FrameLLM for Requirements Generation: A Framework for Reducing Prompt Dependency and Improving Requirement Clarity and Completeness

Mahwish Kundi, Faraz Ahmad, Rosemary Monahan
[1]Maynooth International Engineering College, Maynooth University, Ireland
[2] Abdul Wali Khan University Mardan, Pakistan
[3] Computer Science Department, Maynooth University, Ireland
mahwish.kundi@mu.ie
farazahmad02462@gmail.com
Rosemary.Monahan@mu.ie

Abstract

The adoption of Large Language Models (LLMs) in requirements generation offers significant opportunities for automatic requirements engineering. However, unstructured stakeholder inputs and heavy prompt reliance in LLM-based approaches can often result in inconsistent, incomplete, and ambiguous requirements. To address these challenges, the current study integrates NLP-based preprocessing with FrameNet semantics to guide LLM-based requirement generation. The evaluation considers both quantitative measures (number of generated requirements, execution time) and qualitative aspects (semantic completeness, domain relevance, consistency, and traceability). The initial results show that FrameNet-guided LLMs (LLaMA) effectively generate clear, complete, and consistent requirements while reducing prompt dependency. Across multiple executions of five representative Autonomous Driving System (ADS) requirements, the proposed FrameNet-guided LLM (F_LLM) consistently produced stable, semantically grounded, and traceable requirements, demonstrating improved completeness and efficiency compared to a baseline LLM. The final requirements are generated in the IEEE-830 standard format, making them ready for verification and implementation.

Introduction

The frequent use of LLMs in requirements engineering introduces both opportunities and challenges. LLMs can help to generate, refine, and analyze requirements; however, the generated text is often dependent on the structure and quality of the prompts. This dependency can cause different stakeholders to receive different outputs (text-based requirements) even when using the same input (prompt), which ultimately results in inconsistencies, missing information, and an ambiguous set of requirements. To address these challenges, the proposed research focuses on the following research questions:

RQ1: How can FrameNet-guided LLMs reduce dependence on user-provided prompts?
RQ2: How effective is the proposed FrameNet + LLM approach in generating a consistent, complete, and standardized set of requirements?
RQ3: How can LLMs generate traceable requirements from the same input across multiple attempts?

In the current study, NLP is applied primarily as a preprocessing layer to support FrameNet-based frame detection and subsequent LLM-guided requirement generation. FrameNet has previously been shown to be an effective resource for eliciting requirements, helping to produce more complete, consistent, and coherent requirements (Fontenelle 2003; Alhoshan, Batista-Navarro, and Zhao 2018; Kundi and Chitchyan 2017). Additionally, FrameNet has been shown to be a promising resource for NLP tasks (Rai, Croce, and Basili 2025; Karou Diallo and Zouaq 2025), particularly for semantic role identification and structured knowledge extraction from natural language. Similarly, LLMs are effective in generating, refining, and analyzing requirements (Nadăș, Dioșan, and Tomescu 2025), but their output is often highly dependent on the quality of the prompts (Ellsel and Stark 2025; Narayan, Kumar, and Chacko 2025; Kauffmann 2025). By combining FrameNet frames with LLaMA, our approach uses FrameNet's semantic structures to guide requirement generation, improving clarity, completeness, and consistency. Preliminary results demonstrate that FrameNet-guided LLaMA generation reduces dependence on prompts, enhances traceability, and effectively captures missing or ambiguous information. The final requirements are produced in the IEEE-830 standard format (Okamoto and Kusumoto 2025), making them easy to use for verification and implementation.

To empirically investigate these research questions, we conducted a controlled experimental study on five representative requirements from an Autonomous Driving System (ADS) case study, (Kundi, Ahmad, and Monahan 2025). The study compares a baseline prompt-driven LLM with the proposed FrameNet-guided LLM (F_LLM) across multiple executions of the same inputs. The evaluation considers both quantitative measures (such as the number of generated requirements and execution time) and qualitative aspects (including semantic completeness, domain relevance, consistency, and traceability). The results show that F_LLM

consistently generates stable and meaningful requirements while taking less time than the baseline LLM. By systematically recovering missing semantic roles through FrameNet core and non-core frame elements, the proposed approach minimizes prompt dependency and limits the introduction of irrelevant or fabricated content. These findings demonstrate the effectiveness of combining FrameNet with LLMs for generating consistent, complete, and standardized requirements, providing initial evidence of the feasibility and benefits of the proposed framework.

