GKU Journal of Multidisciplinary Research (GKUJMR)
ISSN: 3049 – 298X, Volume – II, Issue – I, April – 2026

A Structured Review of AI-Assisted Requirements Engineering: Capabilities, Limitations, and Research Gaps

[1] **Mr. Jaskirat Singh**,

Email ID: [1] jaskiratsingh4864@gmail.com

Accepted: 10.04.2026 Published: 30.04.2026 Page No. 06 – 16 DOI: 10.5281/zenodo.19882923

Abstract

The software industry considers that Large Language Models (LLMs) will soon automate the core tasks of Requirements Engineering (RE). Current literature proves this assumption is premature. This paper reviews 25 recent studies to document exactly what AI can and cannot do during the software specification phase. Current studies split the engineering workflow into elicitation, analysis, generation, and traceability. The reviewed data suggests that AI is highly effective at basic pattern recognition and drafting initial text. this literature shows that these same tools consistently fail when processing ambiguous stakeholder language, lose project context during extended interactions, and frequently hallucinate invalid system requirements. The evidence we discovered shows that AI currently functions strictly as a supplementary tool in RE, not an autonomous replacement for human engineers. The paper concludes that future research must prioritize human-in-the-loop validation frameworks rather than focusing purely on automation.

Keywords: Requirements Engineering; Artificial Intelligence; Large Language Models; Software Specification; AI Hallucination; Automation.

1. Introduction:

Requirements Engineering (RE) is about the structural and functional specifications for software development. Tech companies-built AI and large language models specifically to automate the tedious parts of software specification. These models are used by engineers today in the software development life cycle (Hou et al, 2024). They're using natural language processing to detect ambiguity and machine learning to get requirements from customers.

Industry predictions aren't measured performance. AI can't read between the lines. Hybrid machine learning approaches will be required to handle this (Izhar et al, 2015). LLMs forget the architecture and invent impossible system requirements after extensive dialogues. The hallucination is always mentioned in the evaluation of LLMs (Hu et al., 2026).

A recent study assesses the performance of AI for elicitation, analysis, generation, and traceability. Elicitation models recognise functional features in stakeholder data. Analysis models categorise non-functional requirements. Structural inconsistencies are detected. Generative algorithms generate initial user stories (often using training data sets (Yamani et al., 2025). Traceability algorithms re-establish links between the code and the requirements.

The review assesses the empirical results and inherent weaknesses of AI-based RE. Large language models (LLMs) are incapable of substituting human engineers (Edwards et al., 2025). The importance of hybrid intelligence to maintain human control is stressed (Sterling & Oliveira, 2026). The following sections quantify the errors of the AI specification, and the factors that need to be verified by a human-in-the-loop. Research Objectives and Questions:

Current industry expectations regarding automated software specification often ignore operational realities. This review separates proven

utility from theoretical hype. We want to find out what these algorithms get right in the field. The review also maps exactly how the tools fail and points out the massive gaps still left in current software specification research.

The analysis targets three core questions:

RQ1: What specific tasks can AI and LLMs reliably execute across Elicitation, Analysis, Generation, and Traceability?

RQ2: Under what exact conditions do these automated tools fail, lose project context, or hallucinate system requirements?

RQ3: Which validation frameworks are currently missing to support practical human-AI collaboration in the field?

2. Methodology:

This review follows a simplified PRISMA approach for literature selection.

2.1. Search Strategy The search was conducted across Google Scholar and IEEE Xplore. The search string: ("Requirements Engineering" OR "Software Specification") AND ("Artificial Intelligence" OR "Large Language Models" OR "Machine Learning").

2.2. Our search focuses on papers from 2019-2026 that address the use of AI for requirements. We have excluded non-peer-reviewed preprints. we have not included the studies that were focused on other phases of software engineering, such as automatic testing or code generation.

2.3. Selection Process The initial database search found 58 potential papers; only English based literature was considered. Screening the titles and abstracts removed 21 irrelevant articles. We then read the full text of the remaining 37 papers. Twelve were dropped because they lacked clear empirical data or reused older datasets. The final review is based on the 25 selected papers. This number was considered sufficient to capture representative trends across AI-assisted RE applications.

