Proceedings of the Design Society, Volume 6: DESIGN 2026
https://doi.org/10.1017/pds.2026.10607

Context-aware large language models for ambiguity detection in requirements

Victor Vilhelm Poulsen[1,✉], Matthias Guertler[1], Boris Eisenbart[2], Laura Tomidei[1] and Nathalie Sick[1]

[1] University of Technology Sydney, Australia, [2] Swinburne University of Technology, Australia

✉ victor.poulsen@uts.edu.au

ABSTRACT: Requirements quality shapes engineering design, yet natural language specifications remain vulnerable to ambiguity. We investigate how LLMs support ambiguity detection using a hybrid dataset combining NASA JWST requirements with systematically injected defects. Using auto-extracted domain knowledge, we compare a domain-agnostic baseline with a context-aware approach. Incorporating domain knowledge helps LLMs better distinguish genuinely ambiguous requirements from acceptable ones, highlighting the potential of context-aware AI assistants for requirements engineering and early-stage design.

KEYWORDS: requirements management, requirements engineering (RE), artificial intelligence (AI), systems engineering (SE), generative AI

1. Introduction

Requirements are foundational design artifacts that shape engineering systems from conception through deployment (Pahl et al., 2007). They define the future product's characteristics, guide development activities and provide the basis for verification of the delivered system (Hull et al., 2005). As most requirements are written in natural language, their quality can vary substantially in practice, which directly impacts downstream design decisions, rework and project risks (Hull et al., 2005; Pahl et al., 2007). Guidance such as ISO 29148 (2018) and the INCOSE Guide to Writing Requirements (Ryan & Wheatcraft, 2023) codifies good practice such as avoiding vague words and subjective qualifiers, expressing each requirement as a single, self-contained statement, using controlled language to reduce ambiguity and improve quality. Yet ambiguity persists, because many cues are context-dependent and can only be judged with knowledge of the project's terminology, units, system architecture and hierarchy specification structure (Berry & Kamsties, 2004). This ambiguity directly impacts the exploration and narrowing of the design space.

Therefore, in practice, effective ambiguity assessment relies on both linguistic cues and situated domain knowledge. To enhance this task, existing research has explored the use of Artificial Intelligence (AI) techniques for automatically detecting ambiguity in datasets of engineering requirements. Existing automated tools and many emerging AI approaches typically treat requirements as isolated sentences and rely heavily on linguistic patterns and "smells". These methods can identify potentially ambiguous requirements but at the cost of substantial false-positive rates, burdening engineers with large volumes of false alarms. Without access to project context, AI adoption becomes a barrier rather than an enabler, and gives a false sense of support which may ultimately erode trust.

In recent years, the rapid emergence of Generative Artificial Intelligence (GenAI) has unlocked new opportunities for augmenting requirements engineering (RE) and design work (Arora et al., 2024), and ultimately addressing some of the limitations of existing methods for ambiguity detection. Large Language Models (LLMs) have enabled advanced analysis of natural language by capturing semantic nuances, syntactic rules, and contextual cues (Zadenoori et al., 2025). As such, they represent a

promising solution for ambiguity detection, which is a particularly critical and demanding task as many ambiguity judgements rely on system context, not just surface level wording (Araújo et al., 2025). While recent work has explored LLM-based ambiguity detection, current approaches operate on individual requirements without systematic access to project context, leading to high false-positive rates on well-written requirements. There is limited empirical evidence on how project-specific context affects detection performance across different ambiguity types. Furthermore, existing datasets rarely combine realistic requirements, explicit ambiguity labels, and domain context in a form that supports controlled reproducible evaluation.

We address these gaps by investigating the following research questions:

RQ1: *How does a domain-agnostic LLM, prompted with standards-based ambiguity triggers, perform when detecting ambiguity in a hybrid dataset of high-quality and defect-injected requirements?*

RQ2: *How does enriching the LLM with project-specific context affect ambiguity detection performance overall and across different ambiguity mechanisms (lexical, syntactic, semantic, pragmatic)?*

To answer these questions, we use NASA's publicly available James Webb Space Telescope (JWST) Mission requirements document (James Webb Space Telescope Project Mission Requirements Document, 2007) as the empirical basis. These requirements are mature design artefacts from a successfully deployed multi-million USD mission, written with rigorous hierarchical flow-down and embedded in a rich documentation ecosystem. Based on this, we construct a hybrid dataset that combines 246 original JWST requirements with 246 variants containing purposefully induced ambiguity defects. Using this dataset we compare two configurations of LLM based ambiguity detection: a domain-agnostic baseline that applies a guideline style ambiguity rubric evaluation without project knowledge, and a context-aware configuration that augments the same rubric with a "project context" block distilled from the JWST documentation.

