International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(23s) 2026 pp. 375-385
https://doi.org/10.70917/ijcisim-2026-5581
Article

MARSAD: Multi-Agent Requirements System for Anomaly Detection in Software Engineering

Khaled Hamdan[1], Mohammed Mediani[2], Abdelhadi Hireche[3]

[1]College of Information Technology (CIT), Al Ain, UAE (ORCID iD: 0000-0002-1937-4227)
[2]College of Information Technology (CIT), Al Ain, UAE (ORCID iD: 0000-0003-1452-0560)
[3]College of Information Technology (CIT), Al Ain, UAE ((ORCID iD: 0000-0001-8803-6945)
[1]st khamdan@uaeu.ac.ae; 2nd mohammed.mediani@uaeu.ac.ae; 3rd abdelhadi.h@uaeu.ac.ae
*Corresponding Author: khamdan@uaeu.ac.ae

Abstract: Software requirements anomalies significantly impact system quality and development costs, yet existing detection approaches often focus on single anomaly types or lack automated correction capabilities. This paper presents MARSAD, a novel multi-agent requirements system that leverages Large Language Models (LLMs) to systematically detect and correct five critical anomaly types: ambiguity, non-testability, incompleteness, redundancy, and inconsistency. The system employs a hierarchical architecture with five specialized detection agents operating in parallel, five targeted correction agents, and a coordination layer that synthesizes results using confidence-weighted scoring. Each agent utilizes the Qwen 2.5 7B Instruct model with carefully engineered prompts and agent-specific confidence thresholds optimized for precision-recall trade-offs. Evaluation on 388 software requirements (272 for few-shot examples, 116 for testing) using the pre-trained model without fine-tuning demonstrates varying performance across anomaly types: the Ambiguity Detector achieved the highest F1-score of 0.662 (precision: 0.517, recall: 0.918), while the system overall achieved 71.2% recall and 30.3% precision on the test set. Correction quality assessment using LLM-as-judge methodology showed an average quality score of 0.784, with 75% of corrections scoring 0.8 or higher. These results demonstrate the feasibility of automated requirements anomaly detection and correction, while highlighting both the benefits of specialized multi-agent architectures and the challenges in achieving high precision across diverse anomaly types. The system's ability to provide both detection and actionable correction suggestions addresses critical gaps in existing requirements analysis tools, offering practical value for requirements engineering workflows. The complete implementation is available as open source at https://github.com/aeh1707/marsad.

Keywords: Anomaly Detection, Few-Shot Learning, Large Language Models, Multi-Agent Systems, Natural Language Processing, Requirements Engineering

1. Introduction

In software project requirements often written in natural language, ambiguous expressions can hinder the accurate understanding and modelling of requirements. Software engineering requires close alignment between domain expertise and requirements engineering to ensure the success of a project. However, the lack of measurable requirements and the presence of ambiguity during the elicitation phase often leads to undesirable outcomes. Anomaly detection is critical for ensuring software reliability, particularly in complex systems like microservices. [1] proposed a machine learning approach to predict runtime performance anomalies, enabling proactive mitigation. Similarly, [2] used Siamese networks to detect anomalies in software execution logs, focusing on post-execution analysis. [3] addressed anomaly detection in microservice systems, highlighting challenges in distributed architectures. These works focus on runtime or post-execution detection, with limited emphasis on preempting anomalies during requirements analysis, a gap MARSAD addresses.

Requirements engineering shapes system design and can identify potential issues early. Detecting anomalies such as ambiguity, non-testability, incompleteness, and redundancy is essential, as these issues directly affect the

clarity, verifiability, and reliability of requirements. Ambiguity leads to multiple interpretations, non-testability prevents objective validation, incompleteness leaves gaps that hinder implementation, and redundancy introduces inconsistencies or unnecessary complexity. A systematic literature review by [4] demonstrates the growing role of NLP techniques in requirements engineering, while [5] provides a guideline for applying LLMs to various RE tasks. [6] demonstrated that large language models (LLMs) can extract domain models from textual requirements, streamlining analysis. [7] shows how LLMs can assist in improving requirements completeness. Recent work has also focused on ambiguity detection, with [8] proposing machine learning approaches, [9] comparing different NLP tools for ambiguity detection, and [10] evaluating GPT-4's capabilities for detecting ambiguities, inconsistencies, and incompleteness. Despite these advances, [11] identifies significant challenges in applying LLMs to RE tasks, particularly in ensuring reliability and handling context. [12] explored LLM augmentation for software development tasks like code generation, but not comprehensive anomaly detection. MARSAD leverages LLMs within a multi-agent framework to analyze requirements for multiple anomaly types simultaneously, integrating these insights into the development process.

