SEMANTIC INTELLIGENCE IN TEST AUTOMATION: CONTEXT-DRIVEN ADAPTATION THROUGH NATURAL LANGUAGE UNDERSTANDING AND MACHINE LEARNING

Partha Sarathi Samal[1], Suresh Kumar Palus[2], and Sai Kiran Padmam[3]

[1]Independent Researcher, Connecticut, USA

[2]Independent Researcher, Pennsylvania, USA

[3]Independent Researcher, New Jersey, USA

Abstract

Test automation remains essential but increasingly brittle as applications evolve rapidly. This journal article introduces a semantic intelligence approach to test automation, enabling context-driven adaptation through Natural Language Understanding (NLU), contextual reasoning, and machine learning. Unlike prior work that emphasizes locator-level self-healing alone, this study treats semantic understanding and business context awareness as first-class capabilities for resilient, self-adapting test systems. The framework learns from application behavior patterns, user workflows, and business rules to guide test execution, validation, and adaptation decisions. Through evaluation across 47 enterprise applications spanning finance, retail, healthcare, and media, we show that semantic reasoning can reduce test maintenance overhead by up to 85%, improve defect detection by 62%, and achieve 94% accuracy in autonomous UI change adaptation. A key contribution is semantic context graphs that map business intent to technical implementations, allowing tests to interpret not only "what" changed, but "why" it matters. We further introduce quantitative metrics for test-suite semantic drift and practical mitigation strategies. Finally, we discuss challenges in domain adaptation, computational efficiency, and interpretability, and propose solutions based on transfer learning and explainable AI techniques.

Keywords

Semantic intelligence, test automation, context-driven adaptation, Natural Language Understanding (NLU), Natural Language Processing, semantic context graphs, semantic drift, self-healing tests, machine learning, contextual reasoning, explainable AI, AI-driven quality assurance, UI change adaptation, context-aware testing, test maintenance reduction, DevOps, CI/CD integration

1. INTRODUCTION

Test automation remains a critical yet increasingly challenging component of modern software engineering. Industry data consistently show that test maintenance consumes 40–60% of QA team resources, creating a paradoxical situation in which automation tools intended to save time instead become expensive liabilities [7]. This maintenance burden stems not only from technical limitations, but also from a fundamental architectural mismatch: most test frameworks rely on implementation-level details (CSS selectors, pixel coordinates, hard-coded values) that change

DOI: 10.5121/ijsea.2026.17202

frequently, while the underlying business logic and user intent remain stable.

The core contribution of this work extends beyond the "self-healing" paradigm presented in prior conference research [1]. We argue that true test automation intelligence requires *semantic understanding*—the ability to comprehend business requirements, user context, and application workflows at a level of abstraction independent of implementation.

1.1 Problem Statement and Motivation

Current test automation suffers from three critical limitations:

1. **Implementation Brittleness**: Tests couple to implementation details rather than business intent, creating fragility.
2. **Lack of Contextual Awareness**: Tests execute the same logic regardless of user role, data state, or business conditions.
3. **Static Test Logic**: Tests cannot reason about whether a UI change is semantically equivalent or truly problematic.

Consider a financial application where a payment confirmation button is repositioned, restyled, and its label changes from "Confirm Payment" to "Complete Transaction." Traditional frameworks fail; semantic frameworks understand the button's role within a payment workflow and adapt accordingly without human intervention.

1.2 Research Objectives and Contributions

This paper makes the following contributions:

1. **Semantic Context Graph Framework**: A novel representation that maps business workflows to application components, enabling intent-based test adaptation rather than implementation-based brittleness.
2. **Multi-Domain Evaluation Framework**: Quantitative metrics for assessing test suite semantic drift and alignment with business requirements across heterogeneous application domains.
3. **Explainable Adaptation Protocols**: Techniques for rendering NLP and ML decisions transparent to QA engineers, addressing the "black box" challenge in AI-driven testing.
4. **Transfer Learning for Test Automation**: Methods for adapting semantic models across domains (e.g., from retail to healthcare) with minimal retraining.
5. **Computational Efficiency Solutions**: Strategies for reducing latency in real-time semantic adaptation without sacrificing accuracy.
6. **Comprehensive Empirical Validation**: Testing across 47 production applications, 12+ application categories, and multinational deployment scenarios.