Background Work

Large Language Model Meta AI (LLaMA)

The Large Language Model Meta AI (LLaMA) represents a family of transformer-based models developed for research. Ollama provides a free framework to run LLMs on local machines, keeping data private and secure. LLaMA 3.2[1] is available in 1B and 3B parameter versions. It is multilingual and designed to run efficiently locally, without requiring high-end hardware. These qualities make it suitable for research tasks such as automated text generation, summarization, and large-scale text analysis. The framework also provides a lightweight interface and tools for managing model parameters and prompting styles. Its large-scale multilingual pretraining makes LLaMA 3.2 capable of understanding and generating coherent text. Recently, LLMs have been applied in requirements engineering to assist with requirement elicitation, analysis, and generation (Arvidsson and Axell 2023; Khan, Qayyum, and Dar 2025).

FrameNet

Charles J. Fillmore, a linguist, originally introduced the concept of a frame, which later became the foundation for the FrameNet project at the University of California, Berkeley, initiated in the 1990s. FrameNet is an online lexical database comprising Frames, Frame Elements (FEs), and Lexical Units (LUs).FrameNet also provides sentence-level annotations and examples, which help clarify word meanings in different contexts. Instead of defining words individually, FrameNet provides relevant concepts and information for each word, making it easier to understand the exact meaning of a single word. As a result, FrameNet has been shown as a key source for overcoming ambiguity and missing information in requirements engineering (Kundi and Chitchyan 2017; Anwar et al. 2024).

Literature Review

In recent years, the use of LLMs in software requirements analysis has gained significant attention. (Ma et al. 2025) discusses LLM performance in requirements engineering by proposing Requirement-Oriented Prompt Engineering (ROPE). They argue that LLM performance depends more on clear, well-structured requirements than on simple prompt tricks. Their experiments show that training users to write better requirements improves both requirement quality and LLM outputs. However, their approach mainly relies on human training, whereas our work focuses on automatic semantic grounding using FrameNet frames to enrich requirements. (Krishna et al. 2024) used LLMs (GPT-4 and CodeLlama) to generate Software Requirement Specification (SRS) and compared them with human-written specifications. Their results show that detailed prompts and contextual instructions can produce SRS drafts that are more accurate than those written by entry-level engineers. However, the approach still depends on the general text generation of LLM, which can produce inconsistencies and incorrect requirements.

Similarly, (Ren, Nakagawa, and Tsuchiya 2024) proposed a new method for generating models from user reviews of mobile apps using Latent Dirichlet Allocation (LDA) and GPT-4 guided by structured prompts and examples. It stated that traditional clustering approaches struggle to achieve good accuracy and sentiment classification. In general, the study reveals that prompt-guided LLMs produce enhanced requirements and goal models, which provide developers with greater insight.

Another paper introduces REQINONE, an LLM-based agent (GPT-4, Llama 3, and DeepSeek R1) to automate SRS generation and classify requirements as functional or non-functional. GPT-4 achieved high accuracy, while Llama-3 produced clearer requirements (Zhu, Cordeiro, and Sun 2025). Further advancements and optimizations to the models, such as GPT-4/5, Gemini, and locally server-accessible LLaMA-3, made them robust, flexible, and reliable because they are trained on large corpora and a high-instruction-tuned process (Ouyang et al. 2022). To achieve better results, writing effective prompts can be challenging for non-experts who lack knowledge of how to use LLMs. This leads to the importance of developing better End-User Prompt Engineering (EUPE) training (Zamfirescu-Pereira et al. 2023). LLMs present strong potential in Requirements Engineering (RE), but often raise challenges of correctness, semantics, soundness, and trustworthiness. To overcome these issues with LLMS, A. Ferrari proposes two roadmaps: one is the use of Formal Methods (FM) to validate output, and the other utilizes LLMs to make FMs more accessible (Ferrari and Spoletini 2025).

Methodology

This section describes the step-by-step methodology of F_LLM for generating software requirements, combining NLP-based preprocessing, FrameNet semantic frame detection, and guided-LLM requirement generation.

Input Requirement

Initially, a single requirement statement is provided as input to the system. Such requirements are often informal, ambiguous, or incomplete. Let R denotes the set of input requirements.

