Generating Software Requirements Through Abstractions

First Author	Second Author	Third Author	Fourth Author	Fifth Author
Michele Curci	Francesco Borrelli	Marco Ravaioli	Lorenzo Durante	Shahrzad Shivaei
s337117	s337182	s345349	s347143	s343616

Abstract

As software systems grow in complexity, the manual transformation of natural language requirements into structured semantic models remains a significant bottleneck in the software development life-cycle. This report investigates the efficacy of Large Language Models (LLMs)—specifically the Meta-Llama-3.1-8B-Instruct architecture—in automating the extraction of key semantic abstractions such as actors, actions, and logical constraints. We implement and evaluate three distinct architectural patterns using 4-bit quantization to optimize for computational efficiency on local hardware: (1) a Single-Agent model utilizing zero-shot, one-shot, and few-shot prompting to extract all components in a single pass; (2) a Multi-Agent 8-prompt model that assigns a specialized agent to each semantic category; and (3) a Sequential 3-Agent Pipeline (Entity, Action, and Logic agents) that utilizes Context Injection and Smart Filtering to maintain semantic coherence. Our methodology addresses common LLM limitations, such as JSON hallucination and redundant sub-string extraction. Preliminary observations focus on the trade-offs between the execution speed of the single-pass architecture and the granular precision of the multi-agent pipelines. The results provide empirical evidence for the selection of agentic structures in automated requirements engineering, highlighting how distributed reasoning can mitigate information loss in complex logical specifications. The code can be found here.

1 Introduction

The field of Software Engineering relies heavily on the quality and clarity of software requirements. Traditionally, the process of transforming natural language (NL) requirements into structured semantic models is a labour-intensive manual task performed by requirements engineers. This process is prone to human error, inconsistency, and scalability issues, particularly in large-scale projects involving thousands of specifications. As software systems grow in complexity, there is an urgent need for automated tools that can accurately extract semantic abstractions, such as actors, system responses, and logical triggers from unstructured text.

Recent advancements in Large Language Models (LLMs), specifically the Llama 3.1 architecture, have demonstrated significant potential in Natural Language Processing (NLP) tasks. These models, trained on vast datasets, possess an inherent understanding of grammar, context, and domain-specific terminology. However, the application of LLMs to Requirements Engineering (RE) presents unique challenges: requirements often contain nested logic, passive voice ambiguities, and strict semantic definitions that a generic model may struggle to categorize correctly without specialized prompting strategies.

This study investigates and compares three distinct architectural approaches for semantic extraction using the Llama 3.1 8B Instruct model:

- Single-Agent (Standard) Approach: A unified prompt strategy where a single model instance is tasked with extracting all eight semantic categories (Purpose, Trigger, Condition, Precondition, Main actor, Entity, Action, and System response) in a single pass.
- Multi-Agent (Distributed) Approach: A 8-agent configuration where each extraction field is assigned to a specialized "expert" prompt. This minimizes context competition and allows the model to focus its attention on a single definition at a time.
- Tiered Multi-Agent (Pipeline) Approach: A 3-agent pipeline (Entity, Action, and Logic agents) that utilizes "Context Injection." This method mimics a human-like workflow where earlier findings (e.g., identified Entities) are fed into subsequent agents (e.g., Action Ana-

lyst) to improve semantic coherence and mitigate hallucinations.

2 Background

The quest to automate the extraction of semantic components from Natural Language (NL) requirements has its roots in linguistic rule-based systems. Early methodologies relied heavily on Part-of-Speech (POS) tagging and dependency parsing to map grammatical subjects to actors and verbs to actions (Bruegge Dutoit, 2009), While these methods established the groundwork for Requirements Modeling, they often lacked the flexibility to handle the inherent ambiguity and passive voice structures typical of industrial specifications (Pohl, 2010).