This paper has three main contributions: 1) a hybrid labelled dataset derived from JWST mission requirements, 2) a context-aware ambiguity detection pipeline that uses LLMs both to mine project documentation for context, and to perform ambiguity detection with or without that context, and 3) empirical evidence that explicit project context substantially reduces false positives and clarifies when LLM based ambiguity detectors are likely to be useful in design practice. The rest of the paper is structured as follows: Section 2 reviews related work; our methodology is described in Section 3. Section 4 presents the results, which are then discussed in Section 5. Section 6 concludes the paper.

2. Related work

2.1. Requirements quality, ambiguity, and engineering practice

Requirements are central design artefacts, but their typical natural language form makes them vulnerable to multiple kinds of ambiguity. Standards and guidelines such as ISO/IEC/IEEE 29148:2018 and the INCOSE Guide to Writing Requirements emphasise characteristics such as clarity, atomicity, controlled vocabulary, and avoidance of subjective or vague terms (ISO/IEC/IEEE 29148:2018, 2018; Ryan & Wheatcraft, 2023). Despite this guidance, studies show that ambiguity remains one of the most frequent quality issues in requirements, and one that has been hard to overcome (Kamsties, 2005). Berry and Kamsties (2004) argue that many ambiguity judgements depend not only on linguistic clarity but also on a reader's understanding of domain knowledge such as the underlying system and terminology, meaning that ambiguity cannot be assessed purely from surface level text analysis. Empirical research by Massey et al. (2014) similarly found analysts from technical and non-technical background differ in their identification of ambiguity types, indicating the quality assessment itself might be subjective and task-dependent. This also indicates that "perfect" requirements do not exist; instead practitioners should aim for "useful" requirements that are understood well enough, accepting that some ambiguity is inevitable (Berry, 1995; Berry & Kamsties, 2004). This has implications for how automated detection tools should be evaluated, as 100% performance may be neither feasible nor desirable.

Ambiguity has been conceptualised in multiple ways, but four categories outlined by Berry (2003), and Berry and Kamsties (2004), appear frequently in RE literature: lexical (unclear or subjective words), syntactic (interpretation possible due to grammatical structure), semantic (multiple interpretations of meaning possible) and pragmatic (interpretation depends on external context, domain knowledge or assumptions). In literature, particularly the first three have been studied while less focus has been on addressing pragmatic ambiguity (Bano, 2015). However, pragmatic ambiguity is particularly relevant to engineering design, where a single term may have project-specific definitions, units, tolerances and

operational constraints not obvious from the requirement statement itself, highlighting a central challenge that effective ambiguity detection requires both linguistic and domain-specific contextual knowledge.

2.2. Automated requirements engineering support

2.2.1. AI for requirements

Various approaches have been proposed to automated evaluation of requirements quality. Early tools such as NASA's ARM tool (Carlson & Laplante, 2014; Wilson et al., 1997), QuARS (Gnesi & Trentanni, n.d.) and SREE (Tjong & Berry, 2013) relied primarily on rule-based pattern matching and lexicons of forbidden or "dangerous" words. These systems typically achieve high recall by flagging many potentially ambiguous expressions but exhibit limited precision generating high amounts of false positives requiring significant manual filtering as indicated in Table 1.

Table 1. Summary of tools analysis (adapted from Bajceta et al., 2022)

	ARM	QuARS	RETA	RCM
Precision	0.43	0.43	0.41	0.68
Recall	0.08	0.85	0.98	0.38

More recent approaches integrate Natural Language Processing (NLP) pipelines such as part-of-speech heuristics to detect linguistic "smells" including vague terms, weak modality and complex sentence structures (Femmer et al., 2017). However, these remain heavily dependent on shallow textual features and rarely incorporate system context.

Commercial tools such as QVscribe and IBM's RQA operationalise guideline-based checks using proprietary heuristics (IBM, 2023; QRA, 2016). However, their internal logic is largely opaque and published evidence of their performance and context sensitivity is limited. Across both academic and commercial approaches, a constant pattern emerges: tools can detect potential issues, but distinguishing genuinely problematic ambiguity from acceptable phrasing remains difficult without domain grounding.