[1]https://ollama.com/library/llama3.2

Text Preprocessing and Token Extraction

The input requirement is processed using standard NLP (NLTK toolkit) steps:

Sentence Split and Tokenization: Given an input requirement R, standard natural language preprocessing is first applied using the spaCy toolkit. The requirement is segmented and converted into a sequence of tokens. Let T denote the resulting set of tokenized words: 1, where 'wi' represents an individual token extracted from the requirement text.

$$T = \{w_1, w_2, \ldots, w_m\} \tag{1}$$

POS Tagging and Lemmatization: Verbs, nouns, and other linguistic features are identified and lematized into their base forms 2, based on the output of T from formula 1. The lematized tokens are combined into a set 'L' that denotes the resulting set of lemmas. 3.

The requirement R refers to the complete original sentence, while tokens T and lemmas L are intermediate linguistic representations derived from it. Tokens preserve the surface structure of the text, whereas lemmas provide normalized lexical units that support semantic abstraction and subsequent frame identification.

$$l_i = \text{lemma}(w_i) \tag{2}$$

$$L = \{l_1, l_2, \ldots, l_n\} \subseteq T \tag{3}$$

Conversion into embeddings: The requirement is then embedded into embeddings (vectors) through the embedding model of NLTK, as shown in the formula 4 where 'r' represents embeddings and 'E' represents the embedded requirement.

$$r = E(requirement) \tag{4}$$

Extraction of Key Tokens: Particular verbs (actions) and nouns (entities) are extracted from the lemmatized set 'L', yielding a subset of tokens. These tokens are used to trigger the relevant frames

Automatic Frame Detection

Identify Candidate Frames: Using the embedded set of tokens (verbs and nouns), the system automatically identifies possible relevant frames. A suitable frame-selection technique is applied to match tokens with their possible frames. The formula represents the set of frames 'F' concerning its token 'l' 5 .

$$F(l) = \{f_1, f_2, \ldots, f_k\} \tag{5}$$

Predict Frames from Context: Frames are predicted based on the contextual meaning of the tokens. Because different tokens can evoke different frames, and a single frame can be triggered by multiple tokens. As a result, the system may retrieve several candidate frames.

Retrieve Frame Definition and Elements: For each detected frame, the proposed model collects the frame definition, as well as its core and non-core Frame Elements (FEs), from the official FrameNet resource, as shown in the Figure of the interface 1.

Rank or Disambiguate Frames: When multiple frames are selected associated with the same token(s), the system applies a disambiguation mechanism to determine which candidate frames are most relevant to the requirement 8 using the frame definition 'fi' 7 or name 'di' 6. After this, the similarity score is computed using frame definition, its elements, and text (requirement) embeddings to select a contextually correct frame, where 'si' represents the similarity score 9. As a result, only the top-k frames are extracted based on the similarity score for the final set of requirements generation. 10.

$$d_i = name_i \,\|\, definition_i \tag{6}$$

$$f_i = E(d_i) \tag{7}$$

$$F_{candidate} = \bigcup_{l \in L} F(l) \tag{8}$$

$$s_i = \cos(r, f_i) = \frac{r \cdot f_i}{\|r\| \, \|f_i\|} \tag{9}$$

$$TopK = arg\,topk\{s_i\} \tag{10}$$

Produce Final Frame List: The output of this step provides a list of the most relevant FrameNet frames that are semantically and contextually relevant to the text. The user can choose to utilize all detected frames or select specific frames for the subsequent processing steps. This will ensure that the frame is represented only once in the final set 11.

