid Analysis, the architecture is extended based on a feature fusion strategy. As shown in Figure 8, there are two parallel branches in the model. The first branch processes API call sequences with the same CNN - LSTM pipeline described above and the second branch processes normalized Hybrid Analysis features with fully connected layers with dropout and batch normalization.

The learned representations from both branches are combined using feature-level concatenation and passed to subsequent dense layers to get the final classification output. This fusion-based design enables the model to incorporate intrinsic behavioral patterns and external contextual intelligence jointly, without having to change the temporal structure of API sequences. The full setup of the fusion model is summarized in Table 3.

Overall, the proposed hybrid CNN-LSTM architecture offers a balanced trade-off between the modeling capacity and the computational efficiency, which can efficiently detect the malware based on the behavior API sequence alone and the enriched contextual information if it is available.

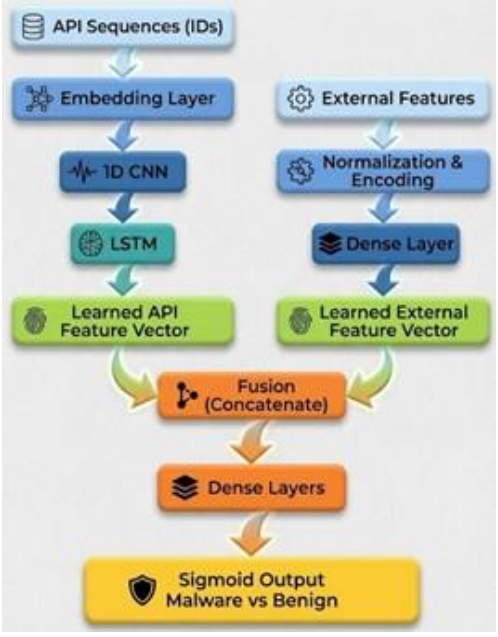

Fig. 8 - Hybrid CNN–LSTM architecture with fusion of API-sequence and external-feature branches (dataset 2).

Table 3 - Configuration details of the CNN–LSTM fusion model (dataset 2).

Layer (Type)	*Output Shape*	*Param*	*Corresponding Block (from Figure 8)*
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)			

2.5. Training hyperparameters

In order to obtain a fair and controlled comparison of the evaluated models, a unified training configuration has been adopted over all experiments. The same optimization strategy and training settings were used for both the API-only CNN- LSTM model (Dataset 1) and Hybrid CNN- LSTM fusion model (Dataset 2) in order to allow performance differences to be attributed to single inclusion of contextual features from outside the signal representation rather than the result of difference in training parameters.

All the models were trained with the AdamW optimizer in combination with the cosine learning rate scheduling strategy Cosine Decay Restarts, which ensures stable convergence and better generalization. A fixed size of 512 was used for all the experiments to balance between the computation efficiency and the memory usage. Binary Cross-Entropy loss function was used for the loss function since the malware detection problem is a binary problem.

In order to mitigate overfitting, dropout was used at several different levels of the network, such as the embedding layers, LSTM layers, and fully connected layers. In particular, recurrent dropout was used within the LSTMs parts with dropout rates varying between 0.30 and 0.35, in order to provide further regularization without impairing the convergence process.

For separation of data, stratified sampling has been applied to maintain the distribution of malware and benign classes. Each data set was divided with 75% for training and testing, and 25% of the data set was for independent testing. From the training portion, 20% was additionally set aside as an internal validation set for monitoring the convergence and picking decision thresholds. Alternative split ratios were investigated during the preliminary experiments but did not provide better stability, and the configuration adopted always ensured evaluation without leakage.

A summary for the training hyperparameters used for both model variants are presented in Table 4, which demonstrates that the same training settings were kept across the API-only and hybrid architectures.

Table 4 - Training hyperparameters for the CNN–LSTM and Hybrid CNN–LSTM models.

Parameter	*CNN–LSTM(dataset 1)*	*Hybrid CNN–LSTM (dataset 2)*
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)
Random seed	*42*	*42*

2.6. Dynamic YARA rule generation

While deep learning models are very accurate at achieving high detection accuracy, they are considered a black box and hard to interpret, which prevents their direct application in an operational cybersecurity environment. To overcome this limitation, the proposed framework includes a dynamic YARA rule generation mechanism that is able to translate learned behavioral patterns to interpretable and deployable rule based signatures.

