

Architecting the future: A governed, AI-enabled pipeline for next-generation requirements engineering

Joseph J. Brooks, *Riverside Research Institute*, jbrooks@riversideresearch.org

Abstract

The traditional “Software 1.0” requirements engineering (RE) lifecycle remains a significant source of defects, cost overruns, and project failure due to its reliance on manual, error-prone processes for interpreting unstructured stakeholder data. While recent advances in artificial intelligence (AI) have demonstrated promise in automating discrete RE tasks, there remains a critical gap in the design of a comprehensive, secure, and governable end-to-end architecture that integrates these capabilities. This study addresses this gap by employing a Design Science Research (DSR) methodology to conceptually design and evaluate a novel artifact: the AI-Enabled Requirements Engineering Pipeline. The proposed architecture integrates a Retrieval-Augmented Generation (RAG) core, a Multi-Agent System (MAS), and an Event-Driven Architecture (EDA) to enable scalable, context-aware automation of the RE lifecycle, from data ingestion and requirements generation to validation, traceability, and model integration. A key contribution of this work is the incorporation of a proactive governance framework that embeds security, auditability, and human oversight directly into the pipeline. This includes role-based access controls at the retrieval layer, defenses against prompt injection and data poisoning, tiered human-in-the-loop (HITL) escalation pathways, and the generation of an AI Bill of Materials (AIBOM) to ensure lifecycle transparency. Additionally, the conceptual architecture introduces automated validation mechanisms, including critic agents leveraging Natural Language Inference (NLI), and a real-time traceability graph to enhance explainability and impact analysis. The artifact is illustrated through a conceptual end-to-end workflow and analytically evaluated against established challenges in traditional RE, including defect propagation, rework costs, and traceability limitations. The findings indicate that the proposed architecture provides a theoretically grounded and practically implementable framework for improving the efficiency,

Keywords: Requirements Engineering, Artificial Intelligence, Event-Driven Architecture, Software 2.0, Systems Engineering, Multi-Agent Systems, Retrieval-Augmented Generation

Introduction

The traditional approach to RE is firmly rooted in the Software 1.0 paradigm, a world where developers manually translate human language into explicit, line-by-line instructions for a computer (Moakhar & Jabbarishiviari, 2024). This process is characterized by manual, time-consuming, and fragmented work. It begins with the difficult task of requirements elicitation, where business analysts attempt to draw out and document stakeholder needs through a series of interviews, workshops, and document analysis (Carbin, 2019). These high-level business requirements must then be painstakingly decomposed into detailed functional and non-functional requirements that developers can implement via the specification, modeling and validation stages of RE (Necula, et al., 2024).

This manual-centric model is fraught with inherent challenges. The process is notoriously slow and labor-intensive, creating a significant bottleneck at the very start of the software development lifecycle (SDLC) (Chakraborty, et al., 2012). More critically, it is a major source of errors; natural language is inherently ambiguous, and the manual translation from stakeholder intent to technical specification is prone to misinterpretation, omission, and inconsistency. Studies show that over half of all engineering defects originate in the requirements phase, making this a critical area for innovation (Necula, et al., 2024). Flawed requirements gathering is a trigger for 68% of failed projects, and nearly half of all projects require significant rework due to subpar requirements (Cheng, et al., 2026). These defects, born from ambiguity and manual error, ripple through the entire development process, leading to costly rework, schedule delays, and products that ultimately fail to meet the needs of their users (Ahmad, et al., 2023).

As systems grow in complexity, these issues are magnified. Modern engineering projects in fields from aerospace to finance involve a multitude of stakeholders, legacy systems, and regulatory constraints, generating a massive volume of unstructured data in the form of technical documents, support tickets, and user feedback (Xie, Culley, & Weber, 2011). Manually analyzing this data to synthesize a coherent and complete set of requirements is becoming practically impossible yet remains the most important steps of systems development work (Hidellaarachchi, et al., 2025). The Software 1.0 approach, which relies on human developers to explicitly code every instruction, simply cannot scale to meet the complexity and speed demanded by the next generation of software. This creates a clear and urgent imperative to re-imagine the very foundation of requirements engineering (Janisar, et al., 2026).

In response to these limitations, a new paradigm is revolutionizing the SDLC, colloquially known as software 2.0. As originally conceptualized by thought leaders like Andrej Karpathy, this marks a fundamental shift away from the traditional method of developers writing explicit, line-by-line instructions (Schlapbach, 2026). Instead, software 2.0 leverages the power of Artificial Intelligence (AI) to create software by training neural networks on high-quality, curated data. In this new paradigm, the very nature of code is redefined. The source code is no longer a human-readable file of logical instructions but the vast dataset itself, which implicitly defines the desired behavior of the program (Dilhara, Ketkar, & Dig, 2021). The neural network architecture acts as a flexible skeleton, and the process of training the network using optimization techniques like gradient descent becomes the act of compilation (Carbin, 2019). The final, compiled program is the trained neural network with its millions of learned weights, a complex and abstract representation of the solution that no human could ever write or comprehend directly (Dilhara, Ketkar, & Dig, 2021).

This marks a profound evolution in the role of developer. Their primary responsibility shifts from being a meticulous coder to becoming a data curator or AI teacher. Instead of writing explicit logic, the developer's main task is to collect, clean, label, and augment the massive datasets that program the system. When the AI produces a flawed output, the solution is not to write a patch of code, but to add more labeled examples to the training data to correct the behavior, effectively teaching the model the right way to perform, a practice that has been deployed by systems engineers at Microsoft (Ahmad, et al., 2023). This approach is uniquely suited to tackling the core challenges of traditional requirements engineering. The problem of scale, manually synthesizing requirements from thousands of documents, user feedback forums, and interview transcripts, is addressed by using Natural Language Processing (NLP) to automatically ingest and analyze this data. The issue of ambiguity in natural language is mitigated by training models on vast amounts of text, allowing them to understand context and nuance far more effectively than a rigid, rule-based system (Sonbol, Rebdawi, & Ghneim, 2022).

Finally, the bottleneck of manual documentation is alleviated by using generative AI to create initial drafts of requirements, (agile) user stories, and even system models. The AI-Enabled Requirements Engineering

Pipeline is the practical embodiment of this philosophy, applying the principles of software 2.0 to the most critical and defect-prone phase of the entire SDLC (Herwanto, Quirchmayr, & Tjoa, 2024). Again, this study presents a conceptual design reference architecture for an AI-Enabled Requirements Engineering Pipeline. While empirical validation remains a subject for future study, this research contributes a theoretically grounded design artifact intended to serve as a blueprint for addressing the limitations of traditional Software 1.0 RE processes.

