17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID # 151

Towards INtelligent automated Functional and Security Testing (INFAST)

D. López-Montero[a]*, A. Benítez-Buenache[a], N. Carvalho-dos Santos[a], J. Gallardo-Lozano[a], M. J. Prokopczyk[a], Y. Al-Khazraji[a], A.M. García-Sánchez[a], M. Matuszak[a], M. Pinto[b], E. Marques[b], E. Ntagiou[c], M. Wallum[c]

[a] *GMV. Department of Artificial Intelligence and Big Data, Isaac Newton, 11, Tres Cantos, Madrid, Spain*, daniel.lopez.montero@gmv.com, abenitez@gmv.com, ncarvalho@gmv.com, jvgallardo@gmv.com, mprokopczyk@gmv.com, yalkhazraji@gmv.com, amgarcia@gmv.com, mmatuszak@gmv.com
[b] *ETAMAX Space GmbH. Lilienthalplatz 1, D-38108 Brunswick, Germany,* pinto@etamax.de, marques@etamax.de
[c] *ESA-ESOC. Ground Systems Engineering and Innovation Department, European Space Operations Centre, Robert-Bosch Strasse 5, 64293 Darmstadt, Germany,* Evridiki.Ntagiou@esa.int, Marcus.Wallum@esa.int
* Corresponding Author

Abstract

Running tests to ensure the correct functioning of a system is commonplace. This increases the time and effort required to define, run, and analyse the results of these tests, usually involving very repetitive tasks for the operators. This practice is common in the European Space Operations Centre, which is the reason for the INFAST project. This project aims to use new solutions based on artificial intelligence to automate functional and security tests in order to improve the correct functioning of all components, reduce the cost of project maintenance and detect security problems or vulnerabilities. For the automatic analysis of functional tests and the identification of possible root causes, the use of Natural Language Processing is proposed. In this way, relationships between test failures are extracted from their texts (e.g., descriptions or logs), with the aim of grouping them and facilitating the identification of possible root causes. The presented approach is based on encoding information using transformers and relating different sources using cosine similarity. After a period of experimentation with real data, the proposed approach meets expectations by being able to adequately cluster and correlate test failures. On the other hand, Reinforcement Learning is proposed for the prioritisation of penetration test tasks. The presented approach is based on the generation of simulated network topology environments for training and the use of Graph Neural Networks as a learning model. This type of neural network has shown promising results in the simulated environments, offering advantages over more traditional architectures. After the conducted study, it was possible to obtain an algorithm that prioritises tasks to be executed during the pentesting process, depending on the environment in which it is found. Finally, the project also explores the use of Explainable Artificial Intelligence, which provides explanations for the decisions made by the employed models, thus avoiding their use as black-box models.

Keywords: Test Automation; Natural Language Processing; Pentesting; Reinforcement Learning; Graph Neural Networks; Explainable Artificial Intelligence

Nomenclature

The set of first n numbers $[n] \coloneqq \{1, \dots, n\}$; graph $G = (V, E)$ with nodes V, and edges E; the concatenation of sequences/vectors two vectors $u \| v$; the product of two sets $A \times B = \{(a, b) : a \in A, b \in B\}$.

Acronyms/Abbreviations

INFAST	INtelligent automated Functional and Security Testing
ESA	European Space Agency
ESOC	European Space Operations Centre
EGS-CC	European Ground Systems–Common Core
AI	Artificial Intelligence
MCS	Mission Control System
SPRs	Software Problem Reports
NLP	Natural Language Processing
Penbox	Penetration Testing and Security Awareness Management in a Box
RL	Reinforcement Learning
XAI	Explainable Artificial Intelligence
MDP	Markov Decision Process
POMDP	Partial Observable Markov Decision Process

SVMs	Support Vector Machines
ART	Automated Regression Testing
MMIT	Man-Machine Interface Testing
EGOS-CC	ESA Ground Operation Software-Common Core
E2EVal	End-to-End Validation
SUT	System Under Test
TF-IDF	Term Frequency-Inverse Document Frequency
BM-25	Best Match 25
MLP	Multi-Layer Perceptron
GNN	Graph Neural Network
GATConv	Graph Attention Convolutional Layer
GCN	Graph Convolutional Network
DQN	Deep Q-Learning
PPO	Proximal Policy Optimization
A2C	Advantage Actor Critic
TPE	Tree-structured Parzen Estimator

1. Introduction

At the European Space Operations Centre (ESOC), complex systems are developed, maintained, and used on a daily basis, with the aim of supporting spacecraft operations on various missions. These systems constitute the ground segment and provide end users with a range of functionalities that are becoming increasingly complex. This increasing complexity leads to a proliferation of test cases to ensure test coverage and increased effort to support test definition, execution, and analysis of results. Under time and cost pressure, incomplete test coverage endangers the functional and security aspects of the mission. Therefore, testing plays a fundamental role in the development of projects, in order to: verify compliance with system requirements, prevent future errors, and detect errors in early phases to improve and ensure the correct functioning of all components, facilitate integration after changes, reduce the cost of project maintenance, and detect security issues or vulnerabilities that can be exploited. As will be seen in the next section, test automation is a very interesting field of study. Therefore, projects such as INFAST serve to investigate this topic and look for solutions that facilitate the testers' tasks.

ESOC and the software industry spend a lot of effort validating the functional and non-functional requirements of their critical platforms. The introduction of the next generation Mission Control System (MCS) software solutions based on the European Ground Systems–Common Core (EGS-CC) framework is a good occasion to evaluate new technologies in the testing process. Given that much of the testing effort is spent in performing repetitive tasks, it seems reasonable that recent advances in Artificial Intelligence (AI), e.g., Machine Learning, could help in the optimization of such testing processes, possibly leading to a higher level of automation in tasks such as test definition, test execution planning, analysis of results, etc. Thus, the INFAST project has identified some of the most promising AI-based techniques for automating tests, both functional and security, to be applied in the mission data system context. Once identified, a prototype tool to automate testing tasks for two main use cases (functional and security) is developed and designed as a proof of concept.

On one hand, the use case for functional testing focuses on the results analysis phase, by means of a system that assists the tester in the errors classification and correlation. For this purpose, different data sources are used, such as: test reports (including test failures), Software Problem Reports (SPRs), log files and the history of source code commits. Therefore, the main objective is to analyse these data sources, to group failed test steps and identify relationships with previous events: Similar past failures, existing SPRs (open or closed), etc. In fact, in case no relationships with previous events are found, the new group of similar failures is identified as a potential new SPR.

As it can be seen throughout the following sections, Natural Language Processing (NLP) is proposed as the main AI technology to obtain the similarities. Due to the large diversity among possible sources and formats, a pre-processing of the data is necessary to extract the information. In fact, the identified data sources do not strictly follow the rules of natural language, making the use case challenging. Once the information has been extracted, to convert the texts into a vector space by means of embeddings, the use of pre-trained models based on transformers is proposed. Finally, these embeddings will allow the use of similarity calculation techniques (such as cosine similarity) to identify past events related to the current failure. Moreover, one of the project's critical objectives is not only to provide a result for the end user, but also to explain what has conditioned this decision. Hence, the use of Explainable AI techniques is also proposed, allowing to provide the most relevant features detected by the solution in a comprehensible format for humans, increasing the level of trust to the AI system.

On the other hand, the focus of the security testing use case has been set on penetration testing ("pentesting") tasks, a key element within cybersecurity to analyse the robustness and security of a system against possible attacks or vulnerabilities. In pentesting, a series of actions are carried out to compromise remote targets by analysing and obtaining information from the targets, and exploiting the vulnerabilities found during the security analysis. However, choosing the most promising action to achieve the goal can be complex and requires a high level of expertise. Obviously, brute-force plans could be carried out, executing all possibilities in an orderly manner, but this is a problem both in terms of time and possible risk of Denial-of-Service (possible cause of unavailability of the resources included in the network environment under test). Penbox (Penetration Testing and Security Awareness Management in a Box) is a proof-of-concept tool developed by ESA to automate the execution of different scans and exploits on a target network. Therefore, another objective of the project is the use of AI techniques to automate the selection of the next actions to be executed by PenBox or a similar tool.

For this second purpose, the use of Reinforcement Learning is proposed, training an agent to select the next action based on the available information and network configurations. This information would be given by the knowledge about the system under test, the success of the previous action, and the observations that occur during its execution. In this way, a reward function is modelled with the aim to maximize it, by selecting the most promising actions to achieve the system compromise, and the pentesting objectives.

Throughout the following sections, the use of different Reinforcement Learning algorithms for the design on the agent is proposed, finally selecting the one with the best performance. In addition, a network/topology simulation environment is used for training for two reasons: First, to avoid the denial-of-service of a real network environment during the process; and second, to obtain a trained solution with sufficient diversity to generalise.

This present study is structured starting with a brief review about the state-of-the-art in Section 2; Section 3 describes the use case implemented for the automation of analysis tasks in executed functional tests; Section 4 shows the use case of pentesting task automation; Section 5 describes the Explainable AI (XAI) techniques employed to avoid the use of black-box models; and finally, some conclusions and future work are presented in Section 6.