Dynamic YARA rule generation is done using only the best-performing model, Hybrid CNN-LSTM fusion model trained on Dataset 2 with a combination of API call sequence and contextual features extracted from Hybrid Analysis. Rule synthesis is based only on the trained model parameters and feature importance signals that are taken out of the model in the training process and the independent test split is strictly kept for the validation process to avoid information leakage.

The rule generation process starts by finding salient behavioral indicators that the model has learned. From the branch of API sequence discriminative API call patterns and their positional relevance in the execution traces are derived based on learned activations and weight magnitudes. In parallel, information from the Hybrid Analysis branch including high impact contextual features (e.g. threat-related scores, detection metadata) is analyzed in order to quantify its contribution to the classification decision.

Based on the extracted feature importance, dynamic YARA rules are built up by mapping the high-confidence behavioral and contextual indicators into rule conditions. API based patterns are encoded in the form of ordered strings or hexadecimal strings, while contextual information is added as numerical or logical constraints to the YARA rule logic. Thresholds for these conditions are empirically determined based on training statistics to achieve a balance between detection sensitivity and false positive rates.

The generated YARA rules are then validated on the independent test split of the Dataset 2, which will allow for an objective evaluation of the detection performance of the rule-based methods, outside of the deep learning inference pipeline. Evaluation metrics are accuracy, precision, recall, and confusion matrix analysis, making it possible to directly compare the results of neural network-based classification with the results of rule-based detection. This process shows that knowledge obtained by deep learning models can be successfully converted into interpretable and operational security rules.

Overall, the dynamic YARA rule generation component is a gap between high-accuracy behavioral modeling and practical malware detection that allows creating rules in an automated, interpretable and deployable way within existing security infrastructures.

3. Experimental results

This section consists of experimental evaluation of the proposed hybrid malware detection framework using two datasets with different feature configurations. The experiments are aimed at evaluating the performance of detection and analyzing the effect of combining contextual intelligence with behavioral API-based analysis.

3.1. Performance Measures

The performance of the proposed models for malware detection has been assessed using common classification metrics used in cybersecurity studies. These are Accuracy and Area Under the ROC Curve (AUC) as the main metric to evaluate the effectiveness of classification [11] and Precision, Recall, F1-score, and Specificity to analyse the false-positive and false-negative behavior [12]. Together, these metrics give a comprehensive and reliable assessment of the performance of malware detection in different experimental settings [13-16].

3.2. Experimental setup

All the experiments were implemented in Python using TensorFlow 2.x and the standard scientific libraries and the hardware has memory of 8 GB and CPU of core i5-gen13. Stratified sampling has been used to maintain the malware-to-benign class distribution. Each dataset was divided into 75% training and validation and 25% independent testing data. An internal validation subset was used to track the convergence and to identify optimal decision thresholds. Identical training configurations were used in all models to provide fair and consistent comparison.

3.3. Results of the proposed system

The evaluation compares three model configurations, namely standalone CNN, standalone LSTM and the proposed model with CNN-LSTM architecture.

On Dataset 1 (API-only) the CNN and LSTM models (baseline models) gave reasonable detection performance, although higher misclassification rates were seen for complex API call sequences. On the contrary, the hybrid CNN-LSTM model can always outperform the base models by capturing the local API interaction patterns as well as the long-term temporal dependencies. This improvement is verified by the confusion matrix analysis presented in Figure 9 which shows the reduction of both false positives and false negatives.