The introduction of the Transformer architecture (Vaswani et al., 2017) catalyzed a shift from rigid rule-based systems to probabilistic deep learning. In the context of Requirement Engineering (RE), Large Language Models have been shown to outperform traditional machine learning models in Named Entity Recognition (NER) and classification tasks due to their pre-trained semantic depth (Zhao et al., 2023). However, the "black box" nature of these models introduces risks of hallucination, necessitating structured prompting techniques to ensure output reliability.

The efficacy of LLMs is heavily dependent on Prompt Engineering (White et al., 2023). Research into "Chain-of-Thought" (CoT) prompting has demonstrated that breaking complex tasks into logical increments significantly improves reasoning (Wei et al., 2022). Our study extends this logic through Multi-Agent Orchestration, a concept where specialized agents collaborate to refine outputs. This mirrors the "Divide and Conquer" strategy in software architecture, reducing the cognitive load on a single model instance to prevent the omission of critical constraints like Preconditions or Triggers.

3 System overview

3.1 Architecture

The system is designed to empirically assess whether LLMs can reliably decompose natural-language software requirements into well-defined semantic abstractions. A linear pipeline architecture was intentionally chosen to isolate the abstraction-extraction capability of the LLM and to write them into a standardized JSON format. The architecture voluntarily excludes any kind of post-processing heuristic, rule based correction or validation model. This is done to ensure that all observed behaviors can be attributed only to the LLM behavior. The core component of the architecture is the LLM abstraction extractor, which in this case was chosen to be the Meta-Llama-3.1-8B-Instruct Model. The execution environment is Google Colab, using the NVIDIA T4 GPU provided. The Llama model is 4-bit Quantized to increase inference speed. Deterministic decoding (temperature = 0) and a fixed maximum number of output tokens of 512 were adopted to reduce output variance, limit hallucinations, and ensure structural consistency between runs. Since the LLM has a probabilistic nature, the system must incorporate a mechanism to ensure integrity of the output data, handling situations where the output is not in JSON format. Such errors are handled via exception management mechanisms recording the error type without interrupting the processing of the remaining requirements.

3.2 Utilized Dataset

This study utilized two datasets of software requirements. A large raw dataset of 250 requirements was initially considered. Although annotations were available, they contained significant noise and inaccuracies, including missing, redundant, and incorrect labels. Predictions were conducted on this dataset to assess model behavior with noisy ground truth conditions. Due to the high cost of manual requirements annotation, and the lack of an unambiguous and unique ground truth, a second, smaller dataset of 50 requirements was constructed. This represents a trade-off between size and annotation reliability, with an effort in trying to correct and harmonize the already existing annotations in a way more consistent with the definition reported below. However, as discussed later, having a unique and globally shared annotation pattern is one of the biggest problems of this task, keeping a subjective part and incline to annotator bias. Requirements were converted from their original CSV format into JSON, to generate a ground truth aligned with the LLM output structure. The dataset includes requirements exhibiting heterogeneous structures, different conditional clauses, implicit actors, single and multiple actions, to stress the most the LLM. The dataset defines eight different types of abstractions to decompose the requirement. While not exhaustive, the schema was designed to balance ex-

pressiveness and annotation feasibility, prioritizing abstractions that are both semantically meaningful and consistently identifiable. The abstractions are the following:

- Purpose: The reason why the functionality described by the software requirement needs to be implemented.
- Trigger: An event establishing a temporal context and a causal link that constrains the requirement's applicability. It is also a condition.
- Condition: Something that limits the scope of application of the requirement.
- Precondition: A condition that must hold in the requirement's context. A precondition is always a condition.
- Main Actor: The main user of the functionality described by the requirement. The main actor is often also an entity.
- Entity: Something involved in the actions described in the requirement. Can be both human or not (e.g. the system).
- Action: Something that happens in the scenario described by the software requirement.
- System Response: The behavior of the system in the described scenario. It is always an action.

