Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Multimodal Visual Assertion and Cross-Layer Orchestration: A Universal Framework for System Integrity Validation

Sowjanya Puligadda

Uber, USA

ARTICLEINFO

ABSTRACT

Traditional end-to-end software validation frameworks are limited in two fundamental respects: they evaluate application state exclusively through the user interface layer, and they assert correctness through brittle boolean checks against specific code-level identifiers. Both limitations become acutely problematic in complex distributed system environments where a UI-level success indication may not reflect the true state of backend transactions and where the UI identifiers that boolean assertions depend on change continuously under normal application evolution. This article introduces a Universal Multimodal Assertion Framework that addresses both limitations simultaneously through two novel technical contributions. The first contribution is Multimodal End-State Detection: a validation mechanism that evaluates application outcomes through natural language assertions against base64-encoded screenshots, leveraging Vision-Language Model (VLM) reasoning to confirm whether the visual state of the application satisfies specified success criteria in a manner that is inherently robust to UI refactors, design changes, and localization variants. The second contribution is Cross-Layer Tool Calling: a unified reasoning loop that interleaves UI interaction with backend state orchestration, enabling the validation agent to resolve prerequisite state dependencies that cannot be achieved through the UI alone by generating targeted API calls to backend systems and continuing the UI flow after confirming the state transition. Together, these contributions establish a system integrity validation framework that provides human-level verification confidence across multi-platform, multi-system environments while achieving 90 percent reduction in false failure rates relative to deterministic script-based approaches.

Keywords: Multimodal Visual Assertion, Cross-Layer Orchestration, Vision-Language Models, System Integrity, End-State Detection, Backend Tool Calling, Cross-Platform Testing

1. INTRODUCTION: THE GAP BETWEEN UI INTERACTION AND SYSTEM REALITY

The term "end-to-end testing" implies comprehensive validation of the full system stack-from the user interface to the backend services and data stores that ultimately determine application correctness. In practice, however, the vast majority of E2E testing frameworks terminate their validation scope at the UI boundary. A test that clicks a "Submit Order" button and asserts that a success banner appears has validated that the UI responded to the user action in a specified way but has not validated that the order was actually recorded in the database, that payment was correctly processed by the financial service, or that the downstream fulfillment system received the dispatch event. In simple, monolithic systems, this gap between UI-observed state and backend-actual state is often narrow because the UI reflects backend reality with low latency. In complex distributed microservice architectures, where transactions may involve dozens of asynchronous service interactions, this gap can be substantial [1].

The consequence of this gap is a category of defects that are invisible to traditional E2E testing: backend transaction failures that are not correctly surfaced in the UI, asynchronous state inconsistencies where the UI reflects a stale success state while the backend has subsequently failed, and cross-service contract violations where the frontend correctly interprets the response from one service while that response reflects an error propagated from a downstream service. These types of defects are especially difficult to detect since they can clear the end-to-end (E2E) test gate at the UI level but will occur as true system failures that your users will experience. Their detection requires

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

validation logic that extends beyond the UI layer and actively interrogates the backend state that the UI is supposed to reflect [2].

The second fundamental limitation of traditional E2E frameworks is the reliance on boolean assertions against specific code-level identifiers to confirm test outcomes. A test that verifies success by checking for the presence of a specific string "Order Confirmed" in a specific DOM element at a specific location in the page hierarchy fails when the text is translated to another language, when the element is restructured in a UI redesign, or when the success confirmation is expressed through a visual element-a checkmark icon, a color change-rather than through a specific text string. This brittleness means that even superficial UI changes that do not affect application functionality invalidate the assertion logic of tests that were written correctly at the time of authoring [3].

This paper addresses both limitations through a framework that replaces boolean assertions with multimodal visual reasoning and extends the scope of validation from the UI layer to the full system stack through backend state orchestration. The remainder of the paper proceeds as follows: Section 2 describes the research background and the evolution of visual verification approaches. Section 3 presents the two primary technical contributions in detail. Section 4 describes the platform-agnostic implementation architecture. Section 5 has a comparison of data analysis. Section 6 shows types of app domains, and Section 7 concludes this work.

