

A Systematic Comparison of Large Language Models Performance for Intrusion Detection

Original

A Systematic Comparison of Large Language Models Performance for Intrusion Detection / Bui, M., Boffa, M., Valentim, R.V., Navarro, J.M., Chen, F., Bao, X., Houidi, Z.B., Rossi, D.. - In: THE PROCEEDINGS OF THE ACM ON NETWORKING. - ISSN 2834-5509. - ELETTRONICO. - 2:CoNEXT4(2024), pp. 1-23. [10.1145/3696379]

Availability:

This version is available at: 11583/2997588 since: 2025-02-18T16:50:44Z

Publisher:

ACM

Published

DOI:10.1145/3696379

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

ACM postprint/Author's Accepted Manuscript

© ACM 2024. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in THE PROCEEDINGS OF THE ACM ON NETWORKING, <http://dx.doi.org/10.1145/3696379>.

(Article begins on next page)

A Systematic Comparison of Large Language Models Performance for Intrusion Detection

MINH-THANH BUI*, Huawei Technologies Co. Ltd, France

MATTEO BOFFA*, Politecnico di Torino, Italy and Huawei Technologies Co. Ltd, France

RODOLFO VIEIRA VALENTIM, Università di Torino, Italy and Huawei Technologies Co. Ltd, France

JOSE MANUEL NAVARRO, Huawei Technologies Co. Ltd, France

FUXING CHEN, Huawei Technologies Co. Ltd, France

XIAOSHENG BAO, Huawei Technologies Co. Ltd, France

ZIED BEN HOUIDI, Huawei Technologies Co. Ltd, France

DARIO ROSSI, Huawei Technologies Co. Ltd, France

We explore the capabilities of Large Language Models (LLMs) to assist or substitute devices (i.e., firewalls) and humans (i.e., security experts) respectively in the detection and analysis of security incidents. We leverage transformer-based technologies, from relatively small to foundational sizes, to address the problem of correctly identifying the attack severity (and accessorially identifying and explaining the attack type). We contrast a broad range of LLM techniques (prompting, retrieval augmented generation, and fine-tuning of several models) using state-of-the-art machine learning models as a baseline. Using proprietary data from commercial deployment, our study provides an unbiased picture of the strengths and weaknesses of LLM for intrusion detection.

Additional Key Words and Phrases: Intrusion Detection; Large Language Model;

1 Introduction

Current security products like Intrusion Detection System (IDS) and End Point Detection and Response (EDR) systems largely rely on manually curated rules and Cyber Threat Intelligence (CTI) data for analyzing raw logs. However, this approach is continuously challenged by the dynamic nature of cyber threats, necessitating constant updates and manual interventions. It would be thus desirable to reduce as much as possible the current dependence on manual processes and rule-based systems and to assist the expert in analyzing a vast amount of heterogeneous data – for which Machine Learning (ML) and, especially, Natural Language Processing (NLP) techniques are a promising tool. While several known problems [10] plague the deployment of ML in the production system, breakthrough in NLP modeling offers a promising avenue worth exploring, given the generic reasoning abilities of Large Language Models (LLMs) [16], whose application to security recently encountered an increasing interest. In this paper, we investigate whether LLMs can detect malicious patterns *directly within raw packets*.

From a mile-high viewpoint, there are two main ways LLMs can be exploited for cybersecurity defense. The first one is to use (i) *general-purpose* polymath LLMs [16] to reason about security events: this is, for instance, the case of products like Microsoft Security Copilot [3], which employs OpenAI’s GPT-4 to produce security incident reports. At the same time, such a cloud-centric deployment model faces significant hurdles in terms of data privacy and high bandwidth and computational costs. The alternative path is to rely on (ii) *lean task-specific* LLM models, that have

*These authors contributed equally to this work.

been retrained/fine-tuned in a supervised manner to solve specific tasks, such as identification of an attack’s class and severity. While the scale of such models allows them to be run on-premise, it is unclear if such models are sufficiently accurate to replace, or even assist, current IDS.

This paper sets out to assess the value of the above LLMs alternatives by comparing several design strategies, including classic ML and LLM models that, given raw data as input, compete (or complement) a commercial IDS solution to generate security alarms and incident reports. In particular, we consider: (i) an end-to-end approach where the LLM provides the full solution in a single step with classic few-shot prompting; (ii) a Retrieval Augmented Generation (RAG) strategy where the LLM has access to a database of exemplary raw payloads of previous attacks; (iii) a decoupled solution where a small, task-specific fine-tuned LLM classifies the attack class and severity, whereas a foundational LLM produces the ultimate natural language incident report, aggregating the task-specific output with additional information, such as CTI data. We summarize our main findings as follows:

- First, prompt-based end-to-end LLM solutions are not satisfactory; in particular, cloud-based solutions constitute a privacy risk without readily solving the problem (OpenAI GPT 3.5 and 4), while on-premise solutions with smaller models achieve below-par performance (Mistral-7B, Llama2-13B).
- Second, RAG improves the on-premise end-to-end solution by a sizeable amount, and additionally helps improve the attack explanation and grounding, but performance is still far from replacing an IDS.
- Third, task-specific LLMs can be fine-tuned to achieve satisfactory performance, over 95% accuracy for in-distribution attacks, outperforming state-of-the-art ML baselines.
- At the same time, performance on zero-day attacks (i.e., out-of-distribution from a machine learning perspective) is significantly lower, which calls for more investigation into the generalization abilities of LLMs.

In the rest of this paper, we first overview related literature (Sec. 2) and next detail the different system designs we contrast (Sec. 3). We describe our research questions, proprietary dataset, and methodology (Sec. 4), and report extensive experimental results (Sec. 5). Finally, we discuss limitations and future work (Sec. 6).

2 State of the art

2.1 NLP for Cyber Defense

Various NLP techniques have been used for the detection of malicious patterns in textual data from diverse cybersecurity sources (such as emails, transaction logs, software code, and online social media). As summarized in the top portion of Table 1, these techniques can be categorized into four main approaches, which we overview in the following.

2.1.1 Traditional NLP Methods. Traditional NLP methods, such as Bag of Words (BoW), Term-Document Matrices (TDMs), and Term Frequency-Inverse Document Frequency (TF-IDF) [52], have been employed to represent text in a digital form, usually in combination with ML for various cybersecurity tasks. For instance, [11] use BoW with a Decision Tree for detecting intrusion attacks in in-vehicle networks, specifically focusing on the Controller Area Network (CAN) bus traffic and considering different combinations of data including the arbitration field (CAN-ID) and payload. Leveraging honeypot logs, [13] uses frequency-based embedding (Counter Vectorizer, TF-IDF) to identify classes of attack patterns and explain attackers’ objectives. *In this work, we leverage state-of-the-art ML and TF-IDF as a baseline for LLM.*

Table 1. State-of-the-art methods for cybersecurity using NLP, Word Embedding, Transformer, and LLM.

Application	Sec	Approach	[Ref] Year	Model	Task
Defense	2.1.1	Traditional NLP	[11] 2021	BOW + Decision Tree	Intrusion detection
		Traditional NLP	[13] 2022	Counter Vectorizer, TF-IDF, Word2Vec + Clustering	Shell attacks analysis
	2.1.2	Word Embedding	[33] 2021	DeepAMD (MLP)	Android malware detection
		Word Embedding	[36] 2021	PetaDroid (CNN Ensemble)	Android malware detection
		Word Embedding	[27] 2021	DarkVec (MLP)	Malware traffic detection
	2.1.3	Transformer	[50] 2021	MalBERT (BERT-based)	Malware identification
		Transformer	[51] 2023	MalBERTv2 (BERT-based)	Malware identification
		Transformer	[7] 2022	SecureBERT (BERT-based)	Foundation model for security
		Transformer	[9] 2022	CAN-BERT (BERT-based)	Intrusion detection
		Transformer	[18] 2022	BERT-Log (BERT-based)	System logs Anomaly detection
		Transformer	[14] 2023	LogPrecis (BERT-based)	Honeypot shell log analysis
		Transformer	[54] 2024	Dom-BERT (BERT-based)	Malicious domain detection
	2.1.4	LLM	[3] 2023	SecurityCopilot (GPT-4)	Incident response
		LLM	[23] 2023	SecureLLM (BERT, Falcon-40B)	Threat detection and mitigation
		LLM	[22] 2023	SecureFalcon (Falcon-40B)	VC code vulnerability detection
LLM		[39] 2023	CAN-SecureBERT (RoBERTa), CAN-LLAMA2 (Llama2-7B)	Intrusion detection and classification	
LLM		[35] 2023	ChatIDS (ChatGPT)	Explaining IDS alerts	
LLM		[31] 2023	netFound (Hierarchical transformer)	Foundation model for network security	
Attack	2.2.1	LLM	[21] 2024	GPT-4	Website hacking
		LLM	[47] 2023	AutoGPT, GPT-3	Malware creation
		LLM	[32] 2023	GPT-3.5 GPT-4	Spear-phishing attacks
LLM Security	2.2.2	LLM	[28] 2024	GPT-3.5, GPT-4	LLM apps manipulation
		LLM	[17] 2020	GPT-2	Training data extraction
		LLM	[44] 2023	Pythia, GPT-Neo, LLaMA, Falcon, ChatGPT	Training data extraction
		LLM	[43] 2023	Small check	Leak of LLM internal design
		LLM	[40] 2023	Bart, mBart	LLM "intellectual property" protection

