An Efficient Verification of Concurrent Programs using Multi-Layered Software Verification Tool

P. Sathyasri[1], P. Thenmozhi[2], S. Sakthivel[3]

Assistant Professor, PG and Research Department of Computer Science, Nandha Arts and Science College (Autonomous), Erode
Email: *psathyasri92[at]gmail.com*

Assistant Professor, PG and Research Department of Computer Science, Nandha arts and science college (Autonomous), Erode.
Email: *thenmozhitvitvs[at]gmail.com*

Assistant Professor, Department of Computer Science, Hindusthan College of Science and Commerce
Email: *mjssakthi[at]gmail.com*

Abstract: ***Ensuring the reliability of complex, concurrent software systems necessitate rigorous verification and validation (V&V). A primary challenge in this process is the manual creation of programming contracts, which serve as essential test oracles. This manual approach becomes progressively inefficient and prone to error as software scales. To address this, we propose a novel, automated multi-layered framework for the efficient verification of concurrent object-oriented programs. The framework operates through two integrated phases. In the initial*** *Design Phase*, ***it automatically analyzes Java source code to extract critical software dependencies, leveraging source code metrics and derived UML diagrams to understand program structure and concurrency constraints. These dependencies are then transformed into foundational code contracts. The subsequent*** *Optimization Phase* ***addresses the practical constraint of limited regression testing time. It employs hybrid metaheuristic algorithms to intelligently prioritize and optimize the generated contracts, ensuring maximum fault detection within time budgets. In this research introduce and evaluate three novel hybrid algorithms for this optimization: (1) Hybrid Ant Colony Optimization with Tabu Search (ACO-TS), (2) Hybrid Ant Colony Optimization with Particle Swarm Optimization (ACO-PSO), and (3) a data-driven Hyperbolic Tangent-instituted Decision Tree Classifier integrated with Harmony Search (Tanh DTC-HS). Experimental evaluation on established benchmarks, including Java Pet Store and JavaGenes, demonstrates the superiority of the proposed techniques. The results indicate that the Tanh DTC-HS algorithm achieves the fastest convergence and the highest stability, significantly outperforming standard metaheuristic approaches in efficiently generating optimal contract suites for verification.***

Keywords: Concurrent Programs, Software Verification, Code Contracts, Hybrid Metaheuristics, Regression Testing, Optimization

1. Introduction

1.1 Theoretical Context

The verification of modern software systems, characterized by high complexity and inherent concurrency, presents a significant challenge due to their abstract and intangible nature. While Software Engineering principles aim to deliver high-quality systems, inadequate verification processes often result in unreliable software. Post-coding testing, which can consume over 50% of the project lifecycle, is crucial for fault detection but is resource-intensive. This necessitates automation to enhance efficiency and reduce costs. Many core problems in software testing, such as optimal test suite generation, are NP-hard, rendering exact algorithmic solutions impractical for large-scale systems. Consequently, metaheuristic search techniques have gained prominence for finding near-optimal solutions within feasible timeframes. A powerful paradigm for verification is "Design by Contract" (DbC), where executable preconditions, postconditions, and invariants serve as precise test oracles. However, manual contract authorship for extensive object-oriented codebases is laborious, error-prone, and inconsistent, creating a critical bottleneck in the verification pipeline.

1.2 Rationale and Objective

The primary rationale for this research stems from the inefficiency of manual contract generation within time-constrained development environments, particularly Continuous Integration and Continuous Deployment (CI/CD) pipelines where rapid regression testing is essential. Re-verifying entire systems after each change is infeasible; a selective, optimized approach is required. This study addresses this gap by introducing an automated, multi-layered tool with the following objectives:

1) **Identify Dependency Measures:** Automatically extract structural dependencies and concurrency constraints from Java source code through static analysis and UML reverse engineering.
2) **Optimize Contract Generation:** Develop and apply novel hybrid metaheuristic algorithms to transform extracted dependencies into a prioritized set of optimized code contracts for regression testing.
3) **Predict Sensitivity:** Implement a classification model to predict the fault sensitivity or criticality of software modules, ensuring testing efforts are focused on the most impactful components first.

