

Gap-Targeted LLM Test Generation for IoT Security: Discovering Novel Vulnerabilities Beyond AutoML-Prioritized Registries

Gabriel Pivoto Ambrósio, Samuel Baraldi Mafra, and Christopher de Souza Lima Francisco

Abstract—Internet of Things (IoT) security test suites are typically drawn from static registries, which caps the attack surface that automated prioritization can explore. We propose a gap-targeted hybrid pipeline coupling Automated Machine Learning (AutoML) test prioritization with Large Language Model (LLM) generated tests conditioned on coverage gaps detected in execution history. Across 15 cells (5 frameworks $\times$ 3 modes) on 13 Dockerized IoT devices spanning seven protocols, LLM-authored tests discovered 17 exclusive scenarios: attack paths never exercised by the registry, spanning six vulnerability categories mapped to the Common Weakness Enumeration (CWE). The approach complements existing security tooling.

Keywords—IoT security, LLM test generation, AutoML, protocol coverage gaps, hybrid testing, CWE.

I. INTRODUCTION

IoT deployments are inherently multi-protocol: a single environment may simultaneously expose Hypertext Transfer Protocol (HTTP) management interfaces, Message Queuing Telemetry Transport (MQTT) brokers, Constrained Application Protocol (CoAP) sensor endpoints, Modbus over Transmission Control Protocol (TCP) control registers, Secure Shell (SSH) administration, and File Transfer Protocol (FTP) firmware-update channels. Their security assessment relies on curated *test registries*, which are catalogs of protocol-specific probes maintained by domain experts [1]. AutoML-based prioritization [2], [3] improves execution efficiency by scoring registry items from execution history. However, this prioritization is bounded by the curated catalog itself. The system can only rank existing tests, so protocols with few registry templates remain permanently under-tested, regardless of scorer quality.

LLMs generate syntactically valid and semantically plausible security probe code from protocol context [4], [5], [6], but most evaluations target general-purpose software and

measure line or branch coverage. Empirical evidence for IoT *security* test generation, where protocol semantics and fault oracles differ substantially from unit testing, is still limited. Traditional security tools such as protocol fuzzers (e.g., Boofuzz [7]) synthesize byte-level mutations effectively but rarely encode the semantic intent behind authentication, resource-exhaustion, or access-control scenarios specific to IoT stacks. LLM-based generation occupies a different niche: translating protocol knowledge into targeted, human-readable test code for scenarios the registry omits.

This paper investigates the following hypothesis pair regarding the effectiveness of LLM-generated tests in IoT security:

H_1 : *LLM-generated tests uncover IoT vulnerabilities that an AutoML-prioritized static registry cannot reach.*

H_0 : *LLM-generated tests uncover no vulnerabilities beyond those reachable by the AutoML-prioritized static registry.*

We make three contributions: (i) a gap-targeted hybrid pipeline combining an AutoML scorer, an LLM generator, and an Abstract Syntax Tree (AST)-based safety validator; (ii) an *exclusive-vulnerability* metric grounded in CWE identifiers; and (iii) an empirical evaluation over approximately 243,000 test executions that rejects H_0 in all 15 experimental cells under Holm–Bonferroni correction, supporting H_1 . The pipeline is implemented in *Emergence*, our IoT security testing framework, which hosts the registry, the AutoML scorer, the LLM generator, the safety validator, the Docker testbed, and an analyst-facing dashboard.

II. BACKGROUND AND RELATED WORK

A. Common Weakness Enumeration (CWE)

We refer to vulnerability categories using the Common Weakness Enumeration (CWE) [8], a community-maintained MITRE catalog that indexes software and hardware weaknesses with stable, vendor-neutral identifiers (e.g., CWE-287 for improper authentication, CWE-22 for path traversal, CWE-200 for information exposure). CWE identifiers allow us to compare findings across protocols and tools and map cleanly to public Common Vulnerabilities and Exposures (CVE) entries.

