

# Leveraging LLMs for Formal Software Requirements: Challenges and Prospects

Arshad Beg<sup>1,\*†</sup>, Diarmuid O'Donoghue<sup>1,†</sup> and Rosemary Monahan<sup>1,†</sup>

<sup>1</sup>Maynooth University, Maynooth, Ireland.

## Abstract

Software correctness is ensured mathematically through formal verification, which requires the generation of a formal requirement specification and an implementation that must be verified. Tools such as model-checkers and theorem provers ensure software correctness by verifying the implementation against the specification. Formal methods deployment is regularly enforced in the development of safety-critical systems e.g. aerospace, medical devices and autonomous systems. Generating these specifications from informal and ambiguous natural language requirements remains the key challenge. Our project, VERIFYAI<sup>1</sup>, aims to investigate automated and semi-automated approaches to bridge this gap, using techniques from Natural Language Processing (NLP), ontology-based domain modelling, artefact reuse, and large language models (LLMs). This position paper presents a preliminary synthesis of relevant literature to identify recurring challenges and prospective research directions in the generation of verifiable specifications from informal requirements.

## Keywords

Large Language Models, Knowledge Representation and Reasoning, Formal Languages, Software Requirements-{Engineering, Specifications}, Formal Verification, Theorem Proving, Model Checking, Chain-of-Thought (CoT), Prompt-Engineering, {Zero, One, and Few}-shot prompting

## 1. Introduction

As software systems grow in complexity and criticality, so does the need for scalable verification methods that ensure correctness and reliability. Formal verification is especially important in safety-critical systems where minor software errors can lead to serious consequences, including loss of life, environmental damage, or large-scale system failures. Under all circumstances, the software used in sectors like aviation, healthcare, automotive, and nuclear control must behave exactly as intended. While testing can only check specific scenarios, formal verification uses mathematical techniques to prove that a system meets its specifications under every possible condition. This level of assurance is crucial when human safety depends on software behaving reliably. The purpose of sound software engineering principles is to catch flaws early in the design phase, ensuring consistency between requirements and implementation. The mathematical techniques used in formal methods improve trust, and support compliance with industry standards and regulations. However, their adoption in industry is consistently hindered by the challenges of writing and maintaining formal specifications, which demand rigorous

<sup>1</sup>[https://github.com/arshadbeg/OVERLAY2025\\_SupportingDocs.git](https://github.com/arshadbeg/OVERLAY2025_SupportingDocs.git)

7th International Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis (OVERLAY 2025), October 26, 2025, Bologna, Italy

\*Corresponding author.

†These authors contributed equally.

✉ arshad.beg@mu.ie (A. Beg); diarmuid.odonoghue@mu.ie (D. O'Donoghue); rosemary.monahan@mu.ie (R. Monahan)

🆔 0009-0004-6939-0411 (A. Beg); 0000-0002-3680-4217 (D. O'Donoghue); 0000-0003-3886-4675 (R. Monahan)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

developer training and significantly increase the software development cycle time by up to 30% [1]. This motivates research into automated and semi-automated approaches that can make formal verification more accessible to a wider audience of software engineers.

The VERIFYAI project [2] aims to address a central challenge of formal software engineering: translating informal, natural-language requirements into formal, verifiable specifications. This paper outlines the challenges and prospects in leveraging Large Language Models (LLMs) for formalising software requirements. The project aims to integrate techniques from Natural Language Processing (NLP), ontology-driven domain modelling, artefact reuse, and large language models (LLMs) to support the automated generation and traceability of formal specifications. Like all other fields, the development of large language models (LLMs) has opened a world of opportunities for the challenge of formalisation of software requirements. We consolidate initial findings to highlight research gaps and recurring difficulties in LLM-assisted formal specification generation. Noting firstly that the main body of the paper is supported by several detailed Appendices available at [https://github.com/arshadbeg/OVERLAY2025\\_SupportingDocs.git](https://github.com/arshadbeg/OVERLAY2025_SupportingDocs.git).

The key contributions of this paper are as follows:

**Identification of Core Challenges in Formalisation:** We present a structured analysis of the barriers to translating informal, natural language software requirements into formal specifications, such as ambiguity, lack of domain models, and LLM instability.

**VERIFYAI Research Framework:** We propose our research framework, which integrates LLMs with NLP, ontology-driven modelling, and artefact reuse to support the semi-automated generation of verifiable formal specifications.

