Nebula: Self-Attention for Dynamic Malware Analysis

Dmitrijs Trizna[1,2,3], Luca Demetrio *Member, IEEE*[3], Battista Biggio, *Senior Member, IEEE*[4], and Fabio Roli *Fellow, IEEE*[3,4]

[1]Microsoft Corporation
[2]Department of Computer, Control and Management Engineering, Sapienza University, Rome, Italy
[3]Department of Informatics, Bioengineering, Robotics, and Systems Engineering, University of Genoa, Italy
[4]Department of Electrical and Electronic Engineering, University of Cagliari, Italy

***Abstract*—Dynamic analysis enables detecting Windows malware by executing programs in a controlled environment and logging their actions. Previous work has proposed training machine learning models, i.e., convolutional and long short-term memory networks, on homogeneous input features like runtime APIs to either detect or classify malware, neglecting other relevant information coming from heterogeneous data like network and file operations. To overcome these issues, we introduce Nebula, a versatile, self-attention Transformer-based neural architecture that generalizes across different behavioral representations and formats, combining diverse information from dynamic log reports. Nebula is composed by several components needed to tokenize, filter, normalize and encode data to feed the transformer architecture. We firstly perform a comprehensive ablation study to evaluate their impact on the performance of the whole system, highlighting which components can be used as-is, and which must be enriched with specific domain knowledge. We perform extensive experiments on both malware detection and classification tasks, using three datasets acquired from different dynamic analyses platforms, show that, on average, Nebula outperforms state-of-the-art models at low false positive rates, with a peak of 12% improvement. Moreover, we showcase how self-supervised learning pre-training matches the performance of fully-supervised models with only 20% of training data, and we inspect the output of Nebula through explainable AI techniques, pinpointing how attention is focusing on specific tokens correlated to malicious activities of malware families. To foster reproducibility, we open-source our findings and models at https://github.com/dtrizna/nebula.**

***Index Terms*—Malware, Transformers, Dynamic Analysis, Convolutional Neural Networks**

I. Introduction

DYNAMIC malware analysis is a crucial task not only for detecting but also for understanding the threats that are widespread over the entire Internet. Once samples are collected, analysts execute malware inside isolated environments (sandboxes or emulators), where they list all the actions performed by the program like network and filesystem access, registry modifications, API calls, and kernel syscalls [1]. These actions are then summarized into textual reports, which are manually analyzed by experts to distill the rationale behind the maliciousness of the analyzed sample. This task is tedious and resource-intensive since it involves domain experts in the process and manual labeling.

Machine learning (ML) techniques, particularly Convolutional Neural Networks (CNNs) and Long Short-term Memory (LSTM) models, are now widely utilized to streamline this process. These models are trained on vast volumes of textual reports, allowing quicker classification of new inputs and reducing human intervention [2], [3], [4], [5], [7], [8]. While CNNs capture local patterns in reports, providing valuable features for neural architectures, LSTM models learn global token relationships [9], [10]. However, these proposed schemes are hindered by three main downsides: (i) convolutions only capture local information, discarding the global correlations contained in reports between actions, while LSTM models struggle in modeling sample behavior based on prolonged token sequences, like a chain of API calls with arguments; (ii) most of the proposed techniques solely rely on homogeneous input data, like API calls [4], [2], [8], rather than leveraging more complete and heterogeneous information representing the behavior of malware samples; and (iii) source code, data, and pre-trained models are typically not available for most of the proposed techniques, hindering reproducibility.

To overcome these issues, we present Nebula, an ML model based on the Transformer architecture [11] trained on reports of different nature and formats. Unlike traditional models, Nebula leverages the *self-attention* mechanism inherent in Transformer neural networks, granting Nebula the capability to discern both local and global relationships in a report. To the best of our knowledge, we are the first to propose general Transformer architecture to tackle both malware detection and classification from raw dynamic log reports. Instead of solely focusing on few portions of reports, we design Nebula to properly work on all the output provided by sandboxes, thus making Nebula able to correlate tokens from different sources. To build Nebula, we consider several data cleaning approaches and and feature extractors, and we deeply study their effect through an extensive ablation study (Sect. IV-C). Through this analysis, we highlight that some standard NLP techniques, like tokenization through Byte Pair Encoding (BPE) can be applied “as-is”, while it is necessary to preprocess data through the lenses of domain knowledge, by replacing mostly-unique tokens like specific IP addresses, hashes, and internet domains. We then test Nebula against different state-of-the-art approaches leveraging both CNNs and LSTMs, and we

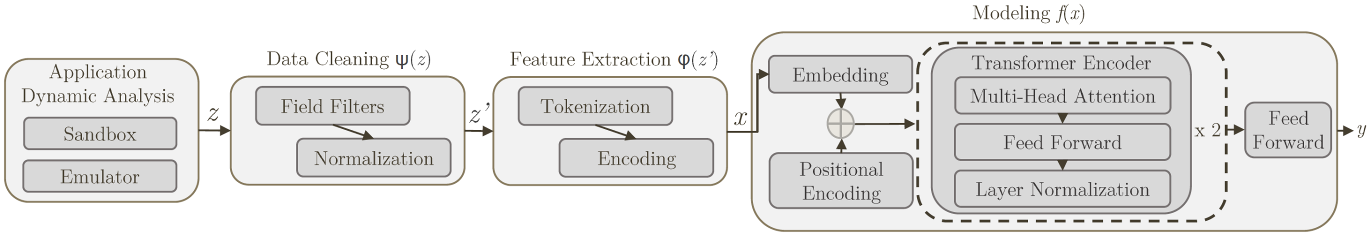


Fig. 1: A schematic overview of Nebula.

benchmark their performances on both malware detection and classifications tasks span on three distinct datasets acquired from different sandbox environments (Sect. IV-D). From our analyses we can conclude that Nebula is, on average, the best model to handle both tasks on all the considered dataset with performances aligned to the state of the art. In particular, we highlight a peak of improvement of 12% true-positive rate on a specific dataset with an very low false-positive rate of 10^{-3} setting. This result is achieved under a strict regime of very low false positive rate, which is a crucial aspect for deployed systems [5], [7]. Also, we inspect how self-supervised learning can reduce the number of training data needed to fine-tune Nebula on the malware detection task (Sect. IV-E). Our results exhibit a positive trend, highlighting that Nebula achieves state-of-the-art performance when firstly pre-trained on 80% of training dataset, and fine-tuned for the downstream task of malware detection with only the remaining 20% of samples. Lastly, we review the output of Nebula through the lenses of explainable AI techniques [14], [17] (Sect. IV-F), by confirming that our model focuses on tokens specific to certain malware behaviors, backed up by domain knowledge. To foster reproducible results, we do not only share the code and pre-trained models of Nebula, but we also re-implement, re-train and release methods that were previously closed source [8], [3].[1]

II. Dynamic Windows Malware Analysis

This section provides the background information necessary to understand the technical advancements made by Nebula.

A. Malware Behavioral Reports

System compromise can have different manifestations depending on impact, like sensitive data exposure or the misuse of computational resources. Adversaries employ a diverse range of methods, from utilizing built-in tools and protocols aligning with a "living-off-the-land" approach to leveraging stolen credentials or employing social engineering tactics to achieve their goals through legitimate user accounts. Threat actors often deploy their own software agents, referred to as malware. According to the 2022 Verizon Data Breach Investigation Report [20], malware was responsible for nearly 40% of breaches. Malware analysis can be segregated into static and dynamic methodologies. The former entails the evaluation of software samples without executing them. On the contrary, dynamic software analysis is a process that commences with the "detonation" of a sample in a controlled environment. Dynamic analysis is done by isolation of application, preventing it from impacting other system parts, while maintaining the realism of potential target system to extract malware actions, producing a behavioral report [16]. These are readable text files that summarize all the meaningful events captured by the sandbox, and they are listed to help the analysis task conducted by humans.

B. Machine Learning Pipeline for Dynamic Analysis

Machine learning has become a significant element in malware analysis, with efficient modeling schemes proposed for both static and dynamic data structures derived from malware. We now describe how ML can employ textual reports by introducing three main steps: (i) *data cleaning* to prepare raw data; (ii) *feature extraction* to create a mathematical representation of reports; and (iii) *modeling* the problem to train the final classifier, as depicted in Fig. 1 given modeling part is performed by Nebula.

Data Cleaning. First, the behavioral report is cleaned and normalized to make the data manageable for further processing. Filters are used to remove unnecessary data and preserve only a specific set of fields, while normalization techniques are applied to systematize values that are stochastic in nature and do not correlate with application behavior like hash-sums or IP addresses. This allows us to introduce domain knowledge [23] and, as shown in Sect. IV-C, improves the model's generalization abilities by reducing variability in values irrelevant to the prediction. We denote this step as $\boldsymbol{z}' = \psi(\boldsymbol{z})$, where $\boldsymbol{z}$ is the raw data collected from the dynamic analysis environment, and $\boldsymbol{z}'$ is the cleaned and normalized textual data.

