Translation Titans, Reasoning Challenges: Satisfiability-Aided Language Models for Detecting Conflicting Requirements

Mohamad Fazelnia
University of Hawaii at Manoa
Honolulu, USA, mfazel@hawaii.edu

Mehdi Mirakhorli
University of Hawaii at Manoa
Honolulu, USA, mehdi23@hawaii.edu

Hamid Bagheri
University of Nebraska-Lincoln
Lincoln, USA, bagheri@unl.edu

ABSTRACT

Detecting conflicting requirements early in the software development lifecycle is crucial to mitigating risks of system failures and enhancing overall reliability. While Large Language Models (LLMs) have demonstrated proficiency in natural language understanding tasks, they often struggle with the nuanced reasoning required for identifying complex requirement conflicts. This paper introduces a novel framework, SAT-LLM, which integrates Satisfiability Modulo Theories (SMT) solvers with LLMs to enhance the detection of conflicting software requirements. SMT solvers provide rigorous formal reasoning capabilities, complementing LLMs' proficiency in natural language understanding. By synergizing these strengths, SAT-LLM aims to overcome the limitations of standalone LLMs in handling intricate requirement interactions. The early experiments provide empirical evidence supporting the effectiveness of our SAT-LLM over pure LLM-based methods like ChatGPT in identifying and resolving conflicting requirements. These findings lay a foundation for further exploration and refinement of hybrid approaches that integrate NLP techniques with formal reasoning methodologies to address complex challenges in software development.

CCS CONCEPTS

• **Software and its engineering** → *Traceability*; • **Computing methodologies** → **Knowledge representation and reasoning**.

KEYWORDS

Software Requirements Conflict, Large Language Models, Formal Reasoning, Satisfiability, SMT, Conflict Detection, LLMs, Reasoning

ACM Reference Format:
Mohamad Fazelnia, Mehdi Mirakhorli, and Hamid Bagheri. 2024. Translation Titans, Reasoning Challenges: Satisfiability-Aided Language Models for Detecting Conflicting Requirements. In *Conference, Preprint*. ACM, New York, NY, USA, Article 111, 5 pages. https://doi.org/10.1145/3691620.3695302

1 INTRODUCTION

In contemporary software development, ensuring the coherence and consistency of requirements is paramount to achieving successful system outcomes [5, 20, 34]. A thorough analysis of the diverse stakeholder-proposed requirements is crucial to uncover contradictions or incompatibilities. Conflicting requirements, if left undetected, can precipitate critical failures upon deployment, jeopardizing reliability and user satisfaction [1, 15, 17, 32]. Therefore, robust methodologies for identifying and resolving such conflicts are indispensable.

Recent advancements in Natural Language Processing (NLP), particularly with Large Language Models (LLMs) such as OpenAI's GPT series, have significantly enhanced various aspects of requirements analysis [25]. These models excel in tasks such as requirements classification [16, 39] and requirement information retrieval [41], leveraging their extensive training on diverse linguistic datasets.

Despite their success, LLMs are prone to generating inaccurate outputs, known as hallucination, particularly in scenarios requiring nuanced reasoning and comprehension beyond surface-level semantics [2, 4, 21, 26, 29]. This limitation poses challenges in accurately detecting and resolving complex conflicts arising from multiple, interacting requirements.

Existing studies have primarily focused on pairwise conflict detection using techniques such as natural language inference (NLI) [14, 16]. These approaches typically involve evaluating the relationship between pairs of requirements to determine if they are conflicting or compatible. While effective for simple conflicts involving two requirements, these methods often fall short when confronted with more intricate conflicts arising from the interaction of three or more requirements. Such complex conflicts require a deeper level of reasoning and analysis that goes beyond the capabilities of conventional LLM-based approaches.

To address this state of affairs, this paper explores a novel integration of LLMs with Satisfiability Modulo Theories (SMT) solvers, termed SAT-LLM, to enhance the detection of conflicting software requirements. SMT solvers are renowned for their prowess in computational logic, adeptly identifying logical inconsistencies and discrepancies within formal specifications. Our hybrid approach aims to leverage the linguistic proficiency of LLMs for natural language understanding and translation tasks, complemented by the rigorous reasoning capabilities of SMT solvers for ensuring logical consistency and detecting conflicts, to address the limitations of standalone LLMs in handling intricate requirement conflicts. Specifically, SAT-LLM endeavors to synergize these strengths to improve the accuracy and effectiveness of requirement conflict identification in software engineering contexts.