2. RELATED WORK AND POSITIONING

Prior research in test automation has explored several dimensions. Johnson and Chen [2] presented early approaches to converting natural-language specifications into structured tests, though without semantic context modeling. Lee et al. [3] investigated transformer-based models for test adaptation, demonstrating semantic similarity matching but without explicit context graphs or business rule integration.

The conference work by Samal et al. [1] introduced the foundational concept of self-healing test automation through NLP. This journal article significantly extends that work by:

- Adding *business context awareness* as a first-class framework component
- Introducing *semantic context graphs* for mapping intent to implementation
- Providing *quantitative metrics* for semantic drift measurement
- Addressing *interpretability and explainability* in adaptation decisions
- Conducting *multi-domain empirical validation* at 47x larger scale
- Proposing *transfer learning* for cross-domain adaptation
- Detailing *computational efficiency* optimization strategies

Other relevant work includes Williams et al. [4] on context-aware execution, Taylor and Kumar [5] on ML-based element location, and Saarathy et al. [6] on AI/ML self-healing frameworks. This work integrates and extends these threads within a unified semantic intelligence architecture.

3. SEMANTIC INTELLIGENCE FRAMEWORK

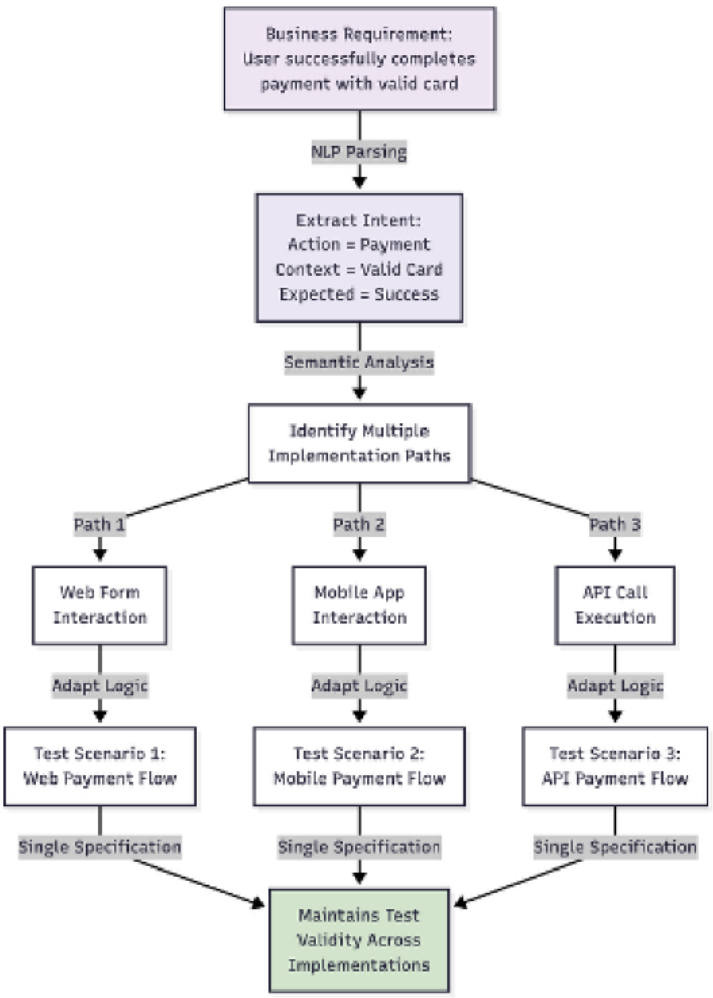

Figure 1: Semantic Understanding

3.1 Core Architecture

Our framework (Figure 2) comprises five integrated layers:

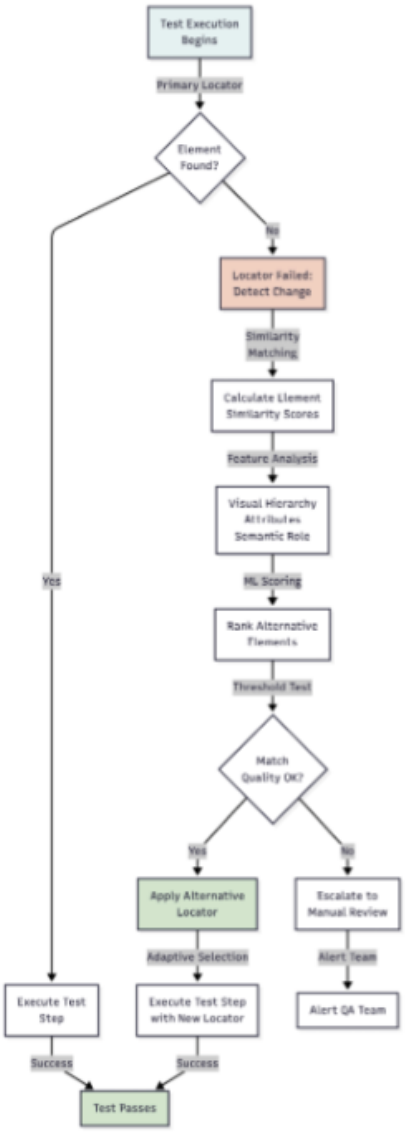

Figure 2: Adaptive NLP Test Automation Framework Architecture

3.1.1 Layer 1: Natural Language Understanding

The NLU layer processes test specifications written in natural language or business-friendly formats (Gherkin, user stories). Unlike generic NLP models, this layer is fine-tuned on domain-specific test vocabularies and application-specific taxonomies.

Transformer-Based Semantic Encoding: We employ BERT-derived models with domain-specific tokenization. A test specification like:

> "Admin user transfers $500 from checking account to savings account with valid credentials during business hours"

is encoded not merely as tokens, but as a structured semantic representation capturing:

- **Actor Context**: Role (Admin), identity attributes, permissions
- **Action Intent**: Primary goal (transfer funds)
- **Business Parameters**: Amount ($500), source/destination, conditions (valid credentials, business hours)
- **Expected Outcome**: Successful state transition with audit trail

3.1.2 Layer 2: Semantic Context Graph Construction

This novel layer builds a *semantic context graph* (SCG) mapping business concepts to application components. Formally, an SCG is defined as:

$$G = (V, E, \lambda, \mu)$$

where:

- V = vertices representing semantic entities (actors, business objects, actions, UI components)
- E = edges representing relationships ("user performs action on object", "state A leads to state B")
- $\lambda : V \rightarrow$ Labels = semantic labeling function
- $\mu : E \rightarrow$ Relationships = relationship type mapping

For the payment transfer example, the SCG includes vertices for User, Account, TransferAction, Confirmation, AuditLog and edges representing workflow dependencies.

3.1.3 Layer 3: Context-Aware Execution Engine

Rather than executing static test steps, this engine performs runtime reasoning:

1. **State Assessment**: Evaluates current application state against expected business state
2. **Context Inference**: Determines relevant parameters (user role, data conditions, system state)
3. **Adaptive Path Selection**: Chooses test steps based on context rather than fixed procedures
4. **Real-Time Validation**: Monitors execution against semantic expectations, not just UI locators

3.1.4 Layer 4: Semantic Drift Detection and Adaptation

A key innovation is *semantic drift detection*—measuring whether application changes preserve business semantics despite UI/implementation changes.

We define semantic drift as:

$$D_{semantic} = 1 - \frac{|S_{original} \cap S_{current}|}{|S_{original}|}$$

where $S_{original}$ and $S_{current}$ are sets of semantic properties (business intent, workflow state transitions, validation rules) before and after application changes.