Feature Extraction. Then, z' undergoes feature extraction denoted $\boldsymbol{x} = \phi(z')$. As a final step, producing a numerical array x, suitable for analysis by ML model. Feature extraction ϕ involves a dichotomy between (a) feature engineering or (b) token encoding. Feature engineering involves the manual or automated selection and transformation of relevant features from the cleaned data z', for instance, feature hashing applied to API call names [8] or regular expression-based feature extractors [8]. Token encoding involves a tokenization step, which transforms the textual data z' into a sequence of tokens and a vocabulary V of all possible tokens. Tokenization can be based on regular expressions, be influenced by a domain knowledge [26], or involve statistical methods like Byte-Pair Encoding (BPE) [27], [28]. The sequence of tokens is then encoded into a numerical array x using an encoding function f, which might be as simple as one-hot encoding, be calculated

[1] https://github.com/dtrizna/nebula

TABLE I: Dynamic malware analysis modeling techniques.

	Data Cleaning $\psi(\boldsymbol{z})$	**Feature Extraction** $\phi(\boldsymbol{z}')$	**Model** $f(\boldsymbol{x})$	**Size**	**Code Released**	**Comment**
Neurlux [3]	✗	Tokenization	CNN, LSTM, Attention	2.8M	✓	
Gated CNN [8]	API filter	Feature Hashing	CNN, LSTM	0.4M	∼	Shared privately
Quo.Vadis [2]	API filter	Tokenization	CNN	1.4M	✓	
JSONGrinder [18]	✗	HMIL [19]	MLP	2.4M	∼	Non-functional
CruParamer [4]	API filter	API "labeling"	CNN, LSTM	–	✗	
Nebula (ours)	API, network, file, registry filters and normalization	Tokenization	Transformer (Self-Attention)	5.6M	✓	

with term frequency-inverse document frequency (TF-IDF), or use embedding function $f : V \rightarrow \mathbb{R}^d$, where d is the embedding dimension.

Modeling. The final step is to use the numerical array x as an input to a ML model $f(x)$, which produces a prediction y of a malware label. The modeling function can be as simple as linear models like logistic regression. However, for behavioral reports, the best schemes incorporate representations of sequential information. This can be achieved by convolutions, recurrent neural networks or self-attention with positional encoding.

C. Review of Dynamic Models

The landscape of behavioral malware analysis showcases a competitive interplay between commercial solutions and academic research, with different attitude towards a modeling an adversary. Commercial anti-virus (AV) and Endpoint Detection and Response (EDR) products have integrated behavioral analytics into their detection methodologies, forming part of a multi-objective heuristic that leverages both static and dynamic analysis. The behavioral components of their multi-objective heuristics are closed, which prohibits their disentanglement on the user side for comparison purposes. This lack of transparency means that we cannot gauge how much of the overall performance of these commercial solutions is attributed to their behavioral modeling component specifically. Also, AVs and EDRs work in *real-time* settings, implying that decisions are taken in a matter of milliseconds, opposed to sandbox analyses that are conducted *offline*, and later evaluated thoguh reports. Due to these discrepancies, in this work we will only focus on academic dynamic malware analysis conducted offline through sandbox analyses, since its comparison with AVs and EDRs would be unfair. In academic research, we encounter several groundbreaking methodologies in dynamic malware analysis that pose a formidable challenge to the current state-of-the-art. To offer a consolidated view of these promising approaches, we have curated a selection of these solutions in Table I, systematizing their respective pipelines according to the steps introduced in Sect. II-B. A common theme among contemporary academic contributions is the employment of traditional techniques, such as one-dimensional convolutions, optionally complemented with recurrent layers through Long Short-Term Memory (LSTM) [9], as part of their core modeling approach f. However, each of these methodologies introduces a unique approach in either data cleaning (ψ) or feature extraction (ϕ) processes, thereby diversifying the analytical landscape of dynamic malware analysis.

Neurlux (Jindal et al. [3]). A distinctive feature of this approach is the absence of operations during the data cleaning phase (ψ), passing raw behavioral reports directly to the feature extraction process (ϕ). This phase involves a simple whitespace tokenization procedure and sequences encoding with a vocabulary size of $V = 10,000$. The resulting sequences are then modeled f using a combination of one-dimensional convolutions, LSTM, and conventional attention mechanisms [10], which is applied to the output of the LSTM layer. Their code is publicly accessible; therefore, we are able to compare our results with this model. However, the data utilized in their research remains undisclosed.

Gated CNN (Zhang et al. [8]). This model introduced an analysis where ψ preserves only API call data, each undergoing a custom feature engineering process during ϕ phase. Then, the sequence of featurized vectors is modeled though a gated convolution network as f. While the code for their model is not released publicly, it was provided by the researchers upon request, enabling us to draw a direct comparison between our results and their model. However, similar to the case of Neurlux, the data utilized in their study has not been released.

Quo.Vadis (Trizna [2]). This hybrid model simultaneously assesses contextual, static, and dynamic features. Their model code is released publicly, which significantly contributes to the transparency of their work. For our analysis, we concentrated on the dynamic component of their pipeline, which data cleaning ψ preserves only API call names. Feature extraction ϕ label-encodes each API call name with a vocabulary of $V = 600$ and subsequently models f with a 1d convolutional neural network. This work is especially notable for its public release of a comprehensive dataset consisting of Speakeasy [29] emulation reports. This allows for the pursuance of both malware detection and type classification objectives.

JSONGrinder (Bosansky et al. [18]). This model provides a unique method for parsing hierarchical JSON reports, originally proposed in [19]. This method employs a combination of Julia libraries, specifically `JsonGrinder.jl` used for feature extraction ϕ and `Mill.jl` for modeling f, data cleaning ψ is omitted. The ϕ phase infers a Hierarchical Multiple Instance Learning (HMIL) schema from the data, constructing a fixed-size vector, while the modeling is based on a multilayer perceptron (MLP) for sample classification. However, it is worth noting that their implementation was not compatible with the latest version of Julia (v1.8.5) at the time of our experiments, causing the original model implementation to fail without modifications. Additionally, Bosansky et al.'s work is notable for its release of a comprehensive dataset useful for malware family classification, which adds considerable value to the existing body of resources in this field.

CurParamer (Chen et al. [4]). This method preserves only API calls from the original report during the data cleaning phase (ϕ). The feature extraction step (ψ) involves a unique approach to API labeling and embedding, which includes parameter-assisted API labeling and sensitivity-inspired API embedding. These techniques utilize domain knowledge to generate more efficient numerical representations of API calls. To model these representations (f), they employ two separate networks based on 2D convolutions and LSTM. Although their feature extraction methodology is intriguing, it is presented with little implementation details, which reduces its replicability. Despite efforts to access the modeling code, the authors made no public version available, even upon private request.

III. Nebula: Transformer Architecture for Dynamic Malware Detection

The design of our dynamic malware analysis pipeline draws from the proven success of the attention mechanism in Natural Language Understanding (NLU). Particularly, the self-attention-based Transformer architecture [11] has demonstrated superior performance over conventional RNN- or CNN-based modeling methods [12], [13]. These successes guided the selection of techniques used during our feature extraction (ϕ) and modeling (f) stages. The most significant deviation from standard NLU pipelines is evident during the data cleaning phase (ψ). Here, we employ a domain-specific parser that (i) retains only those fields from the original structured report relevant for behavior generalization; and (ii) normalizes unconstrained and arbitrary values within such selected fields. In the feature extraction phase (ϕ), we tokenize each report into a sequence of tokens of length N and encode each token based on a vocabulary of size V. When modeling this sequence, we first embed the input vector to a higher dimension, apply position encoding, and then process it through a Transformer encoder layer to apply the self-attention operation. The resulting attended tensors are then forwarded to a classifier that produces the final prediction. A high-level overview of our Nebula modeling scheme is depicted in Fig. 1.

A. Data Cleaning

We detail the data cleaning $\boldsymbol{z}' = \psi(\boldsymbol{z})$ applied by Nebula.

Vocabulary and Field Filters. Machine data is more volumetric and heterogeneous than natural languages. Therefore, it can have a significantly larger vocabulary, as no distinct lexical boundaries or grammatical rules define the language being used. In system logs, it is common to see arbitrary character combinations like `/tmp/83afba/setup.bin` or `jre1.8.0_311`, which explode vocabulary given improper handling. For instance, even after path normalization, we observe more than 6000 unique filepaths, where only roughly 400 paths repeat, and the rest appear only once. The Fig. 2 visualizes the frequency distribution of tokens for different JSON fields in the Speakeasy emulated behavioral report training set [2]. Every additional field included in the analysis increases the vocabulary size. For instance, given filter that uses API calls, file, network, and registry records total vocabulary size is about 2.5M tokens. Given no filters applied, this number jumps close to 8M tokens, exploding vocabulary more than three times and significantly reducing the epistemic density (valuable information per token) of the data.