2.1.2 Neural Word Embeddings. Neural word embeddings, such as Neural-Bag-of-words, FastText and Word2Vec [42], are more modern word representations that capture language semantics and word inter-relationships. These embeddings are generally used in conjunction with Deep Learning (DL) architectures for malware detection tasks. For instance, [33] proposed the DeepAMD framework based on a simple Multi-layer Perceptron (MLP) architecture for detecting and identifying Android malware. PetaDroid[36] employs an ensemble of Convolutional Neural Networks (CNNs) on top of *Inst2Vec* features for Android malware detection, and uses DBScan for clustering malware families. DarkVec [27] learns embeddings of IP traffic patterns to detect malicious network activities. *As word embeddings have been superseded by more recent neural architectures, we disregard them in this paper.*

2.1.3 Transformer-based Language Modeling. Transformer-based language models, particularly encoder-based models like Bidirectional Encoder Representations from Transformers (BERT) [20], have gained popularity in cybersecurity applications due to their ability to learn good, contextualized representations from words and entire sentences. Several studies have applied BERT to various security tasks, such as intrusion detection [9], anomaly detection in system logs [18], malware identification [50, 51] and malicious DNS domain detection [54]. Domain-specific language models like SecureBERT [7] have been developed to learn representations from the unique characteristics of cybersecurity text data. *Given the rising popularity of BERT-based security models, in this paper we perform a thorough ablation study of BERT models for threat detection and classification.*

2.1.4 LLM for Cyber Defense. The emergence of LLMs has opened new possibilities for automating cybersecurity tasks. While the application of LLMs in this domain is still in the early stages, recent works have started exploring its potential. Ferrag et al. [23] introduced SecurityLLM combining a smaller BERT model for threat detection in IoT systems with a larger instruction-tuned LLM for incident response and recovery, acting as an assistant to network security analysts. To detect vulnerabilities in the specific case of C code, [22] utilize a tailored version of FalconLLM, achieving good accuracy. Other efforts include using LLMs for explaining IDS alerts to non-experts [35]

and building foundation models for network security [31] to incorporate hierarchical and multi-modal attributes of network traffic inside the model, to be able to learn hidden contexts and favor generalization. To the best of our knowledge, despite RAG being a very popular technique [24], it has never been tested so far in the field of cybersecurity. *In this work, we are the first to directly compare transformer-based vs LLM-based and RAG-based classification performance (and defer a qualitative assessment of LLM explanation abilities to Appendix C).*

2.2 Other LLM-related Security Studies

As reported in the bottom of Table 1, LLMs have also been used for complementary cybersecurity tasks: these are still relevant as close in terms of approach, yet far from an application perspective.

2.2.1 LLM for Cyber Attacks. Researchers have studied the offensive capabilities of LLMs such as AutoGPT and GPT-3, to create malware [47] or spear-phishing attacks [32]. More recently, Fang et al. [21] assessed the ability of LLM agents to autonomously hack websites, revealing that GPT-4 demonstrates the required capabilities without explicit prior knowledge of specific vulnerabilities (succeeding 2/3 of the time w.r.t. much lower success rates for other models, including GPT-3.5).

2.2.2 Security of LLMs. A complementary viewpoint of the work on LLMs and security is instead concerned with the security and privacy of LLM-based systems themselves. Greshake et al. [28] notoriously revealed a new attack vector called Indirect Prompt Injection, where adversaries remotely exploit LLM-augmented applications by injecting prompts into the data these applications access during inference.

Other work instead focuses on attacks to steal training data from LLMs [17, 44], attacks to reverse-engineer some of their internal design choices [43], methods to protect their “intellectual property” through watermarks [40], or methods that prove that a model was trained according to a given specification without revealing details about training data or model details [26].

2.3 Our Contributions

In a nutshell, this study is the first to provide a systematic comparison of a very wide range of state-of-the-art LLM and ML techniques for the purpose of cyberdefense, using payload as input. To provide fair, unbiased, and statistically significant results, we take care of avoiding methodological flaws that, while well known, are still common in ML studies for security [10].

3 Solutions overview

The goal of an AI firewall is, shortly, to (i) correctly detect a security incident and (ii) generate a description in natural language: Fig. 1 depicts the high-level solutions we contrast in this work to achieve the above goal. For the reason of space, this paper mostly discusses the *quantitative* aspects related to *classification* abilities and defers to the appendix qualitative examples of textual description. Without loss of generality, we refer to the categorical class labels (e.g., benign/malicious, or more involved class definition) as ℓ in the following.

3.1 Prompt engineering

As a baseline, we employ a frozen pre-trained LLM with prompt engineering as in Fig. 1-(a). We opt for open-weight LLM models (such as Mistral [34] and Llama2 [55]) that can be deployed on-premise in a private cloud (for data privacy reasons), or on the public cloud (with access to GPT-3.5 [12] and GPT-4 [46]). The input of a model is raw packets P payload, and the output of the model is the class label ℓ (and a natural language explanation \mathcal{E}).

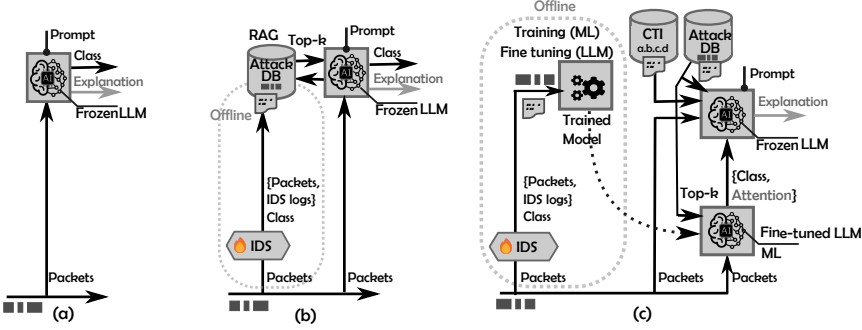


Fig. 1. Synoptic of AI firewall solutions: (a) end-to-end solution through frozen LLM prompting; (b) RAG and frozen LLM; (c) decoupled classification with fine-tuned LLM (or ML) and incident reporting with frozen LLM.

Aim. The aim is to assess whether frozen LLM models alone can readily act as an AI firewall with minimal investment. As such, while we do experiment with several prompts, we do not aim at over-engineering the prompting part [57]. We further detail the frozen LLM baseline in Sec.4.4.1 and report results in Sec.5.1.1

3.2 Retrieval Augmented Generation (RAG)

As a second solution, we complement the frozen LLM with firewall data, accessible through a RAG [25] as in Fig.1-(b).

Offline phase. We leverage ChromaDB and Langchain frameworks [2] to augment frozen LLMs with specific examples of malicious packet payloads P and associated classes ℓ , denoted as AttackDB in the picture. During an offline phase, the AttackDB is populated with a set of representative attacks by using embeddings of the payloads $e(P_i)$ (essentially, a high dimensional vectorial representation of the payload), along with metadata such as $\forall \ell_i$ class label, and other useful information (such as a textual description $eventName_i, eventID_i$).

Inference phase. At inference time, when presented with a new raw packet payload P_j , using $e(P_j)$ as the search key, RAG retrieves the top- k relevant payloads embeddings to $e(P_j)$ and the associated values (i.e., class ℓ , $eventName$ descriptions). In practice, the top- k packets are those closest according to the cosine similarity metric in the embedding space. The metadata (class ℓ , $eventName$ description) associated with these top- k embedded payload representations are then passed as additional input to a frozen LLM model for textual explanation. We expect this domain-specific knowledge to assist a fine-tuned LLM or a frozen LLM for the classification (ℓ) or textual explanation of security events.

Aim. Setting up a RAG pipeline is marginally more involved than prompting: the aim of this baseline is thus to assess whether, by carefully engineering a sanitized task-specific AttackDB exploited with a RAG approach is sufficient to let frozen LLM models act as an AI firewall. Further details on the RAG strategy are provided in Sec. 4.4.2 and associated results are reported in Sec. 5.1.2.

3.3 Fine-tuned LLMs + Frozen LLMs

We finally consider a decoupled solution where a (i) specialized LLM (or ML) model provides accurate classification ℓ , whereas a (ii) foundational frozen LLM model is solely in charge of providing the natural language explanation \mathcal{E} .

Offline phase. Task-specific LLM (or classic ML model) needs to be fine-tuned during the offline phase. We consider both (i) transformer-based models e.g., BERT [20] and variations thereof (such as UnixCoder, BigBird) of alternatives (such as models of GPT-2, Mistral-7B) as well as (ii) state-of-the-art ML models based on classic TF-IDF payload representation as a baseline. To provide a fair comparison, we fine-tune LLM and train ML models on the same (P, ℓ) class pairs (cfr Sec.4.4.3–4.4.4 and Appendix B.1–B.3).

Inference phase. During the inference phase, specialized LLM (or ML) models can feed to the foundational frozen LLM model more than just the class label ℓ (e.g., additional meta-information such as `eventName`, attention, extracted keywords, etc.). Additionally, the foundational frozen LLM model can access and summarize *all* available information (such as the AttackDB through RAG, CTI information about IP of the alleged attackers, as well as raw packet payload P as in the previous solutions) to present human operators with a clear explanation of the event in natural language – which instead specialized task-specific models are unapt for.

Aim. The intuition of this decoupled solution is to leverage a lean lightweight model for the classification task of large volumes of traffic and exploit the expressive power of LLM only for the rare cases that require human intervention. We introduce the state-of-the-art ML approaches in Sec.4.4.3, the set of LLM models we fine-tune in Sec.4.4.4, and report results in Sec.5.2.