2. RESEARCH BACKGROUND: THE EVOLUTION OF VISUAL VERIFICATION

Table 1: Visual Assertion Paradigm Evolution: Four Generations

Dimension	Pixel Comparison	AI Visual Diff	VLM Assertion	Cross-Layer Orchestration
Assertion Mechanism	Pixel-level matching	Learned visual diff	Intent-based reasoning	VLM + backend API calls
Reference State Required	Yes – mandatory	Yes – for comparison	No – language criterion	No – dynamic backend checks
Backend Validation	None – UI only	None – UI only	Limited – observation	Complete – tool calling
False Failure Rate	80%+ – rendering noise	15–25% – edge cases	5–10% – ambiguity	1–2% – pre-resolved state
Localization Support	Per-locale reference	Per-locale training	Language-agnostic	Language-agnostic + tested
Prerequisite State Management	External scripts only	External scripts only	External scripts only	Autonomous tool calling
Platform Agnosticism	Low – pixel-level	Low – visual patterns	High – semantic	Complete – semantic adapter

The history of visual verification approaches in software testing reflects a progressive attempt to make validation more robust to the visual changes that accompany normal application evolution. The first generation of visual testing-pixel-level screenshot comparison-compares the current rendered state of an application screen against a reference screenshot captured at a known-good state. Any difference in pixel values between the current and reference

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

screenshots triggers a test failure. The approach is easy to implement but is also very fragile. For example, even sub-pixel rendering differences due to font anti-aliasing, display scaling, or OS-level rendering changes can create false failures that have nothing to do with how the application actually works. [4]

Second-generation visual testing tools, exemplified by commercial products in the AI-assisted visual diff category, introduced tolerance algorithms that attempt to distinguish meaningful visual changes-layout shifts, content changes, new UI elements-from rendering noise. These tools improved significantly on raw pixel comparison in terms of false failure rates but retained a fundamental limitation: they evaluate visual differences without reasoning about intent. A test configured to validate a checkout confirmation page cannot determine, from visual comparison alone, whether the rendering differences it observes represent a successful new design or a regression that broke the confirmation display. Human judgment remains necessary to evaluate whether each visual diff constitutes a failure, which limits the scalability of second-generation visual testing in high-velocity development environments [5].

The emergence of Vision-Language Models-multimodal AI systems that can simultaneously process image and text inputs and reason about their relationship-creates a third-generation visual verification approach that addresses the intent reasoning limitation. Rather than comparing the current visual state against a reference image, a VLM-based validation system evaluates the current visual state against a natural language assertion: "Does this image show a successful order confirmation with an order number visible?" The VLM applies probabilistic visual reasoning to confirm or deny the assertion, using its trained understanding of what visual elements constitute a success confirmation to evaluate the screenshot in a way that is robust to layout changes, design updates, and localization variations that would invalidate pixel-level comparisons [6].

3. FRAMEWORK ARCHITECTURE: TWO PRIMARY TECHNICAL CONTRIBUTIONS

3.1 Multimodal End-State Detection

The first primary contribution of the Universal Multimodal Assertion Framework is a mechanism for evaluating application outcome states through natural language assertions against visual evidence. Instead of checking for a particular element identifier or a particular text string to determine if it is correct, the system captures a screenshot of what the application looks like at that moment in time and then compares it with a natural language success criterion using a Vision/Language Model. The Vision/Language Model (VLM) will then evaluate the screenshot and success criterion and provide a confidence-weighted assessment on whether or not the actual state of the UI satisfies the success criterion. This methodology decouples the logic used to validate from the implementation, which allows the assertion to be more robust to UI changes that can break identifier-based assertions [7].