In our early experiments, we utilized the conflicting requirement dataset proposed by Fazelnia et al. [16], focusing on software requirements for a Broker system. The dataset is originally composed of 45 requirements with various types of inconsistencies including Composite-Negation, Forbid-Stop, Input-Output, Output-Output, Start-Forbid, Two Frequencies, and Negation. We expanded the

Conferenc '24, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695302

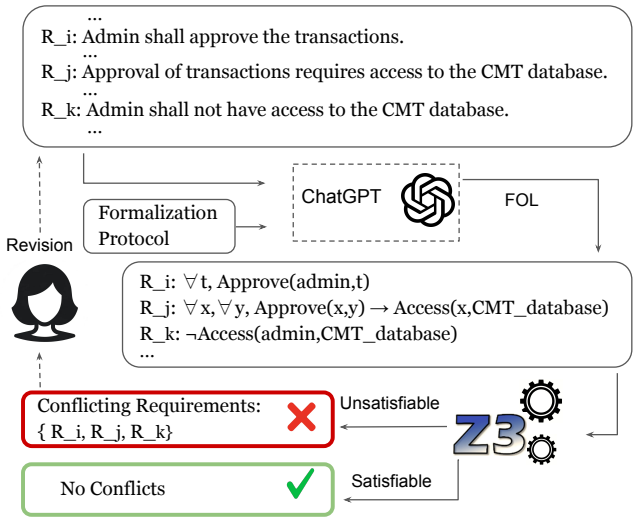

Figure 1: An overview of the proposed SAT-LLM model to detect software inconsistencies

dataset by adding 10 additional requirements, deliberately introducing three new Composite-Negation conflict sets, each involving three or more conflicting requirements. These enhancements were aimed at providing a robust framework for evaluating SAT-LLM's efficacy in identifying and resolving diverse conflict types within software requirements, advancing hybrid approaches in this domain.

The empirical evidence derived from our experiments strongly supports the efficacy of our SAT-LLM approach compared to pure LLM-based methods like ChatGPT in identifying and resolving conflicting software requirements. SAT-LLM demonstrates superior performance, achieving a precision of 1.00, recall of 0.83, and an F1 score of 0.91, highlighting its ability to accurately detect conflicting requirement sets. In contrast, ChatGPT shows moderate precision (0.85), low recall (0.31), and an F1 score of 0.46, indicating limitations in handling complex conflict detection scenarios. These findings not only establish SAT-LLM's capability to accurately identify and resolve conflicts but also set a foundation for refining hybrid approaches that integrate advanced NLP techniques with formal reasoning methodologies to effectively address complexities in software development.

2 SAT-LLM METHODOLOGY FOR ENHANCED CONFLICT DETECTION IN SOFTWARE REQUIREMENTS

This section introduces the SAT-LLM methodology, designed to enhance the detection and resolution of conflicting software requirements by integrating LLMs with SMT solvers. SAT-LLM leverages the strengths of NLP in text understanding and formal methods in logical reasoning, aiming to address challenges posed by ambiguous or complex requirement specifications.

As depicted in Figure 1, the model initially receives a set of requirements. These inputs are processed by LLMs to interpret and translate the requirements from natural language into formal logic, which are then passed to the Z3[11] module for reasoning. We detail each step in the remainder of this section.

2.1 Formalization Protocol

By utilizing First-Order Logic (FOL), SAT-LLM can express complex relationships and dependencies among requirements. FOL extends propositional logic by introducing quantifiers ($\exists$ and $\forall$) for statements about objects, predicates for describing properties or relations, and functions for mapping within a defined domain. Logical connectives ($\wedge$, $\vee$, $\neg$, $\longrightarrow$) further enable the formulation and analysis of intricate requirement specifications, facilitating precise conflict detection and resolution.

Natural language sentences can have several non-equivalent translations in FOL, as NL often incorporates nuances and contextual elements that can be interpreted and formalized in multiple valid ways within FOL.