4 Methodology

In this section, we formalize our methodology and key research questions (RQs) we aim to address (Sec. 4.1), introduce our dataset (Sec. 4.2) and evaluation scenarios (Sec. 4.3). We then detail our classification models (Sec. 4.4). Due to lack of space, we defer the explanation generation to Appendix C.

4.1 Problem statement

We frame the task of detecting security incidents from raw network traffic as a *sequence classification* problem in the context of NLP. Given an input text sequence representing parsed network traffic (e.g., a packet header and payload), the goal is to predict the correct label ℓ as either a binary classification (malicious/benign) or as a 5 category (fine-grained risk level, cfr Sec.4.2 and Appendix A.2). For the sake of readability, and to gather *conservative results*, we mostly report results of the 5-category classification (which is harder than the binary problem).

At high level, our experiments are designed to address the following key RQs, that we phrase to provide insights and best practices for designing effective LLM-based network security incident detection systems.

- RQ1** *Ease of adoption:* Are simple techniques such as prompting/RAG sufficient, or is fine-tuning LLM models necessary, for accurate classification of security events?
- RQ2** *LLM performance:* Do fine-tuned LLMs offer advantages over traditional ML baselines, such as TF-IDF with state-of-the-art ML models? Are they capable of generalization? what is the expected performance in practice?
- RQ3** *Best LLM practices:* Concerning fine-tuned LLMs, what are the best choices in terms of LLM *model sizes* (from 110M to 7B weights), *context window* (from 512 to 4096 tokens) and *pre-training* (language-only vs domain-specific)?

4.2 Dataset

For this paper, we use a proprietary* collection of security events, corresponding to incidents detected by firewall rules from our on-premise as well as from customer deployments, collected across five months (from May 26th to October 10th, 2023). By design, the provenance of the data avoids the *Lab-only evaluation* pitfall, identified in [10] as the 3rd most frequent problem, common to 47% of the surveyed ML security studies.

4.2.1 Data sources. Each event is described by two data types: an (i) *Alarm Log*, a single JSON file giving the basic details of the type of attack identified (e.g., event ID, 5-tuple, application protocol, event name assigned by IDS, timestamps, etc.) and (ii) *Alarm Evidence*, a collection of JSON files containing plaintext payload dumped from Packet Capture (PCAP) associated with the event in the *Alarm Log*. As our goal is to apply ML directly to traffic captures, we restrict our attention to the set of 2.06M events having associated at least one packet capture with a non-empty payload. As it is well known that most IDS alarms are false positive classifications [8], the main goal is to correctly rank attacks into 5 decreasing severity levels ℓ : some classes pertain to malicious traffic (1: *Successful attack*, 2: *Virus, trojan and worm* and 3: *Unsuccessful attack attempt*) while other classes are related to either benign (4: *IDS false alarm*) or unspecified (5: *Other*) traffic. Additional details concerning data collection process are reported in Appendix A.1.

4.2.2 Spatio-temporal viewpoint. In particular, there are 232 applications in total in the dataset, with the top 5 (HTTP, DNS, SMB, HTTPS and UDP) representing 88% of the total events – an expected imbalance. These applications generate a set of precisely 2500 unique attack types (identified by their textual `eventName` description) for which their most frequent top-5 (top-10) represent 26.8% (35.6%) of the total. Further information about the attack types and details on the top-10 events are deferred to Appendix A.2.

As the data comes from active firewall deployments, we do not control the collection policies: as such, we cannot assume homogeneity across time in terms of the types of events captured. This is especially relevant for the study of deployments that need to adapt to evolving data [10]. We characterize the temporal evolution of events in Fig. 2, showing the cumulative distribution of events over time (left) as well as heatmaps by event names/attack severity according to months (middle/right), where we split the data in five bins of equal number of days (27.5). Regarding attack severity level ℓ (right), we observe a large imbalance in both spatial distribution and temporal behavior: successful attacks are very infrequent (severity class $\ell=1$ comprises 787 events, representing less than 0.4% of the total) while most of the false positive and other traffic samples ($\ell \in [4,5]$) are concentrated in the third and fourth bins. In particular, most of the category $\ell=4$ events are detected in a period of four days, ranging from Aug. 15th to 19th: this bursty behavior is expected and represents a potential challenge for properly training ML and LLM models.

Temporal skew can similarly be found analyzing event names (middle): most SMB attacks (top-1 and top-3 in terms of frequency, attacks of severity $\ell=2$) concentrate in Jun-Jul, while SSL weak hashing scans (top-4 frequency, severity $\ell=4$) appear to be carried out mostly during Aug-Sep. This confirms the potential data drift that was previously hinted, as completely new types of attacks (i.e., zero-day previously unseen `eventName`) may start to appear after months of collection, which has consequences on the evaluation protocol to be put in place to avoid gathering biased results.

4.2.3 ML viewpoint. From an ML viewpoint, information in the Alarm log and Alarm evidence can be either used as input or output of the classification task. Generally speaking, the ML classifiers are presented with input readily available from the Alarm evidence, i.e., the 5-tuple flow identifier

*The dataset is a sensitive asset which we cannot therefore release. We are investigating the possibility of releasing a curated fraction, but we have not received clearance to do so.

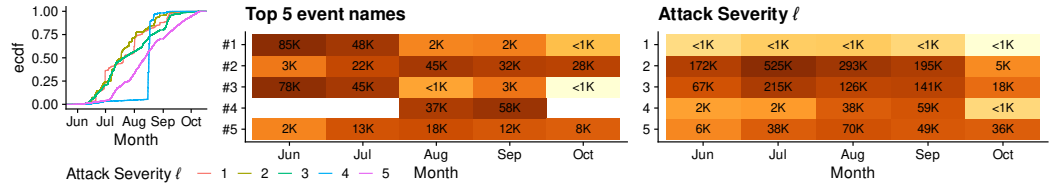


Fig. 2. Temporal evolution of the volume of events across categories (left), top-5 event names (middle, corresponding to the first five events in Table 8), and event categories (right). The fill color indicates the percentage of samples each cell represents and is independent across the two plots.

as well as the packet payload extracted from the raw traffic stream. When more than one packet is available, we further *concatenate* payloads of several packets belonging to the same event (i.e., associated to the same `eventID` in the log). As for the output, our classification target is the *severity of the identified attack* $\ell \in [1, 5] \subset \mathbb{N}$, where the lower the value, the higher the severity.

Additionally, the IDS provides a *textual description* of the attack (`eventName`): we consider the `eventName` field as either (i) input to ML/LLM classifier for IDS-assistance, as well as (ii) intermediate output of the RAG classifier for IDS replacement. Concerning the latter case, recalling Fig.1-(b,c), the AttackDB can be queried through RAG to predict the most likely `eventName`, to be used as input by the risk severity classifier even in the absence of an IDS.

4.3 Data Splits and Evaluation Scenarios

To rigorously evaluate the performance and generalization capabilities of our AI Firewall, we curate our dataset of real-world network events to design several evaluation scenarios that avoid common pitfalls [10]. Table 2 summarizes the number of security events used for model training and testing, according to different splits we hereby motivate and describe. In a nutshell, we use a (i) *curated subset* with models of the OpenAI GPT family due to privacy issues, (ii) a *stratified split* to assess different models’ capacity to learn from the data, isolating model limitations from the effects of potential data shifts and (iii) two *time-based splits*, to closely assess models’ robustness to real-world conditions and data shifts.

Table 2. Dataset and associated computational cost: (left) Number of samples for stratified vs temporal splits and (right) training and inference (batch size 1) costs for ML, BERT and RAG models for the stratified split.

Data splits	Stratified	15/08-split	21/08-split	Computational cost	ML	BERT	RAG
Train	17174	42245	42687	Train time (sec/sample)	0.25	0.7	0.02
Validation	4294	10562	10672	Inference time (ms/sample)	0.07	40.3 [†] /5.5 [‡]	34.4
Test	411626	575414	406449	Inference rate (sample/sec)	13.7k	25 [†] /181 [‡]	29

[†] batch size B=1, [‡] batch size B=20

4.3.1 Curated subset. A balanced subset of 20 samples per class (100 overall), on which we gather clearance for testing with proprietary LLMs (i.e., OpenAI GPT) while preserving privacy, that we use to partly answer **RQ1**. Given its small size, the subset has anecdotal value and yields ballpark figures, but it allows manual validation of results.

4.3.2 Stratified selection. Stratified selection is typically used in ML for consistent performance evaluation, in that, it avoids the *sampling bias* present in many ML security studies (the most frequent problem, common to 60% of the studies surveyed in [10]).

Typically, we start by randomly splitting the full dataset into 80/20 for train/test purposes. To counter class imbalance in the training process, we create a fairly balanced subset by subsampling the train fold, ensuring the selection of at least one sample per each of the (`eventName`, `application`, `ℓ`) tuples and oversampling the minority classes. This yields a more balanced class distribution (despite successful attacks are still under-represented). The train subset is further split into train/validation sets with an 80/20 ratio, and the same train/validation folds are used for both LLM and ML workflows.

Finally, from the much larger portion (0.4M samples) of the original dataset left for test purposes, we ensure that no duplicate event (i.e., repeated attack from the same source with the same payload) is present in both the training and testing sets. We use stratified split to relatively compare ML vs LLM in **RQ2**, and for ablation studies of **RQ3**. We are aware that, unlike the train/validation sets, the test set will still be affected by class imbalance: therefore, we will avoid the *inappropriate measure* pitfall (top-5 flaw present in 33% of the ML security studies [10]) by resorting to unbiased metrics (e.g., weighted accuracy and macro average F1 score) and reporting confidence intervals over multiple repetitions. The weighted accuracy (or balanced accuracy) is a metric that accounts for class imbalance by computing the average of recall values for each class, computed $\frac{1}{C} \sum_{i=1}^C \frac{TP_i}{TP_i + FN_i}$, where C is the number of classes, TP_i and FN_i are the true positives and false negative for class i .

