Neuro-Symbolic Approach to Certified Scientific Software Synthesis

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

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

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

Ibrahim Mujhid
University of Hawaii at Manoa
Honolulu, HI, USA
ijmujhid@hawaii.edu

Md Rashedul Hasan
University of Nebraska-Lincoln
Lincoln, NE, USA
mhasan6@huskers.unl.edu

ABSTRACT

Scientific software development demands robust solutions to meet the complexities of modern scientific systems. In response, we propose a paradigm-shifting Neuro-Symbolic Approach to Certified Scientific Software Synthesis. This innovative framework integrates large language models (LLMs) with formal methods, facilitating automated synthesis of complex scientific software while ensuring verifiability and correctness. Through a combination of technologies including a Scientific and Satisfiability-Aided Large Language Model (SaSLLM), a Scientific Domain Specific Language (DSL), and Generalized Planning for Abstract Reasoning, our approach transforms scientific concepts into certified software solutions. By leveraging advanced reasoning techniques, our framework streamlines the development process, allowing scientists to focus on design and exploration. This approach represents a significant step towards automated, certified-by-design scientific software synthesis, revolutionizing the landscape of scientific research and discovery.

CCS CONCEPTS

• **Software and its engineering → Formal software verification**.

KEYWORDS

Certified by Design, Neuro-Symbolic Approach, Scientific Software Synthesis, Large Language Models, Formal Methods

ACM Reference Format:
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 ACM International Conference, 2024, Location.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3664646.3664776

ACM ISBN 979-8-4007-0685-1/24/07
https://doi.org/10.1145/3664646.3664776

1 INTRODUCTION

In today's era of rapid scientific advancement, software plays a crucial role in facilitating research and innovation. From multi-scale analysis to predictive modeling, scientific software serves as the backbone of modern scientific exploration. However, the development of such software presents numerous challenges, including ensuring correctness, verifiability, and adaptability to emerging architectures [5, 9, 17]. Such software must leverage secure, fault-tolerant, high-performance computing (HPC) solutions at scales of hundreds of exaflops to enable rapid prototyping, simulation, and big data analysis. The software stack of these systems relies on numerous programming frameworks and libraries, requiring programmers to have extensive knowledge for design decisions, trade-off analyses, and system configuration. Without adequate automation and tool support, these tasks become tedious, code becomes increasingly complex, and developer productivity declines [7, 10]. As scientific problems grow more complex, there is an urgent need for innovative approaches to effectively address these challenges.

Despite many advances in the use of large language models (LLMs) for code generation, current state-of-the-art techniques fall short in synthesizing code for entire scientific software stacks. They do not fully exploit emerging architectures nor provide proof that the synthesized code accurately implements the desired scientific concepts. These problems are recognized as crippling challenges in software development for complex HPC, scientific, and control systems [1, 3, 11, 13, 15–17, 20].

Motivated by the pressing demand for reliable and efficient scientific software solutions, we introduce a groundbreaking Neuro-Symbolic Approach to Certified Scientific Software Synthesis. Our motivation stems from the recognition that existing methodologies often fall short in providing scalable, verifiable, and automated solutions for complex scientific software development. Traditional approaches [14, 24, 25] rely heavily on manual intervention, leading to errors, inefficiencies, and delays in the software development lifecycle. Moreover, as scientific research pushes the boundaries of computational capabilities, there is a growing need for software that can harness the full potential of emerging architectures, such as high-performance computing (HPC) solutions.

Against this backdrop, our objectives are twofold: firstly, to streamline the process of scientific software development by leveraging cutting-edge technologies such as large language models (LLMs) [8, 28] and formal methods; and secondly, to ensure the correctness, verifiability, and scalability of synthesized software across diverse scientific domains. *By combining the power of artificial intelligence with rigorous formal reasoning, our approach aims to revolutionize the way scientific software is conceptualized, designed, and implemented.*

In this paper, we present the foundational principles of our Neuro-Symbolic Approach and outline its key components. We demonstrate how this approach empowers scientists to focus on conceptual design and exploration, while automated reasoning agents handle the intricacies of software synthesis. By embracing the concept of certified-by-design software, we aim to propel scientific research forward, unlocking new possibilities for discovery and innovation.