Through our experimentation, we observed that when instructing ChatGPT to translate NL into FOL, it may struggle to grasp the underlying information in each NL requirement. For example, it tends to interpret different terminologies and actions inconsistently, using terms such as 'allow', 'permit', or 'stop' interchangeably without recognizing their specific implications. Similarly, it may fail to recognize that actions like 'modify', 'add', 'remove', and 'update' can often be grouped under a single, more general action term without guidance, leading to unnecessary complexity and potential errors in the translated logic.

Central to SAT-LLM is the Formalization Protocol, which standardizes the translation of natural language requirements into FOL. Specifically, we conducted several experiments to identify the inconsistencies within the translation and developed a protocol for translating NL into FOL. We identified four primary factors contributing to these inconsistencies, which are detailed in Table 1.

This protocol systematically addresses these nuances by standardizing terminology, consolidating actions, decomposing complex requirements, and streamlining function definitions. These efforts reduce ambiguity and enhance the consistency of FOL translations generated by ChatGPT. The protocol, along with the requirements, is provided to ChatGPT to guide the translation into FOL.

2.2 Identifying Conflicts within the Formal Specification

These generated FOL expressions are then submitted to the Z3 SMT solver. The role of the Z3 solver is crucial in this context: it analyzes the set of FOL expressions to determine if they are collectively satisfiable or if there are logical inconsistencies (unsatisfiability) among them.

If Z3 determines that the set of FOL expressions is satisfiable (*SAT*), it indicates that there are no contradictions or inconsistencies among the requirements. This outcome suggests that all specified conditions can coexist without conflict, thereby meeting the system's logical criteria.

However, if Z3 finds the set of FOL expressions to be unsatisfiable (*UNSAT*), it means that there exist conflicting requirements within the set. When faced with an unsatisfiable set, Z3 employs an algorithm, such as the one proposed by Liffiton et al. [28], to pinpoint

Table 1: Factors Contributing to Inconsistencies in NL Requirements to FOL Constraints Translation

Category	Definition (D) and Example (E)
Terminology Consistency	(D) Grouping similar actions under a unified term to avoid inconsistencies, and using negation for antonyms. (E) Allow/Permit: Use "allow", Stop / Prevent: use "Not(allow)", Revise/Modify/Change: Modify
Action Consolidation	(D) Using broder action when certain actions are subset of more general actions. In such cases, both specifications shall be expressed. (E) Use "modify" to encompass "add", "remove", and "update"
Requirement Decomposition	(D) Break down complex requirements involving multiple subjects or actions into simpler, separate statements to clarify. (E) Instead of "X and Y shall have access to the system", use "X shall have access to the system" and "Y shall have access to the system".
Function Streamlining	(D) Simplify function definitions by using only the main verb as the function name. (E) Use "access(system, X)" instead of creating overly specific functions like "haveSystemAccess(X)".

Minimal Unsatisfiable Subsets (MUSes) and Minimal Satisfiable Subsets (MSSes).

The algorithm used typically involves two main components: the MapSolver and the SubsetSolver. The MapSolver identifies unique atomic predicates to prevent the inclusion of supersets of existing unsatisfiable cores and subsets of satisfying assignments. It strategically adds clauses to enforce or avoid specific conditions based on the logical structure of the requirements.

Conversely, the SubsetSolver receives these clauses from the MapSolver. Its role is to evaluate whether these clauses are infeasible (contributing to the unsatisfiability) or if they can potentially be extended or adjusted to form a satisfying set (contributing to satisfiability).

Once the algorithm completes its evaluation, Z3 returns the identified MUSes. These MUSes are subsets of requirements that, when considered together, form a core contradiction within the set. Each MUS represents a minimal group of requirements whose conjunction leads to logical inconsistency.

These identified MUSes are then communicated back to the development team. Their role is crucial for further analysis and action: they highlight specific points of conflict within the requirements. The development team can use this information to revise, refine, or prioritize changes to the requirements to resolve the identified conflicts effectively.

