International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(21s) 2026 pp. 391-405
https://doi.org/10.70917/ijcisim-2026-5315
Article

# Claude-Driven Intelligent Test Automation: A Unified API and UI Framework for Enterprise and Healthcare Compliance

**Pratik Dinkar Rane**

Independent Researcher, USA
ORCID: 0009-0009-5505-3197

**Abstract:** Enterprise and healthcare software systems demand rigorous, compliance-aware testing across both API and user interface layers. Conventional test automation toolchains treat API and UI testing as separate disciplines, forcing organizations to maintain parallel frameworks, duplicate test logic, and reconcile compliance evidence from disparate sources. This fragmentation imposes compounding overhead in regulated environments governed by HIPAA, HL7 FHIR R4, and FDA 21 CFR Part 11. The cost of maintaining disjoint toolchains is particularly acute in healthcare information technology, where test failures may carry patient safety implications and every audit trail gap constitutes a regulatory risk. This paper introduces ClinQA, a large language model orchestrated framework that unifies API and UI test generation through a single compliance-aware pipeline. ClinQA uses a structured prompt schema with five components, which it sends to Claude, an instruction-following language model. It then routes the generated test artifacts to platform-specific adapter modules for Playwright-based UI execution and consumer-driven contract testing for API validation. A four-layer architecture separates intent capture, test generation, execution, and compliance reporting into independently maintainable components. The framework embeds PHI masking, FHIR conformance evidence capture, and 21 CFR Part 11 audit trail emission at the infrastructure level rather than delegating compliance responsibility to individual test authors. Preliminary observations from 12 scenarios spanning Salesforce CRM workflows and FHIR R4 conformance testing indicate a Cross-Layer Coverage Index of 87%, a Test Authoring Effort Reduction of 68% compared to manual authoring, and a Compliance Coverage Score of 94% against active compliance profiles. ClinQA demonstrates that unified LLM-orchestrated test generation is a feasible approach for regulated enterprise and healthcare environments, and it establishes a foundation for controlled empirical validation at production scale.

**Keywords:** Test Automation, Large Language Models, Claude, API Testing, UI Testing, FHIR, HIPAA, 21 CFR Part 11, Enterprise Testing, Healthcare IT, BDD, DevOps

## 1. Introduction

### *1.1 Motivation and Problem Context*

Enterprise software platforms such as SAP ERP and Salesforce CRM present unique test automation challenges. Both platforms expose rich user interface surfaces and REST API endpoints, and both are frequently deployed in tightly regulated industries. Quality assurance teams operating on these platforms must verify functional correctness at the API layer, validate visual workflows at the UI layer, and produce compliance evidence appropriate to the regulatory context. When organizations use separate tools for each layer, test intent is duplicated, maintenance costs double, and compliance evidence must be reconciled manually across tool outputs.

Healthcare information technology systems introduce additional complexity. Systems implementing HL7 FHIR R4 [5] must pass resource conformance testing against defined US Core Implementation Guide profiles. Systems handling protected health information must comply with HIPAA Privacy and Security Rules [6], preventing real

patient data from appearing in test artifacts. Systems falling under FDA 21 CFR Part 11 [7] must generate traceable audit trails for every test execution that touches electronic records. Meeting these obligations simultaneously across a disjoint toolchain requires manual instrumentation that is error-prone and difficult to scale across sprint cycles.

The proliferation of LLMs in software engineering tasks offers an opportunity to address this fragmentation. LLMs capable of processing natural language test intent and emitting structured, executable test artifacts can serve as a single generation engine for both API and UI test scenarios. ClinQA selects Claude as its generation engine based on the instruction following fidelity, structured output reliability, and safety oriented design principles documented in the Anthropic model card and evaluation framework [9]. These characteristics are essential for regulated test generation, where the model must consistently produce schema conformant output while respecting data handling constraints such as PHI exclusion. A compliance aware prompt schema can embed regulatory context directly into the generation request, ensuring that generated artifacts carry appropriate compliance annotations by default. This paper presents ClinQA as a concrete realization of this architecture, targeting the intersection of enterprise platform testing and healthcare compliance automation.

### *1.2 Research Gap and Proposed Contribution*

Existing frameworks address portions of the unified testing problem. EvoSuite [3] generates test suites for Java code but has no mechanism for UI generation or compliance annotation. Playwright and Selenium address UI automation but require separate orchestration for API coverage. Contract testing tools such as Pact [8] validate API contracts effectively but do not generate tests from natural language intent and carry no compliance context. Record-and-replay tools [1, 2] reduce initial authoring effort but produce brittle scripts that degrade rapidly with application changes.

LLM-based test generation is an active research area. Studies report that models such as CodeBERT [11] and instruction-tuned LLMs produce unit tests with measurable accuracy [13, 16, 17]. LLM-assisted mobile GUI testing has been demonstrated at scale [14], and exploratory studies confirm that LLMs can support functional test scenario generation for complex systems [15]. However, no existing framework combines unified API and UI generation, compliance-aware prompting, and structured compliance reporting in a single pipeline. Compliance-aware generation in regulated environments, where prompt content must not include PHI and generated artifacts must carry audit evidence, remains an open problem.

ClinQA addresses this gap through five contributions. First, it defines a four-layer architecture separating intent, generation, execution, and compliance. Second, it introduces a five-component compliance-aware prompt schema for Claude. Third, it implements a PHI masking boundary that prevents protected health information from entering the Claude API surface. Fourth, it provides platform-specific adapter modules for SAP Fiori, Salesforce REST, and FHIR R4 endpoints. Fifth, it integrates Gherkin-based BDD specification parsing as a natural language interface for test intent capture across CI/CD pipelines.

### *1.3 Paper Organization*

Section 2 reviews related work across test automation, healthcare IT standards, LLM applications, and DevOps practices. Section 3 presents the ClinQA architecture in full detail, covering design principles, the four-layer model, the Claude integration model, and the compliance module. Sections 4 and 5 demonstrate application to enterprise and healthcare testing scenarios, respectively. Section 6 describes CI/CD and BDD pipeline integration. Section 7 presents a discussion including a preliminary evaluation. Section 8 concludes with future research directions.

