International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group | Volume 5, Issue 4, 192-202, 2024
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V5I4P120

Original Article

AI-Assisted Requirements Engineering for Agile Software Development: A Transformer-Based Framework for Requirement Classification, Conflict Detection, and Traceability

Dr. Bhaskaran .S[1], Dr. Brindha .GR[2], Dr. Emily Jenifer .A[3], Dr. Kalai Vazhi .R[4]
[1,2,3,4]Assistant Professor, School of Computing, SASTRA Deemed University, Thanjavur, Tamilnadu, India.

Abstract *- Agile software development relies on rapid elicitation, continuous refinement, and iterative delivery of requirements, yet many teams still manage requirements through manual story grooming, spreadsheet-based trace matrices, and ad hoc quality checks. These practices are increasingly insufficient when software systems must satisfy complex functional expectations, non-functional constraints, regulatory obligations, cybersecurity controls, and architecture-dependent deployment risks. This paper proposes T-REACT, a transformer-based requirements engineering framework for agile software development that integrates requirement classification, semantic conflict detection, and automated traceability into a unified decision-support pipeline. Rather than treating requirements engineering as a document-production activity, the proposed framework models backlog items, user stories, acceptance criteria, test cases, defects, architecture decisions, and deployment risks as interconnected semantic artifacts. The framework uses contextual embeddings, fine-tuned requirement classifiers, pairwise contradiction scoring, dependency-aware graph reasoning, and human-in-the-loop governance to support sprint planning, backlog refinement, change impact analysis, and release readiness decisions. The study follows a design-science research approach and presents the framework architecture, data model, model-training workflow, evaluation protocol, agile integration pattern, and governance controls. A demonstration scenario shows how the framework identifies requirement categories, highlights likely conflicts, recommends trace links, and produces auditable explanations for product owners, business analysts, architects, testers, and compliance stakeholders. The paper contributes an end-to-end research artifact that connects transformer-based natural language understanding with agile lifecycle governance, emphasizing explainability, traceability, privacy, continuous learning, and operational adoption. The proposed framework is intended to reduce ambiguity, improve backlog quality, support earlier defect prevention, and strengthen alignment among requirements, design, testing, and release decisions.*

Keywords *- Requirements Engineering, Agile Software Development, Transformers, BERT, Conflict Detection, Requirement Classification, Traceability, Software Governance, Machine Learning, Natural Language Processing*.

1. Introduction

Requirements engineering remains one of the most consequential activities in software development because it shapes the system boundary, expected behavior, quality constraints, stakeholder priorities, and acceptance criteria before implementation decisions become expensive to reverse. In agile software development, requirements are rarely frozen in a single specification; they evolve through user stories, epics, refinement conversations, acceptance tests, architecture decisions, and production feedback. This evolution improves responsiveness, but it also creates a persistent risk of ambiguity, inconsistency, duplication, unverified assumptions, and weak traceability between stakeholder intent and delivered software. Standards for requirements engineering emphasize quality attributes such as correctness, consistency, completeness, feasibility, necessity, traceability, and verifiability, but agile teams often struggle to enforce these qualities continuously during fast sprint cycles [9].

The challenge is not merely that agile requirements are informal. The deeper problem is that agile artifacts are distributed across tools, people, ceremonies, repositories, and delivery pipelines. A single customer capability can appear as an epic in a product management tool, as multiple user stories in a sprint backlog, as acceptance criteria in a ticket, as API behavior in an interface contract, as tests in a continuous integration pipeline, and as risks in a release governance board. When these artifacts are not semantically linked, teams cannot easily determine whether a changed requirement invalidates an architecture decision, whether two stories express conflicting security rules, whether a non-functional requirement has test coverage, or whether a defect indicates an upstream requirement-quality problem. Traditional traceability recovery research has shown that information retrieval can support link discovery among software artifacts, but agile contexts require faster, more contextual, and more continuously updated mechanisms [7].

Recent advances in transformer-based natural language processing provide a foundation for addressing this problem. The Transformer architecture introduced self-attention as a scalable mechanism for modeling token relationships without recurrence, enabling contextual representation of complex language sequences [1]. BERT later demonstrated that bidirectional pre-training can produce language representations suitable for downstream classification and inference tasks with limited task-specific architecture changes [5]. Sentence-BERT extended transformer representations for efficient semantic similarity estimation, which is especially relevant for duplicate requirement detection and traceability recovery [11]. These advances are promising for requirements engineering because requirements are primarily expressed in natural language, but adoption requires more than inserting a classifier into a backlog tool.

Existing requirement classification studies have demonstrated that supervised machine learning can separate functional and non-functional requirements, including categories such as security, performance, usability, operational behavior, and reliability [13]. However, agile requirements engineering requires a broader architecture than classification alone. Conflict detection needs pairwise semantic reasoning, domain constraints, and dependency awareness. Traceability requires similarity search, link ranking, graph updates, and human validation. Governance requires auditability, explanation, privacy, feedback loops, and integration with continuous delivery decisions. Recent work on AI-driven software lifecycle governance and architecture-centered project management indicates that machine learning becomes more valuable when embedded in lifecycle decision processes rather than isolated prediction tasks [8].

