Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Vulnerability detection through machine learning-based fuzzing: A systematic review

Sadegh Bamohabbat Chafjiri [*], Phil Legg, Jun Hong, Michail-Antisthenis Tsompanas

University of the West of England, Coldharbour Lane, Bristol, BS16 1QY, UK

ARTICLE INFO

Keywords:
ML fuzzing
TML fuzzing
DNN fuzzing
RL fuzzing
DRL fuzzing

ABSTRACT

Modern software and networks underpin our digital society, yet the rapid growth of vulnerabilities that are uncovered within these threaten our cyber security posture. Addressing these issues at scale requires automated proactive approaches that can identify and mitigate these vulnerabilities in a suitable time frame. Fuzzing techniques have emerged as crucial methods to preemptively tackle these risks. However, traditional fuzzing methods encounter various challenges, such as a lack of strategy for deep bug identification, time-intensive bug analysis, quality of inputs, seed scheduling and others. To overcome these challenges, diverse Machine Learning (ML) models and optimisation techniques have been employed, including advanced feature engineering, optimised seed selection, refined predictive/fitness models, and Gradient-based optimisation. Furthermore, the use of ML architectures such as Long Short-Term Memory (LSTM), Generative Adversarial Network (GAN), Sequence-to-Sequence (Seq2Seq), and Generative Randomised Unit (GRU), have demonstrated greater effectiveness within ML-based fuzzing. In this paper, we delve into this paradigm shift, aiming to address fundamental challenges across different ML categories. We survey popular ML categories such as Traditional Machine Learning (TML), Deep Learning (DL), Reinforcement Learning (RL), and Deep Reinforcement Learning (DRL), to investigate their potential for enhancing traditional fuzzing approaches. We explore the respective advantages in each category of ML-based fuzzing, while also analysing the challenges unique to each category. Our work provides a comprehensive survey across the fuzzing domain and how machine learning techniques have been utilised, that we believe will be of use to future researchers in this domain.

1. Introduction

Fuzzing is the process of automated software testing to assess how a given system, whether it be an application or a network tool, handles various forms of input, including unexpected and random data inputs generated automatically.

The process of software testing is both part of the established development lifecycle and a key component of software security testing to uncover potential vulnerabilities caused by bugs that could be further exploited by an adversary. Software vulnerabilities such as Heartbleed (Carvalho et al., 2014), Shellshock (Anon, 2014), and log4j (Anon, 2021) are good examples of these vulnerabilities that have been widely reported and are known to have had devastating impact on many organisations. Hence, the identification of software vulnerabilities using fuzzing analysis is vital for maintaining a strong cyber security posture. The traditional concept of fuzzing was introduced by Miller back in 1988 and also through his work in the early 1990s (Miller et al., 1990, 1995). Fuzzing methods have since evolved to cover a range of white-box (Molnar et al., 2008), grey-box (Böhme et al., 2017) and black-box (Abdelnur et al., 2007) methods, giving full, partial and zero access to software details respectively. A variety of fuzzing techniques have since been proposed (Takanen, 2009; Miller and Peterson, 2007; Felderer et al., 2016) including generation-based fuzzing, mutation-based fuzzing and evolution-based fuzzing, including Genetic Algorithms (She et al., 2020) and the popular American Fuzzing Lop (AFL) (Zalewski, 2020). Most recently, and as the motivation for this systematic review, Traditional Machine Learning (TML), Deep Learning (DL), Reinforcement Learning (RL), and Deep Reinforcement Learning (DRL) techniques are now popular methods amongst the research community (Saavedra et al., 2019; Wang et al., 2020a; Miao et al., 2022; Mallissery and Wu, 2023; Daniele et al., 2024) that have been used to help address some of the most pressing fuzzing challenges that exist today:

- **Bug Identification and Prioritisation:** Whilst fuzzing may report crashes, it remains challenging to identify and distinguish different types of bugs, especially on a large scale. Furthermore,

* Corresponding author.
E-mail address: sadegh2.bamohabbatchafjiri@uwe.ac.uk (S. Bamohabbat Chafjiri).

https://doi.org/10.1016/j.cose.2024.103903
Received 17 December 2023; Received in revised form 11 April 2024; Accepted 15 May 2024
Available online 19 May 2024

deciding which crashes should be given higher priority for resolution is complex. ML is effective for classification tasks, and so can be used to classify and prioritise bugs effectively (Tripathi et al., 2017).

- **Time-Consuming Bug Analysis:** Conducting in-depth analysis of identified bugs, and identifying suitable mitigations through patching, can be time-consuming. ML can assist in automating bug analysis to identify suitable patching processes earlier (Zhang and Thing, 2018).
- **Seed Scheduling:** Since some fuzzing techniques rely on seeding (for randomisation), researchers are interested in developing heuristic rules for predicting the scheduling of seed inputs. ML models can provide insights into optimising seed selection strategies for efficient fuzzing (Chen et al., 2020).
- **Data Flow Interpretation:** Current fuzzing techniques rely on interpreting data flow to extract specifications of models. However, there is a need for more complex models that can capture both control flow and data flow features, especially when analysing different protocol specifications automatically. ML can assist in developing more accurate models for this purpose (Lin et al., 2020; Huang et al., 2022a).
- **Syntactic and Semantic Property Prediction:** Predicting both syntactic and semantic properties of a program is a challenging task. ML algorithms can be used to build models that can predict these properties effectively, enhancing fuzzing capabilities (Raychev et al., 2015).
- **Ambiguity in Analysing Defects:** Ambiguity in identifying defects within language interpreters poses a challenge for traditional fuzzers. ML can aid in developing techniques to handle this ambiguity and improve the effectiveness of fuzzing (Sun et al., 2018).
- **Deep Bugs and Input Generation:** Deep bugs often occur during the execution of software processing structured inputs, making them hard to identify and address, particularly in stateful systems. Creating test inputs to trigger these deep bugs automatically is complex. ML can help in generating effective test inputs and addressing deep bugs. Additionally, ML can assist in reducing the rejection rate during the initial syntax parsing stage of mutation-based fuzzing, enhancing its efficiency (Wang et al., 2017).
- **Mutation Operator Selection:** Choosing suitable mutation operators for fuzzing poses a significant challenge, particularly in cases where a uniform distribution of mutators is employed. Additionally, the phenomenon of saturation, as outlined in literature (Groß et al., 2022), exacerbates this challenge by generating inputs that predominantly traverse paths already explored, while neglecting others. ML can assist in making more informed decisions about mutation operator selection, increasing the likelihood of generating interesting inputs (Karamcheti et al., 2018).

In this paper, we conduct a systematic review of the academic literature for ML-based fuzzing. We structure our review around four key types of machine learning: traditional Machine Learning (TML) techniques, Deep Learning (DL), Reinforcement Learning (RL), and Deep Reinforcement Learning (DRL). In this way, we also study the evolution of fuzzing across these four types, with deeper models gained greater traction in recent years as computing power has increased. As part of our review, we highlight four key contributions that this survey can provide for the academic research community:

- **Enriching Traditional Models**: Our survey shows how ML techniques can enhance fuzzing across different stages, such as test-case generation, execution of Software Under Test (SUT), runtime state checking, and analysis of vulnerabilities. Within traditional models, we show the application of feature-based, predictive, fitness, and long-input correlation tracing models. These techniques demonstrate promising results for improving fuzzing effectiveness in detecting vulnerabilities.
- **Expanding DL Techniques**: Our survey explores frequently used exploration and optimisation approaches using DL; specifically, dynamic information prioritisation and gradient-guided optimisation, along with different architectures, such as standard or bidirectional LSTM networks, GANs, and hybrid models combined with seq2seq and GRU models. Our survey also explores techniques deployed against complex targets, such as Kernel and File System (FS) fuzzing, and suitable datasets for Operating System (OS) fuzzing, to emphasise the role of DL models in identifying complex bugs and vulnerabilities.
- **RL and Hybrid Approaches**: Our survey explores how RL can be combined with various techniques, such as LSTM and dynamic mutation analysis, seed scheduling, and Kernel fuzzing. This synthesis of RL along with other techniques illustrates the potential for achieving advanced fuzzing outcomes, such as uncovering bugs in a reduced time frame.
- **Enhancing DRL Landscape**: Our survey illustrates the enrichment of DRL through the integration of a Curiosity-driven model, deep Q-learning (DQL), and frameworks combining DNN and RL. This integration not only expands the landscape of DRL, but also introduces novel approaches for improving the efficiency of fuzzing techniques.

The remainder of the paper is organised as follows. Section 2 provides further motivation for why this review is necessary to the research community. Section 3 outlines our data collection methodology and criteria for how we have conducted the systematic review of the existing academic literature. Section 4 discussed traditional Machine Learning techniques in application to fuzzing, followed by Section 5 that discusses Deep Learning techniques. Sections 6 and 7 explore the use of Reinforcement Learning and Deep Reinforcement Learning respectively. Finally, Section 8 discusses the current research landscape, identifying key limitations and areas of further research, and Section 9 concludes the survey.

2. Motivation

There are several survey papers that have previously addressed the subject of machine learning-based fuzzing, including (Saavedra et al., 2019; Wang et al., 2020a; Miao et al., 2022; Mallissery and Wu, 2023; Daniele et al., 2024). These studies explore the diverse applications of ML techniques in areas such as seed or test case generation and program analysis. For example, Saavedra et al. (2019) surveyed the application of ML in fuzzing, discussing its utilisation in the generation of inputs and the analysis of interesting program states post-fuzzing. They also highlighted challenges such as the limited use of supervised learning techniques and issues related to training data and computational complexity. In Daniele et al. (2024), Daniele et al. surveyed stateful systems and offered a systematic comparison and classification of these fuzzers. In Mallissery and Wu (2023), Mallissery et al. categorise fuzzing methods by application domains and techniques, differentiate between source code-independent and static analysis-based approaches, and explore symbolic and concolic execution through a case study. They also examine target fuzzing domains like firmware/kernel issues and instrumentation error-focused fuzzers, and address associated challenges in fuzzing. In another study by Miao et al. (2022), the authors conducted a thorough investigation of deep learning within fuzzing methodologies, emphasising the role of deep learning in test case creation, monitoring, scheduling, and program analysis. The work also addresses current issues and potential future directions in deep learning-based fuzzing research, underlining its ongoing importance in security research. Similarly, Wang et al. (2020a) presented a systematic survey offering a comprehensive overview of distinct ML-based methodologies, investigating supervised, semi-supervised, and unsupervised techniques. They assess the prevalence of ML across published works, with a particular focus on vulnerability analysis.

The literature survey reveals that ML encompasses an array of supervised, semi-supervised, and unsupervised techniques, facilitating the automated acquisition and modelling of complex systems. The surveyed papers involve diverse fuzzers based on the specific steps at which the designated ML technique is applied. Furthermore, they analyse the prevalence distribution of ML techniques across the spectrum of published works. The authors categorise these algorithms into three general classes (Wang et al., 2020a): TML, DL, and RL, while concurrently examining their role in vulnerability analysis.

Nevertheless, there remains the opportunity for further exploration of diverse papers with a level of detail and under various criteria that have not yet been comprehensively examined in preceding surveys of the literature. For instance, the comparison tables in the referenced paper (Wang et al., 2020a) could benefit from clearer organisation. By providing a concise compilation of the various fuzzing techniques, inputs, outputs and SUTs, researchers will be able to assign algorithms to specific sub-categories much easier. This clearer structure would enhance the understanding of the strengths and weaknesses of each technique, enabling researchers and practitioners to make more informed decisions about which approach best suits their needs. For instance, in a scenario where a fuzzer is associated with a specific architecture, such as LSTM, the connection between the architecture and the literature employing that specific architecture is absent in previous surveys such as Wang et al. (2020a). Examining the number of studies that have utilised the specific architecture is not possible in previous surveys. Consequently, it remains unclear whether the literature aligns with a standard or a hybrid model of the specific architecture mentioned.

As a result, researchers may face challenges in establishing connections between the names or structural attributes of different fuzzers, or in being able to replicate previous studies for further experimentation. This difficulty extends to identifying and categorising papers into different models based on corresponding fuzzer categories, subcategories, and underlying classifiers. Our work aims to help resolve these issues to aid future researchers who are exploring this topic, and to aid reproducibility of scientific experimentation.

3. Data collection methodology

We outline our methodology for data collection, including eligibility criteria of works and methods for catergorisation, to give a comprehensive review of the existing literature and identified gaps. Our selection approach follows that of the PRISMA framework (the Preferred Reporting Items for Systematic Meta-Analysis) (Page et al., 2021). Fig. 1 illustrates the detailed selection process that we have adopted for this survey.

3.1. Eligibility criteria and search terms

To ensure a comprehensive literature review, we carefully reviewed previous systematic literature reviews and then crafted some search terms to identify the most pertinent articles. Search terms, listed in Table 1, were determined to capture relevant research in our topic areas. Additionally, we utilised a bibliography of systematic survey papers published in the past and received personalised recommendations based on our most recent signed-in activity on the databases listed in Section 3.2.

We aimed to achieve broad coverage of relevant literature using these search queries. As a result, we made a deliberate choice to include works found on different platforms.

After collating the search results, we eliminated any duplicate entries, which were available on other well-known databases. The search encompassed various databases, and it was conducted up to October 2023. Subsequently, we screened each paper by reviewing the title and abstract, applying inclusion criteria based on the relevance of the article in ML-oriented fuzzing, particularly focusing on the four categories outlined in Table 1.

Table 1
Topics and corresponding search terms utilised in this study.

Topic	Search query
Traditional Machine Learning Fuzzing	Fuzzing, Machine Learning
Deep Learning Fuzzing	Fuzzing, Deep Learning, Deep Neural Network
Reinforcement Learning	Fuzzing, Reinforcement Learning
Deep Reinforcement Learning	Fuzzing, Deep Reinforcement Learning
Survey Literature in Fuzzing	Survey on ML-based Fuzzing

3.2. Database selection

We chose several databases with a significant computer science component to conduct our literature searches and retrieve relevant publications. The selected databases were ScienceDirect, ACM Digital Library, IEEE Xplore, SpringerLink, and Arxiv. ArXiv is not a peer-reviewed platform; thus, any works referred to from ArXiv must be carefully assessed with a greater level of scrutiny. We selected papers that appear to have been impactful on the fuzzing domain, as judged by further citation in peer-reviewed journal articles, or that have been submitted to the pre-print server only within the last year (i.e. 2023). The growing popularity of releasing pre-print work early on platforms such as ArXiv demonstrates the fast pace of technology and enables inclusion of state-of-the-art techniques in this survey.

3.3. Methodology for categorisation

In light of the escalating significance of robust fuzzing techniques for bolstering software security, our paper presents a pivotal contribution through an exhaustive survey of research literature. Unlike prior surveys, our study delves comprehensively into the employed techniques, input–output mechanisms, and models or architectures across various fuzzing methodologies. By offering an in-depth analysis, we aim to provide a more nuanced understanding of the evolving landscape of fuzzing methodologies, thereby facilitating informed decision-making for researchers and practitioners in the field of software security. We aim to provide a holistic view of the various categories and subcategories within the realm of fuzzing, particularly emphasising the role of ML-based methodologies. A sample representation of ML-based fuzzing is depicted in Fig. 2. Building on the model presented by Cheng et al. (2019), we extend this to encompass all ML-based fuzzing models to provide a general representation of the relationship between input data and the target. The figure illustrates a comprehensive model of ML-based fuzzing. The learning stage from the target may utilise either supervised or unsupervised models, with the choice depending on the specific requirements and the nature of the software being tested.

In this paper, we surveyed research papers from different perspectives where we have four categories listing the most popular techniques used in the fuzzing domain as TML, DL, RL and DRL. We discuss our research approach in the field of ML-based fuzzing, which is detailed in subsequent sections.

