

Reinforcement Learning-Guided Protocol Fuzzing for 5G Core Networks and Beyond

Manh-Dung Nguyen¹, Eden Arfi², Huu Nghia Nguyen¹, Bruno
Vidalenc², Ana Rosa Cavalli¹, and Edgardo Montes de Oca¹

¹ Montimage EURL, 37 rue Bobillot, 75013 Paris, France

² Thales SIX GTS France, Gennevilliers, France

Abstract. The 5G Service-Based Architecture introduces a highly distributed and stateful core network, increasing the difficulty of security testing. Existing fuzzers either rely on instrumentation and coverage feedback, which is often unavailable in 5G deployments, or on static rules that lack adaptability and miss deep protocol states. We present NETWORKFUZZER, a reinforcement learning-guided fuzzing framework for 5G Core networks that combines learned action selection with knowledge from 3GPP specifications and vulnerability reports. NETWORKFUZZER generates protocol-aware and stateful test cases across SBI, NGAP, and NAS, while adapting its strategy using runtime feedback such as responses, errors, and crashes. We evaluate NETWORKFUZZER on open-source 5G Core network implementations, including Open5GS, free5GC, and Ella Core. Results show that NETWORKFUZZER outperforms rule-based fuzzers and matches or exceeds a stronger specification-driven baseline in coverage and defect discovery, while using fewer executions. NETWORKFUZZER detected and confirmed several crashes in the latest version of Open5GS, including one CVE, demonstrating that reinforcement learning combined with protocol knowledge enables more efficient 5G core security testing.

Keywords: Network Fuzzing · Reinforcement Learning · 5G Core

1 Introduction

Context. The transition from monolithic 4G architectures to the 5G Service-Based Architecture (SBA) has fundamentally changed how mobile core networks are designed and deployed. Network functions are now implemented as software services communicating through standardized interfaces, enabled by technologies such as Software-Defined Networking (SDN), Network Function Virtualization (NFV), Mobile Edge Computing (MEC), and network slicing. While this architecture improves flexibility and scalability, it also significantly enlarges the attack surface. Critical protocols, including the Service-Based Interface (SBI), NGAP, and NAS, process complex, stateful interactions whose implementation flaws can lead to authentication bypass, denial-of-service (DoS), incorrect state

transitions, or other security vulnerabilities. Recognizing these risks, 3GPP defines security assurance requirements for robustness against malformed and unexpected inputs, highlighting the need for effective security testing of 5G Core (5GC) implementations.

Fuzzing is an effective technique for uncovering vulnerabilities in network protocol implementations by repeatedly generating inputs and monitoring the target for anomalous behavior. General-purpose network fuzzers such as AFLNET [24] rely on instrumentation and code coverage to guide exploration, making them less suitable for closed-source 5G core systems where such feedback is unavailable. Existing 5G rule-based fuzzers, such as 5GREPLAY [25], also have important limitations: they depend on manually crafted mutation rules, lack context-awareness, and cannot adapt message sequences based on protocol responses. Consequently, they struggle to explore deep, stateful protocol behaviors and may miss vulnerabilities that require adaptive multi-step interactions.

Proposal. To address these limitations, we propose NETWORKFUZZER, a reinforcement learning (RL)-guided network fuzzer for 5GC networks and beyond. The space of valid mutations in 5GC protocols is large, structured, and inherently stateful, where vulnerabilities are often exposed only through sequences of interactions rather than individual messages. Rather than relying on fixed mutation rules or random exploration, NETWORKFUZZER learns an adaptive fuzzing policy from runtime feedback across full fuzzing episodes, enabling it to prioritize promising protocol states, mutation strategies, and multi-step interactions. The action space combines protocol knowledge extracted from 3GPP specifications with attack patterns mined from publicly disclosed vulnerabilities, allowing the agent to generate protocol-aware, semantically valid, and stateful test cases across interfaces such as SBI, NGAP, and NAS. At each step, the agent selects a mutation action, observes server responses and runtime signals (e.g., errors, logs, crashes, and state transitions), and receives a reward that reflects anomaly discovery, deeper execution, and exploration of unseen states. This feedback-driven loop gradually concentrates exploration on strategies most likely to expose faults. The learned policy is guided by rewards derived from behavioral feedback, code coverage (when available), and confirmed failures, enabling efficient exploration of deep protocol states while remaining applicable to both instrumented and black-box deployments.

Contributions. Our main contributions are summarized as follows.

1. We design and implement an RL-guided fuzzing engine that leverages reinforcement learning algorithms to enable the agent to learn mutation strategies that effectively explore deeper execution paths and destabilize target systems through crashes, hangs, and other abnormal behaviors.
2. We conduct extensive experiments on popular open-source 5G core implementations, including Open5GS [4], free5GC [3] and Ella Core [2], demonstrating effective identification of robustness and security vulnerabilities.

The remainder of this paper is organized as follows. Section 2 presents background information on 5G systems and common 5G attacks, while Section 3

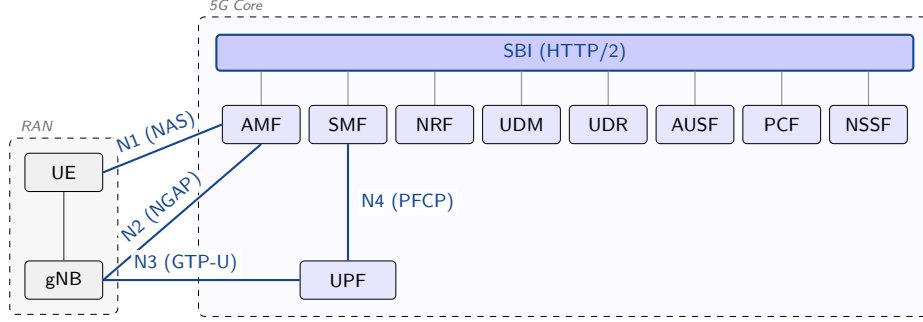


Fig. 1: 5G system architecture and the interfaces relevant to 5GC fuzzing.

reviews related work on network protocol fuzzing for 5G core services. Section 4 describes the design of the RL-guided fuzzing engine tailored to 5G core networks. Section 5 presents the implementation details and experimental evaluation of the proposed framework across different 5G core implementations. Finally, Section 6 concludes the paper and outlines future research directions.

