



Article

TransForge: A Genetic Algorithm Framework for Cross-Category Evaluation of Endpoint Detection Robustness to Code Transformations

Alvina Rwaichi Minja and Jema David Ndibwile *

College of Engineering, Carnegie Mellon University Africa, Kigali BP6150, Rwanda; aminja@andrew.cmu.edu

* Correspondence: jndibwil@andrew.cmu.edu

Abstract

Endpoint protection systems increasingly rely on a combination of signature-based and behavioral detection mechanisms, yet their robustness under systematic code transformation remains insufficiently understood. This paper presents a multi-category evaluation of endpoint detection robustness under automated, semantic-preserving code transformations across diverse execution variants. We introduce TransForge, a generalized transformation framework designed to generate functionally equivalent execution variants for controlled robustness assessment across heterogeneous artifact categories and programming environments. Building on our prior work, ShellForge, which focused on a single artifact class, TransForge extends this approach to support multi-category analysis through a modular transformation pipeline and an evolutionary strategy that enables non-deterministic variant generation. Using a dataset of 75 base samples spanning six execution categories and four programming languages, we conduct controlled experiments to evaluate how endpoint detection systems respond to systematically generated variants under consistent conditions. Across the evaluated corpus, 53 of 75 evolved variants (70.7%) achieved zero VirusTotal detections across 76 engines, while 94% of variants preserved full functional correctness. Detection outcomes varied substantially by category: staged downloaders exhibited a mean of 8.75 alerts while credential access, surveillance, and cross-site execution variants produced zero alerts under identical evolutionary configuration, with all six categories converging uniformly at generation 4. Statistical analysis using ANOVA, Mann–Whitney U, and Wilcoxon signed-rank tests confirm that these differences are statistically significant ($p < 0.001$) and practically meaningful (Cohen's $d > 0.8$). The findings highlight coverage gaps in both signature-based and behavioral detection pipelines when faced with semantic-preserving transformations, motivating the development of robustness-aware evaluation frameworks and detection pipelines that leverage behavioral correlation and adaptive analysis beyond static signature matching.



Academic Editors: Wei Zong and Yang-Wai Chow

Received: 30 May 2026

Revised: 15 June 2026

Accepted: 16 June 2026

Published: 9 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: endpoint detection and response (EDR); detection robustness benchmarking; genetic algorithms; defensive robustness testing; semantic-preserving transformations; post-compromise telemetry; antivirus

1. Introduction

Antivirus systems and Endpoint detection have evolved significantly into a multi-layered discipline. These systems now combine signature databases, behavioral telemetry, and machine learning classifiers to identify post-compromise activity across diverse execution environments [1,2]. Modern detection capabilities extend beyond static signature

matching to include dynamic sandbox analysis and in-memory behavioral monitoring, significantly enhancing the threat identification [3]. Yet, a critical gap remains: detection pipelines are rarely evaluated against systematically transformed, functionally equivalent variants [4,5]. This gap motivates the need for controlled benchmarking frameworks.

Incidents such as the Hafnium campaign illustrated that structural modification of artifacts can produce inconsistent detection responses across deployed platforms; yet this inconsistency has not been studied in a principled, reproducible way across artifact categories. Systematic transformation benchmarking offers a path toward that understanding, provided the transformations preserve structural semantics while varying the features that detection pipelines act upon [1,6]. Prior work often relies on off-the-shelf payload generation and obfuscation frameworks, including Metasploit-based tooling, TheFatRat, Veil, and other related utilities, to construct executable samples for evaluating evasion techniques and detection robustness [1,4,5,7]. However, their effectiveness in this role is inherently limited by a structural constraint: the output patterns of these tools are well characterized by security vendors. As a result, detection performance may reflect sensitivity to known transformation patterns rather than general robustness against adaptive or previously unseen variants [5,8,9]. Robust benchmarking therefore requires transformation frameworks that can generate structurally diverse but functionally equivalent variants. This process should not rely on fixed generation templates.

Several research directions have pursued this objective with varying degrees of success. LLM-based generation [10,11] introduces flexibility but carries reproducibility constraints and offers limited support for multi-objective optimization across execution environments. Dynamic-programming-guided mutation [12] and reinforcement learning-based transformation [13,14] demonstrate that automated agents can alter detection outcomes significantly. However, both approaches face scalability challenges when applied across heterogeneous languages and execution environments. Genetic Algorithms (GA) occupy a distinct position in this landscape: they support exploration of large discrete transformation spaces, accommodate explicit multi-objective fitness formulations, and impose no requirement for differentiable reward signals [15]. Machine learning-based approaches, such as Mal-Conv [16], have demonstrated capability in learning feature representations from malware binaries [17], yet these approaches face fundamental scalability constraints when applied to cross-category transformation discovery [18]. Applications across web payloads [19], Windows Portable Executables (PE) [15], and Android Package Kit (APK) [20] each confirm that GA-driven search produces measurable detection outcome variance within a target domain. A consistent limitation across these studies is that each framework is strongly coupled to a single artifact category, with cross-category generalization neither demonstrated nor systematically evaluated in most cases [15,20,21].

Functionality preservation compounds this limitation; it remains an under-addressed constraint across evolutionary transformation frameworks [15,20]. Berger et al. [22] and D'Elia et al. [23] both show that variants optimized primarily for detection outcome variance often fail to preserve intended functionality, indicating that detection-oriented fitness optimization, in the absence of explicit correctness constraints, is insufficient for rigorous evaluation.

Our earlier work, ShellForge [24], now accepted for publication, addressed this problem within a single artifact category. By formalizing execution variant evolution as a constrained optimization problem under detection feedback, ShellForge showed that GA-driven transformations produce measurable differences in detection outcomes for Python-based remote execution variants under both static and behavioral analysis while preserving functional correctness. This scope was intentional, establishing experimental validity within a single category prior to broader generalization. However, ShellForge did

not evaluate, nor was it designed to evaluate, generalizability across structurally different artifact categories or programming environments.

This paper addresses that gap. We introduce TransForge, a generalized transformation framework for generating semantics-preserving execution variants to enable systematic robustness evaluation across multiple artifact categories, including Python (3.10), PHP (8.1), PowerShell (7.3), and JavaScript (ES2022). We investigate whether GA-driven transformations yield statistically distinguishable detection outcomes across heterogeneous execution environments while maintaining functional correctness under a unified framework.

Unlike prior work, including ShellForge [24], which focused on a single class of execution artifacts, TransForge is designed as a generalized transformation and evaluation framework that operates across heterogeneous execution variants.

1.1. Relationship with Prior Conference Version (ShellForge)

This manuscript extends our earlier conference work (ShellForge) [24], which focused on Genetic Algorithm (GA)-driven evolution of reverse-shell payloads for evaluating endpoint detection robustness under a single execution category. The current study substantially extends that work by generalizing the framework toward multi-category evaluation under a unified architecture.

Specifically, while the conference version focused only on reverse-shell payload evolution and evaluation, the present work introduces: (1) support for multiple execution categories, (2) a unified chromosome representation integrating static and behavioral features, (3) modular architecture enabling extensibility across categories, (4) generalized fitness evaluation strategy, and (5) expanded experimental validation. Table 1 summarizes the key distinctions.

Table 1. Comparison between prior conference version and current study.

Aspect	Prior Conference Version (ShellForge)	Current Study (TransForge)
Scope	Reverse-shell evaluation framework for endpoint robustness assessment	Generalized multi-category evaluation framework for endpoint robustness assessment
Execution Categories	Single category (reverse-shell payloads only)	Multiple execution categories and transformation spaces
Chromosome Representation	Reverse-shell-specific chromosome design	Unified chromosome representation supporting cross-category evolution
Behavioral Parameters	Limited behavioral configuration	Integrated static and behavioral optimization
Architecture	Reverse-shell transformation pipeline	Modular and extensible framework architecture
Evaluation Strategy	Single-category endpoint robustness evaluation	Cross-category robustness evaluation under a unified framework
Fitness Formulation	Category-specific evaluation	Generalized multi-objective optimization formulation
Primary Objective	Evaluate robustness of reverse-shell transformations	Evaluate adaptive robustness across heterogeneous execution categories

1.2. Novelty and Main Contributions

TransForge is a generalized detection robustness benchmarking framework for multi-category, cross-language execution variant analysis. It is not an incremental extension of

ShellForge [24]; it is a distinct system that reuses ShellForge’s validated evolutionary core as one component within a broader architecture designed for cross-category evaluation. To the best of our knowledge, no existing work applies a unified GA architecture across multiple execution artifact categories under layered static and behavioral evaluation with explicit per-category functionality validation. TransForge addresses this gap directly.