2.4. Paper Classification Scheme The papers were classified according to the major contribution made to Requirements Engineering. The categories were: (1) Elicitation - studies focused on extracting requirements from stakeholders; (2) Analysis - studies that deal with requirement classification, ambiguity detection or validation; (3) Generation - studies using AI to generate requirement artifacts; and (4) Traceability - studies linking requirements through development processes. All papers were classified according to their main methodological contribution.

2.5. Data Extraction For each selected study, specific attributes were extracted. We tracked the publication year, the specific RE target area, the exact AI model used, the core contribution, and how researchers tested the tools. These attributes populated the comparative analysis table to identify patterns across the literature.

2.6. Limitations of Methodology The review is limited to selected databases and may not capture all emerging industry work or unpublished studies

Table 1 maps the technical landscape of some studies analysed in this review. We broke down each paper by its specific RE task and the type of AI algorithm deployed to solve it. This summary provides a high-level view of how researchers currently test and validate automated specification tools.

Table 1: Literature Selection Matrix

Paper	Year	Area	AI Technique	Key Contribution	Evaluation Type
Chelieger et al.	2022	Elicitation	Machine Learning	Maps ML tasks for automated requirement extraction.	Conceptual

Paper	Year	Area	AI Technique	Key Contribution	Evaluation Type
Izhar et al.	2025	Analysis	Machine Learning	Tests a hybrid model for ambiguity detection.	Experimental
Yamani et al.	2025	Generation	LLMs	Creates the UStAI dataset for drafting user stories.	Empirical
Hou et al.	2024	Generation	LLMs	Tracks LLM hallucination rates in software engineering.	Empirical
Alturayeif et al.	2025	Traceability	Machine Learning	Reviews automated algorithms for traceability link recovery.	Conceptual
Sterling & Oliveira	2026	Generation	Generative AI	Argues for human-in-the-loop models to stop metacognitive offloading.	Conceptual
Luitel et al.	2024	Analysis	LLMs	Tests LLMs on finding missing requirements in specifications.	Experimental

3. Background And Related Work:

The most recent literature breaks down the use of AI in Requirements Engineering into four operational areas. Figure 1 shows this operationalization. This categorization draws out discrete machine learning abilities from the more general software development processes, and sets the stage for today's research

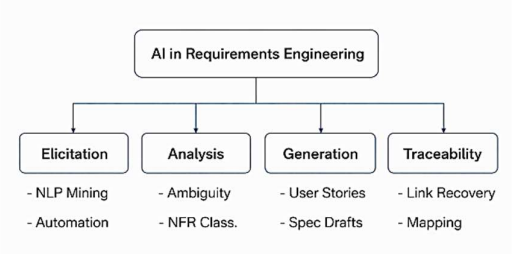

Figure 1: AI in Requirements Engineering Framework

3.1. AI for Elicitation:

Software teams start the baseline by directly extracting the constraints from stakeholders. This raw interview data is now analysed using algorithms instead of text mining. Using machine learning, algorithms search qualitative user feedback for system features independent of human interpretation (Cheligeer et al., 2022). This reduces the time in the initial data collection phase. Engineers employ such systems to convert thousands of unstructured survey responses into structured data quickly than qualitative coding (Siddeshwar et al., 2024). These speeds up time in the "time to baseline" phase. However, these systems don't record implicit variables in interactive sessions (Ahmad et al., 2024). Automation consistently fails to capture implicit cultural restrictions and other non-verbal cues that engineers instinctively note. Models fail to pick up on the intricate power dynamics of stakeholder conversations (Rosado da Cruz & Cruz, 2025). The algorithmically generated output needs significant post-processing. The algorithmically suggested lists need to be validated by domain experts, to ensure the final software specification contains important, un-documented features.