2 Background

2.1 5G Core Architecture and Interfaces

The 5G system consists of the User Equipment (UE), the Radio Access Network (RAN), and the 5G Core (5GC). Figure 1 illustrates the 5G Core components and interfaces relevant to this work. The Service-Based Interface (SBI) connects control-plane functions through HTTP/2 APIs. On the access side, Non-Access Stratum (NAS) signaling is exchanged between the UE and the core, while the NG Application Protocol (NGAP) connects the radio access network to the AMF. The Packet Forwarding Control Protocol (PFCP) allows the SMF to configure forwarding rules in the UPF, whereas the GPRS Tunneling Protocol User Plane (GTP-U) carries user traffic between the radio network and the UPF. These interfaces differ in message formats, transport protocols, and state dependencies, which makes 5G Core fuzzing heterogeneous and requires protocol-aware testing strategies.

2.2 5G Attacks

The 5G Core exposes several interfaces and services, which increases the number of possible attack points. On the control plane, malformed or unexpected signaling messages can cause parsing errors, invalid state transitions, resource exhaustion, or denial of service. Failures may also arise from valid messages sent in an unexpected order or with inconsistent session context. Similar issues can occur in service-based interactions, where a compromised core function may invoke unauthorized services, misuse service discovery, replay valid requests in the wrong context, or send inconsistent data. Authorization mechanisms such as

OAuth reduce these risks, but implementation flaws may still lead to improper access or privilege escalation.

The user plane also presents security-sensitive behavior. Incorrect PFCP rule handling can alter forwarding decisions, while malformed GTP-U packets can affect tunnel or packet processing. For fuzzing, we therefore focus on observable failures such as crashes, hangs, runtime errors, inconsistent states, unauthorized access, and denial-of-service conditions.

2.3 Rule-Based Replay Fuzzer

The development of NETWORKFUZZER builds on a rule-based replay approach [25,8]. An earlier version of NETWORKFUZZER targeted the Digital Imaging and Communications in Medicine (DICOM) protocol [21,22]. In this work, NETWORKFUZZER is extended to 5G Core protocols. The rule-based replay fuzzer starts from valid traffic traces or predefined scenarios, replays protocol exchanges against a target 5GC implementation, and applies predefined mutation rules to selected protocol fields. Replay-based fuzzing is particularly suitable for 5G because many messages depend on previously negotiated values, such as identifiers, session parameters, security contexts, and tunnel information. By preserving this protocol context, the fuzzer can reach realistic protocol states while maintaining sufficiently consistent message sequences for the target to process.

3 Related Work

Stateful Network Protocol Fuzzing. Network protocol fuzzing aims to assess the robustness of protocol implementations by generating unexpected, malformed, or partially invalid message sequences and observing the behavior of the target system. A central difficulty is that network protocols are stateful by design. Messages are processed according to an expected order, and their validity often depends on previous exchanges and on the current state of the server. Consequently, effective fuzzing of network protocols requires considering not only individual messages, but also the sequences in which they appear.

AFLNET [24] is a representative stateful greybox fuzzer that extends coverage-guided fuzzing to network servers by mutating and replaying message sequences while using code coverage and server response codes to approximate protocol states and prioritize state-exploring inputs. Although it provides a strong general-purpose baseline, it remains protocol-agnostic and does not model 5G-specific semantics, such as SUPI, PDU Session ID, SEID, TEID, S-NSSAI, or NF identifiers, whose validity depends on previously established protocol context.

5G-Specific Fuzzing. Several works inject 5G knowledge into fuzzing. He et al. [12] proposed a NAS mutation strategies from 3GPP specifications and captured packets, targeting structured fields such as message types, lengths, reserved values, and IEs. AMFUZZ [13] is a black-box proxy between the gNB and the core that intercepts and modifies NAS/NGAP messages within realistic UE/RAN procedures, preserving the context needed to reach later AMF states. More recently, 5GC-FUZZ [27] proposed a black-box stateful fuzzing using a 3GPP-derived state machine, grammar-aware NAS/NGAP mutations, and

log-based feedback to reach deeper states. FIVGEEFUZZ [9] derives grammars from 3GPP OpenAPI specifications and uses RESTler to generate stateful SBI request sequences; evaluated on free5GC, it uncovered runtime panics, incorrect error handling, and a Cross-Service Token authorization flaw.

These tools show that effective 5G fuzzing requires protocol- and state-aware input generation. However, they rely on predefined rules, manually built state models, or heuristics, and are typically limited to a single protocol interface. As a result, extending them to new protocols or core implementations requires substantial manual effort, and none adapts its fuzzing policy during the campaign.

Learning-Based and RL-Guided Fuzzing. Learning-based methods reduce dependence on fixed rules. In 5G, Nakas et al. [14] proposed a GAN-based generator for PFCP fields (e.g., SEID, sequence numbers) learned from historical attacks, but it neither learns online nor frames fuzzing as sequential decision-making. RL has been used to guide campaigns more directly: SCFUZZ [10] models seed scheduling as a multi-armed bandit and RLGFUZZ [26] uses a D3QN agent over a learned state-coverage map, yet both are evaluated on generic protocols (RTSP, DNS, SIP, DTLS) and rely on code coverage, which is rarely available in realistic 5G cores.

Closer to 5G, Parikh et al. [23] fuzz the RRC layer with system-level rewards (CPU, threads, I/O, crashes) but target the radio side rather than the core, while GORGO [11] uses multiple DQN agents for DoS strategy generation against the SMF/UPF/AMF, optimizing QoS-degradation rewards with generic payloads that do not deeply model SBI, or NGAP/NAS, semantics.

Positioning of Our Work. Table 1 contrasts NETWORKFUZZER with the tools discussed above along two axes: the 5GC protocols a tool exercises and the techniques it employs. A tool is *spec-aware* when its mutations derive from 3GPP OpenAPI definitions or protocol grammars, *coverage-guided* when it evolves inputs from greybox code-coverage feedback, *stateful* when it respects producer-consumer and session dependencies, and *learning-guided* when it uses reinforcement learning to steer input selection.