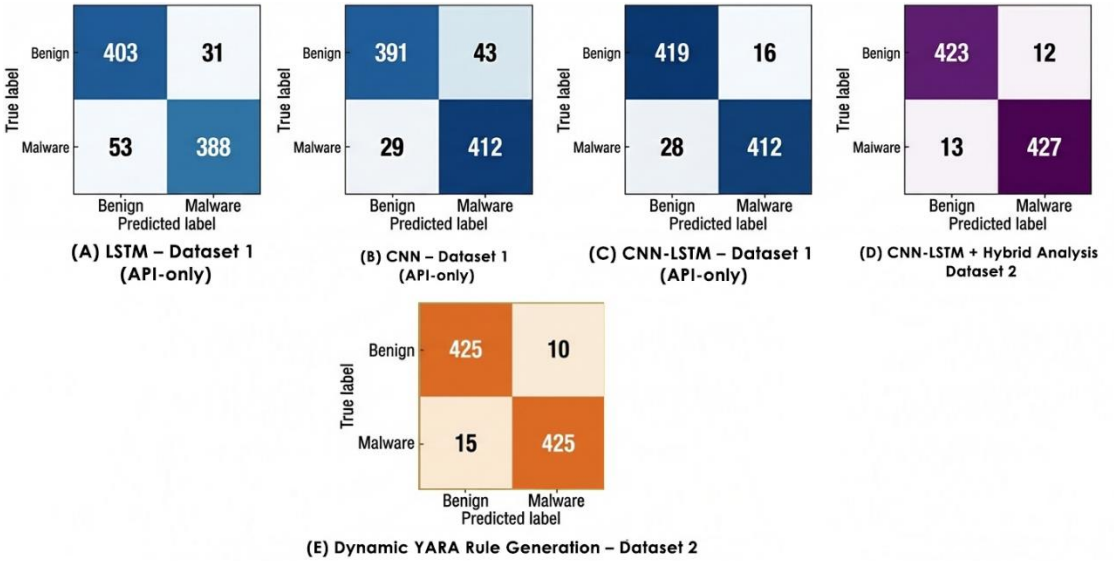

Fig. 9 - Confusion matrices of the evaluated models across the datasets.

When additional features from Hybrid Analysis were added to Dataset 2, even more performance improvements were noted. In this case, the hybrid model showed greater Accuracy and AUC than the API-only configuration, proving the complementary nature of contextual intelligence for behavioural representations to increase discriminative capability. The comparison between the ROC curves in Figure 10 shows this improvement in separability.

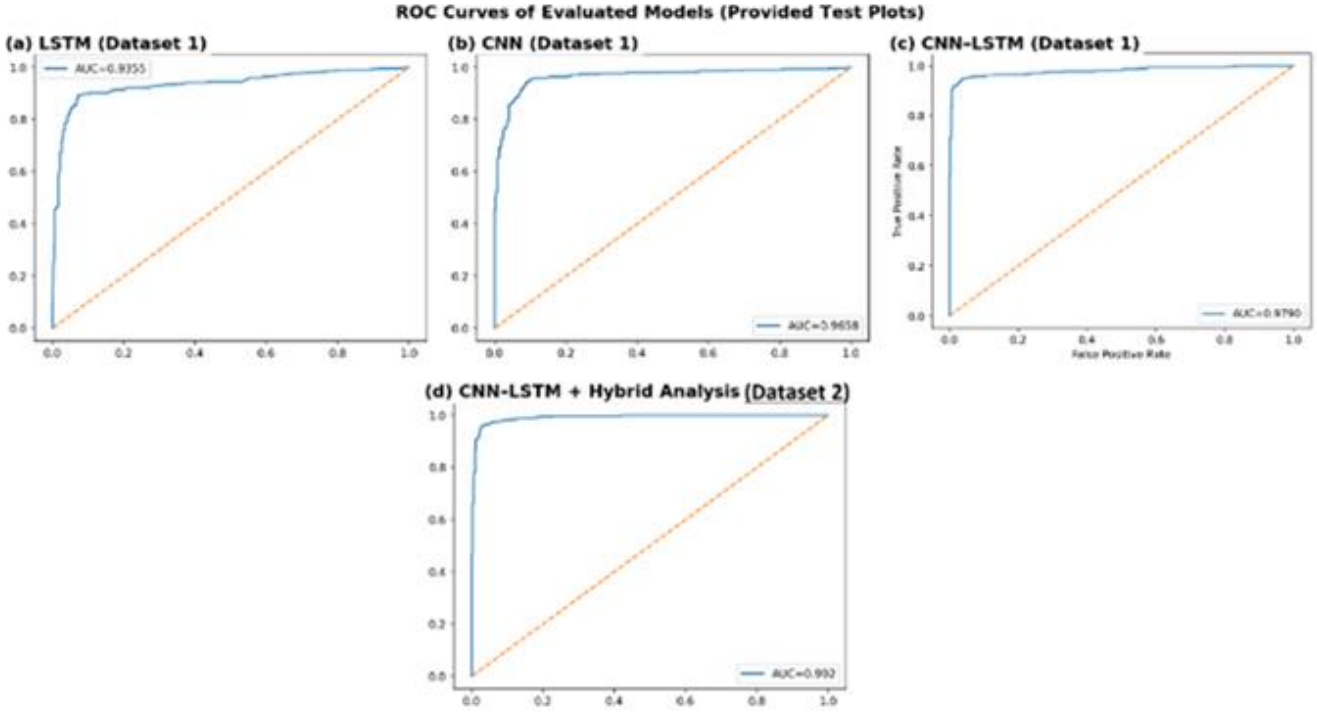

Fig. 10 - ROC curves comparing the discriminative performance of (a) LSTM, (b) CNN, and (c) CNN–LSTM on dataset 1, (d) CNN–LSTM with Hybrid Analysis on dataset 2.