4. TML techniques for fuzzing

TML techniques rely on predetermined feature techniques and mathematical calculations to handle Regression and Classification problems. In TMLs, a supervised model is trained on labelled data. It generates predictions based on discovered patterns in different ways and improves the fitness function that can discover high-quality test cases. In multiple pieces of research, to make predictions or categorise input data, ML techniques rely on predetermined features and mathematical functions that can guide the input. In this section, we introduce three different approaches to TML models: Feature-based Models, predictive/Fitness Models, and long-input correlation tracing models. Each approach highlights specific characteristics of the input

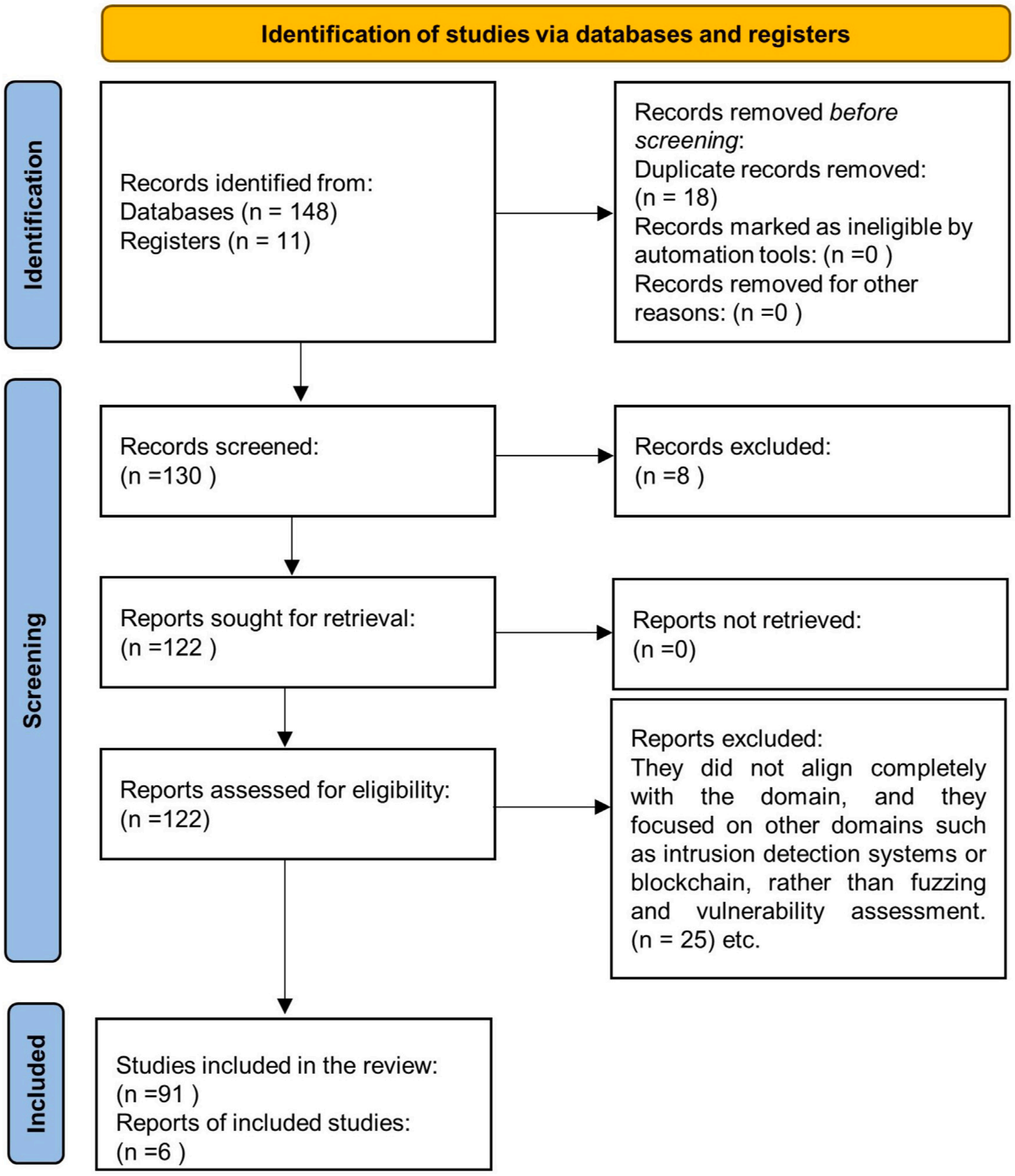
Identification of studies via databases and registers
Identification
Records identified from:
Databases (n = 148)
Registers (n = 11)
Records removed *before screening*:
Duplicate records removed:
(n = 18)
Records marked as ineligible by automation tools: (n =0)
Records removed for other reasons: (n =0)
Screening
Records screened:
(n =130)
Records excluded:
(n =8)
Reports sought for retrieval:
(n =122)
Reports not retrieved:
(n =0)
Reports assessed for eligibility:
(n =122)
Reports excluded:
They did not align completely with the domain, and they focused on other domains such as intrusion detection systems or blockchain, rather than fuzzing and vulnerability assessment. (n = 25) etc.
Included
Studies included in the review:
(n =91)
Reports of included studies:
(n =6)

Fig. 1. Review findings utilising the PRISMA framework.

file and test cases or applies a new model to guide the fuzzing process or optimises current models. In Table 2 we introduced each TMLs model's Specifications. A feature-based protocol model evaluates characteristics and features to aid decision-making using a predefined set of features. On the other hand, predictive or fitness models emphasise the optimal relationship between input and output by predicting the behaviour of the target program using various techniques. Additionally, optimisation models assess the quality of generated inputs across different fuzzer structures and evaluate their impact on code coverage.

4.1. Feature-based models

Feature-based models make decisions about selecting, generating, or modifying inputs based on predefined input variables. They are constructed using training data, allowing them to correlate input attributes with predicted outputs. Techniques like linear regression, decision

Table 2
Different types of TML models.

Model type	Description
Feature-based Models	Evaluates features for decision-making. Uses a predefined set of features.
Predictive/Fitness Models	Predicts target program behaviour. Focuses on the input–output relationship.
Long Input Correlation Tracing Models	Assesses the quality of generated inputs by long input correlation checking. Evaluate its impact on code coverage.

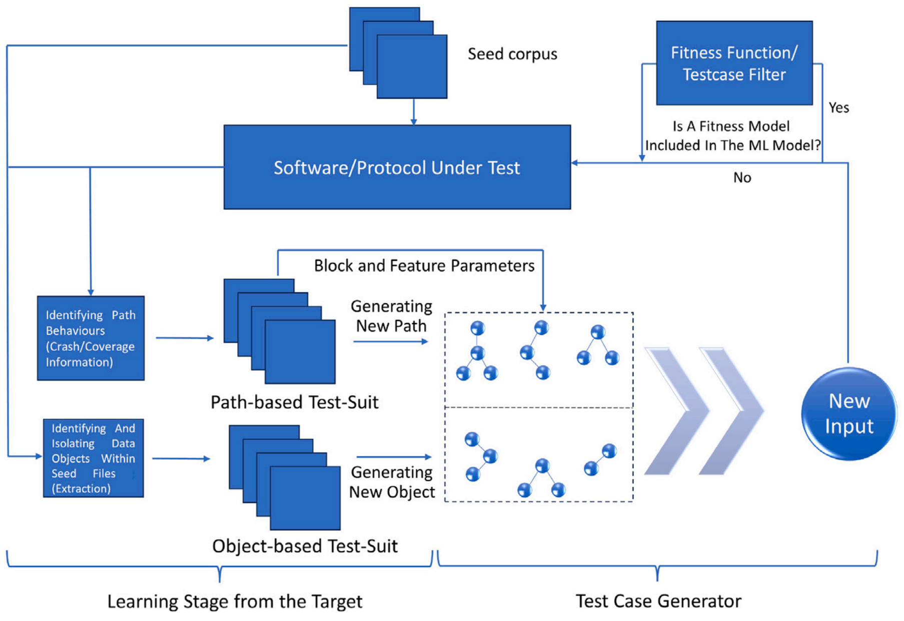
Seed corpus
Fitness Function/ Testcase Filter
Yes
Is A Fitness Model Included In The ML Model?
No
Software/Protocol Under Test
Block and Feature Parameters
Identifying Path Behaviours (Crash/Coverage Information)
Generating New Path
Path-based Test-Suit
Identifying And Isolating Data Objects Within Seed Files (Extraction)
Generating New Object
Object-based Test-Suit
New Input
Learning Stage from the Target
Test Case Generator

Fig. 2. A sample model of ML-based fuzzing structure.

trees, support vector machines, and neural networks fall under this category.

Innovations in this domain, such as Exniffer (Tripathi et al., 2017), utilise SVM and runtime features to identify security-critical crashes and analyse their exploitability. What sets modern approaches apart is their flexible online learning, departing from traditional offline methods and allowing model updates without complete retraining. Another example of this method is MEUZZ (Chen et al., 2020) embodies hybrid features for real-time and offline learning by dynamically adjusting seed schedules based on past decisions. This significantly enhances fuzzing outcomes, focusing on achieving greater accuracy compared to heuristic methods while maintaining fuzzing efficiency.

Feature-based models also excel in reverse engineering. For example, Lin et al. (2020) used clustering and Extended Finite State Machines for effective protocol evaluation. Huang et al. (2022a) employed predefined features in protocol reverse-engineering, addressing security and privacy concerns. Additionally, Feng et al. (2020) proposed the use of techniques such as exception field positioning and employed field attribute set models to pinpoint vulnerabilities in the Modbus protocol, demonstrating the effectiveness of feature-based fuzzing.

In summary, feature-based models prove to be potent tools in fuzzing, leveraging predefined input attributes for better decision-making. Notably, they excel in security analysis and protocol reverse engineering, showcasing their versatility and effectiveness.

4.2. Fitness functions and predictive models

In this section, we present two distinct techniques that utilise efficient models over feature functions, namely fitness functions and predictive models for sophisticated codes. The differences between fitness functions and predictive models in the context of fuzzing are distinct yet complementary. Fitness functions primarily serve to evaluate the quality of test cases by assigning scores based on specific criteria like code coverage or crash detection, helping to prioritise the most promising testcases. In contrast, predictive models aim to simulate the behaviour of the target application when subjected to various inputs, aiding in the identification of inputs more likely to trigger vulnerabilities. Also, predictive models identify various input file characteristics using an approximate algorithm, combining unknown and known properties. This approach slightly differs from feature-based models that rely on the classic method of predefined features for predictions. In this context, predictive models and fitness functions concentrate on probabilistic models approximating complex behaviours in big codes to enhance optimisation efficiency compared to feature-based models using a less complex model.

One notable early model is JSNICE, utilising Conditional Random Fields (CRFs) to predict program properties in JavaScript (Raychev et al., 2015). It transforms input programs into a suitable representation and leverages structured prediction with machine learning, employing graphical models like Conditional Random Fields (CRFs). This approach is essential since both the context and the sequence of blocks and features are important. Additionally, it utilises the Structured Support Vector Machine (SSVM) classifier, optimised through gradient descent, to handle the relationship between structured and interdependent inputs and outputs.

To address uncommonness, Sun et al. developed the Naturalness Heuristic for Fuzzing (NHF), a fitness function for language fuzzing using genetic programming and geometric mean (Sun et al., 2018). NHF computes script probabilities based on the Markov-Chain Probabilistic Context-Free Grammar (MPCFG Model), minimising the influence of script length on the coverage metric and fitness value.

In summary, these models represent advancements in predictive modelling and fitness functions, showcasing the effectiveness and versatility of ML in dynamically predicting program properties through predictive models or optimising the fuzzing process through fitness functions.

4.3. Long input correlation tracing models

The Long Input Correlation Tracing Models for Seed Generation discussed in this section effectively allocate resources in scenarios with multiple choices or options, particularly in decision-making amidst uncertainty, such as selecting well-distributed seed samples and filtering testcases for fuzzing to expose software vulnerabilities (Wang et al., 2017). For example, the length of a script influences the fitness value, where Probabilistic Context-Free Grammars (PCFG) introduced in the previous subsection are only suitable for basic fuzzing tasks, while Probabilistic Context-Sensitive Grammars (PCSG models) are powerful tools for discovering complex dependencies. They enable the fuzzer to generate highly realistic inputs based on both syntax and semantic validation. Skyfire's (Wang et al., 2017) data-driven approach leverages PCSG to generate diverse and well-distributed inputs, tracing longer

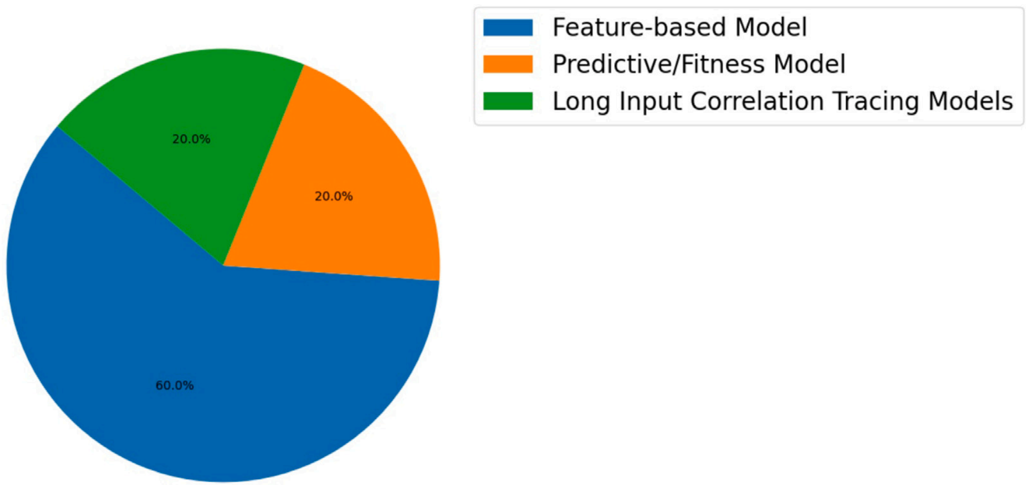

Fig. 3. Frequency of reviewed literature in three subcategories of TML models.

correlations than previously explained. It offers advantages in optimising resources, particularly for fuzzing a closed-source target by using fit test cases compared to PCFG. In the realm of Multi-armed Bandit problems, Thompson Sampling optimisation has been employed in the AFL to adaptively learn mutator distributions and dynamically change resource allocation based on the observed results from earlier test cases, thereby maximising computational efficiency (Karamcheti et al., 2018). Ultimately, this optimisation technique generates diverse and well-distributed inputs, traces longer input correlations, and maximises valuable results, while, optimising computational efficiency.

4.4. Assessment of TML techniques

By examining the pie chart and the distribution of each TML model depicted in Fig. 3, it is evident that most studies have concentrated on fitness models. In comparison, predictive/fitness and Long Input Correlation Tracing Models have a smaller share compared to feature-based models, despite demonstrating potential in enhancing automated capabilities and scalability for fuzzing sophisticated targets. This preference could be attributed to the simplicity of feature-based models, which allows experts in fuzzing to work with relative ease without needing in-depth knowledge or skills required for other supervised learning techniques. Fitness models allow testers to concentrate on specific goals and contexts within the fuzzing process, thus highlighting their advantages in feature selection, adaptability, and customisation. The feature-based model can potentially detect vulnerabilities earlier, reducing the time and resources needed for testing. Nonetheless, predictive/fitness and long input correlation tracing models offer more opportunities for adapting the fuzzing process based on feedback from fuzzing output and previous attempts. Automated fuzzing can thus become more effective over time, especially when employing sophisticated algorithms like genetic algorithms, which are well-suited for maximising code coverage and efficiently allocating resources in the automated fuzzing of complex bugs over a longer duration.

Also, we summarised TML techniques that were surveyed in this study in Table 3. As can be seen, seed file generation is the most common output of different fuzzing techniques and the SVM family as the classifier was the most frequently used in fuzzing. SVMs are helpful when you have more types of data (features) than actual data points (samples). This happens a lot in fuzzing, where you are dealing with a big and complicated range of inputs. SVMs are good at saving memory compared to other classifiers because they only use some of the data points (called support vectors) to make decisions. This is great for fuzzing, where handling lots of data is important. When you have way more types of data than data points, SVMs are less likely to overfit, or get too tuned to the specific data you have, especially if you adjust the controls (regularisation) well. Also, SVMs work well when one type of outcome (like finding a vulnerability) is much rarer than others, which is a common situation in fuzzing.

4.5. Challenges in TML models

While the TML model is effective, its practical use is limited by constraints such as limited code coverage caused by poor datasets. Additionally, TML struggles with complex, high-dimensional data despite its broad applicability. The ability to adapt fuzzing methods to new vulnerabilities or targets is also limited. The choice of TML models depends on the testing environment, the nature of the software, and the availability of domain-specific expertise. This can lead to inefficiency in unfamiliar fuzzing environments. Moreover, the balance between exploration and exploitation (Wang et al., 2021b) in the search space, and the selection of a fuzzing model, depend on the context and requirements of the software, data quality, and implementation complexity. Ensuring the reliability and effectiveness of various fuzzers across different targets remains a significant challenge. In these scenarios where a company lacks this expertise, there is a heightened risk of misallocating resources by focusing on skills or approaches that are not essential. This misdirection can result in inefficiencies, particularly in unfamiliar fuzzing environments, where resources might be better utilised on more critical and relevant tasks. Such a mismatch in resource allocation underscores the importance of aligning expertise with the specific requirements of the TML models and the testing context and highlights the role of automated testing by the integration of semi-supervised and unsupervised learning such as DL, underpinned by the architecture of DNNs, has emerged as a promising solution which is explained in the following section.

5. Deep neural network techniques for fuzzing

The application of DNN techniques is motivated by the challenges encountered in TML methodologies (see Section 4.5). DNNs utilise interconnected nodes across multiple layers to automatically extract hierarchical representations from raw data, allowing for efficient handling of complex predictive and discriminative tasks. Additionally, integrating grey-box fuzzing has shown promise in bug identification across various applications, enhancing the capabilities of DNNs. Specifically, when dealing with complex and high-dimensional data, DNNs outperform TML methods due to their ability to discern intricate patterns and dependencies.

Therefore, in the second category of our systematic survey, we delve into the factors that drive the widespread integration of DNNs

Table 3
Examples of Different Fuzzers Using Different Learning Techniques in the TML Category. C1: Seed File Generation, C2: Message Ready to Send to the SUT, C3: Grammar.

