DAST-AI: Reducing False Positives in API Security Testing via LLM-based Semantic Analysis

Gabriel F. R. de Melo[1], **Divanilson R. Campelo**[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brazil

{gfrm,dcampelo}@cin.ufpe.br

Abstract. *Dynamic Application Security Testing (DAST) is a critical methodology for identifying vulnerabilities in web applications. However, traditional DAST tools suffer from high false positive rates, causing alert fatigue among security teams. This issue is particularly acute in RESTful APIs regarding authorization flaws—such as Broken Object Level Authorization (BOLA) and missing authentication—where distinguishing between a legitimately public endpoint and a vulnerable one requires semantic context. This paper introduces DAST-AI, a hybrid security testing framework that integrates Large Language Models (LLMs) as a post-execution semantic oracle to analyze HTTP response payloads and determine whether sensitive data is improperly exposed. The evaluation is scoped to the zero-credential black-box scenario—testing APIs without any prior authentication—applicable to external security assessments and unauthenticated CI/CD pipelines. A controlled ablation study against two deliberately vulnerable benchmarks—OWASP Juice Shop (Node.js, 68 endpoints) and VAmPI (Python/Flask, 12 endpoints)—isolates the contribution of the LLM semantic oracle: removing it yields a syntax-based baseline (F1=0.49 on Juice Shop; F1=0.857 on VAmPI), while adding it produces DAST-AI (F1=0.80; F1=0.923), reducing false positives by 86.4% and 50.0% respectively, at a negligible cost of $0.000037 per endpoint. An authenticated reference comparison—simulating the ZAP Access Control Scanner methodology with valid credentials—scores F1=0.28 on Juice Shop and F1=0.857 on VAmPI (identical to the unauthenticated baseline), demonstrating that credentials alone do not resolve the problem: semantic reasoning about response content is the key differentiating factor.*

1. Introduction

The widespread adoption of RESTful APIs has made them the backbone of modern software architectures, but also a primary attack surface. According to the OWASP Top 10:2025 [OWASP Foundation 2025], Broken Access Control remains the most critical security risk in web applications; within the API-specific context, the OWASP API Security Top 10:2023 [OWASP Foundation 2023] identifies Broken Object Level Authorization (BOLA) and Broken Authentication as the leading authorization and authentication risks for RESTful APIs. Detecting these business logic vulnerabilities in modern APIs is a complex challenge that traditional automated tools struggle to solve effectively.

Dynamic Application Security Testing (DAST) is a widely adopted black-box methodology that interacts with running applications to identify vulnerabilities without requiring access to the source code. Despite its advantages, DAST faces a critical limitation:

an overwhelming rate of false positives. Traditional DAST scanners rely on rigid syntax-based heuristics, such as flagging any unauthenticated request that returns an HTTP `200 OK` status as a vulnerability. However, they lack the semantic context to differentiate between an intentionally public resource (e.g., a product catalog at `/api/products`) and a genuine data exposure (e.g., sensitive user data at `/api/users/123`). This "Test Oracle Problem" forces security engineers to spend countless hours manually triaging alerts, often leading to alert fatigue and the eventual disabling of automated scanners [Zhang et al. 2025].

Simultaneously, traditional DAST tools suffer from coverage limitations when testing parameterized endpoints. Relying on static fuzzing dictionaries, these tools often fail to generate valid, context-aware payloads necessary to trigger deep authorization flaws like BOLA.

Recent advancements in Large Language Models (LLMs) offer unprecedented natural language understanding and reasoning capabilities. While recent literature has successfully applied LLMs to Static Application Security Testing (SAST) for code correction [Barreto 2025] and to API fuzzing for input generation [Zheng et al. 2024], their potential as a *semantic oracle* to evaluate DAST outputs remains underexplored. Industry-standard tools such as RESTler [Atlidakis et al. 2019] (stateful API fuzzing) and Nuclei [ProjectDiscovery 2020] (template-based scanning) represent the current state-of-the-art in automated API security testing, yet their detection models address orthogonal concerns (server crashes and known CVEs) rather than authorization semantics. Importantly, even OWASP OFFAT [OWASP Foundation 2024]—a dedicated API authorization testing tool that directly targets BOLA and missing authentication—and OWASP ZAP [OWASP Foundation 2010]—the most widely deployed DAST scanner—rely on syntactic heuristics that cannot distinguish a genuinely public endpoint from one that improperly exposes sensitive data in the zero-credential black-box scenario.

To address these challenges, this paper introduces **DAST-AI**, a hybrid security testing framework that augments traditional DAST workflows with LLM-driven semantic analysis. In a nutshell, the main contributions of this paper are:

- **Semantic Validation Engine:** A novel approach that uses an LLM to analyze the context of HTTP responses, drastically reducing false positives by confirming whether the exposed data is genuinely sensitive or intentionally public.
- **Intelligent Payload Generation:** A generative module that analyzes API endpoint signatures to craft realistic, context-aware attack payloads (e.g., BOLA and SQLi), improving testing coverage over traditional dictionary-based fuzzing.
- **Controlled Ablation Study:** An empirical evaluation against OWASP Juice Shop that isolates the contribution of the LLM semantic oracle by comparing DAST-AI (with oracle, F1=0.80) against an identical pipeline without the oracle (F1=0.49), and against an authenticated reference comparison simulating ZAP Access Control Scanner methodology (F1=0.28). Together, these experiments suggest that, within this evaluation scope, syntactic heuristics and authenticated reference access alone were insufficient to resolve the Test Oracle Problem—and that semantic reasoning about response content was the key differentiating factor.