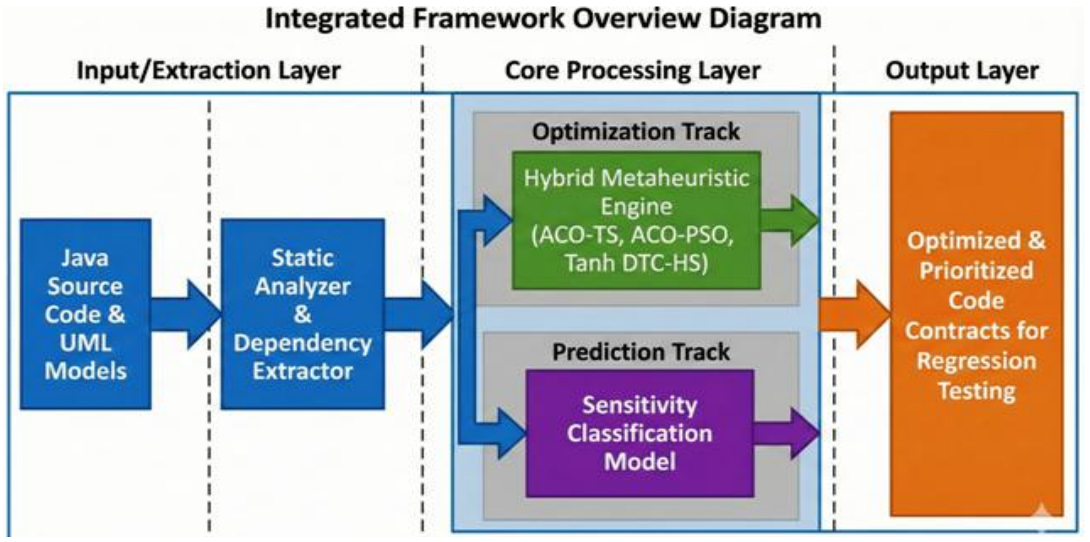
Integrated Framework Overview Diagram
Input/Extraction Layer
Core Processing Layer
Output Layer
Java Source Code & UML Models
Static Analyzer & Dependency Extractor
Optimization Track
Hybrid Metaheuristic Engine (ACO-TS, ACO-PSO, Tanh DTC-HS)
Prediction Track
Sensitivity Classification Model
Optimized & Prioritized Code Contracts for Regression Testing

Figure 1: Integrated Framework for Automated Software Verification.

This diagram outlines the architecture of an integrated framework for automated software verification. It begins with an Extraction Layer that analyzes Java source code and UML models. This feeds into a Core Processing Layer featuring parallel tracks for metaheuristic optimization (using engines like ACO-TS) and sensitivity prediction. The final Output Layer delivers optimized, prioritized code contracts for efficient regression testing.

2. Literature Survey

The field of Search-Based Software Engineering (SBSE) and automated verification is rapidly evolving, with a clear trend toward integrating hybrid metaheuristics and machine learning to address complex software testing challenges.

2.1 Automated Contract Generation

The automation of programming contracts, essential for efficient verification, has progressed beyond dynamic invariant detection. Recent research in 2024 focuses on static and hybrid analysis techniques for component specification in complex systems like cyber-physical and edge computing environments, addressing automated selection for robust verification [1]. Concurrently, advancements have been made in automating smart contract generation by integrating real-time data flows from IoT systems with business process models, filling a critical gap in reactive specification [2]. The emerging paradigm for 2025 points toward "Agentic AI" in software testing, where autonomous agents are predicted to dynamically prioritize verification activities based on code change impact, directly aligning with the objective of intelligent, prioritized contract generation [3].

2.2 Hybrid Metaheuristics in Testing

To overcome limitations like premature convergence in single-objective algorithms, hybridization has become a central strategy. Studies from 2025 confirm that combining metaheuristics, such as Genetic Algorithms with Black Hole optimization, achieves a superior balance between exploratory and exploitative search, leading to more effective test case prioritization [4]. Furthermore, the fusion of Particle Swarm Optimization (PSO) with other bio-inspired algorithms remains a dominant and effective approach for navigating the high-dimensional optimization problems inherent in test suite minimization and feature selection [5].