2. State-of-the-art in test automation

As usual, when a need is identified, lines of research emerge that look at how to address the problem. The case of test automation is no exception and in recent years –especially since the rise of Machine Learning– several studies have appeared to solve it. However, the approach will be very different depending on the type of test and the type of task (generation, execution, or maintenance, to cite a few examples) to be automated.

AI-based solutions are intended to have a major impact on test automation through planning, authoring, development, and maintenance tasks. This type of solution is not yet mature; however, it is a line of research of great interest as can be seen in the survey papers [1], [2] about functional test automation. Within these studies, use cases can be found such as:

- Test generation: The generation of new tests requires expert knowledge. The use of AI can help in the definition of tests based on available information (test and requirements documents, existing tests, detection of system vulnerabilities, etc.).
- Debugging: In the search for test failures, analysis tasks are essential, highlighting root cause analysis, code coverage and runtime monitoring. These tasks require the presence of an expert, and their automation may even be useful as a pre-filtering task. As described in the following sections, one of the selected use cases in the INFAST project falls into this category.
- Maintenance: To check the correct functioning of the systems, regular testing is necessary. This results in long runtimes that can compromise the operation of the system, so the prioritization and selection of the tests to be executed can play a fundamental role.
- Oracle: In this case, the focus is on validation of the visual correction of a GUI by means of AI techniques capable of detecting anomalies imperceptible to the human eye.

On the other hand, the field of Cybersecurity has experienced an unprecedented growth in the last decades because of the improvement in technology, hardware, and algorithms (Machine Learning among them). Until a few years ago, cybersecurity required the supervision and manipulation of experts and highly specialised professionals. Due to the increasing complexity of this task, with the aim of automating and facilitating the supervision of human experts, in-depth research has been done over the last few years. It is worth mentioning the following applications:

- Identification of the most promising attack path based on the network environment (configuration and available information) to automatically find new vulnerabilities.
- Estimation of vulnerability risk with the aim of scoring the robustness of a network based on the graph structure (weakly/strongly connected graph) and initial configuration.

- Based on the state of the network in a penetration test, recommend through an AI assistant the actions to be taken by the operator to find possible vulnerabilities.
- Among the large number of vulnerabilities identified in a network, determining which ones present a best balance between the impact that cause their exploitation and the cost of fixing them.
- Merge/changes evaluation. Examining the cybersecurity posture and associated risk of a merger, acquisition or investment target is a critical component of any due diligence process. Find configurations that allow attacks before it is deployed. Measure the vulnerabilities that emanate from changes in the network structure. The persistent technique combines opportunities like cached credentials and misconfigurations into new attack paths.

Most cybersecurity publications in the field of Machine Learning focus on anomaly and security breach detection [3]. However, the automation of pentesting tasks has also attracted a lot of interest recently [4], [5], the most outstanding being those that base their solution on POMDP [6] and Reinforcement Learning [7]–[9] approaches, as described below. This type of solutions, which is the focus of the second use case of the INFAST project, is part of the Breach and Attack Simulation (BAS) field. The great contribution of this approach is the absence of human dependence with the main objective of automating the entire process, deciding at each moment the next action (scan, exploit, repeat...) in the design of an attack for vulnerabilities detection.

2.1 Supervised and unsupervised approaches

Within the field of Machine Learning, classification tasks have always been a major focus of attention. The main goal of this family of algorithms is to train the classifier with the training data and its corresponding labels, and then classify new data that has never been seen by the system. This is known as a supervised approach. In the context of test automation, this type of solution can be used as an assistant to determine whether a test is useful or not [10]. In addition, most of these solutions have the possibility to work with a soft output, which allows working with probabilities and thresholds, something that can be used for prioritisation and for risk estimation of the identified vulnerabilities or non-compliance failures.

On the other hand, in view of the usual lack of labelled data, unsupervised techniques appear as an alternative. Within these techniques, clustering methods stands out. They are based on classifying samples into groups based on their characteristics. This type of approach would only be useful for obtaining groups of tests with similar characteristics/specifications or whose objectives overlap, which can be used to prioritise [10]–[12] and minimise the number of tests that have a higher test coverage [13].

2.2 Reinforcement Learning

The Reinforcement Learning (RL) technique is ideally suited for action prioritisation tasks, which perfectly fits in the context of test automation. This prioritisation can be included in Continuous Integration (CI) processes or failure detection during the execution of functional tests [12]. However, one of the use cases whose characteristics are best suited to RL are pentesting tasks [7]. The main reason is that it allows to effectively perform broader tests covering a wide variety of attack vectors and to consider complex and evasive attack paths that are difficult to identify and investigate by human testers. For this purpose, the reinforcement learning algorithm can be modelled using Markov Decision Processes (MDPs) or Partially Observed Markov Decision Processes (POMDP) [14]. The MDP is a 4-tuple (S, A, P, R) where S is a set of states, A is a set of actions, P is the probability that we transition into a state s' given state, and action s, a and R is the reward function.

Finally, it is worth mentioning that there are two different approaches to automate pentesting by means of RL [8]: graph-based and tree-based. In attack graph models, each node in the graph represents a host. On the other hand, in attack tree models, each node of the tree represents an attack or a target, and the root node represents the final target of the attack. Thus, to attack the root node it is necessary to have previously attacked the child nodes. In view of the above, this approach would allow the generation of attack paths, the prioritisation of tests, or the search for the optimal configuration of the employed tools, although the last one would be computationally very expensive.

2.3 Natural Language Processing

Within Machine Learning field, Natural Language Processing (NLP) techniques emerge to carry out text analysis tasks. Within these tasks, and apart from well-known tasks such as text classification or translation, the following stand out especially because of INFAST project context:

- Text encoding using word embedding and feature extraction to work with structured data and in order to feed text data to other ML models. In addition, this can be useful for other NLP tasks such as text similarity, where the distance between the texts encoded by the model in the latent space is calculated.

- Knowledge Management through ontologies, allowing to store, index, and search many text documents via a knowledge graph [15]. This would allow the user, for example, to query a system to retrieve requirements for any mission, check lessons learned for any component, find anomalies that are common between two missions, etc.

Most current solutions are based on Deep Learning architectures, especially Recurrent Neural Networks (RNNs) and Long Short-Term Memories (LSTMs), which have already been explained in previous sections. However, in recent years a new technique has emerged that has attracted most of the attention in this field: Transformers [16]. Its main objective is to solve the problem of long-range dependency that appears in texts, where it is often very difficult to relate several occurrences of a concept as being the same. Transformers overcome this challenge by using self-attention, which allows the model to use the words in the sequence being processed to obtain a better representation of each word. The architecture of transformers is defined by a stack of encoding and decoding networks.

NLP techniques allow the analysis of requirements documents and associated functional tests, but it will also be possible to generate new tests based on user stories and acceptance criteria and the classification of test cases [17]. Finally, it is worth mentioning that there is increasing progress in the generation of code from natural language, as demonstrated by the recent OpenAI Codex [18] solution, although such solutions are not yet mature enough.

3. Root Cause Analysis for Functional Testing

As described in [19], there are two main ways to detect anomalies in ESOC ground systems: Manually via an operator/tester or automatically via the ART-MMIT framework. The former takes place when a human identifies data, events and/or observations that deviate from the "expected" behaviour (either because requirements are violated, or basic functionality/performance failed). The latter one is related to current testing approaches for ESOC ground systems, including automatic test execution and results reporting. The test failure is defined by a set of predefined checks of outputs, which can include outputs shown in the GUI. At first, any test failure is assumed to be caused by an anomaly. However, the further processing of the results is processed by a human to confirm failed tests and raise the corresponding SPR.

Anomalies are usually formalised in the form of ARs or SPRs (Anomaly/SW Problem Reports), whose purpose is to provide all necessary evidence and supporting material concerning a detected anomaly (including recovery actions undertaken, where applicable) to allow the recipients to analyse the anomaly and, where applicable, to propose a workaround solution or further recovery action. A typical hurdle faced by the maintenance teams when analysing an SPR is the lack of information in the SPR needed to reproduce and investigate the issue. While automatic anomaly detection is far more efficient than manual detection, it still requires manual intervention on the following steps of evaluating, identifying, and solving anomalies. These steps require specialized effort with knowledge on the testing framework context, in addition to specialized effort with knowledge on the application under test.

Another aspect is that the anomaly identification process is dependent on the individual that reports it and analyses it, which may lead to different interpretations. The chance to identify patterns in different anomalies and possibly uncover common causes is therefore reduced. In the end, the process requires considerable overhead and communication between different parties. The categorization of data combined with the use of intelligent algorithms can provide efficiency and higher quality in the management of anomalies. Bearing all the above in mind, the goal is creating a system that supports the tester with the classification and correlation of errors (test failures) based on historical and current data. For this purpose, the process will be divided into three steps:

- Data ingestion. The tool allows to import data, analysing each new file (test reports and SPRs) to extract relevant information. In this process, test failures (i.e., those steps that have been executed with a failed result) are detected.
- Data processing. The relevant fields (texts in this case) are used by ML algorithms to search for past data related to the test failure to be analysed.
- Recommendation of related sources. The tool will provide a list of related data sources. Thus, the system can identify relationships between a new failure and previous ones, also creating relationships between failures and existing SPRs. If a new group of related failures appears, it could be related with a new detected root cause. Otherwise, the system suggests to the creation of a new SPR because it is a new type of failure.