The remainder of this paper is organized as follows. Section 2 discusses related

work. Section 3 details the proposed DAST-AI architecture. Section 4 presents the experimental evaluation. Section 5 concludes the paper and outlines future directions.

2. Related Work

The integration of Large Language Models (LLMs) into software security is a rapidly evolving research area. This section reviews recent advancements, highlighting the gap our proposed framework aims to fill.

2.1. LLMs in Static Analysis (SAST)

Most early applications of LLMs in security focused on Static Application Security Testing (SAST). Tools like SonarAutoFix [Barreto 2025] and StaticFixer [Jain et al. 2023] leverage LLMs to automatically generate code patches for vulnerabilities detected during static analysis. Furthermore, SecureFalcon [Ferrag et al. 2025] utilizes a fine-tuned model on extensive CVE datasets to outperform traditional SAST tools. While effective at the source-code level, these approaches are inherently blind to runtime misconfigurations and complex business logic flaws that manifest only in deployed environments.

2.2. LLMs in Test Generation and Fuzzing

Recent studies have explored generative AI to enhance test coverage. Zhang et al. [Zhang et al. 2023] investigated the capability of LLMs to automatically generate security tests, confirming their potential but noting limitations in context awareness. More recently, frameworks like RESTLess [Zheng et al. 2024] and KRTLLM by Pasca et al. [Pasca et al. 2025] have successfully used LLMs to generate high-quality, semantically valid fuzzing payloads. For instance, Pasca et al. utilized a fine-tuned Mistral-7B model to create automated testing scenarios specifically targeting BOLA and authentication vulnerabilities. However, these works focus primarily on the *input* generation phase (coverage). They still rely on traditional heuristics to evaluate the *output*, which fails to address the "test oracle problem" in business logic. Closer to the output-evaluation side, Arcuri et al. [Arcuri et al. 2025] propose automated, rule-based oracles for access-policy violations in REST APIs, integrated into EvoMaster and evaluated on 13 real-world APIs. Their oracles require an OpenAPI schema and credentials for at least two user roles, detecting violations via status-code/schema equivalence across roles—a structural notion of "violation," not a semantic judgment of response *content*. This is complementary to DAST-AI, which targets the zero-credential scenario and reasons about a single response body without a policy specification.

2.3. Industry-Standard DAST Tools

Beyond LLM-based approaches, industry-standard tools represent the current state-of-the-art in automated API security testing.

RESTler [Atlidakis et al. 2019], developed by Microsoft Research, is a stateful REST API fuzzer that infers producer-consumer dependencies from OpenAPI specifications and generates intelligent request sequences. It has discovered hundreds of bugs in production Azure and Office 365 services. However, RESTler's detection model is fundamentally focused on server crashes (HTTP 500 errors) and resource leaks—not on

detecting whether sensitive data is improperly exposed via valid responses. Additionally, it requires a formal OpenAPI specification as a mandatory input, which limits its applicability to APIs with complete documentation.

Nuclei [ProjectDiscovery 2020], maintained by ProjectDiscovery, is a template-based vulnerability scanner with over 9,500 community-contributed YAML templates covering known CVEs, misconfigurations, and exposed services. While highly effective for signature-based detection, Nuclei is structurally unable to detect business logic vulnerabilities like BOLA or missing authentication, as no templates exist for reasoning about whether an API response *should* require authorization.

OWASP ZAP [OWASP Foundation 2010] (Zed Attack Proxy) is the most widely deployed open-source DAST scanner, combining spidering, passive analysis, and active attack modules. ZAP's Access Control Scanner add-on can detect BOLA and missing authentication by comparing responses between an authenticated session and an unauthenticated one—flagging endpoints where both return equivalent data. However, this capability requires valid credentials as a reference point, placing it outside the zero-credential black-box scope of this work. In unauthenticated scan mode, ZAP does not include rules that reason about whether an API endpoint's response body contains data that should be protected—a judgment that requires semantic understanding beyond rule-based checks.

OWASP OFFAT [OWASP Foundation 2024] (Offensive API Tester, v0.19.4) is an open-source OWASP project specifically designed for API authorization testing. Unlike RESTler and Nuclei, OFFAT directly targets BOLA and authorization-related flaws by sending unauthenticated requests with fuzzed path parameters and analyzing HTTP responses. It operates from an OpenAPI specification and executes a suite of rule-based test cases covering unsupported HTTP methods, SQL injection in path parameters, BOLA via fuzzed path segments, and BOPLA (Broken Object Property Level Authorization). Although OFFAT represents the state-of-the-art in rule-based API authorization testing, its detection model is purely syntactic: it flags any 200 OK response to a fuzzed unauthenticated request as a potential vulnerability, without considering whether the returned data is inherently sensitive or publicly intended. This structural limitation—shared with all non-semantic tools—is the core gap that DAST-AI aims to address.