3 PRELIMINARY EXPERIMENTS AND FINDINGS IN INCONSISTENCY IDENTIFICATION

Our initial experiments focus on two key research questions:

- **RQ1:** How effective is ChatGPT in identifying conflicting requirements?
- **RQ2:** How does integrating Satisfiability Techniques with ChatGPT improve the effectiveness of conflict detection in software requirements?

3.1 Experimental Setup

We conducted our experiments using the conflicting dataset proposed by [16]. This dataset comprises software requirements for a Broker system, encompassing seven types of inconsistencies: Composite-Negation, Forbid-Stop, Input-Output, Output-Output, Start-Forbid, Two Frequencies, and Negation. Initially containing 45 requirements and 12 conflicting subsets, we expanded the dataset by adding 10 requirements, specifically targeting three Composite-Negation conflicting sets. As a result, the final dataset comprised 55 requirements distributed across 15 subsets of conflicting requirements. Among these subsets, four subsets involve complex composite conflicts (conflicts that occur among three or more requirements), while the others consist of pairs of conflicting requirements.

As for the experimental setup, for translating natural language to FOL, we employed ChatGPT based on the GPT-4 model. Similarly, we used the same model for benchmark tasks where we directly queried ChatGPT to identify conflicting requirements.

We also consider the following evaluation metrics: **True Positive (TP):** The number of conflicting requirements correctly identified along with their conflicting counterparts. **False Positive (FP):** The number of non-conflicting requirements incorrectly identified as conflicting. **True Negative (TN):** The number of non-conflicting requirements correctly identified as such. **False Negative (FN):** The number of conflicting requirements incorrectly identified as non-conflicting.

These metrics are defined and computed as:

Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$

Precision = $\frac{TP}{TP+FP}$

Recall = $\frac{TP}{TP+FN}$

F1-score = $2 \times \frac{Precision \times Recall}{Precision+Recall}$

To fully leverage ChatGPT's potential in identifying inconsistencies, we conducted several experiments to determine the most effective prompt. The prompt plays a crucial role when working with LLMs [37]. Therefore, the final version includes detailed task instructions along with examples.

3.2 Preliminary Results and Analysis

Our experiments revealed that ChatGPT detected 33% of conflicting requirement sets, performing better with direct conflicts than hidden-complex ones. It accurately identified 36% of direct conflicts (two requirements) and 25% of composite-complex conflicts (three or more requirements). Additionally, it mistakenly classified two non-conflicting requirements as conflicting, indicating a need for improvement in its detection capabilities.

Results for RQ1: ChatGPT identified 33% of the conflicting sets. Overall, it achieved the following scores: Precision of 0.85, Recall of 0.31, and an F1 score of 0.46. ChatGPT demonstrated

lower performance in identifying conflicts that are hidden or span across multiple requirements.

The interpretation of the obtained results highlights that ChatGPT struggles to find hidden and composite conflicts, especially when they involve a larger number of requirements, i.e., three or more requirements forming the conflict. Moreover, the occurrence of false positives indicates that ChatGPT may identify a subset of requirements as conflicting when no conflict actually exists, which can be considered a hallucination case. These finding suggest the shortcomings of relying solely on ChatGPT to accurately detect and resolve complex conflicts within the requirements.

On the other hand, we observed that SAT-LLM is able to detect 80% of conflicts correctly. Specifically, SAT-LLM identified 73% of conflicting pairs and 100% of composite-complex conflicting sets. Furthermore, our method did not produce any false positives, which underscores both the accuracy and reliability of the approach in correctly identifying conflicts.

Results for RQ2: SAT-LLM successfully identified 80% of the conflicting sets. Overall, it achieved the following scores: Precision of 1.00, Recall of 0.83, and an F1 score of 0.91. These results highlight the effectiveness of combining ChatGPT with formal reasoning methodologies in identifying conflicting requirements.

We observed that the cases where SAT-LLM couldn't detect the conflict belong to «Forbid - Stop» and «Output - Output» categories of inconsistencies. The former occurs "When the same operation is stopped under a certain condition event and at the same time, is unconditionally forbidden in another requirement," and the latter occurs " if one requirement alters the result (output) or part of another requirement." This underscores the challenges SAT-LLM faces with complex dependencies and nuanced interactions between requirements.