## 2. Background and Related Work

### *2.1 Test Automation in Enterprise Systems*

UI test automation for enterprises has long contended with locator brittleness. DOM identifier-based selectors break when application vendors modify page structure, which occurs with every major upgrade in platform-as-a-

service products such as Salesforce Lightning and SAP Fiori UI5. Waterfall-based self-healing approaches [2] reduce locator maintenance cost incrementally but require substantial repair logic embedded in test frameworks. Record-and-replay tools [1] lower the initial authoring barrier but produce scripts tightly coupled to recorded DOM states, making them unsuitable for environments with frequent UI version changes.

API contract testing emerged as a complementary discipline to address inter-service communication verification. Consumer-driven contract testing through tools such as Pact [8] defines contracts from the consumer perspective and verifies provider conformance independently of service deployment. This approach reduces integration failures in microservice architectures and enables earlier detection of contract drift. However, contract testing tools operate at the API layer only and do not share test intent, execution context, or evidence artifacts with co-existing UI test frameworks.

Cross-layer coverage, meaning the simultaneous validation of API behavior, UI behavior, and compliance constraints for a single business scenario, remains an unresolved gap in enterprise test automation. Existing frameworks force QA engineers to author separate test artifacts for each layer, increasing total authoring time and creating coverage gaps when layer updates are not synchronized. The cost of these gaps is highest in healthcare settings, where API data integrity and UI rendering must be jointly verified to demonstrate clinical safety.

## *2.2 Healthcare IT Standards and Compliance Testing*

HL7 FHIR R4 [5] defines a REST-based interoperability standard for healthcare data exchange. Conformance testing against FHIR means checking that resource representations meet profile constraints in Implementation Guides like US Core, that search parameter semantics give correct results, and that SMART-on-FHIR authorization flows enforce appropriate scopes. The AEGIS Touchstone platform [10] provides a hosted FHIR conformance testing service, but it operates as an external validation tool rather than an embedded test generation framework integrated into CI/CD pipelines.

HIPAA Privacy and Security Rules [6] require that protected health information be handled with appropriate safeguards in all system contexts, including testing. This means that you must use synthetic or de-identified records for test data, and you must not expose PHI in logs, reports, or CI/CD artifacts in your test scripts. Compliance with this requirement in automated testing pipelines is typically achieved through manual test data governance rather than through framework-enforced controls, creating audit exposure in organizations that rely on engineer discipline alone.

FDA 21 CFR Part 11 [7] governs electronic records in FDA-regulated software development contexts and requires traceable audit trails for system changes, including test execution records. In practice, this means that test execution reports for FDA-regulated applications must be retained with metadata linking specific test results to the software version under test. Most test automation frameworks produce execution reports but do not generate audit trail artifacts in a format directly compatible with 21 CFR Part 11 evidence requirements. ClinQA addresses these issues by generating structured audit trail records at the compliance layer rather than relying on post-hoc report processing.

## *2.3 Large Language Models in Software Engineering*

The application of large language models to software engineering tasks has expanded rapidly. CodeBERT [11] demonstrated that pre-trained models over code and documentation outperform task-specific models in code search and summarization. Subsequent research extended LLM applications to test generation, with empirical studies reporting that LLM-generated unit tests achieve competitive coverage compared to EvoSuite-generated suites when measured on assertion accuracy [13]. Wang et al. [16] provide a comprehensive survey of LLM applications in software testing, documenting progress across unit testing, system testing, and metamorphic testing.

LLM-based test generation for GUI applications has progressed significantly. Liu et al. [14] demonstrated that LLM-driven mobile GUI testing achieves broader functional coverage than random or model-based baselines by using functionality-aware decisions. Alshahwan et al. [17] reported production-scale deployment of LLM-generated unit

test improvements at Meta, with engineers accepting a substantial fraction of LLM-suggested changes. Augusto et al. [15] conducted an exploratory study of LLM assistance for system test authoring, finding that practitioners valued LLM output for scenario coverage expansion even when generated tests required manual refinement.

A persistent limitation of current LLM test generation approaches is that they do not include compliance context in generation requests. Existing studies evaluate generated tests on coverage and correctness metrics but do not consider compliance annotation requirements that govern regulated industries. PHI exposure in prompts is a specific risk in healthcare contexts where test intent descriptions may include patient scenario details. ClinQA addresses this limitation through a structured prompt schema that enforces PHI scrubbing before test intent content reaches the Claude API surface.

### *2.4 DevOps, Shift-Left Testing, and BDD*

Shift-left testing advocates integrating quality assurance activities as early as possible in the software development lifecycle. By performing test generation and execution during development rather than after integration, teams detect defects when they are least expensive to fix. Devops and continuous integration practices provide the infrastructure for shift-left testing through automated pipeline triggers, but they do not address the authoring bottleneck: engineers must still write tests before the pipeline can execute them. LLM-driven test generation offers a mechanism to eliminate this bottleneck by generating tests from specification artifacts that exist earlier in the development process than implementation code.

Behavior-Driven Development [19] provides a natural language specification format through the Gherkin Given-When-Then syntax. Gherkin scenarios describe business behavior from a stakeholder perspective and are typically authored before implementation begins. Parsing Gherkin scenarios and translating them into executable API and UI test code is a natural application for LLMs, and ClinQA implements this through its BDD pipeline module. This creates a direct path from business specification to executable test artifact without manual test authoring as an intermediate step.

ClinQA's key design principle is to integrate test generation into CI/CD pipelines as a pre-execution step rather than a post-implementation step. The DevSecOps paradigm, first synthesized as a distinct discipline through early multivocal literature reviews by Myrbakken and Colomo-Palacios [20], established the foundational argument that security and quality activities must be embedded continuously throughout the development lifecycle rather than deferred to late stage gates. Prates and Pereira [19] extended this foundation through a comprehensive survey of contemporary DevSecOps practices and tools, finding that the most effective security and quality postures integrate verification activities into every pipeline stage, with organizations that distribute testing across commit, build, and deployment phases detecting defects significantly earlier than those concentrating verification at release boundaries. ClinQA operationalizes both the continuous integration philosophy articulated in early DevSecOps research [20] and the pipeline stage distribution patterns identified in recent practice surveys [19] by enabling test generation triggers at pull request creation time. This ensures that new code has associated generated test coverage before review completion, embedding compliance aware quality verification at the earliest actionable point in the development workflow rather than treating it as a post-implementation validation activity.

