

Lightweight Multi-Scale CNN Embeddings for Edge-Aware Malware Detection from API-Call Sequences

Giuseppe Bonomo
University of Palermo
Palermo, Italy

giuseppe.bonomo07@community.unipa.it

Marco Siino
University of Catania
Catania, Italy
marco.siino@unict.it

Giacomo Morabito
University of Catania
Catania, Italy
giacomo.morabito@unict.it

Abstract—Dynamic API-call traces provide compact behavioral evidence for malware detection, but practical detectors for mobile, edge, and networked infrastructures must balance classification accuracy, class imbalance, and computational cost. This paper evaluates a lightweight multi-scale convolutional neural network (MS-CNN) embedding family for malware classification from API-call sequences. Rather than claiming a new deep-learning primitive, the study investigates whether a simple TextCNN-style multi-scale architecture can serve as a strong, reproducible baseline for behavioral malware detection in controlled comparisons. The approach maps symbolic API calls into dense embeddings and applies parallel temporal convolutions to capture behavioral motifs at multiple local scales. Experiments compare the default MS-CNN, a higher-capacity MS-CNN variant, classical machine-learning baselines, controlled CNN variants, and a secondary CNN-BiGRU-Attention model on API-Call-Sequences and MalBehavD-V1. Under stratified 10-fold cross-validation, the strongest MS-CNN variant reaches 0.9442 macro-F1 on API-Call-Sequences, while the default MS-CNN reaches 0.9661 accuracy and 0.9661 macro-F1 on MalBehavD-V1. These results indicate that multi-scale CNN embeddings provide a strong accuracy-efficiency trade-off for API-sequence malware detection, while full edge readiness requires hardware-specific profiling.

Index Terms—Malware detection, API-call sequences, multi-scale CNN, dynamic analysis, edge security, mobile network security.

I. INTRODUCTION

Modern malware affects endpoints, cloud services, Internet-of-Things devices, and the communication infrastructure that connects them. This makes behavioral malware detection relevant to mobile, edge, and networked environments, where suspicious software activity may compromise distributed services, monitoring nodes, and command-and-control defenses. Static analysis remains useful but can be weakened by packing, obfuscation, and binary-level transformations. Dynamic analysis instead observes runtime behavior and can expose evidence that is harder to hide, including file operations, registry access, network activity, and API-call sequences [1], [2].

API-call traces provide compact behavioral fingerprints, but practical detectors must handle noisy fixed-length sequences, severe class imbalance, and computational constraints. Recent studies have explored classical machine-learning models,

recurrent networks, attention mechanisms, and graph-based representations for API-call malware detection [2]–[4]. These methods motivate a simpler deployment-oriented question: can a compact sequence-native convolutional model capture discriminative API-call motifs without recurrent processing, transformer-scale attention, or graph construction?

This paper evaluates a lightweight multi-scale CNN (MS-CNN) embedding family for API-call sequence malware detection. The model maps symbolic API identifiers into dense embeddings and applies parallel temporal convolutions with different kernel widths to capture local behavioral motifs at multiple scales, following the general TextCNN-style design used in convolutional sentence classification [5]. The contribution is not a new deep-learning primitive, but a controlled evaluation of a simple and reproducible architecture under imbalanced and deployment-aware conditions.

The main contributions are: i) an MS-CNN embedding family for fixed-length API-call malware classification; ii) a unified comparison against classical baselines, controlled CNN variants, and a CNN-BiGRU-Attention model; and iii) an evaluation on API-Call-Sequences and MalBehavD-V1 using accuracy, macro-F1, ROC-AUC, class-specific metrics, and computational indicators.

II. METHODOLOGY

A. Datasets and Preprocessing

Table I summarizes the two datasets used in this study, which represent complementary class-balance regimes. Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ denote a malware-detection dataset, where $x_i = (a_1, \dots, a_L)$ is an ordered sequence of API-call identifiers and $y_i \in \{0, 1\}$ is the binary label. The experiments use API-Call-Sequences and MalBehavD-V1. API-Call-Sequences contains 43,876 samples and follows the $t_0, \dots, t_{99}$ protocol used by Cannarile et al. [2]; it is highly imbalanced, with 1,079 goodware and 42,797 malware samples. MalBehavD-V1 contains 2,570 balanced samples and is used as a related benchmark for MalDetConv [6], [7]. All samples are padded or truncated to a fixed length of 100 API calls.

TABLE I
DATASET SUMMARY.