Problem Statement

The traditional Software 1.0 RE lifecycles are a primary driver of project failure, characterized by manual, labor-intensive processes that are ill-equipped to manage the volume and complexity of modern systems. This reliance on manual interpretation of disparate, unstructured data sources consistently introduces high rates of defects, such as ambiguity, inconsistency, and incompleteness, at the earliest stage of development. While the potential of AI to automate discrete RE tasks is recognized, a significant gap exists for a comprehensive, end-to-end architectural framework that can orchestrate these capabilities in a secure, governed, and traceable manner. Therefore, the central problem is the lack of a reusable, agentic pipeline architecture that can systematically automate the RE lifecycle, from secure data ingestion to validated model generation, thereby mitigating the root causes of defects and providing the scalability and reliability required for next-generation systems engineering.

Research Questions

RQ1: How can a comprehensive, AI-enabled requirements engineering pipeline be architected to automate the RE lifecycle while ensuring security, governance, and traceability?

RQ2: How does the proposed AI-enabled requirements engineering pipeline mitigate key economic and operational challenges, such as defect propagation, rework, and traceability gaps, associated with traditional requirements engineering processes through its architectural design?

Review of Literature

This section provides a review of the existing body of knowledge across four key domains relevant to this research: (1) the documented failures and economic impact of traditional requirements engineering; (2) the paradigm shift towards AI-driven software 2.0; (3) the application of specific AI techniques to discrete requirements engineering tasks; and (4) the architectural and governance frameworks necessary for building robust, intelligent systems. This review establishes the theoretical foundation for the problem statement and justifies the need for novel, integrated pipeline architecture.

The Economic and Operational Failures of Software 1.0 Requirements Engineering

The traditional software 1.0 RE process, characterized by manual, document-centric, and plan-driven methodologies, is extensively documented in academic literature as a primary source of project failure. A significant body of research over the last decade has focused on quantifying the severe economic consequences of this paradigm. Studies consistently identify that many software defects originate in the RE phase. A report by Alessio et al., (2018) found that many defects can be traced back to errors made during requirements analysis and design. These defects are typically classified into categories of ambiguity, incompleteness, and inconsistency, which are inherent to the use of natural language as the primary medium for specification. Research in the Requirements Engineering Journal highlights that these defect types

account for 73% to 83% of all requirements errors (Nayebi & Maalej, 2016), creating a ripple effect of costly rework throughout the SDLC.

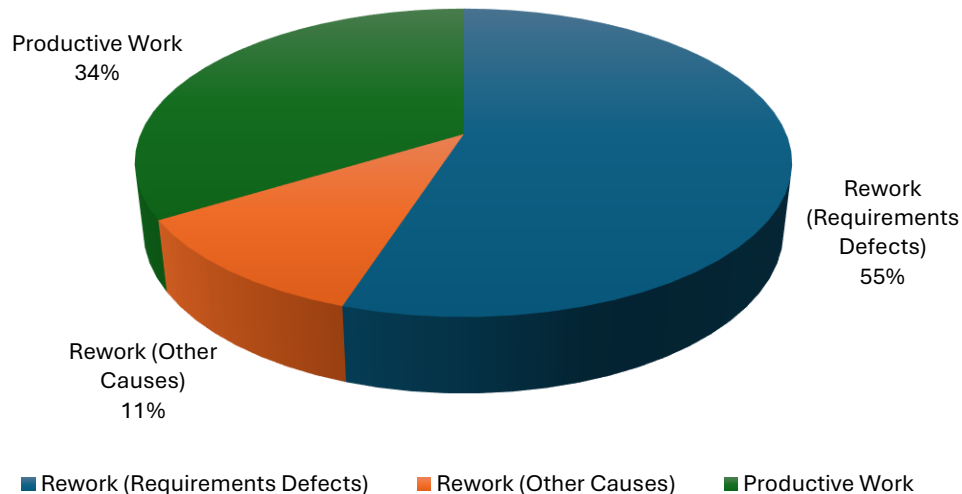


Figure 1. The Anatomy of Project Failure – Rework and Requirements Errors

The economic impact of this rework is staggering. A 2022 report from the Consortium for Information & Software Quality (CISQ) (2022) estimated the cost of poor software quality in the U.S. at \$2.41 trillion. Multiple studies corroborate that rework consumes 30% to 50% of total IT project budgets, with 70% to 85% of that rework being directly attributable to requirements errors (Consortium for Information And Software Quality, 2022). The cost of fixing these defects grows exponentially the later they are discovered. A granular NASA study conducted in 2004 found that a requirements error discovered during operations can cost up to 1,500 times more to fix than if it were caught during the initial design phase; there has been no indications that this statistic has changed over time (NASA Johnson Space Center, 2004). This economic pressure, combined with high project failure rates with some studies indicating that 31-39% of project cancellations are due to poor requirements creates a clear and urgent imperative for a new, more efficient and reliable paradigm (Chari & Agrawal, 2018).

Application of AI Techniques to Discrete RE Tasks

While the software 2.0 concept provides a high-level vision, a significant body of research has focused on applying specific AI and NLP techniques to automate discrete tasks within the traditional RE lifecycle. Research has illustrated the successful use of NLP for mining unstructured data sources. Techniques like sentiment analysis are used to gauge user pain points from app store reviews, while topic modeling can identify recurring themes from thousands of support tickets (Ludwig et al., 2026; Li et al., 2025). Named Entity Recognition (NER) models, particularly those based on transformer architectures like BERT (Bidirectional Encoder Representations from Transformers), have been successfully fine-tuned to identify domain-specific entities (e.g., system components, actors, constraints) from technical documents (Zhang et al., 2026). Large Language Models (LLMs) have been shown to be effective at generating initial drafts of requirements artifacts (Cheng et al., 2026). Studies have illustrated their ability to create user stories with acceptance criteria, use cases, and even initial system models (e.g., UML diagrams) from high-level natural language descriptions (Arora, Grundy, & Abdelrazek, 2024). AI is also being applied to the validation phase. Tools like QRA Corp's QVscribe use NLP to analyze requirements for linguistic quality, flagging ambiguity and inconsistency (O'Dell et al., 2025). Furthermore, Natural Language Inference (NLI) models are being used to create critic agents that can automatically verify the factual consistency of a generated requirement against its source document, a critical step in mitigating model hallucinations (Fazelnia et al.,

2024). While this research illustrates the viability of AI for individual tasks, it also highlights a key gap; the lack of a holistic, orchestrated framework that integrates these disparate capabilities into a single, end-to-end pipeline.