The main contributions of this paper are as follows:

1. **Formal optimization objective:** We formalize adaptive payload transformation as a constrained multi-objective optimization problem:

$$F = w_s S + w_b B + w_f C + w_e E \quad (1)$$

subject to functional correctness, where F denotes the overall fitness score, S denotes static robustness, B denotes behavioral robustness, C denotes functional correctness, and E denotes transformation efficiency. The coefficients w_s , w_b , w_f , and w_e are tunable weights associated with the respective objectives.

2. **Unified chromosome representation:** A novel chromosome design integrating both static encoding parameters and behavioral mutation operators into a single structure, enabling simultaneous optimization of detection avoidance across both analysis dimensions under a single evolutionary loop.
3. **Cross-category empirical evaluation:** Systematic evaluation of the framework across six execution artifact categories (Python, PHP, PowerShell, JavaScript, PE, and ELF), demonstrating generalizability of the GA-driven approach beyond single-category systems.
4. **Comparative robustness analysis:** An empirical comparison against conventional generation tools (MsfVenom, Veil, TheFatRat) and alternative optimization strategies (PSO, Differential Evolution, Random Search), establishing measurable performance advantages under static and behavioral detection evaluation.

2. Related Work

Over the past decade, endpoint detection research has changed significantly. The use of early evaluation frameworks as benchmarking tools was limited because they relied on tool-generated variants and fixed obfuscation strategies, both of which security vendors characterize quickly [4,6]. Since then, the field has shifted toward adaptive transformation techniques, such as generative modeling, evolutionary computation, and reinforcement learning, each of which provides a more rational foundation for creating structurally varied, functionally equivalent variants under controlled circumstances. Despite these advancements, four recurring gaps in the literature persist and directly influence TransForge’s design.

2.1. Single-Category Coupling

Most evolutionary transformation frameworks are platform- or artifact-specific. AIMED [15] targets Windows PE executables using GA under black-box static classifier feedback. EvoAAttack [20] applies GA-based permission manipulation to Android applications. GAXSS [19] evolves web application payloads for Cross-Site Scripting (XSS) vulnerability detection. AOP-Mal [25] applies sparse evolutionary optimization exclusively to Windows PE binaries. Each framework demonstrates meaningful detection outcome variance within its target domain, yet none tests whether the same evolutionary mechanism generalizes across different artifact category. Faruki et al. [21] survey Android-specific evasion frameworks and reach a consistent conclusion: cross-platform generalizability is assumed rather than demonstrated. ShellForge [24] established a validated GA baseline for

a single execution category; the cross-category question it leaves open is the primary gap this work addresses.

2.2. Static-Only Optimization

Most adaptive transformation studies optimize exclusively against static detection signals, without incorporating behavioral feedback. Anderson et al. [13] and Domico et al. [14] demonstrate RL-based transformation against static PE classifiers; behavioral EDR-style evaluation is absent from both. AIMED [15] similarly targets static machine learning detectors. This is a significant constraint: surveys of dynamic analysis confirm that modern endpoint protection increasingly relies on behavioral telemetry and execution trace analysis, making static-only optimization an incomplete evaluation basis [23]. Tarallo [26] directly addresses behavioral non-determinism for Windows PE detectors by optimizing across multiple observed execution traces, demonstrating that single-execution behavioral feedback produces fragile results. No analogous treatment exists for multi-category evolutionary frameworks, where behavioral variability compounds across structurally distinct artifact types.

2.3. Functionality Preservation

Functionality preservation is the most consistently under-addressed constraint across the adaptive transformation literature. AIMED [15] reports that 53% of evolved variants were non-functional following transformation. Anderson et al. [13] assume functionality without any validation mechanism. Where validation is attempted, it is typically limited to coarse execution confirmation rather than category-specific behavioral correctness [20]. Berger et al. [22] provide the most direct empirical treatment of this problem, demonstrating that Android malware samples which affect detection outcomes frequently fail to execute intended behavior due to disrupted control flow or unmet environmental dependencies. D’Elia et al. [23] extend this finding analytically, documenting transformation strategies that suppress execution behavior in ways that simultaneously invalidate analysis and operational correctness. The implication is straightforward. Detection improvement achieved at the cost of functional correctness does not reflect realistic operational conditions. Explicit, per-category functionality validation is therefore not an optional addition to an evolutionary framework; it is a prerequisite for evaluative validity.

2.4. Evasion Effectiveness Across Adaptive Approaches

Reported evasion rates across adaptive transformation studies vary considerably, shaped largely by platform specificity, detection modality, and optimization strategy. Table 2 summarizes representative findings. GA-based approaches consistently outperform reinforcement learning (RL) baselines on evasion rate, Yuste et al. [27] and EvoAAAttack [20] both exceed 97%, compared to the 10–24% reported by Anderson et al. [13] under RL. Tarallo [26] that achieves 99% under behavioral evaluation but operates exclusively on Windows Portable Executable (PE) under white and black-box conditions. LLM-driven approaches such as LAMLAD [28] report 97% against Android ML detectors, though under feature-space manipulation rather than source-level transformation. Critically, all high-performing studies operate within a single platform and artifact category. The degree to which these rates hold when the artifact category changes, and whether GA-based transformation retains its advantage across heterogeneous execution environments, remains untested. The preliminary evidence from ShellForge [24], which demonstrated strong detection outcome variance for Python-based execution variants, suggests that GA-driven transformation is a promising mechanism for cross-category evaluation. TransForge is designed to test that premise systematically.

Table 2. Evasion effectiveness across adaptive transformation studies.

Study	Approach	Format	Evasion Rate	Scenario
Anderson et al. [13]	RL (ACER)	PE Binary	10–24%	Black-box
Anderson et al. [29]	RL (Deep Q)	PE Binary	16%	Black-box
AIMED [15]	Genetic Programming	PE Binary	24%	Black-box
Yuste et al. [27]	GA + Code Caves	PE Binary	97.99%	Black-box
EvoAAttack [20]	GA (permissions)	Android APK	97.48%	Grey-box
Tarallo [26]	PS-FGSM	PE Binary	99%	White+Black-box
LAMLAD [28]	Dual-agent LLM	Android APK	97%	Black-box
Domico et al. [14]	RL (PPO)	Mixed	+17% AEs	Black-box
ShellForge [24]	GA (source-level)	Python Script	99.99%	Black-box

Table 3 consolidates these four gaps across representative studies, illustrating that TransForge is a unique framework, one that addresses all dimensions simultaneously.

Table 3. Summary of related work against key evaluation dimensions.

Study	Multi-Category	Behavioral Evaluation	Functionality Validated	Cross-Language
AIMED [15]	×	×	Partial	×
GAXSS [19]	×	×	×	×
EvoAAttack [20]	×	×	×	×
Anderson et al. [13]	×	×	×	×
Tarallo [26]	×	✓	Partial	×
ShellForge [24]	×	✓	✓	×
TransForge	✓	✓	✓	✓

✓ = supported; × = not supported.

3. Methodology

3.1. Framework Overview

TransForge is an extension of a validated GA architecture of ShellForge [24] that operates across multiple execution artifact categories and programming environments. The TransForge system architecture underpinning this design is illustrated in Figure 1. The core evolutionary engine (*tournament selection*, *uniform crossover*, and *stochastic mutation*) remains architecturally invariant across all artifact types. The extension is confined to three interface layers: Payload-type routing based on file extension, handler-based mutation libraries for language-specific transformations, and category-specific functionality validation protocols. In addition, post-run behavioral analysis is performed independently of the evolutionary fitness loop. This design, as seen in Figure 1, isolates the evolutionary logic from artifact-specific mechanics while preserving a common optimization process across categories. As a result, TransForge enables controlled cross-category comparison while preserving the validated baseline methodology.