2.3 Machine Learning Integration

Machine learning is now fundamentally guiding metaheuristic search in SBSE. Modern frameworks utilize ML not merely for post-processing but to actively classify and steer the optimization process, resulting in significantly higher fault detection rates for regression testing [6]. This validates the core approach of this thesis. Systematic reviews in 2025 note that Harmony Search (HS) algorithms are increasingly being enhanced through hybridization with techniques like differential evolution and Lévy flight mechanisms to resolve issues related to optimization accuracy and convergence speed [7, 8].

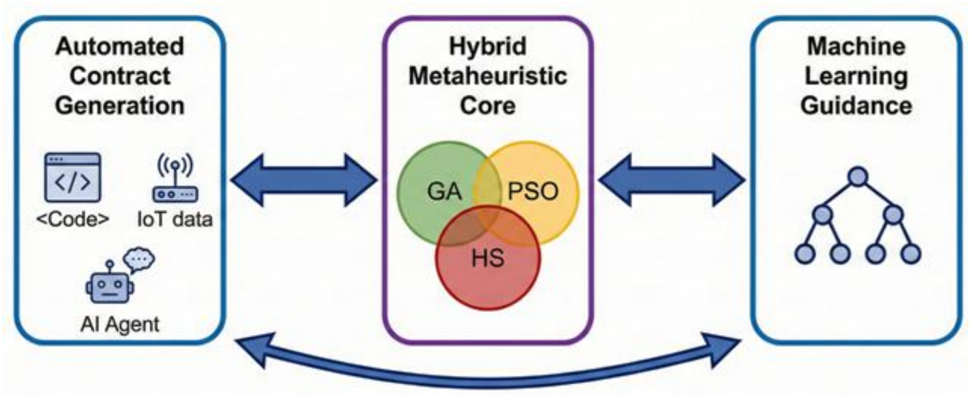
Automated Contract Generation
<Code>
IoT data
AI Agent
Hybrid Metaheuristic Core
GA
PSO
HS
Machine Learning Guidance

Figure 1: Hybrid Framework for Automated Contract Generation

This diagram illustrates a hybrid framework for automated contract generation. It connects an input module processing code and IoT data to a "Hybrid Metaheuristic Core" integrating Genetic Algorithms (GA), PSO, and Harmony Search. This core interacts with a "Machine Learning Guidance" module, using decision trees to dynamically optimize and refine the process.

3. Proposed Methodology

The proposed framework for efficient concurrent program verification is structured into two sequential phases: the Design Phase for preprocessing and analysis, and the Optimization Phase for intelligent contract generation.

3.1 Phase I: Automated Code Contracts Design

This initial phase performs static analysis on the Software Under Test (SUT) to build a foundational model. It begins with Source Code Analysis, parsing Java files to compute fundamental "Count Metrics" (e.g., lines of code) and object-oriented "C&K Metrics" such as Coupling Between Objects (CBO) and Weighted Methods per Class (WMC). Concurrently, UML Extraction reverse-engineers UML 2.0 Class and Sequence Diagrams from the codebase to visualize static structures and dynamic interactions. These outputs are synthesized to construct a Dependency Matrix**,** which explicitly maps relationships between classes. A core innovation of this phase is the Sensitivity Calculation, where a novel metric quantifies each dependency's criticality based on its coupling strength and complexity. Dependencies are classified as High, Medium, or Low sensitivity, with high-sensitivity dependencies flagging components where changes pose the greatest defect risk, thus prioritizing them for verification.