$$f_j \in F_{candidate} \iff f_j \notin F_{candidate} \tag{11}$$

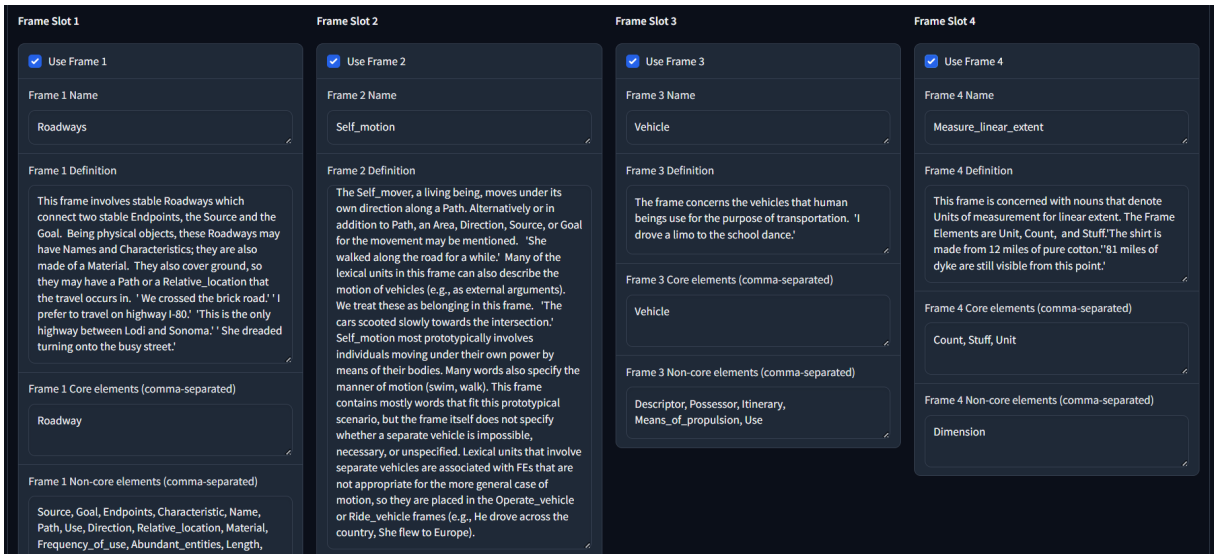

Figure 1: Frame Extraction and Selection for the R01 requirement

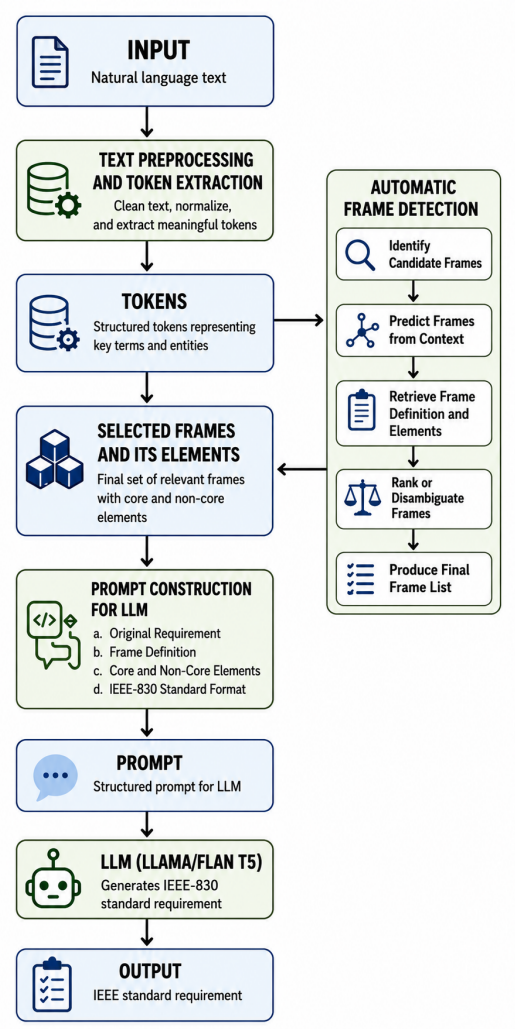

Figure 2: F_LLM for Requirement Generation

Prompt Construction for F_LLM

A prompt template is prepared for the F_LLM (Ollama 3.2 and Flan T-5), which includes the detected frame definitions and a list of core and non-core frame elements with their definitions. Additionally, the template provides instructions for the F_LLM to generate well-structured requirements in the IEEE-830 standard format, utilizing all core and non-core Frame Elements (FEs). The F_LLM receives this structured FE set along with the original requirement context. Using the prompt, the model is guided to reformulate or generate requirements that are complete, consistent, and standardized according to the IEEE-830 format.

Results and Discussion

This section presents the comparative results of requirement generation using a baseline LLM and the proposed F_LLM framework. The complete generated outputs for all runs are provided [2,3].

The experimental study was conducted on five textual requirements from ADS. Each requirement was processed three times using both F_LLM and the baseline LLM. For each requirement, we recorded the number of generated requirements and the execution time for each generation. We further report qualitative observations on semantic completeness, domain relevance, and consistency.