2 RELATED WORK

In the domain of software certification there has been a growing recognition of the limitations of traditional process-oriented standards, particularly in sectors where software plays a predominant role. While standards like those outlined by regulatory bodies such as the U.S. Food and Drug Administration (FDA) emphasize the use of specific development and testing processes, their efficacy in guaranteeing system safety and reliability, especially in software-intensive environments, has been called into question [27]. The discrepancy between adherence to process-oriented standards and actual system performance has been evident in numerous instances, such as the recurring recalls of medical devices due to software quality issues, highlighting the need for alternative certification approaches [26]. Consequently, product-based or case-based certification methodologies [18], centered on the construction of assurance cases, have gained traction among researchers and practitioners. These methodologies pivot towards explicit documentation of arguments substantiated by evidence, aiming to demonstrate that a system satisfies its critical requirements. However, challenges persist in these approaches, including the informal nature of argumentation, the lack of rigor in evidence provision, and the absence of support for software evolution and incremental updates, underscoring the imperative for advancements in certification methodologies. These challenges are more significant for AI enabled software products.

Existing research endeavors have sought to address the shortcomings of current certification paradigms by exploring avenues for formal tool support and automation in assurance case development [4, 21–23]. Pernsteiner et al. introduced a formal assurance case for a radiotherapy system, leveraging verification and analysis tools to provide evidence supporting safety requirements [22]. Similarly, Near et al. developed a formal assurance case for a proton therapy system, employing custom-tailored code analysis tools to verify code-level properties [21]. While these efforts represent strides towards formalization and automation in assurance case construction, the process of developing assurance cases and selecting appropriate evidence-generating tools remains largely manual. Moreover, a body of prior work has focused on specification inference and extensible type systems to synthesize operational specifications and facilitate the application of multiple type systems within a single language [2, 12]. These endeavors, while valuable, underscore the need for further advancements in automated analysis synthesis frameworks, like the proposed NS-CSD approach, to enhance the rigor, efficiency, and scalability of software certification processes in scientific domains.

3 PARADIGM SHIFT: AUTOMATED REASONING IN LLM-BACKED CERTIFIED BY DESIGN SOFTWARE SYNTHESIS

Our Certified by Design approach marks a departure from established practices in scientific software development, ushering in a new era where software engineering aligns closely with the principles of true engineering. By leveraging formal methods, we aim to shift the focus of scientists from manual coding to conceptual design, creativity, and idea exploration. Central to our approach is the creation of a mathematical model of architectural design, allowing for the synthesis of solutions and rigorous validation.

Our approach, termed Neuro-Symbolic Methodology for Certified Software Design (NS-CSD), integrates neuro-symbolic techniques with formal methods to revolutionize the development of scientific software. The NS-CSD framework aims to automate the generation of complex scientific software while ensuring its correctness, security, and other quality attributes. Key components include:

- **Neuro-Symbolic Architecture:** We envision a novel architecture that combines the power of large language models (LLMs) with symbolic reasoning capabilities. This architecture, named Scientific and Satisfiability-Aided Large Language Model (SaSLLM), leverages advances in chain-of-thought (CoT) reasoning to decompose complex scientific concepts into verifiable hypotheses. SaSLLM uses *Satisfiability modulo theories (SMT)* solvers [6] facilitates automated reasoning, verification, and code synthesis.
- **Scientific Domain Specific Language (DSL):** this AIWare relies on an automated approach to generate a DSL tailored to represent scientific reasoning problems in a format accessible to domain scientists. This DSL enables symbolic reasoning and proof checking while maintaining interpretability for non-experts.
- **Generalized Planning for Abstract Reasoning:** To address feasibility analysis challenges, we employ a Generalized Planning (GP) framework [19] for abstract reasoning. GP solvers are utilized to model abstraction and scientific reasoning problems, allowing for efficient and effective handling of complex scientific concepts across disciplinary boundaries, reasoning about the validity of scientific concepts while generating code.
- **Certified by Design Foundation:** Our approach is rooted in the Certified by Design paradigm, shifting the focus of scientific software development towards automated synthesis and certification. Through symbolic AI reluing on mathematical modeling and formal methods, we ensure the correctness, security, and other quality attributes of the synthesized software.
- **Integration and Verification:** NS-CSD integrates LLMs, symbolic reasoning, DSL, and GP solvers to transform scientific concepts into verified software. We employ automated