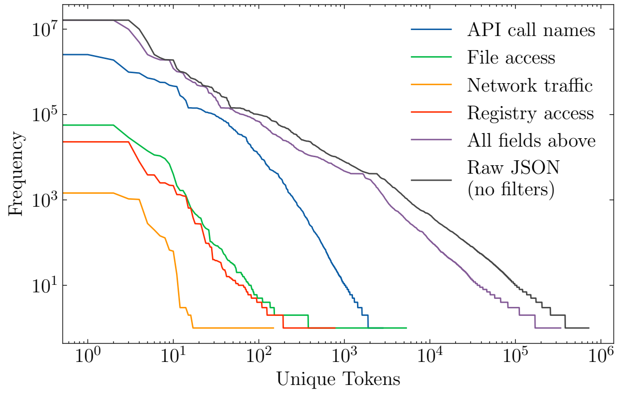


Fig. 2: Visualization of whitespace token frequency in the Speakeasy [29] emulated behavioral report training set [2].

Concerning field filtering, existing dynamic malware modeling techniques fall into two categories: do not implement any filters [3], or use only a single type of information, usually API calls [4], [2]. Focus solely on API calls eliminates valuable behavior representations necessary for establishing effective decision boundaries in malware detection. Research has demonstrated that domain experts rely on a broader range of information when performing actual malware analysis [23]. Based on ablation studies discussed in Sect. IV-C, we preserve the following fields from dynamic analysis report: (i) API call names, arguments, and return codes; (ii) file operation type and path; (iii) network connection port and server name; and (iv) registry access type and key value. We found this combination of fields produces the best generalization and the least overfitting.

Normalization. Retained fields still have unbounded or unpredictable values, which may not inherently contribute to the effectiveness of ML models. For instance, the exact values of IP addresses are not representative *per se* and primarily provide broader context, such as indicating whether the IP is from a private or public network or what autonomous system it belongs to. Similarly, file paths may contain elements like usernames, drive letters, or randomized file and directory names, which have relative contextual significance for behavioral analysis. Hence, the raw values of such fields may not be directly beneficial for modeling, emphasizing the need for suitable normalization before analysis. We incorporate domain knowledge via placeholders by normalizing filepaths, network connection, and registry access information in the following manner: (i) hash-sums in any field, including SHA1, SHA256, and MD5, are substituted with placeholders like `<sha1>`, `<sha256>`, `<md5>` placeholders; (ii) IP addresses are mapped to placeholders symbolizing loopback, private, public, or IPv6 addresses; (iii) recognizable domain names associated with a list of common top-level domains such as `com` or `net` (but not exclusive to these) are assigned the `<domain>` placeholder; (iv) Windows path variables, for instance, `%windir%` or `%userprofile%`, are expanded to a full path; and (v) frequent Windows paths patterns are replaced with specific placeholders such as `<drive>` or `<user>`.

B. Feature Extraction

We now detail the feature extraction $\boldsymbol{x} = \phi(\boldsymbol{z}')$ applied after the data cleaning phase ψ by Nebula.

Tokenization. This operation is pivotal for dealing with textual data, since it divide the input text into several atoms named *tokens*, that represent input data in a comprhensible way for machine learning models. Two basic approaches to tokenization, namely Whitespace and Wordpunct, have been traditionally employed, both using regular expressions to split the text. Former separates words based on spaces, tabs, and newline characters, while latter on top of that uses punctuation as separators. A fragment of whitespace tokenized dynamic analysis report:

```
"0x0", "0x1", "kernel32.getprocaddress",
"0x1000", "0xfa", "kernel32.tlsgetvalue"
```

In contemporary deep learning solutions, a more sophisticated approach to tokenization has emerged [12], predominantly based on Byte Pair Encoding (BPE) [27], which initially served as a data compression algorithm [15]. The adoption of BPE as a tokenizer is attributed to its ability to adapt to various languages and tasks seamlessly. Ideologically, BPE is well suited for machine data such as malware reports, since its data-driven nature allows to learn the optimal tokens scheme directly from the data. Notably, to handle the intricacies of low-level data in dynamic malware reports, we adjust BPE to incorporate all raw bytes and UTF-8 characters as base tokens. This ensures that event the most rare and unique elements of malware report will have a token-level representation. The redacted set of BPE tokens covering the same dynamic analysis report fragment are as follows:

```
"0x", "0x1", "ne", "32.", "kernel32.",
"et", "ad", "getproc", "10", "0xf", "tls
```

Furthermore, for both tokenization schemes, we limit our vocabulary to $V = 50000$ most common tokens and introduce two special tokens to denote all other tokens (`<unk>`) and padding of shorter sequences (`<pad>`).

Sequence Length. In the case of machine data, the tokenized sequences from system log events are typically lengthy. To manage this, we confine behavioral reports to the first N tokens. By keeping the computational budget constant, we evaluate the performance of models with varying sequence lengths. The results of these comparative studies, often referred to as ablation studies, will be detailed in Sect. IV-C, with the final choice of $N = 512$.

C. Model Architecture

We now detail the last component of Nebula, which is the model function f.

Embedding and Positional Encoding. Embedding operation maps the input sequence of integers to a higher dimensional space: $e = E(x) \cdot \sqrt{d_e}$, where $E(x)$ is the embedding of the input x and d_e is the dimension of the embedding, with square root used for scaling. This results in vector $e = [e_1, e_2, ..., e_{pos}, ..., e_N]$, where $e_{pos} \in \mathbb{R}^{d_e}$.

Since our method relies on the Transformer architecture, which lacks the inherent sense of order provided by recurrent models, we need to incorporate positional information in our sequence. There are multiple alternative ways to encode position. We replicate the approach introduced by Vaswani et al. [11], creating a set of sinusoidal functions with different frequencies for each position in the sequence:

$$PE_{(pos,\ 2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \tag{1}$$

$$PE_{(pos,\ 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right), \tag{2}$$

where $PE_{(pos,\ i)}$ is the i-th dimension of the positional encoding of the token at position pos in the sequence, and d is the dimensionality of the model. The $PE_{(pos,\ 2i)}$ and $PE_{(pos,\ 2i+1)}$ terms are used for even and odd dimension i respectively. These values are then added to the embedded vectors e_{pos} to incorporate the positional information into the sequence $e'_{pos} = e_{pos} + PE_{pos}$ where $PE_{pos} = [PE_{(pos,\ 1)}, PE_{(pos,\ 2)}, ..., PE_{(pos,\ d)}]$ is the positional encoding vector for position pos. The result is a sequence of vectors $e' = [e'_1, e'_2, ..., e'_N]$, where each vector represents both the token semantics and its position in the sequence, which can now be fed into the Transformer network.

Neural Layers. We leverage the Transformer architecture, which originally employs both encoder and decoder layers [11]. Our setup utilizes only encoder layers similar to Devlin et al. [13], a design choice that aligns our model with inference task rather than generative objectives as in applications that include decoder [11], [12]. We employ two Transformer encoder layers that align our model size with those of comparable models in Table I. This choice is not restrictive – the model can be scaled up to incorporate more Transformer layers to improve performance, consistent with the principle of model scaling laws [30]. After the self-attention operation, data is forwarded to a classifier for the final prediction. In our implementation, the classifier consists of a fully connected neural network with a single hidden layer composed of 64 neurons and the final layer for binary or multi-class classification.

Reduced Self-attention Span. Input comprised from structured machine data like malware behavior reports contain information in lengthy sequences, which poses a challenge for self-attention architectures like that used by Transformers [11], since such models exhibit quadratic complexity with respect to the sequence length. The self-attention operation can be represented as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{3}$$

where Q, K, and V as queries, keys, and values, respectively used as inputs to a self-attention layer, and d_k is the dimension of the keys. The product QK^T results in a matrix of size $N \times N$, where N is the sequence length. Calculating this product has a complexity of $O(N^2)$, leading to the quadratic computational complexity with respect to the sequence length.

TABLE II: Number of samples per malware family in Avast-CTU Dataset[18].

Family	**Adload**	**Emotet**	**HarHar**	**Lokibot**	**njRAT**	**Qakbot**	**Swisyn**	**Trickbot**	**Ursnif**	**Zeus**	Total
Samples	704	14429	655	4191	3372	4895	12591	4202	1343	2594	54000

TABLE III: Speakeasy Dataset [2] structure and size.

	Training set		Test set	
Sample label	**Size (Gb)**	**Count**	**Size (Gb)**	**Count**
Benignware	127.0	26061	47.0	10000
Backdoor	30.0	11089	7.4	2500
Coinminer	46.0	10044	11.0	2500
Dropper	36.0	11275	9.0	2500
Keylogger	34.0	7817	9.8	2500
Ransomw.	14.0	10014	4.6	2500
RAT	5.5	9537	2.5	2500
Trojan	40.0	13128	7.1	2500
Total	329	98966	98	27500

We propose an alternative approach to reduce computational complexity by partitioning the self-attention operation described in Equation 3 into several independent attention spans instead of applying it to the entire sequence. Assume that the original sequence length N is divisible by the span S, so there are $M = N/S$ spans. Let Q_i, K_i, and V_i denote the queries, keys, and values for the i^{th} span. Then, Attention$(Q_i, K_i, V_i), \quad \forall i \in \{1, 2, ..., M\}$ and independent attention results are concatenated to vector of size N.