Gabriel Pivoto Ambrósio, Instituto Nacional de Telecomunicações, Santa Rita do Sapucaí-MG, e-mail: gabriel.pivoto@inatel.br; Samuel Baraldi Mafra, Instituto Nacional de Telecomunicações, Santa Rita do Sapucaí-MG, e-mail: samuelbmafra@inatel.br; Christopher de Souza Lima Francisco, Instituto Nacional de Telecomunicações, Santa Rita do Sapucaí-MG, e-mail: christopher@inatel.br. This work was partially funded by CNPq (Grant Nos. 403612/2020-9, 311470/2021-1, and 403827/2021-3), by FAPEMIG (Grant Nos. PPE-00124-23, APQ-04523-23, APQ-05305-23, and APQ-03162-24), by the Brasil 6G project (01245.020548/2021-07), supported by RNP and MCTI, and by XGM-AFCCT-2024-5-1-2, supported by xGMobile – EMBRAPII-Inatel Competence Center on 5G and 6G Networks (MCTI grant 052/2023, PPI IoT/Manufatura 4.0).

B. IoT protocol security profiles

Security exposures differ across IoT protocols, and so does the maturity of the tooling that covers them. We review the seven protocols of the testbed in decreasing order of registry maturity, since that ordering anticipates where a generative component can contribute.

At one extreme, HTTP and HTTPS [9] are the best-documented protocols in the security literature: common weaknesses include improper authentication (CWE-287), path traversal (CWE-22), and information exposure (CWE-200), and registry coverage is typically saturated.

MQTT, Modbus/TCP, SSH and Telnet occupy an intermediate position. MQTT (OASIS v5.0 [10]) concentrates vulnerabilities in anonymous broker access (CWE-306), unauthorized topic subscription (CWE-284), and payload injection, and its compact packet format constrains the generative space. Modbus/TCP [13] has minimal authentication primitives, with weaknesses in register authorization (CWE-862) and function-code enumeration (CWE-200). SSH weaknesses concentrate in cryptographic policy (CWE-326) and brute-force hardening (CWE-307), while Telnet transmits credentials in plaintext (CWE-319).

CoAP and FTP sit at the opposite extreme, and it is there that the registry is thinnest. CoAP (Request for Comments (RFC) 7252 [11]) targets resource-constrained nodes; its Observe extension (RFC 7641 [12]) opens a resource-exhaustion vector via persistent subscriptions (CWE-400), semantically distinct from HTTP and underrepresented in general-purpose registries. FTP (RFC 959 [14]) is a legacy protocol with documented issues: bounce attacks (CWE-441, CVE-1999-0017), banner disclosure (CWE-200), cleartext credential transmission (CWE-319), and path traversal via CWD (CWE-22). Since modern deployments trend toward SFTP, FTP registries tend to be thin, even though the corresponding attack patterns are thoroughly documented. This asymmetry between documented risk and available tooling is the gap our generation loop exploits.

C. LLM-based test generation

Recent work investigates LLMs for software test generation, especially in agentic and search-hybrid designs. TEST-FORGE [15] couples an LLM agent with feedback-driven refinement; EVOGPT [16] integrates LLM-synthesized tests as diversified seeds for evolutionary search, helping it escape coverage plateaus; HITS [5] applies method-level slicing to address the context-window problem; and CHATUNITEST [4] and Ryan et al. [6] both show that structured context materially improves test quality. A second line of work catalogues practical limits: test smells [17], version sensitivity [18], brittleness under software evolution [19], and few-shot enhancement [20]. Guilherme and Vincenzi [21] provided an early ChatGPT baseline. None of these works target IoT-protocol security, and none define novelty relative to a curated registry.

D. Novelty metrics for security testing

Coverage-guided fuzzers such as AFL [22] measure novelty by counting new control-flow edges. Our *exclusive-*

vulnerability metric is complementary: instead of measuring where tests *run*, it measures where tests *succeed*. A test that exercises a new code path but finds no vulnerability does not satisfy the criterion; a test that activates a known CWE on a previously untested attack path does.

III. HYBRID TESTING ARCHITECTURE

A. System overview

The architecture described below is the test-generation core of Emergence; the dashboard layer is illustrated later in Figure 2. The pipeline (Figure 1) is organized as two coordinated loops sharing a common log file, `history.csv`. The *prioritization loop* runs every iteration: the AutoML scorer reads the registry and recent execution history, ranks the tests, and the Docker-isolated scanner executes the top-ranked subset. Each result is appended to `history.csv` with `test_origin=registry`.