A quantitative summary of validation and test results are given in Table 5 proving the robustness and stability of the proposed hybrid architecture in all evaluation metrics. A total comparison of trends in performance is aggregated in figure 11.

Table 5 - Performance comparison of baseline, hybrid, and generalized models across all datasets.

Model / Configuration	*Validation Acc.*	*Test Acc.*	*Precision*	*Recall*	*F1-score*	*Specificity*	*AUC*	*Decision Threshold*
LSTM (Dataset 1, API-only)	0.9048	0.9017	0.9030	0.9019	0.9017	0.9309	0.9355	0.50
CNN (Dataset 1, API-only)	0.9143	0.9246	0.9248	0.9245	0.9245	0.9147	0.9658	0.50
CNN–LSTM (Dataset 1, API-only)	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
Dynamic YARA Rules (Dataset 2)	—	0.9714	0.9770	0.9659	0.9714	0.9770	—	—

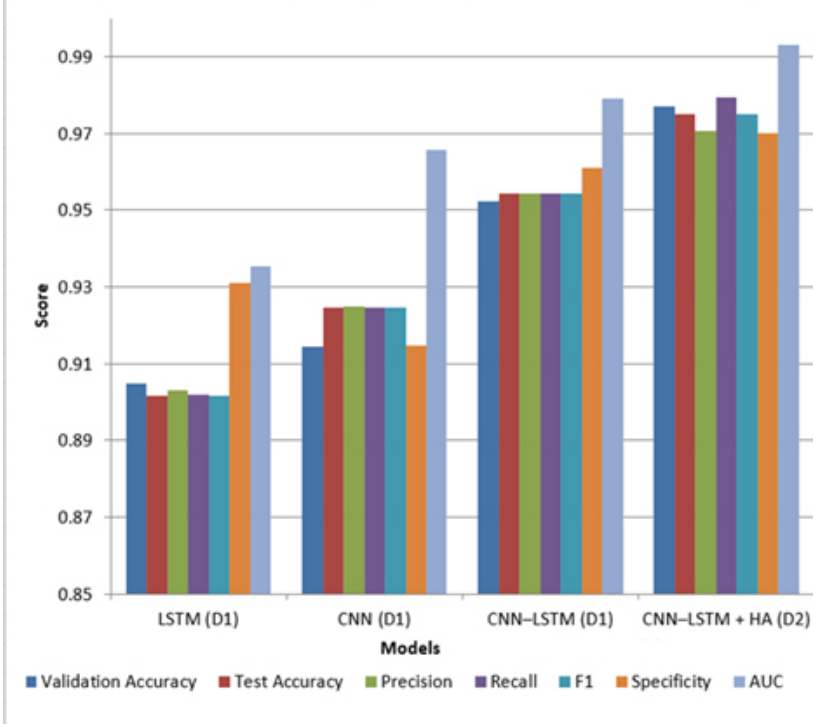

Fig. 11 - Comparative performance metrics of the evaluated models across all datasets.

Radar charts are applied to intuitively compare model performance across various evaluation metrics such as Accuracy, Precision, Recall, F1-score, Specificity and AUC. As shown in Figure 12 for Dataset 1, the baseline CNN and LSTM models have a moderate and unbalanced performance profile while the hybrid CNN-LSTM model has a more uniform and improved metric distribution.

For Dataset 2, which is represented in Figure 13, the addition of Hybrid Analysis features leads to a noticeable amount of radar area expansion, which indicates consistent performance expansion on all metrics. These visual comparisons validate that the proposed hybrid architecture is not only able to improve individual performance measures, but those hybrid architectures exhibit more stable and balanced detection behavior when incorporating contextual intelligence.

