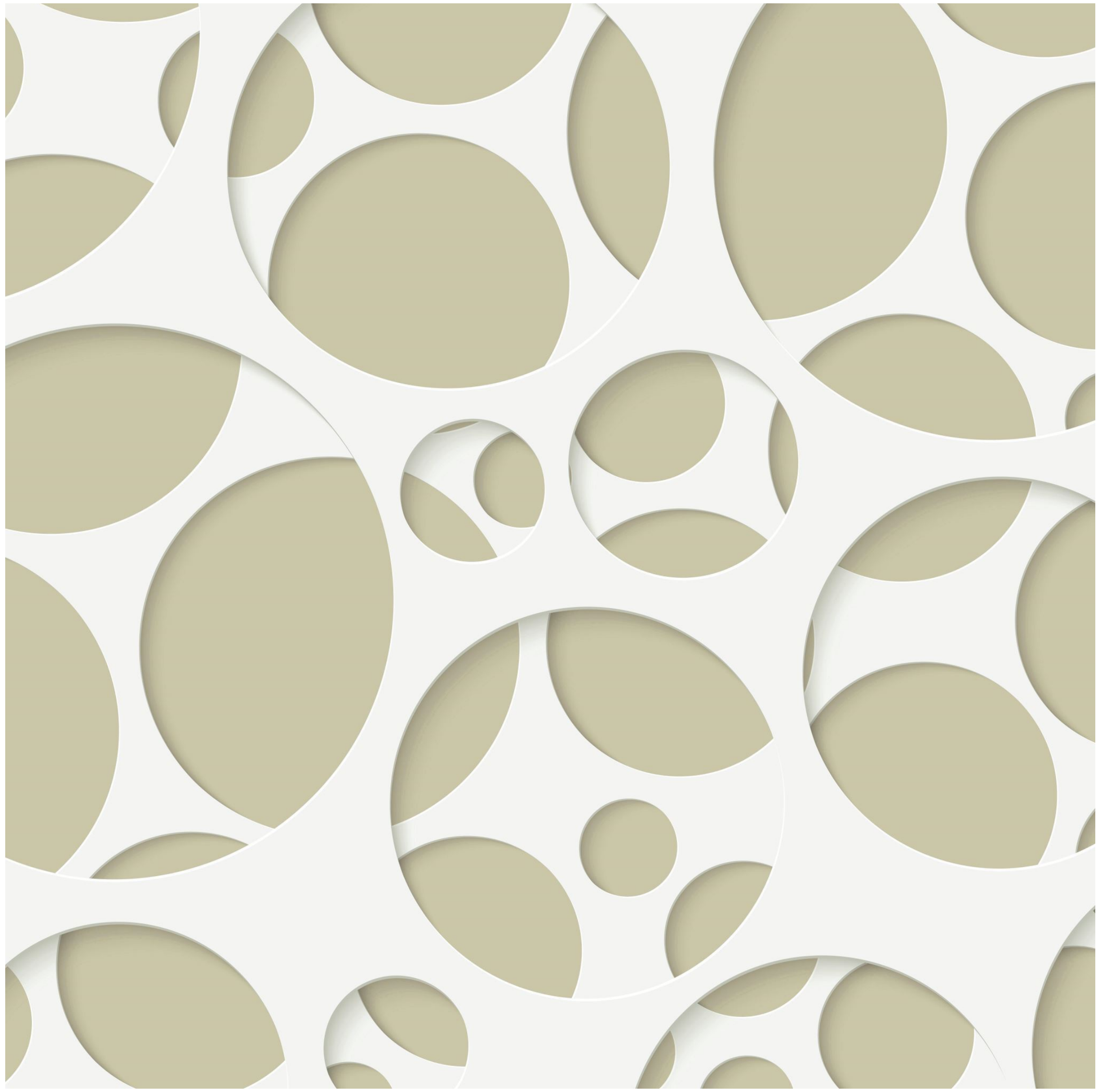
Automatic Test Generation With LLMs Using Natural Language Artifacts