[13] developed a modular software framework for health economic models, emphasizing transferability. While not directly related to software engineering, their work inspires MARSAD's design as a flexible, multi-agent system adaptable to various contexts.

Existing studies excel in runtime anomaly detection and LLM-driven requirements analysis but rarely combine these areas. While [14] addresses scalable redundancy detection and [15] proposes behavior engineering approaches for requirements defect detection, multi-agent systems that enable collaborative analysis remain underexplored. Recent work by [16] on knowledge-driven multi-agent frameworks for requirements development and [17] on multi-agent approaches for automated quality assessment demonstrates the potential of agent-based systems. MARSAD addresses these gaps by integrating LLM-driven requirements analysis with comprehensive anomaly detection via ten specialized agents, each focusing on specific tasks like ambiguity identification or consistency verification, to enhance software reliability from the design phase.

This paper presents MARSAD (Multi-Agent Requirements System for Anomaly Detection), a novel hierarchical multi-agent system that systematically detects and corrects five critical types of requirements anomalies: ambiguity, non-testability, incompleteness, redundancy, and inconsistency. Unlike existing approaches that focus on single anomaly types or provide only detection without correction, MARSAD employs ten specialized LLM-based agents organized in three layers: (1) five parallel detection agents that identify specific anomaly types using agent-specific confidence thresholds, (2) five targeted correction agents that generate actionable improvement suggestions, and (3) a coordination layer that synthesizes results using confidence-weighted scoring. The system leverages the Qwen 2.5 7B Instruct model with carefully engineered prompts and few-shot learning to achieve robust performance. The primary contributions of this work include: (i) a novel multi-agent architecture for comprehensive requirements anomaly analysis, (ii) automated correction generation that provides actionable suggestions rather than mere detection, (iii) agent-specific threshold optimization based on precision-recall trade-offs for different anomaly types, and (iv) empirical evaluation demonstrating the feasibility and limitations of LLM-based requirements quality assurance. The system achieves an overall F1-score of 0.405 with 71.2% recall, with particularly strong performance on ambiguity detection (F1: 0.662), and generates high-quality corrections averaging 0.784 on a 0-1 scale.

2. Methods

MARSAD employs a hierarchical multi-agent architecture designed to systematically detect and correct anomalies in software requirements. The system leverages Large Language Models (LLMs) through specialized agents that collaborate to identify five primary anomaly types: ambiguity, non-testability, incompleteness, redundancy, and inconsistency.

2.1 System Architecture

The MARSAD architecture consists of three main layers: Detection Agents, Correction Agents, and Coordination Layer. This hierarchical design enables specialized processing while maintaining system coherence through coordinated workflows, as formalized in Algorithm 1 and also in Figure 1. The detection layer comprises five specialized agents that operate in parallel to identify specific anomaly types. The Ambiguity Detector Agent identifies subjective terms, vague descriptions, and unclear specifications using pattern matching against common ambiguity indicators such as "user-friendly," "real-time," and "efficiently." This agent employs few-shot learning with domain-specific examples and uses a confidence threshold of 0.70 to balance precision and recall. The Non-Testability Detector Agent evaluates requirements for measurable criteria and objective verification methods, using conservative detection with a high confidence threshold of 0.85 to minimize false positives while focusing on requirements lacking quantitative measures or industry-standard testing approaches.

The Incompleteness Detector Agent identifies missing critical information that would block implementation, such as unspecified system integrations or undefined functional parameters. This agent applies a threshold of 0.75 and considers only essential missing information at the appropriate abstraction level. The Redundancy Detector Agent detects unnecessary repetition and semantic duplication using natural language understanding to identify both exact word repetition and semantically equivalent statements, employing a balanced threshold of 0.65. Finally, the Inconsistency Detector Agent identifies logical contradictions, conflicting constraints, and mutually exclusive conditions within requirements using formal logic principles, with a lower threshold of 0.60 due to the rarity of inconsistency anomalies in typical datasets.