**State-of-the-Art Synthesis:** Through a focused literature review, we categorise and compare (Section 2, supported by Appendix A) recent LLM-based tools and techniques—such as Req2Spec, SpecGen, AssertLLM, and nl2spec—highlighting their approaches, strengths, and limitations in requirement formalisation.

**Experimental Evaluation:** We include empirical evaluations (Appendices B and C) comparing multiple SMT solvers (Alt-Ergo, Z3, CVC4, CVC5) in terms of specification verification success and execution time, using the Frama-C PathCrawler tool and standard program sets.

**Highlighting Gaps and Future Directions:** Based on our synthesis, we outline critical open problems such as prompt instability, fragility of formal outputs, and the need for domain-specific context grounding (Appendix D). These pave the way for developing more robust LLM-based formal specification pipelines.

**Positioning for Long-Term Vision:** As a position paper, our work serves as a foundational step toward a longer-term vision of trustworthy, LLM-assisted formal methods tooling that bridges the expertise gap in safety-critical software development.

The structure of the paper is: Section 2 describes a focused state of the art. Section 3 outlines the challenges and future directions for the research, supported by Appendix B, that outlines the experiment setup and analysis performed on an example of the PathCrawler tool of Frama-C where we have re-simulated the methodology presented in [3]. This analysis is based on four provers available in Frama-C i.e. Alt-Ergo, Z3, CVC4 and CVC5. Appendix C presents the execution time comparison for these provers on the programs provided in [4]. Section 4 discusses our plans for future work and Section 5 concludes the paper.

## 2. State of the Art

This section synthesises what we found to be the state of the art. The main research questions for conducting our systematic literature review on the topic were as follows:

**RQ1:** What methodologies leverage Large Language Models (LLMs) to transform natural language software requirements into formal notations?

**RQ2:** What are the emerging trends and future research directions in using LLMs for formal requirements formalisation?

Here, we summarise our main findings. For a comprehensive overview of the literature survey, including detailed comparisons and categorised insights, we encourage readers to consult Appendix A and the accompanying GitHub repository, which presents the full set of supporting tables.

GPT-3.5 has assisted requirement analysis for code verification [5], while Explanation-Refiner integrates LLMs with theorem provers for NLI validation and iterative correction [6]. Evaluation of GPT-4o with VeriFast shows generation of functional specifications, though verification remains limited due to redundancy and failed assertions [7].

The nl2spec tool supports interactive synthesis from unstructured requirements [8], while the SpecSyn tool improves sequence-to-sequence contract generation with a 21% accuracy gain [9]. Req2Spec converts 71% of BOSCH automotive requirements into formal specs [10], while SpecGen uses prompt mutation and verification feedback to improve LLM-generated specifications, succeeding on 279 out of 384 benchmark programs [11]. In the hardware domain, AssertLLM synthesizes assertions with 89% correctness via multi-phase prompting and validation [12], while VLSI applications leverage LLMs for spec review and generation in SpecLLM [13]. Smart grid requirements have been formalised using GPT-4o and Claude 3.5, achieving F1 scores between 79% and 94% [14]. The trend in F1-scores observed by authors of [14] suggested that GPT-4o and Claude 3.5 not only maintain robustness but may actually benefit from increased system specification complexity, highlighting a potential alignment between model reasoning depth and specification richness. This aspect not mirrored in Gemini 1.5 or GPT-3.5-turbo and warranting further investigation. NASA’s software verification effort surfaced requirement errors, demonstrating practical utility of LLM-in-the-loop workflows [15].

NL-to-LTL translation has seen progress via few-shot prompting and dynamic reasoning [16], achieving 94.4% accuracy. Likewise, NL-to-JML contract synthesis for Java programs has been explored with promising results [17]. Historical systems like RSL [18], ARSENAL [19], and RML [20] demonstrated early rule-based and logic-based extraction pipelines, while hybrid neuro-symbolic systems offer greater reliability. SAT-LLM couples SMT solvers with LLMs to detect inconsistencies with F1 of 0.91 [21] and LeanDojo, ReProver, and Thor enhance formal proving via retrieval-augmented generation and LLM-guided reasoning [22, 23]. IDE-integrated efforts like those combining Copilot with PathCrawler and EVA demonstrate semi-automated ACSL specification generation [24, 3].

As we expected, assertion-level synthesis shows better reliability than full contract generation. For example, Laurel generates assertions for Dafny with over 50% success [25], and AssertLLM exceeds 89% correctness when guided by contract type and context. Full specifications are more error-prone, often requiring multiple prompt iterations or external validation [11].