3.1 Data description

To make the prototype as useful and future-friendly as possible, it is understood that it is focused on EGS-CC/EGOS-CC based systems and test reports. For obvious reasons, the content of this data is private and cannot be shared. For this purpose, two types of data sources have been identified:

- **Test Results/Reports**: XML files that contain the results of the execution of each test step and a log extract. These files are produced by the test execution tool EUDART and stored on the machine where the SUT is (and not kept forever). To test for similarity between test reports, .xml files exported from EUDART have been used, for a total amount of 85 files. Each of these files contains the information generated in E2Eval by the execution of these

tests. There are 57 test failures in these executions. From this source, the following fields will be used: identifier, test report name, test step name and identifier, description of the test step, and output log.

- **SPRs**: As in the previous case, SPRs exported from a real system have also been used to carry out the experiments and prototype development. In this case, it has been obtained from ESA's JIRA projects. There are more than 1000 SPRs. This file will contain several fields, being the summary and description the most relevant.

3.2 Proposed approach: NLP to get similarities

As the fields with the most relevant information are texts, the proposed approach is based on the application of NLP techniques to obtain similarities.

3.2.1 Transformers for text encoding

Texts are an unstructured data type. Therefore, it is essential to start by encoding the information in a structured way before applying other algorithms. Feature Learning (or Representation Learning) will be used for this purpose, since it allows to discover the representations needed to perform other tasks such as information retrieval, classification, clustering, or sentence similarity tasks. This technique encodes texts in a latent space by obtaining embeddings. To do this, a pre-trained Transformers-based model is selected: all-mpnet-base-v2*, which is the best model in several benchmarks. This model encodes texts in a latent space of 768 dimensions. However, it should be noted that the model was trained with general purpose data sets, so it is expected that performance may decrease with domain-specific data (i.e., system logs, test descriptions, etc.).

3.2.2 Similarity

Once the texts have been encoded, the use of cosine similarity (S_C) is proposed to score the similarity between the embeddings $\boldsymbol{x}$ and $\boldsymbol{y}$ of two texts, defined as

$$S_C(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|} = \frac{\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{y}_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2} \sqrt{\sum_{i=1}^{n} (y_i)^2}} \quad (1)$$

In the case of the calculation of the similarity between test failures, two text fields will be used: the description of the failed test and the log of the result/error obtained after its execution. On the other hand, to relate test failures to existing SPRs, only the description fields of both sources have been used. Therefore, the similarity between two test failures will be calculated by the Geometric Mean (GM) of the cosine similarities obtained independently for their descriptions and their logs, calculated as

$$Sim(\mathcal{A}, \mathcal{B}) = GM\left(S_{C_{desc}}, S_{C_{log}}\right) = \sqrt{S_C(\boldsymbol{a}_{desc}, \boldsymbol{b}_{desc}) \cdot S_C(\boldsymbol{a}_{log}, \boldsymbol{b}_{log})} \quad (2)$$

where $\mathcal{A}$ and $\mathcal{B}$ are the test failures under study; $\boldsymbol{a}_{desc}$, $\boldsymbol{b}_{desc}$ are the embeddings of the test descriptions; and $\boldsymbol{a}_{log}, \boldsymbol{b}_{log}$ are the embeddings of the execution logs. The Geometric Mean has been used, but it is also possible to use weighted versions of it to give more emphasis to one of the fields. Thus, the similarity between sources is obtained with a score between 0 and 1, where the higher the value, the greater the similarity. Fig. 1 shows the similarity matrix between all the test failures used in the experiments.

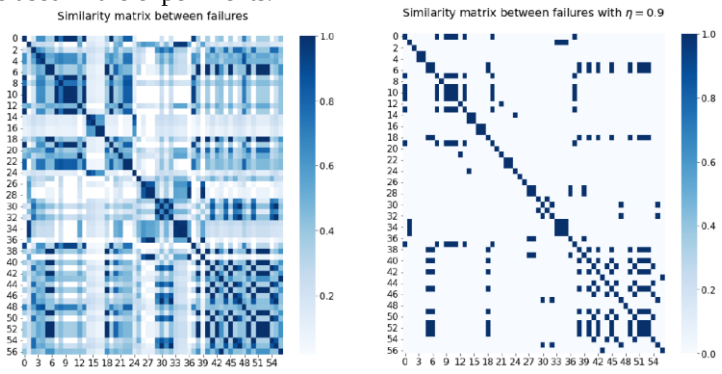

Fig. 1. Similarity matrix before (left) and after (right) applying a threshold η=0.9.

*__MPNet__ family combines strengths of masked and permuted language modelling for language understanding. Source: https://huggingface.co/sentence-transformers/all-mpnet-base-v2. Benchmarks: https://www.sbert.net/_static/html/models_en_sentence_embeddings.html

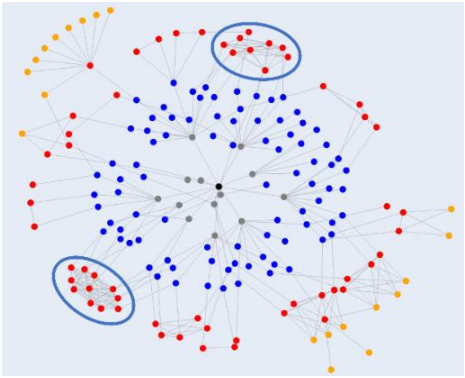

Fig. 2. Similarity graph obtained from employed data: Black node represents the type of project/mission; grey nodes represent different test projects; blue nodes are the executed test reports; red nodes represent the existing test failures; and orange nodes, the SPRs. There are two identified clusters of test failures, marked inside blue circles.

At this point, a score is obtained, but it is still necessary to establish when the system will consider providing the recommendation to the user. For this purpose, a threshold η will be applied. The objective is to recommend only those cases whose score is higher than this threshold. Fig. 1 shows the result of the similarity matrix after applying the threshold $\eta = 0.9$.

3.3 Results

After applying the process described above to all the available data, the similarity graph shown in Fig. *2* is obtained. This graph shows the relationships between test failures belonging to different test projects that evaluate the same system or mission. Two groups of failures are very interconnected, which leads to propose both clusters as a possible root cause of problems in that system. However, the search for SPRs related to the failures does not achieve the initial expectations, since very few relationships between them are obtained. One of the reasons identified is the difference in language between the two data sources. The test report texts are automatically generated by the EUDART system, while the SPR texts are manually written by the operators. Language written by a human will inevitably contain jargon that can hardly be related to automatic texts.

Finally, two methods commonly used for this purpose were tested to determine the performance of the chosen approach: **TF-IDF** (Term Frequency-Inverse Document Frequency) score: It is a technique commonly used in NLP and information retrieval to quantify the importance of a word in a document. The algorithm calculates the term frequency (TF) and inverse document frequency (IDF) values for each word in a collection of documents. The TF value represents the number of times a word appears in a document, while the IDF value represents how many documents the word appears in out of the total number of documents in the collection. The product of these two values gives the TF-IDF score for a given word in a specific document. These scores have been used as features to obtain the cosine similarity in the same way as in the previous case. **BM-25** (Best Match 25): It is a ranking function used in information retrieval to score the relevance of documents based on their content. It is similar in concept to the TF-IDF algorithm, but it is a more complex and sophisticated algorithm that is designed to handle large collections of documents. However, in BM25, the IDF term is calculated differently. It uses a logarithmic scale and normalization to consider the length of the documents and the average length of documents in the collection. Again, these scores have been used as features to obtain the cosine similarity after applying a normalization between 0 and 1.

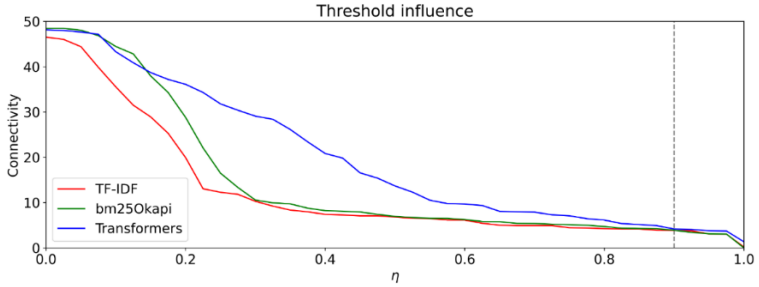

Fig. 3. Connectivity comparison between TF-IDF, BM-25 and all-mpnet-base-v2 (transformers) dependent on the threshold chosen. The vertical line is the final threshold used by the system to classify two pairs as similar.

On the other hand, Fig. 3 shows the comparison of the described algorithms. The figure shows the influence of threshold η on the connectivity between cases. The connectivity between cases is defined as the average number of test failures related to each one of them. As can be seen, the method based on Transformers is able to obtain a greater number of relations between texts.

4. Action Selection for Automated Pentesting

There is plenty of literature [20] and multiple commercial tools that make use of AI algorithms to automate the selection of the most suitable attack set during a pentesting process. For this task, Reinforcement Learning [4], [6], [7], [9] algorithms are widely used to produce a model that imitates the experience and the behaviour of a pentester [8]. In addition, AI driven identification of the most promising attack paths would concur into time savings and more efficiency [21]. Reinforcement learning does not need as much feedback as supervised learning, but it comes with the downside of being very costly in terms of time and resources spent during training.