The selected JSON schema represents every abstraction as a list of strings. This structure is necessary to accommodate the multi-label nature of the requirements, where a substring can cover multiple semantic roles. Finding or constructing a high quality ground truth dataset for software requirements is one of the most significant challenges in this field. Publicly available requirement datasets are extremely scarce and those that exist are not annotated for semantic decomposition. The primary obstacle in creating a baseline annotation is the non unique nature of the task: requirements are often ambiguous and a sentence can be decomposed in many different ways based on the individual interpretation. Moreover, different users can use different kinds of abstractions. This lack of a standardized and correct mapping makes establishing a ground truth problematic that leads to the objective of understanding if LLMs can provide consistent, structured and reasonable interpretations, even in the absence of a universally accepted decomposition.

3.3 Prompt Engineering Techniques

Fine tuning the model was prohibitive due to the lack of ground truth datasets, so a Prompt Engineering approach has been chosen. Two main approaches have been taken into consideration: Single Agent and Multi-Agent. In the Single Agent architecture with a single prompt the model has to correctly abstract the requirement.
In the Single Agent approach the system compares three levels of Prompt-Tuning: A Zero-Shot baseline, containing only the base instructions, to test the model's raw behavior. The Zero-Shot configuration works as a baseline to evaluate the model's capability to interpret and decompose software requirements without a specific domain knowledge.
A One-Shot Agent and a Few-Shot Agent that received respectively 1 and 3 samples to provide some requirement domain knowledge, trying to improve the ability to detect every abstraction correctly without providing excessive information, which could lead to overfitting of linguistic patterns. These configurations test the hypothesis of a minimal domain grounding that can significantly improve abstraction performances. The base instruction is a prompt given to every model to put constraints and instill domain knowledge. It is composed by three main levels:

- Role: Initially the prompt assigns the model a precise expert role in software requirements analysis.
- Extraction: The prompt aims to extract substring as they are found in the requirement, without paraphrasing, and provide the behavior to have in case of absence of correct abstraction. Furthermore it provides a complete and precise abstraction description.
- Formatting: Prompt reaffirms the output structure, explicitly forbidding any additional formatting or conversational output and provides the expected output JSON structure.

In the Multi-Agent the study developed two distinct Multi-Agent System architectures. In contrast to the single prompt approach, these architectures distribute the extraction of the eight semantic abstractions into specialized sub-tasks.

Sequential Context-Aware Pipeline (SCAP): For the first architecture we developed a pipeline with three specialized agents supported by few-shot learning:

- Semantic Entity Agent: Extracts Main Actors and Entities. It enforces a Mirror Rule (Actors are subsets of Entities) and Exclusion Rules to filter out temporal constraints and standalone metrics.
- Context Behavior Analyst: Extracts Action, System Response and Purpose. It employs Context Injection, embedding the Main Actors and the Entities into the prompt.
- Contrastive Logic Expert: Extracts Triggers, Precondition and Condition. It uses Contrastive Few-Shot Prompting to explicitly distinguish between Dynamic Requirements (with temporal triggers) and Static Capabilities (where the Trigger must remain empty).

The process concludes with a deterministic Coordinator module. An algorithm that enforces structural hierarchy (ensuring Triggers are subsets of Conditions) and applies Smart Atomic Filtering, which removes redundant text.

Modular Independent Agents (MIA): This Architecture implements a Flat modular strategy that prioritizes task isolation over sequential dependency. This approach atomizes the complex extraction process into eight distinct, parallel inference calls, one for each specific semantic field (Main Actor, Entity, Action, System Response, Purpose, Trigger, Condition, Precondition).

The architecture operates on the following principles:

- Independent Inference: Each agent function is completely isolated. For instance, the Action extraction agent analyzes the requirement text without any awareness of the Entities identified by other agents.
- Specialized Prompting: Every agent utilizes a highly specific system prompt focused only on its assigned abstraction. The prompt enforces definition-level constraints, comparable to those used in the previous architecture, but does not include any instructions for sharing context or information across agents.
- Error Containment: Decoupling the agents prevents cascading errors. A hallucination or omission in the entity extraction phase remains contained and does not propagate to corrupt the logic or action extraction phases.
- Aggregation: The results produced by the eight separate agents are collected and merged into the final JSON structure only at the end of the process.