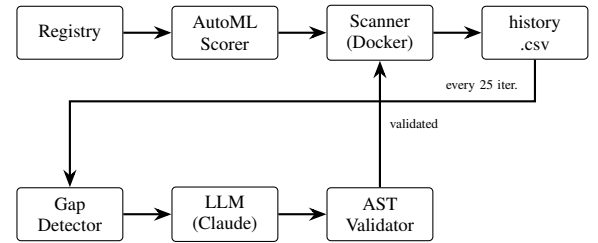


Fig. 1. Hybrid pipeline. The top row is the prioritization loop: Registry → AutoML → Scanner → `history.csv`. The bottom row is the generation loop, fired every 25 iterations: `history.csv` → Gap Detector → LLM → AST Validator. The validated tests feed back into the Scanner, closing the loop.

The *generation loop* runs every 25 iterations. The *gap detector* identifies under-served areas using two criteria: *low-detection protocols*, namely protocol × test-type groups whose rolling detection rate falls below 20% over the last 50 executions; and *underrepresented categories*, namely vulnerability types with fewer than three distinct successful test identifiers. These thresholds are configuration parameters rather than values from a published standard: they were fixed empirically in pilot runs as the least aggressive setting that triggered generation on the thin FTP and CoAP registries without firing on the saturated HTTP one. Detected gaps are assembled into a structured JavaScript Object Notation (JSON) prompt, shown in Listing 1. The LLM returns `pytest` functions. An AST-based validator enforces a strict module allowlist (socket, requests, paramiko, paho.mqtt, aiocoap, pymodbus, ssl, hashlib, pytest) and rejects dangerous imports (os, subprocess, shutil, sys, ctypes) before any generated test reaches the scanner. Finally, the validated tests are tagged `test_origin=llm` and added to the execution pool.

B. Prompt design and provider choice

The system prompt fixes the `pytest` conventions that make the output machine-checkable, and its allowed-library list is kept identical to the AST allowlist so that prompt and validator cannot drift apart. The user prompt is rebuilt

```
[SYSTEM] You are an expert IoT security researcher and
penetration tester. Generate novel pytest security tests
for IoT devices in a controlled Docker lab. OWASP IoT
Top 10 is the reference framework. Conventions:
- function test_<protocol>_<vuln_type>_<id>, decorated
  with @pytest.mark.vuln_id("<unique_id>")
- assert True => vulnerability confirmed;
  pytest.fail() => not found; timeout = 5 s
- docstring explaining the vulnerability; handle
  ConnectionError / TimeoutError gracefully
Allowed: requests, paramiko, paho.mqtt, aiocoap, pymodbus,
socket, ssl, struct, json, base64, hashlib. Do NOT use os,
subprocess, shutil, sys or file writes: network only.

[USER] Generate {max_tests} tests for {device_ip}, ports
{open_ports}, protocols {protocols}. Target a class NOT
covered by {existing_test_ids}. Return JSON with fields
test_id, test_name, pytest_code, vulnerability_type,
severity, references.
```

Listing 1: Condensed system and user prompts. Braces denote values injected at runtime from the gap detector.

at every invocation from the current device state and from the identifiers already in the registry, which is what makes generation gap-targeted rather than generic. We selected Anthropic Claude over OpenAI GPT and Google Gemini for three reasons: the tool-use Application Programming Interface (API) enforces a JSON schema at the API boundary, which reduces parsing failures; the 200,000-token context window accommodates the cumulative list of existing test identifiers; and in pilot runs Claude produced higher AST-validator pass rates. The pipeline is provider-agnostic via an adapter pattern, so a multi-provider comparison is feasible follow-up work.

C. Exclusive-vulnerability metric

Let $\mathcal{T}_L$ (LLM-generated) and $\mathcal{T}_R$ (registry) partition the executed test set; LLM identifiers are name-spaced (`llm_*`), so $\text{id}(\mathcal{T}_L) \cap \text{id}(\mathcal{T}_R) = \emptyset$. Let $V(t) \in \{0, 1\}$ be the vulnerability-found outcome. Define:

$$\text{ExclVuln} = |\{\text{id}(t) : t \in \mathcal{T}_L, V(t) = 1\} \setminus \{\text{id}(t) : t \in \mathcal{T}_R, V(t) = 1\}|. \quad (1)$$

A positive value indicates a vulnerability reachable only via an LLM-authored scenario absent from the registry. We ground ExclVuln categories in CWE for interpretability. We reject H_0 iff $\text{ExclVuln} > 0$ or the Fisher one-sided $p < 0.05$ with LLM rate $>$ registry rate. The disjunction is intentional: a lower aggregate LLM detection rate does not preserve H_0 , since gap-targeted generation probes rarer, previously unexercised attack paths.

