Research Plan

Integrating LLM into automation of TEE specification and verification

Theorem proving is a fundamental approach in formal methods for system verification. Due to its high cost, researchers have explored many techniques to automate the whole process, e.g., Vampire and E in automated theorem proving, and Isabelle's sledgehammer in interactive theorem proving. However, these works still require considerable human efforts in system specification and verification. Recently, the advancement of deep learning, especially the emergence of large language models (LLM), has sparked a notable surge of research exploring these techniques to enhance the process of theorem proving. For example, DSP, Baldur, Leandojo, and Thor.

A promising research topic is to integrate LLM in the automated verification of complex systems, i.e., trusted execution environment (TEE). I am currently studying the security of TEE using theorem prover Isabelle/HOL. TEE is an isolated environment dedicated to safeguarding sensitive data within electronic devices and is implemented in different architectures such as ARM TrustZone and Intel SGX. My research focuses on ARM TrustZone-based TEEs, e.g., OPTEE, open-TEE, and QSEE. My previous works model TEE as multiple abstraction layers and prove properties by refinement. However, the source code layer of TEEs is not modeled and verified due to their complexity. Thus I aim to develop an automated approach for verifying TEEs' source code to uncover their vulnerabilities. Recently, LLMs have shown their great impact on automated theorem proving since the work of GPT-f. Researchers are dedicated to improving the LLM performance on the miniF2F dataset, a cross-system benchmark for formal olympiad-level mathematics. They improve the correctness rate from 20% to nearly 60%, further demonstrating the potential of integrating LLMs with theorem proving. As far as I know, fewer works concentrate on the application of LLMs in a specific domain such as system verification. Thus I aim at integrating LLM into automatically specifying and verifying TEEs to find bugs and design effective tools for this process.

For a detailed plan, I summarize it as four critical steps.

Firstly, I should have a comprehensive understanding of LLM for formal methods. This can be achieved through reading and analyzing research papers and concluding their advantages. I have read some papers about autoformalization, premise selection, and proof search. They made optimizations on automatic theorem proving from different perspectives. For example, DSP formalizes mathematical theorems written in natural language to solve data scarcity. Autospec optimizes the formalization of complex data structures through prompt engineering. Leandojo extracts training data from mathlib and implements the interactions with Lean. LEGO employs a growing library containing verified lemmas to augment the capability of theorem proving. Thor leverages LLM to generate the proof framework and leaves the proofs of subgoals to sledgehammer in Isabelle. Baldur creates a repair model to utilize the error messages returned from the theorem prover to improve accuracy. In my opinion, these innovations can be integrated to assist the automation of TEE verification, which requires further survey. Although most works are not open-source, projects such as Leandojo and LEGO-prover have uploaded their code and model for reference. I can have a deep understanding of these methods by reproducing them.

Secondly, I should capture the features of TEE, particularly its source code. Most TEEs are written in C/C++ which brings serious bugs. From a syntactical view, TEE source code mainly contains if clauses and while loops. It also contains pointers, linked lists, and nested loops which introduce difficulties of autoformalization. From a semantic view, TEE is designed to maintain an isolated secure environment while communication between the normal and secure environment is complex. Tailored properties are required for TEE's specific vulnerabilities to ensure its safety and security. My previous works concentrated on the abstraction layers of TEE and did not delve into its source code. Thus more reading of open-source TEE code is necessary. Besides, I proved information-flow security properties and refinement relations before. However, such properties are not easy to automatically prove. It may be more realistic to find a simpler property, e.g., functional correctness.

Thirdly, I should integrate LLM into verifying the specific properties for TEE. Two popular ways of using LLM are fine-tuning and calling the GPT interface. Fine-tuning is more expensive and requires a large number of training data, which is currently scarce in system verification. While chatGPT is cheaper and promising with few-shot prompting. I also did some experiments on TEE verification using chatGPT4. The preliminary result shows that GPT can generate useful specifications and proof steps with minor errors. My elementary roadmap is as follows. First, generate Isabelle/HOL's specification from TEE's source code, e.g., OPTEE. This process and related data structure can refer to the paper Autospec. We need to make some modifications to TEE's source code, e.g., simplifying some codes and structures unrelated to TEE's safety and security, and rewriting some codes for better formalization while maintaining their semantic identity. Second, find a way to generate the preconditions and postconditions for the Isabelle/HOL code. Many TEE research papers can contribute to this process. Third, using agent prompting to guide GPT. We can learn from Leandojo and DSP to set up the gym-like interactive environment with Isabelle/HOL. We can set up a repair agent to learn feedback from Isabelle/HOL following the work Baldur and build a growing theorem library following LEGO. For proof search, we can follow the best-first search or MCTS. A more promising approach is to generate the entire proof structure by LLMs and solve individual subgoals through the sledgehammer, which is more efficient than a GPT model. After the verification, we need to analyze those results from LLM that cannot pass the check of Isabelle/HOL. We then find and summarize bugs in the source code.

Finally, I intend to design a useful tool for automated TEE verification to lower human efforts and create a benchmark. As I know, only two interactive tools on Lean, namely LeanCopilot and LLMstep, are open-sourced. It would be beneficial to develop Interactive tools on Isabelle/HOL that integrate LLMs. A tool may be preferable for a broader application of LLM rather than competing with other tools on a fixed dataset. Moreover, pioneers are creating benchmarks for different domains such as miniF2F. Training data can be extracted from the TEE formalization and build a benchmark for system verification following Leandojo. We can also extract data from not only TEE but also existing projects such as seL4. Furthermore, I intend to write a paper for a top conference in formal methods, e.g., FM and CAV.

Jilin Hu, PhD student
Zhejiang University