## 3. ClinQA Framework Architecture

### *3.1 Design Principles*

ClinQA is organized around five architectural principles. Compliance-by-default requires that every generated test artifact carries compliance annotations appropriate to the active profile without requiring test authors to add them manually. Tool agnosticism requires that ClinQA generate output compatible with standard execution frameworks rather than proprietary runtimes, enabling organizations to adopt ClinQA without replacing existing CI/CD infrastructure. Auditability requires that every generation event produces a traceable record associating input intent, prompt content, generated artifact, and execution result in a single evidence chain.

Human-in-the-loop oversight requires that Claude is invoked for generation only, never for execution decisions. The framework maintains a strict boundary between the AI generation path and the deterministic execution path. This separation ensures that execution behavior is fully determined by generated artifacts reviewed by human engineers rather than by real-time model inference. Generation-path containment, the fifth principle, operationalizes this separation by ensuring that the Claude API surface is never invoked during test execution, only during scheduled generation runs triggered by pipeline events or manual operator requests.

These principles distinguish ClinQA from approaches that use LLMs for test oracle evaluation or assertion generation during execution. The risks associated with non-deterministic LLM inference during execution are eliminated entirely. The framework accepts the cost of separating generation and execution as a deliberate trade-off to satisfy compliance requirements in regulated environments where determinism, traceability, and human oversight are non-negotiable.

### *3.2 Architectural Overview*

ClinQA implements four loosely coupled layers. Layer 1, the Natural Language Intent Layer, accepts test intent from three input formats: Gherkin feature files, structured JSON test briefs, and free-text natural language descriptions. Intent preprocessing in this layer applies the PHI masking filter before any content is passed to Layer 2. The masking filter uses pattern-based regular expressions for common PHI categories and a named entity recognition model for clinical term detection, producing a scrubbed intent document that contains no protected health information.

Layer 2, the Test Generation and Orchestration Layer, receives the scrubbed intent document together with the active compliance profile and platform context descriptor. This layer constructs the five-component Claude prompt schema and submits it to the Claude API. The structured JSON response is validated against the output schema before being passed to Layer 3. Generation events are logged to a generation audit record containing the prompt hash, model version, timestamp, and the result of the response schema validation.

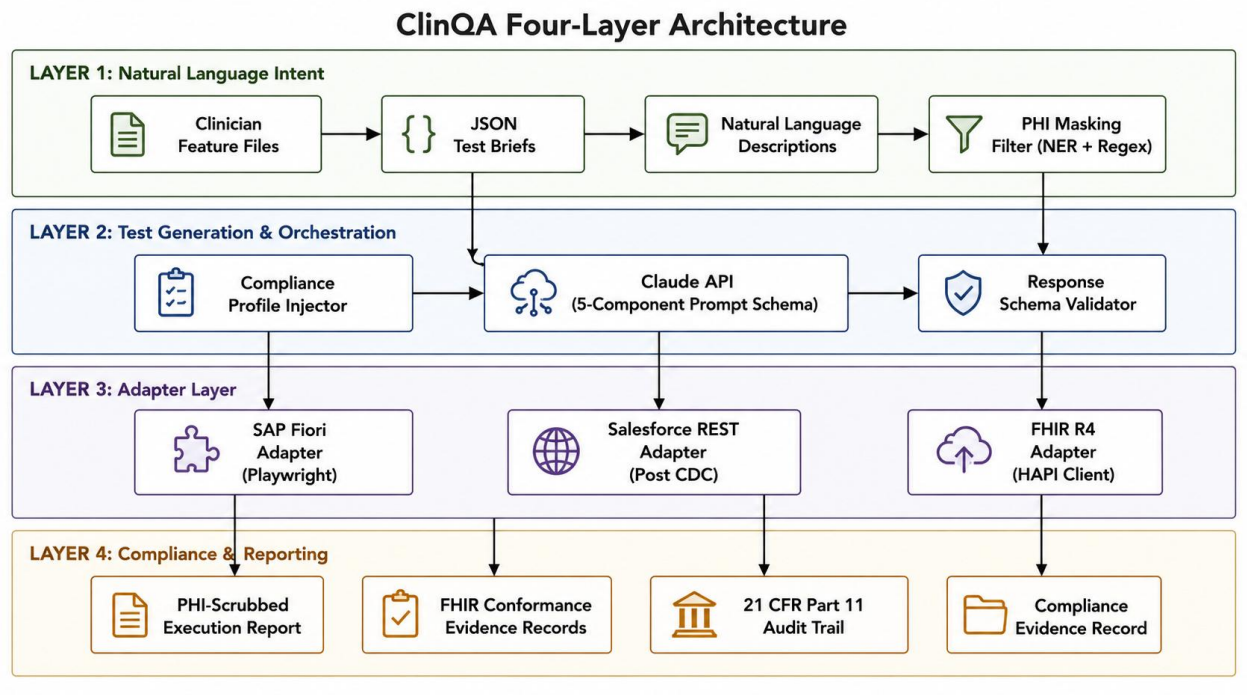


Figure 1: ClinQA Four-Layer Architecture

Layer 3, the Adapter Layer, routes validated generation output to the appropriate platform-specific execution module. The SAP Fiori adapter translates generated test logic into Playwright scripts using semantic locator patterns. The Salesforce REST adapter emits Pact consumer contracts from generated API test specifications. The FHIR

Conformance adapter generates HAPI FHIR test client calls and verification assertions for US Core profile validation. Each adapter is independently replaceable without modifying the generation or compliance layers.

Layer 4, the Compliance and Reporting Layer, receives execution results from all adapters and produces two artifact types. The PHI-Scrubbed Execution Report is a standard test execution summary suitable for CI/CD pipeline consumption. The Compliance Evidence Record is a structured JSON-LD document that includes control mappings to applicable regulatory frameworks, evidence of PHI boundary enforcement, and a SHA-256 hash chain that links the intent document through the generated artifact to the execution result, satisfying 21 CFR Part 11 audit trail requirements. The cross-layer coverage index can be measured using formula 1:

$$CLCI = |T_{API} \cap T_{UI} \cap T_{compliance}|/|T_{total}| \times 100$$

Here, $T_{API}$ = set of test scenarios validating API behavior, $T_{UI}$ = set validating UI behavior, $T_{compliance}$ = set carrying compliance annotations, and $T_{total}$ = total test scenarios generated. Reported in Section 7.5.