2.2.2. LLMs for requirements support

The emergence of LLMs has opened new avenues for RE support. Two recent mapping studies provide a high-level landscape. Arora et al. (2024) investigate the potential for use of LLMs to support and transform requirements engineering activities across the development lifecycle, identifying large potential that is currently held back by unresolved challenges such as prompt robustness, managing large false positives in evaluations, and how to integrate domain understanding. Zadenoori et al.'s (2025) systematic literature review into LLMs for RE finds that the LLM research in RE predominantly focuses on requirements elicitation and validation, rather than defect detection and classification; tasks that were dominant in the previous NLP era. This shift leaves a relative gap in LLM-based work on quality assurance for requirements, including ambiguity detection.

Within that gap, a small but growing body of work has begun to explore LLMs for requirements quality evaluation and ambiguity detection. Fantechi, Gnesi and Semini (2023) provide an early comparison of rule-based NLP (using their own QuARS tool) and ChatGPT (GPT-3) for ambiguity detection in requirements documents. Their preliminary study finds that ChatGPT can detect some defects like incompleteness and inconsistency that rule-based tools miss or struggle with, with precision matching NLP based performance while recall performance varied across model runs. However, their evaluation does not consider how project specific domain context affects detection performance.

Lubos et al. (2024) assess LLM capabilities (using Llama 2 70b) for evaluating software requirements characteristics aligned with the ISO 29148 standard. Their findings mirror a pattern relevant to our work: the LLM achieves high recall but low initial precision, and a combined human-LLM assessment outperforms the LLM alone, reinforcing the case for positioning LLMs as decision-support rather than autonomous gatekeepers. However, their domain tailoring was performed through extensive manual work by RE experts rather than through automated approaches.

More recently, Bashir et al. (2025) use LLMs to detect ambiguity with in-context learning using three industrial railway datasets. They evaluate multiple LLMs with in-context learning and evaluation rationale generation. Their work demonstrates the feasibility of LLM-based ambiguity detection in

industrial settings but relies on demonstration examples for in-context learning rather than systematic integration of project specific domain knowledge. Notably, their industry expert evaluation identified the LLM's inadequate understanding of domain-specific terminology as a key limitation, reinforcing the need for structured domain context integration.

Raj et al. (2025) combine NLP-based detection with fine-tuned Llama-2 models for ambiguity classification across seven ambiguity categories, achieving a precision of 0.79 and recall of 0.82, but without project-specific context, a limitation they explicitly flag as future work.

Collectively, this points to three key insights. First, LLMs outperform or complement rule-based tools for requirements quality tasks, particularly for context-dependent defects that surface level pattern matching cannot capture (Fantechi et al., 2023; Lubos et al., 2024). Second, without domain grounding, LLMs tend to exhibit high recall but low precision, the same over-flagging behaviour observed in traditional tools (Arora et al., 2024; Bashir et al., 2025; Lubos et al., 2024). Third, to the best of our knowledge, no existing work systematically integrates project specific contextual knowledge, nor empirically measures how it affects detection performance across different ambiguity types.

2.2.3. Datasets for ambiguity detection

Progress in automated defect detection, including ambiguity detection, has been slowed by the limited availability of labelled datasets. Existing open-source datasets such as PURE and PROMISE lack the necessary explicit ambiguity labels, domain context and glossaries, and realistic hierarchical structure to use for models (Norheim et al., 2024).

In summary, prior work shows substantial progress in automated requirements analysis, but with three clear gaps. First, most tools operate context-free, treating requirements as standalone sentences and therefore generating high false-positive rates. Second, empirical evidence on the role of project-specific context in ambiguity detection is scarce. Finally, existing datasets do not support controlled, reproducible evaluation across different ambiguity types in realistic design settings.

This work addresses these gaps by constructing a hybrid dataset of real and synthetic requirements from NASA's JWST specification, extracting domain context via LLMs and empirically evaluating the effect of context on ambiguity detection across four ambiguity types.

3. Methodology

This paper explores how project-specific context influences LLM-based ambiguity detection. To this end, we follow a build-to-evaluate process inspired by design science research (Peffers et al., 2007): 1) construct a hybrid, labelled dataset, 2) extract and structure project context, 3) implement two ambiguity-detection configurations using LLMs, and 4) evaluate and compare detection performance across ambiguity types. Figure 1 summarises the overall pipeline from the JWST mission requirements to the evaluation setup.