Dataset	Samples	Max length	Class distribution
API-Call-Sequences	43,876	100	1,079 / 42,797
MalBehavD-V1	2,570	100	1,285 / 1,285

B. MS-CNN Model

The proposed detector maps API-call identifiers to learned dense embeddings and processes the embedded sequence through parallel one-dimensional convolutional branches with kernel sizes 3, 5, 7, and 9. The branch outputs are batch-normalized, pooled using average and max temporal pooling, concatenated, regularized with dropout, and passed to a compact classifier. The default configuration uses 128-dimensional embeddings, 192 channels per branch, dropout 0.35, and cross-entropy loss. The strongest variant increases the number of channels to 256. Additional controlled variants modify embedding size, dropout, and focal loss.

C. Protocol and Baselines

All experiments use stratified 10-fold cross-validation; the outer folds use random seed 42, while 15% of each training fold is reserved for validation using a fold-dependent seed. Deep models are trained for up to 25 epochs with batch size 256 using AdamW, class-weighted cross-entropy, and early stopping based on validation macro-F1; focal loss is used only in the corresponding ablation. For neural models, the standard threshold of 0.50 is compared with a validation-selected threshold that maximizes macro-F1. This is especially relevant for API-Call-Sequences because the goodwill class is strongly underrepresented. The MS-CNN family is compared against RandomForest, ExtraTrees, XGBoost, CatBoost, TF-IDF Logistic Regression, and a CNN-BiGRU-Attention variant. The main metrics are accuracy, macro-F1, ROC-AUC, class-specific precision, recall, and F1, training time, and inference time.

For API-Call-Sequences, the comparison against Cannarile et al. [2] follows the same 100-call feature protocol and uses the reported CatBoost macro-F1 as an external reference. However, the original model is not re-executed on identical splits. For MalBehavD-V1, the comparison against MalDet-Conv [6] is related rather than exact because preprocessing details and evaluation splits may differ. Therefore, the strongest claims are based on internally rerun baselines and ablations, while literature comparisons are used as contextual anchors.

III. RESULTS

A. API-Call-Sequences

Table II summarizes the strongest API-Call-Sequences results. The best macro-F1 configuration is the 256-channel tuned multi-scale CNN, reaching 0.9442 macro-F1 and 0.9941 ROC-AUC. Its macro-F1 numerically exceeds the 0.8770 value reported for CatBoost by Cannarile et al. under the same 100-call feature protocol, although the original model was not re-executed on identical folds.

TABLE II
API-CALL-SEQUENCES RESULTS.

Model	Acc.	Macro-F1	ROC-AUC
CatBoost reference [2]	–	0.8770	0.8201
XGBoost	0.9901	0.8783	0.9846
MS-CNN default tuned	0.9946	0.9419	0.9944
MS-CNN-256 tuned	0.9948	0.9442	0.9941

TABLE III
CONTROLLED MS-CNN ABLATION ON API-CALL-SEQUENCES.

Variant	Acc.	Macro-F1	Train (s)
MS-CNN default tuned	0.9946	0.9419	10.53
MS-CNN base tuned	0.9947	0.9431	11.69
MS-CNN dropout 0.20 tuned	0.9944	0.9398	11.35
MS-CNN emb. 256 tuned	0.9940	0.9344	16.29
MS-CNN-256 tuned	0.9948	0.9442	13.91
MS-CNN focal tuned	0.9939	0.9342	10.49

Table III reports the controlled MS-CNN variants on API-Call-Sequences. The results show that the default configuration is already strong, while increasing channel capacity provides the best macro-F1. Increasing the embedding dimension to 256 does not improve performance under this protocol, suggesting that additional channel capacity is more useful than a larger token embedding for this dataset. Focal loss increases sensitivity to the minority class at the fixed threshold but does not provide the best macro-F1 after validation-based threshold tuning.

B. MalBehavD-V1

Table IV reports the main MalBehavD-V1 results. On MalBehavD-V1, the default CNN embedding with threshold 0.50 reaches 0.9661 accuracy and 0.9661 macro-F1, slightly above the approximately 0.9610 accuracy reported for MalDet-Conv. Because the comparison protocol may not be identical, this is treated as related evidence rather than a strict head-to-head claim.

Unlike API-Call-Sequences, MalBehavD-V1 is balanced. The default MS-CNN therefore performs best at the standard 0.50 threshold, while threshold tuning provides no consistent advantage. The embedding-size and dropout variants remain close to the default model, whereas the recurrent-attention variant is slower and does not improve the average macro-F1. This supports the interpretation that the simple multi-scale convolutional representation is sufficient for the fixed-length MalBehavD-V1 protocol.

Table V further highlights that the MalBehavD-V1 results are not driven by a single isolated run. Several MS-CNN variants remain within a narrow performance band, indicating that the convolutional embedding representation is stable on this dataset. The default setting is retained as the preferred MalBehavD-V1 configuration because it is both the most accurate and among the fastest neural variants.

TABLE IV
MALBEHAVD-V1 RESULTS.

