Explainable Stacking Models based on Complementary Traffic Embeddings

Explainable Stacking Models based on Complementary Traffic Embeddings

Luca Gioacchini*, Welton Santos†, Barbara Lopes†, Idilio Drago‡,
Marco Mellia*, Jussara M. Almeida†, Marcos André Gonçalves†
* *Politecnico di Torino, Turin, Italy,*
{luca.gioacchini, marco.mellia}@polito.it
† *Universidade Federal de Minas Gerais, Belo Horizonte, Brazil,*
{weltonarsantos, barbara.gcol}@gmail.com,
{jussara, mgoncalv}@dcc.ufmg.br
‡ *Università di Torino, Turin, Italy,*
idilio.drago@unito.it

***Abstract*—Network security relies on effective measurements and analysis for identifying malicious traffic. Recent proposals aim at automatically learning compact and informative representations (i.e. embeddings) of network traffic that capture salient features. These representations can serve multiple downstream tasks, streamlining the machine learning pipeline. Researchers have proposed techniques borrowed from Natural Language Processing (NLP) and Graph Neural Networks (GNN) to learn such embeddings, with both lines delivering promising results.**

This paper investigates the benefits of *combining* complementary sources of information represented by embeddings learnt via different techniques and from different data. We rely on classifiers based on traditional features engineering and on automatic embedding generation (borrowing from NLP and GNN) to classify hosts observed from darknets and honeypots. We then *stack* these base classifiers trained on each embedding through meta-learning to combine the complementary information sources to improve performance.

Our results show that meta-learning outperforms each single classifier. Importantly, the proposed meta-learner provides *explainability* on the importance of the embedding types and the impact of each data source on the outcome. All in all, this work is a step forward in the search for more effective, general, understandable, and practical representations that could carry multiple traffic characteristics.

***Index Terms*—Representation learning, traffic classification, meta-learning, model stacking**

1. Introduction

Artificial Intelligence (AI) increasingly supports network analysts and security experts to address growing and complex challenges in the ever-evolving landscape of network security. This surge is notably marked by the number of novel proposals targeting various applications, from traffic classification [1], [16], [27], [28], to cybersecurity problems [9], [25], [30], and unsupervised exploration of traffic [12], [19], [29], to cite a few.

A common trend in such techniques consists of learning *embeddings*, i.e. compact yet highly informative numerical representations of network entities (e.g. hosts [12], ports [4], traffic flows [15], etc.) capturing the main aspects of the traffic. These embeddings are later fed as input to specialised Deep Learning (DL) or Shallow Learning (SL) models to accomplish diverse downstream tasks [17], [22].

Yet, how to represent network data to produce robust embeddings is still an open problem. Many studies rely on classic feature engineering [21] or automatic autoencoders [18], [19] to process network traces that offer traffic-related measurements that are then fed into deep neural networks (DNN). More recently, some authors started using Natural Language Processing (NLP) techniques to build "documents" in which the temporal sequence of *entities* (e.g., host IP addresses appearing on network logs) are treated as sequences of words [12], [29]. Such texts are then processed through specific DNNs to produce context-aware embeddings [23], [24]. Another emerging yet promising approach is representing network data as a graph [11] reflecting the network topology and the relationships between entities. The graph is processed through a Graph Neural Network (GNN), a deep learning model that exploits the structural information of a graph to generate meaningful representations for its nodes and their connections.

These models prove instrumental in summarising network data and measurements, enabling the identification of patterns and anomalies that might go unnoticed through traditional methods [12], [19]. Nevertheless, despite their effectiveness in several tasks, such techniques are often tailored to a specific context, as exemplified by their applications in Wireless Networks [3], Honeypots [20], *inter alia.* In essence, the embeddings produced from these heterogeneous networks are characterised by inherent diversity, providing distinct yet complementary sources of information. *This contextual specificity calls for approaches to seamlessly integrate diverse (complementary) models, data sources, and information to build general knowledge across networks, broadening the range of applications they can serve.*

Early works toward the direction of combining different network data sources explore multi-modality. Typically, they merge the informative content of heterogeneous features that are not originally comparable (e.g. text, images, audio, etc.) [21], [31], [34]. A few works focus on combining specialised models rather than focusing on the raw data [33]. In this work, we focus on a particular type of model ensemble – stacking [14] – which consists of learning how to combine multiple models trained on different sources and fusing their predictions (e.g. through a meta-model in case of Meta-learning) to achieve a more comprehensive view [10] of a given network.

Unlike prior work focusing on multi-modal data belonging to the same networks, here we investigate the benefit of stacking models trained on different representations derived from *complementary sources of information* related to *heterogeneous network setups*.

Namely, we address a supervised host classification problem as a case study. We generate host embeddings using (i) standard features engineering of domain-specific data, (ii) NLP techniques for temporal co-occurrences of hosts [12] and (iii) GNN approaches for capturing the evolution of host interactions [11]. Departing from a hypothesis that these three sources are complementary, we examine the extent to which building classifiers by *stacking* multiple base learners built on top of these complementary embedding representations can boost performance. In particular, we consider a simple naïve majority voting and a full-fledged *Meta-Learning* (ML) stacking that *learns* how to combine the output (class probabilities) of the base learners.

Additionally, we investigate the benefit of combining embeddings learnt from the traffic observed in different sources of data. As use cases, we consider a darknet (or network telescope) and honeypots. The first is a passive monitoring system designed to sense activities on unused portions of the Internet [32]; the latter are decoy systems intentionally set up to attract and interact with cyber attackers [20].

Our experimental evaluation using data collected during 31 days shows that stacking solutions boost performance, confirming our hypothesis of complementarity of the base classifiers. Furthermore, the experiments show the benefits of combining information from different data sources, confirming another source of complementarity. The best stacking alternative –the Meta-Learner– achieves 93% of the F1-score in the host classification task.

Fundamental, the meta-learner layer provides explainability by suggesting which are the features that drive the classifier in making its decision, a key requirement in network security and monitoring where the analyst shall understand the rationale of a machine learning decision.

2. Host Classification Methods

We compare alternative methods used to *classify hosts* exchanging traffic in a network. We consider classifiers trained on host embeddings generated according to different strategies as a baseline. Next, we consider stacking approaches that combine the outputs of those baselines to produce a final classification.

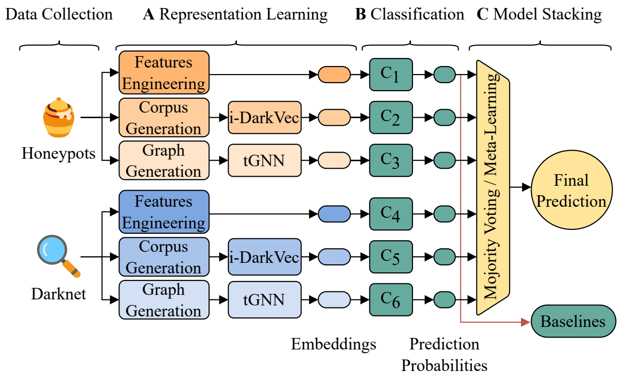


Figure 1: Overview of the full pipeline: first, we collect data from different networks; (**A**) process traffic traces and produce (i) traffic-related domain-specific host embeddings through features engineering, (ii) text host embeddings through i-DarkVec, (ii) graph host embeddings through GNNs; (**B**) train downstream classifiers specialised on each host embedding, retrieving the class prediction probabilities; (**C**) predict the final class of a host, stacking the prediction probabilities of the specialised classifiers.

Figure 1 presents an overview of the full pipeline of execution. From left to right, we first gather a sequence of temporal snapshots of network traffic collected by two network monitors[1], i.e. a darknet and honeypots (see Section 3 for more details). The approach can be easily extended to encompass other scenarios; in the representation learning stage (**A**) we represent the hosts that generated the traffic adopting a specific embedding approach to transform the raw data into features (see Section 2.1); for each feature, we solve a host classification problem (**B**) training a specialised classifier that learns class probabilities (see Section 2.2). For the stacking step, (**C**) we combine the contributions of the single classifiers, i.e., their output class probabilities, to produce a final classification (see Section 2.3).