This paper proposes T-REACT, a transformer-based framework for AI-assisted requirements engineering in agile software development. The name stands for Transformer Requirements Engineering for Agile Classification and Traceability. The framework is designed around three core tasks: requirement classification, conflict detection, and traceability recommendation. Requirement classification labels backlog artifacts according to functional and non-functional categories. Conflict detection identifies contradictions, overlaps, feasibility risks, dependency clashes, and policy violations. Traceability recommendation links requirements to epics, user stories, acceptance criteria, tests, defects, architecture decisions, deployment controls, and compliance obligations. These tasks are unified through a semantic artifact graph that represents agile requirements as evolving nodes and links.

The contribution of this paper is not a review of existing studies. It is a research-oriented framework that specifies an implementable architecture, model workflow, governance mechanism, and evaluation protocol for transformer-assisted requirements engineering. The framework is intentionally designed for agile environments where requirements are continuously updated, teams rely on cross-functional collaboration, and release decisions must consider both product value and technical risk. The paper makes four contributions. First, it defines a transformer-based architecture for requirement classification, conflict detection, and traceability in a single lifecycle pipeline. Second, it introduces a semantic artifact graph that connects natural language requirements with software engineering artifacts. Third, it proposes a human-in-the-loop governance process for agile adoption, including explanation, approval, feedback, and audit controls. Fourth, it presents an evaluation protocol suitable for empirical validation using backlog corpora, benchmark datasets, and industrial agile repositories.

2. Research Problem and Motivation

Agile methods reduce the distance between stakeholder feedback and software delivery, but they also intensify requirements volatility. Product owners refine priorities sprint by sprint, business rules change as market conditions evolve, and technical constraints emerge during implementation. Requirements therefore become living artifacts rather than static contractual statements. This creates a need for continuous requirements intelligence: the ability to classify, compare, link, and govern requirements as they change. Traditional software requirement specifications provide useful structure, but their document-centered orientation is difficult to maintain when teams work through distributed agile backlogs [21].

Three limitations motivate the proposed framework. The first limitation is weak requirement classification. Teams frequently mix functional behavior, performance expectations, regulatory obligations, security controls, user-interface constraints, data governance rules, and operational policies in the same backlog item. This causes planning errors because different requirement types demand different analysis and testing strategies. Security and privacy requirements require threat modeling and compliance review. Performance requirements require measurable thresholds and load tests. Usability requirements require user validation. Reliability requirements require operational observability and failure-mode analysis. Automated classification can help teams route requirements to the right specialists early in the lifecycle. Prior work on automated non-functional requirement classification shows that machine learning can support the early detection of quality-related requirements [3].

The second limitation is insufficient conflict detection. A conflict may be direct, such as one story requiring password expiration every thirty days while another prohibits forced password resets. A conflict may also be indirect, such as a performance requirement for sub-second response time clashing with a compliance requirement for synchronous fraud screening. Agile teams often discover these conflicts during implementation or testing, when remediation is costly. Conflict

detection in requirements engineering has been explored through clustering and machine learning, but many approaches remain detached from daily backlog refinement and architecture governance [25].

The third limitation is incomplete traceability. Traceability is difficult because agile artifacts are numerous, heterogeneous, and continuously changing. A story may be linked to tests at the time of sprint execution but lose alignment after refactoring. A defect may reveal that an acceptance criterion was incomplete, but that insight may not feed back into the requirement model. A compliance requirement may be implemented across multiple services, but release reviewers may not see the full chain of evidence. Systematic mapping of information retrieval approaches to traceability recovery demonstrates the importance of evaluation strength and artifact-linking evidence in software maintenance [19].

The motivation for T-REACT is therefore to create a unified framework that helps agile teams reason about requirements before they become defects, rework, or governance failures. The framework does not replace product owners, analysts, architects, testers, or engineers. Instead, it assists them by surfacing likely classifications, contradictions, missing links, and impact paths. This distinction is important because requirements engineering is a socio-technical activity. Stakeholder intent, domain knowledge, and organizational constraints cannot be fully automated. The role of AI is to reduce cognitive load, improve consistency, and provide decision-support evidence.

3. Conceptual Foundation

The proposed framework builds on three conceptual foundations: transformer language representation, lifecycle governance, and agile artifact traceability. Transformer models are suitable for requirements because they encode context. A requirement such as “The system shall authenticate high-value payments before release” cannot be interpreted only by keyword matching. The word “release” may refer to payment release, software release, or compliance approval depending on surrounding terms. Contextual embeddings help distinguish these meanings, particularly when requirements contain domain-specific vocabulary. RoBERTa showed that pre-training design choices and training data scale significantly affect language model performance, which supports the need to fine-tune and evaluate model variants rather than assume a single generic model is optimal [15].

Lifecycle governance is necessary because AI-generated recommendations can influence planning, testing, and release decisions. In regulated domains, requirements decisions must be auditable, explainable, and aligned with organizational policy. Frameworks for AI-driven decision intelligence in software lifecycle governance emphasize that prediction capabilities should be connected to architecture, testing, risk controls, and operational decision flows [28]. This perspective is central to T-REACT. Requirement classification is not treated as a static label. It becomes an input to downstream routing, test planning, acceptance criteria generation, and release risk assessment.