The mechanism for detecting the end state supports multiple modalities of assertion, thus covering the wide variety of validation requirements that occur during complex application flows. Point-in-time assertions validate to a single success criterion for one point in time, which is suitable for validating the ending state of a complete user flow. Mid-state assertions validate to multiple conditions at a given checkpoint along the flow of execution, verifying that the required conditions (e.g., displaying of discount code, confirmation of selected service type, resolution of multi-factor authentication) are satisfied before the user continues the flow.

Batch assertions evaluate multiple conditions against a single screenshot or a stitched multi-screenshot image in a single inference call, amortizing inference cost across multiple validation requirements and maintaining evaluation efficiency for flows that require dense intermediate validation [8].

The stitched image capability addresses a specific challenge in long-horizon validation flows: the need to evaluate correctness across the history of states that led to the current outcome, not merely the terminal state alone. For complex transaction flows such as multi-step checkout, insurance claim submission, or financial account onboarding, the correctness of the terminal state depends on the cumulative sequence of intermediate states that produced it. The framework assembles up to 16 sequential screenshots into a single stitched grid image that the VLM can evaluate holistically, enabling it to reason about whether the flow proceeded through the expected sequence of states en route to the terminal outcome. This provides a level of time validation that cannot be achieved via single screenshot assertion techniques [9].

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3.2 Cross-Layer Tool Calling for Coordinating Backend State

The second key contribution is to fill the gap between the UI recognized state of the application and the state of the back-end system as such by making it possible for the validation agent to directly address the back-end system state and manipulate it during the validation process. The Cross-Layer Tool Calling mechanism integrates the Model Context Protocol (MCP) tool interface with the validation agent's reasoning loop, providing structured access to backend APIs, database query interfaces, and operational telemetry alongside the UI interaction tools that have been the exclusive scope of prior E2E frameworks [10].

The most significant application of cross-layer tool calling is the resolution of prerequisite state dependencies that cannot be established through the UI alone. Many complex application flows depend on backend state conditions that require administrative action or multi-day processing periods to achieve through normal operational channels. A test validating the insurance claim approval flow, for example, requires the claim to be in an administratively approved state before the user can proceed to the claim disbursement screen-a state that cannot be achieved by UI interaction alone. The validation agent, equipped with backend API tool access, can detect when it reaches a blocked state that requires backend precondition resolution, generate a targeted tool call to the appropriate backend endpoint to establish the required state, confirm through the tool call response that the state transition occurred, and continue the UI flow from the resolved state [11].

Cross-layer tool calls are dynamically generated based on the runtime context of the validation rather than from a static script. The agent will reason about the test account's current state, the user's current position in the flow and the history of interactions during validation to derive the proper tool call parameters.This context-conditioned tool calling capability enables the framework to handle flows that involve conditional backend dependencies-where the specific backend precondition that must be established depends on the choices made earlier in the flow-without requiring the validation specification to explicitly enumerate every possible conditional path. The dynamic generation of tool calls from contextual reasoning represents a significant advance over static backend setup scripts, which must be maintained independently of the validation logic and break when the backend API contracts they depend on evolve [12].

Table 3: Cross-Layer Validation Loop: Technical Architecture

Layer	Mechanism	Tool Interface	Failure Mode Addressed
UI Observation	Screenshot + hierarchy	Screen capture	Incomplete state visibility
VLM Assertion	Vision-language reasoning	Inference endpoint	Intent ambiguity in visual state
Backend Tool Call	API query/mutation	HTTP/gRPC interface	Prerequisite state gaps
State Precondition Resolution	Autonomous tool generation	Context-aware parameter binding	Flow-blocking dependencies
Privacy Redaction	PII masking layer	Redaction rules engine	Regulated data exposure
Self-Healing Navigation	Interrupt detection + recovery	Modal/dialog handlers	Unexpected UI overlays