This architecture serves as a rigorous control group, designed to experimentally isolate and evaluate the specific impact of sequential context sharing (SCAP) versus a purely parallelized, independent approach.

4 Experimental results

In this section, we present the results of the evaluation performed on the Llama-3 architectures. The primary analysis focuses on the Cleaned Dataset to ensure metric reliability, followed by a robustness comparison with the original noisy data.

4.1 Quantitative Performance

The primary evaluation was conducted on the Cleaned Dataset (50 requirements) using an optimized similarity threshold of $\tau = 0.50$. We decided to adopt the SBERT semantic metric because, as shown in Figure 1, exact string matching is too penalizing for this task. The semantic evaluation allows us to recognize correct extractions even when the model uses synonyms or different phrasing.

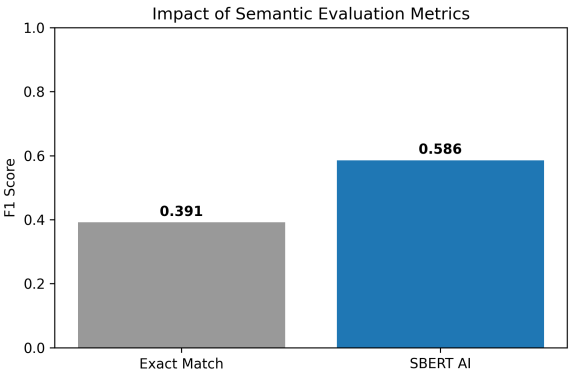

Figure 1: Impact of Semantic Metrics. SBERT evaluation is necessary to avoid penalizing correct paraphrases compared to Exact Match.

Table 1 summarizes the performance of the tested architectures. The SCAP model achieved the highest F1-Score (0.586), outperforming both the One-Shot (0.567) and the Zero-Shot baseline (0.472). A closer look reveals that the MIA configuration prioritized Recall (0.683), demonstrating

the architecture's capability to minimize omissions, albeit with a trade-off in Precision.

Model	Strategy	Prec.	Rec.	F1
Zero-Shot	Baseline	0.671	0.364	0.472
One-Shot	In-Cont Learn	0.686	0.483	0.567
Few-Shot	In-Cont (5-Shot)	**0.723**	0.425	0.535
Multi-agent	SCAP	0.561	0.612	**0.586**
Multi-agent	MIA	0.419	**0.683**	0.519

Table 1: Performance on Cleaned Dataset ($\tau = 0.50$). The SCAP strategy achieves the best balance (F1), while MIA dominates in Recall.

This result is visualized in Figure 2. The Multi-Agent bars demonstrate superior Recall compared to the Single-Agent approaches.

Figure 2: Performance comparison. Multi-Agent architectures demonstrate superior Recall compared to Single Agent baselines.

4.2 The Precision-Recall Trade-off

The metrics highlight a fundamental difference in how the models work. The One-Shot model exhibits "Structural Overfitting": it tends to copy the structure of the prompt example. This results in good Precision (0.686) but low Recall (0.483), as it misses requirements that do not structurally match the example.

On the other hand, the MIA performs "Generative Reasoning". It rewrites and decomposes the requirements. This ensures high Recall (0.683), but it is penalized in Precision (0.419) because the words differ from the Ground Truth. This behavior is confirmed by the ROC analysis in Figure 3. At the optimal threshold of 0.50, the Recall (red line) is dominant, confirming the generative nature of the model.