The experimental process begins by providing the textual inputs to both the baseline LLM and the F-LLM. The baseline LLM relies on prompt-based input, resulting in outputs that may vary with changes in prompt wording. In contrast, the F_LLM framework utilizes the structure of FrameNet

[2]https://drive.google.com/file/d/1ey2Ib5MT9Nq-gypELtzaYRwddeZ2IVoG/view?usp=sharing

[3]https://drive.google.com/file/d/1yPAv4auvFsrc0mik7evfSy1X_ZW1eYjL/view?usp=sharing

REQ ID	Description
R01	The system shall detect and recognize lane markings (solid, dashed, double) in real time to maintain lane discipline.
R02	The system shall detect pedestrian crosswalks (zebra crossings, colored markings) and adjust vehicle behavior accordingly.
R03	The system shall identify signalized junctions (traffic lights) and obey corresponding traffic signals.
R04	The system shall identify non-signalized junctions (stop/yield signs, roundabouts) and make safe decisions based on right-of-way rules.
R05	The system shall maintain an internal representation (map) of all detected physical infrastructure elements for navigation and decision-making.

Table 1: Descriptions of selected requirements (R01-R05) used for execution.

and leverages the core and non-core frame elements of Frame to guide and constrain the generation process. This structured guidance reduces dependence on prompt phrasing and enables the systematic identification of missing semantic roles.

The proposed framework effectively identifies and introduces requirements that were not explicitly stated in the initial inputs. By using frame elements, F_LLM generates requirements that are semantically linked to specific frame roles, ensuring that the generated content remains relevant and structured. As a result, the baseline LLM and F_LLM produce noticeably different outputs. F_LLM produces more consistent, semantically grounded, and purpose-aligned requirements across multiple executions in comparison to baseline LLM.

The execution process used five selected representative requirements (R01 to R05) to carry out tests as shown in Table 1. The requirements define essential testing scenarios, which include lane marking detection (R01), pedestrian crosswalk detection (R02), signalized junction identification (R03), non-signalized junction handling (R04), and internal map maintenance of physical infrastructure elements for navigation (R05).

The comparative generation results for the five selected requirements are presented in the tables below. Qualitative review shows F_LLM's additional items map to explicit frame roles (conditions, failure modes, actor responsibilities); Conversely, the baseline LLM mainly yielded surface decompositions and introduced occasional irrelevant or tangential specifics.

Table 2 shows that the baseline LLM produced 5 to 7 requirements across different runs in an average of 30 seconds, whereas F_LLM consistently produced 44 requirements. F_LLM generation time was comparable or lower (23–25s) despite the much larger output, indicating efficient, structured generation. Figure 3 provides a detailed visualization of the results presented in Table 2 for R01.

Metric	LLM	F_LLM	LLM	F_LLM	LLM	F_LLM
Execution	E1		E2		E3	
Generated Requirements	7	44	5	44	7	44
Execution Time	28 s	25 s	24 s	23 s	48 s	25 s

Table 2: Performance comparison of LLM and FrameLLM for R01

Metric	LLM	F_LLM	LLM	F_LLM	LLM	F_LLM
Execution	E1		E2		E3	
Generated Requirements	10	26	10	26	13	26
Execution Time	1 min 5 s	10 s	1 min 5 s	11 s	1 min 47 s	13 s

Table 3: Performance comparison of LLM and FrameLLM for R02

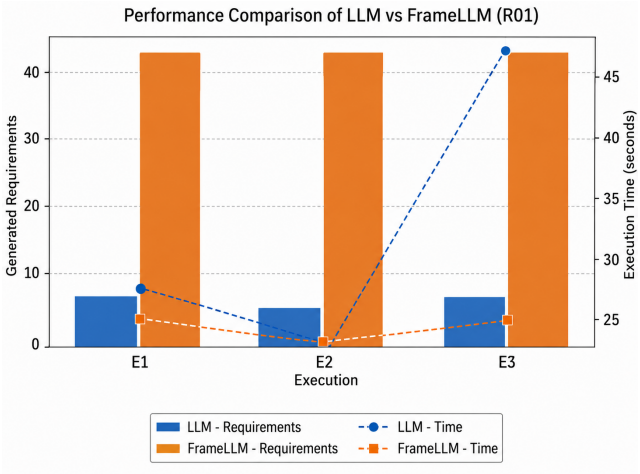

Figure 3: Generated Requirements performance Comparison of F_LLM and LLM of R01

Table 3 reports 10 to 13 requirements for the baseline LLMs versus a stable 26 requirements for the execution time of F_LLM (10-13 s) compared to the LLMs, which are longer (1-minute runs). This indicates the higher computational efficiency of FrameLLM compared to the baseline LLMs. The additional F_LLM requirements describe clear behavioral responses and contextual conditions, while the baseline LLM often failed to specify how vehicle behavior should be modified and sometimes produced unrelated content. Figure 4 illustrates a detailed visual representation of the results reported in Table 3 for R02.