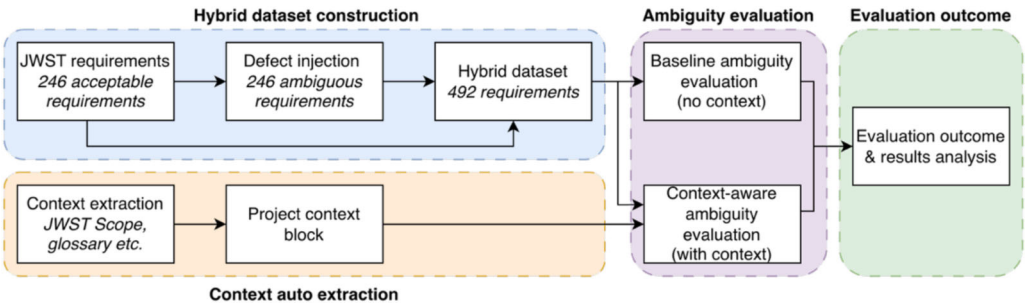

Figure 1. Overview of dataset construction, context extraction and evaluation pipeline

3.1. Dataset development

We developed a hybrid real world/synthetic dataset comprising 246 industry standard quality requirements, and 246 synthetic requirements with controlled defects, by following a two-step process.

Step 1: Identify and clean high-quality requirements

We selected the James Webb Space Telescope (JWST) Mission Requirements Document, revision P (2007) as a representative example of high-quality real-world requirements set. The document is a

comprehensive specification of scope, reference documents, requirements, and qualification and assurance provisions. The requirements have been formulated using rigorous NASA processes, representing mature engineering design practice across diverse technical domains such as system characteristics, reliability and units of measure.

From the complete document, we extracted all 250 requirements verbatim and performed minimal cleaning. We excluded four requirements that included tables due to formatting artefacts causing robustness issues for LLM prompting, resulting in 246 remaining requirements. We treated each of these as non-defect requirements, assuming that a mature engineering organisation's accepted requirements represent acceptable quality.

Step 2: Mutate requirements to become ambiguity-defect requirements

To create synthetic requirements with controlled defects, we systematically injected four types of ambiguity (lexical, syntactic, semantic and pragmatic) into the JWST requirements. We have used an LLM (GPT-4.1 mini) as a controlled text editor applying a unique prompt for each defect type, and separate API call for each requirement. We instructed it to minimally edit each original requirement to introduce only one dominant ambiguity type. The LLM was prompted to preserve core content (e.g. components, parameters, units etc.) while introducing ambiguity through lexical, syntactic, semantic or pragmatic changes, resulting in minimally mutated but defect requirements as seen in Table 2.

Table 2. Examples of original and defect injected requirements

Panel A: Consecutive examples from the round-robin injection scheme (MR-146-149)

ID	Defect type	Original statement	Defect statement
MR-146	Lexical	Protections shall exist to prevent the mutual interference of real-time and stored commanding.	***Adequate*** protections shall exist to prevent the mutual interference of real-time and stored commanding.
MR-147	Syntactic	The Observatory shall remain safe in the event of any command error or break in a command sequence.	The Observatory shall remain safe in the event of any command error or break, in a command sequence. ***(comma injected)***
MR-148	Semantic	The Observatory shall verify all commands received.	The Observatory shall verify all commands received ***by the system***.
MR-149	Pragmatic	The Observatory shall report all verified commands.	The Observatory shall report verified commands ***as appropriate***.

Panel B: Additional examples illustrating the distinction between lexical and pragmatic.

ID	Defect type	Original statement	Defect statement	Why this type
MR-45	Lexical	The planned commissioning phase shall end no later than six months after launch.	The planned commissioning phase shall end ***not much*** later than six months after launch.	"Not much" is an inherently vague quantifier — no context resolves it.
MR-123	Pragmatic	The Observatory shall perform image-based wavefront sensing when commanded.	The Observatory shall perform image-based wavefront sensing when commanded, ***unless operational constraints prevent it.***	"Operational constraints" refers to project-specific conditions defined elsewhere.

We applied a round-robin scheme, mutating the first acceptable requirement with lexical ambiguity, the second with syntactic, the third with semantic, and the fourth with pragmatic, then repeating the cycle. This produced 246 defect requirements, forming a hybrid dataset of 246 acceptable and 246 ambiguous, in total 492 requirements. Within the dataset, the defects are balanced among the four defect types (62 lexical, 62 syntactic, 61 semantic, 61 pragmatic).

3.2. Context extraction

To construct a context block to insert into context-aware ambiguity evaluation prompts, we developed a semi-automatic two-step pipeline for extracting and formatting domain-specific information from the

JWST mission documentation. This assumes, in line with typical systems engineering practice, that requirements documents contain rich contextual information in both explicit form (e.g. mission descriptions, glossaries, referenced documents) and implicit form (domain-specific actors, subsystems, and terminology only mentioned inside requirements statements).