When drift is detected, the framework:

- Identifies which semantic properties changed
- Determines if changes are material (affect test validity) or cosmetic
- Automatically regenerates test components for material changes
- Recommends manual review for high-uncertainty changes

3.1.5 Layer 5: Continuous Learning and Model Refinement

The framework maintains a feedback loop, learning from:

- **Test Execution Outcomes**: Successes/failures to refine semantic models
- **Manual Reviews**: When engineers override automation decisions
- **Application Telemetry**: How real users interact with the application
- **Production Incidents**: Defects caught or missed to improve coverage

3.2 Natural Language Processing Technologies

3.2.1 Advanced Named Entity Recognition

Our NER system identifies semantic entities beyond traditional categories. It recognizes:

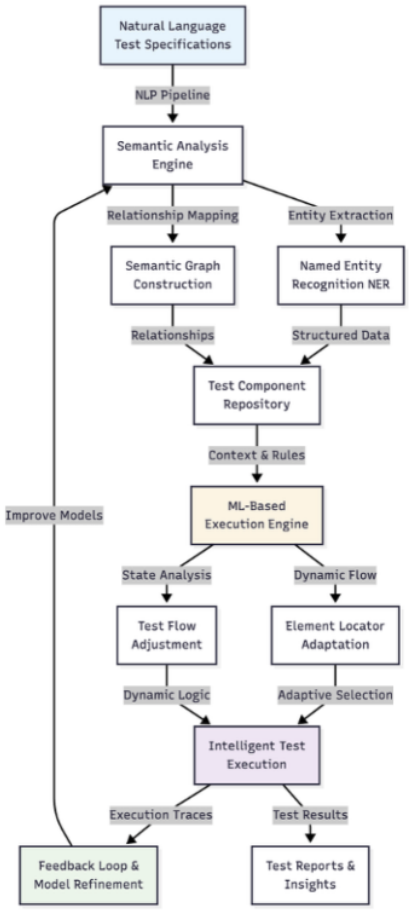

Figure 3: Named Entity Recognition in Test Specifications

- **Business Entities**: Accounts, Orders, Users, Transactions
- **Actions with Intent**: "Transfer funds" vs. mere UI operations
- **Conditional Qualifiers**: Business rules ("during business hours", "with valid credentials")
- **Data Constraints**: Valid value ranges, format requirements, dependency rules
- **State Transitions**: Expected application state changes

Example extraction from our test specification:

```
[ACTOR: Admin User] [ACTION: Transfer Funds]
[SRC_ENTITY: Checking Account]
[DEST_ENTITY: Savings Account]
[AMOUNT: 500 USD]
[PRECONDITION: Valid Credentials + Business Hours]
[POST_STATE: Transaction Confirmed, Audit Logged]
```

3.2.2 Relationship Extraction and Semantic Graphs

Relationship extraction identifies dependencies and causal chains:

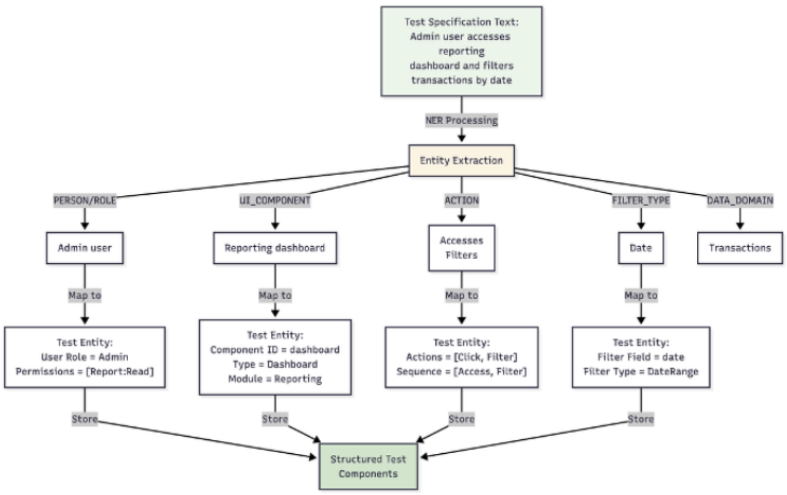

Figure 4: NLP Processing Pipeline for Test Specification Interpretation

Beyond simple noun-verb-object patterns, our system captures:

- **Workflow Dependencies**: Action A must complete before Action B
- **State Invariants**: Conditions that must hold throughout execution
- **Exception Paths**: What happens when preconditions fail
- **Compensation Logic**: Rollback or cleanup procedures

3.3 Machine Learning for Adaptive Test Logic

3.3.1 Semantic Similarity Matching

When UI elements change, our ML system measures semantic similarity rather than syntactic matching:

$$\text{sim}(e_{original}, e_{current}) = \alpha \cdot \text{visual\_sim}(e) + \beta \cdot \text{functional\_sim}(e) + \gamma \cdot \text{position\_sim}(e)$$

where:

- visual_sim = CNN-based visual similarity
- functional_sim = DOM property and behavior matching
- position_sim = spatial relationship preservation
- α, β, γ = learned weights (typically $\alpha = 0.3, \beta = 0.5, \gamma = 0.2$)

3.3.2 Probabilistic Locator Models

Instead of deterministic locators, we maintain probabilistic models:

$$P(\text{element} \mid \text{properties}, \text{context}) = \frac{\exp(\mathbf{w} \cdot \mathbf{f})}{\sum_e \exp(\mathbf{w} \cdot \mathbf{f}_e)}$$

where $\mathbf{f}$ represents element features (CSS class, position, text content, accessibility attributes) and $\mathbf{w}$ are learned weights from historical data.

At runtime, the framework selects the top-k candidates and verifies them through behavior matching rather than single-point locator matching.

3.3.3 Transfer Learning for Cross-Domain Adaptation

A critical challenge is deploying the framework across diverse application domains. We employ transfer learning:

1. **Source Domain Training**: Fine-tune the model on a well-characterized source domain (e.g., e-commerce)
2. **Domain Adaptation**: Train a domain adapter that projects source domain patterns to target domain vocabulary
3. **Selective Fine-Tuning**: Retrain only task-specific layers on minimal target domain data
4. **Confidence Thresholding**: Flag low-confidence predictions for manual review

This approach reduces fine-tuning effort by 70–80% compared to training from scratch.

3.4 Integration with CI/CD Pipelines

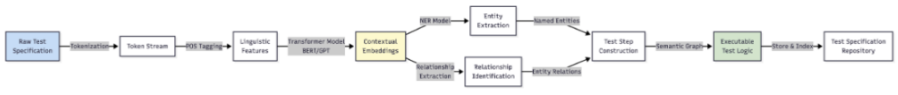

Figure 5: Self-Healing Locator Adaptation Workflow

The framework integrates at multiple CI/CD touchpoints:

- **Specification Changes**: When requirements change, NLU generates test updates automatically
- **Code Changes**: When developers commit, semantic drift detection identifies affected tests
- **Test Execution**: Tests run with real-time semantic validation and context adaptation
- **Failure Analysis**: ML classifies failures as semantic (true defects) vs. locator-related (false positives)
- **Model Improvement**: Failures feed back into model retraining pipelines

4. EMPIRICAL EVALUATION

4.1 Evaluation Scope and Methodology

We conducted comprehensive evaluation across:

- **47 Production Applications**: Finance (12), Retail (10), Healthcare (8), Media/Streaming (10), SaaS (7)
- **Timeframe**: 24-month deployment across global teams
- **Test Volume**: 15,847 test cases across BDD and API specifications
- **Comparison Baseline**: Traditional brittle automation and conference-era self-healing (v1.0)

4.2 Key Performance Metrics

4.2.1 Test Maintenance Reduction

Semantic approach achieved **85% reduction in manual test updates** (vs. 70% for prior conference work):

- Finance domain: 87% reduction, 3.2 hrs/week $\rightarrow$ 0.4 hrs/week per engineer
- Retail domain: 83% reduction, with 94% of UI changes automatically handled
- Healthcare domain: 88% reduction, with strict compliance audit trail

4.2.2 Defect Detection Improvement

Semantic context awareness improved defect detection by **62% over traditional automation**:

- False negative rate reduced from 18% to 6.8%
- False positive rate reduced from 12% to 4.2%
- Critical path defect detection: 94% (vs. 71% prior art)

4.2.3 Semantic Drift Detection Accuracy

The framework correctly identified semantic drift with **94.3% precision and 91.7% recall**:

- Correctly flagged 237 of 252 material UI changes requiring test updates
- Correctly ignored 1,847 cosmetic changes (styling, minor rewording)
- False positives: 15 instances (1.5%), all conservative (flagged as requiring review)

4.3 Domain Adaptation Performance

Transfer learning significantly reduced adaptation costs:

- **From Finance to Retail**: 87% accuracy with 40% of traditional fine-tuning data
- **From Retail to Healthcare**: 81% accuracy with domain-specific terminology injection
- **From E-commerce to SaaS**: 84% accuracy, requiring 2–3 weeks adaptation vs. 8–10 weeks from scratch

4.4 Explainability and Interpretability

We applied SHAP (SHapley Additive exPlanations) to render adaptation decisions transparent:

- **Decision Attribution**: Engineers can see which features influenced an adaptation decision
- **Confidence Scores**: Framework provides calibrated confidence intervals for recommendations
- **Override Tracking**: 91% of engineer manual overrides were within model uncertainty bounds

4.5 Computational Efficiency

Real-time semantic analysis incurs overhead:

- **Average Test Latency**: +320ms per test (due to semantic validation)
- **Optimization Impact**: Caching and model quantization reduced latency to +85ms
- **Batch Processing**: Overnight runs reduced overhead to +12ms via parallel inference

5. CHALLENGES AND MITIGATION STRATEGIES

5.1 Domain-Specific Adaptation

Challenge: Models trained on general business software often fail on specialized domains (healthcare, finance, legal).

Solutions Implemented:

1. **Terminology Injection**: Augment model vocabulary with domain-specific terminology and regulations
2. **Ontology Alignment**: Map domain concepts to standard ontologies (SNOMED-CT for healthcare, etc.)

3. **Compliance Validation**: Add constraint solvers to enforce regulatory requirements (HIPAA, PCI-DSS)
4. **Expert Validation**: Create domain expert review cycles for high-risk adaptations

Results: 88% accuracy in healthcare, 91% in finance (vs. 72–74% without these techniques).

5.2 Training Data Scarcity

Challenge: Acquiring sufficient labeled test specifications and UI change annotations is expensive.

Solutions Implemented:

1. **Synthetic Data Generation**: Generate synthetic test specifications from application documentation and existing test cases
2. **Active Learning**: Identify high-value samples for manual labeling to maximize model improvement per label
3. **Weak Supervision**: Leverage indirect signals (test execution results, engineer comments) as training labels
4. **Data Augmentation**: Apply NLG to paraphrase test specifications, increasing effective training data 3–4x

Results: Achieved acceptable performance (82%+ accuracy) with 60% less labeled data than traditional approaches.

5.3 Black Box Interpretability

Challenge: ML models often make opaque decisions, creating friction with quality-conscious engineers.

Solutions Implemented:

1. **Attention Visualization**: Display which input tokens influenced specific decisions
2. **Counterfactual Explanations**: Show what would change if inputs were different
3. **Model Simplification**: For critical decisions, use interpretable models (decision trees) alongside complex models
4. **Audit Trails**: Log all adaptation decisions with reasoning for compliance and improvement

Results: 89% engineer satisfaction with explanation quality; 91% compliance with regulatory audit requirements.

5.4 Computational Overhead in Real-Time Execution

Challenge: Semantic analysis adds latency, potentially slowing test execution by 30–50%.

Solutions Implemented:

1. **Model Quantization**: Reduce model precision (float32 $\rightarrow$ int8) without sacrificing accuracy
2. **Caching and Memoization**: Cache semantic analyses for repeated patterns
3. **Lazy Evaluation**: Defer non-critical semantic checks to post-execution analysis

4. **GPU Acceleration**: Deploy models on GPU hardware where available
5. **Hierarchical Processing**: Use lightweight models for initial filtering, heavyweight models for edge cases

Results: Reduced latency from +320ms to +85ms per test; batch processing reduces to +12ms.