2.1. Representation Learning: Host Embeddings

In the representation learning stage, we describe the host activities observed within a network by producing host *embeddings*. We can obtain them via classic feature engineering driven by domain knowledge, or via automated learning processes as done in a Deep Neural Network style. In a nutshell, we define an embedding as a numerical representation of a host, e.g. the features that characterise the intensity of traffic hosts generate or the temporal relationships among hosts observed in a network. Here, we describe the three techniques we adopt to produce host embeddings.

2.1.1. Domain-driven Features Engineering. Given the wide range of patterns of a host when exchanging traffic in a network, we focus on features that describe the intensity and type of traffic generated by hosts. We consider information about traffic volume and the range of services that reveal insights into the host's intentions.

We process raw traffic traces collected in different networks to extract traffic-related features. We rely on traditional features engineering techniques to associate each host to a feature vector which summarises its traffic intensity and type, e.g. sum, average, and standard deviation of the packets sent by a host (see the *Host Features* column of Table 5, Appendix A for more details).

[1] In the following we use the term "network" to state the type of data a network monitor exposes.

2.1.2. Text Embeddings. We rely on i-DarkVec [12] to generate host embeddings capturing the temporal co-occurrence of hosts when generating traffic, using the (TCP) port numbers as a proxy for coarsely grouping the traffic by service. i-DarkVec relies on a NLP algorithm, Word2Vec, and incremental training. We refer to [23] for details on Word2Vec and only provide here a brief overview of i-DarkVec.

In a nutshell, i-DarkVec represents hosts sourcing the traffic (identified by IP addresses) highlighting their common communication patterns. We build "documents" that report the sequences of source IP addresses present in traffic traces, grouping senders by the destination TCP port they contact. Analogously to NLP, IP addresses represent "words", and their sequence represents "sentences". We feed the generated documents as input to Word2Vec, which is trained in a self-supervised way [24] to produce contextual embeddings – i.e. hosts co-occurring in time when targeting similar ports belong to the same *context* and appear close to each other in the embedding space.

2.1.3. Graph Embeddings (GNN). We consider the observed sequence of traffic snapshots and represent network traffic as a bipartite graph. At each snapshot t we build a bipartite graph from the raw traffic traces. The two node layers of the bipartite graph are (i) source host nodes and (ii) destination port nodes. A link exists between a pair of nodes if the host sends packets towards the port in the considered snapshot. The link weight is the number of sent packets. Similarly to the feature-engineered embeddings, we associate each node with a traffic-related feature vector. We refer the reader to Table 5, Appendix A for more details.

We feed the graph as input to an i-GCN-GRU [11], a temporal GNN, which we train in a self-supervised way to produce host embeddings. We refer the reader to [11] for a description of i-GCN-GRU. In a nutshell, we train the GNN to predict the presence of a link between two nodes given the links of neighbouring nodes. Upon training, the GNN produces the final host embeddings.

2.2. Classification Task

For each host embedding generated according to one of the methods described above, we train a specialised classifier to accomplish *a host classification task*. Specifically, we aim to identify groups of hosts engaged in similar activities in a network. Hence, we rely on an external ground truth (described in Section 3) whose labels indicate groups of hosts whose coordination is known a priori.

Motivated by the assumption that sufficiently robust and informative embeddings can cater to various tasks independently from the employed model [8], we use a simple k-nearest-neighbours classifier (kNN) [7] that classifies data points based on the class probabilities in their neighbourhood. Specifically, kNN estimates the class probabilities of a given point based on the class frequencies among their k-nearest neighbours in the embedding space. The output is the class with the highest probability.

Besides being fast for practical purposes, kNN is also *calibrated*: it outputs reliable probabilities that are strongly correlated with the accuracy of the model [6]. This is an important requirement for the stacking methods, which combine the probabilities for a final decision.

We refer to the three classifiers built by employing kNN along with one of the host embedding methods described in the previous section as i) "DS" for the domain-specific traffic embeddings, ii) "text" for the i-DarkVec embeddings and iii) "graph" the GNN embeddings, thus highlighting that their differences lie solely on the embedding strategy employed. We take these classifiers as baselines for comparison with the proposed stacking models.

2.3. Stacking Methods

Our stacking approaches leverage the specialised kNN classifiers. We combine the host class probabilities produced by each classifier through two stacking approaches: (i) an unsupervised *naïve* stacking, based on a simple Majority Voting strategy and (ii) a machine learning model *trained to learn how to combine the outputs (probabilities) of the base methods*. We call the latter the *meta-learner (ML)* since it learns based on the output of other models.

Naïve stacking We consider the output of each base model as a "vote" for a class. Each classifier "votes" for the highest probability class, and the naïve stacking model outputs the class with the most votes. In case of a tie, we randomly choose one of the most voted classes.

ML stacking We train a meta-learning model with the class probabilities produced by each base model in a supervised way to accomplish the same host classification task as the base models, i.e., to produce a host class. Given practical applicability concerns and preliminary experiments among several alternatives, we choose a Logistic Regression (LR) model [5] as the meta-learner. LR is very effective for the task, efficient at training and testing time and, like kNN, calibrated. Additionally, unlike more complex alternatives like neural networks, LR provides *explainability* – i.e. the coefficients of the LR models provide insights into the importance of each feature.[2]

In a nutshell, for each class, the meta-learner learns a (LR) function that combines the different (base) class probabilities into a final class probability. That is, given m base models and Y classes, the meta-learner learns Y different functions, one for each class. Each function combines the $Y \times m$ class probabilities produced by all base models and outputs a probability of the data point belonging to the corresponding class. Learning the functions corresponds to learning how to weight the contributions (i.e., the class probabilities) of the different base models to produce a final class probability. By building Y different functions, the ML model allows for different weights to be used for estimating the probabilities of different classes. In the end, the class with the highest estimated probability is chosen.

3. Datasets

We focus on two network deployments commonly used in cybersecurity applications, i.e. darknets and honeypots. We collect 31 days of traffic (from 2022-10-01 to

[2] Notice that the features of the meta-learner are the base class probabilities. Nevertheless, base models are specialised on the different embeddings. Hence, we consider the LR features as a proxy for the embedding resulting from the different methods.

TABLE 1: Number of hosts seen in each network. We report values for (i) each network independently; (ii) hosts active in both the networks ($D \cap H$); (iii) total observed hosts without repetitions ($D \cup H$).

	Darknet D	Honeypots H	$D \cap H$	$D \cup H$
Mirai-like	76 176	92 077	62 955	105 298
Spammer	5 395	6 615	4 189	7 821
ShadowServer	3 074	3 074	3 074	3 074
Driftnet	2 772	3 456	2 772	3 456
InternetCensus	2 400	2 404	2 395	2 409
Bruteforcer	2 618	18 194	2 223	18 589
Censys	1 210	1 510	1 210	1 510
Rapid7	1 255	1 267	1 201	1 321
Onyphe	728	829	725	832
Shodan	295	315	292	318
SecurityTrails	162	162	162	162
Ipip	132	202	132	202
Exploiter	111	895	91	915
IntrinSec	56	149	54	151
Michigan Uni.	30	42	30	42
Unknown	47 093	62 212	34 809	74 565
Total	143 507	193 403	116 314	220 665

2022-10-31) from each network. We reserve the first 20 days for bootstrapping the embedding generation methodologies and focus our analysis and dataset characterisation on the last 11 days. Notice that both datasets cover the same period.

Darknets, D: Darknets (or network telescopes) are sets of IP addresses announced on the Internet but without hosting any services. They collect mostly large-scale Internet scans. We collect data from a /24 darknet. As in previous works [12] we focus on TCP SYN packets and remove hosts that send less than 5 packets per day. This step removes most of the backscattering traffic (i.e., traffic coming from victims of attacks with IP spoofing). The final dataset contains more than 52 thousand hosts that sent more than 15 million SYN packets over 11 days.