We use different LLMs for distinct tasks: *GPT-4.1mini* for defect injection, *Claude Sonnet 4.5* for context extraction and *Llama-3.3-70B-versatile* for ambiguity evaluation. This separation avoids same-model circularity, where a model might artificially recognise patterns that it generated itself rather than performing genuine ambiguity assessment. Using different models provides a more conservative test of the context-aware approach and better approximates realistic deployment scenarios where evaluation tools may use different underlying models than those used for data preparation or context extraction. Because the same detector evaluates both configurations, any systematic bias in how LLMs interpret LLM-generated content applies equally to the baseline and the context-aware conditions, meaning the observed difference between the two reflects the effect of context rather than an artefact of the pipeline.

Step 1: Context mining

We used a LLM (Claude Sonnet 4.5) to scan the JWST Mission requirements document and identify key contextual entities including system elements, segments, roles, interfaces, referenced documents, units and scales, and definitions of critical terms (e.g. "operational JWST system" and "real-time system"). The prompt instructed the model to extract these from both explicit context and the requirement statements, and to summarise them as neutral definitions and patterns avoiding requirements statements verbatim. This produces an auto-extracted context set that captures the project's vocabulary and typical parameter ranges without manual ontology engineering.

Step 2: Context block formatting

In a second prompt, we ask the LLM to rewrite the extracted material into a "Project Context" block suitable for inclusion in evaluation prompts. During this step, all specific numerical values were normalised into indicative ranges (e.g. "0.005 arcsec" to " (10^-3) arcsec") and references to requirement IDs were removed. This reduces the risk that downstream models simply pattern-match exact numbers or sentence fragments from the original specification instead of using the context semantically.

For evaluation, the context block is supplied unchanged to the ambiguity-detection LLM (Llama-3.3-70B-versatile). The detector LLM never sees the original JWST document directly, it only accesses the distilled context and ambiguity rubric plus the individual requirement under test.

3.3. LLM based ambiguity detection

We model ambiguity detection as a binary classification problem over individual requirements: each requirement is either ambiguous or acceptable. The detector is implemented using an LLM (Llama-3.3-70B-versatile, temperature = 1) accessed via the Groq API.

Crucially, the ambiguity detection prompt does not mention our four ambiguity types. Instead it describes ambiguity in terms of vague quantifiers, subjective terms, incomplete conditions, unclear references and inconsistent scope, broadly following the *ISO29148* and *INCOSE Guide to Writing Requirements* (ISO/IEC/IEEE 29148:2018, 2018; Ryan & Wheatcraft, 2023). This yields a holistic standards-aligned ambiguity judgement, independent of our internal lexical/syntactic/semantic/pragmatic labelling. We evaluate two configurations:

1. **Domain-agnostic baseline:** The model receives only the generic ambiguity rubric and the requirements text
2. **Context-aware configuration:** The model receives an additional "project context" block with key information extracted from the JWST documentation. The model is instructed to use this context to resolve apparent ambiguity and to treat only remaining unresolved triggers of ambiguity as ambiguous.

3.3.1. Prompt structure and LLM settings

The prompt structure specifies the model's role in the system prompt, and each evaluation prompt used chain-of-thought and a few-shot prompt, with generic examples of what to look for rather than overly specific examples to avoid pattern matching. The evaluation prompt includes:

- Task instruction
- Ambiguity rubric (1: subjective terms, 2: weak modality, 3: vague quantifiers etc.)
- Evaluation constraints (schema constraint structured output (JSON format), Boolean ambiguity flag (Yes/No), and an explanation which ambiguity triggers were activated)
- Given requirement text.

The ambiguity rubric is identical in both configurations, only the presence of context differs along with an instruction to flag requirement as non-ambiguous if context resolves ambiguity.
The LLM calls used a temperature of 1.0. A sensitivity check at temperature 0.2 produced metrics within 2.6% from the macro metrics reported here, with the relative advantage of context unchanged. To assess robustness across model sizes, we also replicated the full evaluation using Llama 3.1 8B under identical prompt and temperature settings.

3.3.2. Experimental procedure and metrics

For each requirement in the hybrid dataset, we call the LLM under both configurations (domain-agnostic and context-aware) and record whether the model labels it as ambiguous. We treat ambiguous labels as positive, and non-ambiguous as negatives.
Detection performance is evaluated using precision, recall and F1 (Manning et al., 2009), defined in Equations 1–3.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{1}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{2}$$

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

We compare the baseline and context-aware configuration across all metrics and by ambiguity type (lexical/syntactic/semantic/pragmatic).