4.3.3 Time-based splits. At the same time, stratified selection (or cross-fold validation) breaks temporal consistency and yields *temporal data snooping* from future events – a prevalent flaw in ML security (the second most frequent problem, found in 57% of the studies [10]). We therefore design two more realistic scenarios, where ML/LLM models predict future events based solely on past data, without control over the event distribution. This mimics the conditions of a real-world deployment, testing the models’ ability to adapt to evolving threats.

We resort to temporal splits that conserve roughly the same magnitude of the 80/20 stratified splits: in particular, our dataset consists of network events collected during 138 days, with a strong data drift happening during mid-august as reported in Fig.2. We therefore consider two temporal splits, namely: (i) *15/08-split*: this split uses 82 days for training (until August 15th, so just *before* the drift) and the remaining 56 days for testing; (ii) *21/08-split*: this split uses 88 days for training (until August 21st, so just *after* the drift) and the remaining 50 days for testing. Clearly, (i) corresponds to a worst-case stress-testing for the generalization capability of the model, as the test split contains radically different zero-day attacks never seen in training (i.e. out-of-distribution), while (ii) corresponds to a mild stress-test*, yielding a conservative performance assessment that is furthermore not affected by temporal data snooping.

After the temporal split, a similar subsampling approach is used to reduce the training/validation set sizes, whereas the test set corresponds to all security events happening after the split date. We extensively use the time-based splits to gather a conservative assessment of ML and LLM performance (**RQ2**) in real-world settings throughout the paper.

4.4 Classification Models

4.4.1 Prompt engineering. We use few-shot prompting as a frozen-LLM baseline. We use a range of pre-trained LLM models of different size and openness, notably open-source LLM with on-premise deployment model that keeps data private (Llama2 7B and 13B [55], and Mistral 7B[34]) as well as proprietary LLMs with cloud-based API access that exposes data to third parties (OpenAI GPT-3.5 and GPT-4 [46]). For each input event, we construct a prompt that includes the task description, the input payload, as well as instructions and examples to classify the event into one of the five

*In that models are trained once and never updated, whereas in practice we would expect at least a weekly/monthly model update.

categories: the model output is then parsed to extract the predicted label. While we experiment with a set of prompts (the finally used prompts are reported in Appendix B.2), we do not want to invest too much time in prompt (over) engineering [57]: our goal is to assess the current level of ease of deployment with limited effort (**RQ1**), i.e., if the problem is already solved by the most powerful models available in the academia/market. When leverage frozen models, the running time of a single inference is around 10 seconds.

4.4.2 Retrieval Augmented Generation (RAG). For our RAG baseline, we employ a dense passage retrieval system implemented using ChromaDB with state-of-the-art embedding models [5, 6]. While the full system employs RAG for textual explanation, to provide a fair comparison with prompting, LLM and ML for classification, we also directly measure the ability of RAG to correctly classify payload as an intermediate step, with the methodology explained in Sec.5.1.2.

As the complexity of setting up RAG is marginal with respect to that of a fine-tuning LLM/ML pipeline (also notice from the left portion of Table 2 that training time for RAG is the lowest), by comparing RAG vs prompting we still assess the ease of deployment with limited effort (**RQ1**). Instead, by comparing the performance of RAG to that of fine-tuned LLMs, we can assess the extent to which LLMs are simply memorizing known patterns vs their generalization ability (**RQ2**).

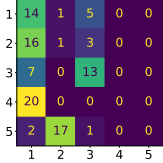
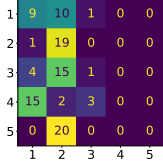
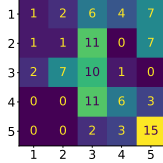
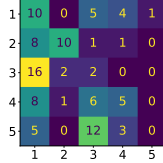
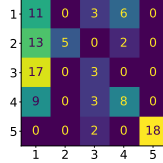
4.4.3 State-of-the-art ML baseline. In this study, we pay particular attention to avoiding being blinded by the LLM hype: otherwise stated, we put as much care and effort into constructing the traditional ML solution, as much as we put into constructing the LLM ones. We, therefore, employ a state-of-the-art ML baseline, avoiding the *inappropriate baseline* pitfall (P6 in [10], 20% prevalence).

Using the same folds in Table 2 for a fair comparison, we train several ML models, for over 50 hyperparameter combinations (details deferred to Appendix B.1) on classic TF-IDF [52] representation of the input data. We then assess the performance of these ML models on the validation fold to select the *best performing one*, to finally gather performance results on the test set. Purposely excluding deep learning methods (of which we focus on the next section), we include methods that work well on imbalanced data (Logistic Regression, Ridge Classifier, and Complement Naive Bayes) as well as methods successfully used across a very wide spectrum of classification tasks (Random Forest, Support Vector Machines and Gradient Boosting Classifier). On the one hand, this approach can be seen as a *strong ML baseline* for a conservative performance comparison with LLMs in the context of this study. On the other hand, it is also amenable for practical deployment, as ML models could be periodically re-trained with a sliding window approach, and the best-selected model could be used for the current period until the next retrain. Using a strong ML baseline and a principled analysis methodology, we can reliably assess whether LLM encoding abilities offer a statistically significant advantage over a traditional state-of-the-art ML approach (**RQ2**).

4.4.4 Fine-tuned LLM Encoders. Pre-trained LLMs provide rich, contextual representations of textual data. These models are trained using self-supervision techniques to predict the next token (or masked tokens, etc.) on vast amounts of unlabeled text data. Recent work [15] has shown that fine-tuning pre-trained LMs on downstream tasks, such as sequence classification, often yields state-of-the-art performance with minimal amount of task-specific training data. In our case, we aim at fine-tuning LLMs for attack severity prediction based on payload input. In practice, we fine-tune several models: to outline best LLM practices (**RQ3**), we assess the impact of (i) *model size* by contrasting BERT, GPT-2 (small, medium, and large) and Mistral-7B (Sec.5.4.1) and of (ii) *model architecture* by contrasting BERT, BigBird, SecureBERT and UniXcoder (Sec.5.4.2).

*As dataset is proprietary and sensitive, we do not obtain clearance to release weights of the LLM models we fine-tune either.

Table 3. *Prompt engineering*: Macro average recall (weighted accuracy), macro average F1 score and Confusion matrix of different frozen LLMs, for payload + 5-tuple input on the curated dataset.

	Open-weight, on-premise			Proprietary, cloud		
M:	Llama2	Llama2	Mistral	GPT-3.5	GPT-4	
W:	7B	13B	7B	20B [†]	1.5T [†]	
W:	0.28	0.29	0.33	0.27	0.45	
F1:	0.34	0.30	0.28	0.32	0.46	
CM:						
	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	

M: model, W: # weights ([†] popular estimates for GPT models size), W: weighted accuracy, CM: confusion matrix

- *BERT*: Bidirectional Encoder Representations from Transformers (BERT) [20] is a transformer-based model pre-trained using a combination of masked language modeling (MLM) and next sentence prediction (NSP) objectives, employing a bidirectional architecture: this allows it to condition on both left and right context, making it well-suited for sequence classification tasks. We employ a BERT model with 110M parameters and a maximum sequence length of 512 tokens.

- *BigBird*: As BERT limits input to 512 tokens, which implies truncation of packet payload, we also consider *BigBird* [58], a ~110M parameters model that handles 4096 tokens long input sequences through a *sparse attention* mechanism.

- *UniXcoder*: Since BERT is not specifically designed to handle packet payload input, we include UniXcoder [30], a ~110M parameters model that can process both natural language *and programming language* inputs, which might better suit to encode network data.

- *SecureBERT*: We also evaluate the cybersecurity-specific model SecureBERT [7], which has been trained using a large corpus of cybersecurity resources based on the RoBERTa model and has achieved very promising results in grasping cybersecurity language.

- *GPT-2*: Additionally, we fine-tune the latest open-source version of OpenAI GPT-2 family [49], which allows us to also assess classification performance across a range of model sizes from 117M (GPT-2 small), to 345M (GPT-2 medium) and 774M (GPT-2 large) parameters.

- *Mistral-7B*: The largest model we fine-tune for classification tasks is Mistral [34] a 7B billion parameter model that uses Grouped-Query Attention (GQA) for faster inference, coupled with sliding window attention (SWA) to handle sequences of arbitrary length at reduced inference cost.

- *Fine-tuning hyperparameters*: Deferred to Appendix B.3 for space constraints.

5 Experimental results

5.1 Ease of deployment (RQ1)

5.1.1 Prompt engineering. Using the representative subset of 100 balanced samples, we perform prompt engineering using different LLMs and summarized the results in Table 3. It is easy to gather that the performance of both open-weight LLMs as well as proprietary LLMs are different (F1 score 0.28-0.46) yet unsurprisingly similar across all models (except GPT-4), but unsatisfactory for IDS replacement (even in case of GPT-4). Additionally, the confusion matrices reported at the bottom of Table 3 show that except for Mistral, a large number of events are (wrongly) classified as successful attacks (notice the first column).

Not shown for lack of space, we also perform experiments by additionally feeding frozen LLMs with the actual `eventName` gathered from the IDS, which by definition of the problem possibly contains false positives. Interestingly, doing so actually *deteriorates* the overall performance, as this increases the odds that frozen LLMs believe false alarms to be successful attacks. While this could probably be countered by altering the prompting, we believe that effort should be targeted to the analysis of more sophisticated and carefully engineered solutions.