Two observations follow from the table. First, no existing tool combines all four techniques: spec-aware fuzzers (e.g., FIVGEEFUZZ) are not learning-based, RL fuzzers are not spec-aware, and AFLNET is coverage-guided but protocol-agnostic. Second, protocol support is often partial: 5GREPLAY applies fixed rules to limited fields, AFLNET requires per-protocol harnessing and seeds, and 5G-specific tools typically target a single interface. Only NETWORKFUZZER is simultaneously spec-aware, coverage-guided, stateful, and learning-guided across SBI, and NGAP/NAS.

The key distinction of NETWORKFUZZER is its adaptive learning strategy. Prior 5G fuzzers rely on fixed rules or manual state models, while RL-based approaches target generic protocols or DoS objectives. NETWORKFUZZER instead learns a policy that jointly selects protocols, states, and mutations. It relies on runtime signals, including responses, errors, logs, timeouts, crashes, restarts, and

Table 1: Comparison of network protocol fuzzing tools for 5GC testing. A half circle (◐) denotes partial or indirect support.

Tool	Protocols		Technique			
	SBI	NGAP/NAS	Spec-aware	Coverage-guided	Stateful	Learning-guided
5GREPLAY	◐	◐	◐	✗	✓	✗
FIVGEEFUZZ	✓	✗	✓	✗	✓	✗
AFLNET	◐	◐	✗	✓	✓	✗
NETWORKFUZZER	✓	✓	✓	✓	✓	✓

alerts, rather than code coverage alone, making it applicable even when instrumentation is limited.

4 RL-guided Fuzzing Engine

4.1 Problem Formulation and Overview

We cast network fuzzing as a sequential decision-making problem and model it as a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. At each step t , the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$, receives a reward $r_t = \mathcal{R}(s_t, a_t)$, and the environment transitions to $s_{t+1} \sim \mathcal{T}(\cdot | s_t, a_t)$. In our setting, $\mathcal{S}$ summarizes the fuzzing context (e.g., recent responses, novelty, and crash signals), while $\mathcal{A}$ includes mutation actions such as specification-driven, 5G-semantic, vulnerability-guided, and stateful sequence mutations. The reward function promotes deeper responses, novel behaviors, and failure-inducing inputs, and $\gamma \in [0, 1]$ balances immediate and long-term exploration toward faulty states.

Figure 2 instantiates this formulation as a *closed interaction loop* comprising four stages. The first, an offline knowledge-extraction stage, runs once before fuzzing: it parses the 3GPP service specifications and mines previously reported issues to extract message structures, field constraints, 5G-semantic value pools, and known attack patterns. These populate the second stage, the action space $\mathcal{A}$ defined above. In the third stage, the RL agent observes the current state s_t and selects a mutation action $a_t \in \mathcal{A}$. Finally, the fuzzing environment constructs a message by combining a valid baseline with the selected mutation, sends it to the live 5GC, and monitors the target’s response, logs, and process liveness, as well as line- and function-level code coverage when an instrumented build is available. From these observations it derives the next state s_{t+1} and the reward r_t , which are fed back to the agent to close the loop and progressively refine its policy. Crash-, hang-, and leak-triggering inputs are de-duplicated and saved to a reproducible corpus for replay and triage.

4.2 Offline Knowledge Extraction

Before fuzzing, NETWORKFUZZER extracts two sources of domain knowledge to populate its action space: (1) *protocol-specification mining*, which derives message structures and constraints, and (2) *vulnerability-report mining*, which extracts recurring attack patterns.

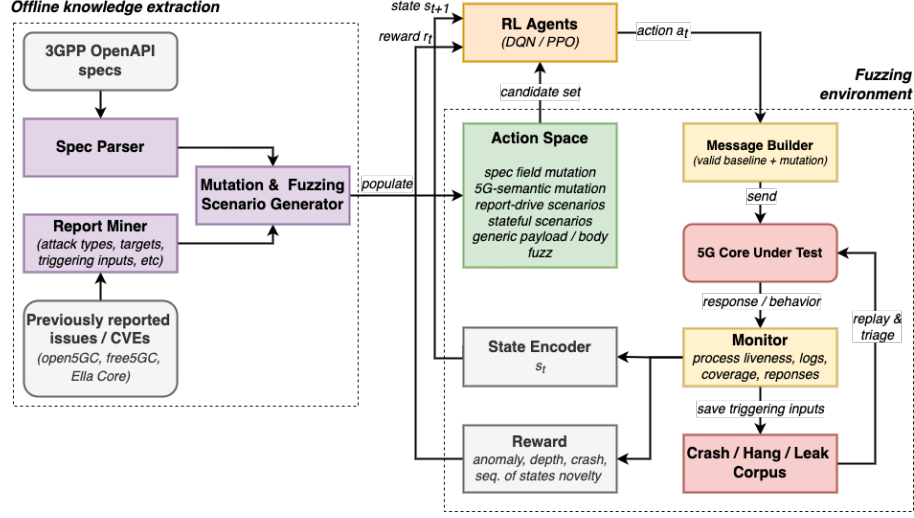


Fig. 2: Architecture of NETWORKFUZZER’s RL-guided Traffic Generator module.

Protocol-specification mining. NETWORKFUZZER automatically extracts protocol definitions, message structures, and constraints from machine-readable specifications to generate structurally valid messages and protocol-aware mutations. In this work, we focus on the SBI, whose APIs are defined by OpenAPI specifications in the 3GPP TS 29.5xx series [5]; other protocols (e.g., NGAP/-NAS) are currently supported through hand-crafted templates. Unlike FIVGEEFUZZ [9], NETWORKFUZZER extracts field-level constraints, enriches values with protocol semantics, and identifies inter-operation dependencies to generate stateful producer-consumer scenarios rather than fuzzing requests in isolation. These specification-derived actions form the basis of the RL-guided fuzzing strategy.