4. Results

4.1. Overall detection performance

Table 3 summarises the overall detection performance. The context-aware configuration improves precision from 0.57 to 0.75 (+32%), with only a marginal drop in recall (from 0.61 to 0.59), yielding an F1 improvement from 0.59 to 0.66. Most notably, false positives on NASA's accepted requirements fall by 42% from 107 to 62, directly addressing the over-flagging problem identified in Section 2.

Table 3. Overall ambiguity detection performance

	Precision	Recall	F1 Score	False positives
Baseline	0.57	**0.61**	0.59	107
Context-aware	**0.75**	0.59	**0.66**	**62**
Improvement	32%	-3%	12%	42%

4.2. Performance by ambiguity type

Breaking performance down by ambiguity type reveals a more nuanced pattern. Table 4 details precision, recall, and F1 for each type, with and without context. Context improves precision for all ambiguity types, while recall remains broadly similar except for lexical ambiguity where it reduces by 19%. As a result, F1 increases for every type, most notably for syntactic (0.56 to 0.64) and pragmatic (0.58 to 0.72).

Table 4. Detection performance across ambiguity types

	Baseline			Context-aware			Improvement		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Lexical	0.53	**0.83**	**0.65**	0.68	**0.67**	0.67	27%	−19%	4%
Syntactic	0.60	0.52	0.56	**0.79**	0.53	0.64	32%	2%	14%
Semantic	**0.61**	0.55	0.58	0.75	0.54	0.63	24%	−2%	9%
Pragmatic	0.52	0.64	0.58	0.77	**0.67**	**0.72**	**47%**	**5%**	**24%**

As a robustness check, we replicated the evaluation using Llama 3.1 8B. The context effect holds: precision improves from 0.78 to 0.88 while recall remains stable (0.53 to 0.52), yielding an F1 of 0.65 versus 0.63 at the baseline.

For syntactic and semantic ambiguity, both types show substantial precision gains with little change in recall. Context allows the model to resolve many structural or scope ambiguities, only flagging those where alternative meanings remain problematic given the JWST domain context.

The largest improvement is seen in pragmatic ambiguity, indicating the model becomes substantially better at distinguishing when phrases are ambiguous given the JWST context. This aligns with the intuition that pragmatic ambiguity is tightly linked to domain knowledge and operational practice.

5. Discussion

The results highlight the dual nature of ambiguity in requirements as both a linguistic and a contextual phenomenon. Three findings stand out. First, applying INCOSE and ISO inspired quality characteristics without context causes the model to act as an oversensitive detector, flagging practitioner-accepted requirements as ambiguous. This mirrors observations that practitioners themselves need sufficient domain knowledge to assess ambiguity (Berry & Kamsties, 2004). Second, enriching the model with project context substantially reduces false positives while maintaining similar recall, with the strongest gains for pragmatic ambiguity where judgements hinge on intent and context rather than wording alone. Third, LLM-based ambiguity checks work better as decision-support tools embedded into workflows that already expose such context, or enable auto-extraction and curation of it, than as standalone automated gatekeepers.

The particularly strong improvement for pragmatic ambiguity underscores that implicit knowledge built into the requirements such as unstated assumptions are where AI support can add substantial value, to avoid issues in early design phases and RE that can cause downstream misalignment and rework.

Commercial tools like IBM's RQA and QVscribe emphasise guideline-checks integrated into existing RE environments. They provide scores and warnings but typically do not expose how project glossaries or architectural knowledge affect their decisions. Our study complements this landscape by detailing a methodology for deriving a labelled dataset from an engineering-grade requirements specification, and by empirically demonstrating that explicit project context can substantially improve LLM-based ambiguity judgements, particularly for pragmatic ambiguity. To illustrate, consider MR-145: "*In conjunction with the on-going execution of stored commands, the Observatory shall have the capability to receive and execute real-time commands from the ground segment.*" Without context, the model flags "real-time" as vague, a defensible judgement in isolation. With context, it recognises that "real-time system" forms part of the glossary, thereby correctly resolving the requirement as non-ambiguous.

The findings have implications for how LLMs should be positioned alongside human requirements engineers. The tested LLM configuration still misclassifies some subtle semantic and pragmatic defects. However, this mirrors a well-known challenge in human ambiguity assessment, where borderline disagreements are better interpreted as legitimate judgement variance than as simple failures. Rather than acting as an oracle, the model is more appropriately viewed as a triage that reduces the volume of text that requires human attention, while leaving final decisions with domain experts, consistent with how commercial tools position themselves (IBM, 2023).