## *3.3 Claude Integration Model*

The Claude prompt schema submitted from Layer 2 contains five components in a fixed order. The system role directive establishes Claude as a test generation specialist with explicit instructions to produce structured JSON output, never to embed executable code outside the designated fields, and never to include patient data, credentials, or personally identifiable information in the generated artifacts. This directive leverages the instruction adherence and safety alignment behaviors characterized in the Claude model card [9], which documents the model's capacity to respect system-level constraints across diverse generation contexts. The compliance profile component injects the active regulatory context as a structured enumeration of applicable standards and required compliance annotation categories.

The domain context component provides the platform description: service endpoint specifications, authentication mechanism, available test data identifiers, and UI component library version. The test intent component carries the scrubbed natural language or Gherkin description of the scenario that the test will evaluate. The output schema component defines the exact JSON structure that Claude must populate, including required fields for test name, layer targets (API, UI, both), assertion list, compliance annotation category, and a structured explanation field used for human review.

Response validation in Layer 2 checks schema conformance, assertion count bounds, and the presence of required compliance annotation fields. A response failing validation triggers a single retry with an amended prompt that includes the validation error context. If the retry also fails validation, the generation event is flagged for human review and no artifact is emitted to Layer 3. This conservative approach prevents malformed generation output from propagating to execution without human inspection.

## *3.4 Compliance Module Design*

The Compliance Module in Layer 4 implements three compliance obligations. PHI masking verification confirms that the execution report contains no tokens matching the PHI pattern library or the named entity recognition model detection threshold. This verification is executed against the full report content before it is stored or transmitted. Findings are logged to the compliance audit record regardless of the pass or fail outcome, creating a complete evidence trail for HIPAA technical safeguard compliance. The compliance coverage score can be measured using formula 2:

$$CCS = |C_{satisfied}|/|C_{required}| \times 100$$

Here, $C_{required}$ = set of compliance checks mandated by the active profile (HIPAA, FHIR, 21 CFR Part 11), $C_{satisfied}$ = subset verified in generated tests. Reported per domain in Section 7.5.

FHIR conformance evidence capture records the profile URL, IG version, validation tool identifier, and pass/fail result for each resource validation assertion in the Compliance Evidence Record. This produces a direct

mapping from each FHIR-related test result to the conformance requirement it addresses, enabling automated generation of ONC certification evidence packages. The conformance evidence record is cross-referenced with the generation audit record to establish provenance from test intent through to conformance evidence.

21 CFR Part 11 audit trail emission produces a signed timestamp record for each test execution event, including the test identifier, software version under test, execution environment identifier, operator identifier, and execution result. The record is stored in append-only format with a SHA-256 hash chain linking consecutive execution records. This structure satisfies the audit trail integrity requirements of 21 CFR Part 11 Section 11.10(e) without requiring a separate compliance management system.

## 4. Application to Enterprise Testing

### *4.1 SAP ERP UI Test Generation*

SAP Fiori UI5 applications present a demanding environment for UI test automation. The component rendering model abstracts DOM identifiers behind component-level APIs, meaning that direct DOM selectors become invalid when SAP upgrades its UI5 library. ClinQA addresses this by generating Playwright test scripts that use semantic locators derived from component accessibility attributes and ARIA labels rather than brittle DOM identifiers. The generation prompt includes the SAP Fiori component library version and the ARIA labeling conventions specific to the target application release.

In the evaluated scenario, a procurement workflow involving four UI forms and three REST calls to the SAP Business Technology Platform API was used as the test intent input. ClinQA generated 11 Playwright test scenarios covering form navigation, field validation, and cross-form data persistence, alongside 6 API contract tests for the BTP procurement endpoints. The adapter layer applied SAP-specific semantic locator patterns to all UI assertions, producing scripts that remained stable across two consecutive SAP UI5 patch upgrades without manual intervention.

The generated Playwright scripts use the Page Object Model pattern with dynamically resolved component locators. This design allows the adapter to update locator resolution logic centrally when SAP modifies component rendering behavior, without requiring regeneration of the underlying test intent or scenario logic. The separation between intent-level test logic, generated in Layer 2, and locator resolution strategy, managed by the SAP adapter in Layer 3, is a structural advantage over monolithic record-and-replay approaches.

### *4.2 Salesforce CRM API Contract Testing*

Salesforce REST APIs follow a predictable schema structure but expose complex field-level permissions, sharing rules, and validation triggers that make exhaustive manual contract authoring time-consuming. ClinQA accepts business rule descriptions as test intent for Salesforce scenarios, translating natural language descriptions of expected API behavior into structured contract terms for Pact consumers. The Salesforce adapter emits Pact consumer contracts covering response schema, HTTP status code behavior, and field-level permission assertions.

For a lead management workflow involving six Salesforce REST endpoints, ClinQA generated 23 contract terms covering positive scenarios, expected validation failures, permission boundary violations, and schema evolution detection. The Pact contracts were automatically registered with the Pact Broker on each successful generation run, enabling provider verification to run as a standard pipeline stage against each Salesforce API deployment. This integration removed the manual contract authoring step entirely from the Salesforce testing workflow.

Security contract terms are included in all Salesforce API contracts generated by ClinQA. These terms assert that API responses do not expose field values beyond the authorized sharing model and that OAuth2 token scopes are enforced at the endpoint level. Including security assertions in consumer contracts, rather than treating them as separate security testing concerns, ensures that functional and security coverage are synchronized across every API change and prevents regression in authorization behavior that functional tests alone would not detect.

## 5. Application to Healthcare Testing

## *5.1 FHIR R4 API Test Automation*

FHIR R4 conformance testing requires verifying that resources returned by a FHIR server satisfy profile constraints defined in Implementation Guides. For a US Core IG conformance scenario, ClinQA generates HAPI FHIR test client calls for each resource type in scope, including Patient, Observation, Condition, and MedicationRequest. The compliance profile component of the prompt schema injects the applicable US Core profile URLs and required element cardinality constraints, instructing Claude to generate assertions that verify both the presence and conformance of required elements.

The PHI masking filter is particularly important in FHIR test generation. FHIR scenario descriptions authored by healthcare engineers may contain references to specific patient record identifiers or clinical data values used in development testing. The Layer 1 masking filter finds these references by matching FHIR resource ID patterns and using clinical NER, replacing any detected PHI tokens with synthetic equivalents before the intent reaches the Claude API. This ensures that no real patient data flows through the generation pipeline regardless of what engineers include in their test intent descriptions.