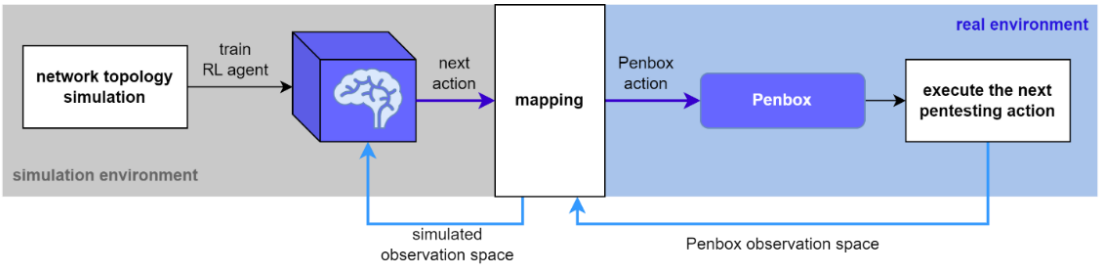

Fig. 4. Design of the solution: Penbox executes in a real environment the actions selected by the RL agent (trained in a simulation environment).

ESOC had already developed a proof of concept for penetration test automation: Penbox. It is an application that contains several pentesting tools with the objective of executing them following a designed attack scenario. This not only requires to be able to execute those tools in the correct order, but also requires that each tool is able to retrieve input parameters from results of tools that preceded in the execution chain. In this context, the INFAST project wants to automate the selection of the action to be executed by Penbox based on the information (observation space) obtained after each execution. In other words, it will no longer be necessary to create an attack scenario, but the attack plan will be created during its execution.

Nevertheless, training an AI algorithm in real-world environments can be costly and challenging. To overcome this, a simulated environment was used to pre-train the agent's policy [22]. The benefits of simulation include: the ability to model relevant system aspects at a higher level of abstraction, such as application-level network communication rather than packet-level simulation, and the ability to ignore low-level details if they are not necessary. Additionally, simulation allows for easy definition of new machine sensors, restricts the action space to a manageable and relevant subset, captures the global state efficiently for easier debugging and diagnosis, and has a lightweight runtime footprint that can be run on a single machine or process. The most important simulation frameworks for cybersecurity are CyberBattleSim[†] and NASim[‡]. For this study, CyberBattleSim was chosen because it admits a higher level of detail and flexibility. Unfortunately, there is no one-to-one mapping between actions from the simulation and PenBox tool, nonetheless, this approach is envisioned as a knowledge transfer task, as both domains are closely related. The following section describes the process followed for topology simulation.

4.1 Generation of simulated environments

The agent's goal is to learn how to exploit the vulnerabilities in any environment (generalization and adaptability), and to learn as fast as possible. Therefore, multiple simulated network topologies should be used to avoid overfitting during training, and testing will be performed on different scenarios. CyberBattleSim does not support random environment generation, thus, an environment generation procedure was developed to obtain realistic scenarios. A combination of public and (GMV's) internal data was used to model the environments. Throughout the following sections, the different steps to generate the network topologies are described in detail.

[†] **CyberBattleSim**: Simulator developed by Microsoft. Source: https://github.com/microsoft/CyberBattleSim
[‡] **NASim**: Python library developed by Schwartz et al. [22] Source: https://github.com/Jjschwartz/NetworkAttackSimulator/

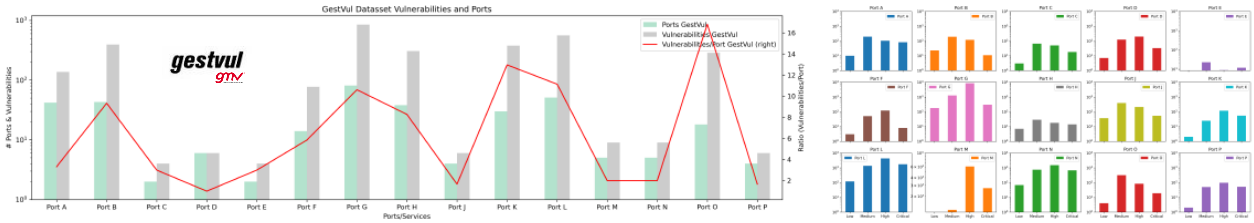

Fig. 5. On the left, GestVul database number of vulnerabilities clustered in different services. The grey columns represent the number of vulnerabilities, the blue ones the number of vulnerable ports, and the red line represent the ratio between vulnerabilities per service. On the right, criticality distribution among different services/ports labelled as low, medium, high, and critical extracted from Nessus plugins.

4.1.1 Global Identifiers Generation

A set of vulnerabilities, properties and ports are defined. Such sets state a common vocabulary shared across multiple environments, thus ensuring a consistent numbering convention that a machine learning model can use to learn from. A joint effort between GMV and ESA cybersecurity experts made possible an accurate representation of a real representation of network properties:

- A set of protocols and services. For the experimentation, FTP, SSH, Telnet, SMTP, DNS, HTTP, SMB, MySQL, and PostgreSQL were selected. However, it is configurable and easy to be modified to consider other protocols.
- Different actions related to the aforementioned included protocols. These actions are those which can be executed in the PenBox tool, including scanning, and exploiting actions for each protocol.
- Properties or potential states that define the observation space regarding the open ports and discovered services.

4.1.2 Vulnerabilities Definition

A vulnerability generator was created using a combination of random variables modelled by real data. Some of the estimated variables were the existence probability, criticality, type, outcome, precondition, rates, and cost. Throughout the following sections, a detailed analysis of the use of that information to model the environment is provided. The **vulnerability existence probability** is approximated by Bernoulli random variable $X_i \sim \text{Ber}(p_i) \quad i = 1, \dots, n_v \; i.i.d.$ where n_v is the number of vulnerabilities, and $\boldsymbol{p} = (p_i)_{i=1}^{n_v}$ is estimated using $\boldsymbol{v} = (v_i)_{i=1}^{n_v} \subset \mathbb{R}_{\geq 0}$, which is the vulnerabilities per service ratio (extracted from GestVul[§]). Let $T: \mathbb{R}^{n_v} \to \mathbb{R}^{n_v}$ be an application such that $\boldsymbol{v} \mapsto \boldsymbol{p}$. T should satisfy the following constrains:

- Maintain relative proportions (ratio): $\boldsymbol{v}_i/\boldsymbol{v}_j = T_i(\boldsymbol{v})/T_j(\boldsymbol{v})\big(= \boldsymbol{p}_i/\boldsymbol{p}_j\big) \quad \forall i \neq j$
- $\sum_{i=1}^{n_v} \boldsymbol{p}_i = m$ where $m \in \mathbb{R}$.

Within this constrains, it can be proven that there exists only one application $T: \boldsymbol{v} \mapsto \boldsymbol{p}$ which is given by $\boldsymbol{p}_i \coloneqq \boldsymbol{v}_i m / \sum_{j=1}^{n_v} \boldsymbol{v}_j$. Furthermore, m can be used to approximate the number of vulnerabilities in each node verifying that $\Pr\{\left|m - \sum_{i=1}^{n_v} X_i\right| < \sqrt{n_v}\} > 0.95$.

4.1.3 Criticality

The criticality of each vulnerability is an additional reward implemented for each vulnerability. To assess the criticality to be associated with a port/protocol and looking for a value taken from a real source, it has been considered if it is possible to exploit this service and that there is the possibility of assessing if it is vulnerable and exploitable. For this reason, the plugins database of the Nessus[**] vulnerability scanning tool has been considered, being able to know for each protocol how many plugins there are that allow vulnerabilities to be detected, and what criticality these vulnerabilities would have. The criticality defined for the results obtained by the Nessus plugins is based on the National Vulnerability Database (NVD)[††], which contains a list of vulnerabilities, including their CVE and their impact, classified by product, impact, attack vector, etc. According to all the above, relative frequencies are calculated for each protocol, based on the number of possible Nessus detections (according to their criticality) for that protocol. In case of the existence of a vulnerability in the created node, the criticality is assigned according to the aforementioned

[§] **GestVul** is a private database owned by GMV containing several thousands of discovered vulnerabilities. Source: https://www.gmv.com/en-es/products/cybersecurity/gestvul

[**] **Nessus** is a vulnerability scanner used to identify potential security weaknesses in their systems and network infrastructure. Source: https://www.tenable.com/products/nessus

[††] **The National Vulnerability Database (NVD)** is a U.S. government repository of information about known cybersecurity vulnerabilities. Source: https://nvd.nist.gov/

probability. Thus, for a node i with a vulnerability of type v, the criticality $C_{i,v}$ of the vulnerability is assigned based on distributions (estimated via relative frequencies):

$$Pr\{C_{i,v} = \mathcal{C}\} \coloneqq \frac{N_{V,\mathcal{C}}}{N_V}, \quad \mathcal{C} \in \{\text{Low}, \text{Medium}, \text{High}, \text{Critical}\} \tag{3}$$

where N_V is the total number of vulnerabilities registered in the database.