Agile traceability requires a graph-oriented representation. Backlog items, tests, defects, architecture decisions, services, APIs, controls, and deployment events are related through many-to-many links. Graph-based modeling is useful because failure propagation and dependency analysis often depend on paths across services and artifacts rather than isolated nodes [24]. In T-REACT, the semantic artifact graph stores both machine-generated candidate links and human-approved links. This separation supports governance because the model can recommend traces, but humans decide whether the links become authoritative.

The framework also draws from research on AI-enhanced observability and automated root cause analysis. Requirements do not stop being relevant after deployment; production incidents often reveal missing non-functional requirements, inadequate acceptance criteria, or inaccurate assumptions. AI-enhanced observability architectures demonstrate the value of connecting runtime evidence to upstream engineering decisions [16]. T-REACT extends this idea by allowing defects and operational incidents to become feedback signals for requirement refinement and traceability correction.

4. Research Objectives and Questions

The main objective of this study is to design a transformer-based framework that improves the quality and governance of agile requirements through automated classification, conflict detection, and traceability recommendation. The objective is operational rather than purely theoretical. The framework must be deployable within agile toolchains, interpretable by practitioners, and adaptable across domains with different vocabulary and compliance constraints.

The study is guided by four research questions:

- RQ1: How can transformer-based language models classify agile requirements into functional and non-functional categories while preserving domain-specific meaning?
- RQ2: How can semantic similarity, natural language inference, and constraint-aware reasoning be combined to detect conflicts among backlog items and related artifacts?
- RQ3: How can transformer embeddings and graph-based link ranking improve traceability among requirements, tests, defects, architecture decisions, and deployment controls?

- RQ4: How can AI-assisted requirements engineering be integrated into agile ceremonies and lifecycle governance without undermining human accountability?

These questions reflect a design-science orientation. The goal is to build and specify an artifact that solves a relevant class of problems, then define how it can be evaluated rigorously. This approach is appropriate because the research problem concerns the design of an integrated socio-technical system, not only the isolated accuracy of a model.

5. Proposed Framework: T-REACT

T-REACT consists of six layers: ingestion, normalization, transformer representation, task-specific inference, semantic artifact graph, and governance interface. The ingestion layer connects to agile repositories such as backlog management systems, test management systems, source repositories, continuous integration pipelines, incident tools, and architecture decision records. Each imported artifact is assigned metadata such as source, author, timestamp, sprint, component, business capability, status, and revision history. This layer ensures that the framework can process agile artifacts as they evolve rather than relying on periodic manual exports.

The normalization layer transforms heterogeneous artifacts into a consistent internal representation. User stories are decomposed into role, goal, benefit, acceptance criteria, and constraints when such fields are available. Free-text requirements are segmented into atomic requirement statements where possible. Defect reports are separated into observed behavior, expected behavior, environment, severity, and reproduction steps. Test cases are represented by objective, precondition, action, expected result, and automation status. This normalization does not force teams to abandon their agile tools; it creates a machine-readable structure behind existing workflows.

The transformer representation layer generates contextual embeddings for each artifact. The framework supports two encoder modes. The first mode uses a fine-tuned transformer encoder for classification and pairwise inference. The second mode uses a sentence-embedding model for scalable similarity search. Sentence-BERT is particularly relevant for traceability because it reduces the computational cost of comparing many candidate artifact pairs while preserving semantic similarity information [11]. The framework stores embeddings in a vector index and updates them whenever artifacts change.

The task-specific inference layer performs three core functions. The classification function assigns labels such as functional, security, performance, usability, reliability, maintainability, operational, compliance, data, interface, and architecture constraint. The conflict detection function compares pairs or clusters of related requirements and assigns conflict types such as contradiction, duplication, threshold mismatch, role-policy mismatch, dependency inconsistency, and feasibility risk. The traceability function recommends links among requirements and downstream artifacts using embedding similarity, metadata compatibility, graph proximity, and historical approval patterns.

The semantic artifact graph stores artifacts as nodes and relationships as edges. Edge types include refines, duplicates, conflicts-with, depends-on, verifies, implements, violates, mitigates, supersedes, and impacts. Edges can be machine-suggested, analyst-approved, rejected, expired, or manually created. This distinction is essential because machine recommendations should not be treated as final truth. The graph can support impact analysis by showing which tests, components, services, or compliance controls may be affected by a changed requirement.

The governance interface presents recommendations to users through role-specific views. Product owners see duplicate and conflicting backlog items. Business analysts see classification quality and ambiguity risks. Architects see dependency and feasibility warnings. Testers see missing verification links. Release managers see traceability coverage and unresolved conflicts. Compliance teams see auditable decision pathways. Explainable AI research in regulated banking contexts highlights the importance of auditable decision pathways when AI supports operational decisions [6]. T-REACT applies that principle to requirements governance.

6. Requirement Classification Model

The classification component treats agile requirements as multi-label text classification instances. A backlog item may be both functional and security-related, or both performance-related and operational. Therefore, T-REACT does not force a single exclusive label unless the organization explicitly requires one. Multi-label classification reflects real agile practice, where one story often contains behavior, constraints, and acceptance criteria. Software requirement classification studies show that feature extraction choices and classifier design influence performance, especially when datasets are imbalanced [27].

The classification model uses a transformer encoder followed by task-specific classification heads. The first head predicts a high-level class: functional, non-functional, constraint, data, interface, compliance, or unknown. The second head predicts finer categories such as security, usability, performance, reliability, maintainability, availability, auditability, privacy, interoperability, and operational support. The third head predicts quality-risk attributes such as ambiguous, unverifiable,

incomplete, compound, inconsistent, or under-specified. This layered output is designed to support agile workflows. A story classified as performance-related and unverifiable can be routed for measurable acceptance criteria before sprint commitment.