4.3 Robustness Analysis

To understand which model is truly better, we compared these numbers with the results on the Original (Noisy) Dataset. This comparison revealed a

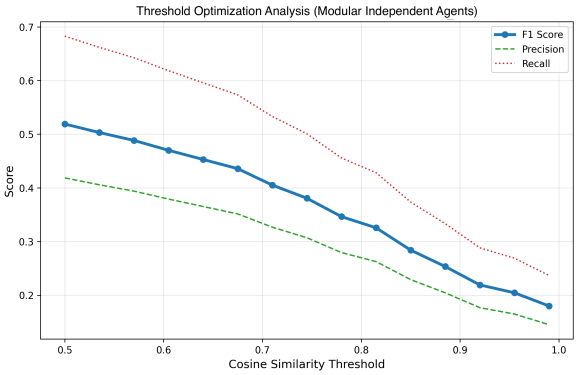

Figure 3: Threshold Optimization Analysis. At $\tau = 0.50$, the MIA shows dominant Recall, confirming its tendency to rephrase rather than copy.

significant insight regarding the robustness of the architectures.

- On the **Cleaned Dataset** (Figure 2), the Multi-Agent SCAP (0.586) slightly outperforms the One-Shot model (0.567). Both models perform adequately when the data is high-quality.
- On the **Original Dataset** (noisy and unstructured), the scenario changes drastically. As shown in Figure 4, the One-Shot performance collapses (F1 drops to 0.437) because the model lacks consistent examples to mimic.

In contrast, the Multi-Agent SCAP architecture maintains a slightly better performance (F1 0.506). This proves that the Multi-Agent approach is much more robust in real-world scenarios characterized by high entropy. While simple models fail without perfect inputs, the agentic workflow successfully identifies requirements even when the input data is messy.

4.4 Granular Analysis & Limitations

We also analyzed which categories are harder to extract. As shown in Figure 5, concrete elements like Entities and Actions are easier to identify. Abstract logic, such as Triggers and Preconditions, is much harder due to complex temporal dependencies.

Finally, the qualitative analysis of the errors showed that many "Hallucinations" of the MIA architecture were actually useful. For example, in *Req_0* ("The product shall be available during normal business hours"), the model decomposed the single sentence into three specific requirements (e.g., "Ensure the product is accessible...", "Maintain high system availability..."). While SBERT counts this as a mismatch against a concise Ground

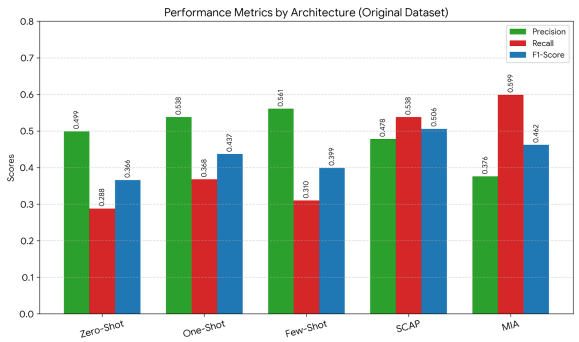

Figure 4: Performance comparison on Original (Noisy) Dataset. Compared to Figure 2, the One-Shot baseline collapses, while Multi-Agent SCAP maintains the lead in F1-Score, demonstrating robustness against data noise.

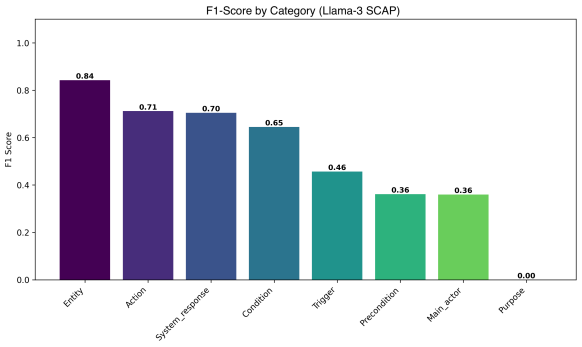

Figure 5: SCAP Category Breakdown. There is a clear gap between concrete entities (high performance) and abstract logic (low performance).

Truth, for a requirements engineer this represents an improvement in specification clarity. SCAP pipeline is never able to predict correctly the purpose of the requirement (empty output or random characters). On the contrary, as illustrated in Figure 6 MIA architecture is able achieve a discrete F1-Score in all categories, identifying purposes but losing accuracy on actions, classifying only the grammatical verb as actions, instead the action itself.