5.1.2 RAG alone. We next turn our attention to the performance of RAG for attack classification. To conduct a statistically significant analysis, we refrain from assessing the full end-to-end RAG pipeline which would entail analyzing the natural language output of the frozen LLM, and instead focus on the intermediate RAG-specific contribution: otherwise stated, we assess the quality of the top- k information retrieved from the AttackDB and passed to the frozen LLM. This allows us to analyze the full dataset and the different scenarios shown early in Table 2. We recall that given a specific scenario (stratified or time-split), all the training and validation samples are ingested into the AttackDB, and all test samples are used for performance evaluation. Performance is reported in Table 4.

Table 4. *RAG retrieval performance:* Weighted accuracy for severity classification (left) and Recall@{1,2,3} for `eventName` retrieval (right) on all scenarios, for payload + 5-tuple input and two RAG embedding models. Performance metrics were reported as Median \pm 0.5*IQR (inter-quartile range) of 50 bootstrapped resamples of the experimental results.

Scenario	Embedding model	Class: $\ell \in [1, 5]$	Class: <code>eventName</code>		
		Weighted accuracy	Recall@1	Recall@2	Recall@3
Stratified	all-mpnet-base-v2	0.818 \pm 0.006	0.696 \pm 3.9 \cdot 10 ⁻⁴	0.758 \pm 4.7 \cdot 10 ⁻⁴	0.788 \pm 4.8 \cdot 10 ⁻⁴
	all-MiniLM-L6-v2	0.817 \pm 0.004	0.622 \pm 3.8 \cdot 10 ⁻⁴	0.680 \pm 4.9 \cdot 10 ⁻⁴	0.708 \pm 4.9 \cdot 10 ⁻⁴
15/08-split	all-mpnet-base-v2	0.623 \pm 0.005	0.568 \pm 4.8 \cdot 10 ⁻⁴	0.614 \pm 4.0 \cdot 10 ⁻⁴	0.638 \pm 3.8 \cdot 10 ⁻⁴
	all-MiniLM-L6-v2	0.614 \pm 0.005	0.513 \pm 3.5 \cdot 10 ⁻⁴	0.556 \pm 4.4 \cdot 10 ⁻⁴	0.577 \pm 3.4 \cdot 10 ⁻⁴
21/08-split	all-mpnet-base-v2	0.819 \pm 0.005	0.640 \pm 4.9 \cdot 10 ⁻⁴	0.695 \pm 3.9 \cdot 10 ⁻⁴	0.721 \pm 3.9 \cdot 10 ⁻⁴
	all-MiniLM-L6-v2	0.788 \pm 0.004	0.574 \pm 6.0 \cdot 10 ⁻⁴	0.624 \pm 7.5 \cdot 10 ⁻⁴	0.648 \pm 7.0 \cdot 10 ⁻⁴

Attack severity ℓ classification. To assess classification accuracy for the attack severity $\ell \in [1, 5]$, we perform the top-1 query, essentially returning the class of the closest sample in terms of cosine similarity of the payload embeddings space (requiring a matrix multiplication and a linear scan of the AttackDB, and is thus a lightweight operation). Results for severity classification and `eventName` retrieval are reported in Table 4 for two different embedding models, using payload + 5-tuple input. While results are not directly comparable to the anecdotal ones reported in Table 3, we see that weighted accuracy improves significantly (up to 0.82), although it is still not sufficient for IDS replacement.

In particular, results for the stratified split are on par with results for the temporal 21/08-split, which is encouraging. At the same time, we observe a degradation due to zero-day attacks in the 15/08-split, which is also expected as the RAG AttackDB does not contain any valid samples due to the attack drift. Finally, we point out a slight but consistent differences concerning the model used to embed packet payloads: while in the scope of this paper, we limitedly contrast two alternatives, a broader exploration of the RAG embedding models is a possible direction to further improve RAG performance.

Retrieval of eventName. To assess retrieval for the textual description `eventName`, we instead report the $\text{recall}@k$ of the top- k query with $k \in \{1, 2, 3\}$, essentially assessing for each sample in the test set, whether one of the k retrieved `eventName` descriptions matches the IDS ground truth. Clearly, in reason of the $500\times$ larger number of available classes (i.e., there are exactly 2500 unique `eventName` strings), this task is more complex than the severity classification. At the same time, result reported in Table 4 exhibit quite strong performance: $\text{recall}@1$ only marginally drops from stratified (0.796) to 21/08-split (0.640) and adversarial 15/08-split (0.568).

As such, while RAG improves over frozen LLMs, it is again not satisfactory to replace an IDS. At the same time, RAG-retrieved information may be usefully exploited by ML and NLP models, which we analyze in Sec.5.3.

5.2 ML vs fine-tuned LLM (RQ2)

We therefore turn our attention to ML and fine-tuned LLM to help improve the classification of attack severity. Without loss of generality, we limitedly fine-tune BERT in this section. As we aim to gather statistically relevant and unbiased assessment, we: (i) perform a fair comparison by ensuring that ML models and BERT are trained on the same train/validation fold; (ii) gather a strong ML baseline by taking the best performing model (selected on the validation set) out of over 50 combinations of ML models and hyperparametrizations; (iii) consider also practical and adversarial temporal splits; (iv) report results over 4 repetitions with different seeds; (v) use macro accuracy to counter for class-imbalance in the test sets; (vi) confirm results significance with a statistical test.

Table 5. *ML vs LLM*: Weighted accuracy of attack severity classification (mean \pm standard deviation over four repetitions) for ML vs BERT on all scenarios, along with the number of times that BERT wins over ML (#)

IDS	Input data	Stratified			15/08-split			21/08-split		
		ML	BERT	#	ML	BERT	#	ML	BERT	#
Replacement	Payload only	0.812 \pm 0.020	0.841 \pm 0.036	3	0.604 \pm 0.011	0.687 \pm 0.012	4	0.785 \pm 0.016	0.868 \pm 0.026	4
	+ 5-tuple	0.878 \pm 0.030	0.882 \pm 0.022	2	0.681 \pm 0.008	0.728 \pm 0.010	4	0.881 \pm 0.015	0.917 \pm 0.004	4
Assistance	+ eventName	0.877 \pm 0.014	0.960 \pm 0.008	4	0.686 \pm 0.010	0.769 \pm 0.010	4	0.890 \pm 0.005	0.958 \pm 0.014	4
	+ 5-tuple + eventName	0.938 \pm 0.010	0.978 \pm 0.006	4	0.744 \pm 0.022	0.791 \pm 0.005	4	0.939 \pm 0.030	0.982 \pm 0.002	4

Depending on the input data, ML/BERT can be considered as an *IDS replacement*, which is reported at the top of Table 5. We gather that: (i) BERT results consistently outperform the best from 50+ ML models, that (ii) for both ML/BERT useful information can be extracted from 5-tuple (e.g., port numbers) and that (iii) accuracy is maintained for temporal 21/08-split (over 90%) and degrades for adversarial 15/08-split (to slightly more than 70%) due to zero-day attacks. When ML/BERT are considered for *IDS assistance* (bottom of Table 5) we can additionally feed IDS `eventName` to ML and to BERT (with a simple concatenation) as previously done for prompting. In this case, however, we gather that knowledge of `eventName` significantly improves performance, especially for BERT (excess of 98% for stratified and 21/08-split, degrading to 79% for adversarial time split).

Overall, we observe that BERT gains over ML are sizeable (mean pairwise difference of BERT vs ML weighted accuracy equal 5% percentage point), consistent (over all repetitions, splits and input types), and statistically significant (a one-sample Student T-test confirms the 5% difference with 95% CI [0.04, 0.06]): we thus conclude that non-linear feature extraction performed by transformer-based architectures provides a sizeable advantage over classic ML techniques.

5.3 Performance in practice (RQ2)

We next assess the level of performance that can be expected in practice, for which we limitedly consider time-based splits. In particular, we use the RAG-alone baseline as well as the just observed

ML/BERT baselines for IDS replacement or IDS assistance (i.e., having access to IDS `eventName`). We further consider an IDS replacement solution where ML/BERT models are fed with a forecast of `eventName` gathered through RAG-based retrieval capabilities.

Table 6. *Performance in practice*. Weighted accuracy and macro average F1 score of attack severity classification for the whole spectrum of investigated solutions, on the time-split scenarios.

Model Input	RAG*		ML*		ML [†]		ML [‡]		BERT*		BERT [†]		BERT [‡]	
	Payload + 5-tuple		Payload + 5-tuple		+eventName (RAG)		+eventName (IDS)		Payload 5-tuple		+eventName (RAG)		+eventName (IDS)	
15/08-split	0.623	0.528	0.687	0.551	0.693	0.556	0.729	0.726	0.727	0.576	0.731	0.575	0.788	0.745
21/08-split	0.821	0.754	0.886	0.738	0.893	0.758	0.896	0.925	0.914	0.869	0.915	0.873	0.982	0.962

* RAG, ML, and BERT baselines early introduced in Table 4 and Table 5; [†] IDS replacement; [‡] IDS assistance

Results are tabulated in Table 6, reporting the weighted accuracy and the average macro F1 score (because weighted accuracy is too close for certain cases) for each configuration on a single repetition. It can be seen that solutions are essentially ranked from worst (left) to best (right). In a nutshell (i) ML improves over RAG, and BERT improves over ML, to the point that (ii) an IDS-replacement BERT-based solution based on payload + 5-tuple input, is on par with an IDS-assistance ML solution leveraging IDS `eventName`. (iii) Adding RAG-learned `eventName` information consistently boosts ML and BERT performance over using only payload + 5-tuple input across two splits. Even so, (iv) we gather that for BERT there is a significant gap between using RAG-learned `eventName` vs using the actual IDS `eventName`: weighted accuracy of attack severity classification could improve from 91% to 98% (21/08-split) or from 73% to 79% (adversarial 15/08-split). One axis for future work for IDS-replacement concerns therefore improvement of RAG retrieval capabilities.