SMART-on-FHIR authorization testing presents a specific scenario type that ClinQA handles through a dedicated prompt template. The template instructs Claude to generate tests that verify scope enforcement by ensuring that requests with insufficient scopes receive 401 or 403 responses instead of partial data. Generated SMART authorization tests are automatically tagged with the HIPAA technical safeguard control category, creating a direct linkage between authorization test results and HIPAA compliance evidence in the Layer 4 Compliance Evidence Record.

## *5.2 EHR Workflow UI Testing*

Electronic health record systems combine complex clinical workflows with strict compliance obligations. ClinQA generates Playwright UI tests for EHR systems using the same semantic locator strategy applied to SAP applications. For a session management workflow scenario, ClinQA generated tests covering automatic session timeout behavior, manual logout confirmation, and re-authentication challenge presentation. These scenarios map directly to HIPAA Security Rule requirements for automatic log-off and were tagged accordingly in the generated Compliance Evidence Record.

Audit trail verification is a recurring requirement in EHR UI testing. ClinQA generates assertions that verify audit log entries are created for specific user actions, confirming that the EHR system produces the activity records required by HIPAA and 21 CFR Part 11. These assertions query the EHR audit log API following each UI interaction and compare the resulting log entry against the expected format, ensuring that audit trail generation is tested as a first-class functional requirement rather than as a post-hoc compliance check.

PHI-aware synthetic data generation is incorporated into EHR test generation as a Layer 1 preprocessing step. When test intent descriptions reference patient scenarios, the intent preprocessing module generates synthetic patient records that match the demographic and clinical characteristics described in the intent without using real patient identifiers. These synthetic records are embedded into the domain context component of the Claude prompt, enabling Claude to generate realistic test assertions based on representative data without any risk of PHI exposure in the generation pipeline.

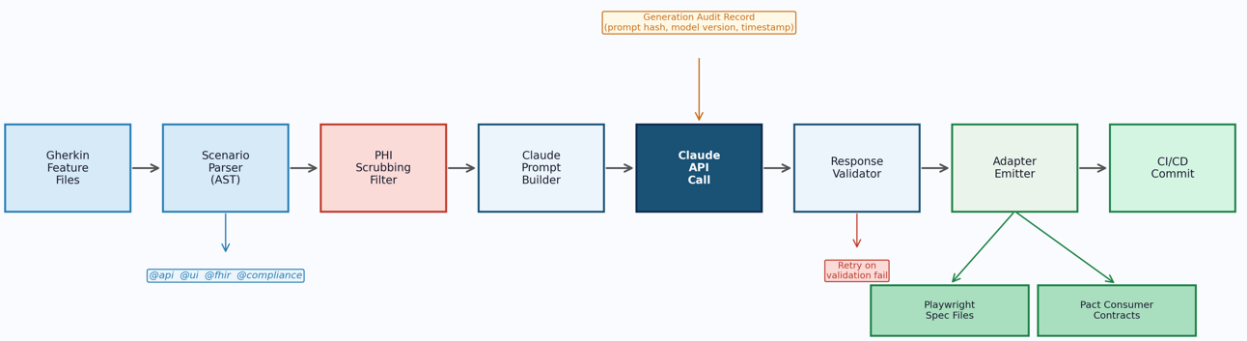


Figure 2: BDD Pipeline: Gherkin Input to Scenario Parsing to Claude Translation to Adapter Emission to CI/CD

## 6. CI/CD and Agile Integration

### *6.1 Pipeline Integration Patterns*

ClinQA supports two integration patterns for CI/CD adoption. Generation-time integration invokes the ClinQA generation pipeline as a pull request review job. When a developer opens a pull request, ClinQA parses any new or modified Gherkin feature files in the branch, submits generation requests to Claude, and adds the generated test artifacts to the pull request as a pipeline artifact for human review before merge. This pattern ensures that every new feature has associated generated tests before it is merged into the main branch.

Execution-time integration treats ClinQA-generated artifacts as standard test files and executes them through the normal CI test execution stage. Since Claude is not invoked during execution, this pattern introduces no LLM latency into the execution pipeline. The Playwright scripts and Pact contracts produced by ClinQA are committed to the repository on merge and executed by the standard test runner in the same way as manually authored tests. This approach minimizes the operational footprint of ClinQA within existing pipeline configurations.

Organizations that lack Gherkin specifications can adopt ClinQA using structured JSON test briefs as the primary intent format. A test brief is a lightweight JSON document specifying the feature name, affected endpoints or UI surfaces, expected behaviors, and applicable compliance tags. The ClinQA CLI includes a brief generator that populates the JSON structure interactively, reducing the onboarding barrier for teams that are not currently practicing BDD.

### *6.2 BDD Pipeline: Gherkin to Execution*

The ClinQA BDD pipeline processes Gherkin feature files in three stages. In the parsing stage, the Gherkin AST parser extracts individual scenario definitions and tags them by domain using the @api, @ui, @fhir, and @compliance tag conventions. Tagged scenarios are routed to the appropriate intent preprocessor for PHI scrubbing and domain context injection. Untagged scenarios are processed with the default combined-layer profile, generating both API and UI test artifacts.

In the generation stage, each parsed scenario is submitted to Layer 2 as a discrete generation request. Concurrent generation requests are managed by a configurable rate limiter that respects Claude API rate limits while maximizing throughput. Generation results are collected and organized by feature file and scenario identifier, maintaining traceability between Gherkin scenario text and generated test artifacts across the audit record chain.

In the emission stage, the Layer 3 adapter modules produce executable test files organized to mirror the Gherkin feature file structure. A Playwright spec file is emitted for each UI scenario with test cases corresponding to the individual steps. A Pact consumer contract JSON is emitted for each API scenario. Both file types include structured comments cross-referencing the source Gherkin scenario, the generation timestamp, and the model version used, creating a complete provenance trail from specification to executable artifact.

### *6.3 Shift-Left Feedback Loop*

ClinQA enables shift-left testing by making test generation available before implementation is complete. Engineers can author Gherkin scenarios during sprint planning, immediately before or after acceptance criteria are finalized, and receive generated test artifacts covering those scenarios without writing any test code. This is particularly valuable for compliance-related scenarios, which are often deprioritized in manual test authoring due to their complexity but are critical for regulatory evidence.