In this way, the complexity is reduced to $O(MS^2)$, improving the model's computational efficiency, especially when $S << N$. Our experiments use $S = 64$ with $N = 512$, resulting in $M = 8$ independent self-attention spans. We observe that reducing attention spans enhances the model's inferential capacity on behavioral reports while adhering to the same computational constraints.

IV. Experimental Evaluation

The following section presents an in-depth experimental evaluation designed to assess the effectiveness and robustness of Nebula. We discuss the dataset used for our experiments (Sect. IV-A), and we outline our setup (Sect. IV-B). We then present our ablation study on the components of Nebula (Sect. IV-C), followed by its comparison with the state of the art (Sect. IV-D). Lastly, we analyse the benefits that self-supervised pre-training has on the required number of data to fit Nebula (Sect. IV-E), and we conclude by analysing its output with explainability techniques, confirming our findings through domain knowledge (Sect. IV-F).

A. Datasets

In our experiments, we evaluate three publicly available datasets by discussing two different types of analysis.

Malware Detection. This binary classification task discerns between benign and malicious software. It's a fundamental task performed by AV and EDR solutions with the aim of detecting malevolent logic running on a system. In real-world applications, it is paramount to maintain severely low false-positive rates to ensure usability and efficiency.

Malware Classification. This is a multi-label classification objective, targeting the attribution of malware samples to a specific type or family. Threat intelligence teams often execute it to study the evolution of malware strains, uncover shared characteristics, and identify potential countermeasures. We now characterize each dataset according to the best practices established in the malware research [16] by its sample size, the environment used for data collection, its applicability for either malware detection and classification tasks, and the availability of separate training and test sets.

Speakeasy Dataset [2]. This dataset[2] was generated using Speakeasy v1.5.9 [29], a Windows kernel emulator, comprising behavioral reports from in total approximately 93,500 samples, with both legitimate and malicious JSON reports. The malicious samples belong to seven distinct malware types, with sample prevalence across labels detailed in Table III. Therefore, the dataset is suitable for both malware detection and classification tasks. The dataset provides a test set explicitly, collected in a different timeframe (April 2022) from the training set (January 2022). This temporal separation facilitates the examination of concept drift in malware behavior.

Avast-CTU Dataset [18]. This dataset[3] houses sandbox reports in JSON format derived from CAPEv2 [31] (a Cuckoo sandbox [32] derivative), with approximately 400,000 samples collected between January 2017 and January 2020. The reports represent ten different malware families (Table II). Due to the absence of legitimate samples, this dataset is solely used for malware classification tasks. Also, this dataset lacks sequential information, and it only provides a summary of the events colelcted by the sandbox. The dataset formation aligns with the splitting approach recommended by Bosnansky et al. [18], in which all samples preceding August 2019 are designated as the training set, while the remainder forms the test set.

Malicious Code Dataset (MCD) [33]. This dataset has approximately 30,000 labeled samples containing API call sequences in XML format without any additional behavioral data (such as filesystem, registry, or network access). The dataset's collection methodology and the environment are not explicitly detailed. The training set contains 10,000 malware and 20,000 goodware samples. As no malware family or type labels are available, this dataset is solely applicable for malware detection task. The test set with 15,000 unlabeled samples cannot be used for evaluation due to the lack of labels. Hence we report mean metrics only on validation sets through cross-validation folds.

B. Experimental Setup

Our experiments were conducted on an NVIDIA Quadro T2000, a standard consumer GPU. To align with the limitations of the hardware capacity, the batch size was fixed at

[2] https://www.kaggle.com/ds/3231810

[3] https://github.com/avast/avast-ctu-cape-dataset

TABLE IV: Mean validation set metrics with different vocabulary sizes on malware detection task from Speakeasy data. Reported TPR is at FPR= 10^{-3}.

Metric	5k	10k	30k	50K	70k
TPR	0.8078	0.7834	**0.8576**	0.8383	0.8407
AUC	0.9965	0.9969	**0.9977**	0.9976	**0.9977**
F1	0.9817	0.9839	0.9861	0.9856	**0.9862**
Acc.	0.9753	0.9782	0.9811	0.9806	**0.9814**

$b = 96$ for all experiments. For optimization, we employed the AdamW optimizer [34] with a static learning rate of $\alpha = 2.5^{-4}$. The hyperparameters were set as $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. An L_2 regularization with a weight decay of $\lambda = 1e^{-2}$ was also implemented. The evaluation metrics were derived from three cross-validation (CV) folds on the training set. The reported metrics are the mean values of the three models evaluated on the validation subsets and a single test set. To ensure fair evaluation given the variations in model size as indicated in Table I, we maintained a constant time budget for training instead of a fixed number of epochs. Each fold was allocated a training duration of five minutes, resulting in a total training budget of 15 minutes per cross-validation run for three folds, excluding pre-processing time. Initial experiments with longer training runs, such as an hour per cross-validation, yielded similar relative outcomes with tolerable deviations.As such, the 15-minute training budget was deemed optimal for subsequent experiments.

C. Ablation Studies

We explore here the impact of variations in model components and their configurations on the final performance of the Transformer model. This helps highlight the effectiveness of individual components in the context of the model's overall performance. For our ablation experiments, we use the Speakeasy Dataset as it offers a comprehensive range of behavioral representations. Furthermore, this data enables us to evaluate malware detection performance using a binary classification objective, yielding more interpretable results.

Vocabulary Size. The impact of varying vocabulary size on the performance of the model using the Speakeasy emulation data is presented in Table IV. The results demonstrate marginal differences in performance within the range of vocabulary size $V \in \{30\ 000, ..., 70\ 000\}$, suggesting that performance in this interval is largely subject to the randomness introduced during model initialization and training. This trend suggests that the model's performance is relatively stable with respect to variations in vocabulary size within this range, indicating a degree of robustness to this parameter. Considering these observations, we chose $V = 50\ 000$ as a good compromise that balances performance and complexity.

TABLE V: Mean F1 values of field filter ablation studies on malware detection task from Speakeasy dataset.

Fields	Val. set F1	Test set F1	Δ
Raw JSON (BPE)	0.9884	0.7495	**0.2389**
Raw JSON (whtsp.)	0.9899	0.7275	**0.2624**
Filtered JSON (BPE)	0.9847	0.9136	*0.0711*
Filt. JSON (whtsp.)	0.9870	0.9068	*0.0802*

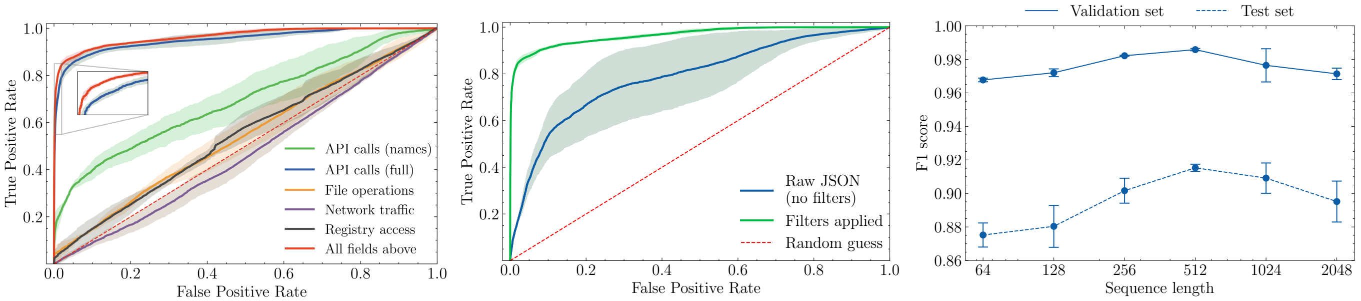
TABLE VI: Mean test set metrics of tokenizer ablation studies on malware detection task from Speakeasy data. Reported TPR is at FPR= 10^{-3}.

Tokenizer	TPR	AUC	F1	Acc.
Wordpunct	0.5540	0.9630	0.9049	0.9041
Whitespace	**0.5703**	**0.9664**	0.9068	0.9053
BPE	0.5213	0.9657	**0.9136**	**0.9104**

Field Filters. Initially, we examine the utility of individual fields for malware detection. Fig. 3a presents the outcomes of experiments in which only a specific single field from the behavioral report is retained. Notably, the most influential component of behavioral representation is the sequence of API calls, especially when arguments are provided alongside the API names. All other fields exhibit inferior performance when considered in isolation. This observation can be rationalized by recognizing that not every type of malware or emulation generates traces in the filesystem, registry, or network – only a limited subset of emulation reports contain this data. However, all samples invariably exhibit a sequence of API calls, which underscores the critical role of API call information in malware detection. However, the inclusion of filesystem, registry, or network information in conjunction with API calls enhances detection capabilities. This synergy enables the model to capture a more comprehensive representation of the software's behavior, improving the accuracy and reliability of its predictions.