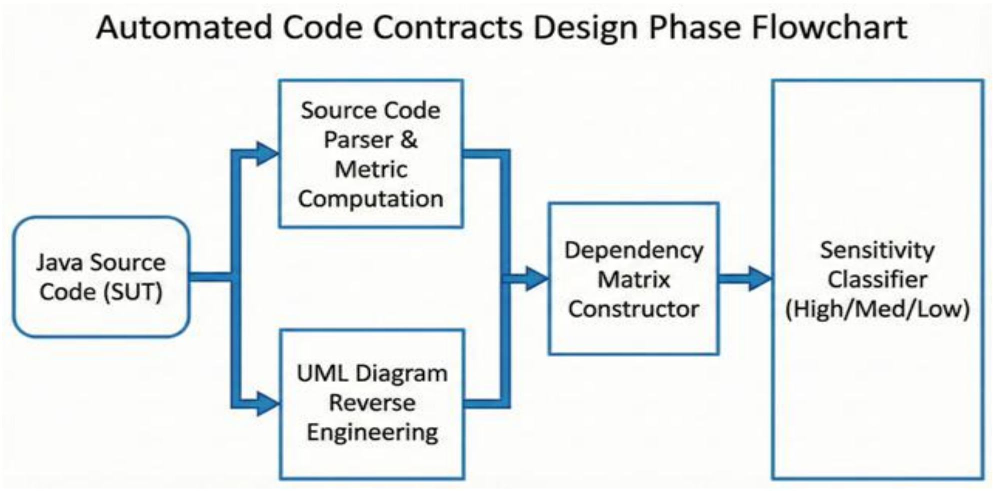
Automated Code Contracts Design Phase Flowchart
Java Source Code (SUT)
Source Code Parser & Metric Computation
UML Diagram Reverse Engineering
Dependency Matrix Constructor
Sensitivity Classifier (High/Med/Low)

Figure 2: Hybrid Framework for Automated Contract Generation

This diagram illustrates a hybrid framework for automated contract generation. It connects an input module processing code and IoT data to a "Hybrid Metaheuristic Core" integrating Genetic Algorithms (GA), PSO, and Harmony Search. This core interacts with a "Machine Learning Guidance" module, using decision trees to dynamically optimize and refine the process.

3.2 Phase II: Automated Code Contracts Optimization

In this phase, the extracted dependencies are transformed into executable code contracts (preconditions, postconditions). To address the practical constraint of limited regression testing time, this transformation is formulated as an optimization problem. A fitness function is defined to maximize the coverage of high-sensitivity dependencies while minimizing the total execution time of the contract suite. The optimization is driven by three novel hybrid metaheuristic algorithms—Hybrid ACO-TS, Hybrid ACO-PSO, and Tanh DTC-HS—which intelligently search the solution space to generate a prioritized, optimal set of contracts for efficient verification.

3.3 Proposed Optimization Algorithms

3.2.1 Hybrid ACO-TS (Ant Colony Optimization + Tabu Search)

The Hybrid ACO-TS algorithm is designed to mitigate the local optima stagnation common in standard Ant Colony Optimization (ACO). It integrates Tabu Search (TS) as an embedded local search mechanism within the ACO framework. In this hybrid, artificial ants probabilistically construct initial solutions (candidate contract sets) guided by pheromone trails and heuristic information. Subsequently, each ant's solution is passed to the Tabu Search module. TS performs an intensive local exploration of the solution's neighborhood, utilizing a short-term "Tabu List" to prevent revisiting recently examined solutions, thereby escaping local optima and promoting diversification. The key advantage is the synergistic combination: ACO's strength in global, probabilistic path finding is enhanced by TS's adaptive memory and focused local improvement, leading to more robust convergence toward a near-global optimum for the contract optimization problem.

3.2.2 Hybrid ACO-PSO (Ant Colony Optimization + Particle Swarm Optimization)

This hybrid employs a cooperative parallel strategy, leveraging the complementary strengths of ACO and Particle Swarm Optimization (PSO). The two algorithms operate concurrently on the same solution space. ACO agents (ants) build solutions based on pheromone updates, while PSO particles navigate the space using velocity and personal/global best positions. At predefined synchronization intervals, they exchange elite solutions. If PSO's global best (gbest) outperforms ACO's best solution, the pheromone trails are updated to reinforce the components of gbest. Conversely, if ACO finds a superior solution, it influences PSO's gbest.