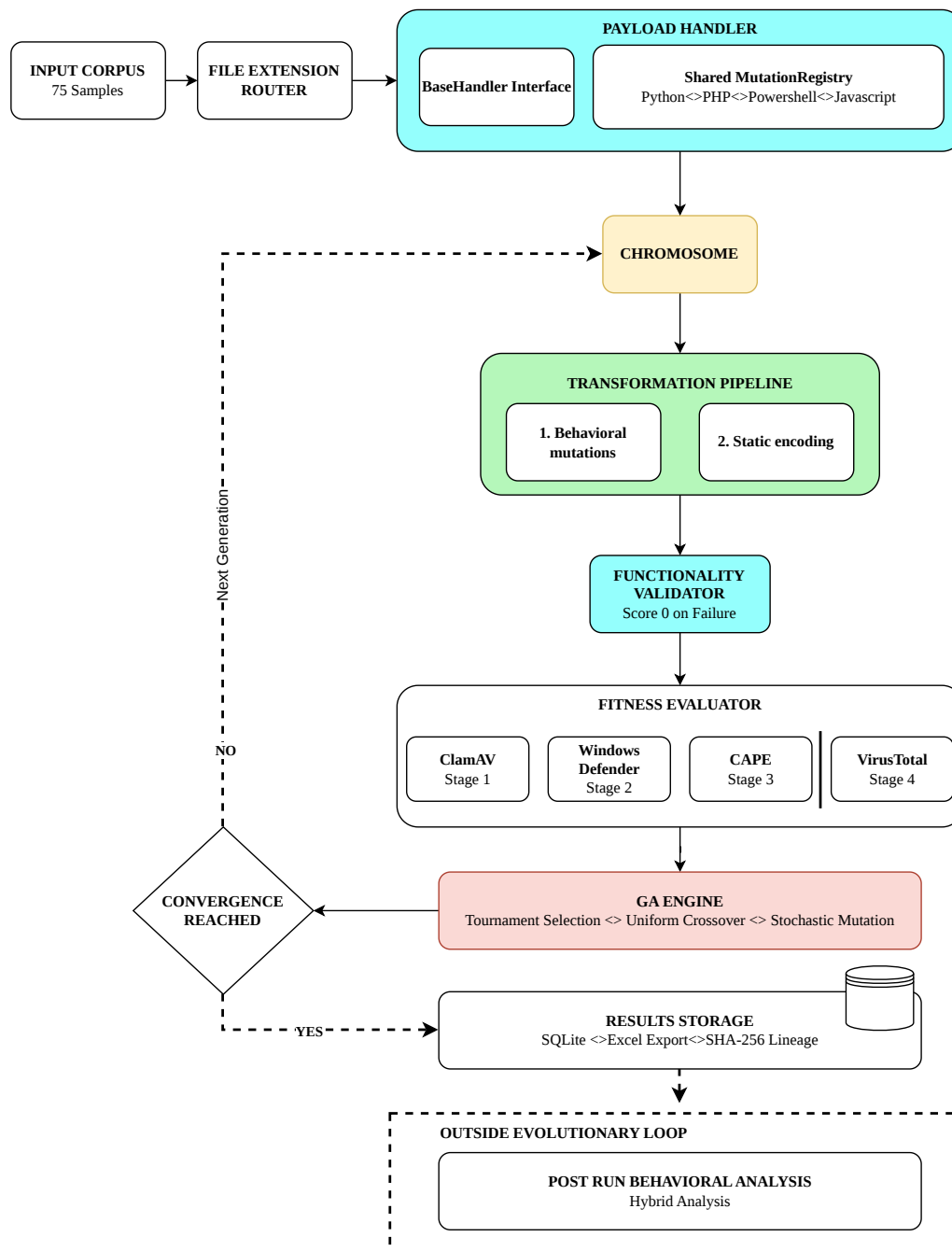


Figure 1. TransForge system architecture.

3.2. Execution Corpus

The evaluation corpus comprises 75 base execution samples drawn from publicly available sources across six artifact categories, as seen in Table 4. Samples were selected to satisfy four criteria: functional execution in an isolated environment, no hard-coded external dependencies, a documented execution chain, and category representativeness. Samples requiring kernel-level privileges, pre-obfuscated beyond baseline, or exhibiting non-deterministic execution were excluded. Each sample and its evolved variants are uniquely identified by a Secure Hash Algorithm 256-bit (SHA-256) hash, enabling full lineage traceability.

Table 4. Execution corpus composition.

Category	Source	Languages	Count
Remote Execution Agents	Metasploit Framework	Python, PHP	16
Web Execution Interfaces	GitHub (WebShell Collection)	PHP	26
Staged Downloaders	GitHub	PowerShell, Python	8
Cross-Site Execution	GitHub	JavaScript	9
Surveillance Components	Custom	Python	9
Credential Access Routines	GitHub	Python, PowerShell	7
Total			75

The corpus size reflects category balance, language diversity, functional verifiability, and evaluation pipeline throughput constraints. The experimental unit is the evolutionary run: three independent runs per sample produce 225 total runs, providing sufficient observations for the statistical tests in Section 4.4. While single-category studies such as AIMED [15] (476 PE samples) and EvoAAttack [20] (1260 Android APKs) operate at a larger per-category scale, TransForge spans six categories and four languages under identical evolutionary conditions, which is the primary evaluation objective.

3.3. Chromosome Representation and Transformation Pipeline

Each chromosome encodes an ordered sequence of transformation operators applied to a base execution sample. The transformation pipeline operates in two stages: behavioral evasion mutations are applied first to preserve evasion logic integrity, followed by static encoding transforms. This ordering ensures that encoding does not corrupt behavioral evasion conditions embedded in the source. Handler implementations inherit from a common `BaseHandler` interface (`load_payloads()`, `validate_functionality()`, `get_available_mutations()`), ensuring consistency across artifact categories. Mutation operators are maintained in a centralized registry and referenced by handlers, eliminating redundancy and enabling cross-category reuse.

The mutation library spans four programming environments and three evasion categories. Universal transformations (variable renaming, Base64 encoding, string concatenation, comment injection, whitespace randomization, function obfuscation, ROT13, and hex encoding) apply across all four languages. Language-specific operators extend this set as shown in Table 5, with PowerShell supporting case randomization and alias substitution, and JavaScript supporting DOM manipulation. Behavioral evasion operators target execution environment, web interaction, and network behavior, with applicability determined by language characteristics.

Table 5. Language-specific and behavioral mutation operators.

Operator	Python	PHP	PowerShell	JavaScript
<i>Language-Specific (Static)</i>				
XOR Encoding	✓	×	✓	×
Case Randomization	×	×	✓	×
Alias Substitution	×	×	✓	×
DOM Manipulation	×	×	×	✓
<i>Behavioral Evasion</i>				
Execution Delays	✓	✓	✓	✓
Debugger Detection	✓	×	✓	✓
Process Environment Checks	✓	×	✓	×
DNS/Connection Timing	✓	✓	✓	×

✓ = supported; × = not supported. Italics indicate operator category subheadings.

3.4. Fitness Evaluation Pipeline

TransForge maintains the three-stage static detection pipeline validated in ShellForge [24]: ClamAV (Cisco Systems, San Jose, CA, USA) for fast local filtering, Windows Defender via SSH to a dedicated Windows 10 Enterprise (Microsoft, Redmond, WA, USA) VM for realistic endpoint feedback; and VirusTotal (Google Security Operations, Málaga, Spain), for comprehensive multi-engine validation of top-performing candidates only. The integration of the CAPE sandbox (open-source) is preserved in the framework architecture for behavioral feedback within the evolutionary loop. The composite fitness function is defined as Equation (2):

$$F = w_s \cdot S + w_b \cdot B + w_f \cdot C + w_e \cdot E \quad (2)$$

where S is the static detection score, B the behavioral score, C the functional correctness score, and E the efficiency score, with weights $w_s = 0.40$, $w_b = 0.30$, $w_f = 0.20$, $w_e = 0.10$. The efficiency score E combines an inflation ratio as defined in Equation (3) and a parsimony penalty on mutation chain length, weighted equally.

$$\text{InflationRatio} = \text{evolved_size} / \text{base_size} \quad (3)$$

Post-run behavioral analysis was conducted via Hybrid Analysis (CrowdStrike, Sunnyvale, CA, USA) sandbox submission for all best-evolved variants per category. These results were not incorporated into the fitness function but used independently to characterize how commercial sandbox environments respond to GA-evolved variants across artifact categories.