Model	Acc.	Macro-F1	ROC-AUC
MalDetConv reference [6]	0.9610	–	–
MS-CNN default, threshold 0.50	0.9661	0.9661	0.9899
CNN embedding-size 256, threshold 0.50	0.9658	0.9657	0.9905
CNN dropout 0.20, threshold 0.50	0.9642	0.9642	0.9913

TABLE V
CONTROLLED MS-CNN ABLATION ON MALBEHAVD-V1.

Variant	Acc.	Macro-F1	Train (s)
MS-CNN default, 0.50	0.9661	0.9661	0.66
MS-CNN emb. 256, 0.50	0.9658	0.9657	0.77
MS-CNN dropout 0.20, 0.50	0.9642	0.9642	0.63
MS-CNN focal, 0.50	0.9630	0.9630	0.60
MS-CNN-256 tuned	0.9615	0.9615	0.77
CNN-BiGRU-Attn, 0.50	0.9603	0.9603	0.83

IV. DISCUSSION

The results support three observations. First, multi-scale CNN embeddings are effective on imbalanced API-call classification, especially when threshold tuning is used to improve macro-F1 under skewed class priors. Second, increasing channel capacity yields the highest observed macro-F1 on API-Call-Sequences, whereas the default configuration remains highly competitive on MalBehavD-V1. This suggests that model capacity should be adjusted according to dataset imbalance and sample complexity. Third, the CNN-BiGRU-Attention variant increases training and inference time without clearly improving the best MS-CNN results, suggesting that recurrence is not necessary under the fixed-length 100-call protocol.

These findings are relevant to communication and edge-security contexts. In mobile and wireless-network environments, such a detector can support lightweight behavioral triage before more expensive sandboxing, traffic correlation, or forensic analysis is triggered. A detector deployed near endpoints, gateways, or resource-constrained monitoring nodes must be accurate, but it should also be simple enough to train, reproduce, and execute efficiently. The MS-CNN family provides a practical compromise: it is more expressive than classical fixed-vector baselines, but less complex than recurrent-attention or graph-construction pipelines.

A. Impact of Class Imbalance

The API-Call-Sequences dataset is strongly imbalanced, with malware samples dominating the label distribution. In this setting, high accuracy can be achieved even when the minority class receives insufficient attention. The reported macro-F1 and minority-class behavior therefore matter more than accuracy alone. For the tuned MS-CNN-256 configuration, mean goodwill precision, recall, and F1 across the ten folds are 0.9209, 0.8638, and 0.8911, respectively. By comparison, XGBoost obtains 0.9273 precision, 0.6479 recall,

TABLE VI
EFFICIENCY AND NEURAL-MODEL COMPLEXITY ON
API-CALL-SEQUENCES.

Model	Macro-F1	Par. (M)	FP32 (MB)	Inf. (s)
XGBoost	0.8783	–	–	0.003
TF-IDF LogReg	0.8232	–	–	0.004
MS-CNN default tuned	0.9419	1.025	4.10	0.034
MS-CNN-256 tuned	0.9442	1.354	5.42	0.040
CNN-BiGRU-Attn tuned	0.9372	1.323	5.29	0.051

and 0.7618 F1 on the goodwill class. Thus, MS-CNN-256 substantially improves minority-class recall while maintaining comparable precision. Threshold tuning improves the trade-off by selecting a validation threshold that better balances the two classes. The best MS-CNN variants improve macro-F1 while maintaining high ROC-AUC, indicating that their probability scores preserve useful ranking information even under imbalance.

The datasets differ in class balance. Threshold-tuned configurations rank highest on API-Call-Sequences because class priors are skewed. On MalBehavD-V1, the default 0.50 threshold is usually enough. This suggests that deployment should not treat the classifier and decision threshold as separate afterthoughts: class priors and operational false-positive constraints should be considered when selecting the final detector.

B. Why Multi-Scale Convolution Helps

API-call sequences contain short behavioral motifs such as file access, process inspection, registry operations, or network-related calls. These motifs may appear with local variation, and their exact positions may differ across samples. Parallel convolutions with multiple kernel sizes are well suited to this structure because they can learn local detectors over different subsequence widths. Average and max pooling then aggregate both distributed evidence and the strongest evidence across the sequence. This explains why the MS-CNN family can outperform fixed-vector classical baselines while avoiding the sequential cost of recurrent layers.

The observed behavior also explains why larger embeddings alone do not dominate the results. API identifiers are symbolic and useful, but the discriminative signal often depends on local combinations rather than isolated token identity. Increasing convolutional channel capacity gives the model more filters for detecting these combinations, while increasing embedding dimensionality mainly expands the representation of individual tokens. The ablation results are consistent with this interpretation: the 256-channel variant improves the imbalanced API-Call-Sequences result, whereas the 256-dimensional embedding variant is not the best performer.