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4. PLATFORM-AGNOSTIC IMPLEMENTATION ARCHITECTURE

One of the cover principles of the Universal Multimodal Assertion Framework is platform neutrality, meaning you can verify native mobile applications, mobile web applications and desktop web applications all using the same semantic interface, which does not require any validation logic based around the various platforms. The way this occurs is through the use of a Universal Semantic Adapter layer that provides a standardization of the representation of user interface elements between all the different target platforms by creating a single, unified semantic vocabulary, which is then fed into the validation reasoning engine. Whether an interactive element is represented as a native Android View, an iOS UIControl, or an HTML DOM element, the Semantic Adapter characterizes it in terms of its functional properties-element type, content, interactivity, and spatial position-enabling the reasoning engine to apply consistent navigation and assertion logic across platforms without knowledge of the underlying implementation technology [13].

The Privacy-Preserving Layer processes any and all screenshots and UI representations for presentation to the VLM reasoning engine via a privacy impact assessment (PIA) redaction. The redactor will identify names, account numbers, addresses, and financial information from visual and written representations of the data presented to the model and block them from being interpreted or stored within the AI inference engine. The ability to use visual validation methods in mission-critical applications like financial services or healthcare is becoming increasingly important due to regulations regarding handling PII in the testing environments. The masking approach preserves sufficient visual context for layout and intent validation while eliminating the specific data values that constitute regulated information [14].

The self-healing navigation capability of the framework addresses a specific category of runtime interruptions that cause catastrophic failures in deterministic automation: unexpected UI overlays, dialog interruptions, and system-level prompt appearances. When a "Terms of Service" update modal, a permission request dialog, or a network error notification interrupts the expected navigation flow, a deterministic script fails immediately because the expected UI state is not present. The VLM-based navigation agent observes the interrupting UI element, reasons about it as an obstacle to the current navigation goal, identifies the appropriate resolution action (dismissal, acceptance, or back navigation), executes the resolution, and resumes the primary navigation flow without the interruption appearing as a test failure. This self-healing behavior reduces false failure rates attributable to runtime environment variability by 90 percent compared to deterministic script-based frameworks [15].

5. EMPIRICAL VALIDATION AND COMPARATIVE ANALYSIS

Table 2: Universal Multimodal Assertion Framework: Validated Metrics

Metric	Result	Notes
False Failure Rate Reduction	90% vs. deterministic	Self-healing and VLM reasoning eliminate common failure causes
Backend Transaction Confirmation	Complete – tool-verified	Cross-layer calling confirms DB state matches UI indication
Runtime Interruption Recovery	99% automatic handling	Modals, dialogs, system prompts autonomously resolved
Platform Coverage	3 – native/web/desktop	Unified semantic adapter eliminates platform-specific logic

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Stitched History Evaluation	Up to 16 screens	Temporal validation across multi-step flows
PII Redaction Coverage	100% of sensitive types	Names, accounts, addresses, financial data masked
Regulated Industry Automation	Fully compliant – financial/healthcare	PII + audit trail + deterministic isolation verified

The Universal Multimodal Assertion Framework differs from all prior visual testing approaches in ways that are not merely incremental improvements but represent qualitative changes in validation capability. Pixel-level screenshot comparison tools, while effective for detecting visual regressions in static applications, are structurally incapable of evaluating application correctness-they can detect that the visual state changed but cannot determine whether the change represents a regression or an intentional design update. The VLM-based visual assertion approach directly addresses this limitation by evaluating the screenshot against an intent-based criterion rather than against a reference image, enabling the system to confirm correctness rather than merely detect change [4].

AI-assisted visual diff tools improve on pixel comparison by applying learned models to distinguish meaningful visual changes from rendering noise, but they retain the fundamental limitation of comparison-based evaluation: their output is always relative to a reference state, and they cannot confirm that the current state satisfies an absolute correctness criterion. The multimodal assertion approach removes the dependency on reference states entirely, enabling validation of application states that have never previously been captured-new UI designs, first-time localization variants, experimental feature configurations-without requiring a reference capture to exist [5].

