

TrajectoryCheck: Trajectory-Level Invariant Validation for Behavioral Drift Detection in Generated Code

Aamish Ahmad
Independent Researcher
India

Abstract—Generated code can appear syntactically correct while still violating important behavioral constraints during execution. This paper presents TrajectoryCheck, a trajectory-level invariant validation framework for detecting behavioral drift in generated software systems. TrajectoryCheck models execution as a sequence of operational phases, assigns local and global invariants to those phases, and evaluates invariant preservation using AND-gate validation. The framework produces interpretable stability scores and drift maps that help identify where behavioral failures occur. We demonstrate the approach on progressively degraded authentication-system implementations, covering stable execution, total invariant collapse, and partial drift recovery.

Index Terms—Generated code, behavioral drift, invariant validation, software testing, trajectory analysis, authentication systems

I. INTRODUCTION

Modern code generation systems can produce programs that compile, pass simple tests, or appear correct under static inspection while still violating critical behavioral constraints at runtime. These failures are especially important in security-sensitive systems, where correctness depends not only on isolated functions but also on the consistency of behavior across a sequence of operations.

Traditional evaluation methods often focus on individual outputs, unit tests, or surface-level code properties. However, many failures emerge only across an execution trajectory, such as registration followed by login, session creation, access control, logout, and post-logout access attempts. A system may satisfy one phase while still breaking the global requirement across the full trajectory.

TrajectoryCheck addresses this gap by introducing a trajectory-centric validation framework. Instead of evaluating generated code as a static artifact, the framework evaluates whether local and global invariants remain preserved across evolving execution states.

The main contributions of this work are:

- A phase-based model for trajectory-level behavioral validation
- AND-gate invariant validation for local and global constraints
- A stability score for summarizing trajectory correctness
- Drift maps for localizing behavioral failures across phases

- A demonstration on degraded authentication-system implementations

II. CORE IDEA

The core idea is to treat generated software behavior as a trajectory rather than a single output. A trajectory is composed of ordered phases, and each phase must preserve its local invariant while also supporting the global invariant of the system.

The framework models validation as:

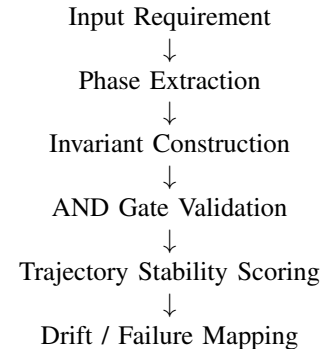
Requirement $\rightarrow$ Phases $\rightarrow$ Local Invariants $\rightarrow$ Global Invariant $\rightarrow$ AND Gates $\rightarrow$ Stability Score

TrajectoryCheck tracks:

- Local invariant preservation
- Global invariant stability
- Phase-wise failures
- Trajectory drift
- Partial drift recovery

III. SYSTEM ARCHITECTURE

The architecture follows a sequential validation pipeline:



The input requirement defines the intended behavior. Phase extraction identifies the key operational steps in the generated system. Invariant construction assigns expected properties to each phase, and AND-gate validation checks whether local and global constraints hold together. The final output is a stability score and a phase-wise drift map.

IV. VALIDATION PIPELINE

The validation pipeline begins with an input requirement, such as a secure authentication requirement. The framework extracts operational phases from the generated implementation and associates each phase with local invariants. A global invariant represents the system-wide behavioral constraint that must remain true across the complete trajectory.

Each phase is validated using AND-gate logic:

$$Gate_i = L_i \wedge G_i \quad (1)$$

where L_i is the local invariant result for phase i and G_i is the global invariant result at phase i . A gate passes only when both invariant checks pass.

The trajectory stability score is calculated as:

$$S = \frac{G_p}{G_t} \times 100 \quad (2)$$

where:

- S = trajectory stability score
- G_p = gates passed
- G_t = total gates

V. EXPERIMENTAL PROGRESSION

Three progressively degraded authentication systems were evaluated. The authentication domain was selected because it naturally contains sequential behavioral requirements, including user registration, password handling, login, session validation, logout, and access-control behavior.

A. Stable Trajectory

The stable implementation preserves all local and global invariants across the full execution trajectory.

- Stability: 100%
- Gates Passed: 5/5
- Classification: Stable

B. Total Drift

The total-drift implementation fails every validation gate, indicating complete collapse of trajectory-level behavioral consistency.

- Stability: 0%
- Gates Passed: 0/4
- Classification: Drifting

C. Partial Drift Recovery

The partial-recovery implementation preserves one gate while failing the remaining checks, showing that a system may recover some local behavior while still violating the global invariant.

- Stability: 25%
- Gates Passed: 1/4
- Classification: Partial Drift

Demo	Gates	Stability	Classification
Stable	5/5	100%	Stable
Total Drift	0/4	0%	Drifting
Partial Recovery	1/4	25%	Partial Drift

TABLE I
TRAJECTORY STABILITY EVALUATION RESULTS

VI. RESULTS

Table I summarizes the observed trajectory stability across the three demonstrations.

The results show that TrajectoryCheck separates fully stable behavior from complete drift and partial recovery. This distinction is useful because generated systems may not fail uniformly. Instead, failures can appear at specific phases and then propagate into later phases.

VII. FAILURE TAXONOMY

Observed trajectory failures include:

- Plaintext password storage
- Missing session invalidation
- Broken authentication edge cases
- Global invariant collapse
- Phase inconsistency propagation

These failures demonstrate why trajectory-level analysis is useful. A generated implementation may appear functional under a simple login test while still failing after logout, mis-handling invalid credentials, or allowing state inconsistencies to propagate.

VIII. LIMITATIONS

This work is an initial prototype and has several limitations:

- Rule-based invariant extraction
- Toy authentication domain
- No probabilistic scoring
- No formal verification layer
- Limited benchmark diversity

IX. FUTURE WORK

Future extensions include:

- LLM-generated invariant extraction
- Temporal trajectory graphs
- Probabilistic drift estimation
- Multi-agent trajectory validation
- Automated repair generation

X. CONCLUSION

TrajectoryCheck demonstrates that behavioral validation can be modeled as a trajectory-level invariant preservation problem. By evaluating local and global constraints across execution phases, the framework provides interpretable stability scores, drift localization, and phase-aware failure analysis for generated systems. The results suggest that trajectory-level validation can complement static evaluation and unit testing when assessing generated code behavior.

ARTIFACT AVAILABILITY

The project artifact is available at:

<https://github.com/aamish-ahmad/trajectorycheck>