4.5 Efficiency

The trade-off for robustness is computational cost. As illustrated in Figure 7, we analyzed the latency on the original dataset. The transition from Single-Agent baselines to the **SCAP** architecture resulted in a latency increase of approximately 30% (**9.7s** vs **12.6s** per requirement).

However, the fully decoupled **MIA** architecture proved to be significantly more expensive, reaching **20.5s** per requirement (+111% latency). This confirms that the SCAP architecture represents the

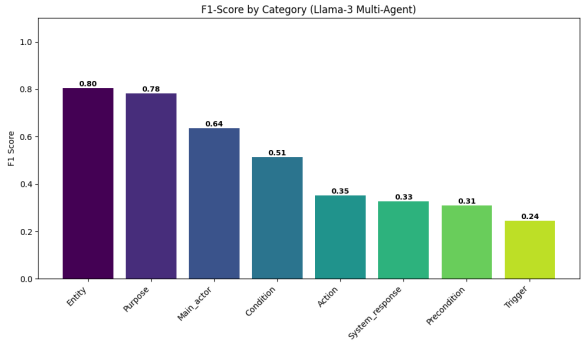

Figure 6: MIA Category Breakdown.

optimal Pareto choice: it achieves the highest F1-Score (0.506) with a manageable latency increase, whereas MIA incurs a high computational penalty for lower overall precision.

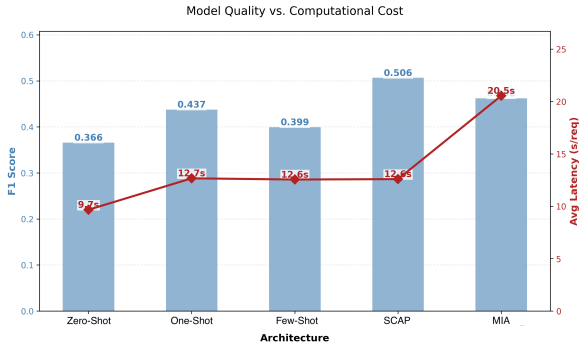

Figure 7: Efficiency Analysis (Original Dataset). The SCAP model offers the best trade-off, improving F1 with moderate latency (12.6s), while MIA suffers from higher computational cost (20.5s).

5 Conclusion

Our results demonstrate that while Single-Agent models achieve high Precision via structural imitation, they are prone to failure in the presence of data noise. In contrast, the Sequential Context-Aware Pipeline (SCAP) emerged as the optimal architecture, achieving the highest F1-Score (0.586) and maintaining stability on noisy datasets where One-Shot baselines collapsed. We identified a critical trade-off: Modular Independent Agents (MIA) maximize Recall (0.683) through generative reasoning but at a 111% latency penalty. SCAP provides a superior Pareto balance, leveraging context injection to mitigate information loss with moderate computational overhead. Future work should focus on fine-tuning for abstract logic categories (Triggers/Preconditions), which remain the primary bottleneck for autonomous RE.

References

[1] Bruegge, Dutoit, 2009, Object-Oriented Software Engineering, Using UML, Patterns, and Java, Available at: https://womengovtcollegevisakha.ac.in/departments/Object-Oriented%20Software%20Engineering%20Using%20UML,%20Patterns,%20and%20Java%20(%20PDFDrive%20).pdf

[2] Pohl, 2010, Requirements Engineering Fundamentals, Available at: https://www.bbau.ac.in/dept/dit/TM/requirementsengi.pdf

[3] Vaswani et al., 2017, Attention Is All You Need, Available at: https://arxiv.org/pdf/1706.03762

[4] Zhao et al., 2023, A Survey of Large Language Models, Available at: https://arxiv.org/pdf/2303.18223

[5] White et al., 2023, A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT, Available at: https://arxiv.org/pdf/2302.11382

[6] Wei et al., 2022, Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, Available at: https://arxiv.org/pdf/2201.11903