An experimental study of Automated Test Case Generation with LLMs

Puspanjali Sarma

Research Proposal

February 2024

# ABSTRACT

Recently, pre-trained large language models (LLMs) have emerged as a ground-breaking technology in the fields of natural language processing and artificial intelligence. These models demonstrate exceptional performance across a wide range of tasks and possess the capability to handle large-scale datasets effectively. Concurrently, software testing stands as a crucial undertaking, playing an essential role in ensuring the quality and reliability of software products. As the scope and complexity of software systems continue to expand, the demand for more successful software testing techniques becomes increasingly urgent. This creates an opportune area for advanced approaches, such as incorporating LLMs.

This research paper delves into the utilization of Large Language Models (LLMs) within the realm of automated test case generation. Covering various aspects such as background information, problem statement, research inquiries, aims and objectives, significance of the study, scope, research methodology, and necessary resources, the paper offers a thorough examination. Furthermore, it provides a detailed research plan to steer the implementation of the study.

# LIST OF FIGURES

[Figure 1.1: Basic Architecture to Integrate LLM with ATCG framework…………………... 12](#_Toc160268188)

[Figure 1.2: Project Plan………………………………………………………………………. 15](#_Toc160268189)

# LIST OF ABBREVIATIONS

LLM…………...…. Large Language Model

ATGC…...........…...Automated Test Case Generation

GPT…………….... Generative pre-trained

QA………………...Quality Assurance

**Table of Contents**

[ABSTRACT 2](#_Toc169446224)

[LIST OF FIGURES 3](#_Toc169446225)

[LIST OF ABBREVIATIONS 4](#_Toc169446226)

[1. Background 6](#_Toc169446227)

[2. Problem Statement OR Related Research OR Related Work 7](#_Toc169446228)

[3. Research Questions (If any) 8](#_Toc169446229)

[4. Aim and Objectives 8](#_Toc169446230)

[5. Significance of the Study 9](#_Toc169446231)

[6. Scope of the Study 9](#_Toc169446232)

[7. Research Methodology 9](#_Toc169446233)

[8. Requirements Resources 13](#_Toc169446234)

[8.1 Hardware Requirements 13](#_Toc169446235)

[8.2 Software Requirements 13](#_Toc169446236)

[8.3 Dataset Specifications 13](#_Toc169446237)

[8.4 Language Model Training Details 14](#_Toc169446238)

[9. Research Plan 14](#_Toc169446239)

[References 16](#_Toc169446240)

# 1. Background

Automated Test Case Generation with Large Language Models (ATCG) involves utilizing these sophisticated AI models to automatically create test cases for software applications. LLMs, equipped with a deep understanding of natural language and programming concepts, can analyze software requirements, specifications, and code snippets to generate test cases that validate the functionality, robustness, and edge cases of the software. By comprehending the intended behaviour of the system and its potential usage scenarios, LLMs can generate test cases that cover a wide range of conditions, reducing manual effort and enhancing test coverage.

However, Automated Test Case Generation with LLMs encounters several challenges. Firstly, accurately understanding complex software requirements and specifications from natural language descriptions can be challenging due to ambiguities, implicit constraints, and varying interpretations. Moreover, generating test cases that cover a comprehensive range of scenarios while avoiding redundancies requires a deep understanding of the software's functionality and potential edge cases. Additionally, ensuring that the generated test cases are efficient, relevant, and comprehensive demands careful consideration of the system's behaviour under various conditions.

To surmount the hurdles in Automated Test Case Generation with LLMs, various strategies can be adopted. These include fine-tuning LLMs with domain-specific data pertinent to software testing, alongside integrating specialized architectures that focus on comprehending software requirements and code structures. Such measures can augment the model's capacity to produce precise and diverse test cases. Implementing feedback mechanisms involving human validation and continual refinement can elevate the quality of generated test cases through iterative learning from the outputs. Additionally, merging LLMs with automated techniques for test case prioritization and optimization can ensure the creation of impactful and efficient test suites, tackling the challenges of comprehensiveness and efficiency in test case generation. In summary, a blend of domain-specific training, model enhancements, iterative refinement processes, and judicious utilization can mitigate the obstacles associated with Automated Test Case Generation using LLMs. We will explore the potential of large language models such as GPT or StarCoder in automatically generating test cases for software applications, with a specific emphasis on evaluating the effectiveness, coverage, and efficiency of the generated test cases.

# 2. Problem Statement OR Related Research OR Related Work

The ever-growing complexity of software applications, coupled with the increasing demand for rapid and reliable testing, poses a significant challenge in the field of software engineering (J. Shore and S. Warden, 2021) (K, 2000) (Siddiqui, 2021). Traditional methods of manual test case generation are not only labour-intensive but also struggle to keep pace with the dynamic nature of modern software (E. Daka and G. Fraser, 2014). Despite advancements in automated testing, existing approaches often fall short of achieving comprehensive effectiveness, coverage, and efficiency in the generation of test cases.

Current automated test case generation techniques, while providing some level of assistance, frequently lack the adaptability and contextual understanding required to thoroughly exercise the intricate functionalities within diverse software applications. This limitation results in suboptimal test coverage, leaving potential vulnerabilities and undiscovered bugs (M. M. Almasi, n.d.) (Zalewski, 2023) (G. Grano, 2018) (A. Panichella, 2020) (E. Daka, 2015). As software systems continue to evolve in complexity, a critical need arises for innovative approaches that can enhance the efficacy of test case generation, ensuring a thorough examination of software functionalities.