4.1.4 Generate Random Network Traffic

Shodan[‡‡] data is used to model the generation of network traffic for each service/port. It keeps an up-to-date record of public ports. The ratio between ports/services and discovered IPs over the internet is defined by s_k where $k = 1, \dots, K$. The generation is divided into two steps: in the first step, a random subnetwork topology is created; and, in the second step, each subnetwork is populated with public and owned data.

4.1.4.1 Subnetwork generation

A subnetwork topology $G = (V, E)$ is created using the Erdős-Rényi binomial random graph model. The existence of each edge is given by a *i.i.d.* Bernoulli random variable with parameter p_{ij}, where $p_{ij} = Pr\{e_{ij} \in E\}$ is the subnetwork interconnectivity parameter. The number of subnetworks $|V|$ and p_{ij} is assigned using the following rule:

$$p_{ij} = \begin{cases} 0.3 & if \;\; i \neq j \\ 1.0 & if \;\; i = j \end{cases} \tag{4}$$

Note that the interconnectivity parameter in the case of the same subnetwork is set to 1.0, which means non-restriction traffic between same subnetworks nodes. Fig. 6 shows an example of a subnetwork of size 10 generated using this procedure.

4.1.4.2 Network generation

Using the generated subnetwork topology, each subnetwork is populated with nodes. In addition, a random variable (*r.v.*) generates the traffic between network nodes (See Fig. 6). Let T_n be the maximum number of nodes over the network. Therefore, the Beta-Binomial *r.v.* (bounded) is used to sample the number of nodes in each clustered named $(S_i)_{i=1}^{|V|}$:

$$S_i \sim \text{BetaBin}\left(n = \frac{T_n}{|V|}, \alpha = 2, \beta = 5\right) \qquad i = 1, \dots, |V| \tag{5}$$

The use of the Poisson distribution (although it is unbounded) was also considered because of its simplicity and the additivity property, which is convenient in this case: $\sum_{i=1}^{|V|} \mathcal{P}(\lambda_i) = \mathcal{P}\left(\sum_{i=1}^{|V|} \lambda_i\right)$. Once the network topology is set, the traffic for each service can be defined. The interconnectivity probability between node $v_i, v_j \in V$ using service $k \in \{1, \dots, K\}$ is distributed as $e_{ij}^k \sim \text{Bernoulli}\left(P_{ij}^k\right)$, where P_{ij}^k is another random variable given by:

$$P_{ij}^k = \begin{cases} \text{Beta}\left(\alpha = 1, \beta = \dfrac{1}{s_k}\right) & if \; v_i \; and \; v_j \; belong \; to \; different \; subnets \\ \text{Beta}\left(\alpha = 5, \beta = \dfrac{1}{s_k}\right) & otherwise \end{cases} \tag{6}$$

Given that $s_k < 0.01$ for $k = 1, \dots, K$, then $\alpha \ll \beta$, thus, the mean of a Beta distribution $\frac{\alpha}{\alpha+\beta} \approx \frac{\alpha}{\beta}$. And therefore, the mean is approximated as follows:

$$\mathbb{E}\left[e_{ij}^k\right] \approx \begin{cases} s_k & if \; v_i \; and \; v_j \; belong \; to \; different \; subnets \\ 5s_k & otherwise \end{cases} \tag{7}$$

This result indicates that the random variable has an expected value proportional to the ratio of the data extracted from Shodan. And connections between nodes on the same subnetwork are 5 times more likely than between nodes in different subnets, as would be expected in a real-world scenario. This value could be minimized to downplay the role that each subnetwork plays in the environment.

4.1.5 Nodes properties definition

CyberBattleSim offers a high degree of customisation of node characteristics, including the following aspects:

[‡‡] **Shodan** is a search engine for internet-connected devices, it allows users to search for specific types of devices connected to the internet using various filters and keywords. Source: https://www.shodan.io/

Fig. 6. On the left, the result of generating a subnetwork topology based on Erdős-Rényi binomial random graph model. On the right, the resulting network topology.

1) List of node **vulnerabilities**. They are randomly selected (random distribution estimated from GestVul vulnerabilities data) from the list of vulnerabilities previously created (Section 4.1.2).

2) Intrinsic node **value.** It describes the importance of the node. A priori is set using a random variable $\mathcal{U}\{0, 100\}$. In addition, this value is used as a reward if the node is compromised.

3) **Properties** of the nodes, some of which can imply further vulnerabilities. They are randomly selected from the chosen vulnerabilities and from the library property list defined in Section 4.1.1.

4) List of **ports and protocols** the node is listening to. It is given by the traffic generated in Section 4.1.4.

5) **Firewall** configuration of the node, which is modelled using the active services obtained from the network traffic and a random variable.

6) **Privilege level** e.g., Admin, User, etc. The initial state of the network is set every node to unknown (non-access) and the initial centre-node as normal user privilege.

7) **Machine Status**: running or stopped. For this purpose, every machine is set to running.

8) **Stopping weight** is the relative node weight used to calculate the cost of stopping this machine or its services (constant value). Many of the aspects of the nodes were already set during the previous steps, and the ones that could not be randomized were selected using the cybersecurity expert knowledge.

4.2 Proposed approach: RL to select the next action

Reinforcement Learning (RL) is considered one of the machine learning techniques closest to how humans learn. It is a paradigm that intends to train machines to maximize the cumulative reward. This reward is given to them for performing certain actions that may influence their environment towards a certain goal. In addition, the ever-increasing computational power available has encouraged both its research and use. Its operation is based on the interaction between the different elements that make up the solution:

- **Agent**: The entity that is going to be trained. It can perform a defined set of actions.
- **Reward**: Signal sent to the agent that measures how good an action taken by the agent was.
- **Policy**: Strategy that the agent follows to map situations to actions, given by the transition probability to change to another state (S').
- **Environment**: Model of the world where the agent lives in. It evolves (changes its state S) according to the inputs from the agent and computes the appropriate rewards.

With the aim of finding the best decision-making policy for the agent, this problem is defined as a Markov Decision Process (MDP). The training structure is quite simple: an agent observes the environment and takes an action following a policy. When the current state of the environment is uncertain, a Partially Observable Markov Decision Process (POMDP) is employed instead. The training process involves the agent observing the environment, taking an action based on a current policy, and receiving a reward based on the effectiveness of the action towards a defined goal.

In addition, different RL algorithms (such as DQN, A2C and PPO) and different network architectures (such as MLP, GNN and recurrent networks) have been run during the project for comparison (as detailed in Section 4.2.2).

4.2.1 Environment

Once the network topology is created, an interface between the algorithm and the simulation is necessary. The environment is the system in which an agent interacts and learns using a feedback loop. The environment is also responsible for providing the agent with the necessary information to make informed decisions, such as the current state of the system. The agent learns by exploring the environment and updating its understanding of the relationship between its actions and the resulting rewards through trial and error. The agent-environment interaction loop has three main components: Observation space, action space and reward model.

Observation Space: The state/observation S_t is a representation of the current agent's perspective at each time-step t. For example, in this case, it is the set of current properties of the network to which the agent has access. The state is the equivalent of the information provided by the established receptors; therefore, it is not a complete representation of the environment. The observation space can be composed of discrete or continuous variables. In fact, the environment's observation space used is a mix of both. The environment state can be referred as the observation due to the reduced access that the agent has to the real state. The problem is formulated in terms of a POMDP. The observation space has a graph data structure. Each node represents a node in the network, and the edges symbolise the traffic between nodes, including the following information:

1) ***Node Properties***: Feature vector of properties for all the discovered nodes. Each element has three different values: 0 means the property is not set for that node, 1 means the property is set for that node, 2 means the property status is unknown.

2) ***Node Privilege Level***: Access privilege level on a given node. There are four possible values: 0 means no access, 1 means local user, 2 means admin user, and 3 means system privilege.

3) ***Edge Traffic***: Annotation added to the network edges created when the simulation is played. There are three different values: 0 means that the source node knows the existence of the target node, 1 means target node has been exploited by the source node, and 2 means the source node has been able to carry out a lateral move to the target node.

4) ***Global Credential Matrix Cache***: Credentials are stored in a matrix, which can be access by the agent to gain access/connect to nodes.

5) ***Global Historical Data***: Previous data is essential for the model to imitate human interaction with the machine. E.g., once the agent has exploited a vulnerability successfully, it no longer makes sense to try the same action again. For that, three different values are provided: the number of tries the algorithm has tried to exploit the vulnerability, historical ratio of failed actions needed to exploit the vulnerability, and whether the action has been previously successful.

Action Space: A design principle adopted for the simulation is to model just enough complexity to represent attack techniques from the MITRE[§§] matrix, while maintaining the simplicity required to efficiently train an agent using Reinforcement Learning techniques.

Let n_c be the number of nodes in the graph and n_e be the number of edges in the current state. The action space contains three types of attacks:

1) Local actions. The set of actions that can be executed/exploited in the same machine. $\boldsymbol{L}(n_c) \coloneqq [n_c] \times [v_l]$ where and v_l is the number of local vulnerabilities.

2) Remote actions. A set of actions whose execution involves two nodes: source and target. $\boldsymbol{R}(n_c) \coloneqq [n_c] \times [n_c] \times [v_r]$, where v_r is the number of remote vulnerabilities.