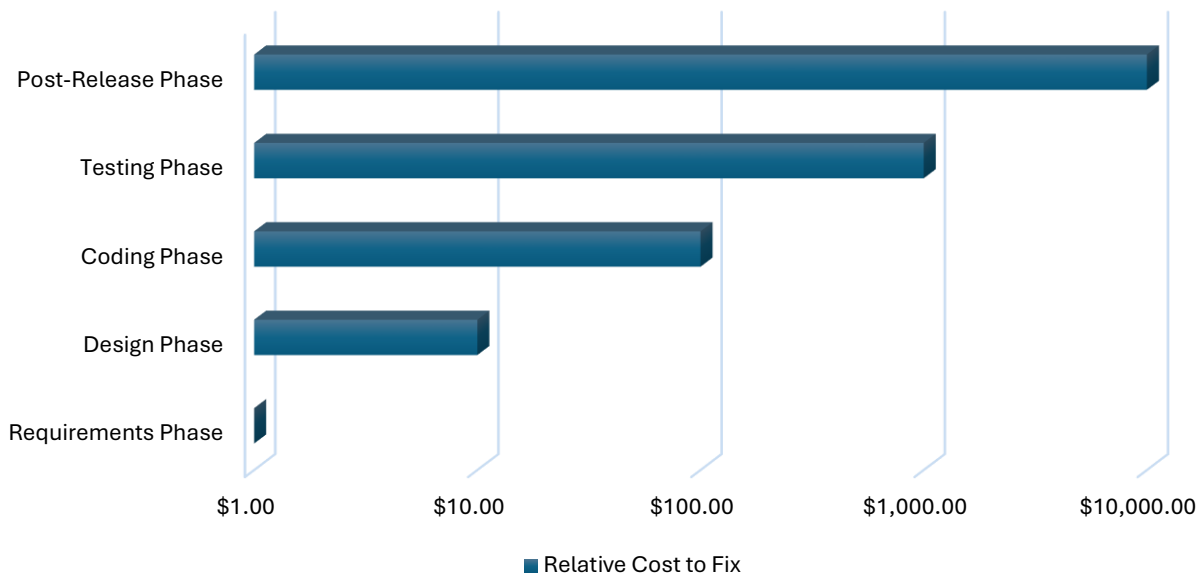


Figure 2. The Exponential Cost of Late Defect Detection

Architectural and Governance Frameworks for Intelligent Systems

Building a comprehensive AI pipeline requires a synthesis of modern architectural patterns and robust governance frameworks. Event-Driven Architecture (EDA), particularly with platforms like Apache Kafka, provides a scalable and resilient backbone for orchestrating complex, asynchronous workflows. By decoupling components and having them communicate via an immutable event log, EDA is ideal for managing the flow of tasks between different AI agents (Kulkarni, 2021). This architecture ensures that if one agent fails, the entire process does not halt, and tasks can be reprocessed. The MAS paradigm, where autonomous, specialized agents collaborate to achieve a common goal, provides a powerful model for structuring the AI pipeline (Maldonado et al., 2024). By assigning specific roles (Elicitation Agent, Validation Agent, Specification Agent), the system becomes modular, scalable, and easier to manage. State management frameworks like LangGraph have emerged as key technologies for orchestrating these complex, stateful agentic workflows (Lei, 2024). A critical challenge for using LLMs in an enterprise context is their lack of access to proprietary, confidential, or up-to-the-minute information. The RAG architecture solves this by connecting the LLM to an external knowledge base (a vector database) (Karakurt & Akbulut, 2025). This grounds the model's outputs in verifiable facts, ensures traceability, and allows for the enforcement of security controls like Role-Based Access Control (RBAC) at the retrieval layer. The adoption of AI introduces new risks that require a proactive governance model. The literature highlights the need for a layered defense against AI-specific attacks like prompt injection and data poisoning (Abtahi, Seoane, & Vega-Barbas, 2026). Furthermore, the need for transparency and auditability has led to the concept of an AI Bill of Materials (AIBOM), which provides a detailed inventory of all models, data, and dependencies (Radanliev et al., 2026). Finally, the recognition that AI models are not infallible has cemented the importance of Human-in-the-Loop (HITL) systems, where human experts are embedded in the workflow to validate critical decisions and provide feedback for continuous model improvement through techniques like Reinforcement Learning from Human Feedback (RLHF) (Kallmes et al., 2025; Retzlaff et al., 2024).

Methodology

To address the identified research gaps, this study employs a DSR methodology. DSR is the most appropriate paradigm as the primary goal is not merely to describe or explain a phenomenon, but to conceptually design and evaluate a novel artifact, the AI-Enabled Requirements Engineering Pipeline, to solve a real-world business problem. The research was conducted in three distinct phases, following the established DSR process model.

Phase 1: Problem Identification and Motivation

This initial phase involved a systematic review of peer-reviewed academic literature and prominent industry reports (e.g., from IEEE, ACM, PMI, and CISQ) to define the problem domain. The objective was to identify and quantify the primary economic and operational failures of traditional software 1.0 requirements engineering, thereby establishing motivation and objectives for the design of a new solution. The findings of this phase are detailed in the Problem Statement and Review of Literature sections.

Phase 2: Design and Development of the Artifact

This phase constitutes the core of the research effort. The primary artifact conceptually designed is the AI-Enabled Requirements Engineering Pipeline architecture. The design process involved synthesizing architectural patterns. By integrating established patterns, including an EDA, a MAS for distributed task execution, and a RAG core for grounded, context-aware AI reasoning, the conceptual system is projected to be robust enough to handle system intricacies.

Next, it was important to define agent roles and interactions. The pipeline is implemented through a set of specialized agents within a MAS, each responsible for a discrete stage of the requirements lifecycle. The Ingestion Agent processes unstructured inputs and retrieves context via the RAG layer, producing structured task artifacts. The Specification Agent transforms these inputs into formal requirements artifacts. The Critic Agent applies NLI to detect ambiguity, inconsistency, and factual errors, while the Validation Agent assesses linguistic quality, completeness, and policy compliance. The Traceability Agent constructs a real-time dependency graph linking requirements to source data and downstream artifacts. By specifying the roles and responsibilities of specialized AI and modeling their asynchronous communication patterns via the event bus; there can be a divide in responsibility of the agentic helpers to ensure work is distributed.