verification techniques to ensure the correctness of the synthesized code and produce proofs of correctness for each stage of the software development lifecycle.

By combining neuro-symbolic techniques with formal methods, the NS-CSD approach offers a revolutionary way to synthesize and certify scientific software, paving the way for enhanced reliability, security, and innovation in scientific research and engineering.

4 EARLY DEMONSTRATION: APPLYING NS-CSD TO CLIMATE MODELING

Scenario: A research institute aims to enhance its climate modeling software to incorporate new scientific findings about cloud feedback mechanisms rapidly. These mechanisms are critical for understanding climate sensitivity to greenhouse gas emissions but involve complex interactions that are difficult to model. The goal is to use the proposed framework to automate the generation of updated simulation code that accurately reflects the latest research on cloud feedbacks.

In our framework, the Neuro-Symbolic Methodology for Certified Software Design (NS-CSD), we follow a series of steps to automate the synthesis of scientific software while ensuring its correctness, security, and other quality attributes.

Step 1: Encoding Scientific Knowledge - We begin by formally encoding the latest scientific theories and findings about cloud feedback mechanisms into a Scientific Domain Specific Language (DSL). This DSL represents the reasoning problem in a format accessible to domain scientists, making it suitable for symbolic reasoning and proof checking.

Step 2: Symbolic Reasoning with SaSLLM - Next, we leverage our novel architecture, the Scientific and Satisfiability-Aided Large Language Model (SaSLLM), to perform symbolic reasoning. SaSLLM decomposes the encoded scientific concepts into verifiable hypotheses using advances in chain-of-thought (CoT) reasoning. This enables automated reasoning, verification, and code synthesis.

Step 3: Generalized Planning for Abstract Reasoning - To address feasibility analysis challenges, we employ a Generalized Planning (GP) framework for abstract reasoning. GP solvers model abstraction and reasoning problems, enabling efficient handling of complex scientific concepts across disciplinary boundaries.

Step 4: Automated Code Generation - Based on the decomposed scientific concepts and verified hypotheses, the GP solvers generate new or modified code segments for the climate model simulations. This includes integrating new equations, adjusting parameters, and ensuring that the generated code adheres to scientific standards.

Step 5: Integration and Verification - Finally, we integrate the synthesized code segments into the existing climate modeling software. Automated verification techniques are then employed to ensure the correctness of the synthesized code, producing proofs of correctness for each stage of the software development lifecycle.

5 CONCLUSION

The paradigm shift towards automated reasoning in LLM-backed Certified by Design synthesis represents a transformative approach to scientific software development. By leveraging neuro-symbolic architectures, DSL, and generalized planning frameworks, this methodology streamlines the transformation of scientific concepts into robust software solutions. With a focus on verifiability, scalability, and efficiency, this approach promises to revolutionize the traditional practices of software engineering, empowering scientists to concentrate on conceptual design and exploration of ideas. As we continue to advance in this direction, the potential for accelerating scientific discovery and innovation is immense, promising a future where the boundaries of possibility in software synthesis are continually pushed forward.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their valuable comments. This work was supported in part by awards CCF-1755890, CCF-1618132, CCF-2139845, CCF-1943300 and CCF-2124116 from the National Science Foundation.