C. Operational Interpretation

For a deployment-aware malware detection pipeline, the detector may be used as a triage module rather than a standalone final verdict. A lightweight API-sequence detector can quickly flag suspicious executions for deeper analysis, sandbox replay, network containment, or human review. In this setting, the

value of the MS-CNN family is its balance between accuracy and simplicity. It avoids handcrafted graph construction and recurrent sequence processing while still learning richer representations than shallow fixed-column baselines.

As shown in Table VI, parameter counts and FP32 weight footprints are computed directly from the architecture definitions and do not require hardware profiling. The default MS-CNN has the smallest neural-model footprint, with 1.025 million parameters and approximately 4.10 MB of FP32 weights. MS-CNN-256 achieves the best macro-F1 and remains faster than CNN-BiGRU-Attention, although its parameter count is slightly higher. This suggests that its computational advantage derives from parallel convolutional processing rather than from parameter count alone. The reported inference times remain software-level measurements and should not be interpreted as hardware-level edge benchmarks. Each inference time is the mean wall-clock time required to process one complete outer test fold. Classical baselines execute on CPU, whereas neural models use the available CUDA device; therefore, cross-family timing comparisons are descriptive rather than hardware-normalized.

D. Limitations and Threats to Validity

The study has several limitations. First, all experiments use a maximum sequence length of 100 to preserve comparability with prior work. This is the most important security limitation: malware can delay malicious actions, sleep, or execute benign-looking calls before revealing harmful behavior. Therefore, the reported setting should be understood as an early-trace benchmark, not as evidence that 100 calls are sufficient for all real-world malware. Second, the MalBehavD-V1 comparison with MalDetConv is related rather than exact because the original preprocessing and evaluation protocol may differ, and the external baselines are not fully rerun under identical splits. Third, runtime is measured at the script level and should be complemented by deployment-specific profiling, including memory footprint, power use, throughput, and latency on target edge hardware.

Another threat to validity is dataset age and representativeness. Malware behavior evolves, and a detector trained on one corpus may not generalize perfectly to new families, new sandbox environments, or adversarially delayed behavior. The use of two datasets partially mitigates this concern, but broader evaluation on temporally separated samples would strengthen the claim. Finally, the current analysis focuses on binary classification. Multi-family malware classification and calibrated risk scoring would require additional evaluation.

V. CONCLUSION

This paper presented a lightweight MS-CNN embedding family for API-call sequence malware detection. Across two datasets, the approach improves macro-F1 over strong API-call baselines on API-Call-Sequences and remains competitive on MalBehavD-V1 with low software-level runtime cost. The strongest evaluated configuration, MS-CNN-256 tuned, obtains 0.9442 macro-F1 on API-Call-Sequences, while the

default MS-CNN obtains 0.9661 macro-F1 on MalBehavD-V1. Future work will evaluate longer traces, memory footprint, model calibration, external datasets, hardware-level edge profiling, and hybrid graph-sequence models under identical deployment-oriented protocols.

AUTHOR CONTRIBUTIONS

G. Bonomo: conceptualization, software, experimental design, data curation, validation, formal analysis, visualization, and original draft writing. M. Siino: research guidance, methodology discussion, supervision, and manuscript review. G. Morabito: supervision, manuscript review, methodological feedback, and funding acquisition.

REPRODUCIBILITY STATEMENT

The code for this paper is available at: https://github.com/Bonomo31/multi_scale_api_malware_detection.

REFERENCES

- [1] E. Amer and I. Zelinka, "A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence," *Computers & Security*, vol. 92, p. 101760, 2020.
- [2] A. Cannarile, V. Dentamaro, S. Galantucci, A. Iannacone, D. Impedovo, and G. Pirlo, "Comparing deep learning and shallow learning techniques for API calls malware prediction: A study," *Applied Sciences*, vol. 12, no. 3, p. 1645, 2022. [Online]. Available: <https://www.mdpi.com/2076-3417/12/3/1645>
- [3] N. Owoh, J. Adejoh, S. Hosseinzadeh, M. Ashawa, J. Osamor, and A. Qureshi, "Malware detection based on api call sequence analysis: A gated recurrent unit-generative adversarial network model approach," *Future Internet*, vol. 16, no. 10, 2024. [Online]. Available: <https://www.mdpi.com/1999-5903/16/10/369>
- [4] C. Li *et al.*, "Dmalnet: Dynamic malware analysis based on api feature engineering and graph learning," *Computers & Security*, vol. 122, p. 102872, 2022.
- [5] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1746–1751.
- [6] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "MalDetConv: Automated behaviour-based malware detection framework based on natural language processing and deep learning techniques," 2022. [Online]. Available: <https://arxiv.org/abs/2209.03547>
- [7] M. Pasco, "MalBehavD-V1: Malware behavior dataset," 2022, gitHub repository. [Online]. Available: <https://github.com/mpasco/MalbehavD-V1>