The Governance Agent enforces security policies, including role-based access control and auditability through AIBOM generation. Engineering a multi-layered security model, including a RAG Security Module to defend against data poisoning and prompt injections, and a comprehensive Governance Framework that defines agent autonomy levels and mandates an AIBOM was critical as AI transitions from dialogue intelligence to decision intelligence, its ability to act independently introduces systemic risks that exceed traditional software vulnerabilities. Finally, the focus was on formalizing the Architecture. Representing the final architecture using a formal modeling language, specifically a SysML Block Definition Diagram (Figure 4), provided a clear and unambiguous technical blueprint that can be tangibly built given the appropriate resources.

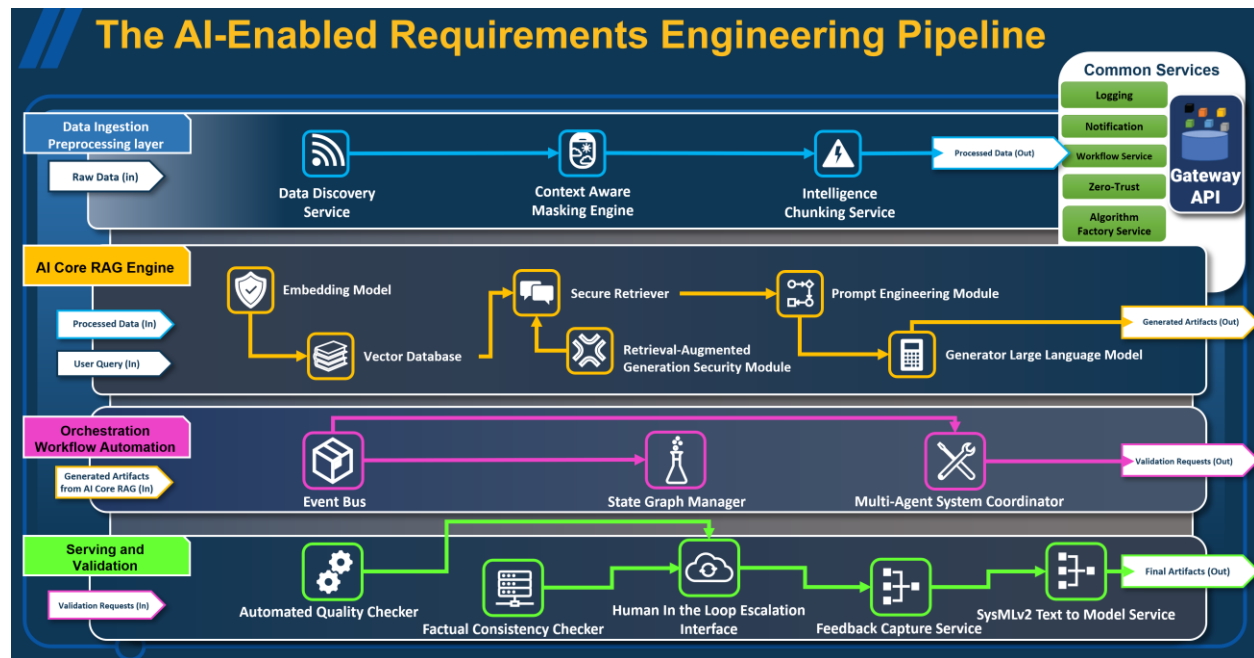


Figure 3. The AI-Enabled Requirements Engineering Pipeline - A Layered Architecture

Phase 3: Demonstration and Evaluation

The utility of the designed artifact is illustrated through a detailed conceptual walkthrough and evaluated against the problems identified in Phase 1. The illustration via case study was used to detail the end-to-end workflow of a requirement moving through the pipeline, to illustrate the artifact's functionality. This includes showing how a task file is ingested, how a user story is created in Jira, how it is automatically validated for factual and consistency quality, and how it is linked to CI/CD processes. The evaluation is conducted by mapping the features of the designed architecture directly to the research questions and the identified problems. The proposed pipeline's components (e.g., automated validation, secure data ingestion, event-driven orchestration) were analyzed to determine how to potentially mitigate the documented challenges of cost, quality, and security in traditional RE processes.

The proposed AI-Enabled Requirements Engineering Pipeline is evaluated using a DSR-aligned approach that combines scenario-based illustration, analytical comparison, and a metric-driven evaluation framework. A representative end-to-end workflow illustrates how unstructured inputs are transformed into validated, traceable artifacts through the integration of a RAG core, MAS, and EDA, including validation via NLI-based critic agents and HITL escalation. The architecture is analytically compared to traditional software 1.0 requirements engineering processes, highlighting improvements in defect mitigation, traceability, and process efficiency. To support systematic assessment, a set of measurable criteria is defined across four dimensions: requirements quality (e.g., ambiguity reduction, consistency, completeness), traceability (e.g., trace link density, propagation latency), process efficiency (e.g., cycle time reduction, rework proxies), and governance (e.g., access control compliance, auditability, adversarial resilience). While this evaluation is conceptual and does not include empirical validation, it establishes a structured foundation for future quantitative assessment and illustrates the architecture's potential to improve the quality, scalability, and governance of requirements engineering in complex system environments.