The Table 4 indicate that the baseline LLM produced 10,12 and 14 requirements, while F_LLM consistently produced 27 requirements across three runs. Execution times differ significantly; the LLM requires an average of 1 minute and 20 seconds, whereas FrameLLM completes the task in 14–16 seconds. Moreover, FrameLLM's outputs consistently cover frame-derived elements such as signal states, timing, and actor coordination, which are essential for the scenario described in R04. In contrast, LLM outputs are more variable and often include irrelevant or loosely

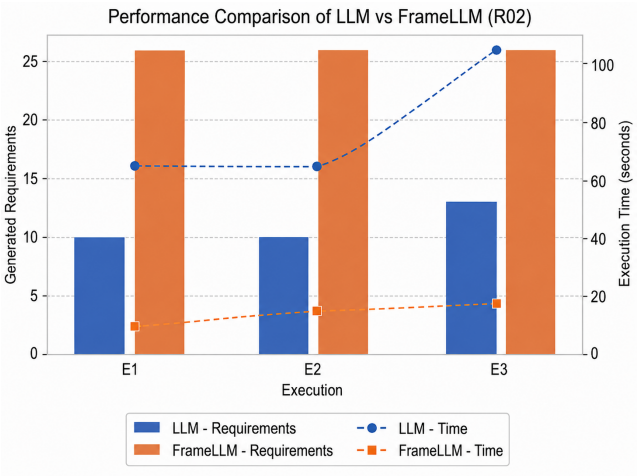

Figure 4: Generated Requirements performance Comparison of F_LLM and LLM for R02

Metric	LLM	F_LLM	LLM	F_LLM	LLM	F_LLM
Execution	E1		E2		E3	
Generated Requirements	14	27	10	27	12	27
Execution Time	1 min 20 s	15 s	1 min 25 s	16 s	1 min 24 s	14 s

Table 4: Performance comparison of LLM and FrameLLM for R03

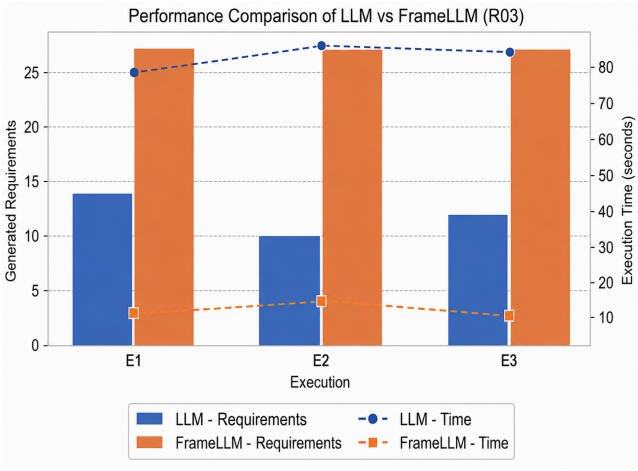

Figure 5: Generated Requirements performance Comparison of F_LLM and LLM for R03

Metric	LLM	F_LLM	LLM	F_LLM	LLM	F_LLM
Execution	E1		E2		E3	
Generated Requirements	3	43	18	43	20	43
Execution Time	40 s	22 s	44 s	20 s	41 s	16 s

Table 5: Performance comparison of LLM and FrameLLM for R04

related information. Figure 5 presents a comprehensive visualization of the findings summarized in Table 4 for R03.

Table 5 shows wide variability for the baseline LLM (3, 18, 20 requirements) versus a stable 43 requirements for F_LLM. F_LLM again produced results much faster (16–22s) than the variable LLM times (40–44s). The expanded output from F_LLM fills in missing semantic roles, such as right-of-way rules, decision conditions, and failure behaviors. In contrast, the baseline LLM sometimes generates too few or too many requirements across different runs and often adds unrelated content that reduces the clarity of the original requirement. The results for R04 shown in Table 5 are visually represented in detail in Figure 6.

