Die Hausaufgaben müssen von jedem Studierenden einzeln bearbeitet und abgegeben werden. Für die Hausaufgabe sind die aktuellen Informationen vom Blog https://seblog.cs.uni-kassel.de/ss24/generative-ki-in-der-software-technik/ zu berücksichtigen.

Abgabefrist ist der 01.07.2024 – 23:59 Uhr

Abgabe

Wir benutzen für die Abgabe der Hausaufgaben Git. In eurem Repository soll sowohl der aktuelle Stand des Berichtsheftes, als auch die zu den jeweiligen Aufgaben gehörenden Ergebnisse an generiertem Code oder anderen Artefakten persistiert werden.

Wir nutzen **weiterhin** das Repository, was in Hausaufgabe 1 bereits unter folgendem Link angelegt wurde:

https://classroom.github.com/a/XeAtEq1Z

Die Bearbeitung der gestellten Aufgaben muss wie in Vorlesung und Übung besprochen in einem einheitlichen Berichtsheft dokumentiert werden.

- **Handschriftliche (Teil-)Berichte werden nicht gewertet.**
- **Nicht dokumentierte (Teil-)Berichte werden mit 0 Punkten bewertet!**
- **Nicht persistierte Lösungen für Aufgaben wie Code oder andere Artefakte führen zu 0 Punkten für diese Aufgabe.**
- **Das Berichtsheft muss als PDF-Datei (.pdf) abgegeben werden. Jedes Experiment ist in einem eigenen Kapitel anzulegen.**

Bericht

Jedes Experiment soll im Berichtsheft auf einer neuen Seite beginnen, sodass eine klare Unterteilung zwischen den einzelnen Experimenten gegeben ist. Ein Experiment ist schriftlich in die Abschnitte **Vorbereitung**, **Durchführung** und **Auswertung** zu unterteilen (je nach Aufgabentyp kann dies ergänzt werden).

Die Gesamtlänge eines Berichts (**minus der Abbildungen**) sollte in etwa bei 1–2 Din A4 Seiten (*Schriftgröße 11–12*) liegen. Wir empfehlen die Anfertigung des Berichtsheftes mit Latex.

Vorbereitung

Pflichtlektüre

Pflichtlektüre ist, sich das Paper „Towards Translating Real-World Code with LLMs: A Study of Translating to Rust“ anzuschauen.

Aufgabe 1 – Refactoring

In dieser Aufgabe soll mithilfe der KI ein Refactoring für eine gegebene Codebasis umgesetzt werden. Der gegebene Code ist in Java geschrieben.

Als Szenario für diese Aufgabe nehmen wir an, dass eine alte Codebasis vorliegt. Diese scheint schlimmer „overengineering“ Code zu sein. Der Code behandelt ein Tool für den Tennissport. Es ist bereits ein Test vorhanden, welcher als korrekt angesehen werden kann.
Wir bekommen die Aufgabe, den Code möglichst gut zu „verschlanken“. Die Methoden `wonPoint()` und `getScore()` müssen in ihrer aktuellen Signatur bestehen bleiben (der Rumpf kann angepasst werden). Ziel ist es, die Anzahl der Zeilen so gut wie möglich zu minimieren; sie sollen aber weiterhin einen menschenlesbaren Codestil besitzen. Das Verhalten des Ausgangscodes ist beizubehalten. Dies kann dadurch sichergestellt werden, dass der Test weiterhin durchläuft.
Als Grundlage für einen möglichen Pull Request für den neuen Code soll die KI den Code nicht nur refactoren, sondern auch die einzelnen Verbesserungen erklären.

Codebasis

Die zugrundeliegende Javadatei können hier heruntergeladen werden:

`https://github.com/sekassel/gkistss24-files`

Folgende Dateien sind für die Aufgabe relevant:

- `Aufgabenblatt 4/tennis/TennisGame.java`
- `Aufgabenblatt 4/tennis/TennisGameTest.java`

Programmverhalten
Das Programm kann genutzt werden, um die Punkte für ein Tennisspiel zu tracken und den korrekten Punktestand anzusagen. (Sets und Matches werden hier ignoriert!)[1]

Bericht

Der durchlaufene Prozess der Codegenerierung soll im Berichtsheft festgehalten werden. Es gilt für diese Aufgabe freie Wahl zwischen ChatGPT und Gemini als Generierungstool. Die Wahl des Tools ist zu dokumentieren.
Dokumentieren Sie in Ihrer Evaluation explizit, inwieweit die im Szenario festgelegten Ziele erreicht werden konnten.

