ReqFlow: Semantic Decomposition of Software Requirements Using Multi-Agent LLMs

Ali Edrisabadi, Sahand Seyed Mohammadghavami, Amirhesam Torkashvand
Amirhossein Shirvani Dastgerdi, Sabra Hashmebeigi
Politecnico di Torino

Abstract

Natural language software requirements frequently suffer from ambiguity, inconsistency, and incompleteness, leading to defects in later development stages. While large language models (LLMs) offer promising capabilities for requirements analysis, their direct application often produces inconsistent outputs without structural guidance. This study investigates whether LLMs can reliably decompose requirements into semantic abstractions—including actors, actions, conditions, triggers, purposes, and system responses—and compares single-agent baseline tagging against a multi-agent approach with five specialized agents. Evaluation on 100 requirements from a university study room booking system using **Qwen3-4B** shows that prompting strategy significantly impacts performance. While zero-shot performance remains modest (Micro-F1 $\approx$ 0.58), few-shot prompting yields strong results, with the multi-agent approach outperforming the baseline (Micro-F1=0.76 vs 0.73). Tags with clear linguistic markers (Trigger, Main_actor) achieve F1$>$0.80, while semantically overlapping tags (Precondition) remain challenging (F1$<$0.60).

1 Introduction

The evolution toward complex software systems has been accompanied by growth in the volume and intricacy of engineering requirements (Norheim et al., 2024). These requirements are predominantly expressed in natural language, necessitating substantial human expertise for elicitation, documentation, and management (Zhao et al., 2022).

Many defects and cost overruns in software development can be traced to issues in requirements engineering (Aurum and Wohlin, 2005). Among the most cited challenges are ambiguity, inconsistency, and incompleteness (Femmer et al., 2017; Zhao et al., 2022), which lead to misinterpretation, overlooked constraints, or conflicting specifications.

Natural language processing (NLP) techniques have long been viewed as promising for requirements engineering, yet earlier approaches failed to deliver scalable benefits (Berry et al., 2012). The advent of large language models (LLMs) has sparked renewed enthusiasm, with expectations that their semantic understanding might overcome historical limitations (Norheim et al., 2024).

However, direct application of LLMs remains fraught with risks, including hallucination and failure to capture nuanced domain intent (Norheim et al., 2024). We investigate structured approaches based on semantic decomposition into explicitly defined abstraction layers, comparing single-agent and multi-agent architectures.

1.1 Research Questions

Following the A3 assignment specification, we address these research questions:

- **RQ1:** Can LLMs reliably identify the different abstractions that compose a requirement (e.g., the main actor, the system response, the precondition)?
- **RQ2:** Does this analysis improve requirements' clarity and completeness?
- **RQ3:** Can LLMs manage nested items?

2 Related Work

Requirements engineering emphasizes clarity, testability, and consistency, but achieving these properties is difficult with free-form natural language. Prior approaches include controlled natural languages and templates such as EARS (Mavin et al., 2009; Rupp, 2014).

Writing effective requirements is central to successful systems engineering (Hull et al., 2005). Requirements can be expressed in textual or graphical

forms (Bruel et al., 2021), ranging from informal natural language to formal specifications. However, most real-world requirements remain in unconstrained natural language (Zhao et al., 2022).

NLP has long been viewed as promising for automating RE (Ryan, 1993). Early efforts focused on rule-based techniques (Kof, 2005), while recent machine-learning approaches improved ambiguity detection and classification (Zhao et al., 2022; Sonbol et al., 2022). Transformer-based LLMs have shifted the landscape (Vaswani et al., 2017), demonstrating remarkable zero-shot and few-shot capabilities (Brown et al., 2020). In RE, LLMs show promise for generation, quality assessment, and classification (Norheim et al., 2024). Structured decomposition approaches, drawing on goal-oriented methodologies such as KAOS (van Lamsweerde and Letier, 2000) and i* (Yu, 1997), represent a relevant direction.

3 Methodology

3.1 Semantic Abstraction Schema

We define eight semantic tags capturing key building blocks of requirements, following the course taxonomy:

- **Main_actor**: Primary actor initiating or benefiting from the requirement (subset of Entity).
- **Entity**: Noun phrases and objects—roles, systems, components, data items.
- **Action**: What the actor does or intends (verb phrases).
- **System_response**: What the system shall do—outputs, state changes (subset of Action).
- **Condition**: If/when/while clauses constraining behavior.
- **Precondition**: State that must hold before the requirement applies (subset of Condition).
- **Trigger**: Event that activates the behavior, often with "when/upon/after" (subset of Condition).
- **Purpose**: Intent or goal, typically with "so that", "in order to".