D. Statistical tests

Per cell, we compute Fisher’s exact test on the 2×2 table (LLM/registry) $\times$ (vuln/no-vuln); the odds ratio (OR) with Woolf logit 95% confidence interval (CI); and the per-iteration Mann–Whitney U with rank-biserial r . Family-wise error across the 15 Fisher tests is controlled by Holm–Bonferroni at $\alpha=0.05$.

IV. EXPERIMENTAL SETUP

The testbed comprises thirteen Dockerized IoT devices exposing the seven protocols of Section II-B, with vulnerabilities seeded from public CVE entries so that the environment mirrors failure modes reported for real-world IoT stacks. The registry prioritized by the AutoML scorer holds 52 templates,

distributed as HTTP (17), SSH (8), MQTT (7), Modbus (7), CoAP (6), Telnet (4) and FTP (3); that distribution reflects the documentation investment each protocol has received publicly, and it is what leaves FTP and CoAP structurally under-served.

Five AutoML frameworks were selected to span the dominant paradigms of the field rather than a single search strategy: H2O [3] and AutoGluon for stacked ensembles, PyCaret for low-code scikit-learn pipelines, TPOT for tree-based genetic programming, and auto-sklearn for meta-learning with Bayesian optimization [2]. Because they optimize in structurally different ways, an effect observed across all five is unlikely to be an artifact of one procedure.

Environmental dynamics follow three simulation modes in which every event is an independent Bernoulli trial per iteration. The *deterministic* mode leaves containers as built (all probabilities zero) and acts as the control. The *medium* mode combines an 8% service-outage probability with 3% patching, 3% credential rotation and 5% patch regression, perturbing the oracle with 1% false positives and 5% false negatives. The *realistic* mode keeps outages at 8% but raises patching and rotation to 5% and regression to 10%, with the same oracle noise. These values reproduce the qualitative behaviour observed in our own laboratory and are design parameters of the study rather than figures taken from a published benchmark. Crossing five frameworks with three modes over 100 iterations at a fixed seed of 42 yields the 15 cells of Section V, with the LLM invoked every 25 iterations for at most ten candidates per call. The corpus totals 242,900 executions, $n_L=29,400$ LLM-authored and $n_R=213,500$ from the registry.

V. RESULTS

A. Per-cell evidence

Table I reports the 15-cell analysis. Each row is the proposed pipeline instantiated with a different AutoML backend, so the framework column identifies a configuration of the proposed method and not a competing baseline; the comparison in every cell is between the LLM-generated and registry populations produced by that same pipeline. The aggregate registry detection rate (52%) far exceeds the LLM rate (12%): odds ratios range from 0.06 to 0.60, all below 1 and all Holm-significant. This pattern is expected, since coverage-driven generation focuses on rarer and unserved attack paths rather than on high-yield scenarios already in the curated catalog. Although LLM tests show a lower per-test detection rate, they target unexplored regions of the attack surface, so the two populations are complementary rather than competitive. What matters for H_1 is that every cell has $\text{ExclVuln} > 0$: LLM tests confirm vulnerabilities on scenarios the registry never attempts, so H_0 is rejected in all 15 cells via the exclusivity criterion, supporting H_1 . Figure 2 shows the corresponding panel of the Emergence dashboard for the H2O $\times$ deterministic cell.

B. What the LLM discovered

Across the 15 cells the LLM produced 20 distinct test identifiers, of which 17 are exclusive: nine on FTP and eight

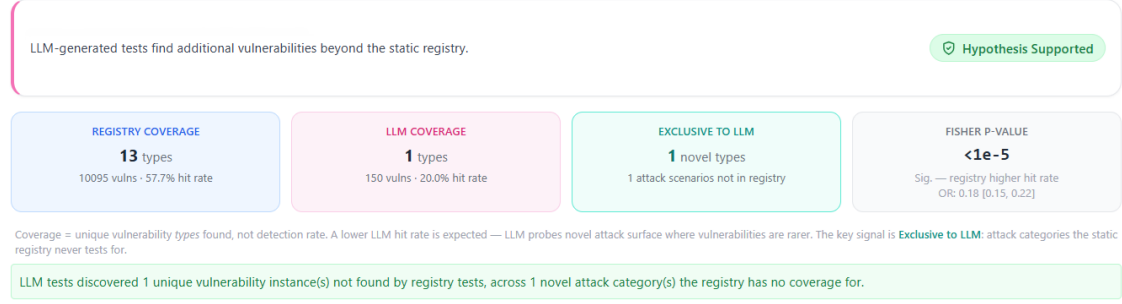


Fig. 2. H_1 verdict panel of the Emergence dashboard for the H2O $\times$ deterministic cell, illustrating how the criterion of Eq. (1) is surfaced operationally rather than comparing methods. Registry coverage is 13 vulnerability types at a 57.7% hit rate; LLM coverage is 1 type at a 20.0% hit rate. The *Exclusive to LLM* card flags one attack scenario absent from the registry, matching ExID=1 in Table I, line 7.