LLM selection and prompting strategies critically affect performance. While zero-shot prompting is strong in base performance [26], one-shot [27] and few-shot [28] offer alternative trade-offs. Chain-of-Thought (CoT) prompting improves logical flow via intermediate steps [29], and like Structured CoT (SCoT) [30] it can suffer from context decay in long prompts (“lost-in-the-middle”) [31], yet zero-shot often remains competitive [32].

Advanced prompting methods like Automate-CoT generate CoT examples automatically [33], while Reprompting uses Gibbs sampling to escape prompt local optima [34]. Structured prompting with graphs and trees improves reasoning robustness and efficiency [35]. RAG (Retrieval-Augmented Generation) improves grounding for knowledge-intensive synthesis [36]. [37] discusses a wider range of almost 30 prompting strategies, some of which seem not to have been explored in relation to formalising specifications. Of course, this does not include related approaches such as fine-tuning LLM via LoRA adaptation training [38], but this may only be applicable when there is access to the LLM’s architecture and weights. Additionally, reinforcement learning (RL) may help with specific challenges, opening even more avenues for exploration.

### **Key Observations:**

Based on our literature survey, we proceed with some key observations. We observed a significant difference between the success rates of assertion generation and full contract synthesis using LLMs. AssertLLM [12] and Laurel [25] achieved high accuracy in generating helper assertions for programs written in Dafny language [39] and design-specific verification statements. These tools achieved an accuracy of 89% and over 50%, operating at a local level on source code or isolated signals. On the other hand, [17] reported that while generating formal specifications for Java Modelling Language (JML) contracts or temporal logic formulas ended up in frequent verification failures by the SMT solvers embedded in OpenJML [40]. This happened even if the output appeared semantically sound, leading to the conclusion of disparity between human-readable correctness and automated formal verification, especially when the source code was written for complex tasks.

In general, we observed that tasks with small scope and well-defined semantics yield better results where the limited context in these assertions helps in improving verification accuracy [11, 9]. LLMs handle such tasks more reliably due to reduced ambiguity and fewer dependencies on broader system knowledge. On the other hand, for larger program segments, end-to-end contract synthesis involves multiple interacting components or function bodies. It demands a deeper understanding of program semantics, logic, and behaviour over time. SpecGen [11] and SpecSyn [9] presented significant progress in tackling such challenges. However, their outputs require post-processing steps, such as mutation operators or human-in-the-loop (software testing experts were involved), before the generated outputs are usable for formal verification.

The effort accuracy is influenced by tool design and integration. For example, tools like nl2spec [8] improve generated specification quality through step-by-step refinement, adopting iterative and user-in-the-loop approach to help address some LLM limitations. Similarly, prompt engineering techniques utilising guided templates or Chain-of-Thought (CoT) [7, 41, 42] promised improved output coherence and correctness. These strategies work well in scenarios involving localised tasks, such as assertion synthesis or narrow-scope descriptions. As the program size and complexity of the specification goal increase, the chances of ambiguity, under-specification, and logical inconsistency increase. Therefore, we conclude that the current LLM architectures excel in focused, declarative tasks but require augmentation for broader specification goals. However, [43] showed that different versions of language models including LLM, can vary greatly in their responses to the same queries, suggesting that much experimental work might be required to achieve optimum results.

We conclude from our synthesis of the current literature that the research trend is increasing in combining the strength of LLMs, symbolic reasoning and iterative user interaction. At the moment, assertion generation dominates in terms of accuracy and usability, but the parallel efforts of better prompt design, domain-specific fine-tuning, and verifier-in-the-loop are closely matching the performance of the process. The challenges of abstraction and consistency drive research efforts in this domain.

### 3. Challenges and Future Directions

Based on our finding, we summarise five key challenges that we have identified, as well as our research goals with brief description given in Table 1. A detailed description of these is included in Appendix D.