Beyond passive flagging, the results also suggest opportunities for interactive disambiguation. When the model identifies an ambiguous requirement, rather than returning a binary label, it could explicitly present alternative readings and invite the author to choose or refine the requirement. This would shift the tasks from abstract quality judgement to concrete decision making about which interpretation should be preserved and is a promising direction for future work.

More broadly, the binary classification used here and in prior evaluations (Bashir et al., 2025; Fantechi et al., 2023; Raj et al., 2025) does not capture the graded nature of ambiguity that practitioners may experience; future work could explore confidence scores or consistency across LLM-generated paraphrases as signals of residual ambiguity severity.

6. Conclusion

We have shown that project-specific context can substantially improve LLM based ambiguity detection, raising precision by 32% and reducing false positives by 42%, with the strongest gains for pragmatic ambiguity.

For the design community, these findings emphasise that requirements exist within interconnected artefact ecosystems; architectures, interface control documents, verification plans, and models, not as isolated specifications. This is especially true in large organisations such as NASA, defence, or public transport, where design decisions are distributed across many artefacts. Our results suggest that LLM based ambiguity checks are most promising when embedded into workflows that already expose such context (e.g. architecture, glossaries, MBSE models), rather than as standalone assessment of individual sentences. By reducing false positives and improving the handling of pragmatic ambiguity, context-aware tools can help design teams spend less effort chasing benign language issues and more effort interrogating the few requirements whose ambiguity is closely tied to system-level assumptions and cross-disciplinary interfaces.

Finally, it is important to note some of the limitations of our study. First, it uses NASA aerospace requirements, specifically mission requirements, which may not fully represent other domains such as automotive or medical devices. JWST requirements are very well structured; in domains with less mature RE practice, structure and ambiguity patterns may differ. The study treats the original JWST requirements as good quality and assumes GPT-4.1 mini injections introduce genuine ambiguity. While manual inspection of the sample supported both assumptions, exhaustive validation was outside the scope of this study. Future replication in domains with different RE maturity and practices, such as automotive, medical devices or defence, would test whether the context effect generalises beyond well-structured aerospace artefacts. Second, our defect injection process produces minimal, controlled defect mutations. While this preserves realism, it may not capture the full variety of naturally occurring defects. Additionally, because defects were generated by an LLM, they may exhibit patterns that are more readily detectable by other LLMs than naturally occurring defects would. Third, whether the context effect transfers beyond the Llama family to other architectures remains an open question. Future work should evaluate this approach across multiple domains, including using real-world datasets. Finally, human studies involving practicing engineers could investigate how LLM feedback affects design discussions, workflows and how to best augment RE practices with AI.

Acknowledgement

This research is supported by an Australian Government Research Training Program (RTP) Scholarship.

References

Araújo, M., Araújo, J., Oliveira, R., Romao, L., & Kalinowski, M. (2025). Domain Knowledge in Requirements Engineering: A Systematic Mapping Study (No. arXiv:2506.20754). arXiv. https://doi.org/10.48550/arXiv.2506.20754

Arora, C., Grundy, J., & Abdelrazek, M. (2024). Advancing Requirements Engineering Through Generative AI: Assessing the Role of LLMs. In A. Nguyen-Duc, P. Abrahamsson, & F. Khomh (Eds), *Generative AI for Effective Software Development* (pp. 129–148). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-55642-5_6

Bajceta, A., Leon, M., Afzal, W., Lindberg, P., & Bohlin, M. (2022). Using NLP tools to detect ambiguities in system requirements—A comparison study. REFSQ Workshops.

Bano, M. (2015, August 24). Addressing the Challenges of Requirements Ambiguity: A Review of Empirical Literature. https://doi.org/10.13140/RG.2.1.3110.7681

Bashir, S., Ferrari, A., Khan, A., Strandberg, P. E., Haider, Z., Saadatmand, M., & Bohlin, M. (2025). Requirements Ambiguity Detection and Explanation with LLMS: An Industrial Study. *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 620–631. https://doi.org/10.1109/ICSME64153.2025.00063

Berry, D. M. (1995). The importance of ignorance in requirements engineering. *Journal of Systems and Software*, 28(2), 179–184. https://doi.org/10.1016/0164-1212(94)00054-Q