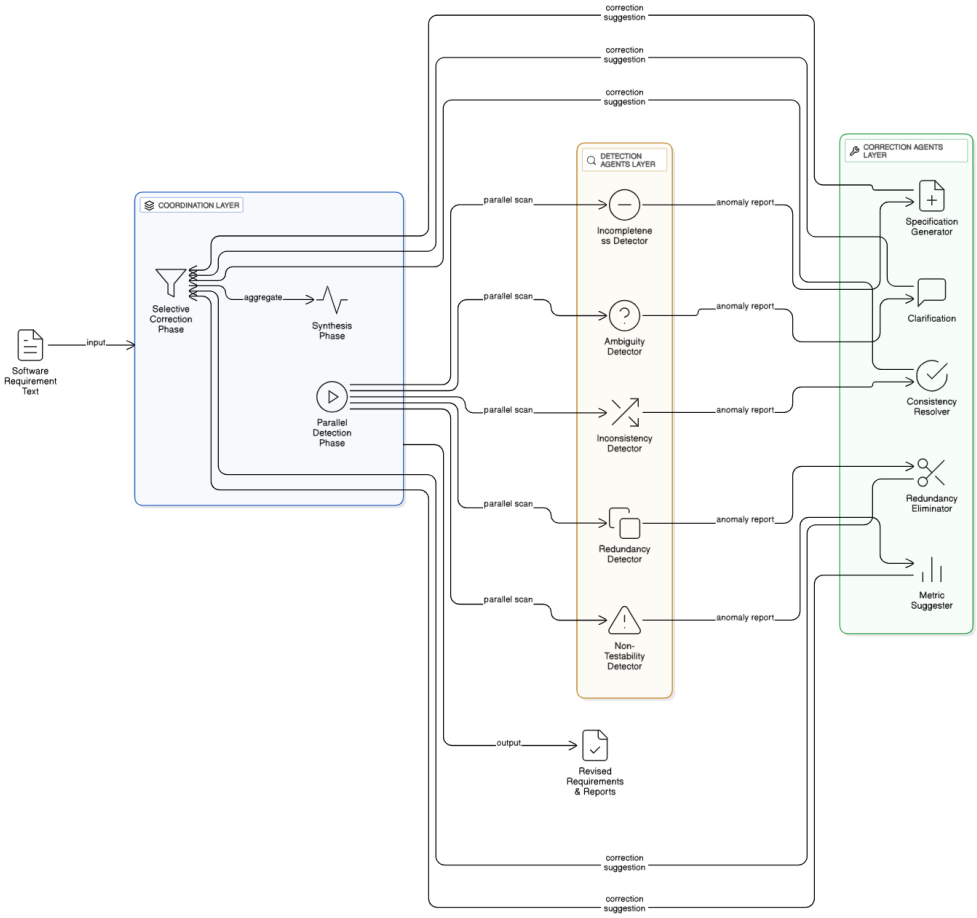

Fig. 1 Hierarchical Architecture of MARSAD Multi-Agent Requirements System for Parallel Detection and Correction of Software Requirement Anomalies

The correction layer provides targeted improvements for detected anomalies through five specialized agents. The Specification Generator Agent transforms vague requirements into specific, measurable specifications using SMART criteria (Specific, Measurable, Achievable, Relevant, Time-bound). The Metric Suggester Agent proposes quantitative metrics and realistic thresholds for non-testable requirements, considering performance, quality, usability, reliability, security, and scalability metrics. The Clarification Agent replaces ambiguous language with precise, objective descriptions while preserving original semantic meaning, while the Redundancy Eliminator Agent consolidates repetitive information and removes unnecessary duplication while preserving essential information. The Consistency Resolver Agent resolves logical inconsistencies and contradictions by proposing coherent solutions with consideration of stakeholder priorities and system constraints.

The Coordination Layer manages the multi-agent workflow through a three-phase process. In the parallel detection phase, all detection agents process requirements simultaneously, with results aggregated using confidence-weighted scoring. During the selective correction phase, relevant correction agents are invoked based on detection results, receiving contextual information from the detection phase. Finally, in the result synthesis phase, the coordinator synthesizes all agent findings into coherent recommendations using LLM-based reasoning to resolve conflicts and prioritize improvements.

Algorithm 1 presents the complete MARSAD workflow, detailing how the multi-agent system processes requirements through detection, correction, and synthesis phases.