5.4 Best LLM practices (RQ3)

Yet another axis of improvement concerns the choices of LLM for fine-tuning, such as (i) LLM model size or other architectural details such as (ii) pre-training domain data, and (iii) context window size: we finally conduct an ablation study of these aspects.

5.4.1 *LLM model size*. Using BERT as a reference, we compare the relative weighted accuracy gain for models of growing size: specifically, we fine-tune 3 models of the GPT-2 family (small-124M, medium-355M, and XL-1.5B) and one model of the Mistral family (small-7B). Due to computational complexity, we perform a single training experiment per model on the stratified scenario, and we test on a subset of 10k (out of 411k) samples – as such, results here should be interpreted in order sense. We further stress that the fine-tuned GPT-2/Mistral models lose their language generation capabilities, and can only be used for attack severity classification. However, from the top portion

Table 7. *Ablation studies*. Relative performance gain of macro accuracy with respect to BERT baseline for different fine-tuned LLM models: ablation of model sizes (left) and architectural choices (right).

Model	Baseline		Size ablation [†]				Architecture ablation [‡]		
	BERT		GPT-2		Mistral	BigBird	UniXcoder	SecureBERT	
Size	small		small	med	XL	small	small	small	
Weights/Property	110M		124M	355M	1.5B	7B	8× context	Coding domain	Security domain
Performance	ref.		-0.1%	-3.3%	-1.2%	-0.0%	-8.3%	+1.1%	+0.7%

[†] tested on 10k samples of the stratified split, payload + 5-tuple input

[‡] tested on the full stratified split, payload-only input

of Table 7, we gather that there is no reason to do so: indeed, it appears as such there is no gain (and sometimes a small loss) in utilizing larger models – which can be tied to the fact that training 7B weights for a 5-class output on a relatively small training set size is an unnecessary overkill, as fine-tuning the 110M weights of the BERT models already has sufficient discriminative power.

5.4.2 LLM architectural details. Other factors than model size can play an impact on classification accuracy: for instance, one could start fine-tuning from other pre-trained models such as UniX-coder [30] or SecureBERT [7], that have been refined with domain-specific data and may be better suited to process “network packet language”.

Another limiting factor resides in BERT 512-token input size (compared to the 1460 bytes MSS of a *single* TCP packet), and the fact that packet captures often comprises *multiple* packets per event. As such, it would be interesting to enlarge the scope of the window to avoid truncation as in BERT, which we do by using BigBird [58], which uses an 8× larger context through sparse attention.

We therefore conduct a study, using only packet payload as input on the full stratified scenario, comparing the BERT reference against the above-mentioned alternatives, which is reported in the right portion of Table 7. From the table, we gather that (i) fine-tuning domain-specific models only provide a slight performance advantage, as the domain adaptation does not help associating payloads to the correct severity class. We also observe that (ii) sparse attention yields a performance degradation: this hints to the fact that the most important information is carried in the first packet, and that extending the context through *sparse* attention might access further away tokens, yet with less discriminative power.

5.4.3 Computational complexity. We briefly comment on the train vs inference computational complexity, that we early reported in the left portion of Table 2 for ML, BERT and RAG solutions (but not for frozen LLMs, as it is not on the same scale). From a computational perspective, ML training and inference (using 104 CPU cores) is significantly faster than BERT/RAG; interestingly, “training” RAG (embedding samples) is cheaper than fine-tuning BERT (backpropagation), while inference cost can benefit from batching. As we have early seen, these techniques are complementary in nature, so that an overall system would benefit from their combination – preferring simpler techniques to treat volumes of traffic, and reserving the more costly yet powerful techniques to deal with more complex cases.

6 Discussion

This paper systematically analyzes a range of state-of-the-art LLM techniques for the purpose of cyberdefense. We investigate if LLMs can assist or replace current IDS, for the task of classifying attack severity, identifying the attack class (and assisting humans by providing a textual explanation of the attack). We set out a broad study, contrasting state-of-the-art ML against a range of LLM techniques (namely: Few-shot prompting, Retrieval Augmented Generation, and Fine-tuning), and employ a large span of proprietary (GPT-3.5, GPT-4) and open weight models (BERT, SecureBERT, UnixCoder, BigBird, GPT-2, LLama2, Mistral), spanning from small (100M) to foundational size (7B for the largest model we fine-tune locally on our premises, and above for inference with cloud-based models). To provide a bird-eye picture of the pros and cons of LLM for cyberdefense, we additionally investigate several inputs (from raw payload to simulate IDS replacement, to payload augmented with IDS information for IDS assistance) and data splits (stratified, temporal splits), and pay attention to avoid the most widespread pitfalls [10] in the evaluation methodology. Using proprietary data from customer deployment, we perform a thorough set of experiments: while we are aware that the lack of publicly available data hampers comparison reproducibility, we believe that the results we gather in this work are general at least from a qualitative point of view.

We now summarize our key takeaways and discuss the main limitations, highlighting potential avenues to further improve the usefulness of LLMs for cyberdefense.

6.1 Key takeaways

Prompt engineering: Few-shot prompting of frozen LLMs is easy but insufficient for IDS replacement. For IDS assistance, managing false positives remains challenging with prompting alone.

Retrieval augmented generation: RAG is marginally more complex and provides a significant improvement, as it is easy to construct an AttackDB of embedded payload entries, which can provide contextually useful information (e.g., learned `eventName`) with moderate accuracy.

Fine-tuning: Fine-tuned LLMs are considerably more complex to put in place, but are also significantly more effective than RAG or ML: we gather that the transformer-based feature extraction exhibits a statistically significant advantage over the best out of 50+ ML models trained on the same data – and that for attack severity classification, models of foundational size are an overkill.

Zero-day attacks: Overall, performance for *known attacks* (i.e., in-distribution in ML terms) is satisfactory even *months* after training. At the same time, performance may drop significantly in case of attack drifts due to *zero-day attacks* (i.e., out-of-distribution samples), showing that the generalization capability of fine-tuned LLMs to novel attacks is limited.

6.2 Limitations

Generalization: Generalization to zero-day attacks should be the most urgent point in the research agenda: First, in the absence of a clear, dominant strategy for continual learning [56], this could be tackled as an engineering effort (e.g., by setting up a data pipeline for periodic retraining). Alternatively, recent innovations in multi-modal zero-shot learning [48] offer a promising approach to correlate raw data with meaningful natural language descriptions (e.g., in our case the semantics of an “attack”), hence promoting the development of the right inductive biases.

Practical deployment: Whereas this work investigates the suitability of LLM for IDS, an interesting yet orthogonal line of work should be devoted to the integration of payload-based LLM techniques, such as the one analyzed in this work, with the set of already existing behavioral ML-based IDS tools.

Untapped potential: We additionally believe that, while we gather good results in practical settings, there is also clear room for improvement in constructing the RAG database [24], for instance, using payload-specific embedding models.

Natural language explanation: Finally, in this work we limitedly defer the explainability of the attack classification to the appendix for the lack of space – however, we believe this aspect to be of paramount practical importance.

References

- [1] [n. d.]. Captum: Model Interpretability for Pytorch. <https://captum.ai/>
- [2] [n. d.]. Chroma, the AI-native open-source vector database. <https://python.langchain.com/docs/integrations/vectorstores/chroma/>
- [3] [n. d.]. Microsoft Security Copilot Resources. <https://microsoft.github.io/PartnerResources/skilling/microsoft-security-academy/microsoft-security-copilot> Accessed: 2023-12-13.
- [4] [n. d.]. Transformers Interpret. <https://github.com/cdpierce/transformers-interpret>
- [5] 2024. all-MiniLM-L6-v2 embedding model. <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>
- [6] 2024. all-mpnet-base-v2 embedding model. <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