Additionally, we investigated two preprocessing modalities: (i) a version that abstains from the application of filters, and (ii) one that incorporates optimal field filters during preprocessing. Table V presents the F1 scores on the validation and test sets of the Speakeasy emulation reports for both the BPE and whitespace tokenization schemes. Remarkably, a significant overfitting issue is present when filters are not employed, evidenced by a difference (Δ) in performance between the validation and test sets. While modeling that employs filters lose about $7\% - 8\%$ of F1 on the test set, the performance of modeling without filters degrades down by $23\% - 25\%$. A visual examination of this trend is depicted in Fig. 3b, where the Receiver Operating Characteristic (ROC) curve for the test set demonstrates significant degradation when filters are not employed. Additionally, the high standard deviation between cross-validation runs suggests a level of model instability or variance in prediction.The observed outcome can be attributed to the presence of unconstrained variables representative of one specific execution, like hash sum or start address memory segment. These fields cause the model to overfit the training data, hindering its generalization and predictive capabilities to unseen data in the test set. Hence, the application of field filters appears instrumental in enhancing model stability and performance, contributing to more reliable and generalizable predictions.

Tokenization. We conducted ablation studies on tokenization to investigate the impact of different tokenization strategies on model performance. Three different tokenizers were tested: BPE [28], Whitespace, and Wordpunct [24]. The test set



(a) Test set ROC of variable filter fields. (b) Test set ROC with and without filter setup. (c) F1 scores with different sequence lengths.

Fig. 3: Results of ablation studies under different configurations on Speakeasy dataset.

TABLE VII: Malware detection metrics on Speakeasy test dataset. Reported TPR is at FPR= 10^{-3}.

Model	Training batches	Test set TPR	 AUC	 F1	 Acc.
Gated CNN [8]	1058	0.2152	0.8879	0.6465	0.7014
Neurlux [3]	7406	0.4250	0.9528	0.8792	0.8786
Quo.Vadis [2]	4761	0.3081	0.9224	0.8065	0.8173
Nebula (BPE)	**2116**	0.5213	0.9657	**0.9136**	**0.9104**
Nebula (whitesp.)	2159	**0.5703**	**0.9664**	0.9058	09053

TABLE VIII: Malware detection metrics on MCD test dataset. Reported TPR is at FPR= 10^{-3}.

Model	TPR	AUC	F1	Acc.
Neurlux [3]	0.8508	0.9942	**0.9687**	**0.9794**
Quo.Vadis [2]	**0.9035**	**0.9950**	0.9613	0.9736
Nebula (BPE)	0.8332	0.9937	0.9653	0.9770
Nebula (whitesp.)	0.8243	0.9932	0.9590	0.9731

F1 scores for different tokenization methods are reported in Table VI. The results reveal that all three tokenization methods deliver comparable mean F1 scores. The BPE tokenizer demonstrates slightly better generalization capabilities, achieving an F1 score on the test set that is almot 1% higher than the others. This observation is further supported by the field filter experiments discussed in the previous paragraph, with results in Table V, where BPE exhibited the smallest performance decrease (Δ) between the validation and test sets. Furthermore, it is noteworthy that the Whitespace tokenizer achieves impressive results on the test set, surpassing the other tokenization methods if evaluated by area under the curve (AUC) or true positive rate (TPR) at false positive rate (FPR) of 10^{-3}, as shown in Table VI. Given competitive performance of BPE and Whitespace, we report metrics of both tokenizers for subsequent malware detection and classification, as well as explainability experiments.

Sequence Length. Fig. 3c depicts the F1 scores on the validation and test sets with varying sequence lengths. The performance on both validation and test sets peaks at a sequence length of $N = 512$. This suggests that sequences of length $N \in \{64, ..., 256\}$ may not encapsulate all the necessary information for effective model inference, leading to a significant drop in test set performance. On the other hand, longer sequences are more computationally demanding, especially when utilizing self-attention-based modeling. Hence, under the same computational time constraints, sequences with length $N \in [1024, 2048]$ yield less robust results.

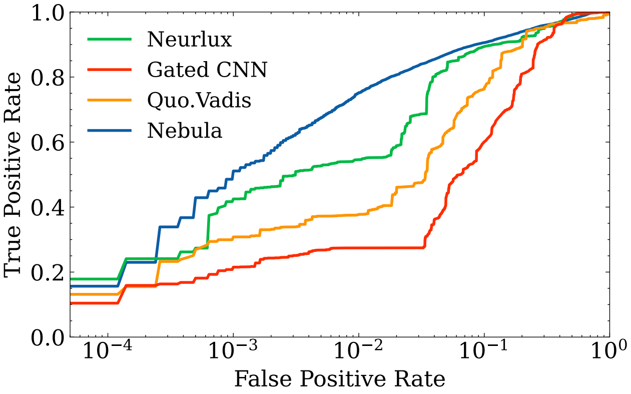


Fig. 4: Mean ROC curves over three cross-validations for malware detection of Neurlux [3], Gated CNN [8], Quo.Vadis [2] and Nebula (with BPE tokenizer) on Speakeasy data test set.

D. Comparison with State of the Art

Malware Detection. In this section, we evaluate the performance of Nebula, with alternative models in the domain of malware detection. Metrics on the Speakeasy dataset [2] reported in Table VII. ROC curve on the test set exemplified in Fig. 4. Four modeling techniques were able to model this data, namely Neurlux presented by Jindal et al. [3], Gated CNN model by Zhang et al. [8], Quo.Vadis released by Trizna [2], and Nebula. Our model surpasses all competitive architectures on Speakeasy emulation data, outperforming all metrics on the test sets in either the whitespace and BPE tokenization modes. This is particularly evident under low false-positive conditions. For instance, with 10^{-3} FPR, Nebula with whitespace tokenization demonstrates 0.570 TPR on the test set.In comparison, the next best performing model, Neurlux, scores 0.42 TPR on the test set.This observation becomes critically significant considering that strict low false positive rates are enforced on production-grade malware detectors [5].

The efficiency of Nebula is also reflected in the number of training batches required. As seen in Table VII, Nebula achieves these results with less than a third of the training

TABLE IX: Mean F1 scores for malware classification objective on Spekeasy dataset.

	Clean	Backdoor	Coinminer	Dropper	Keylogger	Ransomware	RAT	Trojan
Neurlux	0.8453	0.8329	**0.6910**	0.4488	0.2032	0.5527	**0.6625**	0.6153
Gated CNN	0.7588	0.6870	0.5586	0.2015	0.0794	0.3584	0.0000	0.5282
Quo.Vadis	0.8338	0.8520	0.4884	0.3580	**0.2119**	0.6861	0.1195	0.5359
Nebula (BPE)	**0.8526**	**0.8548**	0.6303	0.2850	0.1295	**0.7421**	0.3683	**0.6827**
Nebula (whitsp.)	0.8240	0.8324	0.6214	**0.4615**	0.1179	0.6523	0.1854	0.6486

TABLE X: Mean F1 scores for malware classification objective on Avast-CTU dataset.

	Adload	Emotet	HarHar	Lokibot	Qakbot	Swisyn	Trickbot	Ursnif	Zeus	njRAT
Neurlux	**0.7150**	0.9294	**0.9031**	0.8320	0.9320	**0.9991**	**0.9536**	0.8910	0.6503	0.8479
Nebula (BPE)	0.4390	**0.9392**	0.7763	0.8957	**0.9876**	0.9973	0.9227	0.9362	0.6419	0.8656
Nebula (whitsp.)	0.6975	0.9319	0.8363	**0.9048**	0.9768	0.9984	0.9056	**0.9585**	**0.6690**	**0.8896**

batches required by the second-best model, Neurlux. Turning awareness to the Malware Code Dataset (MCD) [33], the mean validation set metrics are presented in Table VIII. MCD preprocessing is computationally demanding due to the high information density per sample because of lengthy API call traces. This has a detrimental effect on models that employ custom feature engineering schemes, such as Zhang et al. [8], which take several seconds of feature engineering per MCD sample. Processing a training set of 30,000 samples in this manner would take approximately 100 hours—impractical in both experimental and real-world scenarios. Consequently, we excluded this model from our experiments on MCD.

Our observations reveal that Quo.Vadis, a simplistic modeling scheme focused solely on API call names, outperforms both Neurlux and Nebula based on AUC and detection rate under low false-positive conditions. Given the lengthier API call sequences in the MCD data, narrowly focused models like Quo.Vadis might capture more behavior relevant to the data. This outlines the evidence that narrow modeling schemes are still more tuned to this specific data type for specific data sources and can outcompete more general mechanisms.