3.2. AI for Analysis:

Analysis is the time to detect structural problems. Natural language processing models audit text dumps for non-functional requirements classification and identification of natural language ambiguity. Hybrid machine learning systems are specifically designed to verify the clarity of human phrasing of millions of specifications (Izhar et al., 2025). The industry follows up automated reviews to identify conflicting constraints that human eyes can easily overlook. Text processing with algorithms reduces the time required for manual reviews (Necula et al., 2024). The new models can also scan the system architecture to ensure it's complete. Large language algorithms scan tech-ical documents to identify missing security and logical processes (Luitel et al., 2024). Specialised techniques convert natural language constraints to

formal models in an agile setting (Umar et al., 2025). This highlighted detection avoids costly architectural redesigns. The models detect variations from industry best practices im-mediately. Despite this, human engineers still must check the anom-aly to ensure it's a fault and not a deliberate design decision required by the client.

3.3. AI for Generation:

Generation phase expands the effort from text analysis to generating technical documents. The algorithms take the notes of the stakeholders and attempt to generate artifact such as a user story or system design. To enable this, special da-tasets are being prepared to train neural networks in just writing requirements (Yamani et al., 2025). This leveraging of software artifact generation is a huge trend in software engineering today (Cheng et al., 2026). The networks essentially convert well-known patterns into new application outlines. This allows engineering teams to produce standard documents on-the-fly and create early prototypes without having to wait for drafts. But the sheer velocity of this stage poses huge operational risks. Generative models suffer from a lack of domain knowledge, and often invent technical constraints (Norheim et al., 2024). Human engineers produce far better system requirements than models, in direct comparisons (Edwards et al., 2025). Neural networks invent functional dependencies as soon as they leave the protection of their training sets. To keep hallucination rates low, the literature is strongly in favour of hybrid intelligence systems (Sterling & Oliveira, 2026). The main purpose is to avoid developers' naively follow the AI blueprints, showing the generation is unsafe without human validation protocols.

3.4. AI for Traceability:

Software teams lose sight of requirements as software becomes larger. Human link analysis doesn't keep up to agile sprints. Enter the machine learning models, able to regenerate broken links (Alturayeif et al., 2025). They process code commits and tickets to re-establish the traceability. And they don't need any human help. This allows compliance for big projects. It prevents developers from spending hours searching for the function that relates to a client's request. Until that is, the documentation gets out of date. Au-tomated tracing completely fails if the code is changed but new notes are not added. The neural networks need hybrid logic models to be accurate for this (Li et al., 2020). the code changes but the text does not, the AI can't determine what the intent was for the changes (Wang et al., 2019). It simply matches on the text. It will link a database script to a user story because it has the word "login". All these false positives mean high-level engineers must watch the AI. They must clean up the traceability matrix to avoid bad links accumulating.

4. Capabilities of AI in RE:

Software developers spend several weeks interviewing customers and surveying the market to discover what to develop. Machine learning skips this step of reading. Software snares functional requirements directly from the clutter of interview responses without having to label it with a human (Cheligeer et al., 2022). This heavy-lifting makes it possible to process large volumes of stakeholder data that would exhaust a human. Emerging AI models instantly organise these user stories and establish a framework for the project (Siddeshwar et al., 2024). Even when building complex machine learning applications themselves, teams use automated elicitation tools to gather the initial design constraints (Elvira et al., 2024). The main benefit here is raw speed. Developers do not waste their first month of a project just sorting through client emails and meeting notes. The AI organizes the chaos into a structured list of features so the team can start planning the architecture.

Finding logical problems in a massive technical document is incredibly tedious work. Natural language processing tools shine here because they never get tired of reading technical jargon. These systems scan thousands of sentences to find vague words or confusing rules that human reviewers usually miss (Necula et al., 2024). Some specific NLP approaches cross-check requirements across entirely different project domains to spot structural ambiguity before any code gets written (Ferrari & Esuli, 2019). The tools also check if the project blueprint covers everything it needs to. Large language models run automated audits on the specifications to find missing security protocols or forgotten client constraints (Luitel et al., 2024). Recent machine learning models measure exactly how precise a stated requirement is against known industry standards (Izhar et al., 2025). This automated error detection is highly reliable. because of this they can figure out a bad architectural decision early in the planning phase, which saves developers from having to rewrite thousands of lines of code later.