REFERENCES

[1] [n. d.]. DOE Workshop report: The 2015 Cybersecurity for Scientific Computing Integrity - Research Pathways and Ideas Workshop. https://science.energy.gov/~{}/media/ascr/pdf/programdocuments/docs/ASCR_Cybersecurity_20_Research_Pathways_and_Ideas_Workshop.pdf. Accessed: 2019-04-28.

[2] Dalal Alrajeh, Jeff Kramer, Alessandra Russo, and Sebastin Uchitel. 2009. Learning operational requirements from goal models. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*. IEEE Computer Society, USA, 265–275. https://doi.org/10.1109/ICSE.2009.5070527

[3] Rizwan A. Ashraf, Saurabh Hukerikar, and Christian Engelmann. 2018. Pattern-based Modeling of Multiresilience Solutions for High-Performance Computing. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, Berlin, Germany, April 09-13, 2018*. 80–87. https://doi.org/10.1145/3184407.3184421

[4] Hamid Bagheri, Eunsuk Kang, and Niloofar Mansoor. 2020. Synthesis of assurance cases for software certification. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results* (Seoul, South Korea) *(ICSE-NIER '20)*. Association for Computing Machinery, New York, NY, USA, 61–64. https://doi.org/10.1145/3377816.3381728

[5] David E. Bernholdt, Benjamin A. Allan, Robert C. Armstrong, Felipe Bertrand, Kenneth Chiu, Tamara Dahlgren, Kostadin Damevski, Wael R. Elwasif, Thomas Epperly, Madhusudhan Govindaraju, Daniel S. Katz, James Arthur Kohl, Manojkumar Krishnan, Gary Kumfert, Jay Walter Larson, Sophia Lefantzi, Michael J. Lewis, Allen D. Malony, Lois C. McInnes, Jarek Nieplocha, Boyana Norris, Steven G. Parker, Jaideep Ray, Sameer Shende, Theresa L. Windus, and Shujia Zhou. 2006. A Component Architecture for High-Performance Scientific Computing. *IJHPCA* 20, 2 (2006), 163–202. https://doi.org/10.1177/1094342006064488

[6] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[7] Anshu Dubey and Lois Curfman McInnes. 2017. Proposal for a Scientific Software Lifecycle Model. In *Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational and Data-enabled Science & Engineering* (Denver, CO, USA) *(SE-CoDeSE'17)*. ACM, New York, NY, USA, 22–26. https://doi.org/10.1145/3144763.3144767

[8] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1469–1481.

[9] Wolfgang Frings, Dong H. Ahn, Matthew LeGendre, Todd Gamblin, Bronis R. de Supinski, and Felix Wolf. 2013. Massively Parallel Loading. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing* (Eugene, Oregon, USA) *(ICS '13)*. ACM, New York, NY, USA, 389–398. https://doi.org/10.1145/2464996.2465020

[10] Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. 2015. The Spack Package Manager: Bringing Order to HPC Software Chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Austin, Texas) *(SC '15)*. ACM, New York, NY, USA, Article 40, 12 pages. https://doi.org/10.1145/2807591.2807623

[11] Joshua Garcia, Mehdi Mirakhorli, Lu Xiao, Sam Malek, Rick Kazman, Yuanfang Cai, and Nenad Medvidovic. 2023. SAIN: A Community-Wide Software Architecture INfrastructure. In *45th IEEE/ACM International Conference on Software Engineering: ICSE 2023 Companion Proceedings, Melbourne, Australia, May 14-20, 2023*. IEEE, 336–337. https://doi.org/10.1109/ICSE-COMPANION58688.2023.00095

[12] Dimitra Giannakopoulou, Corina S. P"s"reanu, and Howard Barringer. 2002. Assumption Generation for Software Component Verification. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE '02)*. IEEE Computer Society, USA, 3.