Berry, D. M., & Kamsties, E. (2004). Ambiguity in Requirements Specification. In J. C. S. do Prado Leite & J. H. Doorn (Eds), *Perspectives on software requirements*. https://doi.org/10.1007/978-1-4615-0465-8_2
Berry, D. M., Kamsties, E., & Krieger, M. M. (2003). From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity.
Carlson, N., & Laplante, P. (2014). The NASA automated requirements measurement tool: A reconstruction. *Innovations in Systems and Software Engineering*, 10(2), 77–91. https://doi.org/10.1007/s11334-013-0225-8
Fantechi, A., Gnesi, S., & Semini, L. (2023). Rule-based NLP vs ChatGPT in Ambiguity Detection, a Preliminary Study. *CEUR WORKSHOP PROCEEDINGS*, 3378.
Femmer, H., Méndez Fernández, D., Wagner, S., & Eder, S. (2017). Rapid quality assurance with Requirements Smells. *Journal of Systems and Software*, 123, 190–213. https://doi.org/10.1016/j.jss.2016.02.047
Gnesi, S., & Trentanni, G. (n.d.). QuARS A NLP Tool for Requirements Analysis.
Hull, E., Dick, J., & Jackson, K. (Eds). (2005). *Requirements Engineering* (Second Edition). Springer London. https://doi.org/10.1007/b138335
IBM Engineering Requirements Quality Assistant. (2023, May 5). https://www.ibm.com/docs/en/erqa?topic=improving-quality-scores
ISO/IEC/ IEEE 29148:2018. (2018).
James Webb Space Telescope Project Mission Requirements Document (Requirements Document No. JWST-RQMT-000634 Revision P). (2007). Goddard Space Flight Center, NASA. https://spacese.spacegrant.org/uploads/Requirements%20Config/JWST%20Mission%20Requirements%20Document.pdf
Kamsties, E. (2005). Understanding Ambiguity in Requirements Engineering. In A. Aurum & C. Wohlin (Eds), *Engineering and Managing Software Requirement*s (pp. 245–266). Springer. https://doi.org/10.1007/3-540-28244-0_11
Leveraging Natural Language Processing in Requirements Analysis. (2016, November 30). QRA. https://qracorp.com/leveraging-natural-language-processing-in-requirements-analysis/
Lubos, S., Felfernig, A., Tran, T. N. T., Garber, D., El Mansi, M., Erdeniz, S. P., & Le, V.-M. (2024). Leveraging LLMs for the Quality Assurance of Software Requirements. *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, 389–397. https://doi.org/10.1109/RE59067.2024.00046
Manning, C., Raghavan, P., & Schuetze, H. (2009). Introduction to Information Retrieval.
Massey, A. K., Rutledge, R. L., Anton, A. I., & Swire, P. P. (2014). Identifying and classifying ambiguity for regulatory requirements. *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, 83–92. https://doi.org/10.1109/RE.2014.6912250
Norheim, J. J., Rebentisch, E., Xiao, D., Draeger, L., Kerbrat, A., & De Weck, O. L. (2024). Challenges in applying large language models to requirements engineering tasks. *Design Science*, 10, e16. https://doi.org/10.1017/dsj.2024.8
Pahl, G., Beitz, W., Feldhusen, J., & Grote, K.-H. (2007). *Engineering Design*. Springer. https://doi.org/10.1007/978-1-84628-319-2
Peffers, 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. https://doi.org/10.2753/MIS0742-1222240302
QRA Corp. (2016). Leveraging natural language processing in requirements analysis [White paper]. https://qracorp.com/leveraging-natural-language-processing-in-requirements-analysis/
Raj, A., Basit Ur Rahim, M. A., Hussain, S., & Zia, I. (2025). Enhancing Software Requirements Quality: Ambiguity Detection and Resolution Using Large Language Models. In H. R. Arabnia, L. Deligiannidis, F. Shenavarmasouleh, S. Amirian, & F. Ghareh Mohammadi (Eds), *Computational Science and Computational Intelligence* (pp. 340–355). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-95127-5_25
Ryan, M. J., & Wheatcraft, L. (2023). *Guide to Writing Requirements* (4th edn). INCOSE.
Tjong, S. F., & Berry, D. M. (2013). The Design of SREE — A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned. In J. Doerr & A. L. Opdahl (Eds), *Requirements Engineering: Foundation for Software Quality* (pp. 80–95). Springer. https://doi.org/10.1007/978-3-642-37422-7_6
Wilson, W. M., Rosenberg, L. H., & Hyatt, L. E. (1997). Automated analysis of requirement specifications. *Proceedings of the 19th International Conference on Software Engineering - ICSE '97*, 161–171. https://doi.org/10.1145/253228.253258
Zadenoori, M. A., Dąbrowski, J., Alhoshan, W., Zhao, L., & Ferrari, A. (2025). Large Language Models (LLMs) for Requirements Engineering (RE): A Systematic Literature Review (No. arXiv:2509.11446). arXiv. https://doi.org/10.48550/arXiv.2509.11446