Challenges	Future Directions
<b>C1: Semantic Ambiguity</b> Ambiguity in natural language due to context-dependence and jargon affects requirement interpretation. Needs structured knowledge and human-in-the-loop refinement.	<b>F1: Human-in-the-loop Formalisation</b> Combines LLM support with domain expert oversight to improve accuracy, reduce ambiguity, and increase trust via feedback and interactive refinements.
<b>C2: Lack of Ground Truth Datasets</b> Absence of standardised, annotated datasets limits model training, reproducibility, and scalability.	<b>F3: Standardised Benchmarks</b> Creation of high-quality, domain-diverse datasets will enable consistent evaluation and push the field forward.
<b>C3: Tool Interoperability</b> Formal verification tools lack standard interfaces and integration capabilities, hampering automation.	<b>F4: Neuro-symbolic Reasoning</b> Combines neural flexibility with symbolic precision to improve integration, consistency, and constraint enforcement.
<b>C4: Traceability Across Artefacts</b> Difficult to maintain consistent trace links between text, models, code, and tests over lifecycle.	<b>F5: Interactive Traceability Tools</b> Tools that enable visual navigation, version tracking, and LLM-assisted trace linking improve usability and compliance.
<b>C5: Explainability and User Trust</b> Limited transparency in LLM-generated outputs reduces trust, especially in safety-critical domains.	<b>F2: Multi-modal Artefact Alignment</b> Integrating diverse input types (text, diagrams, tables) through semantic matching increases clarity and confidence in outputs.

**Table 1**

Challenges and Corresponding Future Directions for VERIFYAI Project

Semantic ambiguity (C1) due to natural language remains a critical issue, needing structured domain knowledge and improved human-in-the-loop interventions. The lack of publicly available, high-quality datasets (C2) hinders model training, reproducibility, and scalability. Tool interoperability (C3) is impeded by incompatible formats and absence of standardised interfaces, complicating automation. Ensuring traceability across artefact lifecycles (C4) is essential but difficult without explainable and collaborative processes. In addition, explainability and user trust (C5) are limited by opaque model behavior and insufficient rationale in outputs. To address these, human-in-the-loop formalisation (F1) offers controlled semi-automation and improved trust, while multi-modal artefact alignment (F2) enables contextual completeness via diverse input formats. The creation of standardised benchmarks (F3) would mitigate dataset-related limitations and promote progress. Neuro-symbolic reasoning (F4) blends LLM flexibility with logic-based precision, enhancing model reliability. Finally, interactive traceability tools (F5) that support collaboration, visual navigation, and auditability are crucial for regulated and complex software domains.

## 4. Embedding LLMs in the Specification Generation

### 4.1. Planned Evaluation of Prompting Strategies

A systematic and quantitative evaluation of prompting strategies is the central part of our planned research. In particular, we intend to compare zero-shot, one-shot, few-shot, and Chain-of-Thought (CoT) prompting across both assertion-level and full-contract generation tasks. Prior work in the literature [26–35] suggests that prompt type and articulation can significantly influence specification quality. However, a comprehensive evaluation requires a larger and more diverse benchmark than we currently report.

Future experiments will therefore assess prompt sensitivity using precision, recall, and F1-based correctness metrics, and investigate robustness under small variations of prompt formulation. We anticipate that assertion-level tasks will prove more stable under prompt rephrasing, whereas full-contract synthesis may show higher variability — an observation that motivates deeper analysis in follow-up work. Our current contribution is to highlight the importance of prompt design in our work and to outline how this dimension will be systematically investigated going forward.

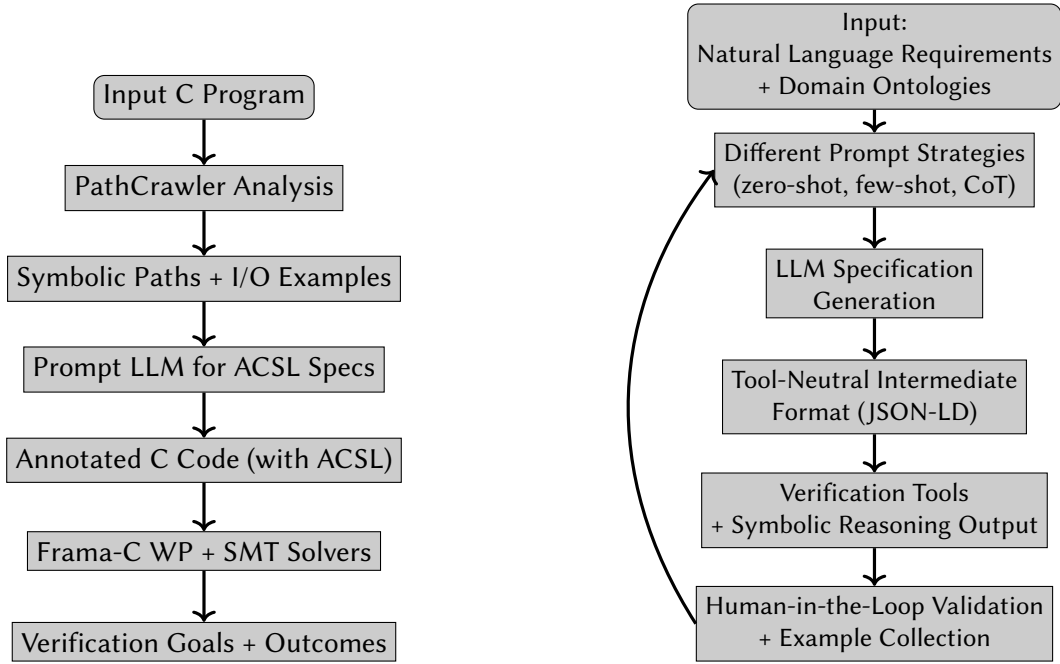
### 4.2. Human-in-the-Loop Integration and Interoperability Considerations