4 RELATED WORK

The identification and management of conflicting requirements in software engineering have been extensively studied, employing various methodologies [19]. Early approaches [20, 24, 35] explored the use of NLP and SMT solvers for detecting inconsistencies among stakeholder requirements and prioritizing them, respectively. Filipovikj et al. [18] introduced automated consistency checking using pattern-based formalization with the Z3 SMT solver, while Brito et al. [6] proposed conflict resolution techniques for diverse stakeholder requirements. These studies primarily focused on direct conflicts, which occur between two requirements [16], or employed traditional methods without leveraging the advanced language understanding capabilities of LLMs. Instead, they involved manual or semi-automated translation of requirements from natural language into formal specifications, a process that is often time-consuming, labor-intensive, and highly susceptible to human error and oversight [7, 12].

The emergence of LLMs has sparked interest in their potential applications within the software engineering community [3, 22], particularly in requirements engineering tasks. Fazelnia et al. [16] demonstrated that formulating classification as an entailment problem enables smaller LLMs, such as RoBERTa [30], to outperform larger ones such as ChatGPT, while their approach revealed limitations in effectively identifying composite conflicts within the requirements. Zhang et al. [40] and Carvallo et al. [8] investigated ChatGPT's effectiveness in requirements information retrieval and various requirements engineering tasks, respectively. Luitel et al. [31] studied the potential of LLMs for detecting incompleteness in software requirements. Fantechi et al. [14] specifically examined ChatGPT's ability to identify inconsistencies in requirements but noted limitations in handling complex, composite conflicts. While studies have shown that LLMs excel in tasks like translation and classification of requirements [13, 23], they are also susceptible to issues like hallucination [4, 27, 33]. Wang et al. [38] demonstrated that ChatGPT struggles to maintain logical consistency, highlighting potential issues with reasoning capabilities.

For the translation tasks, in the past few years LLMs have been widely studied. Endres et al. investigated ChatGPT's translation abilities to transform natural language intent into formal method postconditions [13]. Cosler et al. [10] proposed nl2spec to interactively translate unstructured natural language into temporal logic using GPT models. NL2TL [9] transformed natural language into temporal logic and investigated the performance of different language models including T5 [36] and OpenAI's GPT models.

5 CONCLUSION AND FUTURE WORK

In this work, we propose SAT-LLM, a novel framework that integrates the linguistic capabilities of LLMs with the rigorous logical reasoning of SMT solvers to improve the detection and resolution of complex, composite conflicts in software requirements. By extending requirement analysis beyond pairwise evaluations, SAT-LLM enhances the accuracy and comprehensiveness of conflict detection, handling nuanced requirement interactions more effectively than previous methods. Through empirical validation and comparative analysis, we demonstrate the operational advantages of our approach and highlight its potential for improving the reliability and accuracy of software development processes.

For future work, we plan to focus on improving our model's ability to better understand the semantics needed for seamless and accurate translation from natural language to FOL. We will also extend SAT-LLM's capabilities to address even more intricate interactions and composite conflicts. Given the limited availability of data on conflicting requirements, we have already expanded our dataset to include more complex conflicts. Our future efforts will involve further expanding this dataset to thoroughly evaluate the effectiveness of our methodology in identifying and resolving conflicts in real-world software development scenarios.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments. This work was supported in part by National Science Foundation awards CCF-2139845, CCF-2124116, and CCF-1943300.

REFERENCES

[1] Khlood Ahmad, Muneera Bano, Mohamed Abdelrazek, Chetan Arora, and John Grundy. 2021. What's up with requirements engineering for artificial intelligence systems?. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*. IEEE, 1–12.

[2] Mohannad Alhanahnah, Md Rashedul Hasan, and Hamid Bagheri. 2024. An Empirical Evaluation of Pre-trained Large Language Models for Repairing Declarative Formal Specifications. *CoRR* abs/2404.11050 (2024). https://doi.org/10.48550/ARXIV.2404.11050 arXiv:2404.11050

[3] Mohannad Alhanahnah, Md Rashedul Hasan, and Hamid Bagheri. 2024. An Empirical Evaluation of Pre-trained Large Language Models for Repairing Declarative Formal Specifications. *arXiv preprint arXiv:2404.11050* (2024).