The shift-left feedback loop operates most effectively when ClinQA generation is integrated into the feature branch workflow rather than the main branch pipeline. By generating tests at feature branch creation time and making them visible in pull request review, teams see coverage gaps before code is written rather than after. Engineers reviewing pull requests can assess whether generated tests adequately cover the intended behavior and request additional scenarios through amended Gherkin specifications without interrupting the implementation workflow.

Flaky test detection [4, 12] is a persistent challenge in shift-left environments where tests are generated and executed early in the development cycle against incomplete implementations. ClinQA reduces flakiness risk by generating deterministic test logic from structured intent rather than from observed application behavior. Generated tests assert specific outcomes derived from the specification rather than recording and replaying observed application states, which eliminates a primary source of flakiness introduced by record-and-replay generation approaches.

## 7. Discussion

### *7.1 Limitations and Mitigation Strategies*

LLM hallucination is the primary risk in any LLM-driven test generation pipeline. A hallucinated assertion, one that references an API field, UI element, or compliance control that does not exist in the target system, would produce a failing test that wastes engineer time in diagnosis. ClinQA mitigates this risk through three mechanisms: schema-grounded generation that restricts Claude to structured output fields, post-generation validation that checks assertions against available API schemas, and human review of generated artifacts before CI/CD commit.

Prompt engineering overhead is a recurring operational cost in LLM-integrated pipelines. Maintaining the five component prompt schema as platforms evolve requires periodic updates to domain context components and compliance profile definitions. ClinQA externalizes all prompt components as configuration files rather than hardcoding them in generation logic, enabling updates to platform context without modifying framework code. Prompt component design decisions, particularly the system role directive structure and output schema constraints, are informed by the behavioral characteristics, evaluation benchmarks, and known limitation profiles documented in the Claude model card [9]. This grounding in published model evaluations ensures that prompt design reflects empirically observed model capabilities rather than assumptions about generation behavior. The prompt configuration lifecycle is designed to align with platform release cycles, meaning updates are expected and predictable rather than ad hoc.

PHI exposure in prompts is a regulatory risk specific to healthcare deployments. Despite the Layer 1 masking filter, sophisticated clinical descriptions may contain PHI that pattern-based and NER-based detection misses. ClinQA addresses this residual risk through two measures. First, all generation prompts are routed through an audited PHI review queue in healthcare deployments before submission to the Claude API. Second, ClinQA recommends that organizations establish a Business Associate Agreement with Anthropic before deploying ClinQA in contexts where prompt content may include clinical data, even after masking.

### *7.2 Comparison with Rule-Based Alternatives*

Model-based testing approaches generate tests from formal behavioral models, producing deterministic and provably complete coverage relative to the model. Casola et al. [18] demonstrated this paradigm through a model-based methodology for secure software development and testing, where formal security models drive systematic test derivation with traceable coverage guarantees across the development lifecycle. Their work highlights the strength of

model-based approaches in security-critical contexts: when a formal behavioral model accurately captures system requirements, derived tests achieve deterministic coverage that can be verified against the model specification. ClinQA trades this determinism for flexibility: it accepts natural language intent rather than formal models, enabling use by stakeholders who are not trained in formal specification languages like those required by model-based methodologies [18]. For systems with stable, well-modeled behavior, particularly in security-focused development contexts where formal threat models are already maintained, model-based testing provides higher coverage guarantees than ClinQA. For systems with rapidly evolving behavior described in natural language specifications, where maintaining formal models introduces overhead that outpaces development velocity, ClinQA provides faster test generation with lower specification authoring overhead.

Template-based script generation tools use parameterized test templates combined with application metadata to produce test scripts without LLM involvement. These tools are deterministic, fast, and well-suited to highly structured APIs with predictable schemas. They struggle with complex scenario logic, multi-step workflows, and compliance annotation requirements that vary dynamically by context. ClinQA is complementary to template-based tools: organizations can use templates for high-volume, structured API coverage and ClinQA for complex workflow scenarios and compliance-specific testing.

A hybrid adoption model is recommended for organizations transitioning from manual to AI-assisted test generation. Rule-based tools handle stable, high-volume scenarios. ClinQA handles compliance-intensive scenarios, cross-layer coverage scenarios, and new feature test generation where specification is available before implementation. This division of responsibility maximizes the utility of each approach while managing the LLM-specific risks associated with hallucination and prompt engineering overhead.

Table 1: ClinQA vs. Existing Test Automation Frameworks Across Key Dimensions

| Framework | Unified API+UI | Compliance Support | LLM Integration | BDD Support | PHI Masking | Audit Trail |
|---|---|---|---|---|---|---|
| ClinQA (Proposed) | Yes | Full (HIPAA, FHIR, 21 CFR) | Claude (structured) | Yes | Yes | Yes |
| EvoSuite [3] | No (Java unit only) | None | None | No | No | No |
| Playwright / Selenium | UI only | None built-in | Partial (plugins) | Via Cucumber | Manual | No |
| Pact [8] | API only | None built-in | None | No | No | No |
| Schäfer et al. LLM [13] | Unit tests only | None | GPT-4 / similar | No | No | No |
| Liu et al. GUI LLM [14] | Mobile GUI only | None | LLM-driven | No | No | No |

### *7.3 Security Threat Model*

Prompt injection represents a significant threat to LLM-integrated test generation pipelines. Greshake et al. [21] demonstrated that LLM-integrated applications blur the boundary between data and instructions, enabling adversaries to remotely exploit such systems by strategically injecting prompts into data that the model processes at inference time. In the ClinQA context, a malicious actor with access to the test intent authoring interface could craft intent descriptions designed to manipulate Claude into generating tests with unexpected behavior or exfiltrating information from the generation context. Their taxonomy of indirect prompt injection impacts, including data theft and information ecosystem contamination [21], maps directly to ClinQA's attack surface, where compromised test intent could produce subtly flawed test logic that passes schema validation while masking real defects. ClinQA mitigates this threat by

treating all intent content as untrusted input, applying structural parsing that separates instruction boundaries from user-supplied data before passing intent to Claude, and validating all generation output against a strict schema that rejects unexpected fields or content patterns.