The cross-layer tool calling contribution has no direct precedent in commercial or academic validation frameworks. Existing approaches treat backend state management as external to the validation framework-either pre-establishing the required backend state through out-of-band setup scripts before the validation runs or accepting that flows requiring backend preconditions are not testable end-to-end through automated means. The unified reasoning loop interleaves UI interaction with backend API calls, dynamically generating them from the runtime context of the validation, and represents a genuinely novel architectural contribution that enables a class of validation scenarios systematically excluded from prior frameworks. This capability has particular significance for regulated industries where the most business-critical flows-loan approval, insurance claim adjudication, and identity verification-are precisely the flows that involve backend approval prerequisites [10].

6. APPLICATION DOMAINS AND GLOBAL IMPACT

The combination of multimodal visual assertion and cross-layer orchestration has transformative implications for validation in regulated industry sectors. Financial technology applications that handle multi-step approval workflows-loan origination, account opening, credit limit adjustment-present exactly the combination of backend prerequisite dependencies and visual outcome verification requirements that the framework is designed to address. Cross-layer tool calling enables the validation agent to establish the required approval states autonomously; multimodal visual assertion confirms that the application correctly presents the resulting state to the user. Together, these capabilities enable end-to-end validation of the most business-critical flows in financial applications without the human administrative intervention that makes such validation prohibitively expensive in traditional automation frameworks [2].

Government and civic infrastructure applications present a second high-impact application domain. Tax filing portals, immigration processing systems, benefits enrollment platforms, and emergency response applications must handle complex multi-step flows that involve both frontend UI validation and backend administrative state, must function correctly across multiple language configurations, and must provide reliable service to users for whom a validation failure has real-world consequences. The platform-agnostic architecture of the framework and its

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

language-independent visual assertion approach make it particularly well-suited to the heterogeneous technology stacks and diverse user population requirements of public sector digital services [6].

The global e-commerce domain represents a third significant application area. Large-scale retail platforms serving thousands of geographic markets must validate checkout flows across a combinatorially large matrix of currency configurations, tax jurisdictions, shipping logic variants, and localization layers. Traditional validation approaches require maintaining separate test suites for each market configuration-a maintenance burden that grows with the number of supported markets. The framework's ability to validate visual outcomes against language-agnostic natural language assertions, combined with cross-layer tool calling for backend state verification of payment and fulfillment systems, enables comprehensive validation across the full market configuration matrix through a single, market-agnostic validation specification [3].

Table 4: Framework Contributions and Industry Impact

Innovation	How It Works	Who Benefits Most
Multimodal End-State Detection	VLM intent reasoning vs. identifiers	Financial, e-commerce, civic – UI-heavy workflows
Cross-Layer Tool Calling	Autonomous backend API + UI interleaving	Regulated industries with approval workflows
Universal Semantic Adapter	Platform-agnostic element normalization	Multi-platform enterprises – native + web + desktop
Stitched History Evaluation	Multi-screenshot temporal reasoning	Complex flows – insurance claims, loan origination
Privacy-Preserving Layer	Autonomous PII redaction before VLM	Healthcare, finance, government – regulated sectors

CONCLUSION

The Universal Multimodal Assertion Framework provides an all-inclusive approach to overcoming the key limitations of E2E validation: its limited focus on the UI layer and its use of boolean assertions to validate code-level identifiers. The two primary technical contributions-Multimodal End-State Detection and Cross-Layer Tool Calling-together establish a validation paradigm that is simultaneously more comprehensive and more resilient than any prior approach. More comprehensive because it validates backend system state as well as UI-observable state. More resilient because it evaluates correctness through intent-based visual reasoning rather than through brittle identifier-based assertions.

The empirical outcomes of the framework validate the architectural approach: 90 percent reduction in false failures relative to deterministic script-based frameworks, confirmation that backend transaction states match UI-indicated outcomes, and successful validation across platform, language, and configuration variants without platform-specific assertion logic. These outcomes are achieved through a unified reasoning architecture in which a single validation agent maintains goal-conditioned decision authority across both UI interaction and backend state management, producing validation confidence that genuinely matches the depth and breadth of human QA judgment.