Training data can come from a combination of benchmark datasets, historical organizational backlogs, and expert-labeled samples. Public datasets provide a starting point, but organizational vocabulary usually differs. Banking teams may use terms such as KYC, AML, settlement, reconciliation, and fraud screening. Healthcare teams may use prescription, HIPAA, pharmacy claim, prior authorization, and patient safety. Domain-specific software architectures, such as cloud-native prescription processing systems, demonstrate why requirements models must adapt to operational and regulatory vocabulary [30].

T-REACT therefore uses incremental fine-tuning. An initial model is trained on a general requirements corpus. The model is then adapted using a smaller set of organization-specific labeled requirements. Active learning is used to select uncertain examples for expert review. Human reviewers can correct labels directly in the agile interface. Corrected examples are added to a controlled training pool after governance checks. This process reduces annotation effort and helps the model learn local terminology.

Classification explanations are generated using evidence spans and label rationales. For example, if a story is labeled as security-related, the interface highlights phrases such as “multi-factor authentication,” “privileged access,” or “encryption at rest.” If a story is labeled as unverifiable, the interface highlights vague terms such as “fast,” “user-friendly,” “robust,” or “secure” when no measurable acceptance criteria are provided. The goal is not only to assign labels but also to improve requirement quality during refinement.

7. Conflict Detection Model

Conflict detection in agile requirements is more complex than duplicate detection. Two requirements can use different words but express contradictory behavior. Conversely, two requirements can appear different but be compatible because they apply to different user roles, jurisdictions, system states, or data classes. T-REACT combines transformer-based semantic inference with rule-based and graph-based constraints.

The conflict detection pipeline begins by narrowing candidate pairs. Comparing every requirement with every other requirement is computationally expensive and produces too many irrelevant pairs. Candidate generation uses embedding similarity, shared metadata, same epic, same component, same business capability, same compliance domain, and graph proximity. This step identifies pairs likely to require comparison. The framework then applies a cross-encoder or natural language inference model to classify the relationship between two artifacts as entailment, contradiction, neutral relation, duplication, refinement, or potential conflict.

The model also uses structured constraints. Some conflicts are not purely linguistic. For example, one requirement may specify a maximum response time of two seconds, while another specifies synchronous external validation that usually exceeds that threshold. Another requirement may require data deletion after thirty days, while an audit rule requires retention for seven years. These conflicts involve numeric thresholds, temporal rules, regulatory categories, and operational dependencies. T-REACT therefore extracts entities such as role, action, object, condition, threshold, time window, jurisdiction, service, and data classification.

The conflict types supported by the framework include semantic contradiction, duplicate intent, inconsistent threshold, missing exception, incompatible role authorization, conflicting data retention, interface mismatch, dependency conflict, and feasibility conflict. The output includes a confidence score, evidence spans, affected artifacts, and recommended action. The recommended action may be merge, split, clarify, escalate to architect, add acceptance criterion, create compliance review, or defer until dependency analysis is complete. Automated testing research in enterprise Java systems emphasizes that reliability depends not only on test execution but also on upstream specification clarity and coverage [4].

The conflict detection module is integrated into backlog refinement. When a product owner edits a story, the model checks whether the new wording conflicts with existing stories, tests, compliance controls, or architecture decisions. During sprint planning, unresolved high-severity conflicts are surfaced as readiness risks. During release review, conflicts linked to implemented features are treated as governance exceptions. This integration is important because conflict detection performed only as a periodic audit may discover problems too late.

8. Traceability Recommendation Model

Traceability recommendation is the third core function of T-REACT. The objective is to recover and maintain links among agile artifacts. Traceability is not limited to requirement-to-test links. The framework supports requirement-to-epic, requirement-to-acceptance-criterion, requirement-to-test, requirement-to-defect, requirement-to-code-component, requirement-to-architecture-decision, requirement-to-risk-control, and requirement-to-release links. Traceability recovery research has long

recognized the value of information retrieval for linking software artifacts, but transformer embeddings can improve semantic matching when exact keywords differ [7].

The traceability model uses a hybrid ranking strategy. The first signal is semantic similarity between artifact embeddings. The second signal is metadata compatibility, such as shared component, sprint, epic, product area, or regulatory domain. The third signal is graph proximity, where artifacts connected through common nodes are more likely to be trace candidates. The fourth signal is historical link behavior, where previously approved links provide weak supervision for future ranking. The fifth signal is artifact lifecycle state, because an obsolete requirement should not usually be linked to a current release unless it represents a superseded baseline.

The framework ranks candidate links and presents them with explanations. For example, a requirement about "encrypting prescription attachments in storage" may be linked to a security test that checks server-side encryption, an architecture decision about storage bucket policies, and a compliance control about protected health information. Cloud-native pharmacy automation studies show that operational workflows can involve OCR, microservices, prescription processing, and secure integration, making traceability across artifacts especially important [14].

Traceability links are assigned statuses. A suggested link is generated by the model. A confirmed link is approved by a human. A rejected link is explicitly marked as incorrect. An expired link becomes stale because one artifact changed significantly. A mandatory link is required by organizational policy. These statuses support auditability and continuous improvement. When a confirmed link later becomes semantically weak because a requirement has changed, T-REACT flags it for review.