AI does not just read documents; it writes the initial drafts for the engineering team. Large language models take the sorted client data and instantly generate standard user stories and system blueprints (Marques et al., 2024). The industry is heavily investing in this specific generative capability to speed up the software lifecycle (Cheng et al., 2026). Researchers are even building specialized datasets to train these neural networks purely on how to write better software requirements (Yamani et al., 2025). The major benefit of this generation phase is that it completely removes the blank-page problem. Developers do not have to sit down and write formal technical specifications from scratch. The AI spits out a highly structured preliminary document in seconds. Teams then use this generated draft as a starting point. They review and edit the machine-made blueprint instead of writing the whole thing manually, which drastically cuts down the time it takes to move from the planning phase into active development (Hemmat et al., 2025).

Managing and checking which piece of code matches which original client request is a nightmare on large projects. As software scales, human analysts simply cannot maintain these traceability matrices manually (Cuddeback et al., 2010). Deep learning networks solve this by automating the entire tracking process. As coders commit new changes during rapid agile development sprints, machine learning algorithms automatically restore and update the traceability mapping (Alturayeif et al., 2025). Software developers push new code into the system and these systems trace it back to the original software specifications (Wang et al., 2023). This ongoing mapping is an enormous boon. And it ensures the entire project is compliant and structurally aligned without having to halt the coding process every time the project's spreadsheet needs to be updated. The AI handles the administrative burden of tracking software evolution. This lets developers overhaul the architecture quickly while making sure every new feature still lines up with what the client asked for.

Table 2 isolates the specific operational advantages of artificial intelligence across the four primary engineering domains. In the elicitation phase, machine learning completely changes how teams handle raw input by processing massive stakeholder datasets instantly. This high-speed text mining prevents the usual project bottleneck. For the analysis phase, natural language processing acts as an automated, high-speed audit system. It spots vague wording and missing logical rules before any actual coding begins. This early detection directly stops expensive architectural rework. During the

generation phase, large language models solve the blank-page problem. They instantly draft initial requirement documents so technical writers can start editing rather than writing from scratch. Finally, in the traceability domain, algorithms provide dynamic link recovery. They map new code commits back to the original client rules automatically. This continuous background mapping keeps the entire software architecture compliant without forcing developers to stop their active sprint work.

Table 2: Proven AI Capabilities by Domain

Domain	Primary AI Capability	Practical Engineering Benefit
Elicitation	High-speed text mining	Processes huge stakeholder datasets instantly.
Analysis	Automated NLP audits	Spots vague wording and missing rules before coding.
Generation	Instant requirement drafting	Removes the blank-page problem for technical writers.
Traceability	Dynamic link recovery	Maps new code commits to old rules automatically.

5. Limitations and Failures of AI in RE:

Machines are not good with humans. The algorithms which gather requirements don't read between the lines. They can't be in a room and determine from the body language of an angry stakeholder during an interview (Rosado da Cruz & Cruz, 2025). A client can be highly frustrated with the system and not say so, the machine learning algorithm doesn't pick it up. This causes gaps in the project baseline before writing any code. The systems have difficulty with cross-system journeys that involve context switching (Ahmad et al., 2024). The situation is even worse when people try to use AI to capture requirements to develop real AI systems. The algorithms don't pick up on hidden data constraints and model parameters required to build sophisticated AI systems (Villamizar et al., 2021). They also fail to comprehend inter-disciplinary terms. An AI may read the word "current" and think of it in terms of physics, rather than in electrical engineering, leading to immediate structural inconsistency (Ferrari & Esuli, 2019). The software simply doesn't know the context humans know that helps us to understand the confused client. Even when attempting to formalize rules for agile development, the AI can't convert human wishes into model-driven rules unless a developer holds its hand (Umar et al., 2025). Developers spend hours combing through the AI's raw output to weed out such blindingly obvious errors.

The greatest threat occurs during the generation process. Large language models don't know anything about software engineering. They simply guess the next word that might come based on training data (Hou et al., 2024). This guessing results in hallucinated requirements. The models often come up with complicated technical constraints that are not required by the client or the software (Norheim et al., 2024). This is particularly true with general chatbots. When software teams try to use tools such as ChatGPT to generate their software requirements, the resulting requirement docu-ments look great but are shallow (Marques et al., 2024). They will generate a highly structured user story that looks great but is not doable. In controlled experiments, human students can write better and more reliable software requirements than the powerful neural networks (Edwards et al., 2025). The AI simply makes up functional relationships if it veers even slightly from its main training set. Thorough reviews of the existing literature show we still do not have a way to ensure the LLM's models of requirements are correct (Hemmat et al., 2025). Developers can't rely on the generat-ed blueprints

without scrutinising each line of code. Over-relying on the models makes the developers build features based on complete fantasy. This destroys software architecture and requires huge refactoring (Sterling & Oliveira, 2026).