Malware Classification. Predicting malware family is a multi-label objective, and we report the results of the performances of the considered models in Table IX and Table X the F1 scores on the Speakeasy and Avast-CTU datasets. Due to the lacking of sequential information of the Avast-CTU dataset, we omit Quo.Vadis and Gated CNN from the comparison, as they require temporal information. Thus, we only evaluate this dataset with Neurlux and Nebula. Thus Avast-CTU analysis includes these models only.

Nebula exhibits superior test set F1 scores for 4 out of 7 malware types on Speakeasy (Table IX) data and in 6 out of 10 malware families on Avast-CTU data (Table X). This is particularly noticeable in malware families experiencing significant concept drift, such as polymorphic Emotet [35], in families with many sub-variants, like Zeus [36], or on malware types that exhibit rich and diverse behaviors, such as benignware, backdoors, ransomware, or trojans. *Modus operandi* of such agents require frequent manipulation with network, filesystem, and registry. An examination of metrics on Speakeasy Dataset test set shows that Neurlux still surpasses Nebula in detecting Droppers and RATs, achieving 18% and 30% higher F1 scores, respectively. This might suggest a weakness in Nebula's data-cleaning approach for these particular malware families, indicating a potential avenue for future improvements. Simultaneously, models focusing solely on API calls, for example, Quo.Vadis, exhibit slightly superior performance over the general models for malware families with less diverse behavior, such as in detection of Keyloggers, a malware type that only occasionally interacts with the network or filesystem to store logged keys. Also, we analyze the performances under varying number of target families in malware classification task, depicting the test set accuracies in Fig. 5. Performance of malware classification capabilities drops as number of families grow, suggesting that in practical threat intelligence (TI) applications, it is supposedly better to employ numerous models, each tailored to identify specific key malware families and classes, instead of relying on general classifiers. While the variability of performance for the Avast-CTU dataset remains relatively minor, with only few percentage points of difference, the performance variance on Speakeasy reports notably diverges. Nebula demonstrates notably superior performance to Neurlux, particularly in scenarios involving a smaller number of families, exhibiting at least a 20% accuracy advantage in Nebula's BPE model over Neurlux in tasks encompassing 3-5 target families. This may prove to be particularly valuable in practice, reinforcing the observation for tailored models targeting a lower number of families for optimal performance in TI tasks.

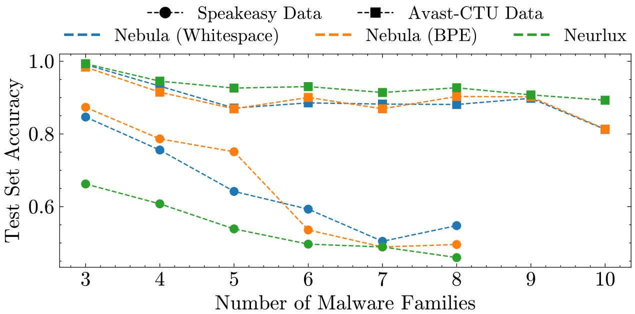


Fig. 5: Test set accuracies with variable number of families used for malware classification task.

E. Self-Supervised Learning Benefits

Since Nebula leverages transformers, we now exploreits capacity for self-supervised learning (SSL), by leveragin unlabeled data to pre-train models. This is achieved through language modeling (LM), with two prominent strategies prevailing in textual data processing: masked language modeling, as exemplified on BERT and related transformer-encoder architectures [13], and autoregressive next-token prediction, characteristic of generative tasks like GPT models [12]. For our study in malware detection, we conduct experiments on both techniques, and we evaluate the performance compared to the fully-supervised settings. Since LM tasks produce logits in size of vocabulary for these experiments we decreased vocabulary size to 8192 for computational reasons. As autoregressive LM requires global attention, we employed a Transformer architecture tailored for these experiments, which discards attention chunking as discussed in Sect. III-C. As for masked LM, we employed the same pre-training parameters as in BERT [13] setup. We designated 80% of the training data as an unlabeled corpus for self-supervised pre-training, while the remaining 20% was allocated for supervised fine-tuning. To provide context, we included two benchmarks as proposed by Apruzzese et al. [6]: (i) an upper bound, representing a supervised model trained on the full dataset with access to all label information, and (ii) a lower bound supervised model that undergoes no pre-training and utilizes only 20% of the training set, akin to the fraction used for LM fine-tuning.

We report ROC curves on test set for all runs in Fig. 6, and we observe a consistent performance pattern across models. As anticipated, the upper bound model exhibits the highest detection rates, while the lower bound model performs the least effectively, with 15% gap between both model detection rates, indicative of the significance of the additional 80% of training data available to the upper bound model. The masked LM model demonstrates the second-worst performance, particularly under the strictest conditions of FPR$= 10^{-3}$, reporting detection rates even inferior to those of the lower bound model. This discrepancy suggests that masked LM pre-training may learn detrimental representations that remain insufficiently adjusted during fine-tuning. In contrast, the autoregressive LM model yields remarkable results, nearly matching the performance of the upper bound supervised model across all FPR ranges and particularly closely aligning with it under the lowest FPR, with only a minimal 3% drop in detection rate. This finding suggests that Nebula can effectively leverage substantially less labeled data by consuming unlabeled samples, thereby reducing human resource requirements and enabling the utilization of vast amounts of PE and DLL files available to community and private businesses in the process.

F. Explaining the Behavior of Nebula

We now explain the behavior of Nebula leveraging two *explainable AI* (XAI) techniques. The first one is *Integrated Gradients* [14], that computes the importance of input features by integrating gradients along a path from a baseline to the input. In our case, we use an empty JSON file as the baseline, which stands for the absence of any behavior. We leverage the GradientSHAP implementation from the SHapley Additive exPlanations (SHAP) framework [37]. Since this technique requires an end-to-end differentiable model, it is not directly applicable in our case due to the presence of the initial embedding layer. To overcome this issue, we extract sample embeddings and obtain explanations from this point onwards, recovering importance values by taking the mean over the embedding dimension. As the second method, we leverage attention activations from the transformer encoder layer to indicate the learned importance of relative token weights within the model. Transformer self-attention layers are multi-headed; in our case, each layer has eight independent heads. We examine all the layers and heads, focusing on the strongest attention weight deviations and investigating their implications. We perform a large-scale analysis, randomly subsampling 1000 samples for each malware type from the Speakeasy dataset. The results are shown in Table XI, from which we can derive the following key aspects:

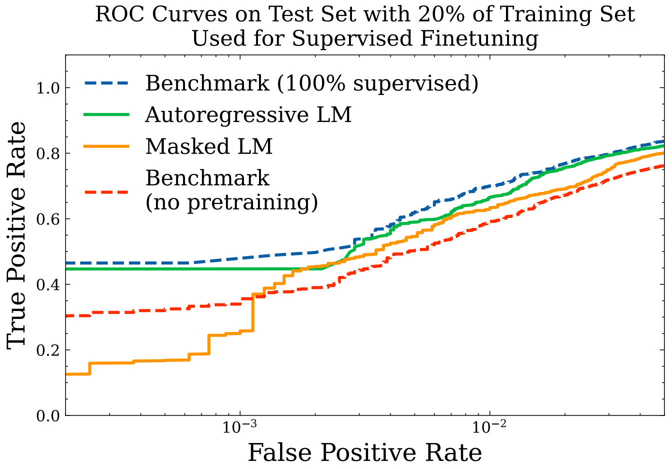
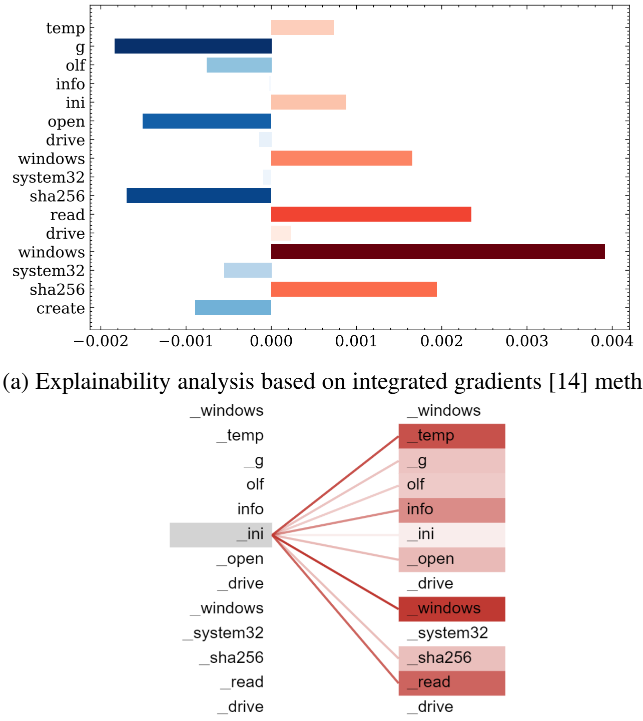


Fig. 6: Self-supervised language modeling (LM) efficiency with 80% of training set used for pre-training and 20% for supervised fine-tuning, compared with two benchmarks without self-supervised LM, representing upper bound with access to all the labels (100% supervised) and lower bound using only fine-tuning data (20% supervised).