Traceability coverage is also measured. Coverage metrics include percentage of requirements with at least one acceptance criterion, percentage of non-functional requirements with measurable tests, percentage of compliance requirements linked to controls, percentage of defects linked to originating requirements, and percentage of architecture decisions linked to impacted capabilities. These metrics support governance dashboards and sprint retrospectives. They also help teams identify systemic weaknesses, such as recurring performance requirements without test coverage.

9. Agile Integration Pattern

T-REACT is designed to support agile ceremonies without turning agile into a heavyweight documentation process. During backlog intake, the framework classifies new items, identifies vague wording, and recommends decomposition when a story appears compound. During backlog refinement, it surfaces duplicates, conflicts, missing acceptance criteria, and likely trace links. During sprint planning, it produces readiness indicators for stories selected into the sprint. During implementation, it updates trace links as code, tests, and defects appear. During sprint review, it compares delivered evidence against acceptance criteria. During retrospectives, it identifies recurring requirement-quality patterns.

The framework can also support continuous integration and continuous delivery. Machine-learning-driven risk detection in banking CI/CD environments shows how deployment decisions can incorporate predictive signals and real-world evaluation evidence [2]. T-REACT extends this idea upstream by adding requirement-quality and traceability signals to release governance. For example, a release may be blocked if a high-risk compliance requirement has no confirmed test link or if a security-related story has an unresolved conflict with an architecture decision.

The agile integration pattern is intentionally lightweight. Recommendations are embedded into existing work items rather than presented in a separate analysis portal. A product owner can see classification labels and conflict warnings directly in the backlog tool. A tester can see missing verification links from the test management interface. An architect can see dependency conflicts in an architecture dashboard. This reduces context switching and improves adoption.

The framework supports role-based thresholds. A product team may allow low-confidence duplicate suggestions during refinement but require high-confidence conflict warnings during sprint commitment. A regulated team may require mandatory human review for compliance-related labels. A platform engineering team may prioritize dependency conflicts and operational traceability. The thresholds are governed by policy, not hardcoded into the model.

10. Governance, Explainability and Privacy

AI-assisted requirements engineering introduces governance risks. A wrong classification may route a requirement incorrectly. A false conflict warning may waste analyst time. A missed trace link may create a compliance gap. A model trained on sensitive backlog data may expose confidential product strategy or personal information. Therefore, T-REACT includes governance controls at the architecture level.

First, the framework separates recommendation from authorization. The model can suggest labels, conflicts, and links, but humans approve authoritative changes. Second, every recommendation includes an explanation. Classification explanations

include highlighted text spans and label rationale. Conflict explanations include the two compared artifacts, evidence spans, conflict type, and impacted graph paths. Traceability explanations include similarity score, metadata overlap, and graph evidence. Third, all actions are logged for audit. The log records model version, artifact version, recommendation output, user decision, timestamp, and rationale.

Privacy is addressed through data minimization, access control, and deployment choice. In some organizations, requirements may contain sensitive customer information, protected health data, financial controls, or proprietary strategy. Federated learning research for privacy-preserving fraud detection illustrates that model training can be designed to reduce centralized data exposure across institutions [18]. For T-REACT, similar privacy principles can be applied through local fine-tuning, restricted embedding storage, encrypted indexes, and role-based access to artifact content.

Explainability also supports organizational learning. When users reject a recommendation, they can provide a reason such as “different jurisdiction,” “obsolete requirement,” “false duplicate,” “accepted exception,” or “domain-specific term.” These reasons become feedback signals for improving the model and refining rules. Over time, the system learns not only language patterns but also organizational decision norms.

Governance must also consider model drift. Requirements vocabulary changes as products evolve. New regulations, architectures, and platform services introduce new terms. A model trained on last year’s backlog may underperform on a new product line. T-REACT monitors drift through changes in confidence distribution, rejection rate, unknown-label frequency, and embedding-cluster shifts. When drift is detected, the framework triggers targeted review and retraining.

11. Demonstration Scenario

Consider an agile team building a digital payment platform. The backlog includes the following requirements:

- R1: “As a retail customer, I want high-value transfers to be approved immediately so that I can complete urgent payments.”
- R2: “The system shall perform enhanced fraud screening before releasing any transfer above the regulatory threshold.”
- R3: “High-value transfer approval shall not exceed two seconds under normal load.”
- R4: “Fraud-screening decisions must be explainable and retained for audit review.”
- R5: “Customers shall be notified when a transfer is delayed due to risk review.”

The classification module labels R1 as functional and performance-sensitive. R2 is labeled as security, compliance, and fraud-risk related. R3 is labeled as performance and operational. R4 is labeled as compliance, auditability, and explainability. R5 is labeled as functional and usability-related. Research on regulatory-grade fraud detection using explainable AI supports the importance of auditable decision pathways in such domains [6].

The conflict detection module identifies a potential feasibility conflict between R2 and R3. The first requirement requires enhanced fraud screening before release, while the second requires approval within two seconds. This is not a direct contradiction because fraud screening may be fast enough, but it is a feasibility risk that requires architectural analysis. The recommended action is to clarify whether the two-second threshold includes external fraud checks, whether asynchronous approval is acceptable, and whether exceptions apply for suspicious transfers.