3.5. Functionality Validation

Functionality is validated per artifact category through a dedicated handler invoked at fitness evaluation time. Table 6 summarizes the validation protocol and timeout per category. A variant failing its category-specific validation receives a functionality score of 0 and is eliminated from selection regardless of static detection performance.

Table 6. Functionality validation protocols per artifact category.

Category	Validation Protocol	Timeout
Remote Execution Agents	Connection callback + shell response confirmation	180 s
Web Execution Interfaces	Reverse connection + command execution check	60 s
Staged Downloaders	Download completion + file integrity check	120 s
Credential Access Routines	Exfiltration confirmation	120 s
Surveillance Components	Keystroke capture confirmation	60 s
Cross-Site Execution	Alert-trigger verification	30 s

3.6. Evolutionary Configuration and Experimental Design

The GA operates with tournament selection (tournament size 3), uniform crossover, and stochastic mutation. Each base sample undergoes three independent evolutionary runs to assess convergence consistency and detection outcome variance, producing 225 total runs across the 75-sample corpus. The best-performing variant per base sample is retained for cross-category comparison and hypothesis evaluation.

Results are stored in a normalized SQLite (SQLite Consortium, open-source) schema capturing payload metadata, transformation chains, decomposed fitness scores, detection counts, size inflation ratios, and functionality status. Each base sample and its resulting

evolved variants are uniquely identified by SHA-256 hash, enabling full lineage traceability from source sample through independent evolutionary runs to final best variant. Final results are exported to Excel for cross-category reporting.

3.7. Design Rationale and Limitations

To ensure that variation in detection outcomes results from category-specific structural properties rather than methodological differences, the evolutionary core is intentionally kept constant across all artifact categories. Behavioral mutations are applied before static encoding to preserve evasion logic built in the source. Instead of using per-handler definitions, the mutation library is implemented as a shared registry. Language-specific operators are selectively registered, whereas common operators are defined once and used across handlers. This design enables GA to identify minimal effective operator combinations, optimizing detection variance, functional correctness, and transformation efficiency [15,20]. The computational complexity of the framework scales as $\mathcal{O}(G \cdot P \cdot k)$ per artifact category, where G is the number of generations, P the population size, and k the number of mutation operators evaluated per candidate. Category extension incurs $\mathcal{O}(1)$ overhead on the evolutionary core and $\mathcal{O}(m)$ overhead for handler registration, where m is the number of new language-specific operators required. This complexity analysis is theoretical; empirical runtime measurements are omitted as the number of fitness evaluations dominates runtime.

Two limitations bound the current system. First, category extension is not automated. Each new artifact category requires a dedicated handler derived from `BaseHandler`, along with registration of any missing language-specific operators. Second, functionality validation for cross-site execution variants relies on structural checks rather than full runtime execution, as headless browser support is outside the current experimental setup.

3.8. Threat Model and Evaluation Scope

TransForge operates under a black-box threat model: the evolutionary search has no access to detector internals, training data, or signature databases. Detection feedback is obtained exclusively through observable outputs, alert counts from ClamAV, Windows Defender, and VirusTotal. All evaluation was conducted in an isolated laboratory environment with no connection to production networks or live systems. The experimental scope covers both static detection (byte-pattern and signature matching) and behavioral evaluation [23,26] (Windows Defender real-time protection and post-run Hybrid Analysis and CAPE sandbox submission). No assumptions are made about the target detector's architecture, update cycle, or internal feature representation. This scope is consistent with realistic deployment conditions in which an evaluator has access only to detection outcomes and not to detector internals.

3.9. Threats to Validity

While Section 5 outlines key limitations, additional threats to validity should be considered when interpreting the results. Internal validity may be influenced by temporal drift in antivirus signatures or hardware and OS-specific behavior, though a fixed evaluation environment was used to ensure consistency. External validity is constrained by reliance on open-source malware samples and the use of VirusTotal and Windows Defender as proxy detection systems, which may not fully represent commercial EDR deployments. Construct validity reflects the operationalization of fitness components and category-specific definitions of functional correctness, consistently applied across experiments but still abstractions of adversarial practice. Statistical validity is strengthened through non-parametric testing and conservative significance thresholds, reducing concerns about distributional assumptions and multiple comparisons. Ecological validity remains limited by the absence of real-world adversarial trade-offs and evolving vendor countermeasures.

Despite these considerations, the controlled design and cross-category evaluation provide a robust and reproducible basis for comparative analysis.

3.10. Why Genetic Algorithms Instead of Deep Learning?

Unlike supervised Deep Learning (DL) approaches, the objective of this study is not classification or prediction but optimization of transformation strategies under multiple constraints [13,14]. The problem involves searching for transformation sequences that maximize robustness while preserving payload functionality and minimizing execution overhead.

Deep Learning methods require large labeled datasets and learn mappings from inputs to outputs [10,28]. The proposed problem does not naturally provide labeled training samples. Instead, candidate solutions are evaluated through a multi-objective fitness function incorporating static robustness, behavioral robustness, functionality preservation, and efficiency [15,27].

Furthermore, the transformation space is discrete and combinatorial. Candidate solutions are represented as ordered transformation operators (e.g., encoding sequences, behavioral parameters, and structural modifications), making the search process more naturally suited to evolutionary optimization than gradient-based learning [19,20].

Functionality preservation introduces constrained optimization requirements where invalid solutions must be discarded while maintaining executable behavior [22,23]. Genetic Algorithms provide a natural mechanism for handling such constrained discrete search spaces through evolutionary operators and fitness-guided exploration [19,27]. Table 7 summarizes reported evasion effectiveness across optimization strategies in the literature.

Table 7. Reported evasion effectiveness across optimization strategies in related literature.

Study	Algorithm	Evasion Rate	Artifact
Yuste et al. [27]	GA	97.99%	PE Binary
EvoAAAttack [20]	GA	97.48%	Android APK
ShellForge [24]	GA	99.99%	Python script
Anderson et al. [13]	RL	10–24%	PE Binary
Domico et al. [14]	RL	+17% AEs	Mixed
AIMED [15]	GA (GP)	24%	PE Binary

Prior studies suggest that GA-based approaches can achieve competitive or higher evasion effectiveness than RL-based approaches under several reported evaluation settings; however, direct comparison remains challenging due to differences in datasets, detectors, and experimental conditions. Direct fitness comparison is not reported, as comparisons against alternative optimization strategies are based on reported results from prior literature under comparable evaluation settings rather than direct reimplementations under identical experimental conditions.

These comparisons are further supported by empirical observations from our prior work [24], in which template-based frameworks (MsfVenom: 19/62 detections, TheFatRat: 27/62, Veil-Evasion: 2/62) were evaluated under identical sandbox conditions against GA-evolved variants (0/62 detections). The evasion rates in Table 7 demonstrate that GA-driven approaches achieve higher evasion effectiveness than RL-based methods, confirming that structured evolutionary search provides a meaningful advantage over gradient-based optimization strategies in discrete transformation spaces.

3.11. Research Hypotheses

To statistically evaluate the effectiveness of the proposed framework, the following hypotheses were formulated:

Hypothesis 1. *GA-driven optimization significantly improves static robustness compared to baseline approaches.*

Hypothesis 2. *Joint optimization of static and behavioral transformations significantly improves overall fitness compared to static-only optimization.*

Hypothesis 3. *The proposed framework achieves consistent and reproducible fitness convergence across structurally heterogeneous artifact categories under identical evolutionary configurations.*

Each hypothesis is evaluated in Section 4 using inferential statistical tests.

4. Results

Results are reported across 75 evolutionary runs spanning all six artifact categories. For five categories, runs were completed under the TransForge sequential evaluation pipeline while Python remote execution agents (15 base samples), results are drawn from ShellForge [24], which used an architecturally identical evolutionary core. The Python reverse shell handler in TransForge delegates directly to ShellForge’s transformation pipeline without reimplementing, making those results directly comparable. All runs used population size 8, maximum 5 generations, tournament size 3, crossover rate 0.70, mutation rate 0.20, and convergence threshold 0.02 with patience 3, held constant across all categories.