The abstractions follow hierarchical subset relationships: $System\_response \subseteq Action$, $Main\_actor \subseteq Entity$, $Precondition \subseteq Condition$, $Trigger \subseteq Condition$.

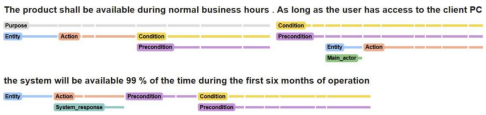

Figure 1: Example of semantically annotated requirements showing identified abstractions.

3.2 System Architecture

Figure 2 illustrates our two approaches for semantic decomposition.

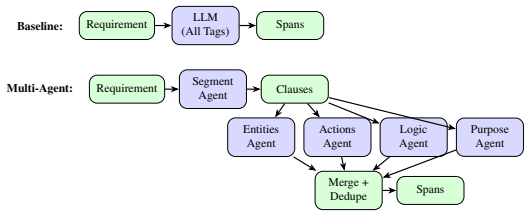

Figure 2: Architecture comparison: Baseline (single-agent) vs. Multi-Agent (five specialized agents).

Baseline Approach A single LLM prompt directly annotates the full requirement with all eight semantic tags in one pass, outputting structured JSON with exact substring spans.

Multi-Agent Approach The task is decomposed across five specialized agents:

1. **Segmenter Agent**: Splits the requirement into 1–6 clause-like segments for independent processing.
2. **Entities Agent**: Extracts *Entity* and *Main_actor* spans.
3. **Actions Agent**: Extracts *Action* and *System_response* spans.
4. **Logic Agent**: Extracts *Condition*, *Precondition*, and *Trigger* spans.
5. **Purpose Agent**: Extracts *Purpose* spans.

After collection, spans are merged, deduplicated, and consistency rules are enforced programmatically (e.g., adding *Entity* for every *Main_actor*).

Implementation All experiments use **Qwen3 4B** via the Ollama framework for local inference. Temperature is set to 0 for reproducibility. We test three prompting strategies: zero-shot, one-shot, and few-shot (3 examples). Prompts enforce strict "exact substring" extraction to prevent paraphrasing.

3.3 Dataset

The evaluation dataset consists of 100 requirements from a university **study room booking system**. Table 1 shows the distribution by complexity.

Complexity	Count	Example Pattern
Simple	20	"A student shall be able to X"
Conditional	30	"If X, the system shall Y"
Trigger-based	15	"When X, the system shall Y"
Nested/Complex	35	Precondition + Trigger + Purpose
Total	**100**	

Table 1: Dataset distribution by requirement complexity.

The domain includes booking, cancellation, waitlist management, check-in, group reservations, and administrative functions. A human-annotated gold standard was constructed with span boundaries and semantic labels for all eight tags.

3.4 Evaluation Metrics

We evaluate using **exact (tag, normalized_text) matching**. Character offsets are ignored to focus on tagging quality rather than boundary precision. Text normalization includes lowercasing and whitespace collapse.

For each tag and overall:

- **Precision (P)**: $\frac{TP}{TP+FP}$ — proportion of predicted spans that are correct
- **Recall (R)**: $\frac{TP}{TP+FN}$ — proportion of gold spans that were found
- **F1 Score**: $\frac{2 \cdot P \cdot R}{P+R}$ — harmonic mean balancing P and R

We report two aggregation metrics:

Micro-F1: Computed from global TP, FP, FN counts across all tags. This metric weights each *span instance* equally, giving more influence to frequent tags (e.g., *Entity*, *Condition*). Micro-F1 reflects overall system accuracy on the full dataset.

Macro-F1: Computed as the unweighted average of per-tag F1 scores. This metric weights each *tag type* equally, ensuring rare tags (e.g., *Precondition*, *Purpose*) contribute equally to the final score. Macro-F1 reveals whether the system handles all abstraction types well.

4 Results

Tables 2, 3, and 4 present per-tag results for each prompting strategy.