[1] De Punktansage im Tennis folgt einem nicht intuitiven System und kann hier zum Verständnis nachgelesen werden: `https://sammancoaching.org/kata_descriptions/tennis.html`.

Aufgabe 2 – Documentation

In dieser Aufgabe soll mithilfe der KI für eine gegebene Codebasis eine Dokumentation generiert werden. Der gegebene Code ist in Java geschrieben.

Als Szenario für diese Aufgabe nehmen wir an, dass wir den Legacy-Code für ein altes Projekt bekommen. Dieser ist in einer Sprache geschrieben, die wir eventuell nicht selbst beherrschen. Wir bekommen die Aufgabe, den Code zu verstehen und dann eine Code-Dokumentation in Form von Kommentaren anzufertigen. Ziel ist es, den gegebenen Code möglichst vollständig von der KI und mit möglichst keinem Eigenaufwand zu dokumentieren.

Codebasis

Die zugrundeliegende Javadatei können hier heruntergeladen werden:

`https://github.com/sekassel/gkistss24-files`

Folgende Dateien sind für die Aufgabe relevant:

- `Aufgabenblatt 4/legacy/Timer.java`

Bericht

Der durchlaufene Prozess der Codegenerierung soll im Berichtsheft festgehalten werden. Es gilt für diese Aufgabe freie Wahl zwischen ChatGPT und Gemini als Generierungstool. Die Wahl des Tools ist zu dokumentieren.
Dokumentieren Sie in Ihrer Evaluation explizit, inwieweit die im Szenario festgelegten Ziele erreicht werden konnten (eventuelle weitere Qualitätseigenschaften der Tests können informell diskutiert werden).

Aufgabe 3 – Cross-Compiling

In dieser Aufgabe soll mithilfe der KI eine Übersetzung für eine gegebene Codebasis in eine andere Programmiersprache durchgeführt werden. Der gegebene Code ist in **Go** geschrieben.

Als Szenario für diese Aufgabe nehmen wir an, dass wir eine Codebasis für ein aktuelles Projekt bekommen. Der Kunde wünscht sich eine Adaption der Anwendung für mobile Geräte. Wir bekommen die Aufgabe, einen Teil der Anwendung in **Java** zu überführen. Zusätzlich ist unser Vorgesetzter daran interessiert, wie die vorliegenden Konzepte in Java übersetzt worden sind.
Ziel ist es, den gegebenen Code möglichst ohne Eigenaufwand übersetzen zu lassen und eine Erklärung/Beschreibung des übersetzten Codes zu bieten. Beim Ausführen des Teilcodes gibt dieser eine Berechnungsdauer aus. Der Kunde wünscht sich, dass die Berechnungszeit möglichst nahe am Original liegt. Außerdem soll überzeugend dargelegt werden, dass der neu übersetzte Code funktional äquivalent zum Ausgangscode ist.

Codebasis

Die zugrundeliegende Javadatei können hier heruntergeladen werden:

`https://github.com/sekassel/gkistss24-files`

Folgende Dateien sind für die Aufgabe relevant:

- `Aufgabenblatt 4/translate/main.go`

Programmverhalten
Das Programm berechnet eine Annäherung von PI mithilfe der Monte-Carlo-Simulation. Darüber hinaus wird Parallelisierung (präziser: Concurrency) genutzt, um die Berechnungszeit zu minimieren. (Im Code finden sich weitere hilfreiche Kommentare.)

Bericht

Der durchlaufene Prozess der Codegenerierung soll im Berichtsheft festgehalten werden. Es gilt für diese Aufgabe freie Wahl zwischen ChatGPT und Gemini als Generierungstool. Die Wahl des Tools ist zu dokumentieren.
Dokumentieren Sie in Ihrer Evaluation explizit, inwieweit die im Szenario festgelegten Ziele erreicht werden konnten.

Hinweis! Diese Aufgabe ist hochexperimentell. Sollte sich herausstellen, dass eine Übersetzung nur mit exponentiellem Zeitaufwand durchführbar ist, dokumentieren Sie lediglich ihre Zwischenergebnisse und brechen sie die weitere Bearbeitung ab.

Anhang

Es folgt eine Auflistung hilfreicher Links und Quellen zu den Themen dieser Veranstaltung. Die Links sind als Hilfestellung zur zur Bearbeitung der Aufgaben angedacht. Alle pdf-Dateien sind ebenfalls auf dem Discord Server einsehbar.

Quellen

- Das Paper „Towards Translating Real-World Code with LLMs: A Study of Translating to Rust“: https://github.com/sekassel/gkistss24-files