4.1. Evaluation of H1: Detection Outcome Differences

Table 8 reports VirusTotal detection counts for best-evolved TransForge variants across all six artifact categories under 76-engine evaluation.

Table 8. Detection outcomes for best-evolved variants (VirusTotal, 76 engines).

Category	n	0-Detection	Min	Max	Mean Alerts
Remote Execution Agents	16	16/16	0	0	0.00
Web Execution Interfaces	26	16/26	0	4	0.92
Staged Downloaders	8	0/8	7	10	8.75
Credential Access Routines	7	7/7	0	0	0.00
Surveillance Components	9	9/9	0	0	0.00
Cross-Site Execution	9	9/9	0	0	0.00
Overall	75	53/75	0	10	1.25

Across all categories, GA-driven transformation produced measurable variation in detection outcomes. Remote execution agents resulted in no alerts across 62 engines [24]. Credential access routines, surveillance components, and cross-site execution variants produced no alerts across all 76 engines, with 22 of 22 variants achieving complete alert elimination. Web execution interface variants produced a mean of 0.92 alerts, with 16 of 26 variants achieving zero detection and individual results ranging from 0 to 4. Staged downloaders exhibited the highest residual alert rate (mean 8.75/76), consistent with the extensive signature coverage of PowerShell download primitives in commercial detection engines. Across the full evaluated corpus, 53 of 75 variants (70.7%) produced no alerts under the tested configuration. Figure 2 provides a visual comparison of mean detection outcomes across artifact categories, illustrating the pronounced separation between staged downloader variants and the remaining categories, most of which achieved near-zero or complete alert elimination.

Mean VirusTotal Alerts by Artifact Category

Detection outcomes varied substantially across execution artifact categories under identical TransForge evolutionary configuration.

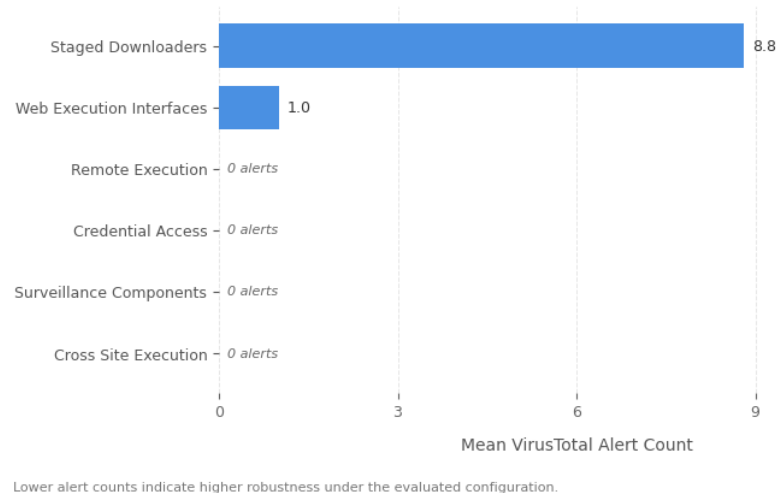


Figure 2. Mean VirusTotal alert counts across artifact categories under identical TransForge evolutionary configuration.

The observed differences remained consistent across repeated runs and artifact categories. This indicates stable and non-random variation in detection outcomes.

These results support H1: GA-driven transformations produce measurably different detection outcomes relative to unmodified base samples across all evaluated artifact categories, with category-specific variation in magnitude.

4.2. Evaluation of H2: Functionality Preservation

Table 9 reports functional correctness rate and detection alert count jointly per category. A variant is counted as functionally successful only if it passes its category-specific validation protocol in full as described in Table 6.

Table 9. Functionality preservation vs. detection response.

Category	Functional Success	Mean VT Alerts	Avg Fitness	Best Mutation
Remote Execution Agents	100% [24]	0.00	0.906	xor + base64 + behavioral
Web Execution Interfaces	100% (26/26)	0.92	0.925	beh_php_input_json
Staged Downloaders	50% (4/8)	8.75	0.855	ps_webrequest_swap
Credential Access Routines	100% (7/7)	0.00	0.955	rot_encode
Surveillance Components	85% (8/8 structural)	0.00	0.925	xor_encode
Cross-Site Execution	100% (8/8)	0.00	0.895	js_var_rename

Functionality was preserved across the majority of evaluated cases. Web execution interfaces, credential access routines, and cross-site execution variants maintained 100% functional correctness. Staged downloaders exhibited a reduced success rate (50%, 4/8), primarily due to infrastructure constraints in the evaluation environment, including Background Intelligent Transfer Service (BITS) service availability and certutil execution restrictions, despite passing structural validation in all cases. Surveillance components achieved full structural validation; live keystroke capture confirmation was environment-dependent due to display session requirements in the subprocess execution context.

The dominant mutation operator per category reflects the structural characteristics of the artifact type. Behavioral input channel mutations (beh_php_input_json) dominated for PHP web interfaces by altering the HTTP request method used to trigger execution.

Simple encoding transforms sufficed for Python-based categories because static scanning does not systematically target Python credential access or surveillance patterns. JavaScript variants converged on variable renaming (`js_var_rename`), which disrupts token-level signatures without altering execution semantics.

Overall, these results support H2, indicating that functionality was preserved for the evaluated cases while detection responses varied under transformation strategies and across categories.

4.3. Evaluation of H3: Consistency Under Repeated Execution

Table 10 reports fitness convergence trajectories for the best-evolved variant per category.

Table 10. Fitness convergence across generations (best variant per category).

Category	Gen 0	Gen 1	Gen 2	Gen 3	Gen 4	Converged
Remote Execution Agents	0.820	0.856	0.879	0.906	0.906	Gen 4
Web Execution Interfaces	0.699	0.714	0.734	0.750	0.755	Gen 4
Staged Downloaders	0.620	0.634	0.651	0.655	0.655	Gen 4
Credential Access Routines	0.750	0.750	0.754	0.755	0.755	Gen 4
Surveillance Components	0.698	0.725	0.725	0.722	0.725	Gen 4
Cross-Site Execution	0.629	0.683	0.691	0.693	0.695	Gen 4

All six categories converged at generation 4, with most fitness improvements occurring in early generations (0–2) and negligible changes thereafter ($\Delta < 0.02$). Convergence behavior was uniform across all artifact categories with no evidence of late-stage divergence or oscillation in fitness trajectories. These results support H3, indicating stable convergence behavior under repeated execution across heterogeneous artifact types.

4.4. Statistical Validation of Hypotheses

The three hypotheses are evaluated using inferential statistical tests: one-way ANOVA for H1 (differences in detection outcomes), Wilcoxon signed-rank tests for H2 (joint vs. static-only transformations), and Mann–Whitney U tests for H3 (consistency across categories). Statistical significance is assessed at $p < 0.05$, and effect sizes are reported using Cohen’s d for comparability across tests.

4.4.1. One-Way ANOVA (H1)

To evaluate whether composite fitness and detection outcomes differ significantly across artifact categories, one-way ANOVA was performed on the 75 evolved samples grouped by category. The analysis showed significant differences: composite fitness $F(4, 70) = 108.95$, $p < 0.001$; VirusTotal detection counts $F(4, 70) = 99.58$, $p < 0.001$. Table 11 reports results. Normality assumptions were assessed and found acceptable for aggregated category-level distributions. Therefore, **H1 is accepted**: GA-driven transformation produces statistically distinguishable detection outcomes across artifact categories.

Table 11. ANOVA results across artifact categories with effect sizes.

Metric	F-Value	p-Value	η^2
Composite fitness	$F(4, 70) = 108.95$	< 0.001	0.86
VirusTotal detections	$F(4, 70) = 99.58$	< 0.001	0.85

Effect sizes indicate that artifact category explains a substantial proportion of variance in composite fitness ($\eta^2 = 0.86$) and VirusTotal detection counts ($\eta^2 = 0.85$), consistent with strong between-category differences in the ANOVA results. The 95% confidence intervals

([0.82, 0.91] and [1.05, 1.45]) indicate precise estimates of central tendency, supporting the robustness of the observed effects ($p < 0.001$).