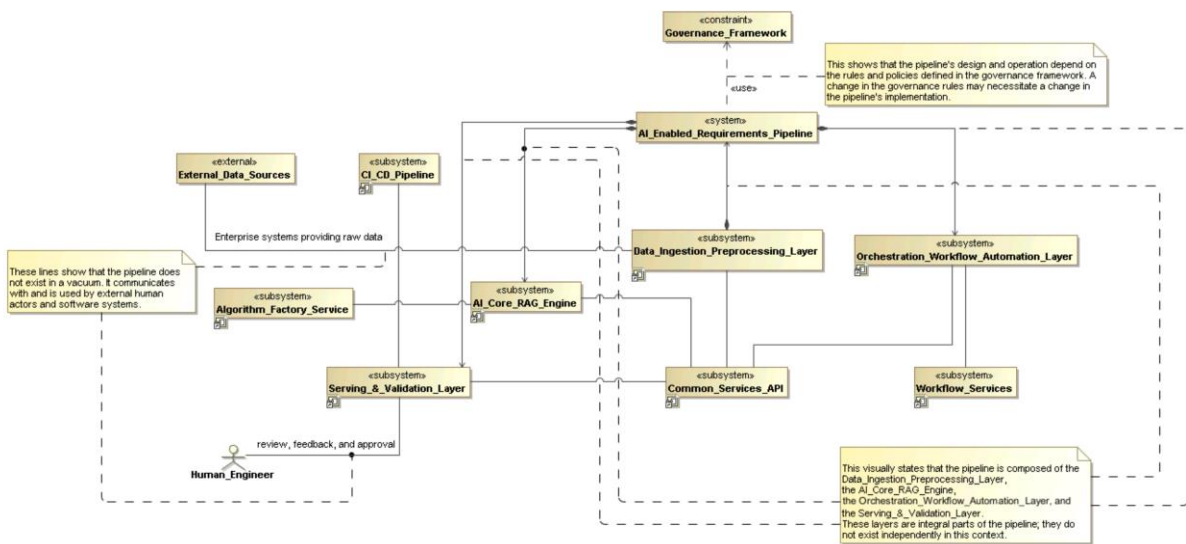


Figure 4. Formal System Blueprint – AI Enabled Requirements Pipeline SysML BDD

Real-world case study example

The automated workflow initiates when a systems engineer uploads a raw task file, such as a transcript from a stakeholder workshop. This upload acts as an event, triggering the EDA. The Governance Agent intercepts this event to enforce RBAC and immediately sanitizes the inputs against adversarial threats, such as prompt injections designed to manipulate the underlying language models. Once secured, the Specification Agent synthesizes this unstructured stakeholder data with historical constraints and zero-trust cybersecurity policies retrieved via the RAG core. RAG is critical here, as it grounds the generative AI's output in an enterprise's verified external knowledge base, effectively mitigating the risk of model hallucinations (Karakurt & Akbulut, 2025).

Leveraging this grounded context, the Specification Agent automatically drafts and populates a formal (agile) user story directly into Jira. By writing directly to Jira, the AI pipeline bridges the gap between conceptual requirements gathering and actionable development tasks.

Crucially, to prevent defect propagation before coding begins, the pipeline employs automated validation. A Critic Agent utilizes NLI to determine whether a hypothesis (the drafted requirement) is true, false, or undetermined based on a premise (the RAG-retrieved policies) (Fazelnia et al, 2024). In this scenario, the Critic Agent successfully identifies a factual contradiction between the stakeholder's requested "offline syncing feature" and the enterprise's retrieved "continuous authentication" policies. Because resolving this fundamental architectural conflict exceeds the AI's autonomous authority, the automated detection halts the pipeline's progression. It triggers a HITL escalation, flagging the Jira ticket for a senior architect to resolve the conflict safely. Following the architect's resolution, the Traceability Agent updates a real-time dependency graph. This dynamically links the finalized, Jira-housed requirement forward to downstream CI/CD test scripts, and backward to the original stakeholder transcript. Finally, an AIBOM is generated to document the specific models, prompts, and datasets utilized, ensuring complete auditability.

If the initial input is an unstructured raw task file, the pipeline's ultimate output is a secure, developer-ready Jira ticket mapped directly to automated testing scripts with complete lifecycle traceability; furthermore, by leveraging its EDA, the system establishes a foundation for continuous integration and real-time adaptation, enabling an autonomous loop that continuously monitors external data sources to automatically generate new, context-aware development tasks.

Table 1. Expected Benefits and Future Evaluation Criteria

Evaluation Dimension	Traditional Challenge	Process	Expected Benefit and Future Evaluation Metric
Requirements Quality	High ambiguity, manual factual checking leading to high defect propagation		Expected Benefit: Reduction in architectural rework. Future Metric: Percentage decrease in defect leakage to the CI/CD pipeline via NLI validation logs.
Traceability	Static matrices fail to capture dynamic links; poor trace link density		Expected Benefit: Minimized propagation latency. Future Metric: Time required to conduct impact analysis on a changed RAG policy using the generated real-time graph.
Process Efficiency	Manual interpretation of transcripts leads to long cycle times and high proxy rework costs		Expected Benefit: Drastic reduction in initial requirements drafting time. Future Metric: Cycle time reduction from raw task file ingestion to Jira user story creation.
Governance and Security	Inconsistent access controls, lack of transparency in how decisions were made.		Expected Benefit: Complete auditability of AI-generated tasks. Future Metric: Number of successful AIBOM generations and prevented unauthorized RBAC actions during simulated red-team tests.

Results

The primary result of this research is the design of the AI-Enabled Requirements Engineering Pipeline, a novel, governed, and agentic architecture. This section details the key components of this artifact and conceptually evaluates its utility against the research questions.

Architectural Blueprint and Interaction Model

The resulting architecture is a holistic, multi-layered system that successfully integrates a MAS, an EDA, and a RAG core. The system is designed to be modular, scalable, and secure, providing a reusable blueprint for next-generation requirements engineering. The results illustrate an expected cohesive workflow orchestrated by the Multi-Agent System Coordinator and the State Graph Manager via the Kafka Event Bus. A Specification Agent, triggered by a “TaskFileReady” event, successfully creates a structured user story in Jira and publishes a “user story created” event. This, in turn, triggers the Critic Agent and Validation Agent to work in parallel. The Critic Agent uses NLI to verify factual consistency, while the Validation Agent integrates with external APIs to perform a multi-faceted quality assessment. The entire process is stateful and resilient, with a HITL escalation path for handling exceptions and a Feedback Capture Service for continuous model improvement.

The Integrated Governance and Security Framework

A key result of this research is the design of a proactive governance and security framework that is woven into the architecture, not bolted on as an afterthought. The governance framework successfully illustrates a model for enforcing data security policies through automated, context-aware masking and RBAC at the retrieval layer. It provides a structure for defining agent autonomy levels and mandates the creation of an AIBOM for auditability. The RAG Security Module provides a concrete, multi-layered defense against AI-specific threats. The design includes specific mechanisms for mitigating data poisoning and prompt injection (e.g., through prompt hardening and output validation).

Discussion of Findings

The results of this design science study present a comprehensive architectural framework, the AI-Enabled Requirements Engineering Pipeline that addresses the research questions posed and offers significant implications for the theory and practice of systems engineering. This section discusses the interpretation of these findings, their relationship to the existing body of literature, and their broader theoretical and practical significance. The primary finding of this study is the successful design and articulation of the AI-Enabled Requirements Engineering Pipeline, a holistic and governed architecture that directly answers RQ1 (How can a comprehensive pipeline be architected?). The research illustrates that it is architecturally feasible to synthesize several advanced paradigms such as MAS, EDA, RAG, and proactive governance into a single, cohesive framework. The detailed specification of the pipeline's layers, the roles of its specialized agents, the event-driven communication patterns, and the integrated security modules provides a novel and implementable blueprint that moves beyond the isolated, tool-based solutions prevalent in current literature.