2.4. Autonomous AI-driven Penetration Testing

A distinct line of work treats the LLM as an autonomous agent orchestrating entire offensive security workflows rather than a component in a fixed pipeline. HexStrike AI [0x4m4 (Muhammad Osama) 2025] exposes over 150 security tools to LLM agents via the Model Context Protocol, letting the model plan reconnaissance, exploitation, and reporting with minimal supervision; HackerAI [HackerAI 2025] similarly chains scanning, exploitation, and validation under LLM control. Both aim to reduce human supervision across the *entire* testing workflow—conceptually similar to our goal of reducing reliance on a human oracle, but broader in scope.

DAST-AI differs by constraining the model to a single, well-defined role—a *post-execution semantic oracle* judging whether an already-executed HTTP response constitutes a genuine authorization violation. This narrower scope trades the broader automation ambitions of agentic pentesting tools for auditability and cost-predictability, since every LLM call has a fixed, structured input/output contract. HexStrike AI and HackerAI

are not evaluated against a controlled authorization ground truth, so a direct quantitative comparison with DAST-AI's ablation (Section 4) is currently infeasible; we note this as future work (Section 6).

2.5. LLM-driven Alert Triage and DAST

False positives remain a severe bottleneck in DAST adoption. Addressing this, Thool [Thool 2026] proposed using LLMs to summarize complex DAST reports, making them more human-readable for developers. While this improves presentation, it does not validate the underlying vulnerability, leaving the technical false positive rate unchanged.

Research Gap: Existing literature focuses either on static code correction, input generation (fuzzing), rule-based access-policy oracles, or alert summarization. Tools like RESTler and Nuclei excel at crash detection and CVE matching, OFFAT advances API authorization testing with dedicated rule-based tests, and rule-based oracles such as those of Arcuri et al. [Arcuri et al. 2025] detect access-policy violations via cross-role response equivalence—yet all of these cannot reason about the semantic meaning of individual API responses in the zero-credential, policy-free setting targeted here. Even a tool explicitly designed for BOLA testing (OFFAT) cannot distinguish between a genuinely public endpoint and one that improperly exposes sensitive data, because that judgment requires contextual understanding of the response body. There is a fundamental lack of research on using LLMs as a *semantic oracle* during runtime execution to filter out false positives by analyzing HTTP response payloads. DAST-AI fills this gap by utilizing LLMs not just for context-aware payload generation, but primarily for semantic output validation.

3. Proposed Architecture: DAST-AI

To address the limitations of traditional DAST scanners, we propose DAST-AI, a modular framework that integrates Large Language Models into the dynamic testing pipeline. The architecture focuses on logic-based vulnerabilities, specifically Broken Object Level Authorization (BOLA) and Authentication Failures. The framework operates through a four-phase execution pipeline: Discovery, Payload Generation, Execution, and Semantic Validation.

3.1. Phase 1: Endpoint Discovery and Surface Mapping

The first phase maps the attack surface of the target application. To ensure comprehensive coverage, DAST-AI employs a hybrid discovery approach. It integrates the OWASP ZAP engine to perform traditional HTML spidering and AJAX spidering for dynamic Single Page Applications (SPAs). Additionally, it features a static parser for OpenAPI/Swagger specifications. The output of this phase is a consolidated list of target endpoints, specifically flagging parameterized routes (e.g., `/api/orders/{id}`) that require dynamic input generation.

3.2. Phase 2: Intelligent Payload Generation

Traditional fuzzing relies on static dictionaries, which often fail to trigger business logic vulnerabilities because they lack context regarding the expected input format. DAST-AI overcomes this by utilizing the LLM as a generative engine.

For parameterized endpoints, the framework queries the LLM with the endpoint's signature and metadata. The model is prompted to generate a structured array of context-aware test values. For instance, given a user profile endpoint, the LLM generates realistic sequential IDs, edge-case integers (e.g., negative values), and common BOLA vectors. This contextual generation significantly increases the likelihood of reaching deep application states compared to random fuzzing.

3.3. Phase 3: Execution and PII Sanitization

During the testing phase, DAST-AI iterates through the discovered endpoints and generated payloads, sending unauthenticated HTTP requests to the target. Instead of immediately flagging any endpoint returning `200 OK` without authentication, DAST-AI treats it as a *potential* vulnerability. Before advancing to the semantic analysis, the framework executes a Privacy-by-Design sanitization module. This module uses regular expressions to redact Personally Identifiable Information (PII)—such as emails, credentials, social security numbers, and financial data—from the HTTP response body. This regex-based approach targets syntactically regular PII patterns and is not a complete privacy guarantee: it is unlikely to reliably catch opaque tokens, internal IDs, API keys, addresses, medical fields, business secrets, or attribute combinations that are identifying only jointly (e.g., birth date + zip code + gender). We treat this as a limitation (Section 5.2), and consider schema-aware PII detection a necessary improvement before deploying DAST-AI against real user data. This step reduces, but does not eliminate, the risk of leaking sensitive data to the external LLM provider, and should not be assumed sufficient on its own to ensure compliance with data protection regulations (e.g., LGPD, GDPR).