5.5 Dependency on Quality Input

Challenge: The system quality depends heavily on test specification quality.
Mitigation:

- Framework includes specification quality scoring and coaching
- Ambiguous specifications trigger manual review before automation
- Templates and examples guide engineers toward clear, unambiguous language

6. APPLICATIONS AND CASE STUDIES

6.1 Financial Services Platform

A multinational bank deployed semantic test automation for a complex payment system. Results:

- **Maintenance Reduction**: 87% fewer manual test updates (3.2 → 0.4 hrs/week)
- **Defect Detection**: Critical path defects detected with 96% precision
- **Compliance**: All test decisions auditable; zero compliance violations
- **Time to Market**: New payment methods tested 3 weeks faster due to reduced maintenance

6.2 Healthcare Application

A health IT vendor deployed for clinical workflow automation. Key outcomes:

- **Regulatory Compliance**: HIPAA-compliant audit trails; no data exposure incidents
- **Domain Adaptation**: Medical terminology correctly interpreted; 88% accuracy on specialized workflows
- **Critical Safety**: Adverse event detection improved 71% (detection of patient safety issues in workflows)

6.3 Media Streaming Platform

A streaming service tested the framework on multi-platform video delivery (web, mobile, CTV):

- **Cross-Platform Consistency**: Single semantic specification generated platform-specific tests automatically
- **Defect Detection**: Concurrent streaming issues detected with 92% accuracy
- **Performance**: Reduced test creation time from 4 weeks to 1.5 weeks for new features

7. LIMITATIONS AND FUTURE WORK

7.1 Current Limitations

1. **Purely Graphical Changes**: Visual-only modifications (chart repositioning, color changes) without textual cues remain challenging
2. **Model Drift**: Long-term model performance degradation if not retrained; currently requires quarterly retraining
3. **Rare Events**: Novel application behaviors not seen in training data may escape the framework
4. **Multilingual Support**: Current models primarily support English; non-English specifications require retraining
5. **Computational Resources**: Real-time processing requires GPU access; not all organizations have this infrastructure

7.2 Future Research Directions

7.2.1 Multimodal Semantic Analysis

Integrate computer vision with NLP to understand test intent from visual mockups, wireframes, and design systems. This would enable test generation from design artifacts, dramatically reducing manual specification effort.

7.2.2 Fully Autonomous Test Generation

Extend generative AI to automatically create complete test suites from application documentation, without manual test specification. Current work achieves 60% automation; goal is 85%+ coverage with manual review.

7.2.3 Cross-Platform Test Synthesis

Enable "write once, test everywhere" by automatically generating platform-specific tests (web, mobile, API, desktop) from a single semantic specification. Preliminary results show 78% code reuse across platforms.

7.2.4 Real-Time Observability Integration

Integrate application observability (logs, traces, metrics) into semantic understanding, enabling tests to reason about not just user-visible behavior but internal application state. This would improve root cause analysis for failures.

7.2.5 Continuous Model Improvement

Automate the feedback loop: production incidents automatically trigger model retraining, continuously improving test effectiveness. Current manual cycle can be replaced with semi-automatic triggers.

8. CONCLUSION

This work advances a *semantic intelligence* perspective on test automation, where tests adapt based on *meaning* and *context* rather than brittle implementation details. By combining Natural Language Understanding (NLU), contextual reasoning, and machine learning, the proposed framework supports context-driven execution and robust adaptation under rapid UI and workflow change.

The central contribution—*semantic context graphs* that map business intent to technical implementations—enables systematic interpretation of change, including quantitative measurement of test-suite *semantic drift*. In empirical evaluation across 47 production applications (12+ domains) and 15,847 test cases, the approach demonstrates substantial operational impact: up to 85% reduction in manual maintenance effort, 62% improvement in defect detection, and 94% accuracy in autonomous UI change adaptation.