TABLE I

H_1 PER-CELL EVIDENCE (PHASE 3, $n_{\text{iter}}=100$, SEED=42). EVERY ROW IS THE PROPOSED PIPELINE WITH A DIFFERENT AUTOML BACK-END. OR = ODDS RATIO LLM VS. REGISTRY; p_H = HOLM-ADJUSTED FISHER p ; EXID = EXCLUSIVE VULNERABLE SCENARIOS.

Framework	Mode	n_L	n_R	OR	p_F	p_H	ExID
AutoGluon	det.	750	14000	0.11	9.8e-134	5.9e-133	2
AutoGluon	med.	1700	14000	0.10	8.8e-255	7.0e-254	8
AutoGluon	real.	750	14000	0.23	2.1e-67	8.3e-67	4
auto-skl.	det.	2500	14000	0.10	<1e-300	<1e-300	3
auto-skl.	med.	3300	14000	0.12	<1e-300	<1e-300	11
auto-skl.	real.	4100	14000	0.11	<1e-300	<1e-300	9
H2O	det.	750	17500	0.18	5.9e-95	2.9e-94	1
H2O	med.	750	14000	0.60	3.4e-11	3.4e-11	8
H2O	real.	750	14000	0.38	3.9e-34	1.2e-33	7
PyCaret	det.	1700	14000	0.11	7.0e-286	6.3e-285	2
PyCaret	med.	2500	14000	0.11	<1e-300	<1e-300	9
PyCaret	real.	3300	14000	0.08	<1e-300	<1e-300	11
TPOT	det.	4100	14000	0.06	<1e-300	<1e-300	2
TPOT	med.	750	14000	0.40	4.2e-31	8.4e-31	7
TPOT	real.	1700	14000	0.20	3.2e-159	2.3e-158	8

on CoAP. Table II groups them into six CWE categories. Exclusivity is decided per identifier by Eq. (1), and the CWE label is the category recorded when that identifier confirmed a vulnerability, so the counts are outcomes of the executed suite rather than a manual classification.

TABLE II

LLM-EXCLUSIVE VULNERABILITY CATEGORIES WITH CWE MAPPING.
ALL 17 EXCLUSIVE SCENARIOS ARE ABSENT FROM THE REGISTRY;
DETECTION COUNTS ARE AGGREGATED OVER THE 15 CELLS.

CWE	Category	Representative scenario	Detects
CWE-200	Information disclosure	llm_ftp_banner_* (3 var.)	4375
CWE-400	Denial of service	llm_coap_observe_flood_* (4 var.)	524
CWE-441	Auth. bypass (bounce)	llm_ftp_bounce_attack	19
CWE-319	Cleartext transmission	llm_ftp_cleartext_cred_tx	72
CWE-22	Path traversal	llm_ftp_path_trav_cwd	3
CWE-918	SSRF	llm_ftp_pasv_ssrf	1

For *FTP*, the registry encodes only authentication (CWE-287). The LLM adds nine exclusive scenarios spanning banner disclosure (CWE-200), the bounce attack via RFC 959 PASV misuse (CWE-441, the CVE-1999-0017 pattern), cleartext credential exposure (CWE-319), traversal via CWD (CWE-22), and Server-Side Request Forgery (SSRF) via PASV

redirection (CWE-918), moving coverage from three scenarios to twelve. For *CoAP*, the registry covers authentication and misconfiguration; the LLM adds eight exclusive scenarios, four Observe-based resource-exhaustion variants (CWE-400/CWE-770, 524 detections) grounded in RFC 7641 semantics and four unauthorized resource-enumeration tests (CWE-284), moving coverage from six scenarios to fourteen.