[4] Hamid Bagheri, Mehdi Mirakhorli, Mohamad Fazelnia, Ibrahim Mujhid, and Md Rashedul Hasan. 2024. Neuro-Symbolic Approach to Certified Scientific Software Synthesis. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*. 147–150.

[5] Brian Berenbach, Daniel Paulish, Juergen Kazmeier, and Arnold Rudorfer. 2009. *Software & systems requirements engineering: in practice*. McGraw-Hill, Inc.

[6] Isabel Sofia Brito, Ana Moreira, Rita A Ribeiro, and João Araújo. 2013. Handling conflicts in aspect-oriented requirements engineering. *Aspect-Oriented Requirements Engineering* (2013), 225–241.

[7] Luiz Carvalho, Renzo Gaston DEGIOVANNI, Matıas Brizzio, Maxime Cordy, Nazareno Aguirre, Yves Le Traon, and Mike Papadakis. 2023. Acore: Automated goal-conflict resolution. In *26th International Conference on Fundamental Approaches to Software Engineering (FASE)*, Vol. 13991.

[8] Juan Pablo Carvallo and Lenin Erazo-Garzón. 2023. On the use of ChatGPT to support requirements engineering teaching and learning process. In *Latin American Conference on Learning Technologies*. Springer, 328–342.

[9] Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. 2023. Nl2tl: Transforming natural languages to temporal logics using large language models. *arXiv preprint arXiv:2305.07766* (2023).

[10] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *International Conference on Computer Aided Verification*. Springer, 383–396.

[11] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[12] Renzo Degiovanni, Facundo Molina, Germán Regis, and Nazareno Aguirre. 2018. A Genetic Algorithm for Goal-Conflict Identification. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 520–531. https://doi.org/10.1145/3238147.3238220

[13] Madeline Endres, Sarah Fakhoury, Saikat Chakraborty, and Shuvendu K Lahiri. 2023. Formalizing Natural Language Intent into Program Specifications via Large Language Models. *arXiv preprint arXiv:2310.01831* (2023).

[14] Alessandro Fantechi, Stefania Gnesi, Lucia Passaro, and Laura Semini. 2023. Inconsistency Detection in Natural Language Requirements using ChatGPT: a Preliminary Evaluation. In *2023 IEEE 31st International Requirements Engineering Conference (RE)*. IEEE, 335–340.

[15] Mohamad Fazelnia, Igor Khokhlov, and Mehdi Mirakhorli. 2022. Attacks, defenses, and tools: A framework to facilitate robust AI/ML systems. *arXiv preprint arXiv:2202.09465* (2022).

[16] Mohamad Fazelnia, Viktoria Koscinski, Spencer Herzog, and Mehdi Mirakhorli. 2024. Lessons from the Use of Natural Language Inference (NLI) in Requirements Engineering Tasks. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. 103–115. https://doi.org/10.1109/RE59067.2024.00020

[17] Mohamad Fazelnia, Ahmet Okutan, and Mehdi Mirakhorli. 2022. Supporting Artificial Intelligence/Machine Learning Security Workers Through an Adversarial Techniques, Tools, and Common Knowledge Framework. *IEEE Security & Privacy* 21, 1 (2022), 37–48.

[18] Predrag Filipovikj, Guillermo Rodriguez-Navas, Mattias Nyberg, and Cristina Seceleanu. 2017. SMT-based consistency analysis of industrial systems requirements. In *Proceedings of the Symposium on Applied Computing*. 1272–1279.

[19] Matthias Galster, Mehdi Mirakhorli, Jane Cleland-Huang, Janet E. Burge, Xavier Franch, Roshanak Roshandel, and Paris Avgeriou. 2013. Views on software engineering from the twin peaks of requirements and architecture. *SIGSOFT Softw. Eng. Notes* 38, 5 (aug 2013), 40–42. https://doi.org/10.1145/2507288.2507323

[20] Vincenzo Gervasi and Didar Zowghi. 2005. Reasoning about inconsistencies in natural language requirements. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 14, 3 (2005), 277–330.