	Fuzzer Name/Structure	Technique	Classifier/Clustering	Input	Output			SUT
					C1	C2	C3	
Feature-based Models	Exniffer (Tripathi et al., 2017)	Analysing run-time information such as LBR register, and core-dump	SVM	Real world benchmark (LAVA, C/C++ test cases)	✓			Real-world apps
	MEUZZ (Chen et al., 2020)	Features extracted through code reachability and dynamic analysis	Random Forest for offline learning and linear model for online learning	Real-world benchmark	✓			Real-world apps
	Compact dynamic fingerprints and incremental learning (Zhang and Thing, 2018)	Online learning algorithm and n-gram analysis and feature hashing	PA classifier	VDiscovery dataset	✓		✓	VDiscover real-world apps
	ReFSM/PRE clustering (Lin et al., 2020)	Extended Finite State Machines (EFSM)	Apriori and K-mean	Real-world network traffic traces		✓		4 Real-world network protocols: FTP, SMTP, BitTorrent and PPLive
	Exception field positioning (Feng et al., 2020)	Dimensionality reduction based on discernibility matrix and feature hashing, using decision tables, discernibility matrix, and mutation probability function	Clustering of similar packets based on attribute reduction	Modbus protocol packets	✓			Vulnerabilities of CNVD/PLC module as hardware
	PRE Methods (Survey paper) (Huang et al., 2022a)	Network trace (NetT) and Execution trace (ExeT)	Feature-based protocol classifier	Network traces or execution traces	✓			Network protocols
Predictive and Fitness Models	JSNICE (Raychev et al., 2015)	Conditional Random Fields (CRFs)	SSVM	JavaScript	✓	✓	✓	Javascript programs
	NHF (Sun et al., 2018)	Integrating Markov-Chain and PCFG	NHF fitness function	Java scripts, POC test cases of bug-reports from SpiderMonkey's bugzilla	✓	✓		JavaScript interpreter
Long Input Correlation Tracing Models	Skyfire (Wang et al., 2017)	Probabilistic Context-Sensitive Grammar (PCSG)	Not Applied	XSL, XML, and JavaScript	✓			XSLT and XML engines (i.e., Sablotron, libxslt, and libxml2)
	Thompson Sampling (Karamcheti et al., 2018)	Fine-tuning mutation	Not Applied	75 DARPA Cyber Grand Challenge (CGC) dataset	✓			Real world apps such as Mozilla Firefox, Adobe Flash, and OpenSSL

in fuzzing. We present a comprehensive survey encompassing optimisation features in DNN models, architectures, complex targets, and advancements using various DNN models, as prevalent in the existing literature. The survey provides insights into two state-of-the-art features, two baseline architectures and some hybrid models, as well as a survey on complex targets within the field of integrating DNN with fuzzing.

5.1. Exploration and optimisation approaches

The robust processing and learning capabilities of DNNs enable effective fuzzing, facilitating comprehensive exploration and learning from vast datasets. DNNs enable fuzzing models to effectively prioritise dynamic information and employ gradient-guided optimisation. These features, essential for leveraging large and complex search spaces, were not feasible with TML models. The limitations of TML models arise from their lack of deep hierarchical learning, limited optimisation techniques, and their structure, which relies on predefined feature sets. Therefore, in this section, we highlight two notable advantages of DNN-based exploration: prioritising dynamic information and leveraging gradient-guided optimisation.

5.1.1. Prioritising dynamic information

In this method, the baseline process of DNN facilitates the extraction of complex patterns and distinctive characteristics related to defect occurrence based on dynamic execution. A good example of this approach is NeuFuzz (Wang et al., 2019), drawing from the principles of PTfuzz (Zhang et al., 2018), which entails the prioritisation of pathways with a heightened likelihood of harbouring vulnerabilities. This model applies LSTM's distinctive memory function making it well-suited for managing extended dependencies, as instances of code linked to vulnerabilities might be situated at considerable distances within the path. One of the challenges is that the interconnections and interactions between different code components might spread across wide distances as software systems get large and complicated. Therefore, the scaling of symbolic execution, a technique that involves running a program with symbolic inputs instead of actual values, thereby allowing the concurrent exploration of multiple program paths, assumes a pivotal role in overcoming the aforementioned challenge. Certain scholarly works delve into strategies aimed at enhancing the scalability of symbolic execution to aid in addressing vulnerabilities and navigating expanded dependencies. For instance, challenges of Scaling Symbolic Execution motivated the utilisation of a neural network model to learn an effective and fast fuzzer "expert policy" from symbolic execution and generate a large number of quality inputs in He et al. (2019). They utilise a Markov Decision Process (MDP) in conjunction with an Imitation Learning-based Fuzzer (ILF system) that incorporates a combination of a Fully Connected Network, a GRU, and a Graph Convolutional Network.

However, generating high-quality seed inputs for fuzzing poses a challenge, and identifying correlations between seed files and program execution requires contextually relevant, and potentially impactful input sequences for software testing. In Cheng et al. (2019), Cheng et al. worked on this issue and introduced a generative model built on a recurrent neural network (RNN) framework to provide high-quality sequences for PDF-based program targets. They utilise a Seq2seq-based transition model to effectively translate inputs into valid PDF files. DeepHunter was another coverage-guided fuzzing framework focused on mutation strategy by employing a metamorphic mutation to generate new valid tests while maintaining semantics, thereby boosting coverage and defect identification (Xie et al., 2019). However, its approach has limitations, restricting multiple transformations and resulting in invalid inputs. To address this, a mixed and constrained mutation (MCM) approach with DeepHunter as the baseline of DL fuzzing was proposed in Park et al. (2022), significantly enhancing fuzzing performance by discovering more seeds and generating diverse adversarial examples. The need to overcome efficiency challenges ultimately led to the development of Tzer (Liu et al., 2022), a practical DL fuzzing technique introduced by Liu et al.. Tzer leverages the low-level Intermediate Representation (IR) of the Tensor Virtual Machine (TVM) tensor compiler, along with a general-purpose mutator, to enhance fuzzing efficiency.

Another challenge during fuzzing is imbalanced, unreachable inputs that arise early on. The lack of balanced labelled data leads to model overfitting. Consequently, the newly generated reachable inputs can differ significantly from those in the training set, causing model prediction failure. These results from novel inputs follow unobserved paths to buggy code, and as a result, training on a single path will not predict the new input's reachability. In Zong et al. (2020), Zong et al. studied this challenge and introduced directed grey-box fuzzing (DGF) called FuzzGuard, a deep-learning-based approach based on accelerating and prioritising techniques such as step-forwarding and representative data selection to handle unbalanced labelled data and limited training time. In addition to the above solution, to enhance coverage and mitigate overfitting, Wang et al. introduced a Fusion neural network named TCNN-BiGRU (Wang et al., 2023), amalgamating TextCNN and Bidirectional Gated Recurrent Unit (BiGRU). This innovative technique incorporates dimensionality reduction to capture both local and global features from vulnerability descriptions. By integrating dropout and early stopping methods, along with weighted word vectors and a Softmax classifier, it effectively combats overfitting, resulting in an enhanced overall performance and increased accuracy.

Using dynamic information collected from small pieces of code extracted from a system is another methodology applied in FreeFuzz which automatically captured dynamic information from code snippets, developer tests, and extensive exploration of open-source DL models, including prominent frameworks like PyTorch 1.8 and TensorFlow 2.4. This approach enables the execution of comprehensive fuzzing on DL libraries (Wei et al., 2022). Notably, the proposed approach presents a robust resolution to address several pivotal challenges, including the constraints posed by limited sources available for generating test inputs, the intrinsic limitations related to the diversity of generated test inputs, and the efficacy bottlenecks observed in the testing processes. This stands in marked contrast to the comparatively constrained capabilities of existing methodologies such as CRADLE (Pham et al., 2019) and LEMON (Wang et al., 2020b) in navigating these complex challenges. However, ensuring DL system quality through systematic testing is crucial and as a solution coverage-guided fuzzing, with its neuron selection strategy, has shown promising results. However, current strategies do not utilise neuron output distributions. DLRegion (Tao et al., 2023) resolves this issue by a seed selection strategy based on input classification confidence and region-based neuron selection to activate valuable neurons for improved coverage of internal states.

In addition, DNNs are increasingly utilised to mitigate evolving network attacks such as DDoS, botnets, and ransomware by uncovering concealed patterns in data streams. To effectively address adversarial ML, the fuzzing approach possesses the capability to generate more comprehensive and nuanced representations of hidden states. AdvFuzzer and LocalFuzzer (Qin and Yue, 2022) provide good examples of employing this solution to defend against black-box attacks on ML models by constructing adversarial scenarios and generating a set of adversarial examples locally. It enables the model to effectively mitigate adversarial ML, which poses a significant security threat to networks.

In addition, Liu and Patras introduce NetSentry, an innovative Network Intrusion Detection System (NIDS) based on Bidirectional Asymmetric LSTM (Bi-ALSTM) with manageable computational overhead and resistance to evasion attacks (Liu and Patras, 2022). Using Bi-ALSTM allows bidirectional capture of dynamic information in temporal sequences. The advantage of this model is that unlike traditional LSTM, which captures context up to a specific time step, Bi-LSTM mitigates this limitation with separate forward and backward LSTM units. These units generate hidden states, merged before input into the function.

A summary of fuzzers utilising various techniques for prioritising dynamic information is presented in Table 4. It is evident that seed file generation and grammar constitute the most prevalent outputs of various fuzzing techniques, while DNN-based models prove instrumental in enhancing the effectiveness of fuzzing for autonomous driving camera technology.

5.1.2. *Gradient-guided optimisation*

Another well-known approach to optimisation in DNN fuzzing has encountered various forms of vanishing gradient problems (Hochreiter, 1998) during the training of recurrent neural networks. By utilising gradients and employing smooth neural networks, such as gradient-guided algorithms in fuzzing, researchers were able to achieve performance improvements in the process of generating fuzzing inputs. Some literature has sought to improve gradient-based optimisation through leveraging gradient information to guide the fuzzing process towards regions of interest in the input space.

In She et al. (2019), NEUZZ is proposed using a gradient-guided input generation scheme with a program smoothing technique where the neural network learns to make smooth approximations. In Li et al. (2022a), Li et al. designed a fuzzer called V-Fuzz with a vulnerability-oriented prediction model based on using an attributed control flow graph (ACFG) and optimise the training process with a stochastic gradient descent (SGD) method to find bugs in binary programs within a limited time-frame. In She et al. (2020), She et al. proposed MTFuzz, a multi-task neural network, to tackle the challenges of collecting numerous diverse samples for training the model. It learns a compact embedding of the input space from various training samples across related tasks (e.g., different coverage predictions). This compact embedding directs the mutation process by emphasising high-gradient areas within the embedding. PreFuzz (Wu et al., 2022), a Neural program-smoothing-based fuzzing proposed, improves gradient guidance and mutation effectiveness compared to Neuzz and MTFuzz on a large-scale benchmark suite. It is based on a resource-efficient edge selection mechanism that identifies "sibling" edges enabling a more focused, resource-efficient, and effective fuzzing approach for large-scale fuzzing.

Existing coverage-guided DNN fuzzers mostly fail to offer high crash diversity, quantity and fast fuzzing, crucial for effective adversarial training. As a solution GradFuzz uses new gradient vector coverage for this purpose, uniquely applying gradients to coverage metrics for efficient and effective fuzzing. GradFuzz a new DNN fuzzer identifies diverse errors on MNIST and CIFAR-10 datasets, aiding adversarial

Table 4
Various DNN-Based Exploration and Optimisation Models Based on Prioritising Dynamic Information. C1: Seed File Generation, C2: Message Ready to Send to the SUT, C3: Grammar.

Name/Structure	Technique	Input	Output			SUT
			C1	C2	C3	
NeuFuzz (Wang et al., 2019)	Vulnerability patterns analysis using a 4-layer LSTM	Real world corpus for LAVA-M and different applications	✓			LAVA-M and nine real-world applications (libtiff, binutils, libav, podofo, bento4, libsndfile, audiofile, nasm)
ILF system (He et al., 2019)	Integrating Fully Connected, Gated Recurrent, and Graph Convolutional Networks and Using Markov Decision Making	Effective Seed Integers (SIs) selected by the symbolic expert and through a process of careful selection over a pool of seed integers			✓	Smart contracts including Solidity code adapted from the OpenZeppelin library
Seq2seq-based transition model (Cheng et al., 2019)	Generative built on RNN identifying correlations between seed files and program execution.	PDF files	✓			MuPDF
DeepHunter (Xie et al., 2019)	Recency-aware and Frequency-aware Seed Prioritisation and metamorphic mutation	MNIST, CIFAR-10, ImageNet	✓		✓	Real world app for autonomous driving camera (image) technology
MCM (Park et al., 2022)	Mixed and Constrained Mutation with baseline of DeepHunter	MNIST, STL-10, and ImageNet	✓		✓	Real world app for autonomous driving camera (image) technology
Tzer (Liu et al., 2022)	Pass mutation alongside low-level IR mutation of TVM tensor compiler	Intermediate Representation Files	✓			Tensor compilers, TVM v0.8-dev (9b034d7) with LLVM-12 (also known as DL compilers)
FuzzGuard (Zong et al., 2020)	Prioritising certain datapoints on unbalanced dataset	Generated input from AFLGo's seed corpus	✓		✓	Bento4, Ettercap, GraphicsMagick, ImageMagick, Jasper, Libming, Libtiff, Libxml2, Podofo, Tcpreplay.
FreeFuzz (Wei et al., 2022)	Traces dynamic information from code snippets, developer tests	Adversarial examples, utilising information gathered from code snippets from the library documentation, library developer tests, and deep learning models in the wild, cause crashes		✓		Deep Learning libraries (PyTorch, TensorFlow)
NetSentry (Liu and Patras, 2022)	A NIDS based on Bi-ALSTM with affordable computational overhead	Two datasets: CIC-IDS-2017 CSE-CIC-IDS2018 and Self-collected traffic			✓	Basic ML/DL structures and three Bi-LSTM variants and State-of-the-art anomaly/intrusion detectors
DLRegion (Tao et al., 2023)	Region-based neuron selection strategies	MNIST, CIFAR-10, SVHN under three models LeNet-1 ResNet-20, VGG-16	✓		✓	Real world app for autonomous driving camera (image) technology or medical diagnoses
TCNN-BiGRU model (Wang et al., 2023)	Fusion neural network and dimensionality reduction with dropout and early stopping for overfitting suppression through weighted word vector and Softmax classifier	National Vulnerability Database dataset (NVD)		✓		National Vulnerability Database (NVD) including Multiple SQL injection vulnerabilities in CARE, Cross Site Scripting, and etc

training without sacrificing crash quantity and fuzzing efficiency (Park et al., 2023).

In Cummins et al. (2018), Cummins et al. present DeepSmith which employs a forget gate with a linear activation operation to model the vocabulary distribution over the encoded corpus and is trained by using SGD that accelerates compiler validation by inferring generative models for compiler inputs and uses where Voting Heuristics for Differential Testing uncovers miscompilations.

Nichols et al. utilised GAN models in conjunction with AFL for a novel seed file initialisation technique, termed "Faster Fuzzing". Their objective was to uncover unexplored code paths within a specified time frame (Nichols et al., 2017). The advantage of the GAN model applied in this research is its utilisation of a binary cross-entropy loss function via SGD for training, a 2-layer DNN with ReLU non-linearity as the inner activation, and a tanh output activation function to discover unexplored code paths within a specified time frame.

In the work presented in Hu et al. (2018), a fuzzer named GANFuzz combines GAN and LSTM architectures to assess implementations of industrial network protocols (INPs), such as the Modbus-TCP protocol. The Training and Message-Generating Module (TMGM) in GANFuzz uses TensorFlow and SGD with a batch size of 64 and implements dropout and L2 regularisation to prevent overfitting during DL training and test case generation. After training, it exports the model parameters to checkpoint files for later use in generating test cases. This model employs an LSTM layer with 32 hidden states for the generator and a convolutional, max-pooling, and softmax layer for its discriminator to generate well-structured input data sequences.

To facilitate large-scale vulnerability discovery in OS systems, Grieco et al. introduce a neural network called VDiscover based on SGD algorithm that utilises static and dynamic features, combined with an ML technique for the classification problem and improving imbalanced dataset called random oversampling, that identifies programs with

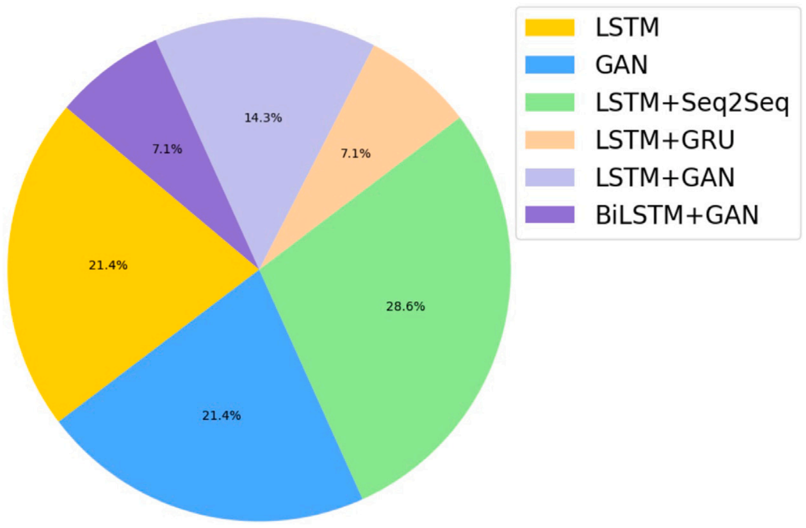

Fig. 4. Frequency of reviewed literature in two subcategories of DNN models.

memory corruptions through a lightweight approach in Grieco et al. (2016).

5.2. Assessment of DNN features

We have compiled a summary of the most prevalent techniques used in DNNs, which are frequently applied in various DNN models. These features are detailed in Tables 4 and 5. Notably, two features stand out as commonly recurring in the literature: prioritising dynamic information and leveraging gradient-guided optimisation techniques. The observation is visually represented in Fig. 4, where it is evident that prioritising dynamic information is more frequently encountered than the use of gradient-guided optimisation. In addition, regarding the summary of fuzzers utilising various techniques for prioritising dynamics in Table 4, it is clear that among the diverse methods of fuzzing, the generation of seed files and the development of grammars are the most commonly observed outcomes.

5.3. Applied architectures in DNN

In this section, we provide an overview of popular DNN architectures i.e. LSTM and GAN as well as hybrid architectures derived from LSTM, Seq2seq and GAN.

5.3.1. Long short-term memory (LSTM) network

LSTM is a type of recurrent neural network (RNN) architecture that addresses the issue of vanishing gradients (Cummins et al., 2018). It improves upon the standard RNN design by introducing a specialised cell for data storage and three gates that regulate the flow of information in and out of the cell. LSTM, due to its inherent memory functions, brings advantages to fuzzing particularly relevant in scenarios where program paths closely resemble statements in natural language, and the determination of code vulnerability is context-dependent. In addition, the advantage of LSTM lies in its adeptness at processing sequential data.