C. Anatomy of an LLM-generated test

Listing 2 reproduces `llm_ftp_bounce_attack` (19 confirmed detections). The test opens a control channel on port 21, authenticates as anonymous, and issues a `PORT` command pointing at an internal IP address rather than the client. RFC 959 mandates rejecting such redirection; daemons that accept it become a usable proxy into internal infrastructure (CVE-1999-0017). The registry contains no equivalent probe; the LLM recovered the protocol semantics from training, and the AST validator admitted the code because its imports are on the allowlist.

VI. DISCUSSION

The pooled odds ratio of 0.13 is, at first sight, unflattering for the LLM, but this reading conflates how often a test succeeds with the kind of vulnerability it can expose. The gap detector deliberately routes the LLM toward areas where the registry already underperforms, so a lower per-test rate is the expected operating regime, not a failure. The exclusive count captures the metric that matters for novelty: 17 of 20 LLM-authored scenarios reveal attack paths that the registry does not encode.

```
# llm_ftp_bounce_attack -- CWE-441, CVE-1999-0017 pattern
import socket, pytest
def test_ftp_bounce(target_ip):
    s = socket.create_connection((target_ip, 21), 3)
    s.recv(1024)
    s.sendall(b"USER anonymous\r\n"); s.recv(1024)
    s.sendall(b"PASS guest@\r\n"); s.recv(1024)
    s.sendall(b"PORT 10,0,0,1,0,25\r\n") # internal host
    reply = s.recv(1024)
    assert not reply.startswith(b"200"), \
        f"FTP bounce accepted: {reply!r}"
```

Listing 2: LLM-generated FTP bounce test (CWE-441), exclusive to the generation loop and absent from the curated registry.

The 17 exclusive scenarios are concentrated entirely on FTP and CoAP, the two protocols with the fewest registry templates (3 and 6). HTTP, MQTT, Modbus, SSH, and Telnet, all with seven or more templates, contribute zero LLM exclusives. We therefore treat the protocol template count as a reasonable predictor of LLM contribution: a team can decide where to invest the LLM API budget by inspecting their registry beforehand.

The proposed approach is complementary to established tooling. Byte-level fuzzers such as Boofuzz [7] excel at synthesizing malformed packets and surfacing memory-safety bugs. The LLM-based generator encodes RFC-defined state machines and CVE-documented attack patterns into readable `pytest` code targeting specific scenarios. Neither approach subsumes the other.

The evaluation does not benchmark the pipeline against fuzzers, random generation, or untargeted LLM generation. This is a deliberate scoping decision: the study isolates the marginal contribution of gap-targeted generation on top of an AutoML-prioritized registry, whereas a four-way comparison would require matched budgets, a common oracle and per-tool tuning, shifting the question to a tool ranking. Ablating the gap detector against untargeted generation is the most immediate extension. Several exclusive findings, notably the FTP bounce attack, correspond to long-public vulnerabilities. The discovery is nonetheless genuine from the pipeline’s perspective, since the pattern is in no registry template. What this paper does not demonstrate is that LLMs surface zero-day-class behaviors.

VII. THREATS TO VALIDITY

Three classes of threat deserve comment. Regarding construct validity, `ExclVuln` counts test identifiers rather than root causes, so two identifiers could probe the same fault; grouping by CWE, as in Table II, collapses the 17 identifiers into six categories and bounds the largest possible inflation of the novelty count, and a formal CVE-level mapping is in progress. Regarding internal validity, a single seed (42) was used in every cell, so within-mode variance cannot be estimated from the present data. Regarding external validity, one LLM provider and a fixed prompt family limit generalization, and although the thirteen containers were seeded from public CVE entries to be representative of real-world IoT stacks, physical devices with heterogeneous firmware may exhibit different protocol-gap profiles; a multi-vendor hardware evaluation remains future work.

VIII. CONCLUSIONS AND FUTURE WORK

We presented a gap-targeted hybrid pipeline coupling AutoML test prioritization with LLM-generated tests guided by coverage-gap detection. Across 15 cells on 13 IoT devices, the pipeline discovered 17 exclusive scenarios spanning six CWE categories, all confirmed under Holm–Bonferroni correction. The exclusive-vulnerability metric, grounded in CWE rather than raw code coverage, provides a principled way to measure novelty in security-specific test generation. A practical take-away is the role of *registry thinness* as a predictor: protocols

with few curated templates (FTP with 3, CoAP with 6) yield the highest LLM return, while saturated protocols (HTTP with 17) benefit little.