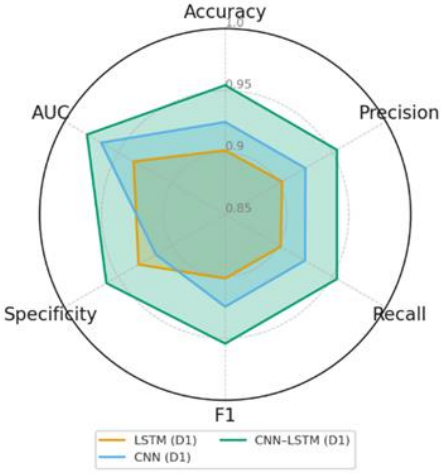

Fig. 12 - Radar chart of models evaluated on dataset 1.

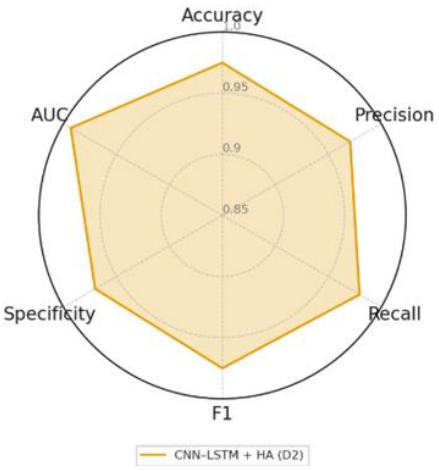

Fig. 13 - Radar chart of models evaluated on dataset 2 (Hybrid Analysis features).

In order to evaluate practicality of the deployability, dynamic YARA rules were created based on the best-performing hybrid CNN-LSTM model that was trained on Dataset 2. The generated rules were tested on the independent test split and showed good detection accuracy with a slight performance decrease when compared to the deep learning classifier. As shown in Figure 14, the rule-based detection kept strong Precision and Recall, which prove that learned behavioral patterns can be effectively translated into interpretable and deployable YARA rules.

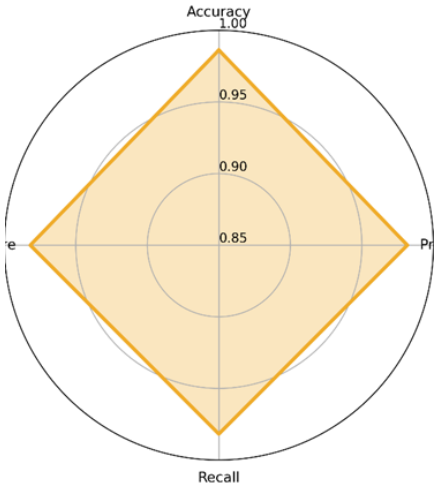

Fig. 14 - Radar chart illustrating the performance of the dynamically generated YARA rules on dataset 2.

4. Discussion

The experimental results show a clear and systematic gain in the detection performance as the proposed framework continually progresses from standalone models to a hybrid architecture to the incorporation of external contextual intelligence. This progression emphasizes the individual and aggregate contribution of each system component to effective behavioral malware detection.

For the API-only configuration (Dataset 1) it can be seen that the standalone CNN and LSTM models have complementary strengths and weaknesses. CNN-based modeling has an effective representation of local API interaction patterns but is limited to model long-term execution behavior and LSTM-based modeling has an effective representation of temporal dependencies and less discrimination in isolation through modeling. The hybrid CNN-LSTM architecture solves these limitations to jointly learn the local and long-range behavioral patterns that will lead to more stable and balanced detection performance.

The fact that Hybrid Analysis contextual features are integrated into Dataset 2 adds more reliability to detection. Complementary information derived from such external intelligence sources as threat-related scores and metadata from sandboxes help to solve ambiguous cases that are not fully accounted for by behavioral API sequences alone. This fusion results in a steady improvement in both false positives and false negatives which proves the worth of context enrichment for the real world malware detection scenarios.

In addition to classification performance, the proposed framework is shown to be applicable in practice with the dynamic generation of YARA rules. By translating learned behavioral and contextual patterns into interpretable rule-based signatures, the system bridges the gap between high-accuracy deep learning models and deployable security mechanisms. The resulting rules maintain strong detection capability while offering transparency and ease of integration into existing security infrastructures.

Compared with representative studies summarized in Table 6, the proposed framework achieves competitive or superior detection performance while maintaining lower computational complexity. Unlike image-based or transformer-heavy approaches, the use of integer-encoded API sequences and a lightweight hybrid architecture enables efficient training and deployment without extensive preprocessing overhead.

Overall, the discussion validates the fact that combining behavioral API modeling with contextual intelligence and rule-based deployment is an effective, interpretable and operationally feasible solution for modern malware detection.

