An Experimental Study of Automated Test Case Generation with LLM

A comprehensive evaluation of the effectiveness of Large Language Models in generating automated test cases

Presenter: Puspanjali Sarma | **University:** Liverpool John Moore University, UK | **Date:** June 2024

Introduction

Overview of ATCG

- Automated Test Case Generation (ATCG) is a crucial aspect of software testing aimed at improving efficiency and coverage by automatically generating test cases based on certain criteria or models.
- ATCG helps in ensuring that software systems are robust, reliable, and meet their specified requirements.

Introduction to Large Language Models (LLMs)

- Large Language Models, like GPT-3.5 and StarCoder, are advanced AI models trained on vast amounts of text data to understand and generate human-like text.
- These models have shown great potential in various applications, including natural language processing, text generation, and now, automated test case generation.

Introduction (continued)

Challenges in Traditional Methods

Manual creation: time-consuming, error-prone

Script-based/model-based: lacks flexibility/comprehensiveness

Potential of LLMs in Generating Test Cases

Automate the generation of diverse and relevant test cases

Adaptable to different coding styles and software requirements

Enhance coverage and efficiency in software testing processes

Overview of LLM Applications in the Software Industry

Used for code completion and bug detection

Aid in documentation and code reviews

Facilitate automated testing and quality assurance

Enhance productivity and reduce manual effort in software development

Context

Unit testing is a critical part of software development, but it can be time-consuming and tedious. This often leads developers to skip writing unit test cases, which can lead to software defects and increased development costs.

What is a unit test case?

- A unit test case is a test that verifies the functionality of a single unit of code. A unit of code can be a function, method, class, or module.
- Unit tests are typically written by developers and are executed as part of the software development process.

Why is the adoption of unit testing among developers low?

- It can be time-consuming & tedious
- Lack of knowledge and experience
- No reward for writing unit tests
- Poor design and documentation of legacy code
- Complex systems and applications

What are the options?

- Generative AI has the potential to automate unit testing, freeing up developers to focus on more creative and strategic tasks

Aim & Objective

- **Aim:**

  To evaluate the feasibility and effectiveness of using LLMs, specifically OpenAI GPT-3.5 and StarCoder, for automated test case generation.

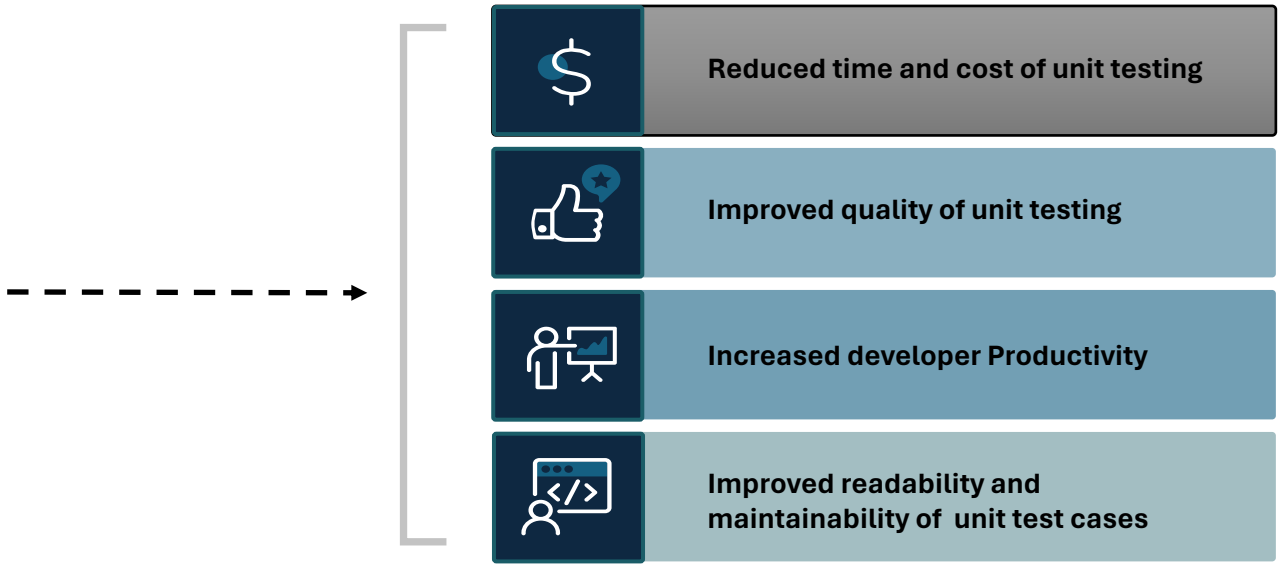
- **Objectives:**
  - Develop an application for LLM-based test case generation.
  - Assess the efficacy and efficiency of LLM-generated test cases compared to traditional methods.
  - Identify the strengths and limitations of each LLM in the context of ATCG.

Problem Statement

Writing unit test cases for ServiceNow code can be **time-consuming** and **tedious**. This leads to **developers not writing unit tests, low-quality unit tests, and outdated unit tests**. Autogenerating unit test cases can address these problems, but there are challenges such as making it **understand Glide classes**, generating comprehensive test cases and keeping them up-to-date.