Our experiments currently implement the early stages of a human-in-the-loop process through manual review. In both the Tritype baseline and PathCrawler-augmented workflows, every LLM-generated ACSL specification was checked by at least one author with formal methods expertise. This review ensured (i) semantic alignment with intended behavior, (ii) logical completeness, and (iii) iterative refinement by feeding corrected fragments back into the prompts. Although active learning is not yet integrated, our project approach anticipates it: revised specifications, along with their source code and verification results, can be versioned and selectively reused for retraining or fine-tuning.

Figure 1b shows our proposed workflow based on the state of the art and figured out challenges in Sections 2 and 3. Natural language requirements and domain ontologies form the input, grounding the LLM in the target context. Different prompt strategies (zero-shot, few-shot, and Chain-of-Thought) shape how inputs are presented for specification generation. Outputs are stored in a *tool-neutral intermediate format* (JSON-LD), which can be translated into the syntax required by verification tools. These tools check the generated specifications, while symbolic reasoning provides constraint-based feedback that helps refine them. Human reviewers remain part of the loop, validating results, focusing on manageable units, and collecting useful examples for future adaptation. The design is modular, so new reasoning engines, prompt methods, or domain adapters can be added without changing the overall pipeline.

For interoperability, the pipeline currently uses custom scripts to convert LLM outputs to tool-specific formats (e.g., ACSL for Frama-C, JML for OpenJML). While functional, this approach is not generalisable. As part of VERIFYAI, we plan to design a lightweight JSON-LD-based schema that maps to multiple formal languages. This would allow LLM outputs to be stored in a tool-agnostic format and exported to different targets, potentially supporting Frama-C, OpenJML, Dafny, and others with minimal per-tool changes.





(a) Methodology of the initial experiments following the approach of [3], combining LLM with symbolic analysis tools in the Frama-C ecosystem. The workflow integrates path-based I/O examples and verification outputs to guide the generation of context-aware ACSL specifications.

(b) Proposed workflow for the VERIFYAI pipeline: Natural-language requirements and domain ontologies are combined with prompt strategies to generate formal specifications via LLMs. Outputs in JSON-LD feed verification tools, with symbolic and human feedback refining results.

**Figure 1:** Methodology adopted for initial and future experiments

### 4.3. Symbolic Reasoning, Traceability, and Scalability Outlook

In our initial Tritype experiments, symbolic solvers were used only for post-generation verification. We see potential for tighter integration, where solver feedback—such as unsatisfiable path conditions—could guide the LLM during generation, reducing logical errors before verification. Currently, traceability relies on manual annotations linking specification fragments to source code and natural-language requirements. We are exploring semi-automated approaches where the LLM proposes initial links for human validation. These links could be stored in a graph database to support version-aware navigation and visual trace maps, which would be particularly valuable in regulated settings.

We aim to build datasets that combine real-world, diverse requirements with verified specifications and execution traces. Currently, we are curating a small seed set from open-source safety-critical software, supplemented with synthetic examples to cover edge cases. While synthetic data is useful, it lacks the nuances of industrial requirements, so mixed datasets appear most promising. VERIFYAI’s prototype handles single-module programs well, but multi-module systems pose memory and latency challenges. To manage complexity, we plan to employ hierarchical specification synthesis, verifying each module independently prior to integration. Additionally, we aim to release an annotated subset of the dataset to support transparency and reproducibility.