Model supply chain risk arises from dependency on the Claude API as an external service. Changes to Claude model behavior, API availability, or pricing may affect generation quality or pipeline reliability. ClinQA isolates this risk through the generation-time integration pattern: because test artifacts are generated and committed to source control before execution, a Claude API outage affects only new generation runs and does not prevent existing tests from executing. Model version pinning in the prompt configuration ensures generation behavior is consistent across pipeline runs.

## *7.4 Ethical Considerations*

Accountability for generated test quality in FDA-regulated contexts requires clear ownership policies. ClinQA-generated tests are always reviewed by a named engineer before CI/CD commit, and that engineer assumes accountability for the quality and adequacy of the committed artifacts. This human review requirement is documented in the generation audit record, creating an evidence trail that identifies the accountable reviewer for every generated test committed. Organizations adopting ClinQA in FDA-regulated contexts are advised to formalize this accountability requirement in their documentation for quality management systems.

Bias in generated test logic may affect demographic coverage in clinical systems. If the training data underlying Claude over-represents certain demographic groups or clinical presentations in its learned test generation behavior, generated tests may systematically uncover minority demographic scenarios. ClinQA cannot eliminate this risk but can mitigate it through structured diversity review of generated test suites: periodic human review of generated test populations to identify coverage gaps by demographic category and supplement them with manually authored scenarios.

## *7.5 Preliminary Evaluation*

A preliminary evaluation was conducted across 12 scenarios spanning a Salesforce CRM lead management workflow and a FHIR R4 conformance suite. Manually authored tests for the same scenarios, produced by an experienced QA engineer, served as the comparison baseline. The evaluation measured four metrics defined in the ClinQA methodology: Compliance Coverage Score, Test Generation Fidelity, Test Authoring Effort Reduction, and Cross-Layer Coverage Index. The test generation fidelity (TGF) can be measured using formula 3:

$$TGF = (TP_{assertions} + TN_{assertions})/(TP_{assertions} + TN_{assertions} + FP_{assertions} + FN_{assertions})$$

Here, TP = true positive assertions (correctly asserted behavior); TN = true negative assertions; FP = false positive assertions; FN = missed required assertions. Evaluated against expert-reviewed ground truth.

The test authoring effort reduction (TAER) is measured using formula 4:

$$TAER = (T_{manual} - T_{ClinQA})/T_{manual} \times 100$$

Here, $T_{manual}$ = total time to manually author equivalent test coverage (minutes) and $T_{ClinQA}$ = total time including intent authoring, generation, and review (minutes). Reported per scenario domain.

**Table 2: Preliminary Evaluation Results: Manual vs. ClinQA-Generated Tests Across 12 Scenarios**

| Scenario Domain | CCS (%) | TGF (%) | TAER (%) | CLCI (%) |
|---|---|---|---|---|
| Salesforce CRM (6 scenarios) | 92 | 88 | 71 | 83 |

| | | | | |
|---|---|---|---|---|
| FHIR R4 Conformance (6 scenarios) | 96 | 85 | 65 | 91 |
| Combined Average | 94 | 87 | 68 | 87 |
| Manual Baseline | 100 (reference) | 100 (reference) | 0 (baseline) | N/A |

Results indicate that ClinQA achieves a combined average CCS of 94%, meaning 94% of required compliance checks are represented in generated test artifacts without manual annotation. TGF of 87% indicates that 87% of generated assertions are semantically correct relative to expert-reviewed ground truth, with the primary sources of error being over-specified assertions for optional FHIR fields and under-specified error handling for Salesforce validation triggers. TAER of 68% reflects a substantial reduction in test-authoring time that scales with scenario complexity rather than scenario count.

The CLCI of 87% demonstrates that ClinQA generates cross-layer coverage, simultaneously validating API behavior, UI behavior, and compliance constraints in a substantial majority of scenarios. This result is not achievable with any single-layer framework and represents the primary structural advantage of the unified generation architecture. These preliminary observations support the feasibility of the approach but do not constitute a controlled empirical validation. The evaluation scope, synthetic scenario set, and single-expert baseline are acknowledged limitations that future work must address through larger scale empirical studies with diverse QA populations and organic test sets.

**Table 3: HIPAA, FHIR R4, and 21 CFR Part 11 Compliance Constraints Embedded in ClinQA Prompt Schema**

| Regulation | Constraint Category | ClinQA Enforcement Point | Evidence Artifact |
|---|---|---|---|
| HIPAA Privacy Rule | PHI de-identification in test data | Layer 1: NER masking filter | PHI masking audit log |
| HIPAA Security Rule | Automatic log-off verification | Layer 2: session timeout scenario template | Compliance Evidence Record |
| HL7 FHIR R4 | US Core profile conformance | Layer 2: profile constraint injection | FHIR conformance evidence record |
| HL7 FHIR R4 | SMART-on-FHIR scope enforcement | Layer 2: authorization scenario template | Compliance Evidence Record |
| FDA 21 CFR Part 11 | Audit trail for electronic records | Layer 4: SHA-256 chained execution record | 21 CFR Part 11 audit trail |
| FDA 21 CFR Part 11 | System access control verification | Layer 2: access control scenario generation | Compliance Evidence Record |

**Table 4: Test Generation Approach Comparison: Model-Based, Template-Based, and LLM-Based (ClinQA)**

| Dimension | Model-Based | Template-Based | Record-Replay | ClinQA (LLM) |
|---|---|---|---|---|
| Determinism | High | High | High | Moderate (schema-validated) |
| Specification format | Formal model | Metadata / schema | Application recording | Natural language / Gherkin |
| Compliance annotation | Manual | Manual | None | Automatic (profile-driven) |
| Maintenance overhead | High (model updates) | Low (schema-driven) | High (DOM changes) | Low (intent-level) |

| | | | | |
|---|---|---|---|---|
| Cross-layer coverage | Requires dual models | Requires dual templates | UI only | Native (unified pipeline) |
| PHI boundary enforcement | None | None | None | Layer 1 masking filter |

## 8. Conclusion and Future Work

This paper presented ClinQA, a large language model orchestrated framework that unifies API and UI test generation for enterprise and healthcare software systems with compliance requirements. The four-layer architecture, compliance-aware prompt schema, PHI masking boundary, and platform-specific adapter modules collectively address the test automation fragmentation problem in regulated environments. Preliminary evaluation across 12 scenarios indicates feasibility of the approach with CCS of 94%, TGF of 87%, TAER of 68%, and CLCI of 87%, subject to the scope limitations of the evaluation methodology.