The traceability module recommends linking R2 to fraud screening service tests, R4 to audit-log retention controls, and R5 to user-notification acceptance criteria. It also recommends linking R2 and R4 to an architecture decision about risk-decision explainability. Comparative studies of machine learning models for defect prediction demonstrate the broader lifecycle value of predictive evidence when connected to software quality decisions [20].

During sprint planning, the framework marks R2 and R4 as not ready until acceptance criteria specify measurable audit and explanation outputs. During implementation, the traceability graph links fraud-service integration tests to R2 and audit-log verification tests to R4. During release review, the dashboard shows that the high-value payment feature still has an unresolved feasibility warning between R2 and R3. The release manager can then request a documented exception or require the team to resolve the requirement mismatch before deployment.

This demonstration illustrates the framework’s practical value. It does not merely classify requirements. It connects classification with conflict detection, traceability, architecture review, testing, and release governance.

12. Evaluation Protocol

A rigorous evaluation of T-REACT should assess model performance, workflow impact, governance quality, and practitioner usability. The first evaluation dimension is classification accuracy. Metrics include precision, recall, macro-F1,

micro-F1, label-wise F1, and calibration error. Macro-F1 is particularly important because non-functional categories are often imbalanced. Deep learning studies on software requirement classification highlight the importance of dataset choice and category balance when evaluating requirement classifiers [23].

The second dimension is conflict detection quality. Metrics include precision at top-k, recall over known conflict pairs, false-positive rate, analyst acceptance rate, and time-to-resolution. Because conflict labels are expensive to obtain, evaluation can combine benchmark cases, expert-labeled industrial pairs, and mutation-based synthetic conflicts. Mutation-based evaluation creates controlled contradictions by altering thresholds, roles, negations, temporal conditions, or data-retention terms. However, synthetic conflicts should be reported separately from real industrial conflicts to avoid overstating external validity.

The third dimension is traceability performance. Metrics include mean average precision, recall at k, mean reciprocal rank, link approval rate, stale-link detection rate, and coverage improvement. The evaluation should compare T-REACT against keyword search, TF-IDF, latent semantic indexing, and generic embedding baselines. Historical traceability work indicates that baseline comparison is essential for understanding whether a new method improves practical link recovery [19].

The fourth dimension is workflow impact. Metrics include reduction in duplicate stories, reduction in late conflict discovery, improvement in acceptance-criteria completeness, improvement in test-link coverage, and analyst time saved during refinement. These metrics should be collected across multiple sprints to avoid overfitting conclusions to one project phase. Decision-intelligence approaches to lifecycle governance emphasize that AI value should be assessed through operational decision outcomes, not only isolated model scores [10].

The fifth dimension is trust and usability. Practitioner studies should measure perceived usefulness, explanation clarity, recommendation fatigue, willingness to adopt, and perceived accountability. A technically accurate model may fail if it interrupts agile flow, produces too many alerts, or lacks clear explanations. Therefore, user experience evaluation is not optional. It is a core component of framework validation.

A recommended empirical study design is a mixed-methods field evaluation. The study can begin with retrospective analysis of historical backlogs to establish baseline performance. It can then proceed to a shadow-mode deployment in which the model generates recommendations without influencing team decisions. Finally, the framework can be activated in selected agile ceremonies, with human approval required for all recommendations. This staged deployment reduces risk and allows comparative analysis.

13. Implementation Architecture

A reference implementation of T-REACT can be built as a set of microservices. The ingestion service connects to backlog, test, source control, CI/CD, and incident tools through APIs. The preprocessing service normalizes artifacts and extracts structured fields. The embedding service generates and stores vector representations. The classification service performs multi-label prediction. The conflict service performs candidate generation and pairwise inference. The traceability service ranks candidate links and updates the semantic artifact graph. The governance service manages approvals, explanations, audit logs, and policy thresholds.

Microservice architecture is appropriate because different components have different scaling requirements. Embedding generation may require GPU acceleration. Graph queries may require a graph database. Audit logs may require immutable storage. User-interface integration may require low-latency APIs. Research on secure microservices architecture for HIPAA-compliant prescription processing demonstrates the importance of security, deployment governance, and platform design when microservices handle sensitive workflows [30].

The framework can be deployed in cloud-native, on-premises, or hybrid environments. In regulated organizations, sensitive artifacts may remain within a private network while generic pre-trained models are imported and fine-tuned locally. The vector index should be encrypted because embeddings may leak semantic information. Access to recommendations should follow the same authorization model as the underlying artifacts. Audit logs should be tamper-evident when used for regulatory evidence.

Model operations are also required. Each model version must be registered with training data lineage, hyperparameters, evaluation results, known limitations, and approval status. When a model is updated, the system should record which recommendations were generated by which version. This supports reproducibility and accountability. Predictive analytics and caching patterns in operational pharmacy systems show how performance optimization must be aligned with domain workflow requirements rather than treated as a purely technical layer [22].

14. Discussion

T-REACT addresses a gap between natural language processing research and agile requirements practice. Many prior studies focus on a single task, such as classifying functional and non-functional requirements. T-REACT instead connects classification, conflict detection, and traceability through a semantic graph and governance workflow. This integration is important because requirement quality problems are interdependent. A vague requirement is harder to classify, harder to test, and harder to trace. A missing trace link may hide a conflict. A conflict may indicate an architectural dependency that was not represented in the backlog.