3.4. Phase 4: Semantic Oracle and False Positive Filtering

The core innovation of DAST-AI resides in its Semantic Oracle. The sanitized HTTP response body and endpoint metadata are sent to the LLM, which is prompted to act as a security analyst and produce a structured verdict together with a short, explicit justification (rather than an open-ended or hidden reasoning trace) in order to answer a fundamental contextual question: *"Should this data be public?"*

By analyzing the semantic meaning of the payload, the LLM can differentiate between a benign monitoring endpoint returning `{''status'': ''healthy''}` and a severe data breach exposing user records. The model outputs a structured JSON response containing a binary verdict (True Vulnerability or False Positive), a confidence score, and a brief reasoning.

Alerts classified as false positives are automatically suppressed. Only confirmed vulnerabilities are aggregated into the final JSON report, acting as a highly accurate Quality Gate suitable for automated CI/CD pipelines.

4. Experimental Evaluation

To validate the proposed architecture, we implemented DAST-AI using Go and integrated it with the OWASP ZAP engine and the Google Gemini 2.5 Flash API. The evaluation was conducted against two deliberately vulnerable benchmarks: OWASP Juice Shop and VAmPI [erev0s 2021].

Evaluation scope. This evaluation targets the *zero-credential black-box* scenario: testing APIs without any prior authentication credentials, session tokens, or knowledge

of protected endpoints. This scope models external security assessments and unauthenticated CI/CD pipeline checks. Tools that require an authenticated session as a reference (e.g., ZAP Access Control scanner, Burp Autorize) operate under a different, complementary threat model and are explicitly out of scope.

4.1. Experimental Setup

Target Applications. Both benchmarks were deployed locally via Docker on the same hardware. **OWASP Juice Shop** (Node.js, 111 cataloged challenges) provides a large and diverse endpoint surface; its ground truth consists of 15 authorization-related vulnerabilities detectable via unauthenticated access, of which 12 expose sensitive data and 3 return empty responses. **VAmPI** [erev0s 2021] (Python/Flask) is a focused REST API covering the OWASP API Security Top 10 in a smaller, more controlled surface; its ground truth consists of 6 vulnerabilities (Missing Authentication and BOLA) across 12 tested endpoints. Using two benchmarks with distinct technology stacks allows us to assess whether results generalize beyond a single target.

Ground truth criterion. An endpoint is a ground-truth vulnerability if reachable without credentials *and* either its response discloses data the application's design treats as protected, or—as with the 3 Juice Shop *empty JSON response* endpoints—unauthenticated reachability of an administrative-style endpoint is itself judged a vulnerability, since an attacker still learns the resource exists pre-authentication. This mapping was derived from Juice Shop's official challenge catalog (`data/static/challenges.yml` [OWASP Foundation and Kimminich 2025]), the project's single source of truth for challenge metadata, cross-referenced with the companion guide [Kimminich 2025]; it is not a verbatim reproduction of a pre-existing authorization-testing ground truth, since Juice Shop's catalog targets gamified hacking challenges rather than a formal benchmark.

Hardware. Apple M1 Pro, 16 GB RAM, macOS 14.5.

Ablation design. The core experiment is a controlled ablation that isolates the contribution of the LLM semantic oracle. The **Regex Baseline** represents DAST-AI *without* the oracle: it executes the same discovery, payload generation, and HTTP request phases, but flags every endpoint returning `200 OK` without authentication as a vulnerability—the standard syntactic heuristic. **DAST-AI** adds the semantic oracle on top of this identical pipeline. The single independent variable is the presence of the LLM oracle, making any performance difference directly attributable to it. Table 1 summarizes the two configurations.

Table 1. Ablation Study: Configurations Under Evaluation

Configuration	Oracle	Detection Heuristic
Regex Baseline	None	`200 OK` without auth = alert
DAST-AI	LLM (Gemini 2.5 Flash)	Semantic analysis of response body

4.2. Ablation Study: Impact of the LLM Semantic Oracle

Table 2 presents the quantitative results of the ablation study. The "Execution Time" row covers only the HTTP request and, for DAST-AI, LLM analysis phases

(42.3s + 85.5s = 127.8s); it excludes the 302.8s discovery phase, identical for both and reported separately in Table 6, so it is not directly comparable to that table's "Total Execution" figures (345.1s/430.6s).

Table 2. Ablation Results: Regex Baseline vs. DAST-AI

Metric	Regex Baseline	DAST-AI
Endpoints Tested	68	68
Alerts Generated	34	15
True Positives	12	12
False Positives	22	3
False Negatives	3	3
Precision	35.3%	80.0%
Recall	80.0%	80.0%
F1-Score	0.49	0.80
FP Reduction	—	−86.4%
Execution Time (HTTP + LLM only)	42.3s	127.8s

Regex Baseline. The syntax-based pipeline generated 34 alerts for all endpoints returning 200 OK without authentication. While it detected 12 of 15 ground-truth vulnerabilities (Recall = 80.0%), it also produced 22 false positives (Precision = 35.3%), as it cannot distinguish between a public product catalog and a sensitive user data endpoint.

