14.10.2025

Lab

Software Development Tools
Winter Semester 25/26

TuCan-No: 20-00-0673-pr
Course Type: 4SWS / 6 CPs
Workload: **~180hours**

Prof. Dr.-Ing. Mira Mezini

Process

- **By Friday, October 17th, 23:59 :** Send an e-mail with your three preferred topics and why you are the right person for these topics to: julius.naeumann@tu-darmstadt.de
  - Subject: "**SDT Project**"
  - If you register as a team, send a **single email** with all members (include their mat-nr and email addresses)
- **Wednesday, October 22nd:** Assignment of topics
- **Next:** *Contact your supervisor* to discuss details of your topic
- **During Lab:** Bi-weekly meetings with supervisor in an agile process
  - Discuss the current state and the next steps
- **End of March:** Final submission of artifacts

CodeQL Query Dataset

CodeQL is a static analysis framework that allows analyzing Codebases through “Queries”. CodeQL can be used to find vulnerabilities in code. We would like to explore the capability of LLMs in writing these queries, and evaluate the capability empirically.

Task: Create a dataset such as HumanEval to Benchmark the ability of LLMs to write CodeQL queries. The dataset should include Problem Statement, Canonical Solution, Tests (Including code to analyze, if applicable)

Suitable for: 2 – 4 people

Languages and Frameworks: Python

Starting point:

- IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities: https://arxiv.org/abs/2405.17238
- CodeQL: https://codeql.github.com/docs/codeql-overview/
- HumanEval: https://github.com/openai/human-eval

Contact: julius.naeumann@tu-darmstadt.de

Evaluate LLMs on Static Analysis Tasks

Static Analysis determines static information of entire programs, like data flows, call chains, and immutability. Given the success of LLMs on code understanding, the question arises: how do LLMs perform compared to existing static analysis frameworks on static analysis tasks? For this comparison, a suitable framework is required to run LLMs on static analysis benchmarks.

Task: Build an extendable framework to evaluate LLMs (like GPT, Claude, Gemini, etc.) on existing static analysis benchmarks. As a first step, integrate an immutability analysis benchmark into it.

Suitable for: 2 – 4 people

Languages and Frameworks: Python, Java

Starting point:

- Do Code LLMs Do Static Analysis? https://arxiv.org/abs/2505.12118
- CORE: Benchmarking LLMs Code Reasoning Capabilities through Static Analysis Tasks: https://arxiv.org/abs/2507.05269
- CiFi: Versatile Analysis of Class and Field Immutability: https://ieeexplore.ieee.org/document/9678903

Contact: daniel.maninger@tu-darmstadt.de & roth@cs.tu-darmstadt.de

Retriever for OpenAPI Specifications

Retrieval-Augmented Generation (RAG) is a technique to enrich the context of a Large Language Model (LLM) with relevant information. This way, they can base their response on the provided facts rather than having to rely on memorized knowledge only. We have developed a code generation benchmark that targets web API invocations, and would now like to evaluate LLMs equipped with RAG on this benchmark. OpenAPI is the specification standard used to document the correct usage of said web APIs.

Task: Implement a retriever that returns relevant chunks from a given web API's OpenAPI specification. Given the special setting, we will develop custom similarity measurement, chunking, and truncation approaches.

Suitable for: 1 – 3 people

Languages and Frameworks: Python

Starting point:

- RAG: https://en.wikipedia.org/wiki/Retrieval-augmented_generation
- OpenAPI: https://github.com/OAI/OpenAPI-Specification

Contact: daniel.maninger@tu-darmstadt.de

Encoding of Tabular Data

Equation Discovery in AI for Science is a field focused on deriving equations from experimental data, typically stored in tables. Recently, deep learning methods—originally designed for sequential text—have been adapted to guide this search. However, their robustness to table variations remains unclear.

Task:

In this project, we aim to investigate how these approaches behave when rows and columns are permuted, Non-standard value ranges are used, column labels are changed, the number of rows and columns is altered, and so on.

Suitable for: 2–4 people

Languages and Frameworks: Python

Starting point: https://arxiv.org/abs/2503.16953

Contact: jannis.brugger@tu-darmstadt.de

Agent Benchmark

SWE-bench Verified is a curated benchmark of 500 real GitHub issues and fixes, each human-verified to ensure the tasks are clearly defined and solvable by an automated system. It serves as a rigorous evaluation set for testing software-engineering agents on realistic bug-fixing and feature-implementation tasks. SWE-agent, an open-source LLM-powered system, is designed to tackle these tasks autonomously by reading, editing, and testing code to produce working pull requests.

Task: Build a benchmarking tool that runs SWE-agent with multiple large language models—such as Claude Sonnet, GPT, and Qwen—on the SWE-bench Verified dataset, using their API endpoints as inputs. The tool should record which models successfully solve each task and generate a comparative performance report based on each task, and analyze the reason for consistently failed tasks.

Suitable for: 2 – 3 people

Languages and Frameworks: Python

Starting point:

- SWE Agent: https://github.com/SWE-agent/SWE-agent
- SWE-Bench Verified: https://huggingface.co/datasets/princeton-nlp/SWE-bench_Verified

Contact: amir.molzam@tu-darmstadt.de

Vulnerability Benchmark

Software vulnerabilities are often categorized using the Common Weakness Enumeration (CWE), which provides a standardized way to describe security flaws in code. To effectively evaluate coding agents and large language models (LLMs) on their ability to handle such weaknesses, it is useful to evaluate them based on coding tasks that target specific CWEs.

Task: Create a tool that provides an API taking two inputs — a CWE number and a desired programming language — and generates a coding task to evaluate coding agents and LLMs with respect to that CWE. The tasks should be based on an integrated repository of tasks from existing CWE benchmarks (SecurityEval, CodeSecEval, SecCodePLT, CWEval-Bench, CyberSecEval).

Suitable for: 2 – 3 people

Languages and Frameworks: Python

Starting point:

- Common Weakness Enumeration: https://cwe.mitre.org/
- https://huggingface.co/datasets/s2e-lab/SecurityEval
- other benchmarks

Contact: amir.molzam@tu-darmstadt.de

Reminder

Please register for the TUCAN course 20-00-0673-pr

Positions & Theses

There are openings for HiWi positions and Master thesis at the STG lab, focusing on AI-based code generation. If you have a strong interest in this area, please contact Amir Molzam Sharifloo (amir.molzam@tu-darmstadt.de). Preference will be given to Master's or Bachelor's students with solid systems and programming skills.