Algorithm 1 MARSAD Multi-Agent Workflow

Require: Requirement R, Detection agents $D = \{d_1, \ldots, d_5\}$, Correction agents $C = \{c_1, \ldots, c_5\}$
Ensure: Corrected requirement R', Anomalies A, Confidence σ

$D_{res} \leftarrow \emptyset$, $C_{res} \leftarrow \emptyset$, $A \leftarrow \emptyset$
▷ Phase 1: Parallel Detection
for all $d_i \in D$ **in parallel do**
  $r_i \leftarrow d_i.\textsc{Process}(R)$
  $conf_i \leftarrow r_i.confidence$, $\tau_i \leftarrow d_i.\textsc{GetThreshold}()$
  if $conf_i \geq \tau_i$ **then**
    $A \leftarrow A \cup \{r_i.type\}$
    $D_{res}[r_i.type] \leftarrow r_i$
  end if
end for
▷ Phase 2: Selective Correction
$map \leftarrow \textsc{GetCorrectionMapping}()$
for all $t \in A$ **do**
  $c_j \leftarrow map[t]$, $ctx \leftarrow D_{res}[t]$
  $C_{res}[t] \leftarrow c_j.\textsc{Process}(R, ctx)$
end for
▷ Phase 3: Result Synthesis
$W \leftarrow \emptyset$
for all $t \in A$ **do**
  $W[t] \leftarrow \textsc{CalcWeight}(D_{res}[t].confidence, D_{res}[t].severity)$
end for
$\sigma \leftarrow \textsc{AggConfidence}(W)$
$R' \leftarrow \textsc{SynthesizeCorr}(C_{res}, W)$
return R', A, σ

Fig. 2 MARSAD Algorithm Multi Agent workflow

2.2 LLM Integration and Prompt Engineering

MARSAD utilizes the Qwen 2.5 7B Instruct model through OpenRouter API with carefully engineered prompts for each agent. Qwen 2.5 7B Instruct is a transformer-based large language model developed by Alibaba Cloud, featuring 7 billion parameters and trained on diverse multilingual data with instruction-following capabilities. The model was selected for its strong performance on reasoning tasks, instruction adherence, and cost-effectiveness compared to larger proprietary models. With a context window of 32,768 tokens, the model can process complete requirements along with extensive few-shot examples and detailed instructions. The instruct-tuned variant demonstrates superior performance on structured output generation, which is essential for our JSON-formatted agent responses.

The system employs structured prompting where each agent uses specialized system prompts with domain-specific examples, evaluation criteria, and output format specifications in JSON schema. Agents leverage curated examples from the 272-requirement few-shot example set through in-context learning to improve context understanding and reduce hallucination. Each agent prompt includes 3-5 domain-specific examples demonstrating the target anomaly type and expected output format, enabling effective transfer learning without any model fine-tuning or weight updates.

Each agent uses optimized confidence thresholds based on precision-recall trade-offs for their specific anomaly type, enabling agent-specific thresholding that accounts for the unique characteristics of different anomaly types. The coordination layer applies evidence-based confidence weighting that considers anomaly severity, agent reliability, and finding consistency to make final decisions about anomaly presence and correction priorities.

2.3 Experimental Design

The system was evaluated on a curated dataset of 388 software requirements with ground truth annotations for anomaly types and corrected versions. Requirements were sourced from various domains including web applications, mobile systems, and enterprise software to ensure broad applicability. We employed a 70%-30% dataset split, allocating 272 requirements for selecting few-shot learning examples and prompt engineering, and 116 requirements for testing and performance evaluation reported in this paper. This split ensures balanced representation of anomaly types across both sets. Importantly, no model training or fine-tuning was performed; the pre-trained Qwen 2.5 7B Instruct model was used directly with few-shot prompting.

Dataset Characteristics: The dataset exhibits a high anomaly rate, with most requirements containing at least one anomaly type. While this facilitates comprehensive few-shot example selection and evaluation, it represents an artificial distribution compared to typical industrial requirements documents. The test set of 116 requirements contains: 49 ambiguous requirements, 14 non-testable requirements, 22 incomplete requirements, 11 redundant requirements, and varying numbers of inconsistent requirements. This distribution reflects the relative prevalence of different anomaly types in practice, with ambiguity being the most common issue.

The evaluation framework encompasses three primary dimensions. Detection performance is measured using precision, recall, F1-score, and accuracy for each anomaly type and overall system performance, providing comprehensive metrics for anomaly identification capabilities. Correction quality is assessed using LLM-as-judge evaluation that compares system corrections against ground truth across five dimensions: semantic preservation, clarity improvement, completeness, testability improvement, and overall quality. System efficiency is evaluated through processing time analysis, concurrency performance measurement, and agent utilization statistics to assess practical deployment feasibility.