NeuFuzz (Wang et al., 2019) is an early example of this arcitecture working with two layers of bidirectional LSTM (stacked LSTM), each layer with 64 LSTM units. DeepSmith (Cummins et al., 2018) applies a 2-layer Neural-Network LSTM model with 512 nodes per layer to model the vocabulary distribution over the encoded corpus, where Voting Heuristics for Differential Testing uncovers miscompilations. In Zakeri Nasrabadi et al. (2021), Zakeri et al. discuss IUST-DeepFuzz based on the Deep neural language model using deep recurrent neural networks using up to two layers of unidirectional and bidirectional LSTMs with 128 and 256 bits of each layer to learn the structure of complex inputs and to achieve higher coverage by distinguishing between data and metadata, targeting both parsing and rendering stages.

5.3.2. Generative adversarial network

The initial idea of deploying a DNN model based on the GAN architecture involves generating format-undefined data by applying mutation operations to the sequence data, and further improving the data's diversity through augmentation techniques. In their work referenced as (Li et al., 2019), Li and colleagues leverage Wasserstein Generative Adversarial Networks (WGANs) to produce fuzzing data for testing industrial control systems (ICSs), specifically targeting the Modbus-TCP and EtherCAT protocols by sending messages to the system.

SmartSeed is another WGAN based model designed to generate efficient seed files, as introduced by Lyu et al. in their work (Lyu et al., 2019). The system leverages the WGAN to create effective seed files and understand the characteristics of valuable files. WGAN offers two significant advantages: self-learning capabilities and flexibility in model selection. In their study, the authors compared two generative models—Multi-Layer Perception (MLP) and Convolutional Neural Network (CNN)—and favoured MLP due to its efficiency and emphasis on quantitative values. Given MLP's superior performance and quicker training time, SmartSeed utilises this model to generate beneficial seed files, employing the acquired knowledge to effectively train the generative model for fuzzing 12 open-source applications listed in Lyu et al. (2019), which work with input formats such as mp3, bmp, or flv.

Ye et al. introduce RapidFuzz (Ye et al., 2021), a GAN-based fuzzer that efficiently generates synthetic test cases and increases capturing data structure features to fuzz real-world libraries and libraries such as image libraries, GNU, Libxml2 and Libjson. The GAN generates similar but distinct numerical distributions, enhancing the mutation process. An algorithm locates GAN-generated hot points. It succeeds in facilitating faster identification of structural features and Fuzzer's test cases significantly improve American Fuzzy Lop's (AFL) performance in speed, coverage, and map size.

5.3.3. LSTM and Seq2Seq

Another approach studied in Augmented-AFL (Blum et al., 2017) involves mutation at optimal locations within input files such as ELF, PNG, PDF, and XML. This approach includes generating appropriate input grammars and guiding input based on the input probability distribution using standard Bidirectional LSTM models with up to two layers. Additionally, it employs two different chunk sizes, namely 64

Table 5
Various DNN-Based Gradient-Guided Optimisation Techniques. C1: Seed File Generation, C2: Message Ready to Send to the SUT, C3: Grammar.

Name/Structure	Technique/Structure	Input	Output			SUT
			C1	C2	C3	
NEUZZ (She et al., 2019)	Smooth neural network approximation	LAVA-M and DARPA CGC bug datasets	✓			Readelf -a, libjpeg, libxml, mupdf benchmarks
V-Fuzz (Li et al., 2022a)	Attributed Control Flow Graph (ACFG) and SGD	Juliet Test Suite	✓		✓	10 real-word Linux applications (pdftotext, pdffonts, pdftopbm, pdf2svg plus libpoppler, MP3Gain, mpg321, xpstopng, xpstops, xpstops, xpstojpeg, cflow) and three programs of the popular fuzzing benchmark LAVA-M (uniq, base64,who)
MTFuzz (She et al., 2020)	Compact Embedding of the Input Space	LAVA-M	✓			Harfbuzz, strip, size, libjpeg, zlib, readelf, objdump, nm, libxml, mupdf
PreFuzz (Wu et al., 2022)	Resource-efficient edge selection mechanism	A dataset including 28 real-world projects	✓		✓	Bison, xmlwf, mupdf, pngimage, pngfix, pngtest, tcpdump, nasm, tiff2pdf, tiff2ps, tiffdump, tiffinfo, libxml, listaction, listaction_d, libsass, jhead, readelf, nm, strip, size, objdump, libjpeg, harfbuzz, base64, md5sum, uniq, who
GradFuzz (Park et al., 2023)	Gradual guidance to misclassified categories based on the gradient vector coverage	MNIST, CIFAR-10	✓		✓	Autonomous technologies like autonomous cars' apps
VDiscover (Grieco et al., 2016)	SGD model using static and dynamic features and classifications	1039 test cases taken from the Debian Bug Tracker			✓	VDiscovery (Debian programs)
DeepSmith (Cummins et al., 2018)	Modelling the vocabulary distribution over the encoded corpus and is trained by using SGD that accelerates compiler validation	Real-world examples in OpenCL kernels			✓	OpenCL compilers
Faster Fuzzing (Nichols et al., 2017)	Using Adam optimiser and SGD for training and 2-layer DNN with a ReLU non-linearity as the inner activation and a tanh output activation	Seed files from native AFL, Rand, LSTM and GAN	✓			Financial system such as ethkey in the Ethereum code base
GANFuzz (Hu et al., 2018)	Training and MSG Generating Module using a combined model of GAN and LSTM and SGD optimisation	Real world Random Payloads by capturing the context information for Industrial network and control systems	✓		✓	Industrial network protocols(INPs) and Industrial control systems(ICS) such as Modbus-TCP simulators

and 128 bits, and utilises Seq2Seq models with Attention mechanisms (Attn) to effectively handle and generate variable-length inputs.

A generative model-based algorithm called Samplefuzz (Wang et al., 2019) utilises a learned distribution, along with user-defined parameters to sample with Fuzzing. It generates an output sequence by sampling characters from the PDF parser based on the learned model which predicts a character with high confidence and substitutes it with a different character of minimum probability from the distribution. The algorithm is evaluated with unsupervised training using epochs and employing a 2-layer seq2seq LSTM model with 128 hidden states for each layer to create test cases while also generating flaws to exercise error-handling code in (Peleg et al., 2017).

In Zhao et al. (2019), Zhao et al. proposed SeqFuzzer, a fuzzing framework with a technique similar to Blum et al. (2017) that employs DL to learn protocol frame structures. This allows for the analysis of not only the actual data being transmitted but also control information when fuzzing the communication traffic. It uses the Seq2seq technique with three-layer deep LSTMs in the encoder–decoder model for network traffic fuzzing. However, the text does not explicitly mention the size of the chunk for the LSTM model used in the Seq2seq training process.

In the publication (Fan and Chang, 2018), Fan et al. introduce a generative black-box fuzzing approach utilising a seq2seq model. This technique employs a 2-layer LSTM, with each layer comprising 128 hidden states, to acquire a generative input model from protocol traffic. This model enables the generation of new network messages for fuzzing protocols such as WarFTPD and Serv-U.

5.3.4. LSTM and GRU

To generate high-quality test cases for web browser testing utilising HTML tags, a fuzzing approach based on sampling from predictive distributions was proposed. This approach employs a Character-level Recurrent Neural Network (Char-RNNs) language model to discern SQL injection vulnerabilities by generating HTML tags and testing the browser's rendering engine through fuzzing. It deploys up to 6 recurrent layers of LSTM and 2 layers of GRU (Sablotny et al., 2019) All models are trained using a batch size of 512. The LSTM and GRU cell dimensions are standardised at 256 for all trained models, while the layer count ranges from 1 to 6, which are well-suited for scenarios involving variable-length and textual input data.

5.3.5. LSTM and GAN

In accordance with Section 5.1.2, Faster Fuzzing employs both GAN and LSTM models. In this study, GAN and LSTM models are not integrated; rather, they are compared to one another. The GAN architecture utilises a binary cross-entropy loss function optimised through SGD during training, and incorporates novel input and output activation functions within a specified time frame, as described by Nichols et al. (2017), and an LSTM model, trained on AFL-generated seed files, creates 40-character seed files using a 128-wide initial layer, dense layer, and softmax activation. Training uses RMS propagation and categorical cross-entropy loss, with a diversity-adjusting temperature parameter. Notably, the GAN architecture outperforms the LSTM architecture. Conversely, in the work presented by Hu et al. (2018), a fuzzer named GANFuzz combines GAN and LSTM architectures using an

Table 6
Different DNN-based architectures to mitigate fuzzing inefficiencies in software and libraries.

Architecture	Name/Structure	Technique
Standard LSTM	DeepSmith (Cummins et al., 2018)	Two layers LSTMs for modelling the vocabulary distribution over the encoded corpus
	NeuFuzz (Wang et al., 2019)	Vulnerability patterns analysis using a 4-layer LSTM
	IUST-DeepFuzz (Zakeri Nasrabadi et al., 2021)	One/two layers(s) LSTM distinguishing between data and metadata
GAN	WGAN (Li et al., 2019)	Applying mutation operations to the sequence data, and through augmentation techniques
	Smartseed (Lyu et al., 2019)	Two generative models—Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN)—and favoured MLP due to its efficiency
	RapidFuzz (Ye et al., 2021)	Using numerical distributions to locate GAN-generated hot-points
LSTM+Seq2Seq	Augmented-AFL (Blum et al., 2017)	Guiding input based on input probability distribution using Seq2Seq model with one/two-layer(s) LSTM
	Samplefuzz (Peleg et al., 2017)	A generative model-based fuzzing using LSTM (standard and bidirectional) Seq2Seq and Attn
	SeqFuzzer (Zhao et al., 2019)	Seq2seq technique with three-layer deep LSTM in the encoder–decoder model for network traffic fuzzing
	A generative black-box fuzzing (Fan and Chang, 2018)	Seq2Seq with 2-layer LSTM
LSTM+GRU	Char-RNN (Sablotny et al., 2019)	Generating test cases for HTML fuzzing utilising sampling from predictive distributions through a Char-RNN language model with Two layers GRU and Six layers LSTM
LSTM+GAN	Faster fuzzing (Nichols et al., 2017)	3 layer LSTM, Gradient decent (Adam optimiser) for training and 2-layer DNN with a ReLU non-linearity as the inner activation and a tanh output activation
	GANFuzz (Hu et al., 2018)	Training and MSG Generating Module using an LSTM layer and SGD
Bi-ALSTM+GAN	NetSentry (Liu and Patras, 2022)	An innovative NIDS based on Bidirectional Asymmetric LSTM (Bi-ALSTM) with affordable computational overhead

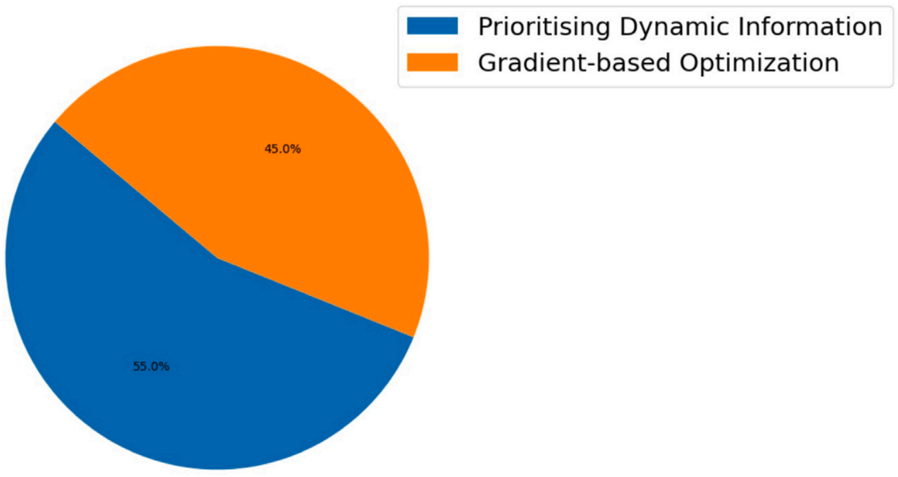

Fig. 5. Frequency of reviewed literature in multiple architecture of DNN models.

LSTM layer with 32 hidden states for the generator and a convolutional, max-pooling, and softmax layer for the discriminator.

5.3.6. *Bi-ALSTM and GAN*

NetSentry which was introduced earlier in Section 5.1, uses a Bidirectional Asymmetric LSTM (Bi-ALSTM) (Liu and Patras, 2022), utilises two distinct LSTM units with 48-bit hidden states, one for forward and one for backward processing. Initially, the hidden states from these units are combined and processed using an activation function. This brings some advantages to evaluating dynamic information bi-directionally across the temporal sequence, aiming to capture a wider range of temporal contexts.

5.4. *Assessment of DNN architectures*

A summary of fuzzers utilising various DNN-based Architectures deployed in fuzzing is presented in Table 6 illustrating the popular architectures employed in DNN-based fuzzing. As depicted in Fig. 4, nearly half of the literature emphasises prioritising dynamic information as the frequently used subcategory in deploying DNN. A frequently used discriminator in DNN-based fuzzing is the softmax layer which offers benefits like multi-class classification, probabilistic output, and fine-grained analysis. These advantages enhance input classification and assessment, improving fuzzing quality for vulnerability identification in various scenarios. Furthermore, the LSTM combined with Seq2Seq was the most applicable hybrid architecture in DNN models presented in Fig. 5 and the LSTM family was the most popular

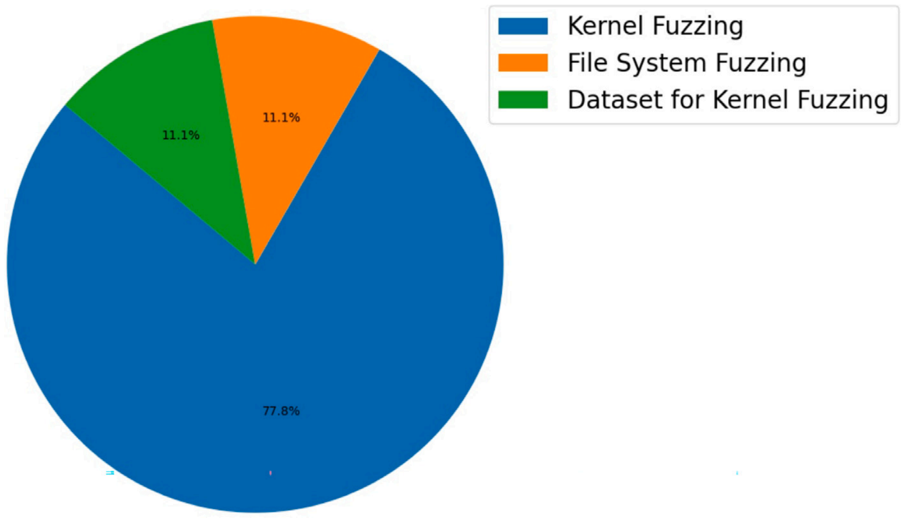

Fig. 6. Frequency of reviewed literature in three subcategory of complex targets.

architecture not only in DNN-based models but also in all four categories of TML, DNN, RL, and DRL when compared to other available architectures.

5.5. *Operating system and file system fuzzing*

In this section, we discuss models that enabled DNN architecture to fuzz complex targets like Kernel and File systems and review an appropriate dataset customised for kernel fuzzing.

5.5.1. *OS fuzzing*

Operating Systems and more specifically kernel components are challenging due to the lack of easily applicable feedback mechanisms, non-determinism, and performance issues after kernel crashes during self-fuzzing. In addition, with the sustained growth of software complexity, finding security vulnerabilities in operating systems has also become an important necessity. Nowadays, OS is shipped with thousands of binary executables, but methodologies and tools for OS-scale program testing within a limited time budget are still missing. Also, OS fuzzing faces challenges. The primary challenge of OS fuzzing lies in the unsuitability of formal checkers for identifying bugs in Operating systems due to their extensive size and complexity, making it unlikely for OS systems to be entirely free of bugs. To address these challenges in Kernel, syzkaller (Anon, 2016) utilises system call data from the Syzlang description to produce sets of system calls that confirm the parameter structure limitations and partial semantic constraints. By repeatedly executing these generated sequences of calls to induce kernel crashes, it identifies kernel bugs.

In pursuit of enhancing large-scale vulnerability detection in operating systems, Grieco et al. introduced a neural network named VDiscover, as previously mentioned in Section 5.1.2. To tackle classification challenges, it employs random oversampling to rectify imbalanced datasets. This approach enables the identification of programs with memory corruptions in a resource-efficient manner, as detailed in their work in Grieco et al. (2016). KAFL (Schumilo et al., 2017) was another approach which was an OS-independent and hardware-assisted coverage-guided kernel fuzzing approach, enabling the fuzzer to analyse the program flow during interrupts. Then, it blacklists non-deterministic basic blocks by re-running inputs and skipping transitions involving blacklisted blocks to enhance fuzzing efficiency and avoid unpredictable paths. The main advantage of this approach is interacting with the targeted OS, using a small user-space component, resulting in minimal performance overhead. OS fuzzer relies heavily on seed system call sequences to test OS kernel vulnerabilities. However, generating effective seeds is challenging due to system call dependencies. Moonshine (Pailoor et al., 2018) introduces a novel strategy for distilling seeds from real-world program call traces, preserving these dependencies. By leveraging lightweight static analysis, MoonShine extends syzkaller and improves code coverage for the Linux kernel compared to traditional hand-coded seed generation methods. FastSyzkaller (Li and Chen, 2019) is another fuzzer that combines the N-Gram model with syzkaller, an open-source fuzzer for Linux, to enhance test case generation and fuzzing efficiency. HEALER (SunHao-0, 2020) employs a similar approach to syzkaller. However, it includes different architectural designs. It relies on an empirical choice table but differs from syzkaller's method of detecting influence relationships between syscalls. These influence relationships are then utilised to guide the generation and mutation of call sequences. TEEFuzzer (Duan et al., 2023b) is another coverage-guided fuzzing framework for trusted execution environments (TEEs). It targets the Open Portable Trusted Execution Environment (OP-TEE) and utilises purpose-built components, including seed generation and mutation modules, to enhance coverage through a contribution-based seed mutation mechanism. It also improves efficiency through a coverage collection module and an automatic bug-reproducing module.

