Large Language Models for development of Haskell programs

Students

Linus Flury

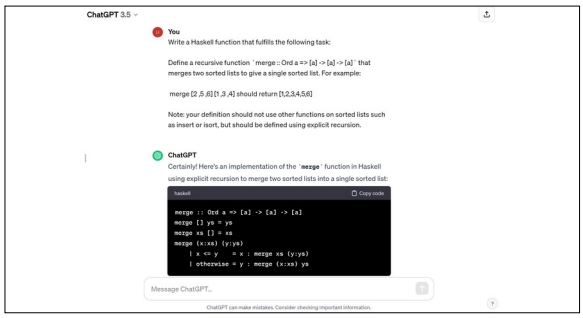
Dominik Castelberg

Introduction: In recent years, Large Language Models (LLMs) have become valuable tools for developers with their ability to quickly scaffold code and act as an impactful accelerator for development teams around the world. With code being heavily standardized on a global scale and an abundance of training data freely available on platforms such as GitHub, it is easy to intuit possible reasons for their performance. While these assumptions hold true for popular languages such as Java, Python or C#, there has been little research in the usage of LLMs as development tools for less commonly used languages such as Haskell. With this project we aim to explore LLM based development support for Haskell and develop an environment, in which such research can be conducted in a quick and efficient manner.

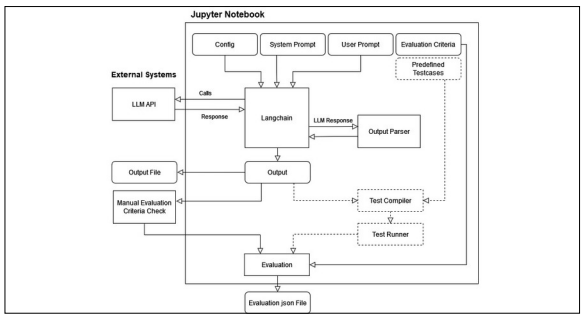
Approach: Four state-of-the-art models (Llama 2, Code Llama, GPT-3.5 and GPT-4) were evaluated based on their performance on tasks typically faced by an automated development support tool. The tasks were scoped and classified into three major categories: Code Generation, Debugging and Testing. For each of these categories, quantifiable metrics for the evaluation of the model performance were defined. These criteria must be interpretable as quantifiable metrics to allow a comparative analysis between models. These metrics were then weighted according to their importance, based on the insight of experts in the field of Haskell development. Each task was executed with three sorting algorithms of varying cognitive complexity in sample implementations. Cognitive complexity was selected as the complexity measure after careful evaluation, to ensure that the ordering of the algorithms is based on complexity of interpretability.
Utilizing cognitive complexity inspired us to frame LLMs as entities whose performance can be analysed through the lens of cognitive load theory. This enabled the differentiation between errors caused by the complexity of a provided algorithm (intrinsic load) and errors caused by unclear instructions (extraneous load).
To accelerate the evaluation processes, a development environment, enabling both the automated testing of generated Haskell code using Jupyter notebooks and the utilization of cloud hosted models with modern LLM tooling like LangChain, was created.

Conclusion: Our work has shown that there are large gaps in the output quality of each model. We have encountered outliers, but we are confident that these outliers were not caused by the intrinsic complexity of the algorithms provided and can be explained with extraneous complexity that lead to the model not understanding the task. This problem can be resolved with further prompt engineering and fine-tuning, which is why we are optimistic about the viability of models such as GPT-4 or Code Llama as supporting tools for Haskell development.
Using Chain of Thought Prompting, which leads models to break down given tasks into sequential subtasks, tends to increase the output quality in general. However the sequential nature of it lead to inconsistencies in the compositions of Haskell's higher-order functions. It lead the models to neglect critical nuances in function composition, resulting in erroneous code generation.

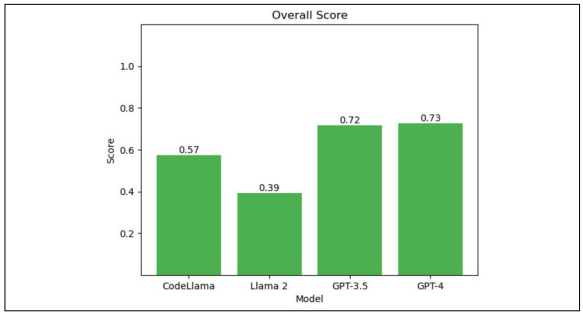
Screenshot of how a student would use Chat GPT 3.5 to help with a task. Accessed on 17.12.2023, https://chat.openai.com/
Own presentment



Jupyter Notebook Setup. Dotted elements are for usecases involving automated code testing.
Own presentment



Normalized average score overall. Models: CodeLlama-13b-instruct, Llama-2-13b, GPT-3.5 Turbo, GPT-4
Own presentment



Advisor
Prof. Dr. Mitra Purandare

Subject Area
Software, Miscellaneous

Project Partner
Zühlke Engineering AG, Schlieren, Zürich