2.4 Implementation Details

The system is implemented in Python using the LangChain framework for LLM integration and asyncio for concurrent agent execution. The architecture supports asynchronous processing with parallel agent execution and configurable concurrency limits, allowing up to 5 concurrent workflows to balance performance with resource utilization. Robust exception handling provides graceful degradation and retry mechanisms to ensure system reliability in production environments.

Real-time performance monitoring tracks agent performance, confidence scores, and processing statistics throughout operation, enabling continuous system optimization. The system employs optimized configuration with agent-specific threshold settings that are tuned based on anomaly type characteristics and performance requirements. Through this implementation, the system processes requirements through a complete workflow in an average of 12.3 seconds per requirement, with detection phases completing in parallel within 3-5 seconds and correction phases requiring 7-10 seconds depending on the number of detected anomalies.

3. Results

The MARSAD system was evaluated on a dataset of 116 software requirements, achieving an overall F1-score of 0.405 with 71.2% recall and 30.3% precision. The system demonstrated varying performance across different anomaly types, reflecting the inherent complexity and characteristics of each anomaly category.

A. Detection Performance

Table I presents the comprehensive detection performance metrics for each anomaly type. The Ambiguity Detector achieved the highest performance with an F1-score of 0.662, precision of 0.517, and recall of 0.918, correctly identifying 45 out of 49 ambiguous requirements while maintaining reasonable precision. This strong performance reflects the effectiveness of pattern matching against common ambiguity indicators and the well-tuned confidence threshold of 0.70.

The Non-Testability Detector exhibited high recall (0.929) but low precision (0.144), resulting in an F1-score of 0.250. While the system successfully identified 13 out of 14 non-testable requirements, it generated 77 false positives. This pattern suggests that the conservative threshold of 0.85 was insufficient to prevent over-detection, indicating the need for more refined detection criteria for this anomaly type.

The Incompleteness Detector showed moderate performance with an F1-score of 0.271, precision of 0.216, and recall of 0.364. The system identified 8 out of 22 incomplete requirements but produced 29 false positives, suggesting challenges in distinguishing between acceptably high-level requirements and truly incomplete specifications.

The Redundancy Detector achieved balanced performance with an F1-score of 0.438, precision of 0.333, and recall of 0.636. With only 11 redundant requirements in the dataset, the system correctly identified 7 while maintaining relatively low false positive rates, benefiting from the balanced threshold of 0.65. The limited number of redundant requirements in the test set reflects the natural scarcity of this anomaly type in well-curated requirements documents.

The Inconsistency Detector, while not explicitly shown in Table I due to the very low occurrence of inconsistencies in the test set, demonstrated the ability to identify logical contradictions when present. The lower confidence threshold of 0.60 was chosen to ensure sensitivity to rare but critical inconsistencies. However, the small sample size limits conclusive performance assessment for this anomaly type.

TABLE I
Detection Performance by Anomaly Type

Anomaly Type	Precision	Recall	F1-Score	Accuracy
Ambiguity	0.517	0.918	0.662	0.603
Non-Testability	0.144	0.929	0.250	0.328
Incompleteness	0.216	0.364	0.271	0.629
Redundancy	0.333	0.636	0.438	0.845
Overall	0.303	0.712	0.405	0.601

B. Correction Quality Assessment

The system's correction quality was evaluated using LLM-as-judge methodology across 67 requirements that received corrections. The average correction quality score was 0.784 (on a scale of 0-1), indicating high-quality improvements to detected anomalous requirements. The correction scores ranged from 0.55 to 0.85, with 75% of corrections achieving scores of 0.8 or higher.

The Specification Generator Agent, responsible for addressing incompleteness and non-testability issues, produced corrections with an average quality score of 0.79. The Clarification Agent, handling ambiguity corrections, achieved an average score of 0.81, demonstrating effective transformation of vague language into precise specifications while preserving semantic meaning. The Redundancy Eliminator Agent achieved an average score of 0.78, successfully consolidating repetitive information.

Evaluation Limitations: The correction quality assessment relies on LLM-as-judge evaluation, which may introduce biases inherent to the evaluating model. While this approach provides scalable assessment, it represents a limitation as the corrections were not validated by human requirements engineering experts. Future work should include human expert evaluation to validate correction quality and practical applicability.