3) Connection. It is the ability to gain access/connect to a node using credentials. $\boldsymbol{C}(n_c) \coloneqq [n_c] \times [n_c] \times [n_{ports}] \times [n_{cred}]$, where n_{ports} is the number of ports/services and n_{cred} is the number of credentials.

Reward Model is a function that maps the state-action pairs of an agent to a scalar value, called the reward signal. It is used to guide the agent's learning process, by providing it with information about the quality of its actions. The reward provided as feedback by the environment is an important part of the learning process, and tuning it correctly is essential. There were previously discussed some rewards such as the criticality (Section 4.1.3), and the node value (Section 4.1.5). In addition, the cybersecurity team has fine-tuned numerous specific rewards, such as 5,000.0 for a solved environment, 5.0 for discovering an unknown node, and -1.0 for a repeated action, to name a few examples.

4.2.2 RL policy

The RL policy is a function that maps states of an environment to actions. Up to now, literature have previously shown that standard MLPs were able to achieve promising results literature [6], [8], [22]–[24] . For this study, the MLP architecture has been implemented and compared to a graph-based one. In addition, attention mechanisms and recurrent layers have been researched. Multiple metrics shows that GNN surpasses conventional MLP in performance on average. In the following sections the steps taken are explained in more detail.

Multi-Layer Perceptron (Policy) has been extensively studied in the literature. However, it has never been deployed on a simulated environment which replicates with a high level of detail compared to real ones. The MLP

[§§] **MITRE ATT&CK®** is a globally accessible knowledge base of adversary tactics and techniques based on real-world observations. Source: https://attack.mitre.org/

policy is fed with a fixed size 1-dimension feature vector. The flattened adjacency matrix is used to represent the graph [8]. It is worth mentioning that the matrix is sparse, and the representation of a graph is not unique. In the top layer of the MLP Policy we introduce the past historical data concatenated with the logits predicted by the policy, as can be seen in (12) and (13). The output size is described in Section 4.2.1. The MLP architecture hyperparameters (non-linearity, hidden layers, nodes per layer, etc) are chosen by the hyperparameter optimizer.

The observation space (non-euclidean) domain Ω is endowed with certain geometric structure and symmetries, so-called *geometric priors* [25]. From now on, we will focus on modern deep learning architectures that exploits the permutation invariance (and equivariance) property that emerges from our cybersecurity domain. The most known architectures with the permutation symmetry group are the Graph Neural Networks (graphs), Deep Sets (sets) and Transformers (complete graph). The function composition preserves the properties. As a result, GNNs were proposed as a replacement for MLP in the policy.

Graph Neural Network (Policy) is a type of neural network that is designed to process data represented as a graph. The GNN-layers used in the policy were Graph Attention Layer (GATConv) [26] and Graph Convolutional Networks (GCNs) [27]. The node and edge features are passed through a Graph Attention Transformer layer, the attention weights and node features are fed into the next layer ((8) where the attention coefficients α_{ij} are computed in (9).

$$\boldsymbol{x}_i' = \alpha_{ii}\boldsymbol{\Theta}\,\boldsymbol{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{ij}\,\boldsymbol{\Theta}\,\boldsymbol{x}_j \qquad (8)$$

$$\alpha_{ij} = \frac{exp(\text{LeakyReLU}\left(\boldsymbol{a}^\top[\boldsymbol{\Theta}\boldsymbol{x_i} \,\|\, \boldsymbol{\Theta}\boldsymbol{x}_j]\right))}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp(\text{LeakyReLU}(\boldsymbol{a}^\top[\boldsymbol{\Theta}\boldsymbol{x_i} \,\|\, \boldsymbol{\Theta}\boldsymbol{x}_k]))} \qquad (9)$$

The next layer is divided into two independent blocks: the node classifier and the edge classifier. Both tasks are relatively well known in the geometric deep learning literature. Graph Convolution Network layers were used for the Node Classifier.

$$\boldsymbol{x}_i' = \boldsymbol{\Theta}^\top \sum_{j\in\mathcal{N}(v)\cup\{i\}} \frac{e_{j,i}}{\sqrt{\widehat{d}_j\widehat{d}_\iota}}\boldsymbol{x}_j \qquad (10)$$

The output is feed into an MLP to predict a classification of each node into possible local actions for each node: $(\boldsymbol{o}_{local})_i = \text{MLP}_{local}(\boldsymbol{x}_i')$, where $\text{MLP}_{local}: \mathbb{R}^{f_n} \to \mathbb{R}^{v_l}$ and f_n is the number of node features. Notice, $\boldsymbol{o}_{local} \in \mathcal{M}_{n_c\times v_l}(\mathbb{R}) \cong \mathbb{R}^{|\boldsymbol{L}(n_c)|}$.

Each local action is represented as a tuple of the node where the action is performed, and the vulnerability or action taken place in the node. Therefore, it makes sense to transform each node into a classification problem in which we try to select the action to perform in the node. The logits obtained are flattened and passed to the next layer. The Edge Classifier selects the remote and connect actions which needs a source and target node (edge) to perform the task, therefore, we interpret the problem as an edge classification problem in which we try to classify each edge into the action to perform (similar to the node classifier). For the remote vulnerability we classify in the number of remote actions and in the connect we classify in the actions space of $[n_{ports}] \times [n_{cred}]$.

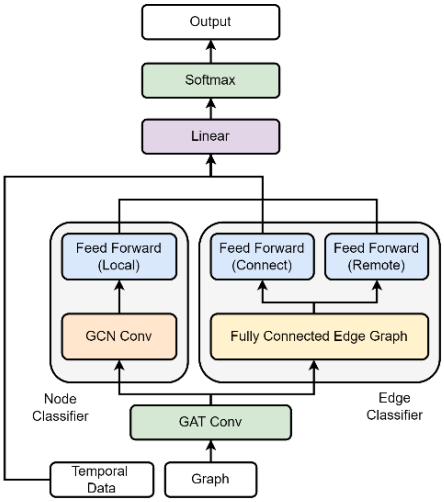

Fig. 7. GNN policy architecture diagram.

The node features extracted from the GATConv layer are represented as edges which we feedforward:

$$\boldsymbol{x}'_{ij} \coloneqq \alpha_{ij}(\boldsymbol{x}_i \| \boldsymbol{x}_i - \boldsymbol{x}_j) \in \mathbb{R}^{2n_c} \qquad \forall i,j = 1,\dots,n_c \tag{11}$$

Then, the result is fed into an MLP to classify each edge: $(\boldsymbol{o}_{remote})_{ij} = \text{MLP}_{remote}(\boldsymbol{x}'_{ij})$, where $\text{MLP}_{remote}: \mathbb{R}^{f_e} \to \mathbb{R}^{v_r}$ and $(\boldsymbol{o}_{connect})_{ij} = \text{MLP}_{connect}(\boldsymbol{x}'_{ij})$ where $\text{MLP}_{connect}: \mathbb{R}^{f_e} \to \mathbb{R}^{v_c}$ and f_e is the number features in the edge. The output shapes are $\boldsymbol{o}_{local} \in \mathcal{M}_{n_c^2 \times v_r}(R) \cong \mathbb{R}^{|\boldsymbol{R}(n_c)|}$ and $\boldsymbol{o}_{connect} \in \mathcal{M}_{n_c^2 \times v_c}(\mathbb{R}) \cong \mathbb{R}^{|\boldsymbol{C}(n_c)|}$. The flattened output logits from each type of action are concatenated: $\mathbf{o} = \boldsymbol{o}_{local} \| \boldsymbol{o}_{remote} \| \mathbf{o}_{connect} \in \mathbb{R}^K$ where $K(n_c) \coloneqq |\boldsymbol{L}(n_c)| + |\boldsymbol{R}(n_c)| + |\boldsymbol{C}(n_c)|$. The logits of each feed forward are concatenated temporal data $(\boldsymbol{H}_i)_{i=1}^{K(n_c)} \subset \mathbb{R}^{f_h}$ of each action where f_h is the number of feature that the temporal data store and it is applied a linear layer (12), where $w_0, b \in \mathbb{R}$ and $\boldsymbol{w} \in \mathbb{R}^l$ parameters to be optimized. Then is applied the softmax activation function to predict the action in (13).

$$\boldsymbol{z} \coloneqq \text{Linear}(\boldsymbol{o}|\boldsymbol{H}) = \boldsymbol{o}w_0 + \boldsymbol{H}\boldsymbol{w}^\top + b \tag{12}$$

$$\sigma(\boldsymbol{z}) \coloneqq \text{Softmax}(\boldsymbol{z}) = \left(\frac{e^{\boldsymbol{z}_i}}{\sum_{j=1}^{K} e^{\boldsymbol{z}_j}}\right)_{i=1}^{K(n_c)} \tag{13}$$

The final logits $\boldsymbol{z} \in \mathbb{R}^{K(n_c)}$ are used to select the action using a probability distribution. categorial distribution was used for PPO and A2C whereas for DQN there was used the argmax because DQN is intrinsically stochastic due to the epsilon parameter used for exploration.

Recurrent (Policy) was proposed because the temporal axis is a crucial property (in cybersecurity) inherent to most computer systems. It was observed that non-recurrent algorithms do not fully grasp the thought process of a pentesting expert, in fact, it is limited in terms of adaptability and adaptability. To address this restriction, two approaches were explored: using a recurrent policy or incorporating historical data into the agent's observation. The second approach was ultimately chosen because the recurrent policy adds noise to an already stochastic environment, which can slow down the training process and prevent convergence.