In addressing RQ2 (How does the proposed AI-enabled requirements engineering pipeline mitigate key economic and operational challenges?), the findings indicate that the proposed architecture provides a robust, multi-faceted mitigation for the primary economic and operational failures of Software 1.0 requirements engineering. The automation of data ingestion and analysis by the MAS directly targets the manual labor bottleneck, which is a primary driver of cost and delay. The Automated Quality Checker and Factual Consistency Checker (critic agent) are designed to systematically detect and flag ambiguity, inconsistency, and factual errors at the point of creation, directly addressing the root cause of the costly rework that plagues traditional projects. Furthermore, the RAG architecture, by grounding the LLM in a secure, enterprise-specific knowledge base, provides a powerful mechanism for mitigating model hallucinations and ensuring that generated artifacts are contextually relevant and traceable. Finally, the Traceability Agent, operating on the immutable event stream, solves the critical problem of poor traceability in agile environments by creating an automated, real-time dependency graph.

The novelty of this framework, when viewed against existing literature, lies not in the invention of any single component, but in its holistic synthesis. While prior research has illustrated the utility of AI for discrete RE tasks (e.g., requirements classification, test case generation), it has largely failed to present an integrated, end-to-end system. This study extends the theoretical concept of software 2.0 by providing a concrete architectural instantiation for the requirements engineering domain. It also resolves a key tension in modern software development by illustrating an architecture that can achieve both the speed and adaptability of agile methods and the rigorous governance and traceability required for mission-critical systems.

Table 2. Expected Comparison Results of Software 1.0 vs. the AI-Enabled Software 2.0 RE Pipeline

Metric Aspect /	Traditional Software 1.0 Process	AI-Enabled Software 2.0 Pipeline
Data Source & Elicitation	Manual interviews, workshops, and document analysis. Labor-intensive and prone to missing insights from large data volumes.	Automated ingestion from diverse, siloed sources (Jira, user feedback, docs). Uses NLP to extract initial requirements and themes.
Analysis & Insight Generation	Manual categorization and prioritization. Susceptible to human bias and subjective interpretation.	AI-driven sentiment analysis, topic modeling, and quality scoring. Provides objective, data-driven insights for prioritization.
Defect Detection & Quality Assurance	Reactive. Relies on manual reviews and inspections. Defects are often found late in the lifecycle, leading to exponential cost increases.	Proactive. Automated critic agents use Natural Language Inference (NLI) for factual consistency checks and integrate with tools like QVscribe for linguistic quality analysis at the point of creation.

Metric Aspect /	Traditional Software 1.0 Process	AI-Enabled Software 2.0 Pipeline
Traceability & Impact Analysis	Manual creation and maintenance of Requirements Traceability Matrices (RTMs), typically in spreadsheets. Brittle, error-prone, and quickly outdated.	Automated, real-time generation of a traceability graph. An event-driven architecture captures every action, enabling instant and comprehensive impact analysis.
Security & Governance	Ad-hoc and procedural. Relies on manual compliance checks and human enforcement of policies.	Governed-by-Design. A machine-readable Constitution enforces rules. Automated data masking, RBAC at the retrieval layer, and proactive defenses against AI-specific attacks (data poisoning, prompt injection).
Process Speed & Agility	Slow, linear, and waterfall-like. High cost of change, making it difficult to adapt to evolving stakeholder needs.	Fast, iterative, and agile. Lowers the cost of change by enabling rapid generation and validation of artifacts. Supports continuous feedback loops.
Role of the Human Engineer	Manual Laborer. Spends significant time on low-value tasks like documentation, data entry, and manual verification.	Strategic Supervisor & AI Teacher. Focuses on high-value tasks: defining intent, validating critical AI outputs, handling exceptions, and improving the system by providing feedback.
Primary Artifacts	Static, text-heavy documents (e.g., SRS). It is difficult to keep synchronized with design and code.	Dynamic, validated, and machine-readable artifacts (e.g., structured JSON, SysML models). A single source of truth that can drive downstream automation.

The practical and theoretical implications of these findings can prove to be profound. For practitioners, this architecture offers a roadmap for transforming RE from a high-risk cost center into a data-driven, value-generating asset. It necessitates a significant organizational shift, redefining the role of the engineer from a manual documenter to a strategic AI teacher and system supervisor. The modular, agent-based design allows for controlled experimentation on the effectiveness of different AI models, governance strategies, and human-computer interaction patterns. However, it is crucial to interpret these findings within the context of the study's methodology. The analytical evaluation presented is based on the architectural design and its alignment with documented problems. This conceptual validation sets the stage for empirical investigation, the absence of which constitutes a primary limitation of the current study.

Limitations of the Study

While this paper presents a comprehensive and theoretically grounded architectural framework, it is essential to acknowledge the limitations inherent in its current scope and methodology. These limitations provide the context for interpreting the results and serve as a foundation for future empirical investigation. The primary contribution of this research is a detailed architectural blueprint, which has been evaluated through a conceptual walkthrough and alignment with documented industry challenges. The study does not include a full-scale implementation or empirical performance benchmarking of the proposed pipeline. As such, the quantifiable benefits cited (e.g., reductions in rework, accelerated timelines) are based on a synthesis of existing literature and serve as theoretical targets rather than measured outcomes of this specific architecture. The practical challenges of integrating the various complex components, from the event-driven backbone to the multi-agent system and the RAG core, are not fully explored in real-world deployment.

The proposed architecture is designed as a reusable pattern, but its effectiveness has not been validated across a diverse range of organizational contexts, scales, or industries. The specific performance and economic benefits of the pipeline will likely vary depending on an organization's existing technical maturity, data governance practices, and cultural readiness for adopting AI-driven automation. The framework's utility in highly regulated versus more agile commercial environments remains an open

question for empirical study. This paper's primary focus is on the technical design of the pipeline and its governance framework. While it establishes the critical role of the HITL, it does not delve deeply into the significant human-computer interaction (HCI), cognitive engineering, and organizational change management challenges. The complexities of designing effective user interfaces for the user review console, managing the potential for skill atrophy in engineers, and fostering trust between human experts and autonomous agents are critical areas that are only addressed at a high level. Finally, the proposed architecture relies on the successful integration of several advanced technologies, some of which are still rapidly evolving. This includes the availability of robust, scalable agent orchestration frameworks (such as LangGraph), highly accurate NLI models for factual consistency checking, and mature security solutions for defending against novel AI-specific attacks. The framework assumes a high level of maturity in these underlying components.