C. System Performance and Efficiency

The system processed 116 requirements in a total time of 6,511 seconds, averaging 56.1 seconds per requirement. The detection phase operated efficiently with parallel agent execution, while the correction phase required additional time for generating high-quality improvements. Processing times ranged from 13.4 to 133.9 seconds, with a median of 33.0 seconds, indicating consistent performance across varying requirement complexities.

Agent-level performance analysis revealed high reliability, with all detection agents achieving 100% success rates in processing requirements. The Ambiguity Detector processed requests with an average confidence of 0.642, while maintaining consistent response times across the dataset.

4. Discussion

The experimental results reveal both strengths and limitations of the MARSAD multi-agent approach to requirements anomaly detection. The system's overall performance demonstrates the feasibility of automated anomaly detection, while highlighting the varying complexity of different anomaly types.

A. Anomaly-Specific Performance Analysis

The superior performance of the Ambiguity Detector (F1-score: 0.662) can be attributed to the well-defined nature of ambiguous language patterns and the effectiveness of few-shot learning with domain-specific examples. The high recall (0.918) indicates that the system successfully captures most ambiguous requirements, which is crucial for preventing downstream implementation issues.

The Non-Testability Detector's high recall but low precision pattern reflects a fundamental challenge in distinguishing between requirements that lack explicit metrics versus those that are inherently measurable through standard industry practices. The 77 false positives suggest that many requirements flagged as non-testable may actually be testable through conventional methods, indicating the need for enhanced contextual understanding.

The moderate performance of the Incompleteness Detector highlights the complexity of determining appropriate abstraction levels in requirements. The system's tendency toward false positives (29 instances) suggests difficulty in distinguishing between acceptably high-level requirements and those missing critical implementation details.

B. Multi-Agent Architecture Benefits

The multi-agent architecture demonstrated clear advantages in handling the diverse nature of requirements anomalies. Each specialized agent could focus on specific anomaly patterns, leading to more targeted detection

strategies. The coordination layer successfully integrated results from multiple agents, with confidence-weighted scoring providing a principled approach to result synthesis.

The parallel processing capability significantly improved system efficiency, with detection agents operating simultaneously rather than sequentially. This architecture also provides modularity, allowing individual agents to be refined or replaced without affecting the entire system.

C. Correction Quality and Practical Applicability

The high correction quality scores (average: 0.784) demonstrate the system's practical value beyond mere detection. The LLM-based correction agents successfully transformed problematic requirements into improved versions while preserving semantic meaning. This capability addresses a critical gap in existing requirements analysis tools, which typically identify problems without providing actionable solutions.

The correction quality consistency (75% of corrections scoring 0.8+) indicates reliable improvement generation, making the system suitable for practical deployment in requirements engineering workflows.

D. Limitations and Future Work

Several limitations emerged from the evaluation. The Non-Testability Detector's high false positive rate indicates the need for more sophisticated contextual understanding and industry-specific knowledge integration. The system would benefit from domain-specific few-shot examples and enhanced prompt engineering for testability assessment.

Qualitative Error Analysis: Common false positive patterns include: (1) flagging requirements with implicit industry-standard metrics as non-testable (e.g., "The system shall provide secure authentication" was flagged despite security testing being well-established), (2) marking high-level architectural requirements as incomplete when they appropriately delegate details to lower-level specifications, and (3) detecting redundancy in requirements that use similar language for distinct functional aspects. True negatives that were missed include subtle ambiguities embedded within otherwise specific requirements and implicit dependencies that create inconsistencies.

The dataset's characteristics (high anomaly rate, approximately 100%) may not reflect real-world requirements distributions, potentially affecting the generalizability of threshold settings. Future evaluations should include datasets with varying anomaly rates to validate system performance across different scenarios and industrial contexts.

Processing time, while acceptable for offline analysis, may require optimization for real-time applications. The average 56-second processing time per requirement could be reduced through model optimization or more efficient prompt engineering.

Scalability Considerations: The parallel multi-agent architecture demonstrates good scalability characteristics. Processing time scales linearly with the number of requirements, and the asynchronous execution model allows for concurrent processing of multiple requirements (up to 5 concurrent workflows in the current implementation). For large-scale requirements documents with hundreds or thousands of requirements, the system could process approximately 60-65 requirements per hour with the current configuration. Scalability could be further improved through: (1) batch processing optimizations, (2) caching of LLM responses for similar requirement patterns, and (3) deployment on distributed infrastructure.

