

Introduction to the HICSS-58 Minitrack on Use of LLMs for Program Analysis and Generation

Mark Sherman
Software Engineering Institute
Carnegie Mellon University
mssherman@sei.cmu.edu

There has been an explosion of work using Large Language Models (LLMs) to assist with the development and evaluation of programs. There is a wide diversity of opinion, ranging from one extreme claiming that LLMs will replace programmers, to suggesting that the use of LLMs will fast track poor and vulnerable code into enterprise systems, and to expectations that LLMs will generate malicious code.

There is a growing collection of papers examining the potential role of LLMs in the software development process. Certain focused applications, such as code completion, have wide adoption and acceptance. This application is well-defined and practitioners generally recognize whether the suggested completion matches their expectation. The limitation of scope mitigates potential errors generated by an LLM.

Recently, specific applications of LLMs have claimed significant results. For example, Andy Jassy, President and CEO, Amazon, recently posted on LinkedIn of experiences at Amazon building and using a specially trained LLM to upgrade Java 8 and Java 11 source code to Java 17 (Jassy 2024). This exercise appears to have the advantage of the transformation being limited to a specific set of languages, source code presumably written to a standard set of practices, leveraging OpenRewrite recipes, and targeting applications for which existing test suites were available to evaluate the translated code. Nevertheless, the reported gains illustrate the potential value in using LLMs.

Other studies considering the more general application of building new source code from scratch or fixing defects in existing source code have mixed results (Sherman 2024). Many times, the programs are small or not representative, e.g., finding greatest common divisor (Sobania 2023) or use common patterns, such as a web application that does a database lookup (Perry 2023). The general consequence derived from these experiences is to use the developer to guide the LLM in

performing the task, usually through some form of prompt engineering (Wei 2023). From this perspective, what is emerging is a generalized version of the code completion task, starting with a less defined starting point. In this spirit, this minitrack contains papers that illustrate experiments in having a programmer guide LLMs to fix or generate code for specific needs.

The first paper discusses the need to build specialized source code analysis tools, customized to either the desired evaluations or the tool deployment environment. The experiment involved collaboration with tool developers and users to create and modify the generated tool for their specific needs.

The second paper considers the more general defect analysis and fixing problem. The focus is how an LLM might integrate into the larger development environment, using existing static code analysis tools and theorem provers. The paper describes a method using prompt engineering, where a technique is illustrated for generating the prompts automatically.

The experiments reported here add to the community's collective wisdom as the search continues for the best way to use LLMs in the development process.

References

- Jassy, A., Sept 2024, https://www.linkedin.com/posts/andy-jassy-8b1615_one-of-the-most-tedious-but-critical-tasks-activity-7232374162185461760-AdSz/
- Perry, N., Srivastava, M., Kumar, D., & Boneh, D., "Do Users Write More Insecure Code with AI Assistants?," CCS '23, November 26-30, 2023, Copenhagen, Denmark, <https://doi.org/10.1145/3576915.3623157>
- Sherman, M., "Should I Trust ChatGPT to Review My Program," InfoSec World, Sept 25-27, 2023, <https://apps.dtic.mil/sti/trecms/pdf/AD1210424.pdf>
- Sobania, D., Briesch, M., Hanna, C., & Petke, J., "An Analysis of the Automatic Bug Fixing Performance of

ChatGPT," Jan 20, 2023,
<https://arxiv.org/pdf/2301.08653v1>
Wei, Y, Xia, C, & Zhang, L., "Copiloting the Copilots:
Fusing Large Language Models with Completion
Engines for Automated Program Repair," ESEC/FSE
'23, December 3–9, 2023, San Francisco, CA, USA,
<https://doi.org/10.1145/3611643.3616271>

Acknowledgements

Carnegie Mellon University 2024

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References herein to any specific entity, product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute nor of Carnegie Mellon University - Software Engineering Institute by any such named or represented entity.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM24-1372