Table 6 - Comparative summary of representative studies and the proposed system.

Authors, Year	Approach / Model	Data Representation	Dataset(s)	Key Results	Main Limitations
Karat et al. [6], 2024	CNN–LSTM	API call sequences	Rohitab + Sysmon (~2,500 samples)	Accuracy ≈ 96%	Small dataset, GPU-intensive training, tool-dependent
Maniriho et al. [1], 2024	GPT-2 + DistilBERT + BiGRU	Early API sequences (20–50 calls)	Sandbox execution traces	Accuracy ≈ 96%	High computational cost, limited long-term behavioral modeling
Qian & Cong [8], 2024	Transformer (CAFTrans)	API call frequency and channel features	Mal-API-2019	F1 = 0.65, AUC = 0.79	Weak temporal modeling, outdated dataset
Li et al. [7], 2022	1D-CNN + Semantic Chain + BiLSTM	API semantic representations	Cuckoo Sandbox (~43k samples)	Accuracy = 97.31%, F1 = 0.97	Concept drift, complex feature engineering, scalability issues
Alshomrani et al. [3], 2025	ConvNeXt + Swin Transformer	Malware binary images	Malimg, MaleVis, VirusMNIST	Accuracy = 94–98%	Heavy preprocessing, high computational cost
Song et al. [2], 2025	Survey (363 studies)	—	Multiple datasets	Identified key research gaps	No executable or deployable system
Coscia et al. [4], 2025	APIARY (Automatic YARA Generation)	Static and dynamic API patterns	Multiple malware datasets	Best accuracy ≈ 97.9%	No deep learning, limited temporal modeling
Zhang et al. [5], 2025	RULELLM (LLM-based YARA)	Source code and metadata	3,200 OSS malware packages	Precision = 85.2%, Recall = 91.8%, F1 = 88.4%	High computational cost, hallucination risk
Proposed System (Dataset 1)	CNN–LSTM (Conv1D + LSTM, integer encoding, dropout)	API call sequences	MalBehavD-V1 + Oliveira (API-only, 3,500 samples)	Acc = 95.43%, AUC = 0.9790	Balanced behavioral baseline; no external intelligence
Proposed System (Dataset 2)	CNN–LSTM + feature-level fusion (API + contextual attributes)	API call sequences + Hybrid Analysis features	Dataset 1 + Hybrid Analysis features	Acc = 97.49%, AUC = 0.9931	Improved reliability via external intelligence; requires API access
Proposed System (YARA Rules)	Dynamic YARA rule generation from CNN–LSTM outputs	API call sequences + Hybrid Analysis features	Dataset 2 (test split)	Acc = 97.14%	Slight accuracy drop vs DL model; high interpretability

5. Conclusion

This study outlined a malware detector as a hybrid cybersecurity system that combines api call sequence analysis of behaviors with dynamic flow of YARA rules. The rationale of the proposed system appears due to the increasing constraints of the conventional malware detection methods based on the high-reliance on either a static signature or a manually developed opinions, which can be easily bypassed in terms of obfuscated, polymorphic, or highly adaptable malware. The suggested structure on the other hand has advantages of integrating Convolutional Neural Networks (CNNs), which tend to learn local and short-range API-call patterns, and Long Short-Term Memory (LSTM) networks, which learn long-range temporal relationships across execution sequences. This architectural combination enables the system to extract fine-grained structural behavior while preserving temporal dependencies that unfold during program execution. Moreover, the external features are integrated and the dynamic YARA rules are generated automatically, which strengthens the detection, interpretability and practicability in real-world cybersecurity settings.

The efficiency of the system proposed was tested by the use of the extensive experiments that were performed on four datasets of various sizes, features configurations, and evaluation purposes. Baseline behavioral detection was tested using dataset 1 which was an academic benchmark comprising of 3,500 samples acquired through the combination of MalBehavD-V1 dataset and Oliveira API call sequence dataset. The standalone CNN model scored 92.46% on this dataset and the standalone LSTM model scored 90.17 on this dataset, which shows the weaknesses of individual models. Conversely, the hybrid CNNLSTM model was much more effective in the detection process as the accuracy represented it as 95.43% with an AUC as 0. 9790, which proved the benefit of considering both local API trends and longer-term time interactions.

dataset 2 augmented the same behavioral sequences with external features that were collected in the Hybrid Analysis platform, such as threat scores and antivirus detection counts. This improvement in performance resulted in an obvious increase in accuracy (97.49%), and AUC (0. 9931) of hybrid CNN-LSTM model and shows the benefit of the use of contextual intelligence in minimizing the false positives and false negatives. The findings support the hypothesis that a more effective and consistent malware detection system can be achieved when the behavioral execution patterns are used with lightweight external features.