[21] Md Rashedul Hasan, Jiawei Li, Iftekhar Ahmed, and Hamid Bagheri. 2023. Automated Repair of Declarative Software Specifications in the Era of Large Language Models. *CoRR* abs/2310.12425 (2023). https://doi.org/10.48550/ARXIV.2310.12425 arXiv:2310.12425

[22] Md Rashedul Hasan, Jiawei Li, Iftekhar Ahmed, and Hamid Bagheri. 2023. Automated Repair of Declarative Software Specifications in the Era of Large Language Models. *arXiv preprint arXiv:2310.12425* (2023).

[23] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F Tichy. 2020. Norbert: Transfer learning for requirements classification. In *2020 IEEE 28th international requirements engineering conference (RE)*. IEEE, 169–179.

[24] Viktoria Koscinski, Celeste Gambardella, Estey Gerstner, Mark Zappavigna, Jennifer Cassetti, and Mehdi Mirakhorli. 2021. A Natural Language Processing Technique for Formalization of Systems Requirement Specifications. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. 350–356. https://doi.org/10.1109/REW53955.2021.00062

[25] Viktoria Koscinski, Sara Hashemi, and Mehdi Mirakhorli. 2023. On-Demand Security Requirements Synthesis with Relational Generative Adversarial Networks. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1609–1621. https://doi.org/10.1109/ICSE48619.2023.00139

[26] Jack Lanchantin, Shubham Toshniwal, Jason Weston, Sainbayar Sukhbaatar, et al. 2024. Learning to reason and memorize with self-notes. *Advances in Neural Information Processing Systems* 36 (2024).

[27] Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023. Halueval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 6449–6464.

[28] Mark H Liffiton and Ammar Malik. 2013. Enumerating infeasibility: Finding multiple MUSes quickly. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings 10*. Springer, 160–175.

[29] Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. 2024. Deductive verification of chain-of-thought reasoning. *Advances in Neural Information Processing Systems* 36 (2024).

[30] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[31] Dipeeka Luitel, Shabnam Hassani, and Mehrdad Sabetzadeh. 2024. Improving requirements completeness: Automated assistance through large language models. *Requirements Engineering* 29, 1 (2024), 73–95.

[32] Dewi Mairiza and Didar Zowghi. 2010. An ontological framework to manage the relative conflicts between security and usability requirements. In *2010 Third International Workshop on Managing Requirements Knowledge*. IEEE, 1–6.

[33] Timothy R McIntosh, Tong Liu, Teo Susnjak, Paul Watters, Alex Ng, and Malka N Halgamuge. 2023. A culturally sensitive test to evaluate nuanced gpt hallucination. *IEEE Transactions on Artificial Intelligence* (2023).

[34] Mehdi Mirakhorli and Jane Cleland-Huang. 2013. Traversing the Twin Peaks. *IEEE Software* 30, 2 (2013), 30–36. https://doi.org/10.1109/MS.2013.40

[35] Francis Palma, Angelo Susi, and Paolo Tonella. 2011. Using an SMT solver for interactive requirements prioritization. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 48–58. https://doi.org/10.1145/2025113.2025124

[36] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[37] Alberto D Rodriguez, Katherine R Dearstyne, and Jane Cleland-Huang. 2023. Prompts matter: Insights and strategies for prompt engineering in automated software traceability. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 455–464.

[38] Boshi Wang, Xiang Yue, and Huan Sun. 2023. Can ChatGPT Defend its Belief in Truth? Evaluating LLM Reasoning via Debate. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 11865–11881. https://doi.org/10.18653/v1/2023.findings-emnlp.795

[39] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. 2024. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. In *Generative AI for Effective Software Development*. Springer, 71–108.

[40] Jianzhang Zhang, Yiyang Chen, Chuang Liu, Nan Niu, and Yinglin Wang. 2023. Empirical Evaluation of ChatGPT on Requirements Information Retrieval Under Zero-Shot Setting. In *2023 International Conference on Intelligent Computing and Next Generation Networks (ICNGN)*. IEEE, 1–6.

[41] Jianzhang Zhang, Yiyang Chen, Nan Niu, and Chuang Liu. 2023. A preliminary evaluation of chatgpt in requirements information retrieval. *arXiv preprint arXiv:2304.12562* (2023).

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009