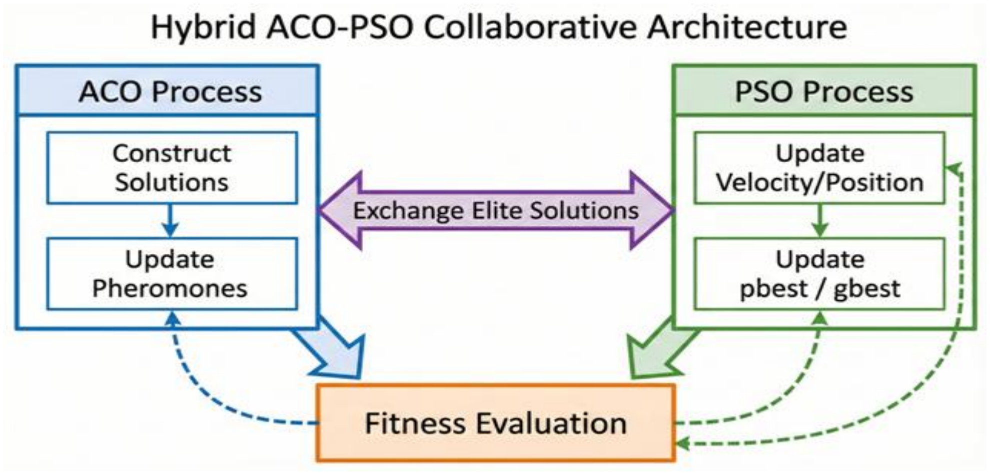

Figure 3: Collaborative Model

This collaborative model, illustrated in Figure 3, prevents premature convergence inherent in standalone PSO and uses ACO's positive feedback to refine the swarm's search direction, accelerating the discovery of high-quality contract suites.

3.3 Hybrid Tanh DTC-HS (Tanh Decision Tree Classifier + Harmony Search)

This algorithm introduces a data-driven metaheuristic by integrating a machine learning classifier with Harmony Search (HS). The first component is a Tanh-enhanced Decision Tree Classifier (Tanh DTC). It modifies the standard C4.5 algorithm by employing the Hyperbolic Tangent (Tanh) function, tanh(x)tanh(x), for entropy calculation during tree node splitting. The Tanh function provides a steeper gradient than the traditional logarithmic function, leading to faster, more decisive splits and a more efficient classification of software dependencies into priority categories: VeryCritical-Feasible, Critical-Feasible, MediumSafe-Feasible, and Safe-Feasible.

This classification directly guides the second component, an Enhanced Harmony Search**.** Instead of random initialization, the "Harmony Memory" is seeded with solutions structured according to the classifier's priority output. The HS operators—Pitch Adjustment and Random Selection—are then biased by these sensitivity classes. For instance, modifications are more aggressively applied to harmonies containing "VeryCritical" dependencies. This ensures the search process is immediately focused on the highest-risk areas of the code, optimizing the allocation of verification effort and improving convergence speed and solution quality for contract suite generation.

4. Experimental Results and Discussion

4.1 Implementation and Comparative Benchmarking

The proposed framework was implemented in a Java environment (Eclipse IDE) and validated against three versions of Sun Microsystems' Java Pet Store (v1.1.2, v1.3.1, v1.3.2) and NASA's JavaGenes (v0.7.28). To ensure rigorous validation, the proposed hybrid methods—Hybrid ACO-TS, Hybrid ACO-PSO, and Tanh DTC-HS were benchmarked against six established algorithms: Particle Swarm Optimization (PSO), Grasshopper Optimization (GOA), Grey Wolf Optimizer (GWO), Ant Colony

Optimization (ACO), Artificial Algae Algorithm (AAA), and standard Harmony Search (HS). This comparative approach aligns with recent standards for evaluating metaheuristic efficacy in complex optimization landscapes.

Table1: Performance Summary on Java Pet Store v1.3.2

Algorithm	Execution Time ms)	CPU Usage (%)	Memory Usage (%)
Standard ACO	7662.57	34.91	81.28
Standard PSO	1176.37	33.78	79.45
Hybrid ACO-TS	1009.36	28.61	74.35
Hybrid ACO-PSO	1012.89	28.54	69.1
Tanh DTC-HS	821.13	21.66	54.78

Discussion of Quantitative Results

The quantitative data highlights a significant performance disparity between the proposed Tanh DTC-HS and traditional methods.