Solution Overview

Autogenerate production-grade (low-latency, accurate, and consistent) unit test cases for ServiceNow code in the Jasmine framework using OpenAI GPT models. The LLM is made to understand JavaScript and Glide classes using a corpus of code and unit test cases. This allows the model to generate high-quality unit test cases, covering most of the possible scenarios.



Literature Review: Background

Manual Test Case Creation

Involves human testers designing test scenarios based on system requirements.

Flexible but time-consuming and error-prone.

Script-Based Test Case Generation

It uses predefined scripts to simulate user interactions.

It is efficient for regression testing but lacks flexibility and is difficult to maintain.

Model-Based Testing

Uses system models to derive test cases.

Ensures comprehensive coverage but relies heavily on accurate and complete models.

Literature Review: LLM-Based Approaches

Limitations of Traditional Approaches

Prone to human error and bias, scalability issues, and lack of adaptability.

Integration of LLMs in Test Case Generation

LLMs can generate diverse and relevant test cases by understanding the structure and logic of code.

They can adapt to new requirements and changes in the software, making them highly versatile.

Comparative Analysis of LLMs

Studies have shown varying levels of success with different LLMs in generating test cases.

GPT-3.5 and StarCoder represent state-of-the-art models with significant potential in this domain.

Research Methodology

Methodology

- **Experiment Design**:
  - Autogenerate unit test cases in Jasmine Framework for ServiceNow JavaScript code.
  - Create a test case generation solution using OpenAI GPT model.

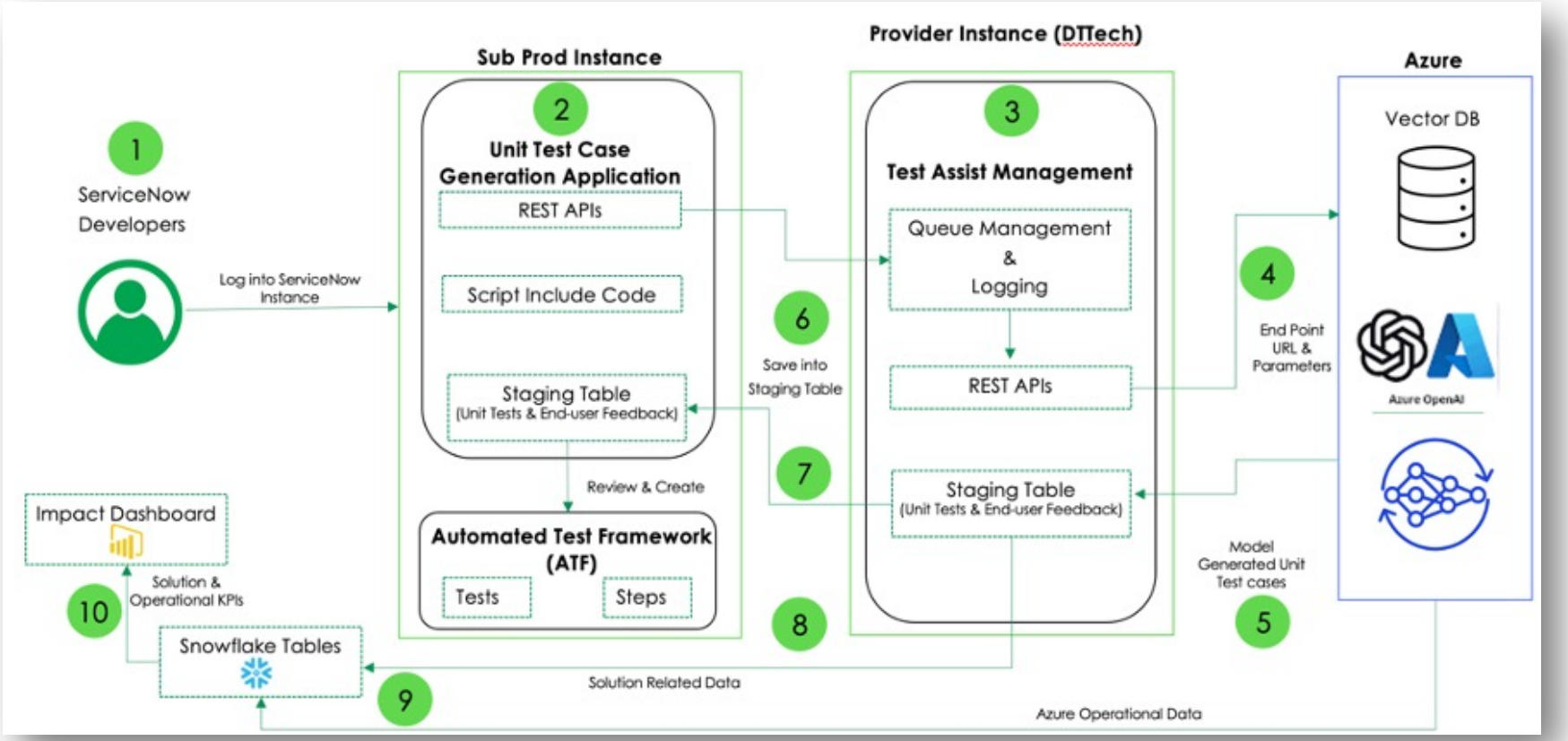
Data Selection and Collection

- **Data Sources**:
  - ServiceNow-specific code from ServiceNow's Code Snippets community repository.
  - "The Stack" dataset from Hugging Face library, including licensed source code files in 358 programming languages.