- Functions from the *advapi32* DLL exhibit a pronounced significance in malware detection. This library provides functionalities that allow programs to interact with the OS, for instance, seeking elevated privileges or manipulating service controls. The prominence of these functions in our findings underscores their recurrent misuse in malware.
- The token `0xcf0000` is used solely with `CreateWindowEx` API call, referencing the style of spawned window: `WS_OVERLAPPEDWINDOW`. We see that the general trend for malware samples in the test set involves the initiation of user interaction with this specific parameter of UI behavior.
- API calls like `setenvnvar` (alias for *SetEnvironmentVar*) and `getsockobj` (alias of *GetSocketObject*) are frequently among the top tokens for malware classifica-

TABLE XI: Top 10 individual tokens impacting the decision of Nebula towards the target class according to integrated gradients [14] XAI methodology based on 1000 samples per class from Speakeasy dataset. Normalized importance scores for each token are reported in parentheses.

Malware Detection		Malware Classification						
Benignware	**Malware**	**Backdoor**	**Coinminer**	**Dropper**	**Keylogger**	**Ransomware**	**RAT**	**Trojan**
0x406018 (1.00)	**advapi32** (1.00)	0x1610e (1.00)	**0xcf0000** (1.00)	**0xcf0000** (1.00)	0x48e000 (1.00)	0x4013a0 (1.00)	**0xcf0000** (1.00)	0x41b000 (1.00)
0xffe2 (0.90)	readfile (0.65)	0x43204c (0.61)	0x413f64 (0.79)	0x59f934 (0.81)	0x599c24 (0.90)	0x42cb3a (0.71)	0x406018 (0.78)	0xc2b0e (0.92)
0x1211f9c (0.87)	0x1211fd8 (0.07)	0x41c024 (0.60)	0x428084 (0.75)	0x64 (0.81)	**0xcf0000** (0.89)	0x4585c8 (0.71)	0x53b9f8 (0.59)	**0xcf0000** (0.87)
0x481488 (0.85)	0x1211f20 (0.06)	0xffe2 (0.54)	**advapi32** (0.63)	0x4635cc (0.75)	0x5f3c24 (0.80)	0xffe2 (0.54)	0x42a4f1 (0.55)	**getsockobj** (0.78)
0x415000 (0.81)	0x78 (0.05)	0x53b9f8 (0.50)	0x4130d4 (0.57)	0x404008 (0.70)	**getsockobj** (0.67)	0x402378 (0.51)	0x42a730 (0.54)	0xa6ee60 (0.77)
findatoma (0.76)	0x1211f7c (0.02)	0x414004 (0.45)	0x42a730 (0.57)	0x405004 (0.62)	0x1211efc (0.62)	0x40a175 (0.48)	0xffe2 (0.52)	0x404008 (0.61)
getcurthread (0.76)	heapalloc (0.01)	0x425363 (0.30)	0x402378 (0.44)	**getsockobj** (0.55)	0x12f000 (0.57)	0x7340 (0.47)	0x402378 (0.46)	123 (0.51)
0x40b010 (0.75)	0x1db10106 (0.01)	0x101c (0.28)	0x6400000 (0.42)	0x402566 (0.53)	0xde10e (0.55)	**setenvnvar** (0.43)	**setenvnvar** (0.44)	0xffe2 (0.45)
0x414000 (0.75)	kernel32 (0.01)	cb (0.28)	**setenvnvar** (0.42)	0x503008 (0.45)	0xffe2 (0.52)	0xbbc (0.41)	0x7090 (0.39)	0x113000 (0.43)
0x7090 (0.75)	getprocheap (0.005)	0xf0c (0.25)	0x414c34 (0.39)	0x408838 (0.41)	0xfeedf030 (0.52)	0x40c7d1 (0.41)	0x49e5cc (0.33)	0x80000 (0.41)



(a) Explainability analysis based on integrated gradients [14] method.

(b) Attention activations [17] at the second self-attention layer.

Fig. 7: Depiction of fragment from "Urelas" trojan dynamic analysis report exhibiting filesystem interactions. Both explainability technique agree on the importance (red) of tokens like `windows`, `temp`, `read`, all linked to filesystem activities likely exploited by the analysed malware.

tion. This indicates the necessity of malware for frequent manipulation with environment variables, and the need to make network connections, as well as an indication of these manipulations as valuable components for the final heuristic by Nebula.

Notably, while Table XI reports the token importance in isolation, these are used by Nebula's self-attention mechanism in relation to all other tokens in sequence. We manually ensure that the importance assigned by the integrated gradients method from SHAP library [14] is directly correlated with attention weights from within Transformer encoder heads. For instance, we showcase the results of both XAI techniques on a specific sample infected with the "Urelas" trojan (SHA1: *c7ee95f0ea78400d5e4938e06fea1bb0c388b565*) in Fig. 7. We find that both integrated gradients and attention activations identify the highest maliciousness indicators within a particular dynamic analysis segment shown in Fig. 7, pinpointing the same tokens representing filesystem manipulations as highly associated with maliciousness.

V. Related Work

We are not the first to explore Transformer applicability for malware detection, by also discussing the applicability of self-attention only for *static* malware analysis, contrary to our contribution on dynamic malware analysis. Li et al. [39] were the first to propose a Transformer-based architecture for static malware analysis applied on assembly instructions. They used a custom architecture called "Galaxy Transformer" to avoid length limitations and construct hierarchical representations. Rudd et al. [40] explored Transformer applicability on static malware detection applied on raw malware bytes. Influenced by the success of the GPT modeling scheme [12], the authors analyzed Transformer decoder with an autoregressive pretraining objective. Pei et al. [41] apply a hierarchical Transformer for code similarity analysis and vulnerability detection. They generate a dataset from benign Linux ELF binaries, obtaining behavioral micro-traces with QEMU based Unicorn emulator. Similarly to our work, existing approaches [21], [22] explored the usage of transformers applied on sequence of API calls, and comparing them with alternative architectures. However, the application of Transformers customized to a dynamic malware context and applicable to variety of telemetries, distinguishes our approach. Moreover, the comparison with other SotA dynamic malware detectors and our exploration of the model's explainability represent additional, distinct contributions with respect to previous work.

VI. Conclusions, Limitations, and Future Work

In this paper, we present Nebula, a novel self-supervised learning transformer model for dynamic malware detection and classification, and we select its components through an extensive ablation study. We show how much the inclusion of different behavioral aspects manifested by malware improves the performance, by also quantifying how much data cleaning procedure boosts the accuracy at test time. We compare our approach against previously-proposed machine-learning methods for dynamic malware analysis, in a pure supervised learning setting, and we show that Nebula often achieves better results than CNNs and LSTMs. In particular, Nebula surpasses, on average, its competitors on both malware detection and classification on three different datasets. We study how self-supervised pre-training can reduce the need for training data, highlighting that the usage of only 20% of the training dataset is enough to reach state-of-the-art performance on malware detection. Lastly, we inspect the output of Nebula through two explainability methods, and we reveal that Nebula is giving attention to relevant tokens associated with malicious activity, by also exhibiting long spans of attention.

Limitations. We have not considered robustness of our model against adversarial malware [25]. We acknowledge that such analysis would be of high interest, however, adversarial attack on dynamic classifier in input space would require algorithmic modification of malware sample without corrupting the malicious logic. To date, only initial methods of such perturbations have been explored [42], with no released implementation to replicate these attacks. Studying and implementing attacks against behavioral classifiers would stand as a contribution on its own, and for this reason we only discuss them as a constraint of our study. Another limitation of Nebula is reliance on quality of dynamic analysis. Some malicious samples will refrain expressing malicious logic given the execution in virtualized or emulated environments. Techniques focused on sandbox evasion techniques [38] will reduce quality of Nebula even more, emphasizing the need of hybrid heuristic that incorporates signature, static, and dynamic methods given deployment in production setting [2]. Lastly, we caution that pre-trained Nebula models we release were trained on just 70k samples and not a general pre-trained malware detectors with real-world predictive power, instead, valuable for further research and experiments on that same dataset only.

Future work. We plan to further investigate the effect of self-supervised learning on Nebula, pre-training it on a much larger data collection with unlabeled samples, and by varying the size of labelled data. We hence envision models trained with a scarcely populated dataset of novel malware, speeding up computations and keeping state-of-the-art performance that permit the deployment of these novel technologies. Finally, we emphasize importance of assessing the adversarial robustness properties of Nebula, seeing high potential in future work on developing novel attack algorithms tailored to bypass dynamic malware detectors and classifiers.

Acknowledgements

This work has been carried out while Dmitrijs Trizna was enrolled in the Italian National Doctorate on Artificial Intelligence run by Sapienza University of Rome in collaboration with the University of Genova. It has been partly supported by the European Union's Horizon Europe research and innovation program under the project ELSA, grant agreement No 101070617; by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU; and by Fondazione di Sardegna under the project "TrustML: Towards Machine Learning that Humans Can Trust", CUP: F73C22001320007.