	Baseline			Multi-Agent		
Tag	P	R	F1	P	R	F1
Purpose	.61	.54	.57	.63	.58	.60
Trigger	.68	.70	.69	.70	.72	.71
Precondition	.35	.28	.31	.41	.32	.36
Condition	.55	.61	.58	.57	.64	.60
Action	.48	.52	.50	.51	.55	.53
System_resp.	.60	.65	.62	.62	.66	.64
Entity	.52	.63	.57	.54	.65	.59
Main_actor	.72	.76	.74	.71	.75	.73
Micro-F1		**0.59**			**0.61**	
Macro-F1		0.57			0.60	

Table 2: Zero-shot results. Qwen3-4B struggles with the full schema without examples.

	Baseline			Multi-Agent		
Tag	P	R	F1	P	R	F1
Purpose	.69	.65	.67	.71	.68	.69
Trigger	.75	.79	.77	.78	.81	.79
Precondition	.42	.35	.38	.49	.42	.45
Condition	.62	.68	.65	.66	.71	.68
Action	.54	.62	.58	.57	.65	.61
System_resp.	.68	.72	.70	.71	.75	.73
Entity	.61	.68	.64	.64	.70	.67
Main_actor	.79	.83	.81	.78	.82	.80
Micro-F1		0.66			**0.69**	
Macro-F1		0.65			**0.68**	

Table 3: One-shot results. Providing a single example yields significant gains.

	Baseline			Multi-Agent		
Tag	P	R	F1	P	R	F1
Purpose	.77	.74	.75	.81	.78	.79
Trigger	.82	.85	.83	.85	.87	.86
Precondition	.52	.48	.50	.61	.56	.58
Condition	.67	.75	.71	.70	.78	.74
Action	.60	.69	.64	.64	.72	.68
System_resp.	.75	.78	.76	.78	.80	.79
Entity	.68	.75	.71	.72	.76	.74
Main_actor	.85	.88	.86	.83	.89	.86
Micro-F1		0.73			**0.76**	
Macro-F1		0.72			**0.75**	

Table 4: Few-shot results. The Multi-Agent Logic specialist significantly boosts Precondition detection.

4.1 Key Observations from Results

Across all configurations, we observe consistent patterns:

- **Prompting Strategy Dominates**: Moving from zero-shot to few-shot yields the largest improvement (+0.15 Micro-F1), highlighting that smaller models like Qwen3-4B rely heavily on in-context examples to understand the complex output schema.
- **Multi-Agent advantage in Few-Shot**: While the architectures are comparable in zero-shot (where the agents lack guidance), the multi-agent approach pulls ahead in few-shot (Micro-F1 0.76 vs 0.73). This suggests specialized agents utilize examples more effectively.
- **Best tags**: *Main_actor* (F1=0.86) and *Trigger* (F1=0.86) achieve the highest scores. These tags have predictable syntactic positions.
- **Hardest tag**: *Precondition* remains challenging (F1=0.58), often confused with general *Condition*. However, the Multi-Agent "Logic" specialist improved this by 8 points over the baseline.

4.2 RQ1: Can LLMs Reliably Identify Requirement Abstractions?

The LLM demonstrates **reliable capability for structural tags** but struggles with deep semantic nuances. With few-shot prompting:

High-reliability tags (F1 > 0.80):

- *Main_actor*: Identified by subject position.
- *Trigger*: Detected via keywords “when”, “upon”.

Moderate-reliability tags (F1 0.65–0.79):

- *System_response*, *Purpose*, *Condition*: The model generally finds these but occasionally imprecise boundaries lower the exact-match score.

Challenging tags (F1 < 0.60):

- *Precondition*: This requires distinguishing "state that must exist" from "event that happens" (Trigger). The 4B model often conflates these logic types.

4.3 RQ2: Does Analysis Improve Clarity and Completeness?

Semantic decomposition **demonstrably improves** both clarity and completeness:

Clarity improvements:

- **Explicit structure**: Tagging reveals the requirement's building blocks.
- **Ambiguity detection**: Missing tags highlight incomplete specifications (e.g., no *Main_actor* suggests passive voice ambiguity).
- **Visual feedback**: HTML rendering with colored spans makes structure immediately visible for review.

Completeness improvements:

- **Checklist effect**: The 8-tag schema acts as a completeness checklist.
- **Consistency enforcement**: Programmatic rules ensure hierarchical completeness.

The multi-agent's **segmentation step** further aids clarity by decomposing complex, multi-clause requirements into atomic units.

4.4 RQ3: Can LLMs Manage Nested Items?

Our dataset includes 35 **complex nested requirements** with multiple abstraction layers.