[13] Michael A. Heroux. 2009. Software Challenges for Extreme Scale Computing: Going From Petascale to Exascale Systems. *The International Journal of High Performance Computing Applications* 23, 4 (2009), 437–439. https://doi.org/10.1177/1094342009347711 arXiv:https://doi.org/10.1177/1094342009347711

[14] Sheng-Kuei Hsu and Shi-Jen Lin. 2011. MACs: Mining API code snippets for code reuse. *Expert Systems with Applications* 38, 6 (2011).

[15] Saurabh Hukerikar and Christian Engelmann. 2017. Resilience Design Patterns: A Structured Approach to Resilience at Extreme Scale. *CoRR* abs/1708.07422 (2017). arXiv:1708.07422 http://arxiv.org/abs/1708.07422

[16] Jinseong Jeon, Xiaokang Qiu, Jeffrey S. Foster, and Armando Solar-Lezama. 2015. JSketch: sketching for Java. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*. 934–937. https://doi.org/10.1145/2786805.2803189

[17] D. S. Katz, L. C. McInnes, D. E. Bernholdt, A. C. Mayes, N. P. C. Hong, J. Duckles, S. Gesing, M. A. Heroux, S. Hettrick, R. C. Jimenez, M. Pierce, B. Weaver, and N. Wilkins-Diehr. 2019. Community Organizations: Changing the Culture in Which Research Software Is Developed and Sustained. *Computing in Science Engineering* 21, 2 (March 2019), 8–24. https://doi.org/10.1109/MCSE.2018.2883051

[18] T. Kelly and R. Weaver. 2004. The Goal Structuring Notation–A Safety Argument Notation. In *Dependable Systems and Networks (DSN) Workshop on Assurance Cases*.

[19] Chao Lei, Nir Lipovetzky, and Krista A. Ehinger. 2024. Generalized Planning for the Abstraction and Reasoning Corpus. arXiv:2401.07426 [cs.AI]

[20] Patrick S. et al. McCormic. 2014. Exploring the Construction of a Domain-Aware Toolchain for High-Performance Computing.. In *2014 Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*. 1–10.

[21] J. P. Near, A. Milicevic, E. Kang, and D. Jackson. 2011. A Lightweight Code Analysis and Its Role in Evaluation of a Dependability Case. In *ICSE*. ACM, 31–40.

[22] S. Pernsteiner, C. Loncaric, E. Torlak, Z. Tatlock, X. Wang, M. D. Ernst, and J. Jacky. 2016. Investigating Safety of a Radiotherapy Machine Using System Models with Pluggable Checkers. In *Proceedings of CAV*. 23–41.

[23] Rui Qiu, Corina S. Pasareanu, and Sarfraz Khurshid. 2016. Certified Symbolic Execution. In *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9938)*, Cyrille Artho, Axel Legay, and Doron Peled (Eds.). 495–511. https://doi.org/10.1007/978-3-319-46520-3_31

[24] Armando Solar-Lezama. 2009. The Sketching Approach to Program Synthesis. In *Proceedings of the 7th Asian Symposium on Programming Languages and Systems* (Seoul, Korea) *(APLAS '09)*. Springer-Verlag, Berlin, Heidelberg, 4–13. https://doi.org/10.1007/978-3-642-10672-9_3

[25] Saurabh Srivastava, Sumit Gulwani, Swarat Chaudhuri, and Jeffrey S. Foster. 2011. Path-based Inductive Synthesis for Program Inversion. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation* (San Jose, California, USA) *(PLDI '11)*. ACM, New York, NY, USA, 492–503. https://doi.org/10.1145/1993498.1993557

[26] U.S. Food and Drug Administration (FDA). [n. d.]. List of Device Recalls. https://www.fda.gov/medicaldevices/safety/listofrecalls. Accessed: 2018-11-14.

[27] U.S. Food and Drug Administration (FDA). 2017. General principles of software validation; final guidance for industry and FDA staff. https://www.fda.gov/downloads/medicaldevices/.../ucm085371.pdf.

[28] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 37:1–37:12. https://doi.org/10.1145/3597503.3623316