5.5.2. *File system fuzzing*

File systems, crucial to OS functionality, suffer from complexity-induced bugs. Classical stress testing and checkers fall short of detecting these issues. Fuzzing emerges as a practical and effective choice due to its simplicity. Yet, fuzzing file systems face three main challenges: performance degradation, image-dependent operations, and bug reproducibility. JANUS (Xu et al., 2019), a feedback-driven fuzzer, addresses these by exploring the two-dimensional input space of file systems, using image-directed file operations, and relying on a library OS and it outperforms syzkaller in finding and reproducing bugs.

5.5.3. *Dataset for kernel fuzzing*

However, kernel and FS fuzzing still face challenges in suitable datasets. Regarding training Host-based intrusion detection systems (HIDS), system call sequences are important as they are traditionally used for HIDS training and existing datasets are not suitable for OS kernel fuzzing. The first large-scale dataset for anomaly detection in the Linux kernel has been introduced in Duan et al. (2023a) called DongTing. It enables HIDs to effectively identify malicious applications by learning from system events representing normal behaviours.

Table 7
Fuzzing complex targets with DNN-based models. C1: seed file generation, C2: message ready to send to the SUT, C3: grammar.

Targets	Name	Technique	Input	Output			SUT
				C1	C2	C3	
Kernel Fuzzing	syzkaller (Anon, 2016)	Generating System Call Sets from Syzlang Data for Parameter Structure Analysis	System call sequences			✓	Linux kernel
	VDiscover (Grieco et al., 2016)	Stochastic gradient descent (SGD) using static and dynamic features and classifications	1039 test cases taken from the Debian Bug Tracker			✓	VDiscovery (Debian programs)
	KAFL (Schumilo et al., 2017)	Analyses program flow during interrupts and blacklists non-deterministic basic blocks by re-running inputs and skipping transitions	System calls through simple driver that contains a JSON parser based on jsmn			✓	Linux, macOS, and Windows kernel components
	Moonshine (Pailoor et al., 2018)	Distilling seeds preserving dependencies and by leveraging lightweight static analysis	System call sequences	✓			Linux Kernel
	FastSyzkaller (Li and Chen, 2019)	Combines the N-Gram model with syzkaller	System call sequences	✓			Linux Kernel
	HEALER (SunHao-0, 2020)	Detecting influence relationships between syscalls by empirical choice-table	System call sequences	✓		✓	Linux kernel
	TEEFuzzer (Duan et al., 2023b)	Contribution-based seed mutation	System call sequences	✓		✓	OP-TEE's official Xtest and Sanity Test Suite covering most functionalities of the trusted OS
File system	JANUS (Xu et al., 2019)	It relies on a library OS and explores the two-dimensional input space of file systems	Seed image and system calls	✓			A library OS and File systems of upstream Linux kernel (ext4, XFS, Btrfs, F2FS, GFS2, ReiserFS, NTFS)
Dataset	DongTing (Duan et al., 2023a)	Large-scale dataset for anomaly detection in the linux kernel	12116 abnormal system calls including POCs collected from kernel fuzzing tools (e.g., syzbot), and 6850 normal system calls from four Linux test suites	✓			200 different Linux kernel releases

5.6. Assessment of DNN model for complex targets and datasets

A summary of fuzzing complex targets with DNN-based model is presented in Table 7. As illustrated in this table, kernel fuzzing receives significant attention, particularly in Operating Systems and other intricate targets like File Systems. Among the diverse methods employed in fuzzing, the generation of seed files and the formulation of grammars emerge as the most prevalent outcomes.

Approximately four-fifths of the papers, totalling 77.8%, of the publications are dedicated to this specific subcategory as depicted in Fig. 6. Additionally, 11.1% of the research endeavours focus on developing high-quality datasets for Kernel fuzzing, contributing significantly to the advancement of Kernel fuzzing practices.

5.7. Persistent challenges in DNN

DNNs have challenges adjusting to new environments and may overfit, which affects their performance in unexpected situations. In contrast to RL's flexibility in dynamic fuzzing environments, their static nature demands periodic retraining for changing situations.

The field of fuzzing research continues to explore new opportunities. The advancement of unsupervised fuzzing methodologies stands to derive significant benefits from the remarkable strides achieved in the field of artificial intelligence (AI). For instance, various techniques such as RL and LSTM combined models, as well as the RL-Vulnerability-guided model, are also applied in fuzzing to guide the exploration of the input space by learning improved strategies within the context of fuzzing. We discuss these techniques in more detail in Section 6.

6. RL techniques for fuzzing

In this section, we will delve into the promising domain of RL and explore how it can effectively overcome challenges posed by traditional DNNs in a multitude of real-world applications. These challenges encompass complex problems involving sequential decision-making, exploration, addressing sparse rewards, adapting to dynamic environments, and numerous other scenarios where the conventional DNN framework may not provide the most optimal solutions. We divide these models into two categories: fuzzing general models of fuzzing targets with lower dependencies (see Section 6.1), and fuzzing complex targets with higher dependencies, such as Kernel fuzzing (see Section 6.2). By the end of this section, readers will gain valuable insights into the versatility and adaptability of RL, which enable it to excel in diverse settings by learning from interactions with the environment.

6.1. General RL models

We conduct a comprehensive survey of various RL models and techniques, highlighting their capabilities and advantages in addressing DNN challenges. The trend in combining RL with fuzzing has an early background. A notable instance of early work in this area dates back to 2010 (Becker et al., 2010) when Becker et al. presented an algorithm based on RL. It involves several steps, including finite-state machine modelling, fuzzing strategies, and RL, enabling the framework to independently acquire knowledge about optimal fuzzing techniques and automate the process of testing IPv6 for stability and reliability.

RL is also paired with LSTM architecture to handle sequential data in RL contexts. While LSTMs are primarily associated with DL, they can enhance RL systems by processing sequences. For instance, in a study by Paduraru et al. (2021), they apply RiverFuzzRL, which offers a ready-to-use RL implementation and customisation options for diverse test targets in binary-level fuzzing. They employ an LSTM architecture inspired by PySE (Koo et al., 2019) to represent fixed-size paths and the longest execution paths of binary blocks, optimising the fuzzing process. Scott et al. introduce “BanditFuzz” (Scott et al., 2021), which leverages multi-agent RL guidance to enhance the performance of SMT solvers. The core idea revolves around training a single agent explorer or a group of agent explorers to maximise a reward signal indicating the effectiveness of the generated inputs, particularly in terms of achieving high code coverage or identifying vulnerabilities. In their 2022 article, Su et al. introduce Reinforcement Learning-Guided Fuzzing (RLF), a pioneering vulnerability-guided fuzzer that leverages DRL. RLF conceptualises the fuzzing process as an MDP and incorporates a reward system that takes into account both vulnerability identification and code coverage improvement (Su et al., 2023). Binosi et al. have introduced Rainfuzz, a dynamic analysis technique that involves the generation of heat-maps from a trained Feed Forward Neural Network (FFNN). These heat-maps are used to pinpoint specific bytes during input mutation, facilitating the repeated execution of a program with diverse inputs to trigger abnormal behaviour. The fundamental principle underlying Rainfuzz is to treat the selection of mutation positions as an RL problem, as discussed in their work (Binosi et al., 2023).

Jha et al. introduce BertRLFuzzer, a novel approach that leverages the Bidirectional Encoder Representations from Transformers (BERT) model as a RL agent. This technique employs semi-supervised learning with BERT to autonomously grasp the application’s grammar from a pre-trained model. Subsequently, it utilises this learned knowledge to launch attacks on the target application, all without the need for explicit user input. You can find more details in their research paper (Jha et al., 2023). In Patil and Kanade (2018), Patil and Kanade adapt AFL’s heuristics by framing them as a contextual bandit problem. They employ a combination of LSTM encoding, FCNN extraction, and the policy gradient method. This novel approach allows them to dynamically adjust the multiplier for fuzzing iterations, which is based on the characteristics of the test cases. Consequently, their algorithm continually refines the policy for generating intriguing test cases. Wang et al. introduce AFL++-HIER as a solution to the complex task of efficiently scheduling a larger set of seeds. This was achieved by incorporating fine-grained coverage metrics, implementing a multi-level coverage metric system, and employing a RL-based hierarchical scheduler, as detailed in their work cited as Wang et al. (2021a). In Choi et al. (2023b), Choi et al. introduce an innovative seed-scheduling method within CGF. This method leverages RL to enhance crash detection performance, irrespective of the specific characteristics of the target program. It achieves this by intelligently determining the optimal sequence for test case execution and efficiently allocating resources for seed mutation.

6.2. *Kernel fuzzing*

RL incorporates exploratory kernel fuzzing, which while sharing some similarities with DNNs (see Section 5.5), boasts distinct characteristics. Specifically, in the context of complex targets with interdependencies such as Kernel and Operating System Fuzzing, RL leverages RL models. To highlight this distinction, we refer to work by Wang et al. (2021b). They have introduced SYZVEGAS, a dynamic fuzzer to address the challenge of parameter optimisation by autonomously adapting task and seed selection via a reward assessment model, employing multi-armed bandit algorithms.

Furthermore, Huang et al. introduce Anon (2016), which combines multi-armed bandits with basic block weight computed through static analysis. This integration allows for dynamic task and seed selection based on a machine model, further enhancing code coverage in syzkaller and Syzvegas (Huang et al., 2022b).

6.3. *Assessment of RL models*

As demonstrated in Table 9, kernel fuzzing emerged as the preeminent target among complex systems, notably fuzzed through the application of RL adaptations and Furthermore, Vulnerability-guided fuzzing (see Table 8), supported by two prominent literature sources, emerged as the dominant approach over other subcategories.

6.4. *Challenges in RL and other previous models: A comparative exploration*

RL techniques significantly enhance the fuzzing process, boosting both its effectiveness and efficiency. They empower the fuzzer to progressively refine its behaviour by utilising more precise predictors and discriminators. This, in turn, leads to notable improvements in code coverage, bug detection, and the overall speed of the process, however, it lacks the ability of complex decisions. DRL systems, designed for complex decision-making tasks, learn behaviour by interacting with environments. This dynamic development process differs significantly from traditional software where logic is explicitly defined. DRL’s absence of explicit logic poses new testing challenges. Traditional fuzzing generates random data, while DRL-focused fuzzing aims to explore various state spaces systematically, addressing these unique DRL testing needs Li et al. (2022d). Additionally, DRL systems differ from DL as they learn through trial and error rather than fixed datasets, requiring thorough environment understanding to prevent biased models (Li et al., 2022d). We discuss further it in the Section 7.

7. DRL techniques

Addressing the challenge posed by the DL approach involves tackling the requirement for a substantial volume of training data, a topic we will explore further in the DRL. Specifically, it discusses the problem of frequent large rewards being given when two states are quite different, which can lead to excessive exploration and jumping between states, even if they have been explored extensively.

Also, the challenge that motivates the utilisation of DRL in the realm of fuzzing is the need to manage the complex, dynamic, and uncertain nature of fuzzing where traditional RL techniques may find it difficult to adapt to the constantly evolving nature of fuzzing settings, which makes them less efficient at finding vulnerabilities. DRL, on the other hand, enables the fuzzer to learn complex policies and is better able to tackle the difficulties brought on by fuzzing scenarios.

In fact, RL faces the issue of optimising an agent’s behaviour in a dynamic system to maximise cumulative rewards and trying to learn how to achieve the highest cumulative reward over time (Bottinger et al., 2018; Pan et al., 2020).

Also, it is crucial to emphasise the significance of tackling challenges related to Syntax and Semantics Validity. In this regard, DRL-based solution stands out as a more adaptable approach when compared to DNNs. For instance, when it comes to fuzzing compilers, a unique set of difficulties arises, encompassing the requirement for valid inputs, the necessity for input diversity, the detection of unusual behaviours, code coverage analysis, performance considerations, handling extensive codebases, and the management of false positives (as discussed in Li et al. (2022c)). To effectively address these compiler-specific challenges, specialised techniques and tools tailored to RL-based approaches prove more advantageous than those centred around DNNs. These techniques often involve the application of a series of mutations, which can potentially increase the likelihood of generating a wider array of input programs. Consequently, this enhances the code coverage of compilers by optimising the accumulated rewards throughout an episode, which terminates at a specific step.

Additionally, there is an issue with exploring an invalid sample space (Gong et al., 2022) and Significant resource consumption during fuzzing (Liang and Xiao, 2022) particularly when seeking to identify deep-seated software defects in hybrid fuzzing (Jeon and Moon,

Table 8
Different RL features. C1: seed file generation, C2: message ready to send to the SUT, C3: grammar.

Features	Name	Technique	Input	Output			SUT
				C1	C2	C3	
Leveraging reinforcement-based fuzzing for IPV6	SARSA Algorithm (Becker et al., 2010)	FSM modelling, fuzzing strategies, and RL.	Applying various strategies and decomposing message types within an FSM machine.		✓	✓	Neighbour Discovery Protocol in IPv6
RL and LSTM combined	RiverFuzzRL (Paduraru et al., 2021) PySE (Koo et al., 2019)	Both process sequential in RL settings using LSTM architecture	XML tags switch inside the XML input buffer			✓	Python benchmark programs (Biopython parewise2, GNU grep, insertion sort, etc.)
RL-Vulnerability-guided fuzzing	BanditFuzz (Scott et al., 2021)	Multi-agent RL-guided fuzzing using a satisfiability Modulo Theories (SMT) solvers.	1,700 syntactically unique inputs generated by BanditFuzz, focusing on floating-point and string SMT theories			✓	State-of-the-art SMT solvers Z3, CVC4, Colibri, MathSAT, and Z3str3
	RLF introduced by Su (Su et al., 2023)	A MDP using a reward system and an LSTM layer for training	Two datasets of smart contracts and seed pools similar to ILF fuzzer			✓	Real world smart contracts and oracles for vulnerabilities
A dynamic analysis of mutation positions based on heatmap	Rainfuzz (Binosi et al., 2023)	The selection of mutation as a reinforcement-learning problem using MDP model	JPEG	✓			libjpeg-turbo: file as a PUT, a binary taken from FuzzBench
Semi-Supervised Learning	BertRLFuzzer (Jha et al., 2023)	Using a Bidirectional Encoder Representations from Transformers model as an RL agent to learn the grammar from a pre-trained model	Real world attack vectors of SQL injection (SQLi), Cross-site Scripting (XSS), and Cross-Site Request Forgery (CSRF) and etc		✓	✓	A variety of real world benchmark including 9 victim websites with up to 16K lines of code, PHP , MySQL, Java
AFL with contextual bandits	AFL CB (Patil and Kanade, 2018)	LSTM encoding, FCNN extraction and the policy gradient method, 100 recurrent units for the LSTM, the default tanh activation function and the state size is of 128 bytes	Real world corpus for applications	✓			A list of targets including binutils, tcpdump, mpg, libpng, gif2png, libxml, addr2line, cxxfilt, elfedit, nm, objcopy, objdump, readelf, size, strings, strip-new, gif2png, libxml2, libpng, mpg321, tcpdump
A reinforcement-learning-based hierarchical scheduler	AFL++-HIER (Wang et al., 2021a)	A multi-level coverage metric and a reinforcement-learning-based hierarchical scheduler	Real world corpus for applications (CGC DARPA dataset)	✓			A prototype on DARPA CGC
Seed scheduling with reinforcement-learning method	Reinforcement Seed Scheduling Algorithm in CGF (Choi et al., 2023b)	Determine the optimal order of test case execution and resource allocation for seed mutation	Binutuls, LAVA-M	✓			nm,objdump,cxxfilt, as, ld, readelf, size, string, uniq, who, md5sum, base64

Table 9
RL for complex targets. C1: seed file generation, C2: message ready to send to the SUT, C3: grammar.

Approach	Name	Technique	Input	Output			SUT
				C1	C2	C3	
Kernel Fuzzing	Syzvegas (Wang et al., 2021b)	Parameter optimisation by reward assessment of seed selection and tasks through multi-armed-bandit algorithms	561 seed programs and a total of 8127 system call sequences	✓		✓	Linux Kernel
	Syzballer (Huang et al., 2022b)	Multi-armed bandits with basic block weight	System call sequences			✓	Linux Kernel

2022). The challenges revolve around the complex and uncertain nature of the environment, the complexity of the agent's model, and the non-deterministic behaviour of both the agent and the environment particularly in the game testing, all of which make it difficult to evaluate the performance of trained agents effectively (Tappler et al., 2022). The challenges pertain to the significant speed gap between RL-based neural network evaluation and the rapid execution of fuzzing actions (Drozd and Wagner, 2018). As a result, traditional fuzzing, DL-based fuzzing and RL-based fuzzing face challenges in finding the optimised solution when compared to DRL-based testing.