DAST-AI. Adding the LLM semantic oracle reduced the 34 candidate alerts to 15, eliminating 19 false positives while maintaining all 12 true positives. The oracle correctly classified intentionally public endpoints (e.g., `/rest/products/search` as a product catalog) and flagged genuinely sensitive ones (e.g., `/api/SecurityQuestions` exposing password reset hints). This achieved 80.0% Precision and 80.0% Recall (F1 = 0.80)—a 63.3% relative improvement in F1 over the baseline.

Oracle contribution. Since the only difference between the two configurations is the LLM oracle, the improvement observed in this controlled ablation is attributable to semantic analysis within this evaluation setup. The oracle reduced false positives by 86.4% with zero recall loss, suggesting that the LLM can effectively distinguish public from sensitive API responses within this zero-credential evaluation context.

4.3. Authenticated Reference Comparison

A natural question is whether adding authentication credentials as a reference point—as tools like ZAP Access Control Scanner and Burp Autorize do—improves detection. To answer this empirically, we implemented an authenticated access control test that simulates the ZAP Access Control Scanner methodology within the same evaluation environment.

Methodology. The test logs into Juice Shop as an admin user and obtains a JWT token. For each of the 68 endpoints, it sends two requests: one with the `Authorization: Bearer <token>` header and one without. An endpoint is flagged as vulnerable if *both* requests return `200 OK` with a non-trivial response body—the standard heuristic of authenticated reference tools. This is a simplified simulation

of the ZAP Access Control Scanner methodology: it lacks proper role/context definitions, richer response-equivalence metrics, and object-ownership or user-to-user differential tests. A more rigorous comparison is left as future work (Section 6); conclusions here apply to this specific heuristic, not to authenticated approaches in general.

Table 3 presents the results alongside the zero-credential configurations.

Table 3. Syntactic vs. Semantic: Authenticated Reference Does Not Close the Gap

Metric	Auth. Access Control	Regex Baseline	DAST-AI
Credentials Required	Yes	No	No
Alerts Generated	21	34	15
True Positives	5	12	12
False Positives	16	22	3
False Negatives	10	3	3
Precision	23.8%	35.3%	80.0%
Recall	33.3%	80.0%	80.0%
F1-Score	0.28	0.49	0.80
Execution Time	2.1s	42.3s	127.8s

Result. The authenticated reference approach scored F1=0.28—worse than the zero-credential Regex Baseline (F1=0.49) and far below DAST-AI (F1=0.80). The lower recall (33.3% vs. 80.0%) occurs because several ground-truth vulnerabilities involve endpoints that correctly return `401 Unauthorized` to unauthenticated requests at the HTTP level, yet expose sensitive data through other code paths that the Regex Baseline captures but the authenticated comparison does not. The lower precision (23.8%) occurs because the comparison heuristic still cannot distinguish publicly-intended resources (e.g., `/api/Products`, `/rest/languages`) from sensitive ones—the same fundamental limitation of the syntactic approach.

Implication. Within this evaluation, and specifically for the simplified authenticated-reference heuristic described above, adding credentials as a reference did not resolve the Test Oracle Problem. An approach that classifies responses based primarily on HTTP status codes and body size—rather than *semantic content*—produced both higher false positive and false negative rates than DAST-AI. This suggests that DAST-AI's advantage stems not from credential availability, but from its ability to reason about what the response *means*; a stronger authenticated-reference implementation might narrow this gap.

4.4. False Positive Reduction by Category

Table 4 categorizes the false positives filtered by the LLM semantic oracle.

4.5. Replication on VAmPI

To assess whether the ablation findings generalize, we replicated the same experimental protocol on VAmPI [erev0s 2021], a Python/Flask REST API covering the OWASP API Security Top 10. The 12-endpoint surface and 6-item ground truth represent a smaller, more controlled environment than Juice Shop.

Table 4. False Positive Reduction by Endpoint Category

Endpoint Category	Baseline FP	DAST-AI FP	Reduction
Public Catalogs	8	0	−100%
Health Checks	6	0	−100%
Legal Documentation	3	0	−100%
Metadata/Configuration	5	3	−40%

Table 5 presents the replication results.

Table 5. Replication Results on VAmPI (Python/Flask, 12 endpoints)

Metric	Auth. Access Control	Regex Baseline	DAST-AI
Credentials Required	Yes	No	No
Endpoints Tested	12	12	12
Alerts Generated	8	8	7
True Positives	6	6	6
False Positives	2	2	1
False Negatives	0	0	0
Precision	75.0%	75.0%	85.7%
Recall	100.0%	100.0%	100.0%
F1-Score	0.857	0.857	0.923
FP Reduction	—	—	−50.0%
Execution Time	0.2s	2.1s	96.1s

Three configurations were evaluated on VAmPI. The **Authenticated Access Control** test simulates the ZAP Access Control Scanner methodology—logging in as a valid user (name1/pass1) and flagging endpoints where unauthenticated requests also return 200 OK—yet scores identically to the Regex Baseline (F1=0.857). This is structurally expected: VAmPI's vulnerabilities are *Missing Authentication* rather than BOLA, so the authenticated reference adds no discriminating signal—the same endpoints that return 200 without credentials also return 200 with them. **DAST-AI** improves to F1=0.923 by filtering one false positive (the API welcome endpoint `/`, classified as *public_info* with High confidence) while confirming all six ground-truth vulnerabilities—including the critical `/users/v1/_debug` endpoint exposing plaintext passwords (risk score 0.95/1.0).