Vulnerability-knowledge mining. NETWORKFUZZER incorporates knowledge extracted from public vulnerability reports, bug trackers, and security advisories. Only vulnerabilities reported before Open5GS v2.7.5, free5GC v4.1.0, and Ella Core v1.10.0 are used; no reports affecting the evaluated versions are included. NETWORKFUZZER generalizes reported defects into reusable attack patterns and mutation strategies targeting common weaknesses such as malformed inputs, missing fields, boundary violations, and incorrect state handling. This knowledge helps assess patch robustness and guides testing across multiple implementations of the same protocol, where similar flaws may recur. NETWORKFUZZER prioritizes historically vulnerable code paths, increasing the likelihood of uncovering security-relevant behaviors, similar to directed fuzzing [7,17].

4.3 State Representation

NETWORKFUZZER abstracts the environment state into a compact behavioral feature vector $s \in \mathcal{S} \subset \mathbb{R}^d$ that summarizes how the target reacted to recent traffic rather than what was sent. Concretely, the observation concatenates:

(i) the last k response types observed on the connection (e.g., success, client rejection, server error, or transport-level framing errors); (ii) a normalized histogram of responses accumulated over the episode, grouped into broad behavioral classes such as success, client rejection, rarer error conditions, server faults, and transport-level or HTTP/2 framing errors; (iii) the number of distinct response types reached so far in the episode, which serves as a cheap proxy for behavioral diversity; (iv) episode progress together with running counters of crashes, hangs, and successful requests; and (v) the normalized values of the currently selected semantic protocol fields.

4.4 Action Space Design

The action space $\mathcal{A}$ is the set of mutation strategies available to the agent, combining offline knowledge sources. Each action is defined as $a = (o, \tau, \sigma)$, where o is the mutation operator, τ the target field, and σ the message sequence providing protocol context. To reduce complexity, the agent limits the number of scenarios per operation, and a pre-execution probe filters unsupported endpoints and targets, ensuring that only reachable actions are explored. NETWORKFUZZER defines five action types:

- *Spec-derived mutations.* Boundary and constraint-violating values from parsed SBI schemas (e.g., out-of-bounds, missing fields, invalid enumerations).
- *Generic payload injection.* Protocol-agnostic fuzzing inputs (e.g., format strings, oversized buffers, nested structures).
- *5G-semantic mutations.* Values tailored to 3GPP semantics for key identifiers (e.g., SUPI/SUCI, S-NSSAI, NF IDs, OAuth scopes) to trigger protocol-specific validation.
- *Report-derived scenarios.* Attack patterns from prior vulnerabilities, including missing fields, type confusion, and token reuse.
- *Stateful actions.* For each case, NETWORKFUZZER builds a multi-step action that first issues a valid producer request, propagates the resulting identifiers, and then fuzzes the dependent consumer call to reach deeper protocol states.

4.5 Reward Function

The reward $\mathcal{R}$ is designed to prioritize *progress toward anomalous and deep behaviors* over mere protocol conformance. It is defined as the sum of three sub-rewards, each aggregating multiple signals:

$$\mathcal{R}(s_t, a_t) = R_{\text{crash}} + R_{\text{explore}} + R_{\text{behavior}} \quad (1)$$

- R_{crash} captures confirmed failures and provides the dominant signal: process crashes yield the highest reward, followed by panics, fatal assertions, and confirmed hangs.
- R_{explore} encourages exploration of new behaviors, following the intuition of coverage-guided fuzzing [1] that exercising previously unseen code or protocol states is more likely to expose defects. It combines two signals: a coverage-guided bonus for newly executed source lines when instrumentation is available, and a coverage-independent sequence-of-states bonus, awarded when the agent reaches a novel state or a novel multi-step state trajectory.

- R_{behavior} captures fine-grained signals such as slow responses, transport errors, and deeper validation triggers.

Furthermore, signals are weighted so that confirmed defects dominate, while exploration and behavioral cues guide the search. To avoid reward duplication, each crash, hang, or anomaly is de-duplicated using a signature, preventing repeated rewards for the same defect.

5 Implementation & Evaluation

5.1 Implementation

NETWORKFUZZER extends 5GREPLAY [25] with protocol-aware mutation operators and an RL-guided fuzzing engine supporting multiple 5G protocols and core services. It uses pluggable adapters for SBI (HTTP/2), and NGAP/NAS, and the Ella Core REST API, enabling protocol-specific message generation within a unified framework. NETWORKFUZZER parses 3GPP OpenAPI specifications to derive schema-aware mutations enriched with vulnerability-knowledge patterns. The RL engine is built with Stable-Baselines3 [6] and Gymnasium, using DQN and PPO with standard settings ($\gamma = 0.99$, and ϵ -greedy annealing from 1.0 to 0.05 for DQN). Rewards combine crash signals, runtime failures, and coverage novelty (via `gcov`). A monitoring module detects crashes, panics, sanitizer reports, connection failures, and memory growth, enabling detection of both functional and resource-exhaustion vulnerabilities.

5.2 Experimental Setup

Research Questions. In this section, we evaluate the effectiveness and efficiency of NETWORKFUZZER across multiple 5GC implementations.

RQ1 (Fuzzing Effectiveness): How does NETWORKFUZZER compare with existing fuzzing tools in coverage, exploration, and bug discovery across open-source 5GC implementations?

RQ2 (Ablation Study): What is the contribution of NETWORKFUZZER’s key components (RL guidance, spec-derived mutations, and stateful chaining) to overall effectiveness?

RQ3 (Real-World Vulnerability Discovery): Can NETWORKFUZZER uncover security vulnerabilities in production 5GC implementations?

Testbed Setup. NETWORKFUZZER can be integrated into open-source 5G testbeds [19] that combine physical and virtual components, including COTS equipment and UE/RAN emulators such as UERANSIM and OpenAirInterface. In this work, however, we use a simplified and fully reproducible setup where each 5GC is deployed locally as a set of services, and NETWORKFUZZER runs on the same machine, interacting directly with network function interfaces. This removes radio-layer variability and enables controlled, repeatable fuzzing with consistent coverage and runtime measurements.