References

[1] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. Challenges and directions in security information and event management (SIEM). In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 95–99, 2018.

[2] Dmitrijs Trizna. Quo vadis: Hybrid machine learning meta-model based on contextual and behavioral malware representations. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*, AISec'22, page 127–136, New York, NY, USA, 2022. Association for Computing Machinery.

[3] Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna. Neurlux: Dynamic malware analysis without feature engineering. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 444–455, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Xiaohui Chen, Zhiyu Hao, Lun Li, Lei Cui, Yiran Zhu, Zhenquan Ding, and Yongji Liu. Cruparamer: Learning on parameter-augmented api sequences for malware detection. *IEEE Transactions on Information Forensics and Security*, 17:788–803, 2022.

[5] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical provenance analysis for endpoint detection and response systems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1172–1189, May 2020.

[6] G. Apruzzese, P. Laskov, and A. Tastemirova, "SoK: The Impact of Unlabelled Data in Cyberthreat Detection," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2022, pp. 20-42.

[7] George Karantzas and Constantinos Patsakis. An empirical assessment of endpoint detection and response systems against advanced persistent threats attack vectors. *Journal of Cybersecurity and Privacy*, 1(3):387–421, 2021.

[8] Zhaoqi Zhang, Panpan Qi, and Wei Wang. Dynamic malware analysis with feature engineering and feature learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:1210–1217, 04 2020.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, San Diego, US, 2015.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukas Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, USA, 2017. Curran Associates, Inc.

[12] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. *OpenAI*, San Francisco, CA, June 2018.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1, pages 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota, USA. Report No. N19-1423, June 2019.

[14] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[15] Philip Gage. A New Algorithm for Data Compression. *The C Users Journal*, 1994.

[16] Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen, "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook," in *2012 IEEE Symposium on Security and Privacy*, pp. 65-79, 2012. DOI: 10.1109/SP.2012.14.

[17] Jesse Vig. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, Florence, Italy, July 2019. Association for Computational Linguistics.

[18] Branislav Bosansky, Dominik Kouba, Ondrej Manhal, Thorsten Sick, Viliam Lisy, Jakub Kroustek, and Petr Somol. Avast-CTU Public CAPE Dataset. Avast Software, AI Center, Dept. of Computer Science, Czech Technical University, Prague, 2022.

[19] Simon Mandlik, Matej Racinsky, Viliam Lisy, and Tomas Pevny. Mill.jl and jsongrinder.jl: automated differentiable feature extraction for learning from raw json data. Avast Software, AI Center, Dept. of Computer Science, Czech Technical University, Prague, 2021.

[20] Verizon Communications. Verizon Data Breach Investigation Report (DBIR). https://www.verizon.com/business/resources/reports/dbir/2022/results-and-analysis-intro/, 2022. Online; accessed May 31, 2023.

[21] Ferhat Demirkıran, Aykut Çayır, Uğur Ünal, and Hasan Dağ, "An ensemble of pre-trained transformer models for imbalanced multiclass malware classification," *Computers & Security*, vol. 121, p. 102846, 2022.

[22] Rajchada Chanajitt, Bernhard Pfahringer, Heitor Murilo Gomes, and Vithya Yogarajan, "Multiclass Malware Classification Using Either Static Opcodes or Dynamic API Calls," in *Home AI 2022: Advances in Artificial Intelligence Conference*, December 3, 2022. Pages 427–441.

[23] Alessandro Mantovani, Simone Aonzo, Yanick Fratantonio, and Davide Balzarotti. RE-Mind: a first look inside the mind of a reverse engineer. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2727–2745, Boston, MA, August 2022. USENIX Association.

[24] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[25] Luca Demetrio, Scott E. Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. *Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*. ACM Trans. Priv. Secur. 24(4), Article 27, September 2021. Association for Computing Machinery, New York, NY, USA. ISSN: 2471-2566. DOI: https://doi.org/10.1145/3473039.

[26] Dmitrijs Trizna. Shell Language Processing: Unix command parsing for machine learning. Proceedings of Conference on Applied Machine Learning for Information Security (CAMLIS), 2021, 2021.

[27] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proc. 54th Annual Meeting of the ACL (Vol. 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, 2016. ACL.

[28] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *EMNLP: System Demonstrations*, pp. 66–71, 2018. ACL.

[29] Mandiant. Speakeasy: portable, modular, binary emulator designed to emulate Windows kernel and user mode malware., 11 2021. https://github.com/mandiant/speakeasy.

[30] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *OpenAI*, San Francisco, CA, 2020.

[31] A. Brukhovetskyy and K. O'Reilly. Cape sandbox v2.1 book, 2022.

[32] Cuckoo Foundation. Cuckoo sandbox. https://github.com/cuckoosandbox/cuckoo. Online; accessed May 30, 2023.

[33] Malicious Code DataSet, Jul 2019. https://github.com/kericwy1337/Datacon2019-Malicious-Code-DataSet-Stage1.

[34] Ilya Loshchilov and Frank Hutter. AdamW: Decoupled weight decay regularization. International Conference on Learning Representations (ICLR), New Orleans, US, May 2019.

[35] Cybersecurity and Infrastructure Security Agency. Emotet Malware. https://www.cisa.gov/news-events/alerts/2018/07/20/emotet-malware, Online, Accessed: January 2024.

[36] Brian Krebs. 'Operation Tovar' Targets 'Gameover' ZeuS Botnet, CryptoLocker Scourge. *KrebsOnSecurity*, Online, June 2014.

[37] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *NeurIPS 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[38] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti, "Detecting environment-sensitive malware," in *Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011. Proceedings 14*, pp. 338–357, 2011. Publisher: Springer.

[39] Miles Q. Li, Benjamin C.M. Fung, Philippe Charland, and Steven H.H. Ding. I-MAD: Interpretable malware detector using galaxy transformer. *Computers and Security*, 108:102371, 2021.

[40] Ethan M. Rudd, Mohammad Saidur Rahman, and Philip Tully. Transformers for end-to-end infosec tasks: A feasibility study. In *Proceedings of the 1st Workshop on Robust Malware Analysis*, WoRMA '22, page 21–31, New York, NY, USA, 2022. ACM.

[41] Kexin Pei, Zhou Xuan, Junfeng Yang, Suman Jana, and Baishakhi Ray. Learning approximate execution semantics from traces for binary function similarity. *IEEE Transactions on Software Engineering*, 49(4):2776–2790, 2023.

[42] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against state of the art api call based malware classifiers. In *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21*, pages 490–510. Springer, 2018.

Dmitrijs Trizna is a Senior Security Researcher at Microsoft Corporation, and a Doctoral Researcher at SmartLab, University of Genova. He has ten years of experience in commercial cyber-security (both blue and red teaming), published research at industrial security conferences like BlackHat US and DefCon (AI Village), and scientific venues like CAMLIS, ACM CCS AISec. Dmitrijs received security certifications like OSCP, SANS (GREM, GDAT), CCNA Security, Standford Online, etc., and participated in cybersecurity trainings organized by NATO.

Luca Demetrio (MSc 2017, PhD 2021) is an Assistant Professor at the University of Genoa. He is currently studying the security of Windows malware detectors implemented with Machine Learning techniques, and he is first author of papers published in top-tier journals (ACM TOPS, IEEE TIFS). He is part of the development team of SecML, and the maintainer of SecML Malware, a Python library for creating adversarial Windows malware.

Battista Biggio (MSc 2006, PhD 2010) is Full Professor at the University of Cagliari, Italy. He has provided pioneering contributions in machine learning security, playing a leading role in this field. His seminal paper on "Poisoning Attacks against Support Vector Machines" won the prestigious 2022 ICML Test of Time Award. His work on "Wild Patterns" won the 2021 Best Paper Award and Pattern Recognition Medal from Elsevier Pattern Recognition. He has managed more than 10 research projects, and serves as a PC member of ICML and USENIX Security, and as Area Chair of NeurIPS. He chaired IAPR TC1 (2016-2020), and served as Associate Editor for IEEE TNNLS, IEEE CIM, and Elsevier PRJ. He is now Associate Editor-in-Chief for PRJ. He is also a senior member of IEEE and ACM, and a member of IAPR and ELLIS.

Fabio Roli received his Ph.D. in Electronic Engineering from the University of Genoa, Italy. He was a research group member of the University of Genoa ('88-'94), and adjunct professor at the University of Trento ('93-'94). In 1995, he joined the Department of Electrical and Electronic Engineering of the University of Cagliari, where he is now Full Professor of Computer Engineering and Director of the Pattern Recognition and Applications laboratory (https://pralab.diee.unica.it/). He is partner and R&D manager of the company Pluribus One that he co-founded (https://www.pluribus-one.it). He has been doing research on the design of pattern recognition and machine learning systems for thirty years. He was a very active organizer of international conferences and workshops, and established the popular workshop series on multiple classifier systems. Dr. Roli is Fellow of the IEEE and of the IAPR.