7.1. DRL models

DRL has emerged as a transformative technology, offering both automation and adaptability within the realm of software testing. This technology's distinctiveness becomes evident when applied to critical areas, such as test case prioritisation within Continuous Integration (CI) pipelines and game testing. In the study by Nouwou Mindom et al. (2023) various DRL frameworks in software testing, including fuzzing, were comprehensively examined. Their findings unequivocally demonstrate the efficacy of DRL frameworks, with select frameworks surpassing the performance of contemporary approaches. However, it is worth noting that this paper primarily focused on surveying the landscape of software testing in CI and game testing, with an implicit review of multiple software testing frameworks and not specifically on fuzzing techniques for general random testing concepts applied to various targets. As a result, further empirical evaluations on benchmark problems are warranted when selecting DRL frameworks for fuzzing to ensure that algorithmic performance aligns seamlessly with specific requirements and objectives.

In this section, our main focus shifts from the aforementioned survey paper. We survey different DRL techniques applied to fuzzing, including Curiosity-driven RL, Deep Q-learning, and the integration of DNNs with RL frameworks, encompassing game testing as well.

7.1.1. Curiosity-driven learning

Curiosity-driven Learning RL is a distinctive approach of DRL started by “Q-testing” in the domain of fuzzing. Q-testing is an approach based on Q-learning in automated software testing that combines random and model-based techniques introduced in Pan et al. (2020). It paired Q-learning with a curiosity-driven strategy employing principles from RL or DRL to guide the exploration of unfamiliar software functionalities. It is important to note that Q-testing is not synonymous with Q-learning; instead, it adopts the concept of Q-values and the essence of RL to formulate a testing strategy. In other words, Q-testing entails the process of deciding which test cases are generated based on Q-values, which stand for the speculated effectiveness or usefulness of a specific task in a given programme state through Curiosity-driven RL.

As a practical example of Q-testing combined with a curiosity-driven approach, Pan et al. introduced LSTM-based Q-testing in their research (Pan et al., 2020), treating Android testing as an MDP. This approach, grounded in RL, combines random and model-based techniques for automated testing of Android apps using Siamese LSTM. It leverages a state comparison module to improve the exploration of unfamiliar app functionalities. Zheng et al. (2021) also introduced an MDP model based on curiosity-driven RL called WebExplor to create effective test cases and to build an automaton for guidance, enhancing testing efficiency. Extensive evaluations show its superiority in failure detection, code coverage, and efficiency.

7.1.2. Deep Q-learning

The Deep Q-learning method involves storing experiences, comprising state, action, reward, and next state, in a buffer. During training, random batches are sampled from this buffer to disrupt the correlation between consecutive experiences. Q-learning, originally introduced by Watkins in Watkins (1989) and Watkins and Dayan (1992), has seen significant advancements through its integration with DNN-based fuzzing techniques, a concept outlined in Bottinger et al. (2018). This combination has proven to be exceptionally effective when confronted with the complex challenges posed by vast state spaces in the domain of fuzzing with complex targets. One of the most notable successes in this area was achieved by Bottinger et al. (2018), who constructed a Q-network RL model based on MDP, leveraging deep Q-learning to optimise fuzzing action selection, ultimately resulting in superior policy mastery and remarkable performance.

In a separate endeavour described in Li et al. (2022c), Xiaoting Li and colleagues introduced an innovative model called FuzzBoost, which approaches compiler fuzzing as a RL problem. This framework incorporates deep Q-learning, facilitating multi-step code mutations and utilising a reward policy grounded in testing coverage, all underpinned by deep Q-learning principles (Li et al., 2022c).

However, DRL encompasses a broader class of algorithms beyond deep Q-learning. DRL refers to the combination of DL techniques, such as DNN, with RL methods to solve complex tasks which is explained in the following.

7.1.3. Integration of DNN with RL framework

In the realm of integrating DNN with RL frameworks, several innovative approaches have been proposed, each with a unique focus and contribution. The integration of DNN with RL frameworks encompasses various methods aimed at directly mapping the state to the action probabilities.

In Li et al. (2022d), Li et al. introduce Agentfuzz, a novel testing framework designed to generate diverse test cases to identify failures in DRL systems. This framework leverages gradient-guided mutators, providing a unique approach to test case diversity and failure detection. Gong et al. present DRLFCfuzzer, a methodology outlined in Gong et al. (2022), which addresses data boundary segmentation and data block selection using DRL. This approach aims to enhance fuzzing efficiency and reduce the exploration of invalid sample spaces, thereby optimising the testing process. Liang and Xiao (2022) propose a directed fuzzing approach based on DRL. Their method employs a DRL network to optimise the selection of test samples and utilises program instrumentation for determining execution paths and distances from the target. This targeted approach improves the effectiveness of the fuzzing process. Jeon and Moon introduce “Dr. PathFinder” in their work (Jeon and Moon, 2022), a concolic execution engine that combines DRL with a hybrid fuzzing approach employing LSTM. This combination of techniques addresses limitations associated with traditional fuzzing methods, especially in bug discovery scenarios where prior knowledge of the target program is lacking. Dr.PathFinder combines “deeper path first concolic execution” with mutational coverage-based fuzzing, incorporating Q-learning to progressively minimise gradient differences, ultimately enhancing the fuzzing process. In Li et al. (2022b), Li et al. introduce ALPHAPROG, which employs multiple reward functions to guide the generation of valid programs for compiler testing. This model balances validity and diversity, increasing the likelihood of identifying vulnerabilities and flaws in the compiler by generating programs that compile successfully while covering a wide range of language patterns.

Tappler et al. address the safety and performance evaluation of DRL agents using a search-based testing framework, as described in Tappler et al. (2022). Their framework combines backtracking-based depth-first search for safety testing and genetic algorithm-based fuzzing for performance testing, creating a tailored testing approach to ensure the safety, robustness, and performance of RL agents in various environments and tasks. In their work (Drozd and Wagner, 2018), Drozd and Wagner introduced Fuzzergym by integrating OpenAI Gym with libFuzzer, connecting LLVM Sanitizers’ program monitors with a neural network. They employ asynchronous buffers to facilitate coordination between the fuzzing process and ML, using RL within an asynchronous architecture for intelligently selecting mutations in software testing.

Each of these contributions represents a unique approach to the integration of DNN with RL frameworks, focusing on diverse aspects of testing, efficiency optimisation, and targeted fuzzing. Together, they offer a comprehensive landscape of innovative techniques and methodologies in this evolving field.

7.2. Assessment of DRL models

A variety of DRL models, including Curiosity-driven RL, Deep Q-learning, and Integration of DNN with RL Framework, have been extensively investigated in the literature, as documented in Table 10. Notably, integration of DNN with RL Framework emerges as the most prevalent technique as depicted in Fig. 7, constituting 63.6% of the explored approaches. This dominance underscores its paramount significance within this domain. Moreover, seed file generation and grammar formulation consistently emerge as the predominant anticipated outputs of different fuzzing techniques.

Table 10
Different deep RL approaches to address fuzzing inefficiencies in software, kernel, and other libraries.

Subcategories	Name	Technique	Input	Output			SUT
				C1	C2	C3	
Curiosity Driven RL	Q-testing on Android (Pan et al., 2020)	Applying Q-testing based on Siamese LSTM with two single-layer networks with 100 hidden neurons using a curiosity-driven strategy	Real world corpus			✓	50 open-source real world applications
	WebExplor (Zheng et al., 2021)	a MDP model based on curiosity-driven RL	Real world corpus	✓		✓	on six real-world projects and 50 web applications
Deep Q-learning	Q-net RL (Bottinger et al., 2018)	Using MDP based on deep Q-learning that learns to choose highly rewarded fuzzing actions	A 168 kByte seed file with 101 PDF objects	✓			PDF parser in the Edge browser, Linux command line converters
	FuzzBoost (Li et al., 2022c)	Automatic code synthesis framework to resolve compiler fuzzing as a RL problem based on multi-step code mutations, MDP and a reward policy	OpenSSL.1.0.1u OpenSSL.1.0.1f busybox.1.21.stable	✓		✓	GCC test suites
Integration of DNN with RL Framework	Agentfuzz (Li et al., 2022d)	Gradient-based mutators	Initial states of RL system as test cases	✓			Deep reinforcement learning system
	DRLFCfuzzer (Gong et al., 2022)	Segmenting data boundaries and selecting data blocks	Jpg, png, jpg, bmp, gz, png	✓		✓	Fuzzer-Test-Suite (guetzil, libpng, libjpeg), Real-world apps(CImg, gzip, pngquant)
	RLF introduced by Ling (Liang and Xiao, 2022)	Directed fuzzing using program instrumentation for execution path and distance, and RL which is modelled by MDP	LAVA-M dataset and PNG	✓			uniq, who, md5sum, and base64, GNU Binutils, LibPNG
	Dr.PathFinder (Jeon and Moon, 2022)	A concolic execution engine using RL with a hybrid fuzzing approach and Q-network architecture and standard LSTM	Symbolic inputs generated by SMT solver	✓			CB-multios dataset (migrated set of CGC binaries to Linux, Windows, and OS X environments)
	ALPHAPROG (Li et al., 2022b)	Sequential mutation rewards using a LSTM layer with 128 neurons	Samples for BF compiler	✓			BF compiler
	Backtracking-based DFS (Tappler et al., 2022)	Modeled by MDP Combining backtracking-based DFS and genetic-algorithm-based fuzzing	Safety test-suites based on the backtracking states of the search, called boundary states			✓	Arcade Learning Environment, OpenAI Gym, Deepmind Control Suite: SafetyGym
	Fuzzergym (Drozd and Wagner, 2018)	Employing OpenAI in LibFuzzer using LSTM with w/64 units, asynchronous buffers and architecture and intelligently selecting mutations through Partially Observable Markov Decision Process (POMDP)	Real world corpus	✓			Libjpeg, libpng, boringssl, re2, sqllite

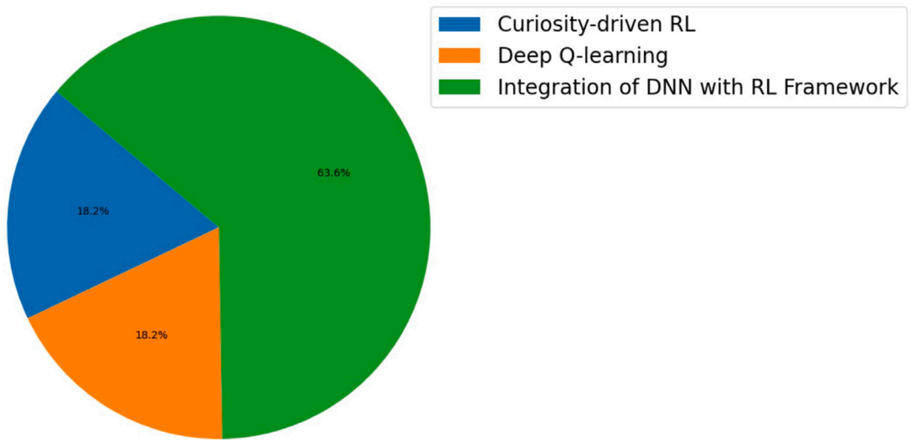

Fig. 7. Frequency of literature in three subcategory of DRL models.

8. Discussion and future work

After conducting a thorough literature review, it is evident that the choice of ML-based fuzzing categories, such as TML, DNN, RL, and DRL, hinges upon the specific target and the scale of the fuzzing task. In other words, the applicability of each category varies depending on the unique characteristics and requirements of the fuzzing scenario. This insight underscores the need for a nuanced approach in selecting the

Table 11
Frequency of mathematical framework or techniques in research literature.

Mathematical frameworks or techniques	Frequently utilised methods across the literature	List of References
Gradient-based optimisation or Mutation	13	Raychev et al. (2015) She et al. (2019) Li et al. (2022a) She et al. (2020) Wu et al. (2022) Park et al. (2023) Cummins et al. (2018) Nichols et al. (2017) Hu et al. (2018) Grieco et al. (2016) Patil and Kanade (2018) Li et al. (2022d) Jeon and Moon (2022)
Multi-armed bandits	5	Karamcheti et al. (2018) Scott et al. (2021) Patil and Kanade (2018) Wang et al. (2021b) Huang et al. (2022b)
Markov-Chain Decision	11	Sun et al. (2018), He et al. (2019), Su et al. (2023), Binosi et al. (2023), Bottinger et al. (2018) Pan et al. (2020), Zheng et al. (2021), Li et al. (2022c), Liang and Xiao (2022), Tappler et al. (2022), Drozd and Wagner (2018)

Table 12
Frequently utilised architectures in optimisation techniques and neural network.

Neural network architecture	Frequently utilised architecture across the literature	List of References
LSTM Family	18	Wang et al. (2019) Cummins et al. (2018) Zakeri Nasrabadi et al. (2021) Peleg et al. (2017) Zhao et al. (2019) Blum et al. (2017) Fan and Chang (2018) Sablotny et al. (2019) Nichols et al. (2017) Hu et al. (2018) Liu and Patras (2022) Paduraru et al. (2021) Koo et al. (2019) Su et al. (2023) Patil and Kanade (2018) Pan et al. (2020) Jeon and Moon (2022), Drozd and Wagner (2018)
GAN Family	6	Li et al. (2019) Lyu et al. (2019) Ye et al. (2021) Nichols et al. (2017) Hu et al. (2018) Liu and Patras (2022)
Seq2seq	6	Cheng et al. (2019) Blum et al. (2017) Wang et al. (2019) Zhao et al. (2019) Blum et al. (2017) Fan and Chang (2018)

most suitable ML-based fuzzing technique, tailored to the particular context at hand.

In our review, we also found that different TML models can enhance traditional fuzzing approaches by providing data-driven decision-making capabilities. Specifically, the Feature-Based Protocol Model improves traditional fuzzing by evaluating the characteristics and features of the target program's protocol. However, it relies on a predefined set of features representing key aspects of the protocol. As a solution to this limitation, predictive and fitness models aim to establish an optimal relationship between input and output for the target program. This does come with further challenges since selecting the right features for predictive functions can be a time-consuming and domain-specific task, making it difficult to scale up for larger and more complex software systems. In such cases, seed generation with a long input correlation tracing model may be more feasible since they can prioritise inputs that enhance code coverage, particularly for closed-source targets. While TML models effectively identify promising inputs and enable the assessment of the quality of generated test cases, they fall short in comparison to DNN models when it comes to exploration and optimisation.

DNN offer much promise for developing fuzzing capabilities, since the interconnected nodes and multiple layers models can automatically extract hierarchical representations from raw data, enabling efficient handling of complex predictive and discriminative tasks. DNNs are not without their challenges, such as the risk of over-fitting, which can impact their performance in unexpected situations. To address the challenges posed by DNN models, various DNN-based optimisation techniques, including neural network architectures and RL and DRL techniques, offer improved strategies for dealing with complex and high-dimensional data, such as Kernel fuzzing. In our survey, we found that while DNN techniques have been applied to fuzzing various complex targets, such as Operating Systems, there is a notable absence of exploration regarding the application of RL techniques to fuzz File Systems or related areas. Moreover, there is a notable scarcity of work focusing on the development of high-quality datasets tailored for RL-based fuzzing methodologies. RL and DRL techniques provide greater flexibility in dynamic fuzzing environments, and through continual learning paradigms they can better adapt to changing circumstances within the environment. We believe that there is still great potential for RL and DRL methods to be explored further through future research.

8.1. Assessment of mathematical framework and different architectures

A summary of the frequency of usage of various optimisation techniques, including gradient-guided optimisation, the Multi-armed bandit, and Markov-Chain Decision across the four categories are presented in Table 11. Gradient-based optimisation or mutation is the most frequently used optimisation technique in the fuzzing process, combined with all categories, including TML, DNN, RL, and DRL, and the number of literature references is 13. The reason for its frequent use may be that it is the easiest method to implement and adapt to diverse scenarios.

Table 12 lists architectures frequently utilised optimisation techniques and neural network across the literature. Frequently used neural network architectures across the literature include LSTM, which was applied in DNN, RL, and DRL 18 times, and GAN, a specific type of neural network architecture used for data generation tasks, which was mentioned 6 times. The reason for their frequent utilisation may be their effectiveness in modelling complex data patterns and generating realistic data samples. Seq2seq was mentioned in the literature 6 times. The reason for its frequent use might be its effectiveness in handling Seq2Seq tasks, such as machine translation and text summarisation.

Multi-armed bandit was referenced 5 times in the literature. The reason for its repeated appearance might be its suitability for solving problems involving decision-making under uncertainty, such as resource allocation and online advertising optimisation. Hence, the popularity of employing LSTM combined with Seq2Seq in sequential data processing strategies is well-founded.

The specific requirements of the fuzzing task and the characteristics of the input data will determine the selection of the LSTM variant and the number of layers to be utilised in the fuzzing process. Deep architectural designs may not always be necessary for effective fuzzing; simpler LSTM variations, such as standard LSTM or bidirectional LSTM, can serve as viable alternatives.

Table 13
Number of LSTM layers and LSTM models deployed in different literature.