Automated traceability algorithms quickly run into problems when facing undocumented software evolution. The software automation is only feasible if the code and text are very similar. If the developer rewrites a legacy module but uses different variable names, the machine learning is lost (Alturayeif et al., 2025). The neural network connections are merely word matches. The algorithms just use keywords, which do not capture any meaningful information (Wang et al., 2019). This huge blind spot means the tools propose thousands of false positives. They will link a random database script to a random user story because they both contain a word like "login" or “execute". Research on deep learn-ing in software development demonstrates that these algorithms are often used as black boxes, which makes it hard for developers to comprehend why a specific user story should be connected to a specific software code file (Wang et al., 2023). To over-come this issue researchers, need to create hy-brid systems that force the AI to follow logical rules (Li et al., 2020). Prior to this huma-n-designed logic layer, the automated traceability matrix gradually becomes filled with junk data. These software engineers are put on a tedious semi-automated grind where they spend long hours in manual verification of links and removal of false positives from the simple classi-fiers (Etezadi et al., 2025).

Figure 2 illustrates how these individual algorithmic errors snowball into catastrophic failure. It documents the entire engineering process from initial stakeholder interview to the final software. An error doesn't remain localised. For instance, if an elicitation model fails to pick up on a cultural nuance during initial stakeholder interviews, it will pass that along. The analysis tools interpret this bad data and spit out false positives due to misinterpreted cross-disciplinary terminology. Generative models then hallucinate unfeasible technical constraints based on bad analysis. The traceability algo-rithms then embed the hallucinations forever into the architecture by linking arbitrary keywords together. This failure showcases how au-tomated bugs create a snowball effect. Programmers can't simply fix a traceability bug in the last stage. They need to clean the data all the way back at the top of the pipeline to prevent a system collapse.

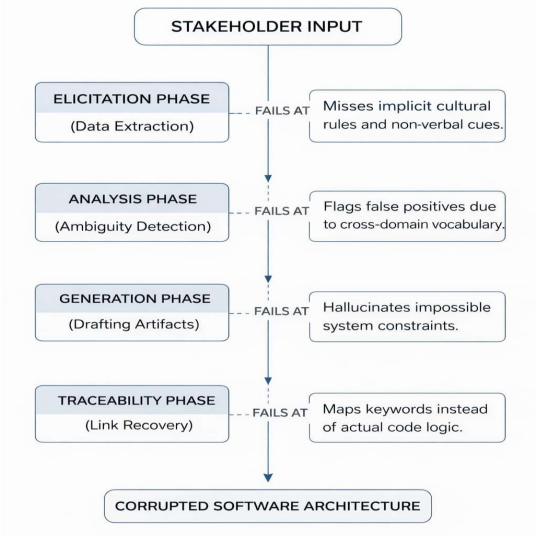

Figure 2: AI Failure Pathways in Requirements Engineering

6. Research Gap and Future Works:

The software industry has absolutely no working frameworks for human-AI collaboration. Studies claims we need a "human in the loop" to stop AI failures. Nobody knows what that means during a live agile sprint. Right now, validation is just a senior engineer reading every single machine-generated word to catch hallucinations. This brute-force manual review kills the entire speed advantage of using automation. The field urgently needs hybrid intelligence protocols so developers do not end up acting as glorified AI spell-checkers (Sterling &

Oliveira, 2026). Future tools must force the AI to flag its own uncertainty. If a generative model highlights the exact technical constraints it guessed at, reviewers can just check those specific lines instead of reading a fifty-page document. Until researchers design these specific workflows, AI remains a dangerous experimental tool instead of a safe engineering standard (Hemmat et al., 2025).