Evaluation Setup. We evaluate NETWORKFUZZER on the latest available releases of three open-source 5G core implementations (Open5GS v2.7.7, free5GC

v4.2.2, and Ella Core v1.11.1). None of these versions were used in the vulnerability-knowledge base. Open5GS (C, modular architecture), free5GC (Go, microservices), and Ella Core (Go, monolithic design) represent different implementation styles. We fuzz the SBI over HTTP/2 and, where applicable, NGAP, and NAS. RQ1 and RQ2 use Open5GS as the primary benchmark due to its maturity and consistent coverage instrumentation, while RQ3 evaluates generality on free5GC and Ella Core.

Baselines. We compare NETWORKFUZZER against 5GREPLAY [25], a rule-based replay fuzzer, and FIVGEEFUZZ [9], a protocol-aware SBI fuzzer. As FIVGEEFUZZ is not open source, we reimplemented its mutation strategies from the paper. All experiments run on the same KVM-based Ubuntu 20.10 VPS (4 Intel Haswell vCPUs @ 2.4 GHz, 8 GB RAM) with identical budgets and $N = 5$ trials.

Metrics. Line and function coverage are measured using `lcov`; since coverage is computed over the full codebase, only relative differences are meaningful. Confirmed defects are reproducible sanitizer reports or assertion aborts verified by replay. We also report time-to-first-crash (TTFC), Vargha–Delaney effect size (A_{12}), and the two-sided Mann–Whitney U test.

5.3 RQ1: Fuzzing Effectiveness

Comparison with 5GREPLAY. We evaluate NETWORKFUZZER against 5GREPLAY on the Open5GS AMF over its N2 interface, using the two mutation configurations 5GREPLAY natively supports: NGAP-envelope mutations (rules 6–10) and NAS-decoder mutations (rule 4). Both tools use the same packet-processing engine and run under identical conditions ($N = 5$ independent 30 min trials per track), isolating the effect of RL-guided fuzzing.

Table 2 and Figure 3 show that NETWORKFUZZER matches or exceeds 5GREPLAY’s coverage on both tracks despite operating several orders of magnitude more slowly: function coverage increases from 11.3% to 11.8% ($A_{12} = 1.00$) and line coverage from 6.4–6.5% to 6.7% ($A_{12} = 0.96$ –1.00), with the throughput gap reflecting the cost of policy inference and frequent AMF restarts after defect-triggering executions. More importantly, on both tracks NETWORKFUZZER triggers two UB defects in all 5/5 trials, whereas 5GREPLAY triggers none: a left-shift of a negative value in the NGAP/ASN.1 open-type decoder and a misaligned member access in the NAS mobile-identity decoder. The identical outcome across both injection layers shows that NETWORKFUZZER’s advantage is not tied to a particular protocol layer but generalizes to malformed inputs reaching the AMF.

Comparison with FIVGEEFUZZ. We compare NETWORKFUZZER against FIVGEEFUZZ on the Open5GS SBI over HTTP/2, targeting the SMF. Both share the same packet-processing engine and 3GPP-spec-mined foundation, so the comparison isolates NETWORKFUZZER’s issue-driven scenario construction, synthesized by mining previously reported vulnerabilities and RL-guided action selection over the schema-only FIVGEEFUZZ baseline. The budget is wall-clock ($N = 5$ trials of 7200 s each) rather than fixed-execution, so throughput and

Table 2: NETWORKFUZZER vs. 5GREPLAY on the Open5GS AMF (N2) under NGAP-envelope (Track 1) and NAS-decoder (Track 2) mutations.

Metric	5GREPLAY	NETWORKFUZZER	A_{12}	p
<i>Track 1: NGAP-envelope</i>				
Line coverage (%)	6.5 ± 0.3	6.7 ± 0.1	0.96	0.016
Function coverage (%)	11.3 ± 0.6	11.8 ± 0.1	1.00	0.008
Throughput (msg/s)	2.1×10^4	0.5	0.00	0.008
Time-to-first-crash (s)	—	15	—	—
Confirmed defects	0 (0/5)	2 (5/5)	1.00	0.008
<i>Track 2: NAS-decoder</i>				
Line coverage (%)	6.4 ± 0.3	6.7 ± 0.0	1.00	0.008
Function coverage (%)	11.3 ± 0.5	11.8 ± 0.0	1.00	0.008
Throughput (msg/s)	2.1×10^4	1.3	0.00	0.008
Time-to-first-crash (s)	—	45	—	—
Confirmed defects	0 (0/5)	2 (5/5)	1.00	0.008

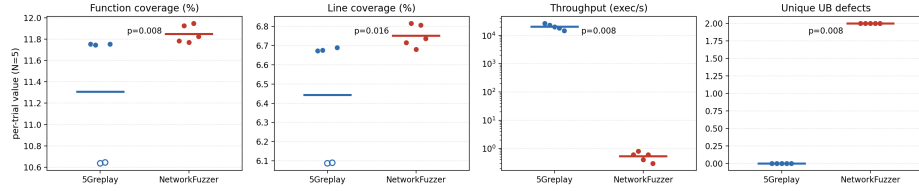


Fig. 3: Per-trial results for NETWORKFUZZER vs. 5GREPLAY on the AMF N2 interface, under NGAP-envelope mutations.

execution count are measured outcomes reported alongside coverage, time-to-first-crash, and confirmed defects.

Table 3 and Figure 4 show that the two tools achieve parity on confirmed defect discovery: both reproduce the same two remotely triggerable assertion-abort (DoS) defects in all five trials. They are also statistically indistinguishable in time-to-first-crash (median 84 vs. 135 s, $A_{12} = 0.52$), with both finding the first defect within one to two minutes. Coverage differs by at most 0.2 percentage points despite FIVGEEFUZZ’s higher throughput (2.4 vs. 1.9 msg/s) and $\sim 25\%$ more executions. Thus, NETWORKFUZZER matches FIVGEEFUZZ’s defect yield and coverage while issuing fewer requests, showing that RL-guided input generation maintains effectiveness despite its higher per-request cost.

Answer to RQ1. NETWORKFUZZER outperforms 5GREPLAY and matches the spec-guided FIVGEEFUZZ. On the AMF, it finds two UB defects in all 5/5 trials while 5GREPLAY finds none, with equal-or-higher coverage. On the SMF, it reproduces the same two confirmed defects as FIVGEEFUZZ with comparable coverage and time-to-first-crash using $\sim 25\%$ fewer requests.