Fuzzer	Number of layers	LSTM model
NeuFuzz (Wang et al., 2019)	4	Bidirectional LSTM
NetSentry (Liu and Patras, 2022)	1	Bi-ALSTM with 64-bit initial input 48 hidden states
Faster fuzzing (Nichols et al., 2017)	1	Standard LSTM with “128-wide initial layer” (it is not explicitly stated)
Samplefuzz (Peleg et al., 2017)	2	Seq2seq LSTM model with 128 hidden states
Augmented-AFL (Blum et al., 2017)	1, 2	Standard LSTM and Bi-LSTM with 64-, 128-bit Chunk Seq2Seq and Seq2Seq+Attn models
A generative blackbox fuzzing (Fan and Chang, 2018)	2	LSTM with 2 hidden (no specific name mentioned for fuzzer layers), and each layer consists of 128 hidden states combined with seq2seq
DeepSmith (Cummins et al., 2018)	2	Standard LSTM network of 512 nodes per layer
Char-RNN (Sablotny et al., 2019)	2, 6	Two-layer GRU and one to six layer standard LSTM where training batch size is 512 and internal size of the LSTM and GRU cells are 256
IUST-DeepFuzz (Zakeri Nasrabadi et al., 2021)	1	Unidirectional LSTM with 128 units in each layer
	2	Unidirectional LSTM with 128 bits in each layer
	2	Unidirectional LSTM with 256 bits in each layer
	2	Bidirectional LSTM with 128 bits in each layer
SeqFuzzer (Zhao et al., 2019)	3	Deep LSTM with seq2seq model. The LSTM size is not explicitly specified but it works with standard Ethernet packets Protocol messages (4 states machine) data are exported as 8-byte hex C arrays file (total 64 bits)
GANFuzz (Hu et al., 2018)	1	Standard LSTM with 32 hidden states
AFL-CB (Patil and Kanade, 2018)	1	Standard LSTM with 100 recurrent units state is a stream of 128 bytes
Dr.PathFinder (Jeon and Moon, 2022)	2	Standard LSTM (not specified)
RiverFuzzRL (Paduraru et al., 2021)	1	Standard LSTM (not specified)
PySE (Koo et al., 2019)	1	Standard LSTM (not specified)
RLF introduced by SU (Su et al., 2023)	1	Standard LSTM (not specified)
Curiosity Driven Testing (Pan et al., 2020)	1	Siamese LSTM (not specified)
Fuzzergym (Drozd and Wagner, 2018)	1	Standard LSTM with 64 units

Table 13 presents information on the number of LSTM layers and LSTM models deployed in different literature. This table aids readers in assessing the problem's complexity, and the availability of training data, and provides a reasonable estimate of the computational resources required for executing a fuzzing campaign based on the number of LSTM layers and their models.

8.2. Limitations

Although this structured presentation enriches our understanding of the prevalence of these techniques and architectures in the literature, offering a comprehensive overview of their usage and significance, it is imperative to acknowledge the persistent, unresolved challenges within the realm of fuzzing that warrant consideration in future research. We discuss potential topics which can be considered for future work as following:

- **Improving Mathematical models:** Fuzzing has exhibited promising potential through existing mathematical models however, there remains ample opportunity for additional research and refinement. For instance, within the context of fuzzing, leveraging a Markov chain presents an avenue to model the interrelation among distinct states of input data. In this representation, each state embodies a specific structural composition or pattern of the input data. A critical area for advancement involves enhancing the Markov chain to adeptly depict event sequences, wherein the probability of each event is contingent solely upon the state achieved in the preceding event.
- **Coverage Improvement:** Fuzzing aims to explore as much of the program's code and behaviour as possible. Future work should focus on enhancing better methods of leveraging features, predictive/fitness models and more reliable coverage metrics and techniques to ensure thorough code exploration, including complex and less-accessible parts of the code.
- **Targeting Specific Vulnerabilities Versus robustness of fuzzing against input Variability:** Developing specialised fuzzing strategies and techniques tailored to identify specific vulnerabilities holds significant promise. Concurrently, ensuring the resilience of fuzzing tools against a broad spectrum of input variability—encompassing malformed, unexpected, or intentionally adversarial inputs—is paramount for their efficacy in practical, real-world contexts. Fuzzing methodologies necessitate customisation to effectively discern distinct vulnerability categories, such as memory leaks and race conditions. However, they must also possess robustness to withstand stress testing and unanticipated variations in input. Keeping the balance between these two factors should not cause adversarial scenarios where malicious actors attempt to bypass or deceive fuzzing systems' robustness against unanticipated variations in input. Achieving a balance among the factors mentioned earlier ensures that fuzzing techniques exhibit greater resilience against intentional evasion endeavours.
- **Handling Input Complexity and Non-Functional Properties:** Many real-world applications and systems accept complex and structured inputs (e.g., file formats, and network protocols). Developing fuzzing techniques that handle and mutate such complex inputs effectively is a significant challenge. Also, beyond functional correctness to handle input complexity, considering non-functional properties such as performance, reliability, and resilience is essential. Developing techniques to incorporate these aspects into fuzzing is still an open challenge.
- **Resource Efficiency and Integration with Software Development Lifecycle:** Optimising the resource consumption of fuzzing tools is an ongoing challenge, given that fuzzing often requires significant computational resources. Discovering methods to enhance fuzzing efficiency without compromising effectiveness is imperative. For example, investigating bugs through an efficient architecture by seamlessly integrating fuzzing into the software development lifecycle ensures continuous and automated testing while conserving resources.
- **Fuzzing Web Applications and APIs:** Extending fuzzing to web applications, APIs, and other internet-facing systems is a vital area for improvement. Developing fuzzing methodologies that can effectively test web-based interfaces and protocols is essential

given the prevalence of web applications by automated Exploit Generation. After identifying vulnerabilities, automating the process of generating effective exploits or proofs of concept remains a challenging task. Future work should aim to automate the exploit generation process for identified vulnerabilities.

Addressing these challenges will contribute to advancing the field of fuzzing and improving the overall security of software systems.

9. Conclusion

In this study, we investigated a paradigm shift in fuzzing techniques through different techniques of ML. We have categorised these ML-based techniques into four distinct categories: TML, DL, RL, and DRL. Our investigation has revolved around the assessment of their potential to augment conventional fuzzing methodologies. First, we conducted a comprehensive survey of various strategies, such as feature engineering, predictive models, fitness models, and Long Input Correlation Tracing Models, as they relate to TML models. Additionally, we have elucidated the persistent challenges encountered when dealing with high-dimensional data in these traditional ML models.

Furthermore, we have discussed how the incorporation of state-of-the-art architectural components, such as LSTM, GAN, Seq2Seq, and GRU, within the framework of DNN models can significantly enhance the performance of fuzzing techniques. However, it is important to acknowledge that the issue of overfitting necessitates the consideration of more adaptable models. This avenue of exploration can be pursued through the utilisation of semi-supervised learning, vulnerability-guided fuzzing, contextual bandit algorithms, and a RL-based hierarchical scheduler within RL models.

Additionally, our research has delved into the areas of Curiosity-driven model, Deep Q-learning, and the integration of DNNs within the RL framework for applications in DRL as it relates to fuzzing. Moreover, we have underscored the pivotal role played by DNN and RL models in enhancing the efficacy of vulnerability detection and the identification of complex bugs, including those occurring in Kernel and File systems.

Our study highlights the transformative role of ML-based methods in seed selection, message generation for system fuzzing, and fuzzing grammar optimisation. By exploring ML techniques in fuzzing algorithms, our survey serves as a valuable resource for researchers and practitioners, providing insights into the most common and effective models and techniques. It aids in shaping the roadmap for applying diverse strategies in advancing fuzzing research.

CRediT authorship contribution statement

Sadegh Bamohabbat Chafjiri: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology. **Phil Legg:** Writing – review & editing, Supervision. **Jun Hong:** Supervision. **Michail-Antisthenis Tsompanas:** Writing – review & editing, Supervision.

Data availability

No data was used for the research described in the article.

Acknowledgments

We would like to thank the College of Arts, Technology and Environment at the University of the West of England, for funding this research through the PhD studentship scheme.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abdelnur, H., Festor, O., State, R., 2007. KiF: a stateful SIP fuzzer. In: Proceedings of the 1st International Conference on Principles, Systems and Applications of IP Telecommunications. pp. 47–56.

Anon, 2014. CVE-2014-6271. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-6271.

Anon, 2016. syzkaller. URL https://github.com/google/syzkaller, (Updated page accessed in August 2023).

Anon, 2021. CVE-2021-44228. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=2021-44228.

Becker, S., Abdelnur, H., State, R., Engel, T., 2010. An autonomic testing framework for IPv6 configuration protocols. In: Stiller, B., De Turck, F. (Eds.), Mechanisms for Autonomous Management of Networks and Services. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 65–76.

Binosi, L., Rullo, L., Polino, M., Carminati, M., Zanero, S., 2023. Rainfuzz: Reinforcement-learning driven heat-maps for boosting coverage-guided fuzzing. In: De Marsico, M., di Baja, G.S., Fred, A.L.N. (Eds.), Proceedings of the 12th International Conference on Pattern Recognition Applications and Methods. ICPRAM 2023, Lisbon, Portugal, February 22-24, 2023, SCITEPRESS, pp. 39–50. http://dx.doi.org/10.5220/0011625300003411.

Blum, W., Rajpal, M., Singh, R., 2017. Not all bytes are equal: Neural byte sieve for fuzzing. Cornell University Library, URL https://www.microsoft.com/en-us/research/publication/not-all-bytes-are-equal-neural-byte-sieve-for-fuzzing/.

Böhme, M., Pham, V.-T., Nguyen, M.-D., Roychoudhury, A., 2017. Directed greybox fuzzing. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 2329–2344.

Bottinger, K., Godefroid, P., Singh, R., 2018. Deep reinforcement fuzzing. In: 2018 IEEE Security and Privacy Workshops. SPW, IEEE Computer Society, Los Alamitos, CA, USA, pp. 116–122. http://dx.doi.org/10.1109/SPW.2018.00026, URL https://doi.ieeecomputersociety.org/10.1109/SPW.2018.00026.

Carvalho, M., DeMott, J., Ford, R., Wheeler, D.A., 2014. Heartbleed 101. IEEE Secur. Privacy 12 (4), 63–67. http://dx.doi.org/10.1109/MSP.2014.66.

Chen, Y., Ahmadi, M., Mirzazade farkhani, R., Wang, B., Lu, L., 2020. MEUZZ: Smart seed scheduling for hybrid fuzzing. In: Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses. RAID '20.

Cheng, L., Zhang, Y., Zhang, Y., Wu, C., Li, Z., Fu, Y., Li, H., 2019. Optimizing seed inputs in fuzzing with machine learning. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings. ICSE-Companion, pp. 244–245. http://dx.doi.org/10.1109/ICSE-Companion.2019.00096.

Choi, G., Jeon, S., Cho, J., Moon, J., 2023b. A seed scheduling method with a reinforcement learning for a coverage guided fuzzing. IEEE Access 11, 2048–2057. http://dx.doi.org/10.1109/ACCESS.2022.3233875.

Cummins, C., Petoumenos, P., Murray, A., Leather, H., 2018. Compiler fuzzing through deep learning. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2018, Association for Computing Machinery, New York, NY, USA, pp. 95–105. http://dx.doi.org/10.1145/3213846.3213848.

Daniele, C., Andarzian, S.B., Poll, E., 2024. Fuzzers for stateful systems: survey and research directions. ACM Comput. Surv. 56 (9), http://dx.doi.org/10.1145/3648468.

Drozd, W., Wagner, M.D., 2018. FuzzerGym: A competitive framework for fuzzing and learning. arxiv abs/1807.07490.

Duan, G., Fu, Y., Cai, M., Chen, H., Sun, J., 2023a. DongTing: A large-scale dataset for anomaly detection of the linux kernel. J. Syst. Softw. 203, 111745. http://dx.doi.org/10.1016/j.jss.2023.111745, URL https://www.sciencedirect.com/science/article/pii/S0164121223001401.

Duan, G., Fu, Y., Zhang, B., Deng, P., Sun, J., Chen, H., Chen, Z., 2023b. TEE-Fuzzer: A fuzzing framework for trusted execution environments with heuristic seed mutation. Future Gener. Comput. Syst. 144, 192–204. http://dx.doi.org/10.1016/j.future.2023.03.008, URL https://www.sciencedirect.com/science/article/pii/S0167739X23000857.

Fan, R., Chang, Y., 2018. Machine learning for black-box fuzzing of network protocols. ISBN: 978-3-319-89499-7, pp. 621–632. http://dx.doi.org/10.1007/978-3-319-89500-0_53.

Felderer, M., Büchler, M., Johns, M., Brucker, A.D., Breu, R., Pretschner, A., 2016. Chapter one - security testing: A survey. In: Memon, A. (Ed.), In: Advances in Computers, vol. 101, Elsevier, pp. 1–51. http://dx.doi.org/10.1016/bs.adcom.2015.11.003.

Feng, W., Lai, Y., Liu, Z., 2020. Vulnerability mining for modbus TCP based on exception field positioning. Simul. Model. Pract. Theory 102, 101989. http://dx.doi.org/10.1016/j.simpat.2019.101989, URL https://www.sciencedirect.com/science/article/pii/S1569190X19301224, Special Issue on IoT, Cloud, Big Data and AI in Interdisciplinary Domains.

Gong, K., Yang, W., Cui, B., Chen, C., 2022. DRLFCfuzzer: fuzzing with deep-reinforcement-learning under format constraints. In: 2022 2nd International Conference on Electronic Information Engineering and Computer Technology. EIECT, pp. 374–380. http://dx.doi.org/10.1109/EIECT58010.2022.00080.

Grieco, G., Grinblat, G.L., Uzal, L., Rawat, S., Feist, J., Mounier, L., 2016. Toward large-scale vulnerability discovery using machine learning. In: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. CODASPY '16, Association for Computing Machinery, New York, NY, USA, pp. 85–96. http://dx.doi.org/10.1145/2857705.2857720.

Groß, T., Schleier, T., Müller, T., 2022. ReFuzz - structure aware fuzzing of the resilient file system (ReFS). In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. ASIA CCS '22, Association for Computing Machinery, New York, NY, USA, pp. 589–601. http://dx.doi.org/10.1145/3488932.3523260.

He, J., Balunović, M., Ambroladze, N., Tsankov, P., Vechev, M., 2019. Learning to fuzz from symbolic execution with application to smart contracts. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. CCS '19, Association for Computing Machinery, New York, NY, USA, pp. 531–548. http://dx.doi.org/10.1145/3319535.3363230.

Hochreiter, S., 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 6, 107–116. http://dx.doi.org/10.1142/S0218488598000094.

Hu, Z., Shi, J., Huang, Y., Xiong, J., Bu, X., 2018. GANFuzz: A GAN-based industrial network protocol fuzzing framework. In: Proceedings of the 15th ACM International Conference on Computing Frontiers. CF '18, Association for Computing Machinery, New York, NY, USA, pp. 138–145. http://dx.doi.org/10.1145/3203217.3203241.

Huang, Y., Shu, H., Kang, F., Guang, Y., 2022a. Protocol reverse-engineering methods and tools: A survey. Comput. Commun. 182, 238–254. http://dx.doi.org/10.1016/j.comcom.2021.11.009, URL https://www.sciencedirect.com/science/article/pii/S0140366421004382.

Huang, Z., Song, X., Luo, Y., Yang, J., Cui, B., 2022b. Syzballer: Kernel fuzzing based on basic block weight and multi-armed bandit. In: 2022 IEEE 8th International Conference on Computer and Communications. ICCC, pp. 2364–2369. http://dx.doi.org/10.1109/ICCC56324.2022.10065711.

Jeon, S., Moon, J., 2022. Dr.PathFinder: hybrid fuzzing with deep reinforcement concolic execution toward deeper path-first search. Neural Comput. Appl. 34 (13), 10731–10750. http://dx.doi.org/10.1007/s00521-022-07008-8.

Jha, P., Scott, J., Ganeshna, J.S., Singh, M., Ganesh, V., 2023. BertRLFuzzer: A BERT and reinforcement learning based fuzzer. arXiv:2305.12534.

Karamcheti, S., Mann, G., Rosenberg, D., 2018. Adaptive grey-box fuzz-testing with thompson sampling. In: Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security. AISec '18, Association for Computing Machinery, New York, NY, USA, pp. 37–47. http://dx.doi.org/10.1145/3270101.3270108.

Koo, J., Saumya, C., Kulkarni, M., Bagchi, S., 2019. PySE: Automatic worst-case test generation by reinforcement learning. In: 2019 12th IEEE Conference on Software Testing, Validation and Verification. ICST, pp. 136–147. http://dx.doi.org/10.1109/ICST.2019.00023.

Li, D., Chen, H., 2019. FastSyzkaller: Improving fuzz efficiency for linux kernel fuzzing. J. Phys. Conf. Ser. 1176 (2), 022013. http://dx.doi.org/10.1088/1742-6596/1176/2/022013.

Li, Y., Ji, S., Lyu, C., Chen, Y., Chen, J., Gu, Q., Wu, C., Beyah, R., 2022a. V-Fuzz: Vulnerability prediction-assisted evolutionary fuzzing for binary programs. IEEE Trans. Cybern. 52 (5), 3745–3756. http://dx.doi.org/10.1109/TCYB.2020.3013675.

Li, X., Liu, X., Chen, L., Prajapati, R., Wu, D., 2022b. ALPHAPROG: Reinforcement generation of valid programs for compiler fuzzing. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36, No. 11. pp. 12559–12565. http://dx.doi.org/10.1609/aaai.v36i11.21527.

Li, X., Liu, X., Chen, L., Prajapati, R., Wu, D., 2022c. FuzzBoost: Reinforcement compiler fuzzing. In: Information and Communications Security: 24th International Conference, ICICS 2022, Canterbury, UK, September 5–8, 2022, Proceedings. Springer-Verlag, Berlin, Heidelberg, pp. 359–375. http://dx.doi.org/10.1007/978-3-031-15777-6_20.

Li, T., Wan, X., Özbek, M.M., 2022d. AgentFuzz: Fuzzing for deep reinforcement learning systems. In: 2022 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, pp. 110–113. http://dx.doi.org/10.1109/ISSREW55968.2022.00049.

Li, Z., Zhao, H., Shi, J., Huang, Y., Xiong, J., 2019. An intelligent fuzzing data generation method based on deep adversarial learning. IEEE Access 7, 49327–49340. http://dx.doi.org/10.1109/ACCESS.2019.2911121.