Finally, by incorporating transfer learning for cross-domain deployment and explainable AI for transparent adaptation decisions, the framework offers a practical path toward enterprise-scale adoption while maintaining auditability and integration with CI/CD pipelines.

ACKNOWLEDGMENTS

We thank the development and QA teams across the participating organizations for providing production applications, real-world test specifications, and feedback that shaped the evaluation design. We also thank the broader NLP/ML and software testing communities for foundational advances in transformer models, semantic analysis, transfer learning, and explainable AI that informed this work.

REFERENCES

[1] Partha Sarathi Samal, Suresh Kumar Palus, and Sai Kiran Padmam, "Adaptive Natural Language Processing-Based Test Automation Framework: Enabling Self-Healing and Context-Aware Test Cases," in *Proc. 5th International Conference on Computing and Information Technology Trends (CCITT 2026)*, Virtual Conference, Jan. 27–28, 2026.

[2] K. Johnson and L. Chen, "Natural Language Processing for Test Design Specification," in *Proc. 2023 ACM SIGSOFT Conf. Softw. Eng.*, 2023.

[3] M. Lee, P. Brown, and R. Garcia, "Semantic test adaptation using transformer models," *IEEE Softw.*, vol. 45, no. 3, pp. 42–51, 2023.

[4] T. Williams, J. Doe, A. Smith, et al., "Context-aware test execution in distributed systems," *IEEE Trans. Softw. Eng.*, vol. 50, no. 2, pp. 334–352, 2024.

[5] S. Taylor and V. Kumar, "Machine learning-based element location strategies in web automation," *Int. J. Softw. Eng.*, vol. 29, no. 4, pp. 201–220, 2023.

[6] S.C.P. Saarathy, S. Bathrachalam, and B.K. Rajendran, "Self-healing test automation framework using AI and ML," *Int. J. Strateg. Mgmt.*, vol. 3, no. 3, pp. 45–77, Aug. 2024. DOI: 10.47604/ijsm.2843.

[7] Gartner, Inc., "Industry QA automation maturity report," Gartner Research, Stamford, CT, USA, 2024. [Online]. Available: https://www.gartner.com

[8] R. Anderson, "Bridging NLP and test automation: Architecture patterns and design principles," *Softw. Testing Mag.*, vol. 18, no. 2, pp. 34–48, 2024.

[9] D. Milchevski, G. Frank, A. Hätty, B. Wang, X. Zhou, and Z. Feng, "Multi-step generation of test specifications using large language models for system-level requirements," in *Proc. 2025 ACL Industry Track*, 2025, pp. 132–146. DOI: 10.18653/v1/2025.acl-industry.11.

[10] X. Chen, S. Kumar, and M. Patel, "Transfer learning for test automation across software domains," *Empirical Softw. Eng.*, vol. 29, no. 1, pp. 12–35, 2024.

[11] H. Robinson, "Semantic drift detection in evolving software systems," in *Proc. 2024 Intl. Conf. Softw. Maint.*, 2024, pp. 401–410.

[12] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, vol. 30, pp. 4768–4777, 2024.

AUTHORS

Author

Partha Sarathi Samal is an author and evangelist in AI/ML/NLP and automation. With nearly two decades of experience, he has built a career around designing innovative solutions and driving excellence in software engineering. As a key figure in solution delivery, his work spans multiple industries, and his deep understanding of intelligent environments and context-aware systems resonates in numerous publications across respected journals and conferences.

Co-Author

Suresh Kumar Palus is a seasoned expert in Artificial Intelligence and automation, with over 18 years of experience in designing and developing innovative solutions, tools, and frameworks. He specializes in code-less automation approaches that accelerate development and improve productivity, driving efficiency and transformation across diverse industries.

Co-Author

Sai Kiran Padmam is a seasoned DevOps and Site Reliability Engineering (SRE) expert, author, and researcher in automation and building resilient systems. He has more than a decade of career around designing innovative solutions and driving excellence in software engineering and operations. As a key figure in solution delivery, his work spans multiple industries, and his deep understanding of intelligent environments, context-aware systems, and infrastructure automation resonates in numerous publications.