5.4 RQ2: Ablation Study

We answer RQ2 on the Open5GS SMF over its SBI interface using the same target, harness, and metrics as RQ1, with five independent 2 h trials per configuration. We evaluate five configurations to isolate NETWORKFUZZER’s key com-

Table 3: NETWORKFUZZER vs. FIVGEEFUZZ on the Open5GS SMF SBI.

Metric	FIVGEEFUZZ	NETWORKFUZZER	A_{12}	p
Line coverage (%)	7.3 ± 0.1	7.5 ± 0.1	0.88	0.042
Function coverage (%)	11.9 ± 0.1	12.1 ± 0.1	0.88	0.042
Throughput (msg/s)	2.4 ± 0.1	1.9 ± 0.1	0.00	0.011
Time-to-first-crash (s)	135	84	0.52	1.00
Confirmed defects	2 (5/5)	2 (5/5)	0.50	1.00

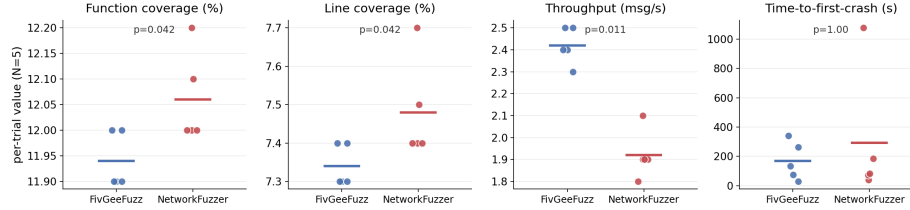


Fig. 4: Per-trial results for NETWORKFUZZER vs. FIVGEEFUZZ on the SMF SBI/HTTP-2 interface.

ponents: (1) *Full*, with RL guidance, 3GPP-spec-derived mutations, and stateful producer-consumer chaining; (2) *NoRL*, replacing RL with uniform random action selection; (3) *NoSpec*, removing spec-derived mutations; (4) *NoChains*, removing stateful chaining; and (5) *Floor*, retaining only generic mutation operators. Results are summarized in Table 4, Figure 5, Figure 6, and Figure 7.

Full achieves the highest coverage of all five configurations, reaching 7.54% line and 12.12% function coverage (Table 4), despite executing only 8685 requests compared with 14280 for *NoRL* (Figure 5). *Full* and *NoRL* share the largest action space, whereas *Floor* exposes the smallest and least diverse set of mutations (Figure 7). Removing RL has by far the largest impact: coverage drops to 7.02% line and 11.66% function, showing that RL improves exploration efficiency rather than throughput. *NoRL* also exhibits the largest variability across trials, consistent with unguided action selection. Stateful chaining is the second most important component, reducing line coverage to 7.28% when removed, whereas omitting specification-derived mutations has only a minor effect (7.50%). Overall, *Full* improves line coverage by 9.9% and function coverage by 5.0% over the *Floor* configuration, with RL accounting for the largest share of the improvement. All configurations reproduce the same two confirmed SMF assertion-abort defects within the two-hour budget. The difference lies in exploration: unguided configurations repeatedly rediscover the same shallow crashes, whereas *Full* triggers substantially fewer repeated crashes and instead spends more of its budget exploring new execution paths (Figure 6). TTFC is similar across all configurations, indicating that the benefit of RL is broader exploration rather than faster defect discovery.

Table 4: RQ2 leave-one-out ablation on Open5GS SMF (SBI).

Metric	Floor	NoChains	NoSpec	NoRL	Full (vs. Floor)
Line coverage (%)	6.86±0.50	7.28±0.19	7.50±0.06	7.02±0.59	7.54±0.14 (+9.9%)
Function coverage (%)	11.54±0.49	12.00±0.15	12.06±0.08	11.66±0.55	12.12±0.12 (+5.0%)
Throughput (msg/s)	2.08±0.15	1.38±0.33	0.64±0.43	1.98±0.15	1.22±0.40 (−41.3%)
Action-space size	1031	1456	1632	2892	2892 (+180.5%)
Proc-down events	7.2±2.3	2.8±1.9	1.4±1.5	8.2±3.5	3.8±2.2 (−47.2%)
TTFC (min)	13.8±10.1	37.5±20.2	27.7±31.5	24.4±20.3	17.7±9.8 (+28.1%)

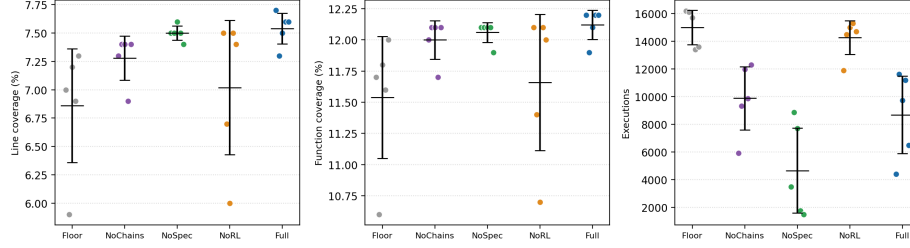


Fig. 5: Per-trial line coverage, function coverage, and executions across the five ablation arms (5 trials each, 2 h budget); black bars mark the mean±s.d.

Answer to RQ2. All three components contribute, but RL guidance has the largest impact. Removing RL causes the greatest coverage loss (*Full* achieves +7.4% higher line coverage than *NoRL*) despite executing 39% fewer requests, demonstrating substantially higher exploration efficiency. Stateful producer–consumer chaining provides the second-largest gain, while 3GPP-spec-derived mutations offer a smaller incremental benefit. Although all configurations discover the same two defects within the budget, the complete system explores more broadly and achieves the highest coverage.

5.5 RQ3: Real-World Vulnerability Discovery

Detected Vulnerabilities. Table 5 summarizes the reproducible vulnerabilities uncovered by NETWORKFUZZER in the latest releases of the evaluated 5G core implementations. It both discovers a previously unknown vulnerability and independently rediscovers five previously reported bugs, validating its effectiveness against third-party ground truth. Confirmed crash vulnerabilities were observed only in Open5GS. All vulnerabilities were confirmed and fixed by the developers.