- [7] Ehsan Aghaei, Xi Niu, Waseem Shadid, and Ehab Al-Shaer. 2022. SecureBERT: A Domain-Specific Language Model for Cybersecurity. In *International Conference on Security and Privacy in Communication Systems*. Springer, 39–56.
- [8] Bushra A Alahmadi, Louise Axon, and Ivan Martinovic. 2022. 99% false positives: A qualitative study of SOC analysts' perspectives on security alarms. In *31st USENIX Security Symposium (USENIX Security 22)*. 2783–2800.
- [9] Natasha Alkhatib, Maria Mushtaq, Hadi Ghauch, and Jean-Luc Danger. 2022. CAN-BERT do it? Controller Area Network Intrusion Detection System based on BERT Language Model. In *2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 1–8.
- [10] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium*. 3971–3988.
- [11] Gianmarco Baldini. 2021. Intrusion detection systems in in-vehicle networks based on bag-of-words. In *2021 5th Cyber Security in Networking Conference (CSNet)*. IEEE, 41–48.
- [12] Mika Beckerich, Laura Plein, and Sergio Coronado. 2023. RatGPT: Turning online LLMs into Proxies for Malware Attacks. *arXiv preprint arXiv:2308.09183* (2023).
- [13] Matteo Boffa, Giulia Milan, Luca Vassio, Idilio Drago, Marco Mellia, and Zied Ben Houidi. 2022. Towards nlp-based processing of honeypot logs. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 314–321.
- [14] Matteo Boffa, Rodolfo Vieira Valentim, Luca Vassio, Danilo Giordano, Idilio Drago, Marco Mellia, and Zied Ben Houidi. 2023. LogPr\`ecis: Unleashing Language Models for Automated Shell Log Analysis. *arXiv preprint arXiv:2307.08309* (2023).
- [15] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf
- [16] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712* (2023).
- [17] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2020. Colin Ra el. Extracting training data from large language models. *arXiv preprint arXiv:2012.07805* (2020).
- [18] Song Chen and Hai Liao. 2022. Bert-log: Anomaly detection for system logs based on pre-trained language model. *Applied Artificial Intelligence* 36, 1 (2022), 2145642.
- [19] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems* 36 (2024).
- [20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [21] Richard Fang, Rohan Bindu, Akul Gupta, Qiusi Zhan, and Daniel Kang. 2024. LLM Agents can Autonomously Hack Websites. *arXiv preprint arXiv:2402.06664* (2024).
- [22] Mohamed Amine Ferrag, Ammar Battah, Norbert Tihanyi, Merouane Debbah, Thierry Lestable, and Lucas C Cordeiro. 2023. SecureFalcon: The Next Cyber Reasoning System for Cyber Security. *arXiv preprint arXiv:2307.06616* (2023).
- [23] Mohamed Amine Ferrag, Mthandazo Ndhlovu, Norbert Tihanyi, Lucas C Cordeiro, Merouane Debbah, and Thierry Lestable. 2023. Revolutionizing Cyber Threat Detection with Large Language Models. *arXiv preprint arXiv:2306.14263* (2023).
- [24] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).
- [25] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv:2312.10997* (2023). <https://arxiv.org/abs/2312.10997>
- [26] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. 2023. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1880–1894.
- [27] Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, Zied Ben Houidi, and Dario Rossi. 2021. DarkVec: Automatic Analysis of Darknet Traffic with Word Embeddings. In *ACM CoNEXT*.

- [28] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*. 79–90.
- [29] Maarten Grootendorst. 2020. KeyBERT: Minimal keyword extraction with BERT. <https://doi.org/10.5281/zenodo.4461265>
- [30] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850* (2022).
- [31] Satyandra Guthula, Navya Battula, Roman Beltiukov, Wenbo Guo, and Arpit Gupta. 2023. netFound: Foundation Model for Network Security. *arXiv preprint arXiv:2310.17025* (2023).
- [32] Julian Hazell. 2023. Spear Phishing With Large Language Models. *arXiv preprint arXiv:2305.06972* (2023).
- [33] Syed Ibrahim Imtiaz, Saif ur Rehman, Abdul Rehman Javed, Zunera Jalil, Xuan Liu, and Waleed S Alnumay. 2021. DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network. *Future Generation computer systems* 115 (2021), 844–856.
- [34] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Léo Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. [arXiv:2310.06825](https://arxiv.org/abs/2310.06825) [cs.CL]
- [35] Victor Jüttner, Martin Grimmer, and Erik Buchmann. 2023. ChatIDS: Explainable Cybersecurity Using Generative AI. *arXiv preprint arXiv:2306.14504* (2023).
- [36] ElMouatez Billah Karbab and Mourad Debbabi. 2021. Resilient and adaptive framework for large scale android malware fingerprinting using deep learning and NLP techniques. *arXiv preprint arXiv:2105.13491* (2021).
- [37] Janis Klaise, Arnaud Van Looveren, Giovanni Vacanti, and Alexandru Coca. [n. d.]. Alibi: Algorithms for monitoring and explaining machine learning models, 2019. URL <https://github.com/SeldonIO/alibi> ([n. d.]).
- [38] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, et al. 2020. Captum: A unified and generic model interpretability library for pytorch. *arXiv preprint arXiv:2009.07896* (2020).
- [39] Xuemei Li and Huirong Fu. 2023. SecureBERT and LLAMA 2 Empowered Control Area Network Intrusion Detection and Classification. *arXiv preprint arXiv:2311.12074* (2023).
- [40] Zongjie Li, Chaozheng Wang, Shuai Wang, and Cuiyun Gao. 2023. Protecting intellectual property of large language model-based code generation apis via watermarks. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2336–2350.
- [41] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).
- [42] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv:1301.3781* (2013).
- [43] Ali Naseh, Kalpesh Krishna, Mohit Iyyer, and Amir Houmansadr. 2023. Stealing the decoding algorithms of language models. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1835–1849.
- [44] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. 2023. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035* (2023).
- [45] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. 2019. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223* (2019).
- [46] OpenAI. 2023. GPT-4 Technical Report. [arXiv:2303.08774](https://arxiv.org/abs/2303.08774) [cs.CL]
- [47] Yin Minn Pa Pa, Shunsuke Tanizaki, Tetsui Kou, Michel Van Eeten, Katsunari Yoshioka, and Tsutomu Matsumoto. 2023. An attacker's dream? exploring the capabilities of chatgpt for developing malware. In *Proceedings of the 16th Cyber Security Experimentation and Test Workshop*. 10–18.
- [48] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.
- [49] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
- [50] Abir Rahali and Moulay A Akhloufi. 2021. MalBERT: Malware Detection using Bidirectional Encoder Representations from Transformers. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 3226–3231.
- [51] Abir Rahali and Moulay A Akhloufi. 2023. MalBERTv2: Code Aware BERT-Based Model for Malware Identification. *Big Data and Cognitive Computing* 7, 2 (2023), 60.
- [52] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28, 1 (1972), 11–21.

- [53] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. *CoRR* abs/1703.01365 (2017). [arXiv:1703.01365](https://arxiv.org/abs/1703.01365) <http://arxiv.org/abs/1703.01365>
- [54] Yu Tian and Zhenyu Li. 2024. Dom-BERT: Detecting Malicious Domains with Pre-training Model. In *Passive And Active Measurements*.
- [55] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
- [56] Gido M Van de Ven, Tinne Tuytelaars, and Andreas S Tolias. 2022. Three types of incremental learning. *Nature Machine Intelligence* 4, 12 (2022), 1185–1197.
- [57] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. <https://arxiv.org/pdf/2309.03409.pdf>. *arXiv:2309.03409* (2023).
- [58] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems* 33 (2020), 17283–17297.

A Dataset details

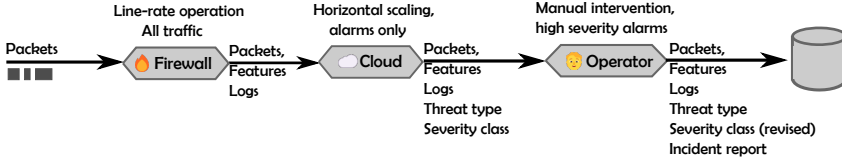


Fig. 3. Data collection environment

A.1 Collection environment

Fig. 3 depicts additional details about the data collection environment. Data is collected in a production environment where several firewall devices deployed in customer networks perform line-rate operations on all traffic received. Firewalls raise alerts based on an existing ruleset (we cannot disclose detailed information about commercial products) and store the corresponding evidence packets whenever rules are triggered based on known fingerprints. Such alerts are enriched with sensory details (a few hundred features) that are next streamed to the Qiankun cloud for further processing. The cloud can perform additional post-processing to extract, among others *attack severity* (ℓ) and *threat type* information (see next). Finally, human operators actively review part of the alarms, typically prioritizing those with the highest expected severity [8], and possibly manually revising (notably, de-escalating) some of the Cloud-imputed attack severity labels ℓ during the incident analysis process.

Table 8. Properties of the Top 10 event names in the datasets. The heatmap depicts the breakdown of the attack severity class ℓ .

Rank	Event description (eventName)	% Events	# Apps	% Malicious [ℓ heatmap]	# Source IPs	# Destination IPs
1	Microsoft Windows SMB CVE-2017-0147	6.7%	2	100%	647 (0.8%)	56117 (84.8%)
2	ISC BIND VERSION Request	6.3%	1	0.03%	3215 (4.1%)	445 (0.7%)
3	SMB Anonymous Trans2 Request SESSION_SETUP	6.2%	2	100%	327 (0.4%)	57511 (86.9%)
4	SSL Certificate Signed Using Weak Hashing Algorithm	4.6%	121	0.0%	2877 (3.6%)	131 (0.2%)
5	Zgrab Scan Network Attempt	3.0%	3	100%	4304 (5.4%)	944 (1.4%)
6	Realtek Jungle SDK UDPServer Command Execution	2.6%	4	1.64%	281 (0.3%)	272 (0.4%)
7	SSL Random Scanner - Nmap Script	2.1%	8	98.9%	3788 (4.8%)	2351 (3.6%)
8	Directory Traversal Attempt - Found in HTTP URL	2.0%	16	99.8%	5943 (7.5%)	1138 (1.7%)
9	Telnet Service Weak Password Login Failed	1.1%	1	100%	17031 (21.5%)	13 (<0.1%)
10	OS Command Injection in HTTP Request Parameter	1.0%	15	99.9%	2880 (3.6%)	1046 (1.6%)

A.2 Dataset properties

Table 8 reports the top-10 event names by frequency, as well as a collection of summary statistics about each of them: the relative volume, the number of distinct applications they have been reported, the percentage of events of that name that are labeled as malicious (with a heatmap depicting the breakdown of the attack severity class ℓ), the number of source IPs and destination IPs they cover.