Several directions remain open. We plan to extend the evaluation to additional LLM providers (OpenAI GPT, Google Gemini) and to physical IoT devices with diverse firmware, add CVE-level mapping to the exclusive-vulnerability metric, ablate the gap detector to quantify its marginal contribution over untargeted generation, and investigate whether LLM-driven generation can surface zero-day-class behaviors beyond reproducing known patterns.

REFERENCES

- [1] OWASP Foundation, “OWASP Internet of Things Top 10,” 2018. <https://owasp.org/www-project-internet-of-things/>
- [2] M. Feurer, A. Klein, K. Eggenberger, J. T. Springenberg, M. Blum, and F. Hutter, “Efficient and Robust Automated Machine Learning,” *Advances in Neural Information Processing Systems 28 (NeurIPS)*, pp. 2962–2970, 2015.
- [3] E. LeDell and S. Poirier, “H2O AutoML: Scalable Automatic Machine Learning,” in *Proc. 7th ICML Workshop on Automated Machine Learning (AutoML)*, 2020.
- [4] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, “ChatUniTest: A Framework for LLM-Based Test Generation,” in *Companion Proc. 32nd ACM Int. Conf. Foundations of Software Engineering (FSE)*, pp. 572–576, 2024.
- [5] Z. Wang, K. Liu, G. Li, and Z. Jin, “HITS: High-Coverage LLM-Based Unit Test Generation via Method Slicing,” in *Proc. 39th IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, pp. 1258–1268, 2024.
- [6] G. Ryan, S. Jain, M. Shang, S. Wang, X. Ma, M. K. Ramanathan, and B. Ray, “Code-Aware Prompting: A Study of Coverage-Guided Test Generation in Regression Setting using LLM,” *Proc. ACM on Softw. Eng.*, vol. 1, no. FSE, 2024.
- [7] J. Pereyda, “Boofuzz: Network Protocol Fuzzing for Humans,” 2020. <https://github.com/jtpereyda/boofuzz>
- [8] The MITRE Corporation, “CWE – Common Weakness Enumeration,” 2024. <https://cwe.mitre.org/>
- [9] OWASP Foundation, “OWASP Top 10,” 2021. <https://owasp.org/www-project-top-ten/>
- [10] OASIS, “MQTT Version 5.0,” OASIS Standard, Mar. 2019.
- [11] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” IETF RFC 7252, Jun. 2014.
- [12] K. Hartke, “Observing Resources in the Constrained Application Protocol (CoAP),” IETF RFC 7641, Sep. 2015.
- [13] Modbus Organization, “MODBUS Application Protocol Specification V1.1b3,” Apr. 2012.
- [14] J. Postel and J. Reynolds, “File Transfer Protocol (FTP),” IETF RFC 959, Oct. 1985.
- [15] K. Jain and C. Le Goues, “TestForge: Feedback-Driven, Agentic Test Suite Generation,” *arXiv:2503.14713*, 2025.
- [16] L. Broide, R. Stern, and A. Mordoch, “EvoGPT: Leveraging LLM-Driven Seed Diversity to Improve Search-Based Test Suite Generation,” *arXiv:2505.12424*, 2025.
- [17] W. C. Ouédraogo, Y. Li, X. Dang, X. Tang, A. Koyuncu, J. Klein, D. Lo, and T. F. Bissyandé, “Test Smells in LLM-Generated Unit Tests,” *arXiv:2410.10628*, 2024.
- [18] M. Konstantinou, R. Degiovanni, and M. Papadakis, “How Well LLM-Based Test Generation Techniques Perform with Newer LLM Versions?,” *arXiv:2601.09695*, 2026.
- [19] S. Haroon, M. T. Khan, and M. A. Gulzar, “Evaluating LLM-Based Test Generation Under Software Evolution,” *arXiv:2603.23443*, 2026.
- [20] A. Chudic and G. Çalık, “Automated Test Suite Enhancement Using Large Language Models with Few-Shot Prompting,” *arXiv:2602.12256*, 2026.
- [21] V. Guilherme and A. Vincenzi, “An Initial Investigation of ChatGPT Unit Test Generation Capability,” in *Proc. 8th Brazilian Symp. Systematic and Automated Software Testing (SAST)*, pp. 15–24, 2023.
- [22] M. Zalewski, “American Fuzzy Lop (AFL),” 2015. <https://lcamtuf.coredump.cx/afl/>