The framework also aligns with the growing role of machine learning in software development. Studies on the future of software development and the expanding role of ML models suggest that AI will increasingly assist planning, analysis, testing, and lifecycle decision-making [26]. However, T-REACT avoids the assumption that AI should autonomously decide requirement correctness. Requirements are negotiated artifacts. They express stakeholder values, domain policies, business constraints, and engineering trade-offs. Human review remains essential.

Another important implication is that agile requirements engineering should become evidence-driven. Agile teams often rely on conversation and collaboration, which are valuable, but those conversations should produce traceable decisions when systems are complex or regulated. T-REACT provides mechanisms for capturing those decisions as approved labels, accepted conflicts, clarified requirements, and confirmed trace links. This improves continuity when team members change and supports audits when delivery decisions are questioned.

The framework is especially relevant for domains where requirements intersect with operational risk. Fraud detection, healthcare workflows, cloud observability, distributed systems, and CI/CD governance all involve requirements that have technical and regulatory consequences. AI-driven architecture and cybersecurity risk mitigation research shows that lifecycle optimization and risk control increasingly converge in modern software organizations [12]. T-REACT contributes to this convergence by bringing requirements intelligence closer to engineering execution.

15. Threats to Validity

Several threats to validity must be considered. The first is dataset bias. Public requirements datasets may not represent industrial agile backlogs, and industrial datasets may reflect one organization’s vocabulary. Fine-tuning on one domain may reduce generalizability to another. The second threat is label subjectivity. Requirements classification is not always objective. Analysts may disagree about whether a requirement is functional, operational, security-related, or compliance-related. Inter-annotator agreement should therefore be reported.

The third threat is conflict-label incompleteness. Known conflicts in historical datasets may represent only conflicts that were discovered and documented. Undiscovered conflicts cannot easily be counted. This affects recall evaluation. The fourth threat is recommendation fatigue. Even accurate models may become disruptive if they generate too many warnings. Evaluation should therefore include human acceptance rate and alert burden.

The fifth threat is organizational adoption. A framework that works in a controlled research setting may fail in real teams if it does not integrate with existing tools and ceremonies. The sixth threat is model drift. Agile vocabulary, architecture, and regulations change over time. Continuous monitoring and retraining are required. Fault prediction studies comparing machine learning models show that model effectiveness varies across datasets and conditions, reinforcing the need for repeated empirical validation [29].

16. Conclusion

This paper proposed T-REACT, a transformer-based framework for AI-assisted requirements engineering in agile software development. The framework integrates requirement classification, conflict detection, and traceability recommendation within a semantic artifact graph and human-in-the-loop governance workflow. It is designed for agile environments where requirements evolve continuously and where product, architecture, testing, compliance, and deployment decisions must remain aligned.

The paper presented the research problem, conceptual foundation, architecture, model workflow, agile integration pattern, governance controls, demonstration scenario, and evaluation protocol. The central argument is that transformer models can improve requirements engineering only when embedded into lifecycle decision processes. Classification without traceability is limited. Traceability without conflict detection is incomplete. Conflict detection without governance is risky. T-REACT addresses these dependencies through an integrated architecture.

Future empirical work should implement the reference architecture, evaluate it on public and industrial requirements corpora, compare multiple transformer variants, measure workflow impact across sprints, and study practitioner trust. Additional research should explore domain adaptation, privacy-preserving training, graph neural reasoning over artifact

networks, and integration with automated test generation. AI-assisted requirements engineering has the potential to reduce ambiguity, prevent late rework, improve compliance evidence, and strengthen agile delivery quality, but its success depends on careful design, transparent governance, and rigorous validation.

References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 30*, 5998–6008.

[2] Thalakanti, R. R., & Goud Bandari, S. S. (2024). Intelligent Continuous Integration and Delivery for Banking Systems using Machine Learning Driven Risk Detection with Real World Deployment Evaluation. International Journal of AI, BigData, Computational and Management Studies, 5(4), 168-175. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I4P118

[3] Cleland-Huang, J., Settimi, R., Zou, X., & Solc, P. (2007). Automated classification of non-functional requirements. *Requirements Engineering*, *12*(2), 103–120. https://doi.org/10.1007/s00766-007-0045-1

[4] Gudi, S. R. (2023). Enhancing Reliability in Java Enterprise Systems through Comparative Analysis of Automated Testing Frameworks. International Journal of Emerging Trends in Computer Science and Information Technology, 4(2), 151-160. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P115

[5] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 4171–4186.

[6] Bandari, S. S. G., Sivva, S. D., & Thalakanti, R. R. (2024). Regulatory Grade Fraud Detection using Explainable Artificial Intelligence with Auditable Decision Pathways and Empirical Validation on Banking Data. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(3), 139-147. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I3P115

[7] Hayes, J. H., Dekhtyar, A., & Osborne, J. (2003). Improving requirements tracing via information retrieval. *Proceedings of the 11th IEEE International Requirements Engineering Conference*, 151–161.

[8] Gunda SK, Yettapu SDR, Bodakunti S, Bikki SB. Decision Intelligence Methodology for AI-Driven Agile Software Lifecycle Governance and Architecture-Centered Project Management, 2023 Mar. 30;4(1):102-8. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P112

[9] ISO/IEC/IEEE. (2018). *ISO/IEC/IEEE 29148:2018 Systems and software engineering—Life cycle processes—Requirements engineering*. International Organization for Standardization.