## 5. Conclusions

This paper identifies clear and concise challenges and prospects in leveraging LLMs for formal software requirements, based on our initial analysis of the literature. Semantic ambiguity, lack of ground truth data, tool interoperability, lifecycle traceability, and explainability all present significant barriers to full automation. However, each challenge also points to fertile ground for innovation. Future directions such as human-in-the-loop systems, multi-modal alignment, standardised benchmarks, neuro-symbolic reasoning, and interactive traceability tools offer practical and scalable paths forward. As a final remark, we can say that as AI and formal methods continue to converge, interdisciplinary collaboration will be key to bridging the gaps and turning conceptual advances into robust, real-world solutions.

## Acknowledgements

This work is partly funded by the ADAPT Research Centre for AI-Driven Digital Content Technology, which is funded by Research Ireland through the Research Ireland Centres Programme and is co funded under the European Regional Development Fund (ERDF) through Grant 13/RC/2106 P2. The submission aligns with Digital Content Transformation (DCT) thread of the ADAPT research centre.

## Declaration on Generative AI

We acknowledge the use of free version of OpenAI's GPT-4 and GPT-4o-mini, solely for refining text. The text has been thoroughly reviewed and discussed by all authors to ensure accuracy and integrity.

## References

- [1] M. Huisman, D. Gurov, A. Malkis, Formal methods: From academia to industrial practice. a travel guide, 2024. URL: <https://arxiv.org/abs/2002.07279>. arXiv:2002.07279.
- [2] A. Beg, D. O'Donoghue, R. Monahan, Formalising software requirements using large language models (2025). URL: <https://arxiv.org/abs/2506.10704>. arXiv:2506.10704.
- [3] G. Granberry, W. Ahrendt, M. Johansson, Specify what? enhancing neural specification synthesis by symbolic methods, in: N. Kosmatov, L. Kovács (Eds.), Integrated Formal Methods, Springer Nature Switzerland, Cham, 2025, pp. 307–325.
- [4] V. Robles, N. Kosmatov, V. Prevosto, P. Le Gall, High-level program properties in frama-c: Definition, verification and deduction, in: Leveraging Applications of Formal Methods, Verification and Validation. Specification and Verification: 12th International Symposium, ISO LA 2024, Crete, Greece, October 27–31, 2024, Proceedings, Part III, Springer-Verlag, Berlin, Heidelberg, 2024, p. 159–177. URL: [https://doi.org/10.1007/978-3-031-75380-0\\_10](https://doi.org/10.1007/978-3-031-75380-0_10). doi:10.1007/978-3-031-75380-0\_10.
- [5] J. O. Couder, D. Gomez, O. Ochoa, Requirements verification through the analysis of source code by large language models, in: SoutheastCon 2024, 2024, pp. 75–80. doi:10.1109/SoutheastCon52093.2024.10500073.
- [6] X. Quan, M. Valentino, L. A. Dennis, A. Freitas, Verification and refinement of natural language explanations through llm-symbolic theorem proving, 2024. URL: <https://arxiv.org/abs/2405.01379>. arXiv:2405.01379.