Aakash Reddy Bhoomidi

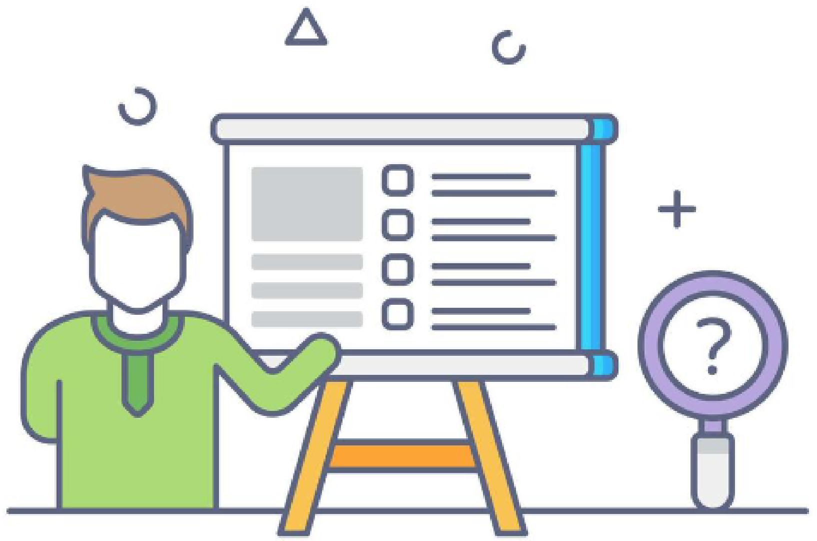
Tejas Kumar Leelavathi

Varsha Chandrahasareddy Mulangi

Veda Varshitha Oruganti



Objective



- The problem we are solving is the challenge of automatically generating test cases from natural language software requirements with minimal human intervention.

- The primary goal of our research project is to harness the power of LLM to generate a broad range of test cases directly from natural language requirements, aiming for good code coverage.

- This involves accurately identifying not only the explicit requirements but also boundary cases and exceptional cases, thereby enhancing the reliability and efficiency of the software testing process.

Motivation

- **Increased Efficiency**

  Generates large volumes of test cases quickly Reduces time to market for software products

- **Enhanced Test Coverage**

  Automatically covers a wide range of scenarios, including edge cases Improves software reliability and performance

- **Resource Optimization**

  Frees up human resources for complex testing and analysis Allows focus on innovative and critical problem

- **Cost Reduction**

  Minimizes manual effort in test creation and execution Reduces overall software development and maintenance costs

- **Quality Improvement**

  Ensures consistent and comprehensive testing Detects defects early, improving software quality

Research Questions

- ➢How effectively does ChatGPT 3.5 convert natural language requirements into functional test cases that achieve comprehensive code coverage?

- ➢Does fine-tuning ChatGPT 3.5 on a specific dataset improve its test case generation capabilities, especially in producing test cases that meet code coverage objectives, including generating tests for boundary conditions and exceptional cases?

- ➢Can the fine-tuned ChatGPT 3.5 model effectively generalize its capabilities to new requirement documents not included in the training data?

Approach

- We have divided our approach into 3 different sections answering our 3 Research Questions:

Section 1

- Generating test cases using ChatGPT 3.5 and ECMA 262 specifications.
- Aim to generate a comprehensive set of 1000 test cases per function, incorporating random inputs to ensure a broad testing scope that covers various potential use cases and scenarios.
- Test implementation of Node.js and Rhino using these test cases.
- Employ an iterative approach, regenerating test cases as needed until they execute without errors

Section 2

- ➢ Create dataset from natural language description and test cases generated by SWAMI
- ➢ Fine-tune ChatGPT 3.5 with the dataset.
- ➢ Focusing on boundary conditions and exceptional cases.
- ➢ Proceeding to the following stage akin to Section 1

Section 3

- Generating test cases using fine-tuned ChatGPT 3.5 and ECMA 334 specifications.
- Aim to generate a comprehensive set of 1000 test cases per function, incorporating random inputs to ensure a broad testing scope that covers various potential use cases and scenarios.
- Employ an iterative approach, regenerating test cases as needed until they execute without errors
- Execute on Roslyn, a C# compiler.
- Evaluate the results.

Why we think this approach has a good chance of being successful?

•Requirement documents explicitly outline the intended functionalities and user expectations for the software. A reliable requirement document ensures that the test cases generated are fully aligned with the project's goals and stakeholders' expectations.

•This alignment is crucial for validating that the software meets its intended purposes and delivers value to its users. Leveraging Large Language Models enables the AI to comprehend requirements and specifications that are written in plain English or any other language the model supports.

Why we think this approach has a good chance of being successful?

- This alignment is critical because it ensures that the software is tested against the actual needs and expectations of the users and stakeholders, rather than just the technical implementation.
- The purpose of LLMs is to comprehend and handle human language. They are able to interpret the minute details found in requirement documents, such as context, limitations, and exceptions, which other test generation tool could miss.
- Since we are generating 1000 test cases for one functionality, we're able to extend our coverage across more sections of the code. Furthermore, this approach enables us to uncover exceptional scenarios that hadn't been anticipated before.

Metrics

- **Code Coverage** measures the percentage of the software's code that is executed when the test suite runs. High code coverage suggests that a large portion of the code has been tested, which theoretically reduces the chance of undiscovered bugs.

- **Branch coverage** a subset of code coverage, specifically measures the coverage of conditional branches in the codebase. This metric assesses whether both the true and false branches of each conditional statement (such as if-else statements) have been executed during testing. Achieving high branch coverage means that the test cases have effectively exercised different decision-making paths in the code, which is essential for validating the software's logic and correctness

Thank you