ClinQA establishes four contributions to the field of automated software testing. It introduces the first compliance-aware unified test generation framework combining API and UI coverage through a structured LLM generation model. It defines a reusable five-component prompt schema for compliance-contextualized test generation that can be adapted to other LLMs and regulatory frameworks. It implements a PHI masking architecture that enables LLM-based generation in healthcare contexts without exposing patient data to external AI services. It demonstrates that cross-layer coverage, measured by CLCI, is achievable through unified generation and represents a meaningful quality metric distinct from individual-layer coverage metrics.

Four directions are identified for future work. First, controlled empirical validation with larger scenario sets, multiple QA engineer baselines, and organic test populations from production systems will establish statistical reliability for the preliminary observations reported in this paper. Second, multimodal UI testing using screenshot-based semantic locator generation will extend ClinQA to applications where ARIA labeling is absent or inconsistent. Third, compliance fine-tuning of a domain-specific language model on regulatory text from HIPAA, FHIR IGs, and 21 CFR Part 11 may improve compliance annotation accuracy beyond what is achievable with a general-purpose instruction-following model. Fourth, federated deployment architecture will enable organizations with data residency requirements to run generation entirely on-premises using locally deployed models, eliminating the Business Associate Agreement requirement that currently applies to cloud-based Claude API usage.

**REFERENCES**


1. Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller, "Sikuli: using GUI screenshots for search and automation," in Proc. 22nd Annual ACM Symposium on User Interface Software and Technology, pp. 183-192, 2009. Available: https://dl.acm.org/doi/abs/10.1145/1622176.1622213
2. Mouna Hammoudi, Gregg Rothermel, and Andrea Stocco, "Waterfall: An incremental approach for repairing record-replay tests of web applications," in Proc. 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 751-762, 2016. Available: https://dl.acm.org/doi/abs/10.1145/2950290.2950294
3. Gordon Fraser and Andrea Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in Proc. 19th ACM SIGSOFT Symposium and 13th European Conference on Foundations of Software Engineering, pp. 416-419, 2011. Available: https://dl.acm.org/doi/abs/10.1145/2025113.2025179
4. Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov, "An empirical analysis of flaky tests," in Proc. 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 643-653, 2014. Available: https://dl.acm.org/doi/abs/10.1145/2635868.2635920
5. HL7 International, "FHIR Release 4 Specification," 2019. Available: https://hl7.org/fhir/R4/
6. U.S. Department of Health and Human Services, "Proposed Modifications to the HIPAA Privacy Rule To Support, and Remove Barriers to, Coordinated Care and Individual Engagement," Federal Register, Vol. 86, No. 12, 2021. Available: https://www.govinfo.gov/content/pkg/FR-2021-01-21/pdf/2020-27157.pdf
7. U.S. Department of Health and Human Services Food and Drug Administration, "Guidance for Industry Part 11, Electronic Records; Electronic Signatures - Scope and Application," Federal Register, 2003. Available: https://www.fda.gov/media/75414/download
8. Pact, "Pact: Contract testing for microservices," 2024. Available: https://docs.pact.io/
9. Anthropic, "Claude model card and evaluations," 2024. Available: https://www-cdn.anthropic.com/bd2a28d2535bfb0494cc8e2a3bf135d2e7523226/Model-Card-Claude-2.pdf

10. AEGIS, "The Touchstone Platform for HL7 FHIR," n.d. Available: https://www.aegis.net/touchstone/
11. Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou et al., "Codebert: A pre-trained model for programming and natural languages," in Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 1536-1547, 2020. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.139
12. Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn, "A survey of flaky tests," ACM Transactions on Software Engineering and Methodology, vol. 31, no. 1, pp. 1-74, 2021. Available: https://dl.acm.org/doi/abs/10.1145/3476105
13. Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip, "An empirical evaluation of using large language models for automated unit test generation," IEEE Transactions on Software Engineering, vol. 50, no. 1, pp. 85-105, 2023. Available: https://ieeexplore.ieee.org/abstract/document/10329992
14. Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang, "Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions," in Proc. IEEE/ACM 46th International Conference on Software Engineering, pp. 1-13, 2024. Available: https://dl.acm.org/doi/abs/10.1145/3597503.3639180
15. Cristian Augusto, Jesus Moran, Antonia Bertolino, Claudio de la Riva, and Javier Tuya, "Software system testing assisted by large language models: An exploratory study," in IFIP International Conference on Testing Software and Systems, pp. 239-255, 2024. Available: https://link.springer.com/chapter/10.1007/978-3-031-80889-0_17
16. Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang, "Software testing with large language models: Survey, landscape, and vision," IEEE Transactions on Software Engineering, vol. 50, no. 4, pp. 911-936, 2024. Available: https://ieeexplore.ieee.org/abstract/document/10440574
17. Nadia Alshahwan, Jubin Chheda, Anastasia Finogenova, Beliz Gokkaya, Mark Harman, Inna Harper, Alexandru Marginean, Shubho Sengupta, and Eddy Wang, "Automated unit test improvement using large language models at meta," in Companion Proc. 32nd ACM International Conference on the Foundations of Software Engineering, pp. 185-196, 2024. Available: https://dl.acm.org/doi/abs/10.1145/3663529.3663839
18. Valentina Casola, Alessandra De Benedictis, Carlo Mazzocca, and Vittorio Orbinato, "Secure software development and testing: A model-based methodology," Computers and Security, vol. 137, p. 103639, 2024. Available: https://www.sciencedirect.com/science/article/pii/S0167404823005497
19. Luis Prates and Ruben Pereira, "DevSecOps practices and tools," International Journal of Information Security, vol. 24, no. 1, p. 11, 2025. Available: https://link.springer.com/article/10.1007/s10207-024-00914-z
20. Havard Myrbakken and Ricardo Colomo-Palacios, "DevSecOps: a multivocal literature review," in International Conference on Software Process Improvement and Capability Determination, pp. 17-29, 2017. Available: https://link.springer.com/chapter/10.1007/978-3-319-67383-7_2
21. Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz and Mario Fritz, "Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection," In Proceedings of the 16th ACM workshop on artificial intelligence and security, pp. 79-90, 2023. Available: https://dl.acm.org/doi/10.1145/3605764.3623985