4.2.3 Training & Hyperparameter selection

We employed the Tree-structured Parzen Estimator (TPESampler) algorithm to optimize a variety of agents with varying hyperparameters. These hyperparameters included the learning rate, epsilon value for epsilon-greedy algorithms, architecture (MLP vs GNN), number of hidden layers, and loss function. Through hundreds of experiments, we determined that the epsilon value, number of hidden channels, and learning rate had the greatest impact on the performance of the agents in the given environment.

Once the hyperparameters have been selected, the agents have been trained using a A100 GPU and i7700K CPU. The DQN algorithm was finally used for training, as A2C and PPO offers very unstable training and does not achieve convergence. A total of eight simulated environments have been used: 4 for training and 4 for testing. Each of these environments contains 15 nodes (5 servers and 10 machines). The training process consisted of 600 epochs, with 2,000 iterations per epoch, and a maximum of 50 iterations per episode. The batch size was set to 16 and the replay buffer was set to 20,000. The Adam optimizer was chosen for training with the following learning rates: 1.25e-3 (GNN) and 2.65e-3 (MLP). The number of hidden channels was 10 (GNN) and [50,50] (MLP). The exploration rate, ϵ, was a linear decay scheduler (initial: 1.0, final: 0.1, decay: 1e4 steps). We found that GNN architecture performed better than standard MLP. It achieved the highest accuracy in the training environments with same number of iterations. Additionally, GNN showed better performance in the testing environment and was able to generalize to environments that were different from the ones used in training.

4.3 Results

The agent successfully learned to achieve high rewards using the limited information available in the simulated environment. As training progressed, it continually improved its performance in both the training and testing sets. This was made possible by the agent's ability to identify and leverage the underlying patterns present in the distribution of environments, enabling it to make accurate predictions about the optimal action to take. Despite the high variability between different episodes of the environment, the agent was able to adapt to the highly stochastic nature of the simulation.

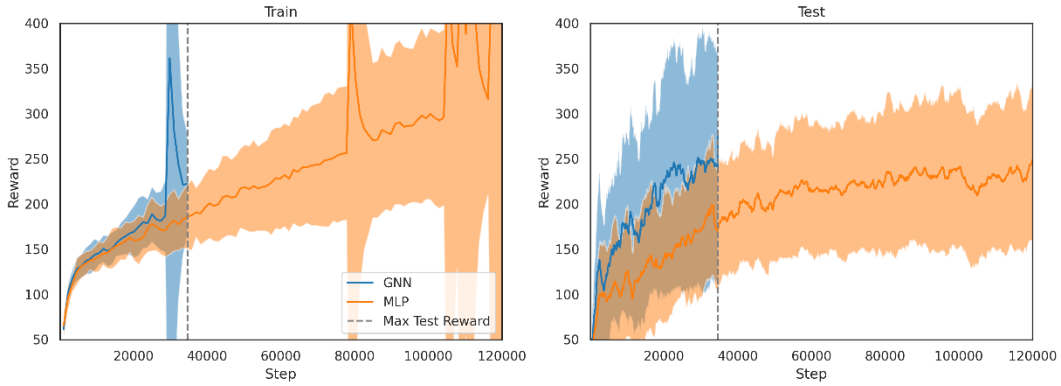

Fig. 8. Comparison of training and testing performance between a GNN and an MLP architecture for an DQN-RL agent highlights a clear difference in their performance on the testing set, despite similar performance on the training set.

Finally, once the agent has been trained, it has been integrated into the Penbox tool. To do this, it was necessary to adapt both the tool's space of possible actions and the agent's observation space. To validate its correct functioning, it has been executed in a virtual environment with known vulnerabilities and the agent is able to select the next actions to be executed, obtaining information on the vulnerabilities. However, the validation phase in the real environment is still in progress and will be presented in future publications.

5. Explainable AI

Explainable AI (XAI) is the branch of Artificial Intelligence that focuses on the machine learning interpretability. In order to ensure transparency, trustworthiness, informativeness, confidence, fairness, and causality, explainability is defined as the ability to interpret, understand, and explain the technical processes of an AI system and the associated human decisions, in this case by technicians and testers. XAI has been identified in recent times as an utmost need for the adoption of ML methods in real-life applications [28]. The implemented models belong to the black-box category. Among explainable algorithms for black-box models, it was chosen the Integrated Gradients [29] technique, which is based on saliency maps and was theoretically proven to meet the sensitivity and implementation invariance axioms.

$$\mathtt{IG}_i(\boldsymbol{x};\mathtt{F}) \coloneqq (\boldsymbol{x}_i - \boldsymbol{x}'_i)\int_{\alpha=0}^{1}\frac{\partial \mathtt{F}\left(\boldsymbol{x}' + \alpha(\boldsymbol{x}-\boldsymbol{x}'_i)\right)}{\partial \boldsymbol{x}_i}d\alpha \quad (14) \qquad \mathtt{IG}_i^{approx}(\boldsymbol{x};\mathtt{F}) \coloneqq \frac{1}{m}(\boldsymbol{x}_i - \boldsymbol{x}'_i)\sum_{k=1}^{m}\frac{\partial \mathtt{F}\left(\boldsymbol{x}' + \frac{k}{m}(\boldsymbol{x}-\boldsymbol{x}')\right)}{\partial \boldsymbol{x}_i} \quad (15)$$

where $\mathtt{F}\colon \mathbb{R}^n \to [0,1]$ is the deep learning model, i is the feature that is explained, and $\boldsymbol{x}'$ is a baseline relative to $\boldsymbol{x}$. In (14) the input attribution is calculated in a theoretical standpoint, although in practice, (15) is an approximation used instead of the former. Authors pointed out that the approximation solution requires the number of iterations m between 20 and 300 steps to converge. The gradient can be tracked using *tf.gradients*, *torch.autograd* or *jax.grad* (notice that F is required to be differentiable). The following shows how this technique can be applied to the design of the pentesting use case presented in the previous section.

The feature attribution is calculated using the Integrated Gradients algorithms with the GNN policy. Let $S_t = (\boldsymbol{X}, \boldsymbol{E}, \mathcal{N}, \boldsymbol{H}) \in \boldsymbol{S}$ be a network state where $\boldsymbol{X} \in \mathcal{M}_{n_c \times f_n}(\mathbb{R})$ is the node feature matrix, $\boldsymbol{E} \in \mathcal{M}_{n_e \times f_e}(\mathbb{R})$ is the edge feature matrix, $\mathcal{N} \subseteq \{1,\dots,n_c\}^2$ is the edge index matrix (equivalence relation indicating graph connectivity), $\boldsymbol{H} \in \mathcal{M}_{K(n_c)\times f_h}(\mathbb{R})$ matrix storing temporal data. Given a trained $\mathtt{GNN}\colon \boldsymbol{S} \to \mathbb{R}^{K(n_c)}$, which maps the input graph data (network state) to the logits of possible actions; and a $\mathtt{Distribution}\colon \mathbb{R}^{K(n_c)} \to \{1,\dots,K(n_c)\}$, which fits the logits to a custom distribution function and output a sample. Let F be a differentiable function compatible with the Integrated Gradient algorithm:

$$\begin{aligned} \mathtt{F}: \boldsymbol{G} = \left(\mathcal{M}_{n_c\times f_n}(\mathbb{R})\middle|\mathcal{M}_{n_e\times f_e}(\mathbb{R})\middle|\mathcal{M}_{K(n_c)\times f_h}(\mathbb{R})\right) \cong \mathbb{R}^{n_c f_n + n_e f_e + K(n_c) f_h} &\to [0,1] \\ (\boldsymbol{X},\boldsymbol{E},\boldsymbol{H}) &\mapsto \mathtt{Distribution} \circ \mathtt{GNN}(X,E,\mathcal{N},H) \end{aligned} \quad (16)$$

The Integrated gradient algorithm is applied using F and the concatenation of $\boldsymbol{X}, \boldsymbol{E}$ and $\boldsymbol{H}$:

$$(\boldsymbol{X}', \boldsymbol{E}', \boldsymbol{H}') = \mathrm{IG}^{approx}(\boldsymbol{X}\|\boldsymbol{E}\|\boldsymbol{H};\mathtt{F}) \quad (17)$$

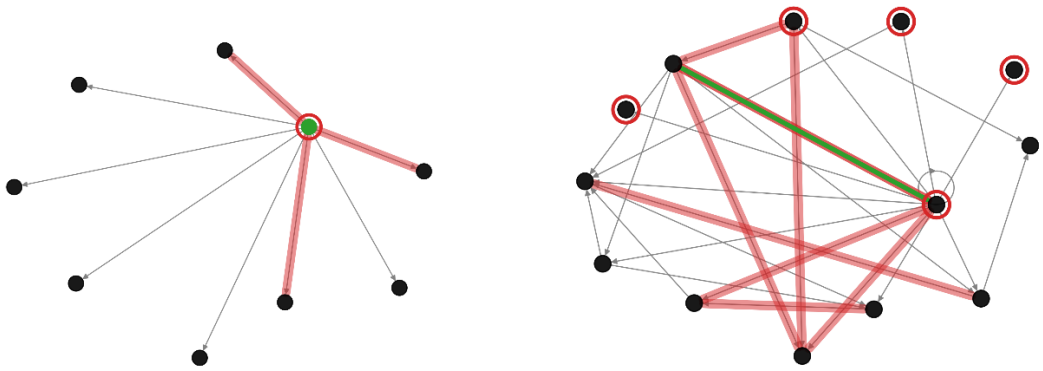

Fig. 9. Result of XAI explanations over two different states S_t. The red lines (edge features) and red dots (node feature) denote the importance attributed for a single chosen action (green). *Local* actions are represented in the node in which it is executed (green dot), while *Remote* and *Connect* actions are represented by an edge connecting the source and target nodes (green edge).