Honeypots, H: Honeypots are sets of IP addresses hosting (real or emulated) vulnerable services to attract attackers. They provide a controlled environment for the observation of malicious activities. We install a setup based on the honeypots distributed by the T-Pot project [13]. We deploy only low-interaction honeypots, exposing remote terminal services (e.g., SSH and telnet), remote desktop, and file-sharing services (e.g., based on SAMBA), among others. Each service logs all TCP connections and application interactions, which include not only scan traffic but also brute-force login attempts and initial exploit steps. We rely on the T-Pot 20.06 bundle and deploy services on their standard ports. Moreover, we deploy the DPIPot [32], which performs deep packet inspection to handle traffic reaching non-standard ports.

We consider here the SYN packets of successfully negotiated TCP sessions with the honeypots and apply the same filtering approach used for the darknets. The final dataset contains 73 thousand hosts sending more than 60 million TCP flows over 11 days.

Ground Truth: We perform the final host classification task relying on a ground truth that contains groups of hosts engaged in similar activities. We rely on four data sources: (i) ground truth available from [12] (ii) the presence of fingerprints of Mirai-like malware observed in received packets [2]; (iii) information from a public repository of acknowledged scanners,[3] i.e. non-hostile hosts performing scanning activities; (iv) labels provided by domain experts (*Bruteforcer*, *Spammer* and *Exploiter*) based on the activity observed on the honeypots. The resulting ground truth labels 54% of the hosts observed within each network into 15 classes. Such labelled hosts are responsible for 34% of the observed darknet flows and 43% of the honeypot flows over 11 days. We mark all the remaining hosts as the additional class, *Unknown*, having, in total $Y = 16$ classes. Notice that, as the *Unknown* class includes hosts whose characteristics we cannot verify, we consider samples belonging to this class when training the models, but do not report classification metrics for them.

In Table 1 we report the number of hosts, divided by class, in each network: (i) Darknet D and Honeypot H independently; (ii) hosts observed in both networks (i.e. $D \cap H$); and (iii) the total number of unique hosts observed (i.e. $D \cup H$). A significant portion of the hosts is active in both networks (≈52%). The table highlights a high imbalance in the datasets towards the most popular class – *Mirai-like* – having around half of the hosts in both networks. Conversely, classes like *Michigan Uni.* or *SecurityTrails* are very small, with just a few dozen or a few hundred hosts. Such a high skewness in class distribution challenges learning-based methods, like ML stacking, and requires proper addressing to avoid biasing results towards the largest class. We address this issue in Section 4.

4. Experimental Setup

Our study is driven by two research questions: *RQ_1: What are the benefits of stacking in comparison with the baselines?* and *RQ_2: What is the benefit of combining multiple data sources?* Next, we describe the evaluation scenarios and methodology devised to address these questions.

4.1. Evaluation scenarios

Focusing on host classification use case, we derive three scenarios for testing the stacking alternatives. Each scenario characterises the data used as input to build the base classification models. They are:

Scenario 1 – Darknets only (D): only data from the darknet is used to build the three baselines (DS, text and graph), and the stacking method combines the produced class probabilities. As darknets provide mostly visibility on scanning traffic, here we test whether stacking methods help to better classify the categories of scanners.

Scenario 2 – Honeypots only (H): only data from the honeypots is used to build each of the base models. We combine their outputs with the meta-learner. Here we want to validate whether our approach and findings are verified in a different host classification scenario, in particular considering the completely different attacking steps observed in honeypots.

Scenario 3 – Both data, common hosts ($D \cap H$): in this scenario, we compare the stacking methods with different data sources, i.e. the stacking of the six base

[3] https://gitlab.com/mcollins_at_isi/acknowledged_scanners

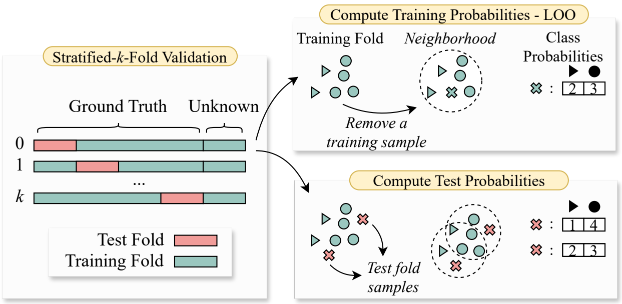


Figure 2: Overview of the validation methodology for ML stacking. Green indicates training fold samples, red indicates test fold samples and different shapes indicate different classes.

models built on different networks independently (three on D and three on H).

We use subscripts D, H and $D \cap H$ to distinguish across the scenarios. For a fair comparison, in this scenario, all methods are tested on the same dataset (i.e., data in $D \cap H$).

4.2. Evaluation Methodology

Using our ground-truth classes we build all models and perform predictions of the host categories using daily traffic. We use the per-class F1 Score as a primary evaluation metric. We then report the average per-class F1 Score along with the standard deviation across the 11 testing days. We also produce an aggregated result by averaging the daily per-class F1 Score, weighting all classes equally (macro-average, to avoid biases towards the more popular classes).

Baselines: We consider as baselines the six classifiers that use as input the daily embeddings produced by each representation learning technique. At each day we evaluate the whole embedding space through a leave-one-out validation [26]. In a nutshell, we iteratively consider each observation (i.e. host) as a test sample. For each test sample, we feed the embeddings of all the hosts as input to the kNN classifier, retrieving the class probabilities for the considered sample. We then assign the class with the highest probability to the test sample, repeating the process for all observations.

Naïve stacking: Since this method predicts the class of a test sample by just applying majority voting on the class probabilities output by different baselines, we employ the same leave-one-out validation strategy to evaluate its performance.

ML stacking: Since the ML approach requires the *training* of the meta-learner, we use a different validation methodology, i.e. a stratified 10-fold cross-validation (see Figure 2). In a nutshell, we split each daily dataset into 10 equally-sized folds; 9 folds are used for training the meta-learners and the remaining fold is used for testing. We rotate the fold 10 times so that each fold is used as a test set once.[4]

TABLE 2: Overview of the main results: Classification F1-Scores (11-day average) of all methods (baselines and stacking) and information sources. Best results in **bold**.

Scenario	Baseline			Stacking		Support
	DS	Text	Graph	Naïve	ML	
1: Darknet D	0.83	0.76	0.81	0.85	**0.90**	96 414
2: Honeypot H	0.73	0.86	0.74	0.82	**0.91**	131 191

Scenario	Stacking				Support
	Naïve	ML$_D$	ML$_H$	ML$_{D\cap H}$	
3: Common $_{D\cap H}$	0.91	0.91	0.91	**0.93**	81 505

At each round of the cross-validation, we first retrieve the class probabilities for each sample in the *training* from the set of kNN classifiers, restricting the leave-one-out methodology to the training samples only. To avoid biases via data leakage, we retrieve the probabilities for the *test* samples using only the training fold (i.e. for each *test* sample we consider its nearest neighbours among the *training* samples, as illustrated in the right bottom part of Figure 2). We then train the meta-learners with the probabilities for the training set. Finally, for testing, we apply the meta-learners to the test samples (using as input the probabilities obtained with the training) to produce the final class probabilities (and thus the class assignments) of the test samples.

Notice that for the baselines and naïve stacking we learn the class probabilities via leave-one-out on the entire dataset (but the test sample itself), whereas for the ML stacking, we learn the probabilities only from the training fold (9 folds out of 10). This slightly smaller training set (90% of the one used in the leave-one-out) may negatively impact the performance of the ML stacking. Yet, we understand this is a potential cost to be paid for using the ML stacking, which requires a different data split into training and test sets to train the meta-learners.[5]

The high prevalence of hosts labelled as "Unknown" and whose characteristics we cannot verify, could lead to misclassification in the learning process of the ML. Hence, we undersample the training data through random sampling, i.e. we reduce the size of the "Unknown" class to the size of the third largest class in the respective datasets.