Liang, X., Xiao, T., 2022. RLF: Directed fuzzing based on deep reinforcement learning. In: 2022 International Conference on Machine Learning, Control, and Robotics. MLCR, pp. 127–133. http://dx.doi.org/10.1109/MLCR57210.2022.00032.

Lin, Y.-D., Lai, Y.-K., Bui, Q.T., Lai, Y.-C., 2020. ReFSM: Reverse engineering from protocol packet traces to test generation by extended finite state machines. J. Netw. Comput. Appl. 171, 102819. http://dx.doi.org/10.1016/j.jnca.2020.102819, URL https://www.sciencedirect.com/science/article/pii/S1084804520302897.

Liu, H., Patras, P., 2022. NetSentry: A deep learning approach to detecting incipient large-scale network attacks. Comput. Commun. 191, 119–132. http://dx.doi.org/10.1016/j.comcom.2022.04.020, URL https://www.sciencedirect.com/science/article/pii/S0140366422001335.

Liu, J., Wei, Y., Yang, S., Deng, Y., Zhang, L., 2022. Coverage-guided tensor compiler fuzzing with joint IR-pass mutation. Proc. ACM Program. Lang. 6 (OOPSLA1), http://dx.doi.org/10.1145/3527317.

Lyu, C., Ji, S., Li, Y., Zhou, J., Chen, J., Chen, J., 2019. SmartSeed: Smart seed generation for efficient fuzzing. arXiv:1807.02606.

Mallissery, S., Wu, Y.-S., 2023. Demystify the fuzzing methods: A comprehensive survey. ACM Comput. Surv. 56 (3), http://dx.doi.org/10.1145/3623375.

Miao, S., Wang, J., Zhang, C., Lin, Z., Gong, J., Zhang, X., Li, J., 2022. Deep learning in fuzzing: A literature survey. In: 2022 IEEE 2nd International Conference on Electronic Technology, Communication and Information. ICETCI, pp. 220–223. http://dx.doi.org/10.1109/ICETCI55101.2022.9832143.

Miller, B.P., Fredriksen, L., So, B., 1990. An empirical study of the reliability of UNIX utilities. Commun. ACM 33 (12), 32–44. http://dx.doi.org/10.1145/96267.96279.

Miller, B.P., Koski, D., Lee, C.P., Maganty, V., Murthy, R., Natarajan, A., Steidl, J., 1995. Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services. Tech. rep, Computer Sciences Department, University of Wisconsin-Madison.

Miller, C., Peterson, Z.N.J., 2007. Analysis of Mutation and Generation-Based Fuzzing. Tech. rep.

Molnar, D., Godefroid, P., Levin, M., 2008. Automated whitebox fuzz testing. In: Network and Distributed System Security Symposium. NDSS, pp. 416–426.

Nichols, N., Raugas, M., Jasper, R., Hilliard, N., 2017. Faster fuzzing: Reinitialization with deep neural models. arXiv:1711.02807.

Nouwou Mindom, P.S., Nikanjam, A., Khomh, F., 2023. A comparison of reinforcement learning frameworks for software testing tasks. Empir. Softw. Eng. 28, 111. http://dx.doi.org/10.1007/s10664-023-10363-2.

Paduraru, C., Paduraru, M., Stefanescu, A., 2021. RiverFuzzRL - an open-source tool to experiment with reinforcement learning for fuzzing. http://dx.doi.org/10.1109/ICST49551.2021.00055.

Page, M.J., McKenzie, J.E., Bossuyt, P.M., Boutron, I., Hoffmann, T.C., Mulrow, C.D., Shamseer, L., Tetzlaff, J.M., Akl, E.A., Brennan, S.E., Chou, R., Glanville, J., Grimshaw, J.M., Hróbjartsson, A., Lalu, M.M., Li, T., Loder, E.W., Mayo-Wilson, E., McDonald, S., McGuinness, L.A., Stewart, L.A., Thomas, J., Tricco, A.C., Welch, V.A., Whiting, P., Moher, D., 2021. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. BMJ 372, http://dx.doi.org/10.1136/bmj.n71, arXiv:https://www.bmj.com/content/372/bmj.n71.full.pdf, URL https://www.bmj.com/content/372/bmj.n71.

Pailoor, S., Aday, A., Jana, S., 2018. MoonShine: Optimizing OS fuzzer seed selection with trace distillation. In: 27th USENIX Security Symposium (USENIX Security 18). USENIX Association, Baltimore, MD, pp. 729–743, URL https://www.usenix.org/conference/usenixsecurity18/presentation/pailoor.

Pan, M., Huang, A., Wang, G., Zhang, T., Li, X., 2020. Reinforcement learning based curiosity-driven testing of android applications. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2020, Association for Computing Machinery, New York, NY, USA, pp. 153–164. http://dx.doi.org/10.1145/3395363.3397354.

Park, L.H., Chung, S., Kim, J., Kwon, T., 2023. GradFuzz: Fuzzing deep neural networks with gradient vector coverage for adversarial examples. Neurocomputing 522, 165–180. http://dx.doi.org/10.1016/j.neucom.2022.12.019, URL https://www.sciencedirect.com/science/article/pii/S0925231222015168.

Park, L.H., Kim, J., Park, J., Kwon, T., 2022. Mixed and constrained input mutation for effective fuzzing of deep learning systems. Inform. Sci. 614, 497–517. http://dx.doi.org/10.1016/j.ins.2022.10.079, URL https://www.sciencedirect.com/science/article/pii/S0020025522011999.

Patil, K., Kanade, A., 2018. Greybox fuzzing as a contextual bandits problem, arxiv abs/1806.03806.

Peleg, H., Singh, R., Name, Y., 2017. Learn&fuzz: Machine learning for input fuzzing. In: Proceedings of ASE'2017 (32nd International Conference on Automated Software Engineering). Urbana-Champaign, pp. 50–59.

Pham, H.V., Lutellier, T., Qi, W., Tan, L., 2019. CRADLE: Cross-backend validation to detect and localize bugs in deep learning libraries. In: 2019 IEEE/ACM 41st International Conference on Software Engineering. ICSE, pp. 1027–1038. http://dx.doi.org/10.1109/ICSE.2019.00107.

Qin, Y., Yue, C., 2022. Fuzzing-based hard-label black-box attacks against machine learning models. Comput. Secur. 117, 102694. http://dx.doi.org/10.1016/j.cose.2022.102694, URL https://www.sciencedirect.com/science/article/pii/S016740482200092X.

Raychev, V., Vechev, M., Krause, A., 2015. Predicting program properties from "big code". In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '15, Association for Computing Machinery, New York, NY, USA, pp. 111–124. http://dx.doi.org/10.1145/2676726.2677009.

Saavedra, G.J., Rodhouse, K.N., Dunlavy, D.M., Kegelmeyer, P.W., 2019. A review of machine learning applications in fuzzing. arXiv:1906.11133.

Sablotny, M., Jensen, B.S., Johnson, C.W., 2019. Recurrent neural networks for fuzz testing web browsers. In: Lee, K. (Ed.), Information Security and Cryptology – ICISC 2018. Springer International Publishing, Cham, pp. 354–370.

Schumilo, S., Aschermann, C., Gawlik, R., Schinzel, S., Holz, T., 2017. kAFL: Hardware-assisted feedback fuzzing for OS kernels. In: 26th USENIX Security Symposium. USENIX Security 17, USENIX Association, Vancouver, BC, pp. 167–182, URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schumilo.

Scott, J., Sudula, T., Rehman, H., Mora, F., Ganesh, V., 2021. BanditFuzz: Fuzzing SMT solvers with multi-agent reinforcement learning. In: Huisman, M., Păsăreanu, C., Zhan, N. (Eds.), Formal Methods. Springer International Publishing, Cham, pp. 103–121.

She, D., Krishna, R., Yan, L., Jana, S., Ray, B., 2020. MTFuzz: Fuzzing with a multi-task neural network. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. In: ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA, pp. 737–749. http://dx.doi.org/10.1145/3368089.3409723.

She, D., Pei, K., Epstein, D., Yang, J., Ray, B., Jana, S., 2019. NEUZZ: Efficient fuzzing with neural program smoothing. In: 2019 IEEE Symposium on Security and Privacy, Vol. 1. SP, pp. 803–817. http://dx.doi.org/10.1109/SP.2019.00052.

Su, J., Dai, H.-N., Zhao, L., Zheng, Z., Luo, X., 2023. Effectively generating vulnerable transaction sequences in smart contracts with reinforcement learning-guided fuzzing. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. ASE '22, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/3551349.3560429.

Sun, X., Fu, Y., Dong, Y., Liu, Z., Zhang, Y., 2018. Improving fitness function for language fuzzing with PCFG model. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference, Vol. 01. COMPSAC, pp. 655–660. http://dx.doi.org/10.1109/COMPSAC.2018.00098.

SunHao-0, 2020. HEALER: A program-analysis-guided fuzzer for linux kernel drivers. https://github.com/SunHao-0/healer (Updated page accessed in August 2023).

Takanen, A., 2009. Fuzzing is still widely unknown. https://www.computerworld.com/article/2769688/fuzzing-is-still-widely-unknown.html. (Accessed: 27 August 2022).

Tao, C., Tao, Y., Guo, H., Huang, Z., Sun, X., 2023. DLRegion: Coverage-guided fuzz testing of deep neural networks with region-based neuron selection strategies. Inf. Softw. Technol. 162, 107266. http://dx.doi.org/10.1016/j.infsof.2023.107266, URL https://www.sciencedirect.com/science/article/pii/S0950584923001209.

Tappler, M., Cano Córdoba, F., Aichernig, B.K., Könighofer, B., 2022. Search-based testing of reinforcement learning. In: Raedt, L.D. (Ed.), Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence. IJCAI-22, International Joint Conferences on Artificial Intelligence Organization, pp. 503–510. http://dx.doi.org/10.24963/ijcai.2022/72, Main Track.

Tripathi, S., Grieco, G., Rawat, S., 2017. Exniffer: Learning to prioritize crashes by assessing the exploitability from memory dump. In: 2017 24th Asia-Pacific Software Engineering Conference. APSEC, pp. 239–248. http://dx.doi.org/10.1109/APSEC.2017.30.

Wang, J., Chen, B., Wei, L., Liu, Y., 2017. Skyfire: Data-driven seed generation for fuzzing. In: 2017 IEEE Symposium on Security and Privacy. SP, pp. 579–594. http://dx.doi.org/10.1109/SP.2017.23.

Wang, Q., Gao, Y., Ren, J., Zhang, B., 2023. An automatic classification algorithm for software vulnerability based on weighted word vector and fusion neural network. Comput. Secur. 126, 103070. http://dx.doi.org/10.1016/j.cose.2022.103070, URL https://www.sciencedirect.com/science/article/pii/S016740482200462X.

Wang, Y., Jia, P., Liu, L., Huang, C., Liu, Z., 2020a. A systematic review of fuzzing based on machine learning techniques. PLOS ONE 15 (8), 1–37. http://dx.doi.org/10.1371/journal.pone.0237749.

Wang, J., Song, C., Yin, H., 2021a. Reinforcement learning-based hierarchical seed scheduling for greybox fuzzing. In: Proceedings 2021 Network and Distributed System Security Symposium.

Wang, Y., Wu, Z., Wei, Q., Wang, Q., 2019. NeuFuzz: Efficient fuzzing with deep neural network. IEEE Access 7, 36340–36352. http://dx.doi.org/10.1109/ACCESS.2019.2903291.

Wang, Z., Yan, M., Chen, J., Liu, S., Zhang, D., 2020b. Deep learning library testing via effective model generation. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. In: ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA, pp. 788–799. http://dx.doi.org/10.1145/3368089.3409761.

Wang, D., Zhang, Z., Zhang, H., Qian, Z., Krishnamurthy, S.V., Abu-Ghazaleh, N., 2021b. SyzVegas: Beating kernel fuzzing odds with reinforcement learning. In: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, pp. 2741–2758, URL https://www.usenix.org/conference/usenixsecurity21/presentation/wang-daimeng.

Watkins, C., 1989. Learning from Delayed Rewards (Ph.D. thesis). University of Cambridge, England.

Watkins, C.J., Dayan, P., 1992. Q-learning. Mach. Learn. 8 (3–4), 279–292.

Wei, A., Deng, Y., Yang, C., Zhang, L., 2022. Free lunch for testing: Fuzzing deep-learning libraries from open source. In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, Association for Computing Machinery, New York, NY, USA, pp. 995–1007. http://dx.doi.org/10.1145/3510003.3510041.

Wu, M., Jiang, L., Xiang, J., Zhang, Y., Yang, G., Ma, H., Nie, S., Wu, S., Cui, H., Zhang, L., 2022. Evaluating and improving neural program-smoothing-based fuzzing. In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, Association for Computing Machinery, New York, NY, USA, pp. 847–858. http://dx.doi.org/10.1145/3510003.3510089.

Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S., 2019. DeepHunter: A coverage-guided fuzz testing framework for deep neural networks. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2019, Association for Computing Machinery, New York, NY, USA, pp. 146–157. http://dx.doi.org/10.1145/3293882.3330579.

Xu, W., Moon, H., Kashyap, S., Tseng, P.-N., Kim, T., 2019. Fuzzing file systems via two-dimensional input space exploration. In: 2019 IEEE Symposium on Security and Privacy. SP, pp. 818–834. http://dx.doi.org/10.1109/SP.2019.00035.

Ye, A., Wang, L., Zhao, L., Ke, J., Wang, W., Liu, Q., 2021. RapidFuzz: Accelerating fuzzing via generative adversarial networks. Neurocomputing 460, 195–204. http://dx.doi.org/10.1016/j.neucom.2021.06.082, URL https://www.sciencedirect.com/science/article/pii/S0925231221010122.

Zakeri Nasrabadi, M., Parsa, S., Kalaee, A., 2021. Format-aware learn&fuzz: deep test data generation for efficient fuzzing. Neural Comput. Appl. 33 (5), 1497–1513. http://dx.doi.org/10.1007/s00521-020-05039-7.

Zalewski, M., 2020. American fuzz lop. https://github.com/google/AFL.

Zhang, L., Thing, V.L.L., 2018. Assisting vulnerability detection by prioritizing crashes with incremental learning. In: TENCON 2018 - 2018 IEEE Region 10 Conference. pp. 2080–2085. http://dx.doi.org/10.1109/TENCON.2018.8650188.

Zhang, G., Zhou, X., Luo, Y., Wu, X., Min, E., 2018. PTfuzz: Guided fuzzing with processor trace feedback. IEEE Access 6, 37302–37313. http://dx.doi.org/10.1109/ACCESS.2018.2851237.

Zhao, H., Li, Z., Wei, H., Shi, J., Huang, Y., 2019. SeqFuzzer: An industrial protocol fuzzing framework from a deep learning perspective. In: 2019 12th IEEE Conference on Software Testing, Validation and Verification. ICST, pp. 59–67. http://dx.doi.org/10.1109/ICST.2019.00016.

Zheng, Y., Liu, Y., Xie, X., Liu, Y., Ma, L., Hao, J., Liu, Y., 2021. Automatic web testing using curiosity-driven reinforcement learning. In: Proceedings of the 43rd International Conference on Software Engineering. ICSE '21, IEEE Press, pp. 423–435. http://dx.doi.org/10.1109/ICSE43902.2021.00048.

Zong, P., Lv, T., Wang, D., Deng, Z., Liang, R., Chen, K., 2020. FuzzGuard: Filtering out unreachable inputs in directed grey-box fuzzing through deep learning. In: 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, pp. 2255–2269, URL https://www.usenix.org/conference/usenixsecurity20/presentation/zong.

Sadegh Bamohabbat Chafjiri is a postgraduate researcher at the University of the West of England, specialising in vulnerability assessment and software testing. His Ph.D. focuses on an AI-enabled software security framework, with an emphasis on fuzz testing. With a background in entrepreneurship and IT management, Sadegh holds a bachelor's degree in electrical engineering and two master's degrees in telecommunication engineering, specialising in cryptography and IT management with a data analytics focus. He has extensive experience in security engineering, data analytics, AI-enabled solutions, and the Internet of Things. Leveraging his expertise, Sadegh has consistently delivered innovative solutions in software security and sustainable infrastructures.

Phil Legg is a Professor of Cyber Security at the University of the West of England (UWE Bristol). His research interests span across cyber security, machine learning, visualisation, and human–computer interaction, to better understand the detection and mitigation of security threats. He has led various research activities supported by DSTL, NCSC, UKRI, CCAV, and CPNI, along with industry and academic collaborators. He has published over 60 academic journals and conference papers across his research interests (citation count: 1694; h-index: 20), with successful research funding of over £2.2M. He is the Programme Leader of the M.Sc. Cyber Security and Co-Director of the NCSC-supported Academic Centre of Excellence in Cyber Security Education.

Jun Hong is a Professor of Artificial Intelligence in the Department of Computer Science and Creative Technologies at UWE Bristol. He is a member of the Computer Science Research Centre (CCRC). His current research interests are centred around data extraction, integration, linkage, and analytics (health, consumer, business, administrative, social media/Web, structured and unstructured data, privacy preservation); graph mining and social network analysis; and intelligent autonomous systems (AI planning and BDI-based multi-agent systems).

Michail-Antisthenis Tsompanas holds a bachelor's, master's, and doctorate degrees in Electrical and Computer Engineering from Democritus University of Thrace, Greece. Currently, he is a Lecturer in Computer Science at the University of the West of England. With expertise in modelling biological processes and both conventional and unconventional computing, he has published in renowned scientific journals. His research interests encompass unconventional and bio-inspired computations, modelling and simulation, electronic systems design, Cellular Automata theory, and applications. He has contributed to multiple European and part-national part-European projects. Since 2009, he has been a member of the Technical Chamber of Greece.