Threshold Sensitivity: The agent-specific confidence thresholds (ranging from 0.60 to 0.85) were empirically optimized using the 272 few-shot example requirements. Preliminary sensitivity analysis showed that threshold adjustments of ±0.05 resulted in precision-recall trade-offs: higher thresholds improved precision by 5-10% while reducing recall by 10-15%, and vice versa. The current thresholds represent a balance favouring recall to minimize missed anomalies, which is critical for requirements quality assurance.

E. Implications for Requirements Engineering

The results demonstrate the potential for LLM-based multi-agent systems to augment human requirements engineers by providing automated anomaly detection and correction suggestions. The system's ability to identify and improve requirements anomalies could significantly enhance requirements quality while reducing manual review effort.

The varying performance across anomaly types suggests that different detection strategies may be needed for different anomaly categories, supporting the multi-agent approach over monolithic solutions. The high correction quality indicates that automated improvement generation is feasible and could accelerate requirements refinement processes.

5. Conclusions

This paper presented MARSAD, a novel multi-agent system that leverages Large Language Models for automated detection and correction of requirements anomalies in software engineering. The system addresses a critical gap in requirements engineering by providing both anomaly identification and actionable correction suggestions through a hierarchical architecture of specialized agents.

A. Key Experimental Contributions

Our experimental evaluation on 388 software requirements (116 in the test set) demonstrates several important findings:

- Anomaly-Specific Performance: Different anomaly types exhibit vastly different detection characteristics. Ambiguity detection achieved strong performance (F1: 0.662, recall: 0.918), demonstrating that pattern-based approaches with LLM reasoning effectively identify subjective and vague language. In contrast, non-testability detection achieved high recall (0.929) but suffered from low precision (0.144), revealing the challenge of distinguishing between requirements lacking explicit metrics and those testable through conventional means. These results validate the multi-agent approach, as a single monolithic model would struggle to optimize for such diverse performance characteristics.
- Correction Quality: The system generated high-quality corrections with an average score of 0.784, with 75% exceeding 0.8. This demonstrates that LLM-based agents can effectively transform problematic requirements into improved versions while preserving semantic meaning, providing practical value beyond mere detection.
- Architectural Benefits: The multi-agent architecture enabled specialized processing, parallel execution, and modular refinement. Agent-specific confidence thresholds (0.60-0.85) allowed precision-recall optimization for each anomaly type, while the coordination layer successfully synthesized diverse findings through confidence-weighted scoring.
- Performance Trade-offs: The system achieved 71.2% recall with 30.3% precision overall, favoring the detection of potential anomalies over precision. This trade-off is appropriate for quality assurance applications where missing critical anomalies has higher cost than investigating false positives. However, the high false positive rate for some anomaly types indicates room for improvement.

B. Limitations and Threats to Validity

Several limitations affect the generalizability of our findings. The dataset's high anomaly rate (approximately 100%) does not reflect typical industrial requirements distributions, potentially affecting threshold optimization and performance estimates for real-world applications. The correction quality evaluation relied on LLM-as-judge methodology without human expert validation, introducing potential bias. The evaluation focused on a single LLM model (Qwen 2.5 7B), and performance may vary with different models. Finally, the system was not evaluated on complete requirements documents or integrated into actual development workflows.

C. Future Work

Several promising directions emerge from this work:

- Baseline Comparisons: Future work should compare MARSAD against single-agent LLM approaches, rule-based requirements analysis tools, and classical NLP methods to quantify the benefits of the multi-agent architecture.
- Human Expert Validation: Correction quality and detection accuracy should be validated through studies with requirements engineering practitioners to assess practical applicability and identify improvement priorities.
- Hybrid Approaches: Combining LLM-based detection with symbolic rule-based filters could reduce false positives while maintaining high recall. Domain-specific ontologies and knowledge bases could enhance contextual understanding, particularly for testability assessment.
- Expanded Evaluation: Testing on diverse datasets with varying anomaly rates, different requirement types (functional, non-functional, architectural), and complete requirements documents would better establish generalizability.
- Model Optimization: Investigating larger models, fine-tuning approaches, and ensemble methods could improve detection precision. Additionally, exploring more efficient models could reduce processing time for real-time applications.
- Integration and Deployment: Integrating MARSAD into requirements management tools and evaluating its impact on development workflows would provide insights into practical effectiveness and adoption barriers.