We run our experiments following the same setup of [11], [12]. Hence, we generate text embeddings and graph embeddings $\in \mathbb{R}^{128}$ following the same methodology of the reference works. For the kNN, we set $k = 3$ and compute the neighbourhood through cosine distance.

5. Experimental Results

5.1. Overview of the main results

Table 2 summarizes the main results across all methods, datasets and scenarios. As shown, Meta-Learner (ML) is the overall best method, being superior not only to all baselines but also to the Naïve stacking approach.

When we use data from the darknet only (Scenario 1: D), both stacking methods outperform the best baseline (Domain-Specific - DS). The ML is the overall best

[4] Notice that, as discussed in Section 3, since we do not report the performance metrics for the hosts labelled as "Unknown", we include them always in the *training* fold, but disregard them in the test.

[5] Learning the model with a leave-one-out approach would lead to an extremely high computational cost and a lack of model generalization.

performer, with F1-Score gains of 7% over DS (0.90 compared to 0.83) and of 6% over naïve stacking. Hence, integrating complementary information by combining the baselines improves classification. Such a combination requires learning beyond majority voting.

When we use data from the honeypots only (Scenario 2: H), text embeddings lead to the best base results (0.86 F1-Score). Yet, ML is still the best performer, with F1-Score gains of $\approx 6\%$ over the baseline. Notably, the naïve stacking is not able to outperform the best baseline (0.82 compared to 0.86), possibly due to the poorer performance of the other two baselines, which may have biased the simple majority voting decisions. Intuitively, a naïve stacking approach is sensitive if "individual voters" do not perform well. This further demonstrates the power and flexibility of the ML stacking, which adapts to different scenarios, outperforming naïve stacking by $\approx 11\%$.

As shown in the bottom part of Table 2, relying on data from both networks (Scenario 3: $D \cap H$) benefits all stacking approaches. The combination of six baselines, as opposed to only three, explores more sources of information, bringing further improvements to ML (0.93 of F1-Score). Despite also naïve stacking benefits from more sources, the simple majority voting decisions are not robust.

All in all, despite the smaller support in this scenario (rightmost column in Table 2), ML$_{D \cap H}$ outperforms ML in both Scenarios 1 and 2. Hence, the benefits of bringing different sources of information outweigh the drawbacks of reducing the training data. In practical terms, ML$_{D \cap H}$ allows cybersecurity applications to increase precision when identifying the activity of hosts in the network, at the expense of missing some hosts not visible on both darknet and honeypot sensors.

5.2. Breaking down of the results per class

Table 3 shows the results for Scenario 2 (H) broken down per class. The relative performance of the baselines varies greatly across classes, with no overall single winner. This motivates the stacking of those methods to explore their complementarity, resulting in improved performance across most classes – ML is the best alternative for 10 out of 15 classes. When not strictly the best, ML$_H$ performs on par with the text embeddings baseline (as in the *Rapid7* and *ShadowServer* classes).

Notably, the performance gains for some classes are impressive, e.g. +11% of ML$_H$ over DS embeddings for *Exploiter*. Additionally, when one baseline strongly outperforms the others, ML seems to be able to "choose" the best source performing on par with it – e.g. *Ipip* for which the graph embeddings strongly influence the ML$_H$ (≈ 0.60 of F1-Score in both cases). As already mentioned, in such cases the naïve stacking is not robust enough to limit the contribution of the worst performers. Similar considerations can be drawn from Table 6 for Scenario 1 (D) in Appendix B.

In Table 4, we compare the stacking performance in Scenario 3. The best performer between ML$_D$ and ML$_H$ trained on embeddings learnt from a single network (3 sources) varies across classes with some large performance gaps. This is somehow expected since darknets and honeypots observe different types and stages of attacks. Indeed, the two networks offer complementary information favouring different classes.

TABLE 3: Classification average F1-Scores (11-day average). Hosts observed within Honeypot H. Best results are in **bold**.

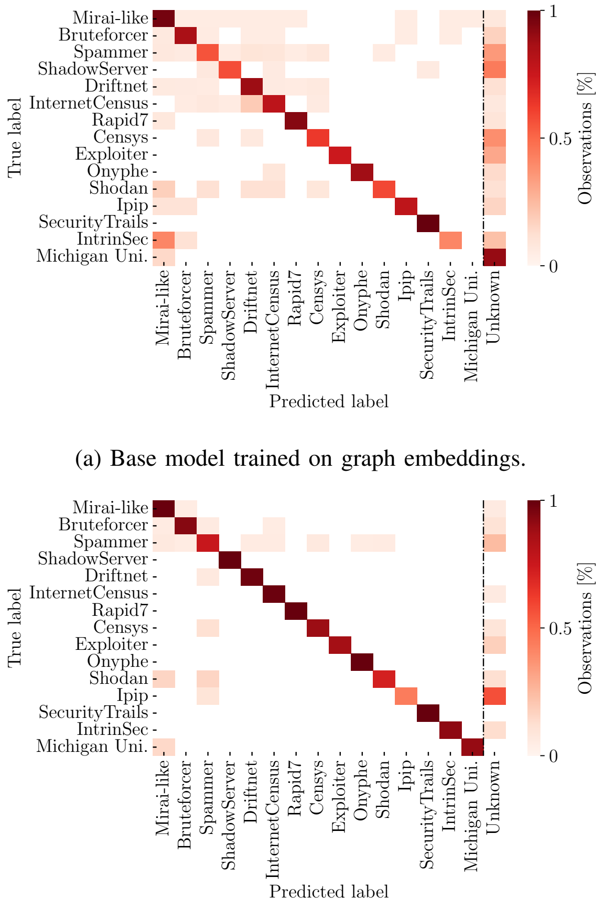
	Baseline			Stacking		
	DS	**Text**	**Graph**	**Naïve**	**ML$_H$**	**Support**
Mirai-like	0.97±0.01	**0.99±0.00**	0.98±0.00	**0.99±0.00**	**0.99±0.00**	92 077
Bruteforcer	0.91±0.01	0.90±0.02	0.91±0.01	0.94±0.01	**0.95±0.00**	18 194
Spammer	0.73±0.03	0.80±0.01	0.66±0.04	0.76±0.02	**0.83±0.03**	6 615
ShadowServer	0.71±0.03	**1.00±0.00**	0.70±0.03	0.94±0.01	0.99±0.00	3 074
Driftnet	0.92±0.03	0.96±0.01	0.79±0.05	0.89±0.02	**0.99±0.00**	3 456
InternetCensus	0.89±0.05	0.97±0.01	0.76±0.06	0.95±0.02	**0.98±0.00**	2 404
Rapid7	0.91±0.05	**1.00±0.00**	0.94±0.02	0.86±0.03	0.99±0.00	1 267
Censys	0.80±0.02	0.92±0.03	0.72±0.05	0.99±0.00	**0.93±0.03**	1 510
Exploiter	0.81±0.15	0.72±0.09	0.73±0.18	0.79±0.15	**0.90±0.04**	895
Onyphe	0.85±0.05	0.97±0.01	0.85±0.05	0.94±0.01	**0.98±0.00**	829
Shodan	**0.80±0.03**	0.75±0.07	0.65±0.10	0.76±0.06	0.78±0.04	315
Ipip	0.14±0.18	0.37±0.17	0.60±0.10	0.36±0.15	**0.61±0.15**	202
SecurityTrails	**1.00±0.00**	0.98±0.01	0.96±0.06	0.99±0.02	**1.00±0.00**	162
IntrinSec	0.22±0.18	**0.78±0.07**	0.41±0.15	0.57±0.13	0.74±0.33	149
Michigan Uni.	0.00±0.00	**0.98±0.03**	0.41±0.28	0.50±0.34	0.94±0.03	42
Average	0.73±0.04	0.86±0.02	0.74±0.03	0.82±0.03	**0.91±0.02**	131 191

TABLE 4: Comparing Meta-Learner (ML) classification F1-Scores (11-day average). Results are referred to hosts active in both networks $D \cap H$. The best average results are in **bold**.