Recommendations for Future Research

The limitations of the current study give rise to several critical and promising avenues for future research. The most immediate and necessary next step is the development of a prototype of the AI-Enabled Requirements Engineering Pipeline. Future research should focus on implementing this prototype in a controlled environment and conducting rigorous empirical studies to measure its performance against traditional baselines. Key Performance Indicators (KPIs) should include:

- Reduction in time spent on requirements elicitation and analysis.
- Decrease in the rate of requirements-related defects (e.g., ambiguity, inconsistency) identified in later stages.
- Cycle time from a new requirement request to a validated artifact.

Conclusion

The traditional Software 1.0 requirements engineering process is a primary source of defects and a major bottleneck to innovation. This paper proposes a transformative solution: a governed AI-Enabled Requirements Engineering Pipeline built for the software 2.0 era, where the focus shifts from writing code to curating data and orchestrating autonomous agents. By leveraging a reusable architecture centered on a secure RAG engine, organizations can automate the requirements lifecycle from data-driven discovery to the generation of traceable requirements, tests, and code. The core of this architecture is a robust RAG system that grounds LLMs in an enterprise's specific knowledge base, protected by defenses against data poisoning and prompt injections. This allows autonomous AI agents, orchestrated through advanced patterns like Multi-Agent Systems and Event-Driven Architectures, to perform complex tasks while being guided by human experts. This human-AI partnership is formalized through a comprehensive governance framework that manages risks across data security, model bias, agent behavior, and intellectual property. It defines clear roles, levels of autonomy, and structured HITL escalation pathways to ensure AI augments, rather than replaces, human intelligence. To build trust and ensure accuracy, the integration of Explainable AI (XAI) techniques and rigorous validation are non-negotiable (Necula et al.,2024).

References

- Abtahi, F., Seoane, F., & Vega-Barbas, M. (2026, Jan 23). Data Poisoning Vulnerabilities Across Healthcare AI Architectures: A Security Threat Analysis. *Journal of Medical Internet Research*, 28. doi:<https://doi.org/10.2196/87969>
- Ahmad, K., Abdelrazek, M., Arora, C., Bano, M., & Grundy, J. (2023). Requirements Engineering for Artificial Intelligence Systems: A Systematic Mapping Study. *Information and Software Technology*, 158(107176), 1-45. Retrieved from <https://doi.org/10.1016/j.infsof.2023.107176>
- Alessio, F., Gori, G., Rosadini, B., Trotta, I., Bacherini, S., Fantechi, A., & Gnesi, S. (2018, Feb 20). Detecting requirements defects with NLP patterns: an industrial experience in the railway domain. *Empirical Software Engineering*, 23, 3684-3733. doi:<https://doi.org/10.1007/s10664-018-9596-7>
- Arora, C., Grundy, J., & Abdelrazek, M. (2024). Advancing Requirements Engineering through Generative AI: Assessing the Role of LLMs. *Generative AI for Effective Software Development*, 158, 129-148. Retrieved from <https://arxiv.org/pdf/2212.10693>
- Brooks, J. (2025). Leveraging ai-driven requirements for sysml modeling of the iobt: A Comprehensive Investigation. *Issues in Information Systems*, 26(1), 46. doi:10.48009/1_iis_2025_105
- Carbin, M. (2019). Overparameterization: A connection between software 1.0 and software 2.0. *Leibniz International Proceedings in Informatics (LIPIcs). 3rd Summit on Advances in Programming Languages (SNAPL 2019)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:<https://doi.org/10.4230/LIPIcs.SNAPL.2019.1>
- Chakraborty, A., Baowaly, M. K., Arefin, A., & Bahar, A. N. (2012, May). The Role of Requirement Engineering in Software Development Life Cycle. *Journal of Emerging Trends in Computing and Information Sciences*, 3(5), 723-729. Retrieved from https://www.academia.edu/download/46873830/The_Role_of_Requirement_Engineering_in_S20160628-6755-11nb1pt.pdf
- Chari, K., & Agrawal, M. (2018). Impact of incorrect and new requirements on waterfall software project outcomes. *Empirical Software Engineering*, 23, 165-185. Retrieved from 10.1007/s10664-017-9506-4
- Cheng, H., Husen, J. H., Lu, Y., Racharak, T., Yoshioka, N., Ubayashi, N., & Washizaki, H. (2026). Generative AI for Requirements Engineering: A Systematic Literature Review. *Software: Practice and Experience*, 56, 141-170. doi:<https://doi.org/10.1002/spe.70029>
- Consortium for Information And Software Quality. (2022). *The Cost of Poor Software Quality in the US: A 2022 Report: From Problem to Solutions*. Austin: Consortium for Information And Software Quality. Retrieved from <https://www.it-cisq.org/wp-content/uploads/sites/6/2022/11/CPSQ-Report-Nov-22-2.pdf>

- Delligatti, L. (2014). *SysML Distilled: A Brief Guide to The Systems Modeling Language*. Upper Saddle River, NJ: Addison-Wesley.
- Dilhara, M., Ketkar, A., & Dig, D. (2021, July 23). Understanding Software-2.0: A Study of Machine Learning. *Transactions on Software Engineering and Methodology (TOSEM)*, 30(4), 1-42.
doi:<https://doi.org/10.1145/3453478>
- Fazelnia, M., Koscinsk, V., Herzog, S., & Mirakhorli, M. (2024). Lessons from the Use of Natural LanguageInference (NLI) in Requirements Engineering Tasks. *2024 IEEE 32nd International Requirements Engineering Conference* (pp. 103-115). IEEE.
doi:<https://doi.org/10.1109/RE59067.2024.00020>
- Helwa, H., & Taweel, A. (2026). FORESIGHT: Software Defects Prediction from Requirements Change Requests Using Machine Learning Methods. *Systems*, 14(4), 342.
doi:<https://doi.org/10.3390/systems14040342>
- Herwanto, G. B., Quirchmayr, G., & Tjoa, A. M. (2024). Leveraging NLP Techniques for Privacy Requirements Engineering in User Stories. *IEEE Access*, 12.
doi:10.1109/ACCESS.2024.3364533
- Hidellaarachchi, D., Grundy, J., Hoda, R., & Mueller, I. (2025). Understanding the influence of motivation on requirementsengineering-related activities. *Empirical Software Engineer*, 49.
doi:<https://doi.org/10.1007/s10664-025-10725-y>
- Janisar, A. A., Meidan, A., Shafee bin Kalid, K., Sarlan, A., Gilal, A. R., Maiwada, U. D., & Bin Faheem, Z. (2026, March). Enhancing Security Requirements Engineering for IoT Smart Homes: A BERT-based Approach to Automate Completeness Checking. *Scientific African*, 31.
doi:<https://doi.org/10.1016/j.sciaf.2026.e03275>
- Kallmes, K. M., Thurnham, J., Sauca, M., Tarchand, R., Kallmes, K. R., & Holub, K. J. (2025, October 25). Human-in-the-Loop Artificial Intelligence System for Systematic Literature Review: Methods and Validations for the AutoLit Review Software. 13.
doi:<https://doi.org/10.1002/cesm.70059>
- Karakurt, E., & Akbulut, A. (2025). Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs) for Enterprise Knowledge Management and Document Automation: A Systematic Literature Review. *Applied Sciences*, 16(1), 368.
doi:<https://doi.org/10.3390/app16010368>
- Kulkarni, C. (2021, May). Event-Driven Intelligence: Crafting Real-Time Predictive Workflows in the Cloud. *International Journal of Engineering Technology Research & Management (IJETRM)*, 5(5), 7. Retrieved from <https://ijetrm.com/issues/files/Jun-2021-18-1750267880-MAY202121.pdf>