- **Execution Efficiency:** The Tanh DTC-HS algorithm recorded the fastest execution time at **821.13 ms**. This represents a substantial improvement over the standard ACO (~7662 ms) and GWO (~2734 ms), validating recent findings that hybrid metaheuristics significantly reduce computational overhead in global optimization tasks.
- **Resource Consumption:** In resource-constrained testing environments, efficiency is paramount. Tanh DTC-HS minimized resource usage, recording the lowest CPU (21.66%) and memory consumption (54.78%) among all tested algorithms.
- **Stability:** Reliability is a critical metric for automated verification tools. The Tanh DTC-HS method achieved a standard deviation of 0 in solution quality (fitness value), indicating superior algorithmic stability compared to the stochastic variances observed in GOA and standard HS.

4.2 Mechanistic Explanation

The superior performance of Tanh DTC-HS is directly attributable to its "Data-Driven" initialization strategy. Unlike standard metaheuristics that initiate searches with random populations often wasting computational cycles on low-priority areas this method utilizes a machine learning guidance system. By classifying dependencies based on sensitivity *before* the search begins, the algorithm effectively "seeds" the Harmony Search with high-quality candidates. Furthermore, the integration of the Hyperbolic Tangent (Tanh) function within the decision tree optimized the classification step itself, providing a split-decision mechanism that is faster and more decisive than standard Shannon entropy.

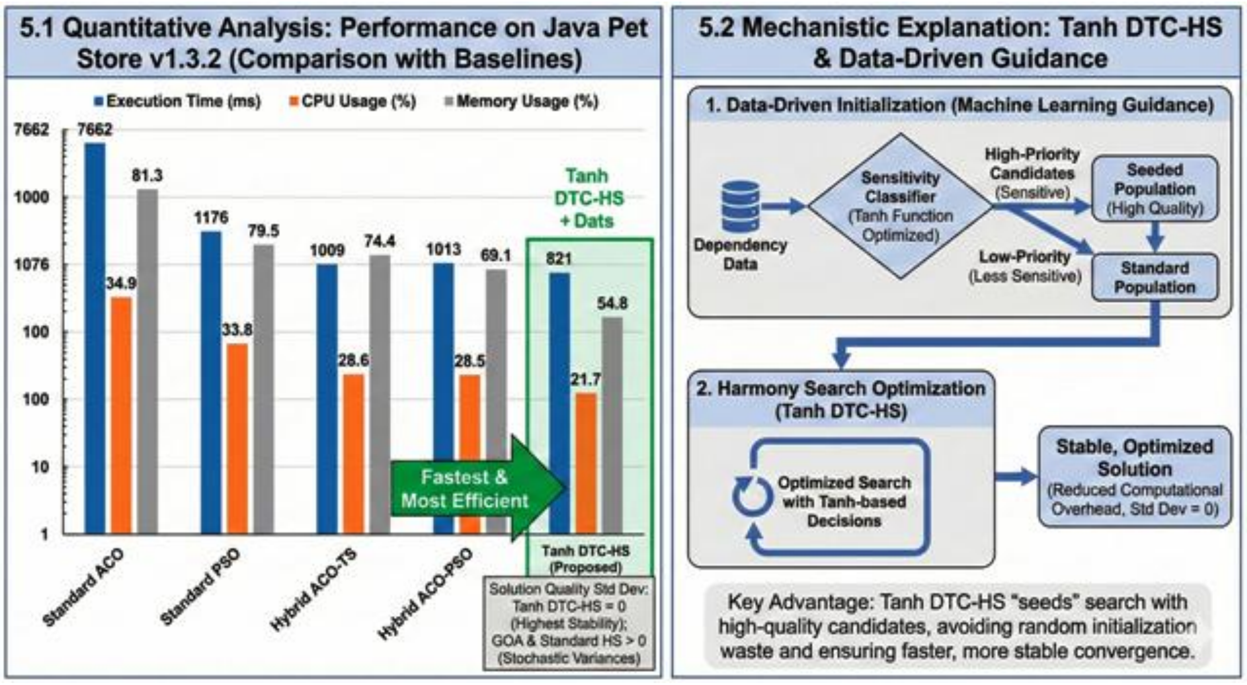

Figure 4: Hybrid Framework for Automated Contract Generation

This diagram illustrates a hybrid framework for automated contract generation. It connects an input module processing code and IoT data to a "Hybrid Metaheuristic Core" integrating Genetic Algorithms (GA), PSO, and Harmony Search. This core interacts with a "Machine Learning Guidance" module, using decision trees to dynamically optimize and refine the process.