	3 Sources		6 Sources		Support
	ML$_D$	**ML$_H$**	**Naïve**	**ML$_{D \cap H}$**	$D \cap H$
Mirai-like	0.99±0.00	0.99±0.00	**1.00±0.00**	0.99±0.00	62 955
Spammer	**0.87±0.01**	0.82±0.02	0.80±0.03	**0.87±0.01**	4 189
ShadowServer	**0.99±0.01**	**0.99±0.00**	0.97±0.01	**0.99±0.00**	3 074
Driftnet	0.99±0.00	0.99±0.00	**1.00±0.00**	0.99±0.00	2 772
InternetCensus	**0.99±0.00**	**0.99±0.00**	**0.99±0.00**	**0.99±0.00**	2 395
Bruteforcer	0.82±0.02	0.81±0.02	0.71±0.03	**0.90±0.01**	2 223
Censys	0.98±0.00	0.96±0.01	0.97±0.01	**0.99±0.00**	1 210
Rapid7	0.99±0.00	0.99±0.00	**1.00±0.00**	**1.00±0.00**	1 201
Onyphe	**0.99±0.00**	0.98±0.00	**0.99±0.01**	**0.99±0.00**	725
Shodan	**0.85±0.02**	0.80±0.03	0.84±0.04	0.84±0.03	292
SecurityTrails	**1.00±0.00**	**1.00±0.00**	**1.00±0.01**	**1.00±0.00**	162
Ipip	0.97±0.03	0.83±0.09	0.94±0.06	**0.99±0.01**	132
Exploiter	0.38±0.28	**0.64±0.17**	0.56±0.27	0.53±0.30	91
IntrinSec	0.87±0.04	0.90±0.04	0.92±0.08	**0.96±0.04**	54
Michigan Uni.	**0.98±0.02**	0.92±0.02	**0.98±0.03**	**0.98±0.02**	30
Average	0.91±0.02	0.91±0.02	0.91±0.02	**0.93±0.02**	81 505

Relying on data from both networks (6 sources), ML$_{D \cap H}$ improves both the single network performance ML$_D$, ML$_H$ and the naïve stacking. Indeed, the ML-based stacking produces the best results for 11 out of 15 classes and delivers performance on par with the single network stacking for 3 other classes. The only exception is *Exploiter*, for which ML$_H$ significantly outperforms ML$_{D \cap H}$. For instance, the *Exploiter* class is composed of hosts that achieve the deeper stages in the attack chains observed in the honeypots. As darknets provide mostly information about scanning traffic, their contribution to the identification of this class is expected to be marginal. Therefore, ML$_{D \cap H}$ is penalized by the reduction in coverage during training and the expected lack of quality of classifiers built with darknet traffic alone. Indeed, the ML$_H$ model is trained on 829 *Exploiter* samples, compared to ML$_{D \cap H}$, trained on 91 hosts active in both the networks (compare results and support for this class in Table 3 and Table 4).

Finally, we note that the gains of $ML_{D \cap H}$ are obtained in the presence of a very high data skewness:



(a) Base model trained on graph embeddings.

(b) ML_H model trained on 3 sources.

Figure 3: Confusion matrix from the final classification task. Scenario 2 (H). One testing day, i.e. 2022-10-28.

the majority class (Mirai-like) corresponds to 78% of the instances, and is roughly 15× larger than the second largest class. Such large skewness is a challenge to any classifier due to the bias towards the majority class. Yet, as Table 4 shows, $ML_{D \cap H}$ manages to achieve almost perfect results even for some of the smallest classes. The undersampling procedure of the unknown class used in the training of the meta-learners (see Section 4) helps mitigate this issue. Nevertheless, even with the undersampling, the good performance in some of the smallest classes is positively surprising. Next, we delve deeper into the explanations of these results.

6. Explaining the Stacking Results

6.1. Classification Results

We focus on a sample day of Scenario 2 (honeypots) and compare confusion matrices resulting from two approaches: (i) the base model trained on graph embeddings (Figure 3a) and (ii) ML_H trained on three different embeddings (Figure 3b). In each matrix, the cell indicates the number of samples of a *true* class assigned to the *predicted* classes. See also Figure 5, Appendix C for the matrices with the relative values.

We see that ML_H drastically reduces the confusion among the classes. The graph embeddings used in Figure 3a deliver information about the relationships between hosts and the contacted honeypot services. Despite the good classification performance (notice the almost diagonal matrix that indicates correct class assignments), classes like Mirai-like, Bruteforcer and Spammer are problematic – see the light-red area in the top-left corner of Figure 3a. It is largely expected that the hosts performing brute-force attacks or trying to send spam generate high-intensity traffic to only a few services. This is also the case for the typically aggressive scans performed by Mirai-like bots. From Figure 3a we conjecture that the graph embeddings have not captured (and cannot capture) these characteristics well.

Conversely, Figure 3b confirms that combining heterogeneous sources of information represented by the three embeddings drastically improves the classification performance. More than 80% of the senders are correctly classified (values on the diagonal). Including information related to the temporal co-occurrence of hosts (text embeddings) and the intensity of generated traffic (DS embeddings) drastically reduces the number of misclassified samples. Notice also the reduced number of samples misclassified as Unknown. Similar observations can be made for the comparison with the other methods.

6.2. Meta-Learner Coefficients

Recall we adopt a set of LR functions as meta-learner, which output a final probability for each target class given the base models' probabilities. An advantage of using LR is that we can analyse the final coefficients to infer the relative importance (and contribution) of different inputs to the final classification.[6]

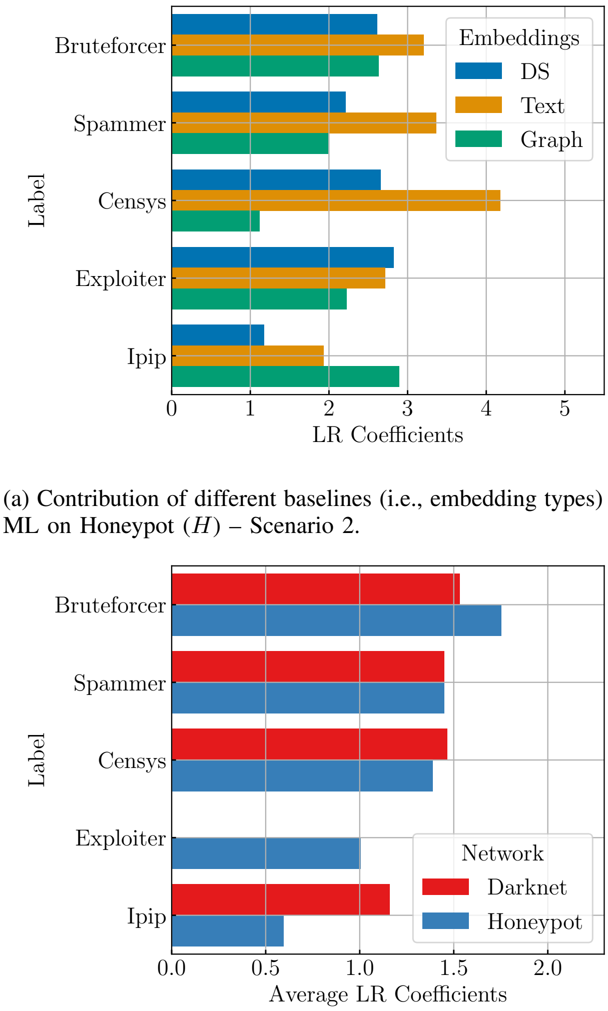
Embeddings contribution We firstly focus on Figure 4a, which reports the LR coefficients of the base models trained on different embeddings obtained in one day of Scenario 2. We report results for five classes. We consider the coefficient values (x-axis) as a measure of the relative importance the meta-learner gives to each base classifier when assigning samples to a given label (y-axis). For the sake of completeness, we report the complete set of coefficients for all classes in Figure 6, Appendix C.