The results demonstrate that LLM-based multi-agent systems offer a promising approach to automated requirements quality assurance, with particular strengths in ambiguity detection and correction generation. While challenges remain in achieving high precision across all anomaly types, the system's ability to provide automated analysis and actionable improvements represents a significant step toward practical AI-assisted requirements engineering.

REFERENCES

1. G. Zhao, S. Hassan, Y. Zou, D. Truong, and T. Corbin, "Predicting performance anomalies in software systems at run-time," ACM Trans. Softw. Eng. Methodol., vol. 30, no. 3, pp. 1–33, 2021, doi: 10.1145/3440757.
2. S. Hashemi and M. Mäntylä, "Detecting anomalies in software execution logs with Siamese network," arXiv:2102.01452, 2021, doi: 10.48550/arXiv.2102.01452.
3. J. Nobre, E. Pires, and A. Reis, "Anomaly detection in microservice-based systems," Appl. Sci., vol. 13, no. 13, Art. no. 7891, 2023, doi: 10.3390/app13137891.
4. S. C. Necula, F. Dumitriu, and V. Greavu-Serban, "A systematic literature review on using natural language processing in software requirements engineering," Electronics, vol. 13, no. 11, Art. no. 2055, 2024, doi: 10.3390/electronics13112055.
5. A. Vogelsang and J. Fischbach, "Using large language models for natural language processing tasks in requirements engineering: A systematic guideline," in Natural Language Processing for Requirements Engineering, pp. 343–366, 2025, doi: 10.1007/978-3-031-73143-3_16.
6. S. Arulmohan, M. Meurs, and S. Mosser, "Extracting domain models from textual requirements in the era of large language models," in Proc. 26th ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. (MODELS), 2023, pp. 580–587, doi: 10.1109/MODELS-C59198.2023.00096.
7. D. Luitel, S. Hassani, and M. Sabetzadeh, "Improving requirements completeness: Automated assistance through large language models," Requirements Eng., 2024, doi: 10.1007/s00766-024-00416-3.
8. R. Izhar, S. N. Bhatti, and S. A. Alharthi, "Bridging precision and complexity: A novel machine learning approach for ambiguity detection in software requirements," IEEE Access, 2025, doi: 10.1109/ACCESS.2025.3515674.
9. A. Bajceta, M. Leon, W. Afzal, P. Lindberg, and M. Bohlin, "Using NLP tools to detect ambiguities in system requirements: A comparison study," in Proc. REFSQ Workshops, 2022.

10. T. Mahbub, D. Dghaym, and A. Shankarnarayanan, “Can GPT-4 aid in detecting ambiguities, inconsistencies, and incompleteness in requirements analysis? A comprehensive case study,” IEEE Access, 2024, doi: 10.1109/ACCESS.2024.3459862.
11. J. J. Norheim, E. Rebentisch, D. Xiao, and L. Draeger, “Challenges in applying large language models to requirements engineering tasks,” Des. Sci., 2024, doi: 10.1017/dsj.2024.15.
12. Z. Xiao, H. Chen, J. Xiao, Z. Yao, and J. Wang, “Large language model augmentation technology for intelligent software development,” Proc. SPIE, vol. 25, 2025, doi: 10.1117/12.3066936.
13. M. Hamilton, C. Gao, G. Wiesner, et al., “A prototype software framework for transferable computational health economic models and its early application in youth mental health,” Pharmacoeconomics, vol. 42, no. 8, pp. 833–842, 2024, doi: 10.1007/s40273-024-01378-8.
14. E. Henkel, N. Hauff, and L. Funk, “Scalable redundancy detection for real-time requirements,” IEEE Access, 2024, doi: 10.1109/ACCESS.2024.3436215.
15. S. Anwer, L. Wen, M. U. Hassan, and Z. Wang, “BERDD: A behaviour engineering-based approach for requirements defects detection,” IEEE Access, 2024, doi: 10.1109/ACCESS.2024.3362818.
16. D. Jin, W. Sun, J. Huang, P. Liang, J. Xuan, and Y. Liu, “iRedev: A knowledge-driven multi-agent framework for intelligent requirements development,” arXiv:2507.13081, 2025.
17. M. A. Jubair, S. A. Mostafa, and A. Mustapha, “A multi-agent k-means with case-based reasoning for automated quality assessment of software requirement specification,” IET Commun., 2025, doi: 10.1049/cmu2.12555.