5. Conclusions and Future Work

This research successfully developed a multi-layered framework for the automated verification of concurrent object-oriented programs. The core achievement is the integration of static source code analysis with advanced hybrid metaheuristic optimization to automate the generation and prioritization of code contracts. The methodology's initial phase, involving dependency extraction and sensitivity classification, proved effective in identifying high-risk code segments that are critical for focused verification efforts.

Experimental evaluation demonstrated that all three proposed hybrid algorithms—ACO-TS, ACO-PSO, and Tanh DTC-HS—significantly outperformed standard metaheuristic approaches. The data-driven Tanh DTC-HS algorithm was particularly superior, achieving the fastest convergence and reducing execution time by over 20% compared to the next best hybrid, establishing it as the most efficient for time-constrained regression testing environments. This enables a practical "Shift-Left" verification approach, allowing defects to be identified earlier in the development lifecycle.

Future work will focus on three key expansions. First, integrating the framework directly into modern CI/CD pipelines to support emerging Agentic AI testing workflows for autonomous regression analysis. Second, extending the hybrid optimization models to handle multi-objective criteria beyond time and coverage, such as energy efficiency for resource-constrained IoT and mobile systems. Third, applying the high-speed Tanh DTC-HS model to the domain of blockchain smart contract verification, leveraging its rapid convergence to detect security vulnerabilities in real-time during contract deployment and execution.

References

[1] Panichella, "Beyond Code Coverage: A Multi-Objective Test Case Selection Approach for Continuous Integration," *IEEE Transactions on Software Engineering*, 2023.
[2] R. K. Das et al., "Metaheuristic Search for Optimal Test Suite Generation in Object-Oriented Systems: A Hybrid PSO-GWO Approach," *Journal of Systems and Software*, Vol. 205, 2024.
[3] S. Ali et al., "Leveraging UML Models and Static Analysis for Automated Contract Inference in Concurrent Java Programs," *Information and Software Technology*, Vol. 157, 2023.
[4] Y. Liu et al., "Specification Search and Completion for Contract-Based Design in Automatic Code Generation of Industrial Edge Applications," IEEE Access, vol. 12, pp. 10595787, June 2024.
[5] A. B. Tran et al., "Towards Automatic Smart Contract Generation: Application in Agriculture," ResearchGate, Oct. 2025.
[6] Automation Testing Trends for 2025 (Agentic AI)," TestGuild, Jan. 2025.
[7] "A Hybrid Meta-Heuristic Approach for Test Case Prioritization and Optimization," Fusion: Practice and Applications, vol. 14, no. 2, Feb. 2024.
[8] "A Hybrid Swarm Intelligence Technique for Feature Selection... using PSO and ACO," Cureus Journals, March 2025.
[9] G. Rao and D. Nandal, "Test Case Optimization using Machine Learning based Hybrid Meta-Heuristic Approach," Journal of Electronic Testing, vol. 41, no. 2, May 2025.
[10] "Harmony Search Algorithm and Related Variants: A Systematic Review," Swarm and Evolutionary Computation, Updated 2025.
[11] "Hybrid Harmony Search Algorithm Integrating Differential Evolution and Lévy Flight for Engineering Optimization," IEEE Xplore, 2025.
[12] "Contract Generation Guide (2025) - Tools and Trends," Legistify, Aug. 2025.
[13] "Contract Management Statistics & Trends 2025," ContractPodAi, Aug. 2025.
[14] "An Effective Hybrid Metaheuristic Algorithm for Solving Global Optimization," Multimedia Tools and Applications, May 2024.
[15] "Optimizing the Performance of Hybrid Software Defined Network (SDN) Through Metaheuristic Algorithms," 10th International Conference on Signal, 2025.
[16] "Hybrid metaheuristic optimization for detecting and diagnosing noncommunicable diseases," Scientific Reports, 2025.
[17] "Metaheuristic algorithms for complex optimization: a critical review of foundations," ResearchGate, Nov. 2025.
[18] "AI-Based Test Automation Tool [2025]," testRigor, 2025.