As shown in the figure, the contributions of the baselines to the ML classification vary across classes. For instance, for high-intensity traffic classes, like Bruteforcer and Spammer, the meta-learner assigns almost equal importance to the three baselines (which is consistent with their roughly similar performance in Table 3), with an emphasis on the text baseline, which reflects the temporal co-occurrence of hosts [12] indicating similar scanning activities.

For the Censys class, the meta-learner gives much more importance to the text embedding classifier. This is expected, considering the known extreme coordination of Censys hosts in their scanning routine [12].

Conversely, for the Exploiter class, the domain-specific features contribute slightly more than the other

[6] The comparison of coefficients of different LR functions can be done since all functions take as input the same class probabilities, which are normalised values between 0 and 1.



(a) Contribution of different baselines (i.e., embedding types) to ML on Honeypot (H) – Scenario 2.

(b) Contribution of different networks on common hosts ($D \cap H$) – Scenario 3.

Figure 4: Logistic Regression coefficients for one testing day, i.e. 2022-10-28.

baselines, whereas for Ipip, the graph embeddings have the highest coefficients. This is in line with the superior performance of text and graph baselines for the respective classes in Table 3 and testifies that these classes exhibit peculiar patterns in the DS and Graph features, respectively.

In a nutshell, the meta-learner successfully learns to differently weight the baselines based on their performance for each class, reducing misclassifications.

Data contribution: Finally, we focus on one day of Scenario 3, $\text{ML}_{D \cap H}$ trained on data observed from honeypots and darknet networks. In Figure 4b, for each class, we compute the average of the LR coefficient over the three base models. We consider the average values as a proxy for the importance the meta-learner gives to the information coming from the different sources of data (i.e. darknet or honeypots).

Interestingly, for some classes whose ground truth labels can only be derived from honeypot logs, such as Bruteforcer and Exploiter, the meta-learning uses mostly honeypot-based classifiers for its decision. Notice how the embeddings from honeypots have high coefficients for the Bruteforcer class. Similar behaviour is expected for the Exploiter class. The meta-learner considers *only* honeypot-based classifiers for this class, as none of the hosts have been active in the darknet on the considered day.

In the case of Spammer and Censys for which stacking three baseline models from the same network leads to comparable performance (see ML_D and ML_H in Table 4), the meta-learner weights almost equally the two sources of information improving the final classification. Conversely, for the Ipip class, stacking models trained on darknet embeddings (ML_D in Table 4) leads to superior performance than the honeypot case (ML_H in Table 4). Once again, the meta-learner successfully learns how to combine the base predictions favouring the source with the higher informative content (i.e. the darknet).

All in all, by investigating the LR coefficients we can provide insights on the informative content the different sources bring. In most cases, this combination follows the intuition coming from our domain knowledge. This further corroborates the benefit of combining heterogeneous information through model stacking while providing the analysts useful insights on which features drive the most the classification process.

7. Conclusions

We investigated the benefits of stacking models trained on embeddings generated with different techniques and from different types of input data for cybersecurity applications. Our results show that (i) combining different embeddings learnt from the same network leads to improvements in solving host classification; (ii) combining information from the different networks, particularly with meta-learning techniques, allows enriching the representations improving the quality of the embedding and leads to the best classification results; (iii) stacking models through meta-learning provides some degree of interpretability, allowing to understand the different contribution of each source of information to the final classification.

Future developments will include the exploration of other networks beyond the cybersecurity application and the exploration of other embedding generation techniques, along with the impact of different meta-learning models and strategies. Additionally, the adoption of more sophisticated explainability techniques, like SHAP, could deepen the understanding of the obtained representations.

Ethical Considerations

Our work focuses on technical advancements in using model stacking to enhance cybersecurity and deepen the knowledge of traffic observed in different networks. There is no sensitive information in the data we collect. As we do not probe deeper into the identity of the sender or gather additional personal information, no further ethical considerations apply to our research.

Our primary objective is to improve the effectiveness of network security measures through innovative AI solutions.

Acknowledgement

The research leading to these results has been partly funded by the project SERICS (SEcurity and RIghts In the CyberSpace - PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union, as well as the ACRE (AI-Based Causality and Reasoning for Deceptive Assets - 2022EP2L7H) and xInternet (eXplainable Internet - 20225CETN9) projects - funded by European Union - Next Generation EU within the PRIN 2022 program (D.D. 104 - 02/02/2022 Ministero dell'Università e della Ricerca). This manuscript reflects only the authors' views and opinions and the Ministry cannot be considered responsible for them. This work was also supported by Brazil's CNPq, CAPES, Fapemig, Fapesp and AWS.

References

[1] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges. *IEEE Transactions on Network and Service Management*, 2019.

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *26th USENIX security symposium (USENIX Security 17)*, 2017.

[3] Llorenç Cerdà-Alabern, Gabriel Iuhasz, and Gabriele Gemmi. Anomaly detection for fault detection in wireless community networks using machine learning. *Computer Communications*, 2023.

[4] Dvir Cohen, Yisroel Mirsky, Manuel Kamp, Tobias Martin, Yuval Elovici, Rami Puzis, and Asaf Shabtai. DANTE: A Framework for Mining and Monitoring Darknet Traffic. In *Computer Security – ESORICS 2020*, 2020.

[5] D. R. Cox. The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1958.

[6] Washington Cunha, Celso França, Guilherme Fonseca, Leonardo Rocha, and Marcos André Gonçalves. An Effective, Efficient, and Scalable Confidence-based Instance Selection Framework for Transformer-Based Text Classification. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023.

[7] Pádraig Cunningham and Sarah Jane Delany. k-Nearest Neighbour Classifiers - A Tutorial. *ACM Computing Surveys*, 2021.

[8] Claudio M. V. de Andrade, Fabiano M. Belém, Washington Cunha, Celso França, Felipe Viegas, Leonardo Rocha, and Marcos André Gonçalves. On the class separability of contextual embeddings representations – or "The classifier does not matter when the (text) representation is so good!". *Information Processing & Management*, 2023.

[9] Mohamed Amine Ferrag, Leandros Maglaras, Sotiris Moschoyiannis, and Helge Janicke. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 2020.

[10] Jing Gao, Peng Li, Zhikui Chen, and Jianing Zhang. A Survey on Deep Learning for Multimodal Data Fusion. *Neural Computation*, 2020.

[11] Luca Gioacchini, Andrea Cavallo, Marco Mellia, and Luca Vassio. Exploring Temporal GNN Embeddings for Darknet Traffic Analysis. In *Proceedings of the 2nd on Graph Neural Networking Workshop 2023*, 2023.

[12] Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, Zied Ben Houidi, and Dario Rossi. i-DarkVec: Incremental Embeddings for Darknet Traffic Analysis. *ACM Transactions on Internet Technology*, 2023.

[13] github rashelbach. T-Pot - The All In One Honeypot, 2024.

[14] Christian Gomes, Marcos Goncalves, Leonardo Rocha, and Sergio Canuto. On the Cost-Effectiveness of Stacking of Neural and Non-Neural Methods for Text Classification: Scenarios and Performance Prediction. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 2021.

[15] Idio Guarino, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescapè. Explainable Deep-Learning Approaches for Packet-Level Traffic Prediction of Collaboration and Communication Mobile Apps. *IEEE Open Journal of the Communications Society*, 2024.

[16] Eyal Horowicz, Tal Shapira, and Yuval Shavitt. A few shots traffic classification with mini-FlowPic augmentations. In *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022.

[17] Zied Ben Houidi, Raphael Azorin, Massimo Gallo, Alessandro Finamore, and Dario Rossi. Towards a systematic multi-modal representation learning for network data. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, 2022.

[18] Jonas Höchst, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben. Unsupervised Traffic Flow Classification Using a Neural Autoencoder. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, 2017.

[19] Michalis Kallitsis, Rupesh Prajapati, Vasant Honavar, Dinghao Wu, and John Yen. Detecting and Interpreting Changes in Scanning Behavior in Large Network Telescopes. *IEEE Transactions on Information Forensics and Security*, 2022.