The result is a one-to-one correspondence between the input and feature importance attributed. Notice that MLP policy explanations are not included because it is similar to the GNN. In order to be visualized, the node and edge feature attributions are aggregated, then, top-k nodes and edges are highlighted. An example of the results is shown in Fig. 9.

6. Conclusions and further work

The INFAST project has been used to explore new solutions based on Artificial Intelligence for test automation at ESOC. During this project, GMV carried out two very different use cases. The first one aimed at automating the analysis tasks related to the search for root causes of functional test failures. For the second one, the main objective was to prioritise penetration tests in the area of cybersecurity, thus selecting the next action to be taken based on the information available. The development of both required a long period of experimentation in order to adapt the solutions to the needs identified. As a result, a proof-of-concept tool has been developed to demonstrate its benefits in real test environments. Although this phase is still in progress, the project has allowed several conclusions to be drawn, which are presented below.

Firstly, the use of NLP techniques (including Transformers, TF-IDF or BM25) to search for similarities between test failures from their descriptions and logs appears to be a suitable approach. This has allowed failures to be clustered and possible root causes to be identified. However, the search for relationships between test failures and SPRs based on their texts has served to identify difficulties in relating texts written in different registers: in this case, an automatic text generated by a tool and another text written by an operator. One solution may be to try to standardise the way the two texts are generated in the future, so that they have a similar structure and language. In addition, further steps to improve this use case were identified. The first one, which is the most important one, is the use of additional data sources such as SUT logs, test specifications and/or repository commit history. The second one is to explore other techniques, such as automatic question-answering for root cause identification using the errors identified in the logs, named entity recognition and regular expressions for text pre-analysis, or the use of Bayesian root cause identification techniques.

For the second use case, the INFAST project was used to investigate the use of an RL-based approach for prioritising pentesting tasks. To this end, a simulated network topology generator was developed for training RL agents. These topologies aim to be as realistic as possible in order to adapt the agent's behaviour to what it will encounter in a real environment. Various architectures have been used, with the use of GNNs standing out. This type of network has shown great potential for solving problems such as the one under investigation. However, this is only the first step in this line of research and there are several options to be explored in the future, including: improving the transfer of knowledge from the simulated environment to the real one (sim2real); improving the simulation environment by including new actions and protocols; or exploring subfields of reinforcement learning such as offline learning, meta-learning, and active learning. In addition, the use case focuses on prioritising actions in pentesting tasks, where the goal is to identify and exploit all existing vulnerabilities. However, another approach could be the Red Team, where the prioritisation of actions is focused on finding a vulnerable machine and making it their own with maximum privileges.

Finally, it is worth mentioning the use of XAI techniques during the project. This has made it possible to explain the decision-making process of the models used, avoiding the use of so-called black box models, with the aim of adapting the solution to the current context, which favours the use of trustworthy AI.

References

[1] F. Ricca, A. Marchetto, and A. Stocco, "AI-based Test Automation: A Grey Literature Analysis," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2021, pp. 263–270. doi: 10.1109/ICSTW52544.2021.00051.

[2] A. Trudova, M. Dolezel, and A. Buchalcevova, "Artificial Intelligence in Software Test Automation: A Systematic," 2020.

[3] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu, "A Survey on Machine Learning Techniques for Cyber Security in the Last Decade," *IEEE Access*, vol. 8, pp. 222310–222354, 2020, doi: 10.1109/ACCESS.2020.3041951.

[4] N. A. Almubairik and G. Wills, "Automated penetration testing based on a threat model," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, Dec. 2016, pp. 413–414. doi: 10.1109/ICITST.2016.7856742.

[5] R. Gangupantulu *et al.*, "Using Cyber Terrain in Reinforcement Learning for Penetration Testing," *ArXiv210807124 Cs*, Aug. 2021, Accessed: Aug. 28, 2021. [Online]. Available: http://arxiv.org/abs/2108.07124

[6] J. Schwartz, H. Kurniawati, and E. El-Mahassni, "POMDP + Information-Decay: Incorporating Defender's Behaviour in Autonomous Penetration Testing," *Proc. Int. Conf. Autom. Plan. Sched.*, vol. 30, pp. 235–243, Jun. 2020, Accessed: Aug. 28, 2021. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/6666

[7] M. C. Ghanem and T. M. Chen, "Reinforcement Learning for Efficient Network Penetration Testing," *Information*, vol. 11, no. 1, Art. no. 1, Jan. 2020, doi: 10.3390/info11010006.

[8] Z. Hu, R. Beuran, and Y. Tan, "Automated Penetration Testing Using Deep Reinforcement Learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, Sep. 2020, pp. 2–10. doi: 10.1109/EuroSPW51379.2020.00010.

[9] R. Elderman, L. Pater, A. Thie, M. Drugan, and M. Wiering, "Adversarial Reinforcement Learning in a Cyber Security Simulation," Feb. 2017. doi: 10.5220/0006197105590566.

[10] L. Rosenbauer, D. Pätzel, A. Stein, and J. Hähner, "A Learning Classifier System for Automated Test Case Prioritization and Selection," *SN Comput. Sci.*, vol. 3, no. 5, p. 373, Jul. 2022, doi: 10.1007/s42979-022-01255-1.

[11] B. Busjaeger and T. Xie, "Learning for test prioritization: an industrial case study," Nov. 2016, pp. 975–980. doi: 10.1145/2950290.2983954.

[12] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration," *Proc. 26th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, pp. 12–22, Jul. 2017, doi: 10.1145/3092703.3092709.

[13] M. J. Arafeen and H. Do, "Test case prioritization using requirements-based clustering," in *2013 IEEE sixth international conference on software testing, verification and validation*, 2013, pp. 312–321.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018.

[15] A. Pingle, A. Piplai, S. Mittal, A. Joshi, J. Holt, and R. Zak, "Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2019, pp. 879–886.

[16] A. Vaswani *et al.*, "Attention Is All You Need." arXiv, Dec. 05, 2017. doi: 10.48550/arXiv.1706.03762.

[17] S. Tahvili, L. Hatvani, E. Ramentol, R. Pimentel, W. Afzal, and F. Herrera, "A novel methodology to classify test cases using natural language processing and imbalanced learning," *Eng. Appl. Artif. Intell.*, vol. 95, p. 103878, 2020.

[18] M. Chen *et al.*, "Evaluating large language models trained on code," *ArXiv Prepr. ArXiv210703374*, 2021.

[19] J.-C. Berton, K. Widegard, E. Gomez Gomez, C. Smith, and L. Teixeira, "The ESOC End-to-End Ground Segment Reference Facility," in *2018 SpaceOps Conference*, 2018, p. 2513.

[20] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, "A Survey of Deep Learning Methods for Cyber Security," *Information*, vol. 10, no. 4, Art. no. 4, Apr. 2019, doi: 10.3390/info10040122.

[21] K. Qian, D. Zhang, P. Zhang, Z. Zhou, X. Chen, and S. Duan, "Ontology and Reinforcement Learning Based Intelligent Agent Automatic Penetration Test," in *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, Jun. 2021, pp. 556–561. doi: 10.1109/ICAICA52286.2021.9497911.

[22] J. Schwartz and H. Kurniawati, "Autonomous Penetration Testing using Reinforcement Learning," *ArXiv190505965 Cs*, May 2019, Accessed: Aug. 28, 2021. [Online]. Available: http://arxiv.org/abs/1905.05965

[23] S. Niculae, D. Dichiu, K. Yang, and T. Back, "Automating Penetration Testing using Reinforcement Learning," p. 13.

[24] M. C. Ghanem and T. M. Chen, "Reinforcement Learning for Intelligent Penetration Testing," in *2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, Oct. 2018, pp. 185–192. doi: 10.1109/WorldS4.2018.8611595.

[25] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges," *ArXiv210413478 Cs Stat*, May 2021, Accessed: Dec. 25, 2021. [Online]. Available: http://arxiv.org/abs/2104.13478

[26] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks." arXiv, Feb. 04, 2018. doi: 10.48550/arXiv.1710.10903.

[27] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks." arXiv, Feb. 22, 2017. doi: 10.48550/arXiv.1609.02907.

[28] A. B. Arrieta *et al.*, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI." arXiv, Dec. 26, 2019. doi: 10.48550/arXiv.1910.10045.

[29] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic Attribution for Deep Networks." arXiv, Jun. 12, 2017. doi: 10.48550/arXiv.1703.01365.