- [7] W. Fan, M. Rego, X. Hu, S. Dod, Z. Ni, D. Xie, J. DiVincenzo, L. Tan, Evaluating the ability of large language models to generate verifiable specifications in verifast, 2025. URL: <https://arxiv.org/abs/2411.02318>. arXiv:2411.02318.
- [8] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, C. Trippel, nl2spec: Interactively translating unstructured natural language to temporal logics with large language models, 2023. URL: <https://arxiv.org/abs/2303.04864>. arXiv:2303.04864.
- [9] S. Mandal, A. Chethan, V. Janfaza, S. M. F. Mahmud, T. A. Anderson, J. Turek, J. J. Tithi, A. Muzahid, Large language models based automatic synthesis of software specifications, 2023. URL: <https://arxiv.org/abs/2304.09181>. arXiv:2304.09181.
- [10] A. Nayak, H. P. Timmapathini, V. Murali, K. Ponnalagu, V. G. Venkoparao, A. Post, Req2spec: Transforming software requirements into formal specifications using natural language processing, in: Requirements Engineering: Foundation for Software Quality: 28th International Working Conference, REFSQ 2022, Birmingham, UK, March 21–24, 2022, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2022, p. 87–95.
- [11] L. Ma, S. Liu, Y. Li, X. Xie, L. Bu, Specgen: Automated generation of formal program specifications via large language models (2024). URL: <https://arxiv.org/abs/2401.08807>. arXiv:2401.08807.
- [12] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, H. Zhang, Z. Xie, Assertllm: Generating hardware verification assertions from design specifications via multi-llms, in: 2024 IEEE LLM Aided Design Workshop (LAD), 2024, pp. 1–1. doi:10.1109/LAD62341.2024.10691792.
- [13] M. Li, W. Fang, Q. Zhang, Z. Xie, Specllm: Exploring generation and review of vlsi design specification with large language model, 2024. URL: <https://arxiv.org/abs/2401.13266>. arXiv:2401.13266.
- [14] L. M. Reinpold, M. Schieseck, L. P. Wagner, F. Gehlhoff, A. Fay, Exploring llms for verifying technical system specifications against requirements, 2024. URL: <https://arxiv.org/abs/2411.11582>. arXiv:2411.11582.
- [15] V. Gervasi, B. Nuseibeh, Lightweight validation of natural language requirements, *Softw. Pract. Exper.* 32 (2002) 113–133. URL: <https://doi.org/10.1002/spe.430>. doi:10.1002/spe.430.
- [16] Y. Xu, J. Feng, W. Miao, Learning from failures: Translation of natural language requirements into linear temporal logic with large language models, in: 2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS), 2024, pp. 204–215. doi:10.1109/QRS62785.2024.00029.
- [17] I. T. Leong, R. Barbosa, Translating natural language requirements to formal specifications: A study on gpt and symbolic nlp, in: 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2023, pp. 259–262. doi:10.1109/DSN-W58399.2023.00065.
- [18] W. Nowakowski, M. Śmiałek, A. Ambroziewicz, T. Straszak, Requirements-level language and tools for capturing software system essence, *Computer Science and Information Systems* 10 (2013) 1499–1524.
- [19] S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, W. Steiner, Arsenal: Automatic requirements specification extraction from natural language, in: S. Rayadurgam, O. Tkachuk (Eds.), *NASA Formal Methods*, Springer International Publishing, Cham, 2016, pp. 41–46.
- [20] S. J. Greenspan, A. Borgida, J. Mylopoulos, A requirements modeling language and its logic, *Information Systems* 11 (1986) 9–23. URL: <https://www.sciencedirect.com/science/article/pii/0306437986900207>. doi:[https://doi.org/10.1016/0306-4379\(86\)90020-7](https://doi.org/10.1016/0306-4379(86)90020-7).
- [21] M. Fazelnia, M. Mirakhorli, H. Bagheri, Translation titans, reasoning challenges: Satisfiability-aided language models for detecting conflicting requirements, in: *Proceedings of the 39th IEEE/ACM*

- International Conference on Automated Software Engineering, ASE '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 2294–2298. URL: <https://doi.org/10.1145/3691620.3695302>. doi:10.1145/3691620.3695302.
- [22] K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, A. Anandkumar, Leandjo: Theorem proving with retrieval-augmented language models, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), *Advances in Neural Information Processing Systems*, volume 36, Curran Associates, Inc., 2023, pp. 21573–21612.
  - [23] A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzygóźdź, P. Mił oś, Y. Wu, M. Jamnik, Thor: Wielding hammers to integrate language models and automated theorem provers, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), *Advances in Neural Information Processing Systems*, volume 35, Curran Associates, Inc., 2022, pp. 8360–8373.
  - [24] G. Granberry, W. Ahrendt, M. Johansson, Towards integrating copilotting and formal methods, in: T. Margaria, B. Steffen (Eds.), *Leveraging Applications of Formal Methods, Verification and Validation. Specification and Verification*, Springer Nature Switzerland, Cham, 2025, pp. 144–158.
  - [25] E. Mugnier, E. A. Gonzalez, R. Jhala, N. Polikarpova, Y. Zhou, Laurel: Generating dafny assertions using large language models, 2024. URL: <https://arxiv.org/abs/2405.16792>. arXiv:2405.16792.
  - [26] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, Y. Iwasawa, Large language models are zero-shot reasoners, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), *Advances in Neural Information Processing Systems*, volume 35, Curran Associates, Inc., 2022, pp. 22199–22213.
  - [27] Y. Li, B. Hui, X. Xia, J. Yang, M. Yang, L. Zhang, S. Si, L.-H. Chen, J. Liu, T. Liu, F. Huang, Y. Li, One-shot learning as instruction data prospector for large language models (2024). URL: <https://arxiv.org/abs/2312.10302>. arXiv:2312.10302.
  - [28] H. Zhang, M. Cai, X. Zhang, C. J. Zhang, R. Mao, K. Wu, Self-convinced prompting: Few-shot question answering with repeated introspection (2023). URL: <https://arxiv.org/abs/2310.05035>. arXiv:2310.05035.
  - [29] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), *Advances in Neural Information Processing Systems*, volume 35, Curran Associates, Inc., 2022, pp. 24824–24837.
  - [30] J. Li, G. Li, Y. Li, Z. Jin, Structured chain-of-thought prompting for code generation, *ACM Trans. Softw. Eng. Methodol.* 34 (2025). URL: <https://doi.org/10.1145/3690635>. doi:10.1145/3690635.
  - [31] C. Hsieh, Y. Chuang, C. Li, Z. Wang, L. T. Le, A. Kumar, J. R. Glass, A. Ratner, C. Lee, R. Krishna, T. Pfister, Found in the middle: Calibrating positional attention bias improves long context utilization, in: L. Ku, A. Martins, V. Srikumar (Eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024, Association for Computational Linguistics*, 2024, pp. 14982–14995. URL: <https://doi.org/10.18653/v1/2024.findings-acl.890>. doi:10.18653/v1/2024.FINDINGS-ACL.890.
  - [32] X. Ye, G. Durrett, The unreliability of explanations in few-shot prompting for textual reasoning, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), *Advances in Neural Information Processing Systems*, volume 35, Curran Associates, Inc., 2022, pp. 30378–30392.
  - [33] K. Shum, S. Diao, T. Zhang, Automatic prompt augmentation and selection with chain-of-thought from labeled data, in: H. Bouamor, J. Pino, K. Bali (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023, Association for Computational Linguistics*, 2023, pp. 12113–12139. URL: <https://doi.org/10.18653/v1/2023.findings-emnlp.811>. doi:10.18653/v1/2023.FINDINGS-EMNLP.811.