[20] Seungjin Lee, Azween Abdullah, Nz Jhanjhi, and Sh Kok. Classification of botnet attacks in iot smart factory using honeypot combined with machine learning. *PeerJ Computer Science*, 2021.

[21] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification. In *Proceedings of the ACM Web Conference 2022*, 2022.

[22] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip S. Yu. Graph Self-Supervised Learning: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space, 2013.

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, 2013.

[25] Preeti Mishra, Vijay Varadharajan, Uday Tupakula, and Emmanuel S. Pilli. A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 2019.

[26] Annette M. Molinaro, Richard Simon, and Ruth M. Pfeiffer. Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, 2005.

[27] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. Towards the Deployment of Machine Learning Solutions in Network Traffic Classification: A Systematic Survey. *IEEE Communications Surveys & Tutorials*, 2019.

[28] Shahbaz Rezaei and Xin Liu. Deep Learning for Encrypted Traffic Classification: An Overview. *IEEE Communications Magazine*, 2019.

[29] Markus Ring, Alexander Dallmann, Dieter Landes, and Andreas Hotho. IP2Vec: Learning Similarities Between IP Addresses. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017.

[30] Bushra Sabir, Faheem Ullah, M. Ali Babar, and Raj Gaire. Machine Learning for Detecting Data Exfiltration: A Review. *ACM Computing Surveys*, 2021.

[31] Amin Shahraki, Mahmoud Abbasi, Amir Taherkordi, and Mohammed Kaosar. Internet Traffic Classification Using an Ensemble of Deep Convolutional Neural Networks. In *Proceedings of the 4th FlexNets Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility*, 2021.

[32] Francesca Soro, Mauro Allegretta, Marco Mellia, Idilio Drago, and Leandro M. Bertholdo. Sensing the Noise: Uncovering Communities in Darknet Traffic. In *2020 Mediterranean Communication and Computer Networking Conference (MedComNet)*, 2020.

[33] Yang Yang, Yu Yan, Zhipeng Gao, Lanlan Rui, Rui Lyu, Bowen Gao, and Peng Yu. A Network Traffic Classification Method Based on Dual-Mode Feature Extraction and Hybrid Neural Networks. *IEEE Transactions on Network and Service Management*, 2023.

[34] Ruijie Zhao, Mingwei Zhan, Xianwen Deng, Yanhao Wang, Yijun Wang, Guan Gui, and Zhi Xue. Yet Another Traffic Classifier: A Masked Autoencoder Based Traffic Transformer with Multi-Level Flow Representation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Appendix

1. Host Features

We here complement the information about the embeddings generation explained in Section 2.

TABLE 5: Host and port domain-specific (DS) embeddings extracted through standard features engineering.

Host Features		**Port Features**	
#Contacted network ports		#Hosts contacting a port	
STATS(#Packets per network port)		STATS(#Packets per source host contacting a port)	
#Contacted network hosts		0-valued dummy feature	
STATS(#Packets per network host)		0-valued dummy feature	

Packets per source host	STATS(Size)
	STATS(TTL)
	STATS(MSS)
	STATS(WIN)
	STATS(TS)
		Packets per network port	STATS(Size)
			STATS(TTL)
			STATS(MSS)
			STATS(WIN)
			STATS(TS)

In Table 5 we provide an overview of the features generated for source hosts targeting a network and the destination ports. The function $\text{STATS}(\cdot)$ extracts the sum, minimum, maximum, average and standard deviation of the provided entity.

Notice that, in the case of DS embeddings, we produce only the *Host Features*. In the case of tGNN embeddings, we assign features to both hosts and port nodes of the graph.

2. Darknet Classification Performance.

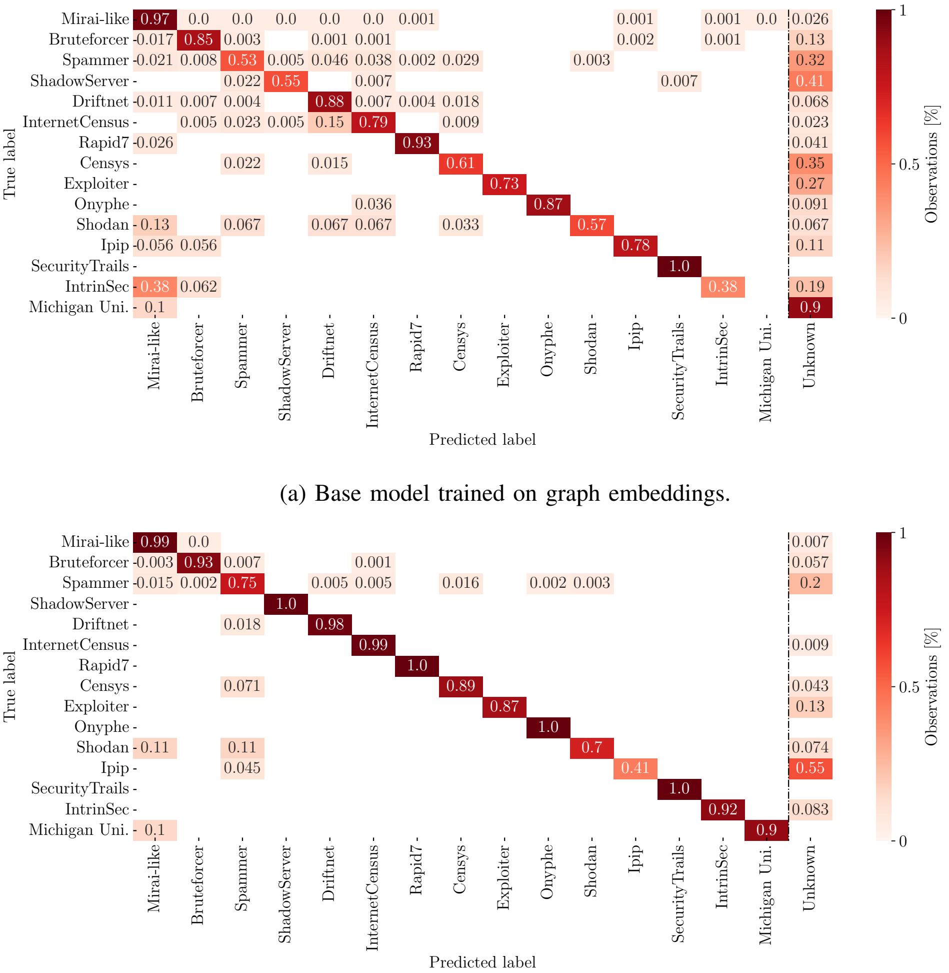
Table 6 complements the per-class results of the downstream classifier discussed in Section 5.

3. Explainability

We here complement the analysis of the Meta-Learner explainability provided in Section 6. In Figure 5 we report the confusion matrix obtained when classifying hosts observed only within Honeypot (Scenario 2 – H) during 2022-10-28, Model ML_H. In Figure 6, we show the complete set of LR coefficients for Scenario 2 of Figure 4.

TABLE 6: Classification average F1-Scores over 11 testing days. Hosts observed within the darknet D. Best average results are in **bold**.