- Lei, Z. (2024). LangGraph vs LangChain: Which Framework Is Best for AI Agents? *MetaVision Journal of Multidisciplinary Studies (MVJMS)*, 1(4), 16-26. Retrieved from <https://sinojournals.com/index.php/mvjms/article/download/10/9>
- Li, Z., Li, Y., Zhang, T., Gao, C., & Ding, Z. (2025, December 24). Towards More Accurate App Review Response Generation with Fine-Grained Filtering. *ACM Transactions on Software Engineering and Methodology*, 34. doi:<https://doi.org/10.1145/3786790>
- Ludwig, S., Danaher*, P. J., Yang, X., Lin, Y.-T., Abedin, E., Grewal, D., & Du, L. (2026). Extracting Consumer Insight from Text: A Large Language Model Approach to Emotion and Evaluation Measurement. *arXiv preprint arXiv:2602.15312*, 67. Retrieved from <file:///C:/Users/106599~1/AppData/Local/Temp/MicrosoftEdgeDownloads/11d74a09-c00a-4789-82b1-32a6fdf772d8/2602.15312v1.pdf>
- Maldonado, D., Cruz, E., Torres, J. A., Cruz, P. J., & Del Pilar Gamboa Benitez, S. (2024, May 29). Multi-Agent Systems: A Survey About Its Components, Framework and Workflow. *IEEE Access*, 12, 26. doi:<https://doi.org/10.1109/ACCESS.2024.3409051>
- Moakhar, A. S., & Jabbarishiviari, M. (2024). *Software 1.0 Strengths for Interpretability and Data Efficiency*. João Pessoa: The second Tiny Papers Track at ICLR 2024. Retrieved from <https://openreview.net/forum?id=gyl8r8ANcd>
- NASA Johnson Space Center. (2004). *Error Cost Escalation Through the Project Life Cycle*. Retrieved from <https://ntrs.nasa.gov/api/citations/20100036670/downloads/20100036670.pdf>
- Nayebi, M., & Maalej, W. (2016). Defect data analysis of requirements over a project lifetime. *Requirements Engineering*, 21(3), 265-282. doi:10.1007/s00766-015-0222-3
- Necula, S.-C., Dumitriu, F., & Greavu-Şerban, V. (2024). A Systematic Literature Review on Using Natural Language Processing in Software Requirements Engineering. *Applications of Artificial Intelligence, Machine Learning, Deep Learning, and Explainable AI (XAI)*, 2055. doi: <https://doi.org/10.3390/electronics13112055>
- O'Dell, C. H., Harvey, V. L., Anderson, M. K., Weimer, W. A., & Carvell, D. W. (2025). *Code Quality Risk Assessment: A Novel Process for Quantifying Structural Code Quality at NASA*. The National Aeronautics and Space Administration (NASA). The National Aeronautics and Space Administration (NASA). Retrieved from https://ntrs.nasa.gov/api/citations/20250009084/downloads/CQRA_TMTemplateAllFourPagesDec.pdf
- Peffer, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), 45-77. doi:<https://doi.org/10.2753/MIS0742-1222240302>

- Radanliev, P., Santos, O., & Maple, C. (2026, Jan 20). Operationalising artificial intelligence bills of materials for verifiable AI provenance and lifecycle assurance. *Frontiers in Computer Science*, 8, 17. doi:<https://doi.org/10.3389/fcomp.2026.1735919>
- Retzlaff, C. O., Das, S., Wayllace, C., Mousavi, P., Afshari, M., Yang, T., . . . Holzinger, A. (2024, Jan 30). Human-in-the-Loop Reinforcement Learning: A Survey and Position on Requirements, Challenges, and Opportunities. *Journal of Artificial Intelligence*, 79, 57. doi:<https://doi.org/10.1613/jair.1.15348>
- Schlapbach, A. (2026, March). *The Convergence of Schema-Guided Dialogue Systems and the Model Context Protocol: A New Paradigm for Agentic Interoperability*. Swiss Federal Railways, SBB IT. doi:<https://doi.org/10.48550/arXiv.2602.18764>
- Sonbol, R., Rebdawi, G., & Ghneim, N. (2022). The Use of NLP-Based Text Representation Techniques to Support Requirement Engineering Tasks: A Systematic Mapping Review. *IEEE Access*, 10, 20. doi:10.1109/ACCESS.2022.3182372
- Xie, Y., Culley, S. J., & Weber, F. (2011, August). Applying Context to Organize Unstructured Information in Aerospace Industry. *DS 68-6: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design*, 6, 15-19. Retrieved from https://www.designsociety.org/download-publication/30649/applying_context_to_organize_unstructured_information_in_aerospace_industry
- Zhang, J., Shi, F., Xu, C., Li, Y., Yin, X., & Ge, M. (2026). CyMapNER: a named entity recognition model for cyberspace surveying and mapping. *Cybersecurity*, 9(26), 26. doi:<https://doi.org/10.1186/s42400-026-00578-3>
- Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., . . . Cui, B. (2025, November 9). Retrieval-Augmented Generation for AI-Generated Content: A Survey. *Data Science and Engineering*, 29. doi:<https://doi.org/10.1007/s41019-025-00335-5>