We also lack realistic ways to test these models. Current benchmarks test language models against perfectly clean computer science problems. Real client data is never clean. It is a mess of contradictions and missing variables. Models trained on pristine academic datasets crash completely when given an actual disorganized client transcript. The field needs new benchmarks designed strictly around the chaos of real-world software engineering (Hu et al., 2026). These new tests must track exact hallucination rates. Engineering teams need to know exactly how often an algorithm fakes a functional dependency before they trust it to write a blueprint. In the future studies we should stop treating artificial intelligence as a magic bullet. Researchers must isolate which specific machine learning architecture works for elicitation and which works for traceability. Building domain-specific testing environments is the only way to separate real operational utility from academic hype (Norheim et al., 2024).

7. Conclusion:

Software companies' continuous attempts to automate the whole specification process. The field data shows this just does not work right now (Hou et al., 2024). Machine learning gives teams a huge head starts during early planning. They process messy stakeholder interviews and sort the raw data without any human help (Cheligeer et al., 2022). The algorithms spot confusing wording in giant technical files much faster than a tired developer can (Izhar et al., 2025). Some tools even draft basic frameworks specifically for agile sprints (Umar et al., 2025). This makes AI an excellent tool for sorting text. But the whole system crashes when it tries to grasp actual software architecture. The models lack the real-world context to figure out what a confused client is really asking for (Ahmad et al., 2024). They fail hard at extracting the hidden data rules you need to build complex machine learning applications (Villamizar et al., 2021). The generation tools just invent impossible technical constraints out of nowhere (Norheim et al., 2024). Traceability algorithms map bad links using basic keyword matching instead of reading the real code logic (Alturayeif et al., 2025). Allowing these tools run by themselves can cause the architecture failure before anyone writes a single line of actual code.

Automated tools are not going to replace human engineers. The industry needs to stop chasing full autonomy and focus entirely on hybrid intelligence models (Sterling & Oliveira, 2026). Developers spend way too many hours right now just double-checking AI output. They must manually catch the algorithmic lies and delete all the fake tracking links (Etezadi et al., 2025). Researchers must build workflows where language models flag when they are guessing about a technical artifact (Cheng et al., 2026). The testing benchmarks also need a total overhaul. We need tests built on ugly, real-world client data instead of clean academic computer science problems (Hu et al., 2026). Using generic chatbots to write system rules just makes all these architectural risks worse (Marques et al., 2024). Until researchers fix these specific functional gaps, generative AI is just a dangerous drafting assistant. It is not a reliable engineering standard (Hemmat et al., 2025). Teams should absolutely use AI to speed up data extraction and text audits. But they must force senior engineers to stay in the loop. A human expert must validate every single blueprint to stop

algorithmic mistakes from ruining the logical architecture (Wang et al., 2023).

8. Implications For Future Work:

The software industry needs to completely change how it builds AI tools for requirement engineering. The industry needs to abandon the idea of a single, all-purpose AI system. Future development must focus on creating narrow, highly specialized algorithms. We need one specific tool that only handles data elicitation and a completely different tool that only maps traceability links. Building domain-specific tools is the absolute only way to reduce the massive hallucination rates we see right now. Developers also need completely new interface designs. Future AI dashboards must force the language model to show its statistical confidence for every single sentence it writes. If an algorithm is only fifty percent sure about a system constraint, the text needs to visibly highlight itself on the screen. This leads to the human engineer to review the exact point of failure instead of wasting hours guessing where the machine made a mistake.

Future AI tools also must completely rethink how they handle non-functional requirements. Right now, automated models only look at basic system performance or data security. They completely ignore human psychological barriers. For instance, AI generates a specification for an e-governance platform, it almost never includes constraints for cognitive load or technology anxiety. This algorithmic blind spot causes the digital exclusion of older adults and other vulnerable groups. The models just blindly assume every end-user has perfect digital literacy. Researchers must build specific training datasets that teach language models how to draft psychological constraints. Engineering teams need the AI to automatically flag when a proposed system flow will overwhelm an older user. If we do not force the automation to prioritize these hidden human-centric barriers, the industry will just use machine learning to build highly exclusionary software much faster than before.