Data Pre-processing

- **Steps to pre-process datasets**
  - Tokenization
  - Special Token Handling
  - Code Formatting
  - Language-Specific Pre-processing
  - Handling Dependencies and Imports
  - Error Handling and Exception Management
  - Removing Noise
  - Variable Renaming
  - Dataset Cleaning and Augmentation
  - Code Annotation (Optional)
  - Normalization and Standardization
  - Length Limitation

Solution Architecture



Results Analysis

- **GPT-3.5 Performance**:
  - Superior in test case generation time and fault detection rate.
  - High code coverage and efficiency.
  - Generates concise and highly relevant test cases.
- **StarCoder Performance**:
  - Slightly higher accuracy but longer test cases.
- **Comparison with Industry Benchmarks**:
  - Both models outperform industry benchmarks in several key metrics.
  - GPT-3.5's test cases are more practical for real-world applications.
- **Example Test Cases**
  - **Code Snippet**: Returns the number of active incidents assigned to a user.
  - **Expected Test Case**: Verify correct number of active incidents.
- **Generated Test Cases**:
  - GPT-3.5: Matches expected test cases, concise.
  - StarCoder: Matches expected test cases, longer.

Metric	StarCoder	GPT-3.5	Industry Benchmark
Test Case Generation Time	5 minutes	3 minutes	4 minutes
Code Coverage	85%	92%	88%
Fault Detection Rate	78%	90%	85%
Efficiency(Test Cases/Minute)	30	40	35
Average Length of Test Cases	7.24 lines	5.95 lines	N/A
Number of Unique Test Cases	41	37	N/A

Summary of Results

- **Summary Insights**
  - GPT 3.5 Turbo generates accurate unit test cases for well-known programming languages like JavaScript.
  - Detecting Glide classes remains challenging for GPT models.
  - Prompt engineering greatly influences Language Models (LMs) responses.
  - Using Vector DB as an alternative enhances solution performance by providing context.
  - Key Vector DB steps: Document Split, Tokenization, Embedding, Similarity Search, and Retrieval.
  - Consistent LM responses achieved by adjusting parameters like temperature and top_p.
  - End-to-end implementation requires thorough testing by Subject Matter Experts (SMEs).
  - Proficiency in Microsoft Azure is crucial for developing reliable solutions.

- **Conclusion**
  - **GPT-3.5 is the preferred choice** for automated test case generation due to its balance of speed, coverage, fault detection, and practicality in real-world applications.
  - StarCoder performs well but is less efficient and comprehensive compared to GPT 3.5.
  - Advanced language models like GPT 3.5 can significantly improve software quality and reliability.

Conclusion

Discussion and Conclusion

- **Experiments and Findings**:
  - LLMs generate valid and diverse unit tests given appropriate input-output examples and domain embeddings.
  - Tests are executable and verifiable using standard frameworks.
  - Domain embeddings enhance LLMs' ability to generate relevant tests.
  - Fine-tuning on domain-specific corpora (Stack Overflow, GitHub) improves test quality and coverage.
  - Optimal domain embeddings depend on component complexity (simple components: Stack Overflow; complex components: GitHub).
  - LLMs support multiple programming languages with varying test quality based on embedding availability.
  - Test quality and coverage vary with granularity levels (functions, classes, modules).

Recommendations

- **For Practitioners and Researchers**:
  - Use LLMs as complementary tools for generating and executing unit tests, especially for complex components.
  - Fine-tune LLMs on relevant domain-specific corpora for improved test quality and coverage.
  - Configure LLMs for appropriate programming languages and granularity levels to ensure valid and compatible tests.
  - Maintain human supervision to verify test correctness and completeness.

Future Recommendations

- Explore other methods to obtain domain embeddings, such as self-supervised learning, meta-learning, or multi-task learning, and compare their performance and effectiveness with fine-tuning.
- Investigate the impact of different LLM architectures, such as GPT-4, GPT 4.o, LLAMA, Mistral, BERT, or XLNet, on the test generation and execution, and evaluate their trade-offs in terms of speed, accuracy, and scalability.
- Extend the scope of testing to include other types of software components, such as web services, databases, or graphical user interfaces, and other aspects of testing, such as test case prioritization, test suite minimization, or test oracle generation.
- Develop methods to evaluate and improve the reliability, robustness, and security of the tests generated by LLMs, and detect and mitigate potential biases, errors, or vulnerabilities.

Thank You!