	Baseline			**Stacking**		
	DS	**Text**	**Graph**	**Naïve**	**ML$_D$**	**Support**
Mirai-like	**1.00±0.00**	0.99±0.00	**1.00±0.00**	**1.00±0.00**	0.99±0.00	76 176
Spammer	0.77±0.03	0.76±0.02	0.74±0.02	0.77±0.03	**0.86±0.01**	5 395
ShadowServer	0.70±0.03	**0.99±0.00**	0.70±0.02	0.88±0.02	**0.99±0.00**	3 074
Driftnet	**1.00±0.00**	0.98±0.01	0.98±0.01	0.99±0.00	0.99±0.00	2 772
Bruteforcer	0.49±0.04	0.44±0.05	0.50±0.05	0.46±0.05	**0.75±0.03**	2 618
InternetCensus	0.95±0.03	0.97±0.01	0.91±0.04	0.97±0.01	**0.99±0.00**	2 400
Rapid7	0.98±0.01	**1.00±0.00**	0.97±0.02	**1.00±0.00**	0.99±0.00	1 255
Censys	0.97±0.02	0.71±0.05	0.81±0.04	0.90±0.03	**0.98±0.00**	1 210
Onyphe	0.95±0.03	0.96±0.02	0.93±0.03	0.97±0.01	**0.99±0.00**	728
Shodan	**0.88±0.03**	0.76±0.07	0.67±0.12	0.81±0.04	0.84±0.02	295
SecurityTrails	0.96±0.04	**1.00±0.01**	0.93±0.03	0.96±0.02	**1.00±0.00**	162
Ipip	**0.98±0.04**	0.07±0.08	**0.98±0.02**	**0.98±0.04**	0.97±0.03	132
Exploiter	0.37±0.22	0.20±0.28	**0.38±0.19**	0.29±0.20	0.32±0.23	111
IntrinSec	0.49±0.23	**0.88±0.10**	0.73±0.19	0.85±0.08	0.87±0.06	56
Michigan Uni.	0.63±0.10	**0.98±0.03**	**0.98±0.03**	**0.98±0.03**	**0.98±0.02**	30
Average	0.83±0.03	0.76±0.03	0.81±0.03	0.85±0.03	**0.90±0.02**	96 414



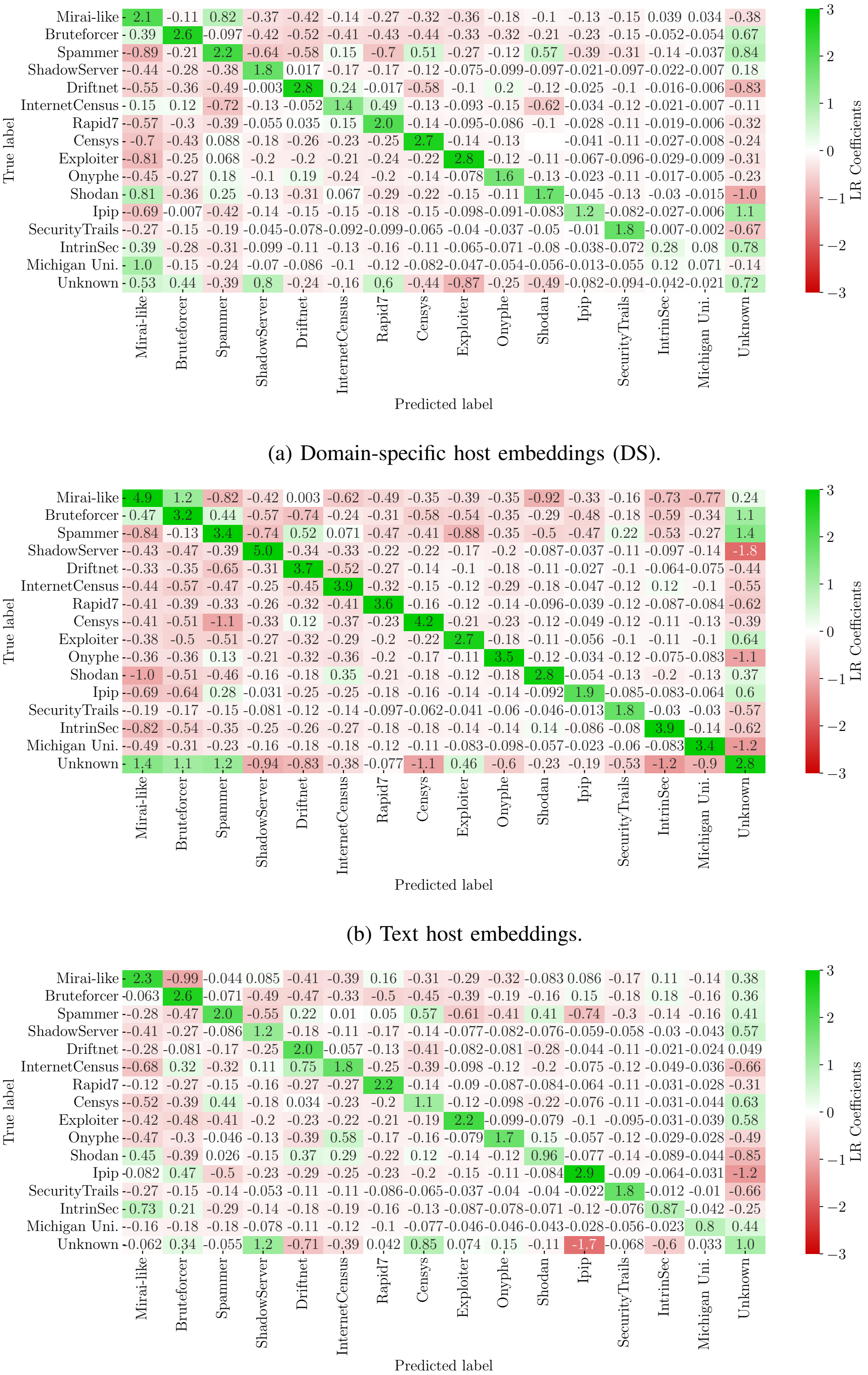
True label	Mirai-like	Bruteforcer	Spammer	ShadowServer	Driftnet	InternetCensus	Rapid7	Censys	Exploiter	Onyphe	Shodan	Ipip	SecurityTrails	IntrinSec	Michigan Uni.	Unknown
Mirai-like	0.97	0.0	0.0	0.0	0.0	0.0	0.001					0.001		0.001	0.0	0.026
Bruteforcer	0.017	0.85	0.003		0.001	0.001						0.002		0.001		0.13
Spammer	0.021	0.008	0.53	0.005	0.046	0.038	0.002	0.029			0.003					0.32
ShadowServer			0.022	0.55		0.007							0.007			0.41
Driftnet	0.011	0.007	0.004		0.88	0.007	0.004	0.018								0.068
InternetCensus		0.005	0.023	0.005	0.15	0.79		0.009								0.023
Rapid7	0.026						0.93									0.041
Censys			0.022		0.015			0.61								0.35
Exploiter									0.73							0.27
Onyphe						0.036				0.87						0.091
Shodan	0.13		0.067		0.067	0.067		0.033			0.57					0.067
Ipip	0.056	0.056										0.78				0.11
SecurityTrails													1.0			
IntrinSec	0.38	0.062												0.38		0.19
Michigan Uni.	0.1															0.9

(a) Base model trained on graph embeddings.

True label	Mirai-like	Bruteforcer	Spammer	ShadowServer	Driftnet	InternetCensus	Rapid7	Censys	Exploiter	Onyphe	Shodan	Ipip	SecurityTrails	IntrinSec	Michigan Uni.	Unknown
Mirai-like	0.99	0.0														0.007
Bruteforcer	0.003	0.93	0.007			0.001										0.057
Spammer	0.015	0.002	0.75		0.005	0.005		0.016		0.002	0.003					0.2
ShadowServer				1.0												
Driftnet			0.018		0.98											
InternetCensus						0.99										0.009
Rapid7							1.0									
Censys			0.071					0.89								0.043
Exploiter									0.87							0.13
Onyphe										1.0						
Shodan	0.11		0.11								0.7					0.074
Ipip			0.045									0.41				0.55
SecurityTrails													1.0			
IntrinSec														0.92		0.083
Michigan Uni.	0.1														0.9	

(b) ML$_H$ model trained on 3 sources.

Figure 5: Confusion matrix resulting from the final classification task. Scenario 2 (H). One testing day, i.e. 2022-10-28.



(a) Domain-specific host embeddings (DS).

(b) Text host embeddings.

(c) Graph host embeddings.

Figure 6: Logistic Regression coefficients for one of the testing days, i.e. 2022-10-28. Contribution of different embedding types on Honeypot (H) –Model ML$_H$. Scenario 2.