University computer science programs also must adapt immediately. Professors are still teaching students how to write software requirements entirely from a blank page. This is becoming a total waste of time. Future curriculum must teach junior engineers how to aggressively audit AI-generated blueprints instead (Sterling & Oliveira, 2026). Students should learn how to spot algorithmic hallucinations and fix broken traceability matrices. On the corporate side, software companies must establish strict new policies for their agile development sprints. Teams cannot legally or functionally rely on a machine learning model to sign off on a system architecture. Future research must define exactly who takes the blame when an AI hallucinates a requirement that causes a massive software crash. Until the industry sets hard legal and operational boundaries, relying blindly on automated specification remains a massive corporate liability (Etezadi et al., 2025).

Declaration of Conflicting Interests: The authors report no personal or financial conflicts of interest related to this study.

References:

[1] Ahmad, K., Arora, C., Abdelrazek, M., Grundy, J., Vasa, R. (2024). Requirements Elicitation in the Age of AI: A Tool's Multi-system Journey. In: Kaindl, H., Mannion, M., Maciaszek, L.A. (eds) Evaluation of Novel Approaches to Software Engineering. ENASE 2023. Communications in Computer and Information Science, vol 2028. Springer, Cham. https://doi.org/10.1007/978-3-031-64182-4_4

[2] Asma Yamani, Malak Baslyman, and Moataz Ahmed. 2025. Leveraging LLMs for User Stories in AI Systems: UStAI Dataset. In Proceedings of the 21st International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE

'25). Association for Computing Machinery, New York, NY, USA, 21–30. https://doi.org/10.1145/3727582.3728689

[3] Cheligeer C, Huang J, Wu G, Bhuiyan N, Xu Y, Zeng Y. Machine learning in requirements elicitation: a literature review. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*. 2022;36:e32. doi:10.1017/S0890060422000166

[4] Edwards S, Nonso-Anyakwo K, Summers J, Adebayo O. Prompt engineering study: comparing pre-service engineers to large language models in requirements generation. *Proceedings of the Design Society*. 2025;5:2661-2670. doi:10.1017/pds.2025.10280

[5] Etezadi, Romina & Abualhaija, Sallam & Arora, Chetan & Briand, Lionel. (2025). Classifier or Prompt: A Case Study on Legal Requirements Traceability. 10.48550/arXiv.2502.04916.

[6] Ferrari, A., Esuli, A. An NLP approach for cross-domain ambiguity detection in requirements engineering. *Autom Softw Eng* **26**, 559–598 (2019). https://doi.org/10.1007/s10515-019-00261-7

[7] H. Villamizar, T. Escovedo and M. Kalinowski, "Requirements Engineering for Machine Learning: A Systematic Mapping Study," 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Palermo, Italy, 2021, pp. 29-36, doi: 10.1109/SEAA53835.2021.00013.

[8] H.Cheng, J. H.Husen, Y.Lu, et al., "Generative AI for Requirements Engineering: A Systematic Literature Review," Software: Practice and Experience56, no. 2 (2026): 141–170, https://doi.org/10.1002/spe.70029.

[9] Hemmat A, Sharbaf M, Kolahdouz-Rahimi S, Lano K and Tehrani SY (2025) Research directions for using LLM in software requirement engineering: a systematic review. Front. Comput. Sci. 7:1519437. doi: 10.3389/fcomp.2025.1519437

[10] Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., ... & Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, *33*(8), 1-79. https://dl.acm.org/doi/full/10.1145/3695988

[11] Li, T., Wang, S., Lillis, D., & Yang, Z. (2020). Combining Machine Learning and Logical Reasoning to Improve Requirements Traceability Recovery. *Applied Sciences*, *10*(20), 7253. https://doi.org/10.3390/app10207253

[12] Luitel, D., Hassani, S. & Sabetzadeh, M. Improving requirements completeness: automated assistance through large language models. *Requirements Eng* **29**, 73–95 (2024). https://doi.org/10.1007/s00766-024-00416-3

[13] Marques, N., Silva, R. R., & Bernardino, J. (2024). Using ChatGPT in Software Requirements Engineering: A Comprehensive Review. *Future Internet*, *16*(6), 180. https://doi.org/10.3390/fi16060180