Finally, Table 6 reports that the baseline LLM generated 20–26 requirements while F_LLM produced a stable 18 requirements. Despite producing fewer items for this case, F_LLM maintained fast generation (10–12 s) and produced focused, frame-justified requirements (map content, update policies, consistency checks). The larger but variable output of the baseline LLM again included many peripheral or implementation-style details that were not semantically present in the original requirement. A detailed graphical representation of the results listed in Table 6 for R05 is shown in Figure 7.

Across all five requirements, F_LLM delivered a stable, consistent set of requirements that are enriched with FrameNet core and non-core elements context and were produced with low execution time. The baseline LLM showed higher variability, longer average execution time, and a recurring tendency to add irrelevant or fabricated content rather than systematically recover missing semantic roles. These patterns support the claim that F_LLM reduces prompt dependency and produces more semantically coherent, testable, and traceable requirements.

Conclusion

The results of the current study demonstrate that combining FrameNet with an LLM improves the quality of automatically generated requirements. The proposed F_LLM framework reduces the dependence on prompt wording. It produces more consistent outputs by using core and non-core frame elements to identify missing semantic roles. These roles are often overlooked by baseline prompt-driven LLMs. The results of the experimental study show that F_LLM consistently produces stable, meaningful, and

Metric	LLM	F_LLM	LLM	F_LLM	LLM	F_LLM
Execution	E1		E2		E3	
Generated Requirements	20	18	26	18	22	18
Execution Time	52 s	10 s	53 s	12 s	50 s	12 s

Table 6: Performance comparison of LLM and FrameLLM for R05

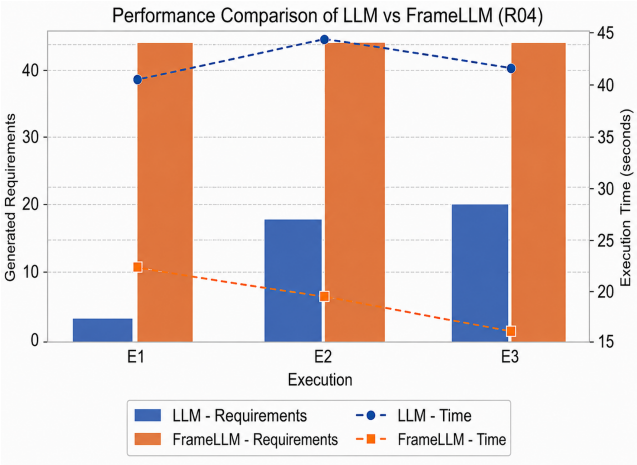

Figure 6: Generated Requirements performance Comparison of F_LLM and LLM for R04

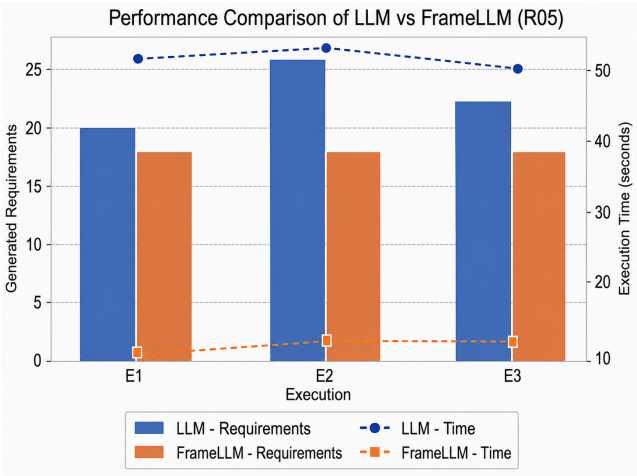

Figure 7: Generated Requirements performance Comparison of F_LLM and LLM for R05

well-structured requirements sets as compared to baseline LLM. For example, in the case of R01, the baseline LLM frequently produces outputs that are inconsistent across multiple runs (e.g., 7, 5, and 7 requirements generated in E1, E2, and E3, respectively), with execution times of 28, 24, and 48 seconds, and often includes irrelevant or unnecessary content. On the other hand, F_LLM consistently generates more structured and semantically grounded requirements (44 requirements in each run) while achieving lower execution times of 25, 23, and 25 seconds for E1, E2, and E3, respectively. Additionally, formatting the requirements according to the IEEE-830 standard enhances readability, traceability, and ease of verification and implementation. Although the findings are encouraging, this work has some limitations. The evaluation is based on a small number of requirements and a single application domain. Future work should evaluate the framework on larger datasets, additional engineering domains (e.g., healthcare), and integrations of FrameNet with different LLMs (e.g., GPT-based models). Furthermore, the results should be supported by empirical user studies, and the key elements for formal verification of AI-based systems should also be explored. Overall, this research demonstrates that FrameNet-guided LLMs offer a practical and effective step toward more reliable, consistent, and traceable requirements engineering.