Under the same fuzzing campaigns, free5GC and Ella Core exhibited no crashes, although NETWORKFUZZER identified an authentication-logic flaw and a goroutine panic in session-expiry handling in Ella Core. In addition, NETWORKFUZZER detected memory-growth anomalies in several Open5GS network functions, notably the UDM and PCF, suggesting potential resource-exhaustion weaknesses. One possible explanation is the difference in implementation language and runtime environment: Open5GS is implemented in C with manual

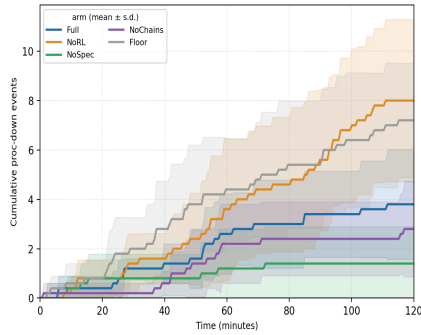


Fig. 6: Cumulative process-down events over time per arm.

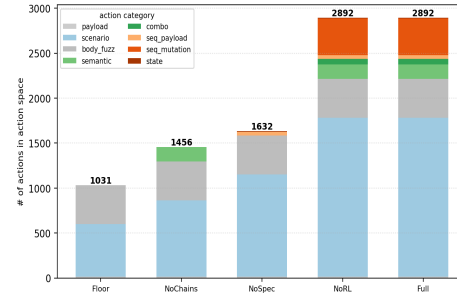


Fig. 7: Action-space composition per arm, colored by mutation category.

Table 5: Vulnerabilities confirmed by NETWORKFUZZER in the latest releases.

Core	Component	Vulnerability	Type
Open5GS	SMF, AMF, UDM	Empty/null-byte SUPI fatal assertion due to an incomplete patch (#4532 [†])	Crash
Open5GS	UDR	Malformed <code>pei</code> (missing separator) aborts <code>context-data</code> update (#4411)	Crash
Open5GS	UDR	Prefix-only <code>supi=imsi</code> aborts <code>ogs_dbi_subscription_data()</code> (#4412)	Crash
Open5GS	UDM	<code>purgeFlag:true</code> without registration state aborts UECM update (#4420)	Crash
Open5GS	NRF	Subscription-pool exhaustion via repeated <code>POST /subscriptions</code> (#4465)	Crash
Open5GS	SMF	Malformed <code>pduSessionId</code> aborts <code>smf_sess_find_by_psi()</code> (#4471)	Crash

[†] First discovered and reported by us (CVE-2026-78810).

memory management, whereas free5GC and Ella Core are implemented in Go and benefit from stronger runtime safety and error-handling mechanisms.

Detailed Analysis of an Incomplete Patch. We identify a DoS vulnerability (issue #4532³) in Open5GS v2.7.7 by leveraging knowledge from a previously patched issue in v2.7.5 (PR #3742⁴). The original fix addressed an AUSF crash caused by an empty SUPI/SUCI being used as a zero-length hash key by adding input validation at a single entry point; however, it only resolved a local code path, while similar unchecked usages persisted in other network functions, including SMF, AMF, and UDM.

Through vulnerability-knowledge mining, NETWORKFUZZER generalizes this issue into an attack pattern: *subscriber identifiers used as hash keys may fail when empty or malformed*. Guided by this pattern, NETWORKFUZZER prioritizes SBI operations involving SUPI/SUCI fields and applies boundary-value mutations, which quickly trigger an identical crash in the SMF. Further anal-

³ <https://github.com/open5gs/open5gs/issues/4532>

⁴ <https://github.com/open5gs/open5gs/pull/3742>

ysis reveals similar issues in other network functions, indicating an incomplete fix. This result highlights the value of vulnerability-knowledge mining: instead of rediscovering a known bug, NETWORKFUZZER generalizes its root cause to uncover new instances across components.

Answer to RQ3. NETWORKFUZZER identifies reproducible vulnerabilities, including crashes and memory leaks, across SMF, AMF, UDM, UDR, and NRF, with most crash-inducing defects observed in Open5GS. Vulnerability-knowledge mining further generalizes a single patched defect into an attack pattern that exposes incomplete fixes across components.

5.6 Threats to Validity

Our evaluation is subject to several limitations. First, the experiments use five independent trials per configuration, which limits statistical power. We mitigate experimental variability by using identical hardware, software environments, fuzzing budgets, and target configurations across tools. Since FIVGEEFUZZ is not publicly available, its mutation strategies were reimplemented from the published description, which may introduce implementation differences. In addition, coverage measurements depend on instrumentation and should therefore be interpreted primarily in terms of relative differences rather than absolute values.

Second, our evaluation covers three open-source 5G core implementations and focuses mainly on SBI and NGAP/NAS interfaces. The simplified local testbed does not capture all characteristics of commercial 5GC deployments, distributed environments, or radio-layer variability. Thus, the results may not directly generalize to other implementations, protocols, or production environments.

6 Conclusion & Future Work

We present NETWORKFUZZER, a reinforcement-learning-guided fuzzer for 5GC networks that combines RL-guided action selection, 3GPP-spec-derived semantic mutations, and stateful producer-consumer message chaining. NETWORKFUZZER efficiently explores complex protocol states while adapting its strategy during execution. Across open-source 5GC implementations, NETWORKFUZZER outperforms replay-based fuzzing and matches the spec-guided baseline, achieving higher coverage and consistently reproducing confirmed defects. Future work includes evaluating additional protocols, exploring advanced AI techniques [15,18], including LLM-driven protocol reasoning [16], and integrating AI-driven defensive mechanisms [20] for secure 5G/6G networks.

Acknowledgement

This work was supported by the H2020 project INTACT, NETWORK, RESILMESH, and SECASSURED projects, funded by EU’s Horizon Europe (HE) research and innovation programme respectively under the HE Grant Agreement Numbers 10126241, 101139285, 101119681, and 101225858. Views and opinions expressed are, however those of the authors only and do not necessarily reflect those of the EU or HE. Neither the EU nor the granting authority can be held responsible for them.