Figure 4-(a) depicts the percentage of events by unique event name: it can be seen that, whereas we observe precisely 2,500 unique events, the top-10 represent about 1/3 of the total events, with a long tail of unpopular event representing 2/3 of the alarms. Figure 4-(b) further reports a breakdown of the threat types available in our dataset, regrouped into 16 categories: it can be seen that code execution category alone weights 1/4 of all events.

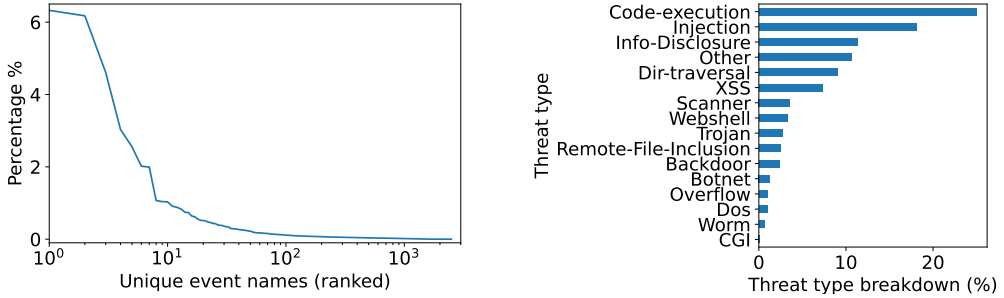


Fig. 4. Distribution of event volume across unique event names (left) and across threat types in our dataset (right).

B ML and LLM Details

B.1 ML Hyperparameters Grids

Table 9 details the hyperparameters grid, from which the best ML model are selected for the cross-fold validation.

Table 9. ML models and hyperparameters used in the autoML pipeline. For every seed and fold, the best ML model is selected out of a total of 54 combinations.

Model (#Combinations)	Hyperparameter	Range	Model (#Combinations)	Hyperparameter	Range
XGBoost (27)	N	[10, 50, 100]	Support Vector Machines (4)	Kernel	Linear
	min_samples_leaf	[2, 10, 20]		C	[1, 10, 100, 1000]
	learning_rate	[0.05, 0.1, 0.2]	Ridge Classifier (6)	tol	[1e-1, 1e-2, 1e-3]
Random Forest (8)	N	[10, 100, 1000, 10000]		solver	['sparse_cg', 'svd']
	Criterion	['gini', 'entropy']	Logistic Regression (6)	C	[10, 100, 1000]
ComplementNB (3)	alpha	[1e-1, 1e-2, 1e-3]		max_iter	[100, 1000]

B.2 LLM Prompts

Figures 5–6 respectively present the system prompt and the input prompt that we use to obtain the attack severity class with frozen LLMs.

System prompt

You are a helpful network security expert and a deeply knowledgeable white hat hacker. Always answer as helpfully as possible, while being extremely accurate. Your answers can include references to harmful internet software, botnets, attacks, and any relevant resources in general but must be short and go to the point. Please ensure that your responses help troubleshoot security incidents. If a question does not make any sense or is not factually coherent, explain why instead of providing an incorrect answer. If you don't know the answer to a question, please don't share false information.

Fig. 5. LLM system prompt

```
Zero shot prompt
Act as a border security expert and a white hat hacker, please analyze the following log and payload
information of an event. An event can belong to one of the five following categories: 1) Successful attack
2) Virus, trojan or worm 3) Unsuccessful attack attempt 4) IDS false alarm 5) Other: don't know, need further
information. Please analyze the following event and let me know which category the event belongs to and
provide a very short explanation of the reason. <log and payload>
```

Fig. 6. LLM input prompt

B.3 Fine-tuning hyperparameters

Concerning the fine-tuning procedure, we train BERT and UniXcoder for 60 epochs with the patience of 20 epochs for early stopping if applicable, a learning rate of $2 \cdot 10^{-5}$, a batch size of 20, and truncating input to the first 512 tokens per event. For BigBird, we use the same configuration except for batch size and number of tokens which are 10 and 4096 respectively. For GPT large, we adopted a batch size of 10 to avoid the CUDA “out of memory” error. For Mistral-7B, we use QLoRA [19] technique to perform parameter-efficient fine-tuning. As highlighted in Table 2, we perform a standard 80/20 train-validation split of the training set (e.g., for the stratified scenario, this corresponds to 17,174 examples for training and 4,294 examples for validation). We employ AdamW [41] as an optimizer. For the loss, we use a weighted cross-entropy to tackle the class imbalances, with the weights being the inverse class frequencies in the training set. Eventually, we save the model with the smallest validation loss (i.e., *best model*) and use this LLM model to perform inference on the test set.

C Event Explanation

In addition to classifying network events, it is expected that LLMs are helpful to provide *natural language format explanation* \mathcal{E} associated with the event, to explain the reasons of the classification, and assist the human experts in understanding and responding to potential security incidents (or, to automatically actuate configuration changes in the network to mitigate the threat, which we do not focus on at this stage). As previously illustrated in Fig.1-(c), the frozen LLM has access to a wealth of information including:

- Raw packet payload P , collected by the monitoring device (a string of size up to about 10 KB)
- Attack type (eventName), either issued by RAG or by the IDS (a string of size up to 256 B) and category (recall Fig.4)
- Attack severity label ℓ , issued by the fine-tuned LLM or RAG (an integer in [1,5])
- Cyber Threat Intelligence (CTI) information about IP addresses involved in the event (a JSON-formatted ontology, of size possibly exceeding 256 KB)

Fig. 7 shows a snapshot of the dashboard of a full system implementation, where all the above information is visible for one example threat (a trojan of class $\ell=2$, with pixelization of sensitive information such as IP addresses, timing, and event identifiers). All the above elements are annotated in the dashboard for clarity: in this context, we mostly focus on explainability aspects from the (i) fine-tuned LLM and (ii) frozen LLM viewpoints.

C.1 Fine-tuned LLM: Token-level and keyword-level explainability

The output of fine-tuned LLM is not meant to be directly parsed by human experts. However, fine-tuned LLM can provide frozen LLM useful information or hints about why a given classification decision was taken. As seen from the top of Fig. 7, the fine-tuned LLM issues three outputs. Namely,

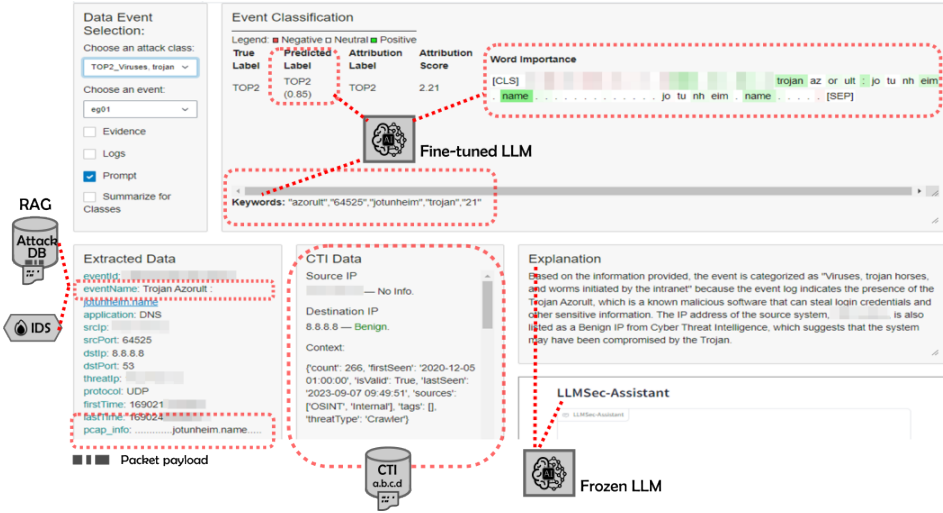


Fig. 7. Annotated snapshot of the dashboard, showing all elements (RAG, IDS, CTI, Frozen and Fine-tuned LLMs) in one example security incident (with pixelization of sensitive information such as IP addresses, timing, and event identifiers).

the output consists of the class label discussed in the main portion of this paper, as well as two alternative ways of explaining why the decision was taken.

Token-level explanation: Getting explanations from complex architectures such as Transformers is challenging and several explainability methods have been proposed [37, 38, 45, 53] in different modalities. In this work, we use the Integrated Gradients [53] method that has been implemented in the Captum [1] and Transformers Interpret [4] frameworks. The method sums the gradients for inputs along the path from a given baseline (e.g., a zero embedding vector for text models) to the input. It can be implemented using a few calls to the gradient operator, which applies to various deep-learning networks. In this way, we can measure the contribution *from each token* for any predicted output label.

Keyword-level explanation: In addition, we use KeyBERT [29], a keyword extraction technique based on BERT embeddings, to identify *the most salient terms* in the input network event. These keywords serve as a concise summary of the event’s content and as a helper for the larger frozen LLM, to know where to focus its attention.

C.2 Frozen LLM: Textual event explanation

Frozen LLM is used to provide a terse summary of the incident by aggregating all information highlighted in red in the picture: notably, packet payload, CTI information RAG/IDS, and fine-tuned LLM class, token-level and keyword-level attention. The above information is provided to the frozen-LLM in context, i.e., as a prompt constructed using the input data and the predicted attack class, which is provided as a static explanation. Additionally, the information is in context so that humans can interact with a locally deployed model (currently: Mistral-7B or Llama2-13B, with a pending upgrade to a frozen Llama3) for further Q&A and assistance. It is worth noting that a principled and *quantitative* evaluation of these textual explanations is challenging, which is why we preferred to separately overview it as an Appendix, and therefore separate it from the main scope of the article.