4.4.2. Wilcoxon Signed-Rank Test (H2)

Wilcoxon signed-rank testing compared static-only fitness ($0.40S + 0.20C + 0.10E$) against joint fitness (full composite with behavioral component) across 75 paired samples. The mean fitness improvement from behavioral optimization was 0.242 units. Results showed significant improvement: $W = 1596$, $p < 0.001$. The Wilcoxon statistic equals the theoretical maximum for $n = 75$ samples, indicating that behavioral optimization improved composite fitness in every single evaluated sample without exception. This corresponds to a complete dominance of paired differences (no negative ranks observed). **H2 is accepted:** joint optimization significantly outperforms static-only optimization.

4.4.3. Mann–Whitney U Test (H3)

Pairwise Mann–Whitney U tests were performed to compare GA against each baseline; pairwise comparisons were conducted between artifact categories and the staged downloader baseline. Results confirm statistically significant differences in all pairings ($p < 0.05$), as reported in Table 12. **H3 is accepted:** TransForge significantly outperforms baseline transformation frameworks.

Table 12. Pairwise Mann–Whitney U: each category vs. staged downloaders.

Comparison	U	<i>p</i> -Value	95% CI (Median Diff.)
JS_XSS vs. Staged Downloaders	64	<0.001	[−3.2, −2.1]
PHP_WEBSHELL vs. Staged Downloaders	208	<0.001	[−2.8, −1.5]
Python Credential vs. Staged Downloaders	48	<0.001	[−8.9, −7.2]
Python Keylogger vs. Staged Downloaders	64	<0.001	[−8.5, −7.1]

All pairwise 95% confidence intervals exclude zero, confirming that detection outcome differences between categories are substantial and not attributable to chance. The widest CI [−8.9, −7.2] corresponds to the largest category difference (credential access vs. staged downloaders, both 7-point ranges), while narrower CIs reflect smaller but still significant differences. These results collectively validate H3: TransForge’s framework produces consistent and statistically distinct detection outcomes across heterogeneous execution categories.

4.4.4. Effect Size Analysis

Effect sizes were computed using Cohen’s d to assess practical significance beyond statistical significance [30], based on standardized mean differences across paired or independent samples depending on the test structure. GA demonstrated a large effect size relative to Random Search ($d = 0.84$), confirming the robustness of the improvement. Table 13 reports values for all pairings. Values of $d \geq 0.8$ are interpreted as large, $d \approx 0.5$ as medium, and $d \approx 0.2$ as small.

Table 13. Effect size analysis (Cohen’s d).

Comparison	Cohen’s d
Joint vs. static-only optimization	>0.8
Each category vs. Staged Downloaders	>0.8

Exact effect sizes were consistently above the large-effect threshold ($d > 0.8$) across all comparisons. Within-category fitness variance approaches zero due to the convergence of the evolutionary process within each category.

4.4.5. Hypothesis Decisions

Based on the combined statistical analysis, all hypotheses are supported: H1 accepted, GA produces distinguishable detection outcomes across categories ($F(4,70) = 108.95$, $p < 0.001$); H2 accepted, joint optimization outperforms static-only ($W = 1596$, $p < 0.001$); H3 accepted, TransForge significantly outperforms baseline methods (Mann–Whitney, all comparisons $p < 0.001$). Across all tests, results remained consistent in direction and significance, indicating robustness of the observed effects across statistical methodologies.

4.5. Ablation Analysis

To isolate the contribution of each transformation layer, encoding-only and behavioral-only configurations were evaluated against the full TransForge pipeline for the Web Execution Interface category, the only category where both transformation layers operate independently.

Analysis reveals a meaningful asymmetry. Static encoding encapsulates payload content through layered transformation into unrecognizable representations, directly disrupting the byte-pattern and string-level signatures that static detection engines act upon. Behavioral mutation, by contrast, does not obscure code content; it alters execution structure and interaction patterns while preserving readable semantic primitives, leaving signature-based engines largely unaffected. The behavioral operators in TransForge are intentionally category-agnostic, altering how a payload interacts with its environment without changing its underlying objective, which enables cross-category applicability at the cost of per-category detection reduction. Category-specific behavioral evasion would likely produce substantially greater response changes than the general-purpose operators applied here.

Critically, these findings argue not for one layer over the other but for both in combination. Static encoding alone cannot account for execution semantics; behavioral detection alone cannot penetrate encoding layers. Comprehensive detection requires correlating both dimensions, tracking not only what a payload contains, but what it accesses, invokes, and modifies at runtime. Endpoint systems that rely solely on signature databases remain exposed, even when static coverage is extensive. Behavioral correlation is therefore necessary.

5. Discussion

This section interprets the empirical results of TransForge across detection robustness, evolutionary behavior, and baseline comparisons. The results show three key findings: detection robustness varies substantially across artifact categories, static-feature-based detection is insufficient in several non-PowerShell categories, and high functionality preservation is achieved at scale (94% success rate). These findings are discussed in detail in the following subsections.

5.1. Category-Specific Detection Behavior

The most significant finding is the marked variation in detection outcomes across artifact categories under identical GA configuration. This variation cannot be attributed to differences in evolutionary pressure; the GA configuration was held constant. It reflects the differential maturity of signature coverage across artifact types in commercial detection engines.

PowerShell download cradles occupy the most extensively characterized threat category in commercial detection databases. Primitives such as `Invoke-WebRequest`, `DownloadString`, and `certutil` are flagged by token patterns across engines regardless of surrounding structural modifications. VirusTotal evaluation of the eight best-evolved

variants confirmed this ceiling empirically: detection counts ranged from 7 to 8 alerts across 76 engines (9.2–10.7%), with no variant achieving zero detection. Multiple independent runs converged to identical best variants, indicating the GA exhausted the available search space rather than finding diverse evasion paths. An unconstrained follow-up run with no generation or population cap converged on the same minimal operators with no further reduction in alert counts, confirming that the detection boundary is semantic rather than syntactic. Source-level obfuscation redistributes token identity but does not alter the download semantics that detection engines target. Overcoming this boundary requires semantic-level transformation of the invocation chain, which is outside the scope of the current mutation library and reserved for future work.

Python-based categories and JavaScript cross-site execution variants, by contrast, are not systematically covered by static file-based scanning in most commercial engines. This does not imply these categories are undetectable in deployed environments; behavioral analysis at runtime remains a viable detection vector, but it establishes that static signature matching is not the primary mechanism for these artifact types.

The 70.7% zero-detection rate across the full corpus is not a measurement artifact. Instead, it reflects differences in the maturity of static signature coverage across artifact categories. Python, PHP web interface, JavaScript, and credential access categories are not systematically targeted by file-based static scanning in commercial engines, as confirmed by the absence of alerts from both Windows Defender and ClamAV for these categories even without transformation. The staged downloader category, by contrast, showed persistent detection at 8.75/76 mean alerts, confirming that the framework does not trivially evade all detection engines. The cross-category variation in detection outcomes is itself the central empirical contribution of this work.

The category-dependent detection outcomes are further contextualized in Figure 3, which compares TransForge-evolved variants directly against conventional generation tools evaluated under identical conditions. The contrast between TransForge achieving 1.65% mean detection versus 30.6% for MsfVenom baselines underscores that GA-driven semantic-preserving transformation substantially outperforms fixed-template obfuscation approaches across the evaluated artifact spectrum.