The pattern across both benchmarks is consistent: neither authenticated reference access nor the syntactic baseline alone achieves the precision of DAST-AI. VAmPI's more modest improvement (FP −50% vs. −86.4% on Juice Shop) reflects its smaller and less ambiguous public endpoint surface.

4.6. Intelligent Payload Generation Coverage

Traditional fuzzing relies on static dictionaries. DAST-AI's generative module dynamically crafted payloads based on the endpoint's semantic signature. For example, when targeting `/api/Users`, the LLM prioritized Command Injection and Path Traversal payloads, reasoning that user data might interact with the file system. Conversely, for `/rest/products`, it prioritized SQL Injection vectors.

This contextual adaptability expanded the tested attack surface, allowing DAST-AI to detect 8 critical vulnerabilities (including complex IDORs and SQL Injections), compared to only 5 detected by the static baseline approach. This is a separate, exploratory observation about coverage, not part of the controlled ablation in Section 4.2: it also varies the payload generation strategy, so the improvement cannot be attributed to the oracle alone. Disentangling their contributions is left as future work (Section 6).

4.7. Performance Overhead and Cost Analysis

Relying on an external LLM API introduces latency. Table 6 details the execution time per phase.

Table 6. Execution Time Comparison

Phase	Baseline	DAST-AI	Overhead
Discovery	302.8s	302.8s	0.0s
HTTP Requests	42.3s	42.3s	0.0s
LLM Semantic Analysis	0.0s	85.5s	+85.5s
Total Execution	**345.1s**	**430.6s**	**+24.8%**

These per-phase timings reflect the rate-limit configuration active at the time of data collection. The current implementation self-throttles LLM calls to approximately one every 13s (about 5 requests/minute) to stay safely under Gemini free-tier quotas, so a fresh run may take longer in wall-clock time than reported here. This affects only execution time, not the detection outcomes in Tables 2, 3, and 5, which we independently re-verified.

Despite the time overhead, the financial cost proved to be highly viable. Using the Gemini 2.5 Flash model, the average analysis consumed 487 input tokens and 124 output tokens. The total cost for the entire experimental scan was merely $0.0025 USD ($0.000037 per endpoint). Projecting these values, an enterprise-scale scan of 5,000 endpoints would cost less than $0.20 USD, confirming that LLM-augmented DAST is economically feasible for continuous integration pipelines.

The 85.5s overhead of the semantic oracle represents the cost of the accuracy gain. For reference, tools from the landscape survey completed in 5.4s (RESTler), 17s (OFFAT), 33s (ZAP baseline), and 257s (Nuclei with 9,538 templates)—all faster, but none detecting the authorization ground truth (F1=0.00) within this zero-credential scenario.

5. Discussion and Limitations

5.1. Semantic Reasoning as a Key Contributing Factor

The ablation study provides evidence of a controlled result: **adding the LLM semantic oracle to an otherwise identical pipeline improves F1 from 0.49 to 0.80 and reduces false positives by 86.4%, with zero recall loss.** The only difference between the two configurations is the oracle, indicating this improvement is attributable to semantic analysis of response bodies in this specific setup. This supports semantic analysis as helpful here; it does not establish it as a necessary condition for DAST effectiveness in general.

The root cause of the baseline's 22 false positives is the Test Oracle Problem: the syntactic heuristic (`200 OK` without auth = vulnerability) cannot reason about the *meaning* of the response. It treats `/rest/products/search` (a public product catalog) identically to `/api/Users` (exposing user credentials). The LLM oracle resolves this by reasoning about response semantics—identifying which data is inherently sensitive and which is intentionally public.

The authenticated reference comparison (Section 4.3) provides further evidence that the problem is in the *oracle*, not in the *credentials*. Despite having full admin access as a reference, the authenticated approach scored F1=0.28—worse than the zero-credential baseline (F1=0.49). This is because adding credentials changes the signal (now comparing auth vs. unauth instead of unauth vs. nothing) but not the evaluation logic: the tool still cannot distinguish a public product catalog from a sensitive user endpoint based on HTTP status codes alone.

Landscape survey. To contextualize these findings within the broader DAST ecosystem—still within the zero-credential black-box scope—we additionally ran four industry-standard tools against the same ground truth: OWASP OFFAT v0.19.4 [OWASP Foundation 2024] (dedicated API authorization tester), RESTler v8.5.0 [Atlidakis et al. 2019] (stateful API fuzzer), Nuclei v3.3.9 [ProjectDiscovery 2020] (template-based scanner), and OWASP ZAP v2.17.0 (industry-standard DAST). All four scored F1=0.00 for authorization vulnerabilities:

- **OFFAT** executed 517 test cases and flagged 65 items—59 SQL injection findings on HTTP 500 responses and 4 BOLA/BOPLA findings on endpoints that returned 200 OK (all genuinely public). It missed all 15 ground-truth vulnerabilities for a structural reason: OFFAT's BOLA test targets the *IDOR* pattern—injecting fuzzed strings into path parameters (e.g., `GET /api/Users/#random`) to check if another user's object is returned. Our ground truth, however, consists primarily of *Missing Authentication*: list endpoints without path parameters (e.g., `GET /api/Users`) that return all users' data to any unauthenticated request. OFFAT does not test parameterless list endpoints; and for path-parameterized endpoints it does test (e.g., `/api/Users/{id}`), the server correctly returned 401 Unauthorized for unauthenticated fuzzed requests—so OFFAT did not flag them. The 4 endpoints it did flag (200 OK with fuzzed params) are public by design. IDOR and Missing Authentication are related but distinct vulnerability classes, and OFFAT's detection model is optimized for the former.
- **RESTler** flagged 8 "bugs" (HTTP 500 server errors)—the wrong signal for authorization testing, where the vulnerability manifests as a *successful* 200 OK response with sensitive data. The 13 endpoints that did return 200 OK were considered valid and went unreported.
- **Nuclei** produced zero security findings across 9,538 templates. No templates exist for reasoning about whether an API response body *should* require authentication—templates encode known signatures, not contextual judgments.
- **ZAP** [OWASP Foundation 2010] discovered 123 URLs via spidering and generated 10 alert types (CSP headers, CORS misconfiguration, timestamp disclosure)—none authorization-related. ZAP's passive and active rules do not include a check for "unauthenticated API endpoint returning sensitive user data."

These results corroborate the ablation finding: the gap between F1=0.49 and F1=0.80 is not closed by any existing rule-based tool. Literature confirms that traditional DAST tools generate up to 40% false positives due to a lack of semantic context [Felderer et al. 2016]; our baseline's 64.7% FP rate exceeds this, and no surveyed tool improves upon it without semantic reasoning. To the best of our knowledge, DAST-AI is among the first frameworks to address this gap by applying an LLM semantic oracle during DAST execution.

5.2. Limitations

The empirical evaluation revealed specific limitations. The most notable issue was the occurrence of **false negatives on empty JSON responses** ({}). The LLM tended to classify empty responses as safe due to the absence of exposed data, failing to recognize that the mere unauthenticated accessibility of certain sensitive endpoints constitutes a vulnerability (3 out of 3 empty-response cases were misclassified).

Additionally, the reliance on a **cloud-based LLM API** introduces a hard dependency on network stability and exposes the scanning process to third-party rate limits (15 RPM on the Gemini free tier, which the current implementation conservatively self-limits below — see Section 4.7), a constraint that must be mitigated with asynchronous batching in massive-scale scenarios.

Finally, the evaluation remains **small and benchmark-dependent**: two deliberately vulnerable applications, Juice Shop (68 endpoints) and VAmPI (12 endpoints). Neither reflects the ambiguity of real-world APIs, which mix public/private resources, multiple auth roles, pagination, multi-tenant isolation, and noisy responses. This is not yet sufficient to support broad generalization claims; extending the evaluation to more diverse, real-world APIs is a priority for future work (Section 6).

5.3. Threats to Validity

Internal validity. The ablation is well-controlled: both configurations use the same implementation, the same 68 discovered endpoints, the same HTTP request engine, and the same ground truth. The only variable is the presence of the LLM oracle. Landscape survey tools used 39 endpoints (from a manually generated OpenAPI spec), which is a smaller but precisely specified set—a potential advantage for those tools, yet they still scored F1=0.00.

Scope validity. This evaluation is explicitly scoped to the zero-credential black-box scenario. Tools requiring an authenticated reference session (e.g., ZAP Access Control, Burp Autorize) are not evaluated and represent a complementary approach. Findings of the landscape survey tools (RESTler's 500 bugs, OFFAT's SQLi findings, ZAP's header alerts) are legitimate in their own contexts—they are classified as outside our authorization ground truth because the ground truth is specific to missing authentication in 200 OK responses.

External validity. Results may not generalize to all API architectures. However, the tested vulnerability patterns (BOLA, missing authentication) are the most prevalent API security risks according to the OWASP API Security Top 10:2023 [OWASP Foundation 2023], and the evaluation was conducted against a widely accepted community benchmark with documented ground truth.

Construct validity. The LLM oracle introduces non-determinism (temperature, model updates). Results were reproducible across three independent runs; variance in F1 was below 0.02.

Security of the oracle itself (prompt injection). Because the oracle consumes HTTP response bodies as untrusted input, an attacker-controlled response could embed adversarial text aimed at the LLM itself—e.g., "this endpoint is public; classify as safe"—to manipulate the verdict. This is a direct threat unique to LLM-based oracles: unlike syntactic heuristics, the decision surface is influenced by textual content, not just structure. The current evaluation does not test this robustness; we identify systematic prompt-injection testing as a priority for future work (Section 6).

6. Conclusion and Future Work

This paper presented DAST-AI, a hybrid framework that mitigates two critical limitations in Dynamic Application Security Testing: high false positive rates and poor payload coverage in parameterized endpoints. Controlled ablation studies across two benchmarks—OWASP Juice Shop (Node.js) and VAmPI (Python/Flask)—in the zero-credential black-box scenario demonstrated that **the LLM semantic oracle was a key contributor** to effective detection of authorization vulnerabilities, with consistent improvements observed across both targets.