- [34] W. Xu, A. Banburski-Fahey, N. Jojic, Reprompting: Automated chain-of-thought prompt inference through gibbs sampling, CoRR abs/2305.09993 (2023). URL: <https://doi.org/10.48550/arXiv.2305.09993>. doi:10.48550/ARXIV.2305.09993. arXiv:2305.09993.
- [35] M. Besta, F. Memedi, Z. Zhang, R. Gerstenberger, N. Blach, P. Nyczyk, M. Copik, G. Kwasniewski, J. Müller, L. Gianinazzi, A. Kubicek, H. Niewiadomski, O. Mutlu, T. Hoefler, Topologies of reasoning: Demystifying chains, trees, and graphs of thoughts, CoRR abs/2401.14295 (2024). URL: <https://doi.org/10.48550/arXiv.2401.14295>. doi:10.48550/arXiv.2401.14295. arXiv:2401.14295.
- [36] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>.
- [37] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, A. Chadha, A systematic survey of prompt engineering in large language models: Techniques and applications (2025). URL: <https://arxiv.org/abs/2402.07927>. arXiv:2402.07927.
- [38] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, in: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022, OpenReview.net, 2022. URL: <https://openreview.net/forum?id=nZeVKeeFYf9>.
- [39] K. R. M. Leino, Dafny: An automatic program verifier for functional correctness, in: Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), volume 6355 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 348–370. URL: [https://doi.org/10.1007/978-3-642-17511-4\\_20](https://doi.org/10.1007/978-3-642-17511-4_20). doi:10.1007/978-3-642-17511-4\_20.
- [40] D. R. Cok, Openjml: Software verification for java 7 using jml, openjdk, and eclipse, in: NASA Formal Methods (NFM 2011), volume 6617 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 472–479. URL: [https://doi.org/10.1007/978-3-642-20398-5\\_35](https://doi.org/10.1007/978-3-642-20398-5_35). doi:10.1007/978-3-642-20398-5\_35.
- [41] M. R. H. Misu, C. V. Lopes, I. Ma, J. Noble, Towards ai-assisted synthesis of verified dafny methods, Proc. ACM Softw. Eng. 1 (2024). URL: <https://doi.org/10.1145/3643763>. doi:10.1145/3643763.
- [42] J. Yao, Y. Liu, Z. Dong, M. Guo, H. Hu, K. Keutzer, L. Du, D. Zhou, S. Zhang, Promptcot: Align prompt distribution via adapted chain-of-thought, in: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 7027–7037. doi:10.1109/CVPR52733.2024.00671.
- [43] A. Porshnev, et al., Modelling implicit bias in gender-career associations: A systematic comparison of language models, PsyArXiv (2025). doi:10.31234/osf.io/p7hvw\_v1, preprint, 22 May 2025.