References

Alhoshan, W.; Batista-Navarro, R.; and Zhao, L. 2018. A FrameNet-based Approach for Annotating Natural Language Descriptions of Software Requirements.

Anwar, S.; Shelmanov, A.; Arefyev, N.; Panchenko, A.; and Biemann, C. 2024. Text augmentation for semantic frame induction and parsing. *Language Resources and Evaluation*, 58(2): 363–408.

Arvidsson, S.; and Axell, J. 2023. Prompt engineering guidelines for LLMs in Requirements Engineering.

Ellsel, C.; and Stark, R. 2025. Advancing Requirements Engineering with Large Language Models. *Procedia CIRP*, 136: 701–706.

Ferrari, A.; and Spoletini, P. 2025. Formal requirements engineering and large language models: A two-way roadmap. *Information and Software Technology*, 181: 107697.

Fontenelle, T. 2003. Framenet and frame semantics. *International Journal of Lexicography*, 16(3): 231–231.

Karou Diallo, P. A. K.; and Zouaq, A. 2025. Enhancing Frame Detection with Retrieval Augmented Generation. *arXiv e-prints*, arXiv–2502.

Kauffmann, T. 2025. Leveraging LLMs for interpreting historical source code: a case study of the Apple Lisa through critical code studies. *AI & SOCIETY*, 1–21.

Khan, J. A.; Qayyum, S.; and Dar, H. S. 2025. Large Language Model for Requirements Engineering: A Systematic Literature Review.

Krishna, M.; Gaur, B.; Verma, A.; and Jalote, P. 2024. Using llms in software requirements specifications: An empirical evaluation. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, 475–483. IEEE.

Kundi, M.; Ahmad, F.; and Monahan, R. 2025. Refining Environmental Requirements for Autonomous Driving Systems: Leveraging the FRAV Framework. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 15–22. IEEE.

Kundi, M.; and Chitchyan, R. 2017. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*, Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017, 224–231. United States: Institute of Electrical and Electronics Engineers (IEEE).

Ma, Q.; Peng, W.; Yang, C.; Shen, H.; Koedinger, K.; and Wu, T. 2025. What should we engineer in prompts? training humans in requirement-driven llm use. *ACM Transactions on Computer-Human Interaction*, 32(4): 1–27.

Nadăș, M.; Dioșan, L.; and Tomescu, A. 2025. Synthetic data generation using large language models: Advances in text and code. *IEEE Access*.

Narayan, A.; Kumar, S. M.; and Chacko, A. M. 2025. Trust at risk: Detecting misinformation in LLM-generated product reviews and its implications for consumer behavior and platform governance. *Telematics and Informatics Reports*, 100285.

Okamoto, R.; and Kusumoto, S. 2025. Towards the Automatic Restructuring of Software Requirements Specifications to Conform to Standards Using Large Language Models. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*, 467–475. IEEE.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744.

Rai, S. I.; Croce, D.; and Basili, R. 2025. Injecting Frame Semantics into Large Language Models via Prompt-Based Fine-Tuning. In Frermann, L.; and Stevenson, M., eds., *Proceedings of the 14th Joint Conference on Lexical and Computational Semantics (*SEM 2025)*, 31–47. Suzhou, China: Association for Computational Linguistics.

Ren, S.; Nakagawa, H.; and Tsuchiya, T. 2024. Combining prompts with examples to enhance llm-based requirement elicitation. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, 1376–1381. IEEE.

Zamfirescu-Pereira, J. D.; Wong, R. Y.; Hartmann, B.; and Yang, Q. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI conference on human factors in computing systems*, 1–21.

Zhu, T.; Cordeiro, L. C.; and Sun, Y. 2025. ReqInOne: A Large Language Model-Based Agent for Software Requirements Specification Generation. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*, 449–457. IEEE.