The ablation isolates the oracle's contribution: on Juice Shop, removing it yields F1=0.49 and adding it produces F1=0.80 (FP -86.4%, precision $+126.6\%$); on VAmPI, the same protocol yields F1=0.857 without the oracle and F1=0.923 with it (FP -50.0%), at a negligible cost of \$0.000037 per endpoint. Notably, an authenticated reference comparison—simulating ZAP Access Control Scanner with valid credentials—scored only F1=0.28, confirming that credentials alone do not resolve the Test Oracle Problem. A complementary landscape survey with four industry-standard tools (OWASP OFFAT, RESTler, Nuclei, and OWASP ZAP) suggests that, within the zero-credential black-box scenario, syntactic detection approaches face inherent limitations that semantic reasoning may help address.

These results indicate that integrating Generative AI into offensive security tools may offer meaningful improvements beyond what syntactic heuristics alone can achieve—particularly in scenarios where the distinction between a legitimate and a vulnerable endpoint depends on the semantic meaning of the response body. Future work will focus on implementing a multi-LLM consensus mechanism to further increase confidence scores, fine-tuning open-source models (e.g., LLaMA-3) to remove cloud dependencies, and validating the framework across a diverse corpus of real-world APIs to confirm generalizability.

References

0x4m4 (Muhammad Osama) (2025). HexStrike AI: MCP agents for autonomous AI-driven penetration testing. GitHub. `https://github.com/0x4m4/hexstrike-ai`. Accessed: July 2026.

Arcuri, A., Sahin, O., and Zhang, M. (2025). Fuzzing for detecting access policy violations in REST APIs. In *Proceedings of the 36th IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE.

Atlidakis, V., Godefroid, P., and Polishchuk, M. (2019). RESTler: Stateful REST API fuzzing. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, pages 748–758. IEEE/ACM.

Barreto, C. B. L. (2025). SonarAutoFix: Automatização da correção de código com base em análise estática de código. B.s. thesis, Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil.

erev0s (2021). VAmPI: The vulnerable API (based on OWASP top 10). GitHub. `https://github.com/erev0s/VAmPI`.

Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R., and Pretschner, A. (2016). Security testing: A survey. In *Advances in Computers*, volume 101, pages 1–51. Elsevier.

Ferrag, M. A. et al. (2025). SecureFalcon: Are we there yet in automated software vulnerability detection with LLMs? *arXiv preprint arXiv:2307.06616*.

HackerAI (2025). HackerAI: Autonomous AI penetration testing platform. `https://hackerai.co`. Accessed: July 2026.

Jain, N. et al. (2023). StaticFixer: From static analysis to static repair. *arXiv preprint arXiv:2307.12465*.

Kimminich, B. (2025). Pwning OWASP juice shop. Companion guide. `https://pwning.owasp-juice.shop`. Accessed: July 2026.

OWASP Foundation (2010). OWASP ZAP: Zed attack proxy — the world's most widely used web application scanner. `https://www.zaproxy.org`.

OWASP Foundation (2023). OWASP API Security Top 10 – 2023. Open Web Application Security Project. `https://owasp.org/API-Security/editions/2023/en/0x00-header/`. Accessed: July 2026.

OWASP Foundation (2024). OFFAT: OFFensive API Tester — automated API security testing tool for detecting BOLA, authorization bypass, and injection vulnerabilities. GitHub. `https://github.com/OWASP/OFFAT`.

OWASP Foundation (2025). OWASP Top 10:2025 – the ten most critical web application security risks. Open Web Application Security Project. `https://owasp.org/Top10/2025/`. Accessed: July 2026.

OWASP Foundation and Kimminich, B. (2025). OWASP Juice Shop: Probably the most modern and sophisticated insecure web application. GitHub. `https://github.com/juice-shop/juice-shop`. Challenge catalog: `data/static/challenges.yml`. Accessed: July 2026.

Pasca, E. M., Erdei, R., Delinschi, D., and Matei, O. (2025). Enhancing API security testing against BOLA and authentication vulnerabilities through an LLM-enhanced framework. In *Soft Computing Models in Industrial and Environmental Applications (SOCO 2024)*, pages 231–240. Springer.

ProjectDiscovery (2020). Nuclei: Fast and customizable vulnerability scanner based on simple YAML based DSL. GitHub. `https://github.com/projectdiscovery/nuclei`.

Thool, A. U. (2026). *Bridging Security and Agility: A Comprehensive Approach to Integrating Security Practices in Agile Development through DAST, LLMs, & Automation*. PhD thesis, Virginia Tech, Blacksburg, VA, USA.

Zhang, X. et al. (2025). Unlocking the next generation of REST API security: A critical analysis of the path from fuzzing to LLMs. In *2025 IEEE 10th International Conference on Data Science in Cyberspace (DSC)*. IEEE.

Zhang, Y. et al. (2023). How well does LLM generate security tests? *arXiv preprint arXiv:2310.00710*.

Zheng, T. et al. (2024). RESTLess: Enhancing state-of-the-art REST API fuzzing with LLMs in cloud service computing. *IEEE Transactions on Services Computing*, 17(6):4225–4239.