References

1. Aflplusplus: The afl++ fuzzing framework (2026), <https://aflplusplus.com/>
2. Ella core (2026), <https://github.com/ellanetworks/core>
3. free5gc (2026), <https://github.com/free5gc/free5gc>
4. Open5gs (2026), <https://github.com/open5gs/open5gs>
5. Openapi specification files for 3gpp 5g core network (2026), https://forge.3gpp.org/rep/all/5G_APIs
6. Stable baselines3 (2026), <https://github.com/DLR-RM/stable-baselines3>
7. Böhme, M., Pham, V.T., Nguyen, M.D., Roychoudhury, A.: Directed greybox fuzzing. CCS '17 (2017)
8. Caccavale, F.G., Nguyen, H.N., Cavalli, A., Montes De Oca, E., Mallouli, W.: Http/2 attacks generation using 5greplay. In: Proceedings of the 18th International Conference on Availability, Reliability and Security (2023)
9. Chen, A., Preatoni, R., Brighente, A., Conti, M., Nita-Rotaru, C.: Cross-service token: Finding attacks in 5g core networks. CoRR **abs/2509.08992** (2025)
10. Cheng, M., Zhu, K., Chen, Y., Lu, Y., Chen, C., Yu, J.: Reinforcement learning-based multi-phase seed scheduling for network protocol fuzzing. Electronics **13**(24), 4962 (2024). <https://doi.org/10.3390/electronics13244962>
11. Giapantzis, K., Zarras, A., Kalafatidis, S., Lalas, A., Votis, K., Xenakis, C.: GORGO: A reinforcement learning model for dos vulnerability analysis on 5g and beyond networks. Journal of Network and Systems Management **34**, 10 (2026). <https://doi.org/10.1007/s10922-025-09980-7>
12. He, F., Yang, W., Cui, B., Cui, J.: Intelligent fuzzing algorithm for 5g nas protocol based on predefined rules. In: 2022 International Conference on Computer Communications and Networks (ICCCN). pp. 1–7 (2022). <https://doi.org/10.1109/ICCCN54977.2022.9868872>
13. Mancini, F., Da Canal, S., Bianchi, G.: AMFuzz: Black-box fuzzing of 5g core networks. In: 2024 19th Wireless On-Demand Network Systems and Services Conference (WONS). pp. 17–24 (2024). <https://doi.org/10.23919/WONS60642.2024.10449510>
14. Nakas, G., Radoglou-Grammatikis, P., Amponis, G., Lagkas, T., Argyriou, V., Goudos, S., Sarigiannidis, P.: 5G-Fuzz: An attack generator for fuzzing 5gc, using generative adversarial networks. In: 2023 IEEE Globecom Workshops (GC Wkshps). pp. 347–352. Kuala Lumpur, Malaysia (2023). <https://doi.org/10.1109/GCWkshps58843.2023.10464928>
15. Nguyen, H.N., Nguyen, M.D., Montes de Oca, E.: A framework for in-network inference using p4. ARES '24
16. Nguyen, L., Nguyen, M.D., Bui, A.H., Cavalli, A.R., de Oca, E.M.: Llms in security testing and monitoring: An initial study. In: International Conference on Availability, Reliability and Security
17. Nguyen, M.D., Bardin, S., Bonichon, R., Groz, R., Lemerre, M.: Binary-level directed fuzzing for Use-After-Free vulnerabilities. In: 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)
18. Nguyen, M.D., Bouaziz, A., Valdes, V., Rosa Cavalli, A., Mallouli, W., Montes De Oca, E.: A deep learning anomaly detection framework with explainability and robustness. In: Proceedings of the 18th International Conference on Availability, Reliability and Security (2023)
19. Nguyen, M.D., La, V.H., Cavalli, A.R., De Oca, E.M.: Towards improving explainability, resilience and performance of cybersecurity analysis of 5g/iot networks

- (work-in-progress paper). In: 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)
20. Nguyen, M.D., Mallouli, W., Cavalli, A.R., Montes de Oca, E.: Ai4soar: A security intelligence tool for automated incident response. ARES '24 (2024)
 21. Nguyen, M.D., Méndez Hurtado, M., Benito-Parejo, M., Cavalli, A.R., de Oca, E.M.: Autonomous ai-driven pentesting agent for healthcare systems security. In: International Conference on Availability, Reliability and Security. Springer (2026)
 22. Nguyen, M.D., Nguyen, H.N., de Souza, L.M., Karagiannis, S., Cavalli, A.R., de Oca, E.M.: An intelligent network fuzzer for protocol testing in healthcare systems. In: Availability, Reliability and Security (2025)
 23. Parikh, D.P., Dessources, D.A., Tripathi, N.D., Reed, J.H., Burger, E.W.: Reinforcement learning-based fuzzer for 5g rrc security evaluation. IEEE Access **14**, 38264–38274 (2026). <https://doi.org/10.1109/ACCESS.2026.3672003>
 24. Pham, V.T., Böhme, M., Roychoudhury, A.: AFLNet: A greybox fuzzer for network protocols. In: 2020 IEEE 13th International Conference on Software Testing, Verification and Validation (ICST). pp. 460–465 (2020). <https://doi.org/10.1109/ICST46399.2020.00062>
 25. Salazar, Z., Nguyen, H.N., Mallouli, W., Cavalli, A.R., Montes de Oca, E.: 5greplay: A 5g network traffic fuzzer-application to attack injection. In: Proceedings of the 16th International Conference on Availability, Reliability and Security (2021)
 26. Shen, Y., Liu, Y., Zhou, Y.: RLGfuzz: Reinforcement learning guided fuzzing with state-coverage mapping environment. In: 2024 IEEE 32nd International Conference on Network Protocols (ICNP) (2024). <https://doi.org/10.1109/ICNP61940.2024.10858502>
 27. Sun, Y., Liu, X., Sun, Q., Wang, J., Tian, L., Liu, J.: 5GC-Fuzz: Finding deep stateful vulnerabilities in 5g core network with black-box fuzzing. In: IEEE INFOCOM 2025 – IEEE Conference on Computer Communications. pp. 1–10 (2025). <https://doi.org/10.1109/INFOCOM55648.2025.11044489>