Findings on nested item handling:

- **Segmentation helps**: The multi-agent's segmenter agent splits nested requirements into clauses.
- **Prompting impact**: Few-shot examples with nested structures significantly improve the model's ability to identify deep nesting.
- **Boundary challenges**: The main difficulty for the 4B model is precise span boundaries in nested text.

5 Discussion

RQ1 Discussion: Abstraction Identification Reliability Our results confirm that relatively small LLMs (4B parameters) can identify requirement abstractions if provided with few-shot examples. Zero-shot performance is insufficient for reliable engineering use. The multi-agent approach aids significantly in distinguishing semantically similar tags like *Precondition* and *Condition* by isolating the logic extraction task.

RQ2 Discussion: Clarity and Completeness Benefits The structured decomposition provides tangible benefits for requirements quality. By exposing the semantic building blocks, analysts can quickly identify:

- **Missing actors**: Requirements without *Main_actor* often use passive voice.
- **Implicit conditions**: Absent *Precondition* tags may indicate unstated assumptions.

RQ3 Discussion: Nested Item Handling Complex requirements with multiple nested abstractions present the greatest challenge. The segmentation step proves valuable, but segmentation errors can propagate.

Baseline vs. Multi-Agent Trade-offs The single-agent baseline is faster and cheaper, but the multi-agent system offers higher precision for complex tags. For simple requirements, the Baseline is sufficient (F1 0.73), but for safety-critical logic, the Multi-Agent's superior handling of Preconditions (0.58 vs 0.50) is valuable.

Limitations

- Single domain (study room booking) limits generalizability.
- Use of a small model (Qwen3 4B) sets an upper bound on performance; larger models (70B+) would likely exceed F1 0.85.
- Exact substring matching is a strict metric that may penalize valid synonyms.

6 Conclusion

This study addressed three research questions about LLM-based semantic analysis of software requirements.

Answers to Research Questions:

RQ1 (Abstraction Identification): Yes, LLMs can identify abstractions, provided few-shot prompting is used. Performance correlates with the distinctiveness of linguistic markers.

RQ2 (Clarity and Completeness): Yes, semantic decomposition improves clarity by making structure explicit and improves completeness by highlighting missing semantic roles.

RQ3 (Nested Items): LLMs can manage nested items, though with varying success. Segmentation into clauses is a successful strategy for handling complexity.

References

Aybüke Aurum and Claes Wohlin, editors. 2005. *Engineering and Managing Software Requirements*. Springer.

Daniel M. Berry, Ricardo Gacitua, Peter Sawyer, and Sri Fatimah Tjong. 2012. The case for dumb requirements engineering tools. In *Requirements Engineering: Foundation for Software Quality (REFSQ 2012)*, pages 211–217. Springer.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Jean-Michel Bruel, Sophie Ebersold, Florent Galinier, Manuel Mazzara, Alexander Naumchev, and Bertrand Meyer. 2021. The role of formalism in system requirements. *ACM Computing Surveys (CSUR)*, 54(5):1–36.

Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. 2017. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213.

Elizabeth Hull, Ken Jackson, and Jeremy Dick. 2005. *Requirements Engineering*, 2 edition. Springer.

Leonid Kof. 2005. Natural language processing: Mature enough for requirements documents analysis? In *International Conference on Application of Natural Language to Information Systems*, pages 91–102. Springer.

Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. 2009. Easy approach to requirements syntax (ears). In *2009 17th IEEE International Requirements Engineering Conference*, pages 317–322. IEEE.

Johannes J. Norheim, Eric Rebentisch, Dekai Xiao, Lorenz Draeger, Alain Kerbrat, and Olivier L. de Weck. 2024. Challenges in applying large language models to requirements engineering tasks. *Design Science*, 10:e16.

Chris Rupp. 2014. *Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil*, 6 edition. Hanser.

Kevin Ryan. 1993. The role of natural language in requirements engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 240–242. IEEE.

Rashad Sonbol, Ghazwan Rebdawi, and Nada Ghneim. 2022. The use of nlp-based text representation techniques to support requirement engineering tasks: A systematic mapping review. *IEEE Access*, 10:62811–62830.

Axel van Lamsweerde and Emmanuel Letier. 2000. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008.

Eric Yu. 1997. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pages 226–235. IEEE.

Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J. Letsholo, Muideen A. Ajagbe, Edaena E. Chioasca, and Riza T. Batista-Navarro. 2022. Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys*, 54(3):55:1–55:41.