[10] Sivva, S. D. (2023). An end-to-end AI-based systems engineering paradigm for lifecycle governance, predictive quality assurance, automation economics, and cybersecurity intelligence. Journal of Frontiers in Multidisciplinary Research, 4(1), 600–604. https://doi.org/10.54660/.JFMR.2023.4.1.600-604

[11] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, 3982–3992.

[12] Balerao, M. (2023). A converged artificial intelligence architecture for innovation, software lifecycle optimization, and cybersecurity risk mitigation. International Journal of Multidisciplinary Futuristic Development, 4(1), 117–120. https://doi.org/10.54660/IJMFD.2023.4.1.117-120

[13] Kurtanović, Z., & Maalej, W. (2017). Automatically classifying functional and non-functional requirements using supervised machine learning. *Proceedings of the 25th IEEE International Requirements Engineering Conference*, 490–495.

[14] Gudi, S. R. (2024). AI-Driven Fax-to-Digital Prescription Automation: A Cloud-Native Framework Using OCR, Machine Learning, and Microservices for Pharmacy Operations. International Journal of Emerging Research in Engineering and Technology, 5(1), 111-116. https://doi.org/10.63282/3050-922X.IJERET-V5I1P113

[15] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

[16] Manga, I., Sivva, S. D. ., & Manga, V. K. (2024). The Adaptive Intelligence in Cloud Systems: A Unified Architecture for AI Enhanced Observability and Automated Root Cause Analysis. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(1), 160-166. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I1P115

[17] Dalpiaz, F., Dell'Anna, D., Aydemir, F. B., & Çevikol, S. (2019). Requirements classification with interpretable machine learning and dependency parsing. *Proceedings of the 27th IEEE International Requirements Engineering Conference*, 142–152.

[18] Thalakanti, R. R., Goud Bandari, S. S., & Sivva, S. D. . (2024). Federated Learning for Privacy Preserving Fraud Detection across Financial Institutions: Architecture Protocols and Operational Governance. International Journal of Emerging Research in Engineering and Technology, 5(2), 108-114. https://doi.org/10.63282/3050-922X.IJERET-V5I2P111

[19] Borg, M., Runeson, P., & Ardö, A. (2014). Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering, 19*(6), 1565–1616. https://doi.org/10.1007/s10664-013-9255-y

[20] S. K. Gunda, "Comparative Analysis of Machine Learning Models for Software Defect Prediction," 2024 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, 2024, pp. 1-6, https://doi.org/10.1109/ICPECTS62210.2024.10780167

[21] IEEE Computer Society. (1998). *IEEE Recommended Practice for Software Requirements Specifications: IEEE Std 830-1998*. Institute of Electrical and Electronics Engineers.

[22] Gudi, S. R. (2024). Leveraging Predictive Analytics and Redis-Backed Caching to Optimize Specialty Medication Fulfillment and Pharmacy Inventory Management. International Journal of AI, BigData, Computational and Management Studies, 5(3), 155-160. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I3P116

[23] Li, B., Liang, Z., Li, Z., & Zhu, L. (2022). Automatically classifying non-functional requirements using deep neural network. *Pattern Recognition, 132*, 108948. https://doi.org/10.1016/j.patcog.2022.108948

[24] Mutyam, N. (2024). Graph-based modeling of service dependencies for predicting failure propagation in distributed systems. International Journal of Multidisciplinary Evolutionary Research, 5(1), 113–116. https://doi.org/10.54660/IJMER.2024.5.1.113-116

[25] Elhassan, H., Abaker, M., Abdelmaboud, A., & Rehman, M. B. (2022). Requirements engineering: Conflict detection automation using machine learning. *Intelligent Automation & Soft Computing*, *33*(1), 259–273. https://doi.org/10.32604/iasc.2022.023750

[26] Gunda, S. K. G. (2023). The Future of Software Development and the Expanding Role of ML Models. International Journal of Emerging Research in Engineering and Technology, 4(2), 126-129. https://doi.org/10.63282/3050-922X.IJERET-V4I2P113

[27] Canedo, E. D., & Mendes, B. C. (2020). Software requirements classification using machine learning algorithms. *Entropy*, *22*(9), 1057. https://doi.org/10.3390/e22091057

[28] Sivva, S. D., Thalakanti, R. R., Bandari, S. S. G., & Yettapu, S. D. R. (2023). AI-Driven Decision Intelligence for Agile Software Lifecycle Governance: An Architecture-Centered Framework Integrating Machine Learning Defect Prediction and Automated Testing. International Journal of Emerging Trends in Computer Science and Information Technology, 4(4), 167-172. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I4P118

[29] S. K. Gunda, "Fault Prediction Unveiled: Analyzing the Effectiveness of Random Forest, Logistic Regression, and KNeighbors," 2024 2nd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS), Erode, India, 2024, pp. 107-113, https://doi.org/10.1109/ICSSAS64001.2024.10760620

[30] Gudi, S. R. (2024). Design and Evaluation of Secure Microservices Architecture for HIPAA-Compliant Prescription Processing on AWS and OpenShift. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(2), 144-149. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I2P116