One of the main contributions which this work has made is an incorporation of dynamic YARA rule generation. Through the most efficient model used on dataset 2, behavioral and context patterns that the deep learning architecture trained on were converted to executable YARA rules. On the test split of dataset 2 when the generated YARA rules are considered on their own, the achieved accuracy was 97.14, and there was also a small decrease in performance relative to deep learning classifier. This finding proves the fact that high level knowledge acquired by neural models can be effectively transferred into interpretable and deployable rule based signatures.

Overall, the experimental findings on 2 heterogeneous datasets indicate that the hybrid cybersecurity system proposed is highly accurate on detection, well generalized and makes sense in practice. The proposed framework offers competitiveness or better performance than the literature reviewed in the discussion section with the lightweight design avoiding expensive preprocessing pipelines or transformer-based architectures with high computational demands. By combining behavioral API sequence modeling, external features, and dynamic YARA rule generation, this work contributes a scalable and operationally viable solution for modern malware detection.

Though these are the strengths, the suggested framework has a number of limitations that can be discussed in the future. The contributions of LSTM layers are increasing the memory usage and time-to-train than convolutional architectures. Use of fixed length API sequences can be a constraint in terms of flexibility in analysis of variable length traces or real time monitoring. Additionally, external features capabilities might not necessarily work in every deployment environment because of network limitations, or missing metadata. Future studies can focus on finding more efficient sequence modeling architectures such as lightweight attention module or optimistic recurrent units so as to minimize the computational cost without compromising the accuracy. Expanding the framework to achieve multi-classification of malware families and performance of the framework in the context of noisy or adversarial operational environments would also increase practical usefulness and adaptability of the framework to the constant change of the field of cybersecurity.

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] 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.

[3] 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.

[4] A. Coscia, R. Lorusso, A. Maci, and G. Urbano, "APIARY: An API-based automatic rule generator for YARA to enhance malware detection," COMPUTERS & SECURITY, vol. 148, p. 104146, 2025.

[5] X. Zhang, X. Du, H. Chen, Y. He, W. Niu, and Q. Li, "Automatically generating rules of malicious software packages via large language models," in PROC. ARXIV PREPRINT SERIES, Online Conf., 2025.

[6] 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.

[7] 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.

[8] L. Qian and L. Cong, "Channel features and API-frequency-based transformer (CAFTrans) for malware identification," SENSORS, vol. 24, no. 2, p. 580, 2024.

[9] M. Pasco, "MalBehavD-V1 dataset," GitHub repository, 2025. [Online]. Available: https://github.com/mpasco/MalBehavD-V1

[10] Zarganaut, "API call sequences dataset," GitHub repository, 2025. [Online]. Available: https://github.com/zarganaut/API-call-sequences

[11] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," Scientific Reports, vol. 14, Art. no. 6086, 2024, doi: 10.1038/s41598-024-56706-x.

[12] J. Opitz, S. Mendoza, and A. Frank, "A Closer Look at Classification Evaluation Metrics and a Critical Reflection of Common Evaluation Practice," Transactions of the Association for Computational Linguistics, vol. 12, 2024.

[13] J. Carrasco, S. García, M. del Mar Rueda, S. Das, and F. Herrera, "Recent trends in the use of statistical tests for comparing machine learning algorithms," Pattern Recognition, vol. 97, 2020, Art. no. 107183.

[14] D. Chicco, M. J. Warrens, and G. Jurman, "The Matthews correlation coefficient (MCC) is more informative than Cohen's kappa and Brier score in binary classification assessment," IEEE Access, vol. 9, pp. 78368–78381, 2021, doi: 10.1109/ACCESS.2021.3084050.

[15] 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, art. 1457619, 2024, doi: 10.3389/fbinf.2024.1457619.

[16] A. Vanacore, M. S. Pellegrino, and A. Ciardiello, "Fair evaluation of classifier predictive performance based on binary confusion matrix," Computational Statistics, vol. 39, pp. 363–383, 2024, doi: 10.1007/s00180-022-01301-9.