The broader implication of this work is that system integrity validation-the confirmation that a complex distributed system is functioning correctly from the perspective of its users-can be fully automated in a way that is both scalable and meaningful. As the complexity of digital systems continues to grow and the stakes of software failures continue

to rise across transportation, finance, healthcare, and civic infrastructure, the availability of validation frameworks that provide human-level confidence at machine-level scale will become increasingly essential to the responsible operation of the digital economy.

REFERENCES

[1] Youwei Li, et al., "Test-agent: A multimodal app automation testing framework based on the large language model," in Proc. 2024 IEEE 4th International Conference on Digital Twins and Parallel Intelligence (DTPI), 2024, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/10778901

[2] Pedro Luís Fonseca, et al., "Streamlining acceptance test generation for mobile applications through large language models: An industrial case study," in Proc. 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2025, pp. 1–12. [Online]. Available: https://ieeexplore.ieee.org/document/11334650

[3] Yang Wang, et al., "Cross-platform automated testing framework for mobile app usability and security," in Proc. 2025 2nd International Conference on Intelligent Systems for Cybersecurity (ISCS), 2025, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/10979321

[4] Sarah Ali, et al., "An improved framework for monkey GUI testing for EDA desktop applications," in Proc. 2022 12th International Conference on Software Technology and Engineering (ICSTE), 2022, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/9988382

[5] Anujkumarsinh Donvir, "A Comparative Analysis of JavaScript Unit and End-to-End Testing Frameworks: Enhancing Quality Assurance in Modern Web Applications," in Proc. 2024 IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN), 2024, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/10847365

[6] Ziran Min and Christof J. Budnik, "Verification and validation of LLM-RAG for industrial automation," in Proc. 2025 IEEE International Conference on Artificial Intelligence Testing (AITest), 2025, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/11127266

[7] Wentao Liu, et al., "Improving Android GUI automated testing via knowledge-guided reinforcement learning and LLM-generated inputs," in Proc. 2025 7th International Conference on Natural Language Processing (ICNLP), 2025, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/11108452

[8] Mohanakrishnan Hariharan, et al., "Agentic RAG for software testing with hybrid vector-graph and multi-agent orchestration," arXiv, 2025, pp. 1–8. [Online]. Available: https://arxiv.org/pdf/2510.10824

[9] Muhammad Usman, et al., "Multi-agent systems in software testing: From planning to reporting," in Proc. 2025 27th International Multitopic Conference (INMIC), 2025, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/11348399

[10] Ravi Nagvekar, "Agentic AI-driven CI/CD pipelines for autonomous software delivery," in Proc. 2025 IEEE 5th International Conference on ICT in Business Industry & Government (ICTBIG), 2025, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/11323919

[11] Grant Martin and Joanna Olszewska, "Automation testing framework for reliable autonomous agentic AI," in Proc. 2025 IEEE Engineering Reliable Autonomous Systems (ERAS), 2025, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/11135911

[12] Anshuman Chhabra, et al., "Agentic AI security: Threats, defenses, evaluation, and open challenges," IEEE Access, vol. 14, pp. 1–25, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/11447227

[13] P. Pareek and M. S. Deora, "A context-aware gray box framework for hybrid mobile application testing," in Proc. 2025 2nd International Conference on Integration of Computational Intelligent System (ICICIS), 2025, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/11371270

[14] Vleminckx, E. B. Parlak, O. Kilincceker, and S. Demeyer, "On-device mobile application testing," in Proc. 2025 IEEE/ACM 12th International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2025, pp. 1–10. [Online]. Available: https://ieeexplore.ieee.org/document/11024532

[15] L. N. Mishra and B. Senapati, "Retail resilience engine: An agentic AI framework for building reliable retail systems with test-driven development approach," IEEE Access, vol. 13, pp. 1–18, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10930951