My research interests lie in automated software engineering, with a focus on enhancing developer productivity and improving code quality. My work primarily explores program analysis and synthesis, software security and architecture, formal methods, and the application of LLMs. With the growing role of LLMs in software engineering, I foresee a renewed emphasis on formal approacheseither to verify the correctness of generative AI outputs or to directly produce provably correct outputs using LLMs. Building on my prior and current research, my future work seeks to integrate formal methods with ML-supported program analysis and synthesis to address emerging challenges in this evolving domain.

Previous Research Experience. During my bachelor's and master's studies, I built a strong foundation in software engineering theories. Over fourteen years in the software industry, I gained extensive practical experience and insights into challenges faced by software engineers, such as production delays and mismatches between desired specifications and actual implementations. These experiences motivated me to develop effective solutions that enable programmers to create correct, trustworthy software while streamlining the production process. During my Ph.D., I investigated various facets of automated software engineering, focusing on software security, program analysis, and synthesis. My work incorporated aspects of program comprehension, repository mining, API recommendation, formal methods, and ML to address challenges within these domains.

– **Software Security:** In software security, I focused on identifying legacy security flaws from an architectural perspective. This work included examining software quality attribute traceability in legacy systems, software certification, and detecting security tactics [3]. Additionally, I applied machine learning models (e.g., BERT and traditional models) to identify security issues arising from the absence of appropriate architectural tactics [1].

– **Program Analysis:** My work on program analysis addressed challenges related to code quality and implementation correctness using techniques such as dataflow and control-flow analysis. This included inter-procedural program analysis to detect incorrect software architectural tactic implementations and provide recommendations for their resolution [7, 6]. To overcome the limitations of state-of-the-art call-graph constructors in handling dynamic programming features, we developed a method for constructing sound call-graphs for object serialization [2]. Additionally, my work addressed a central challenge in binary analysis through the development of a formally verified binary pointer analysis, which provides guarantees for the mapping between pointers and memory locations (conditionally accepted to ICSE'25).

– **Program Synthesis:** In program synthesis, I developed an inter-procedural approach that goes beyond recommending program fixes. This approach addresses the synthesis of interrelated but discrete code snippets that implement a use case and automatically integrates them into existing code bases [4, 5]. This work earned the First-place award in the *ASE'21* research competition and underscores the importance of (semi) formal synthesis, even in the current era of ML-based automated programming. In addition to my Ph.D. research, I undertook two Ph.D. research internships at Google and Palo-Alto Research Center (PARC), where I contributed to projects on program analysis and software synthesis, focusing on ML-based generative models and evolutionary search-based synthesis. These efforts resulted in two inventions (pending patents) and two papers currently under preparation.

Current Research. During my postdoctoral research, I expanded my focus to formal methods and LLMs, two areas that I believe are increasingly important given the rise of unverified ML-generated outputs. My current research encompasses formally verified software securitysuch as binary pointer analysis and software compartmentalizationand LLM-supported code translation. The former focuses on (i) developing a formally verified framework for reasoning about pointers' attributes in binaries (conditionally accepted at ICSE'25) and (ii) leveraging formal methods to mitigate CVE exploits through software compartmentalization (manuscript in preparation). In the latter, I am developing a formally verified LLM-supported approach for translating C to safe Rust, which has formed the basis for a DARPA proposal submission (as co-PI).

Future Research. Building on my past and current research expertise, I plan to pursue three primary research directions that align with the topics of interest at leading software engineering conferences (e.g., ICSE, ASE, FSE, OOPSLA) and journals (e.g., TSE, ToSEM, IEEE Software), as well as the priorities of major funding agencies, including NSF, DARPA, and DoE.

– **Formally Verified ML-supported Inter-procedural Programming:** While automated programming and program synthesis hold promise, current methods often struggle with real-world programming challenges. Most approaches focus on intra-procedural synthesis, generating isolated code blocks based on specifications. However, real-world programming frequently requires inter-procedural synthesis to construct interrelated code blocks across different program components. During my Ph.D., I developed a semi-formal approach for inter-procedural synthesis to implement architectural tactics and integrate them into programs, addressing real-world challenges. I am now extending this work with formal methods for greater precision and reliability. Additionally, I have prototyped an approach combining LLMs with formal verification for program translation, submitted as a DARPA proposal, where I am a co-PI.