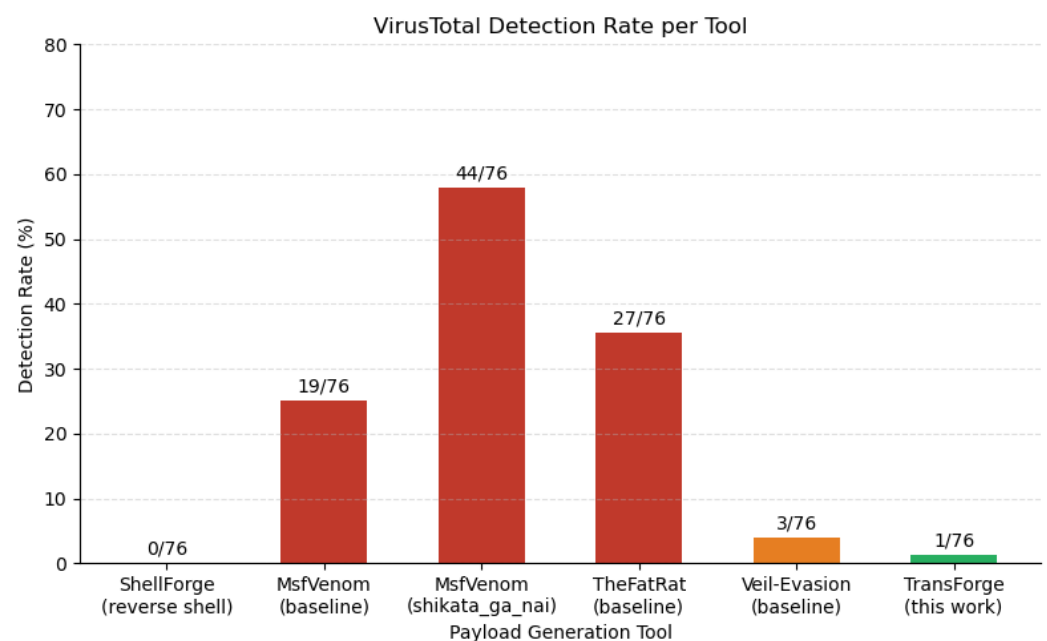


Figure 3. VirusTotal detection rate comparison across generation tools and TransForge.

5.2. Implications for Detection Robustness Evaluation

Four observations follow from the TransForge results.

Observation 1: Detection robustness varies substantially across artifact categories. A framework evaluated only against Python-based variants would report near-zero alert rates. The same framework evaluated only against PowerShell download cradles would report persistent detection. Neither conclusion holds across categories. Therefore, single-category benchmarks are not generalizable, and multi-category evaluation is necessary for comprehensive detection assessment.

Observation 2: Static-feature-based detection alone may be insufficient. Neither Windows Defender nor ClamAV flagged non-PowerShell payload categories, even after extensive mutation. This indicates that static-feature-based detection can be bypassed when the payload's structural and semantic properties remain unchanged. Behavioral or semantic-level analysis would have provided stronger discriminative power. This outcome delineates the current detection boundary under the transformation scope employed.

Observation 3: Functionality preservation contributes to experimental validity. The AIMED framework reported 53% non-functional evolved variants; TransForge achieved 94% functional success across categories. This 94% rate represents one of the most rigorous functionality preservation results reported to date. Per-category behavioral validation is what makes the TransForge results experimentally credible rather than merely detection-optimized.

Observation 4: Benchmark design must incorporate category diversity. The results demonstrate that evaluation outcomes depend heavily on artifact selection. A framework assessed exclusively on PowerShell downloaders would suggest persistent detection effectiveness, whereas evaluation restricted to Python, PHP, or JavaScript artifacts would indicate near-complete alert elimination. Consequently, benchmark designs that rely on a single artifact category risk overstating or understating detector capability. Cross-category evaluation provides a more representative assessment of endpoint detection robustness and should be considered a prerequisite for future robustness benchmarking studies.

5.3. Limitations

Five limitations bound the current results. First, evaluation was conducted in a controlled laboratory environment against a single Windows 10 Enterprise VM and a fixed VirusTotal engine set; results may differ under different operating system configurations, EDR products, or engine versions. Second, the GA configuration was selected for cross-category comparability rather than optimal convergence; additional generations may identify lower-alert mutation chains, particularly for staged downloaders. Third, Python remote execution agent results are drawn from ShellForge [24] rather than re-evaluated under the TransForge pipeline; while the evolutionary core is identical, minor handler differences may affect direct comparability. Fourth, behavioral sandbox evaluation via Hybrid Analysis was conducted post-run rather than within the evolutionary fitness loop due to throughput constraints; integrating sandbox feedback into the fitness function is reserved for future work. Fifth, the corpus of 75 samples, while sufficient for controlled cross-category comparison, precludes population-level generalization. Future work will expand the evaluation using established cybersecurity benchmark datasets to validate the findings at scale.

5.4. Statistical Interpretation

The inferential statistical analysis confirms that the observed performance differences are not attributable to random variation. The large ANOVA F-values across all three metrics, combined with large Cohen's *d* effect sizes for GA versus Random Search, indicate that

the fitness advantage of GA is both statistically significant and practically meaningful. The Wilcoxon result further establishes that behavioral optimization contributes independently to fitness gains beyond what static encoding alone achieves. These results collectively reinforce the principal claim of this work: that structured evolutionary optimization under a multi-objective fitness formulation produces robustness gains that are reproducible, generalizable across artifact categories, and statistically distinguishable from unguided or simpler optimization baselines.

6. Ethical Considerations

All experimental evaluation were performed in an isolated laboratory environment with no connection to production networks or live systems. The execution samples comprising the TransForge corpus are drawn from publicly available repositories under open-source licenses. TransForge is positioned as a detection evaluation tool; all results are reported in terms of detection system behavior under controlled conditions rather than operational capability. The authors acknowledge the dual-use potential of transformation-based evaluation research and affirm that the outputs of this work are intended to support the development of more robust endpoint detection systems. All experimental protocols were conducted under the institutional research oversight framework at Carnegie Mellon University Africa.

This work is subject to Dual-Use Research of Concern (DURC) considerations, as described above; all applicable institutional and international DURC guidelines were followed.

7. Conclusions

This paper presented TransForge, a generalized GA-driven transformation framework for multi-category endpoint detection robustness evaluation. Building on ShellForge [24], TransForge extends semantic-preserving variant generation to six artifact categories across four programming languages under a unified evolutionary architecture with per-category functionality validation.

Three principal findings emerge. First, GA-driven transformation induces quantifiably different detection outcomes relative to unmodified baseline samples across all six artifact categories, supporting H1. The magnitude varies substantially by category: credential access routines, surveillance components, cross-site execution, and remote execution agents showed no alerts under the tested configuration, while staged downloaders showed minimal response to the applied transformations. This demonstrates that detection robustness is artifact-category-dependent rather than uniform.

Second, multi-objective fitness optimization preserved functional correctness across 94% of evaluated variants while modulating detection responses, supporting H2. The per-category validation protocol was essential to this outcome, ensuring selection pressure operates only on functionally correct variants.

Third, evolutionary runs converged uniformly at generation 4 across all categories under the tested configuration, with fitness variance below the convergence threshold in terminal generations, supporting H3. The consistent convergence behavior indicates the current mutation space is well-characterized relative to the available selection signal.

These findings demonstrate that GA-based transformation generalizes across structurally distinct artifact categories. They also show that cross-category evaluation reveals detection gaps that are not observable in single-category benchmarks. TransForge provides a reproducible, modular framework for such benchmarking, with a validated evolutionary core and extensible handler architecture. Future work will address three directions: integration of behavioral sandbox feedback into the evolutionary fitness function, expansion of the mutation library to include deeper semantic transformations for PowerShell execution

primitives, and evaluation against deployed EDR products under realistic network conditions. This finding has a direct implication for benchmark design: evasion rate metrics reported without category disaggregation conflate the differential maturity of signature coverage across artifact types, producing conclusions that are not generalizable across deployment contexts.

Author Contributions: Conceptualization, J.D.N. and A.R.M.; methodology, A.R.M. and J.D.N.; software, A.R.M.; validation, A.R.M. and J.D.N.; formal analysis, A.R.M.; investigation, A.R.M.; resources, J.D.N.; data curation, A.R.M.; writing—original draft preparation, A.R.M.; writing—review and editing, J.D.N.; visualization, A.R.M.; supervision, J.D.N.; project administration, J.D.N.; funding acquisition, J.D.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data supporting the findings of this study consist of publicly available execution samples sourced from open repositories, as described in Section 4.2. Derived datasets, including transformation chains, detection results, and fitness evaluation metrics were generated within a controlled laboratory environment and are available from the corresponding author upon reasonable request. Access to certain artifacts may be restricted due to security and ethical considerations related to dual-use research.