[14] Necula, S.-C., Dumitriu, F., & Greavu-Șerban, V. (2024). A Systematic Literature Review on Using Natural Language Processing in Software Requirements Engineering. *Electronics*, *13*(11), 2055. https://doi.org/10.3390/electronics13112055

[15] Norheim JJ, Rebentisch E, Xiao D, Draeger L, Kerbrat A, de Weck OL. Challenges in applying large language models to requirements engineering tasks. *Design Science*. 2024;10:e16. doi:10.1017/dsj.2024.8

[16] Nouf Alturayeif, Jameleddine Hassine, Irfan Ahmad,Machine learning approaches for

automated software traceability: A systematic literature review,Journal of Systems and Software,Volume 230,2025,112536,ISSN 0164-1212,https://doi.org/10.1016/j.jss.2025.112536

[17] R. Izhar, S. N. Bhatti and S. A. Alharthi, "Bridging Precision and Complexity: A Novel Machine Learning Approach for Ambiguity Detection in Software Requirements," in IEEE Access, vol. 13, pp. 12014-12031, 2025, doi: 10.1109/ACCESS.2025.3529943.

[18] Rosado da Cruz, A. M., & Cruz, E. F. (2025). Machine Learning Techniques for Requirements Engineering: A Comprehensive Literature Review. *Software*, *4*(3), 14. https://doi.org/10.3390/software4030014

[19] S. Wang et al., "Machine/Deep Learning for Software Engineering: A Systematic Literature Review," in IEEE Transactions on Software Engineering, vol. 49, no. 3, pp. 1188-1231, 1 March 2023, doi: 10.1109/TSE.2022.3173346.

[20] Kumar, R. (2025, May 24). The ethics of innovation: Human-centered design as a pillar of AI futures. Paper presented at One-Day International Multidisciplinary Conference, DRT's A.E. Kalsekar Degree College & IIRA.

[21] S. Wang, T. Li and Z. Yang, "Exploring Semantics of Software Artifacts to Improve Requirements Traceability Recovery: A Hybrid Approach," 2019 26th Asia-Pacific Software Engineering Conference (APSEC), Putrajaya, Malaysia, 2019, pp. 39-46, doi: 10.1109/APSEC48747.2019.00015

[22] Sterling, L., & Oliveira, E. (2026). Hybrid Intelligence in Requirements Education: Preserving Student Agency in Refining User Stories with Generative AI. *Information*, *17*(2), 166. https://doi.org/10.3390/info17020166

[23] T. Elvira, T. T. Procko and O. Ochoa, "Requirements Elicitation for Machine Learning Applications: A Research Preview," 2024 Conference on AI, Science, Engineering, and Technology (AIxSET), Laguna Hills, CA, USA, 2024, pp. 218-221, doi: 10.1109/AIxSET62544.2024.00042.

[24] Kumar, R. (2025, March–April). Transforming student evaluation and feedback through AI-driven automated assessment. International Journal for Multidisciplinary Research (IJFMR), 7(2). https://doi.org/10.36948/ijfmr.2025.v07i02.42212

[25] Umar MA, Lano K and Abubakar AK (2025) Automated requirements engineering framework for agile model-driven development. Front. Comput. Sci. 7:1537100. doi: 10.3389/fcomp.2025.1537100

[26] V. Siddeshwar, S. Alwidian and M. Makrehchi, "A Systematic Review of AI-Enabled Frameworks in Requirements Elicitation," in IEEE Access, vol. 12, pp. 154310-154336, 2024, doi: 10.1109/ACCESS.2024.3475293.

[27] Xing Hu, Feifei Niu, Junkai Chen, Xin Zhou, Junwei Zhang, Junda He, Xin Xia, and David Lo. 2026. Assessing and Advancing Benchmarks for Evaluating Large Language Models in Software Engineering Tasks. ACM Trans. Softw. Eng. Methodol. Just Accepted (December 2026). https://doi.org/10.1145/3786771

[28] Kumar, R. (2025, April). Adaptive cybersecurity with AI: Enhancing threat detection and response in intrusion detection systems. Indian Journal of Modern Research and Reviews. https://doi.org/10.5281/zenodo.15319429