Looking ahead, my goal is to leverage formal methods and language models in two key areas: (i) extracting precise specifications from loosely defined requirements and (ii) establishing a robust cycle of code generation and formal verification by integrating tools such as SMT solvers and theorem provers for semantic equivalence assessment and feedback. These advancements aim to provide a strong foundation for reliable and efficient code generation. Drawing on my experience from industry internships at Google, where I worked on LLM-based synthesizers, and my postdoctoral projects, I aim to realize this approach within five years. This work seeks to bridge formal methods with practical software development. As an extension of my previous research, this direction aligns with NSF grant opportunities, with DARPA serving as a potential alternative funding source, provided a relevant BAA aligns with this work.

– **Large-scale Code Repair:** Expanding on my previous research, I plan to develop a hybrid solution that integrates formal and ML-based approaches to address the limitations of current large-scale code repair methods. Existing methods often require highly specific input specifications, limiting their robustness across diverse codebases.

Over the next 5-8 years, I aim to create a hybrid solution that combines the rigor of formal methods with the scalability of ML-based approaches, particularly leveraging LLMs. This solution will identify code sections requiring repair, generate patches automatically, and integrate them seamlessly into existing codebases. Beyond simple bug identification, this approach offers a comprehensive solution for large-scale code maintenance and repair. This research aligns with the funding priorities of agencies such as DARPA, DoE, DoD, and NSF. The project's foundation in prior research makes it well-suited for NSF funding, while its ambitious scope aligns with DARPA's objectives, provided it addresses their specific priorities.

– **Self-evolvable Software:** Over the next 5-10 years, I aim to develop a framework for self-evolving, reconfigurable software systems. While existing research focuses on isolated aspects like API migration, code repair, or ML-based code transitions, there is an opportunity to create a unified framework that consolidates these efforts.

My vision is to develop a generalized approach that learns from various software evolution scenarios, enabling adaptive and autonomous software reconfiguration. This framework will integrate API migration, code repair, and ML-based transitions, providing a comprehensive solution for self-evolving software. By creating systems capable of autonomously adapting to changing requirements and environments, this work aims to advance the resilience and dynamism of software systems. This ambitious research direction aligns with DARPA's mission. While DARPA frequently issues project solicitations via BAAs, researchers can also propose ideas directly to program managers for project-specific funding opportunities.

Research Grant Proposal. During my Ph.D. and postdoctoral research, I contributed to grant proposals for NSF, DARPA, and DoE, playing a pivotal role in submitting two DARPA grant proposals as PI (BAA number HR001123S0039) and co-PI (BAA number DARPA-PS-24-20). These collaborative efforts involved partnerships with institutions such as the University of Notre Dame, Virginia Tech, Penn State, SUNY-Stony Brook, Korea University, and SRI International.

My experience in submitting grant proposals has provided valuable insights into articulating research objectives, designing methodologies, and defining expected outcomes effectively. Leading and co-leading DARPA proposals has demonstrated my ability to manage the complexities of interdisciplinary research while facilitating collaboration across institutions and researchers.

As I advance in my academic career, I am committed to continuing grant proposal submission across various funding agencies. Through these efforts, I aim to secure support for innovative research that leverages my expertise in automated software engineering, formal methods, and ML-based approaches. By collaborating with a diverse network of researchers, I look forward to leading and contributing to projects that address critical challenges in software engineering.

References.

[1] Ahmet Okutan, Ali Shokri, Viktoria Koscinski, Mohamad Fazelinia, and Mehdi Mirakhorli. A novel approach to identify security controls in source code. *arXiv preprint arXiv:2307.05605*, 2023.

[2] Joanna CS Santos, Mehdi Mirakhorli, and Ali Shokri. Seneca: Taint-based call graph construction for java object deserialization. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):1125–1153, 2024.

[3] Joanna CS Santos, Ali Shokri, and Mehdi Mirakhorli. Towards automated evidence generation for rapid and continuous software certification. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 287–294. IEEE, 2020.

[4] Ali Shokri. A program synthesis approach for adding architectural tactics to an existing code base. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1388–1390. IEEE, 2021.

[5] Ali Shokri. *Inter-procedural Program Synthesis for Automatic Architectural Tactic Implementation.* PhD thesis, Rochester Institute of Technology, 2023.

[6] Ali Shokri and Mehdi Mirakhorli. Arcode: A tool for supporting comprehension and implementation of architectural concerns. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 485–489. IEEE, 2021.

[7] Ali Shokri, Joanna CS Santos, and Mehdi Mirakhorli. Arcode: Facilitating the use of application frameworks to implement tactics and patterns. In *2021 IEEE 18th International Conference on Software Architecture (ICSA)*, pages 138–149. IEEE, 2021.