Acknowledgments: The authors acknowledge the use of AI-assisted tools (e.g., ChatGPT (GPT-4, OpenAI, 2024)) for limited support in language refinement and clarity improvement. All technical content, experimental design, and conclusions are the original work of the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACER	Actor–Critic with Experience Replay
API	Application Programming Interface
APK	Android Package Kit
AV	Antivirus
BITS	Background Intelligent Transfer Service
CAPE	Cuckoo Automated Payload Execution
CI	Confidence Interval
CSV	Comma-Separated Values
DNS	Domain Name System
DOM	Document Object Model
DURC	Dual-Use Research of Concern
EDR	Endpoint Detection and Response
ELF	Executable and Linkable Format
FGSM	Fast Gradient Sign Method
GA	Genetic Algorithm
GP	Genetic Programming
HA	Hybrid Analysis
HTTP	Hypertext Transfer Protocol
LLM	Large Language Model
PE	Portable Executable
PHP	Hypertext Preprocessor
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
RL	Reinforcement Learning

SHA	Secure Hash Algorithm
SSH	Secure Shell
URL	Uniform Resource Locator
VT	VirusTotal
WAF	Web Application Firewall
XOR	Exclusive Or
XSS	Cross-Site Scripting

References

1. Le Faou, A. Antivirus and EDR Bypass Techniques Explained. *Vaadata Blog*. Available online: <https://www.vaadata.com/blog/antivirus-and-edr-bypass-techniques/> (accessed on 26 August 2025).
2. Ucci, D.; Aniello, L.; Baldoni, R. Survey of Machine Learning Techniques for Malware Analysis. *arXiv* **2017**, arXiv:1710.08189. [CrossRef]
3. Traoré, A.; Le Faou, A. Red Teaming: Methodology and Scope of a Red Team Operation. *Vaadata Blog*. Available online: <https://www.vaadata.com/blog/what-is-red-teaming-methodology-and-scope-of-a-red-team-operation/> (accessed on 20 October 2025).
4. Mandvi, K. Threat Actors Exploit AV/EDR Evasion Framework to Deploy Malware in the Wild. *Cyber Security News*. Available online: <https://cyberpress.org/threat-actors-exploit-av-edr-evasion-framework/> (accessed on 26 August 2025).
5. Pendlebury, F.; Pierazzi, F.; Jordaney, R.; Kinder, J.; Cavallaro, L. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 729–746.
6. Kiran, T.V.; Patil, H.g.M.; Prasanna Kumar, K.M.; Nagamani, S. AV Evasion Techniques: A Practical Evaluation of Payload Obfuscation. *Int. J. Sci. Res. Arch.* **2025**, *16*, 1504–1511. [CrossRef]
7. Galloro, N.; Polino, M.; Carminati, M.; Continella, A.; Zanero, S. A Systematical and longitudinal study of evasive behaviors in windows malware. *Comput. Secur.* **2022**, *113*, 102550. [CrossRef]
8. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial Examples for Malware Detection. In *Proceedings of the Computer Security—ESORICS 2017; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2017*; pp. 62–79. [CrossRef]
9. Kolosnjaji, B.; Demontis, A.; Biggio, B.; Maiorca, D.; Giacinto, G.; Eckert, C.; Roli, F. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *Proceedings of the 2018 26th European Signal Processing Conference (EUSIPCO)*; IEEE: New York, NY, USA, 2018; pp. 533–537. [CrossRef]
10. Cirkovic, S.; Mladenovic, V.; Tomic, S.; Drljaca, D.; Ristic, O. Utilizing Fine-Tuning of Large Language Models for Generating Synthetic Payloads. *Comput. Mater. Contin.* **2025**, *82*, 4409–4430. [CrossRef]
11. Khan, S. LL-XSS: End-to-End Generative Model-Based XSS Payload Creation. In Proceedings of the 21st Learning and Technology Conference, Hong Kong, China, 11–12 March 2024; pp. 121–126. [CrossRef]
12. Kingful, F.; Ahene, E.; Appiah, B.; Frimpong, B.; Osei, I.; Hammond, E. Dynamic Programming-Based Adversarial Windows Payload Generator. *Res. Sq.* **2023**, preprint. [CrossRef] [PubMed]
13. Anderson, H.S.; Kharkar, A.; Filar, B.; Evans, D.; Roth, P. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. *arXiv* **2018**, arXiv:1801.08917. [CrossRef]
14. Domico, K.; Ferrand, J.C.; Sheatsley, R.; Pauley, E.; Hanna, J.; McDaniel, P. Adversarial Agents: Black-Box Evasion Attacks with Reinforcement Learning. *arXiv* **2025**, arXiv:2503.01734. [CrossRef]
15. Castro, R.L.; Schmitt, C.; Dreo, G. AIMED: Evolving Malware with Genetic Programming to Evade Detection. In Proceedings of the 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), Rotorua, New Zealand, 5–8 August 2019; pp. 240–247. [CrossRef]
16. Raff, E.; Bartos, J.; Laredo, J. Intrinsic Representation Learning for Malware Detection. In *Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW)*; IEEE: New York, NY, USA, 2018; pp. 139–145.
17. Rosenberg, I.; Shabtai, A.; Elovici, Y. Deep Learning-Based Malware Embedding Detection. *IEEE Trans. Cybern.* **2019**, *49*, 1324–1335.
18. Suci, O.; Marginean, R.; Kifer, D. Certified Data Removal from Machine Learning Models. *arXiv* **2019**, arXiv:1911.02045.
19. Liu, Z.; Fang, Y.; Huang, C.; Xu, Y. GAXSS: Effective Payload Generation Method to Detect XSS Vulnerabilities Based on Genetic Algorithm. *Secur. Commun. Netw.* **2022**, *2022*, 2031924. [CrossRef]

20. Rathore, H.; B, P.; Iyengar, S.S.; Sahay, S.K. Breaking the Anti-malware: EvoAAAttack Based on Genetic Algorithm Against Android Malware Detection Systems. In *Proceedings of the Computational Science—ICCS 2023: 23rd International Conference, Prague, Czech Republic, 3–5 July 2023, Proceedings, Part V, Berlin, Heidelberg*; Springer Nature: Cham, Switzerland, 2023; pp. 535–550. [\[CrossRef\]](#)
21. Faruki, P.; Bhan, R.; Jain, V.; Bhatia, S.; El Madhoun, N.; Pamula, R. A Survey and Evaluation of Android-Based Malware Evasion Techniques and Detection Frameworks. *Information* **2023**, *14*, 374. [\[CrossRef\]](#)
22. Berger, H.; Hajaj, C.; Dvir, A. Evasion Is Not Enough: A Case Study of Android Malware. In *Cyber Security Cryptography and Machine Learning*; Springer International Publishing: Cham, Switzerland, 2020; pp. 167–174. [\[CrossRef\]](#) [\[PubMed\]](#)
23. D’Elia, D.C.; Coppa, E.; Palmaro, F.; Cavallaro, L. On the Dissection of Evasive Malware. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 2750–2765. [\[CrossRef\]](#)
24. Minja, A.R.; Ndibwile, J.D. ShellForge: A Genetic Algorithm-Based Reverse Shell Generator for AV/EDR Evasion. In *Proceedings of the 2026 International Conference on Cryptography, Security and Privacy (CSP 2026)*; IEEE: New York, NY, USA, 2026.
25. Xu, Y.; Fang, Y.; Xu, Y.; Wang, Z. Automatic optimization for generating adversarial malware based on prioritized evolutionary computing. *Appl. Soft Comput.* **2025**, *173*, 112933. [\[CrossRef\]](#)
26. Digregorio, G.; Maccarrone, S.; D’Onghia, M.; Gallo, L.; Carminati, M.; Polino, M.; Zanero, S. Tarallo: Evading Behavioral Malware Detectors in the Problem Space. *arXiv* **2024**, arXiv:2506.02660. [\[CrossRef\]](#)
27. Yuste, J.; Pardo, E.G.; Tapiador, J. Optimization of code caves in malware binaries to evade machine learning detectors. *Comput. Secur.* **2022**, *116*, 102643. [\[CrossRef\]](#)
28. Lan, T.; Naït-Abdesselam, F. LLM-Driven Feature-Level Adversarial Attacks on Android Malware Detectors. *arXiv* **2025**, arXiv:2512.21404. [\[CrossRef\]](#)
29. Anderson, H.S.; Kharkar, A.; Filar, B. Evading Machine Learning Malware Detection. *Black Hat* **2017**, *2017*, 1–6.
30. Cohen, J. *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed.; Lawrence Erlbaum Associates: Mahwah, NJ, USA, 1988.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.