In this context, the utilisation of Large Language Models (LLMs) like GPT (Generative Pre-trained Transformer) and StarCoder presents a promising avenue for addressing the shortcomings in automated test case generation (al., 2022) (S. K. Lahiri et al., 2022). These sophisticated language models possess the capacity to understand and generate human-like text, potentially enabling them to comprehend software specifications and generate relevant test cases autonomously.

However, the precise extent to which these LLMs can contribute to the effectiveness, coverage, and efficiency of generated test cases remains an open question (E. Arteca, 2022). Understanding the nuances of how these models operate in the context of software testing is crucial for unlocking their full potential. This research aims to delve into this territory, examining the challenges associated with existing manual and automated test case generation methods and to investigate how LLMs can be leveraged to overcome these challenges, ultimately leading to enhanced testing outcomes in terms of effectiveness, coverage, and efficiency.

Therefore, the problem at hand is to ascertain the feasibility and practicality of integrating large language models such as GPT (al., 2020)or StarCoder (Anon., n.d.) ("Starcoder: A state-of-the-art LLM for code" , Nov. 2023) into the test case generation process and to evaluate their impact on the quality, comprehensiveness, and resource efficiency of generated test cases. Addressing this problem is fundamental for advancing the state-of-the-art in automated testing and ensuring the robustness of software applications in the face of evolving complexities.

# 3. Research Questions (If any)

The following research questions are suggested for each of the research objectives as highlighted as follows.

* How can Large Language Models (LLMs) improve automated test case generation?
* What challenges exist in integrating LLMs into the test case generation process?
* How can the effectiveness of LLM-based test case generation be evaluated and measured?

# 4. Aim and Objectives

The study begins by examining the background of software testing and the challenges associated with manual test case generation. It then delves into the existing methodologies and tools, identifying their limitations and paving the way for the integration of LLMs in this domain. The research questions guiding this investigation revolve around understanding how effectively GPT 3.5, StarCoder etc can contribute to generating robust test cases, addressing the nuanced requirements of diverse software applications.

This research investigates the feasibility and effectiveness of using LLMs for automated test case generation.

The objectives are as follows:

* To explore existing techniques and methodologies for ATCG.
* To investigate the capabilities of LLMs in understanding software specifications and generating test cases.
* To develop an LLM-based test case generation application.
* To assess the efficacy and efficiency of LLM-based test cases in comparison to conventional methodologies.
* To identify limitations and challenges associated with using LLMs for ATCG.
* To propose guidelines and best practices for integrating LLMs into the ATCG process.

This research aims to assess the overall effectiveness of LLMs in test case generation, aiming for comprehensive coverage of software functionalities. Specific objectives include measuring the quality of test cases produced, evaluating the coverage of various code paths, and assessing the efficiency in terms of resource utilization and execution time.

# 5. Significance of the Study

This study holds significance in the field of software engineering by potentially revolutionizing the way test cases are generated. If successful, it could lead to more efficient and effective testing processes, resulting in higher software quality and reduced development costs. Additionally, the findings of this research can provide valuable insights into the capabilities and limitations of LLMs in the context of software testing for both developers and QA teams, contributing to the broader understanding of AI in software engineering.

# 6. Scope of the Study

This study will focus on the application of LLMs, particularly GPT-3.5, for automated test case generation in software engineering. The scope includes exploring various techniques and methodologies for integrating LLMs into the ATCG process, evaluating the effectiveness of LLM-based test cases, and identifying challenges and limitations. The study will primarily involve experimental research, including case studies and empirical evaluations.

# 7. Research Methodology

The research methodology involves the selection, fine-tuning, and integration of GPT and StarCoder models into a test case generation framework. A variety of software applications and datasets will be used to experimentally validate the effectiveness, coverage, and efficiency of the generated test cases. Evaluation metrics will be established to measure the quality of test cases, and comparisons will be made with existing automated test case generation techniques.

The research methodology will involve the following steps:

* **Literature Review:** Conduct a comprehensive review of existing literature on ATCG, LLMs, and related topics.
* **Experimental Design:** Design experiments to evaluate the effectiveness and efficiency of LLM-based test case generation compared to traditional methods.
* **Data Collection:** Collect software artefacts, including specifications, code snippets, and existing test cases, for experimentation.
  1. **Data set:** The dataset to be used is: [Link](https://github.com/ServiceNowDevProgram/code-snippets/tree/main/Script%20Includes), [Link](https://huggingface.co/datasets/bigcode/the-stack-dedup)
  2. **Data pre-processing:** For data pre-processing steps needed for code generation tasks using Large Language Models (LLMs) like GPT or StarCoder, consider the following:
     + **Tokenization**: Convert the raw source code into a sequence of tokens recognizable by the language model. This involves breaking down the code into individual tokens such as keywords, identifiers, operators, and literals.
     + **Special Token Handling**: Handle special tokens or symbols that may not be directly recognized by the language model. For example, comments, string literals, and special characters may need special pre-processing to ensure they are appropriately handled during code generation.
     + **Code Formatting**: Normalize the formatting of the code to ensure consistency and readability. This may involve standardizing indentation, line breaks, spacing, and other stylistic elements to align with the conventions accepted by the language model.
     + **Language-Specific Pre-processing**: Apply language-specific pre-processing steps based on the programming language being used. Different languages may require specific handling for features such as syntax differences, language constructs, and code idioms.
     + **Handling Dependencies and Imports**: Address dependencies and import statements within the code. This may involve resolving external dependencies, managing module imports, and ensuring that the necessary libraries or packages are accessible during code generation.
     + **Error Handling and Exception Management**: Handle error conditions and exceptions that may arise during code generation. This includes pre-processing to manage error messages, exception handling constructs, and error-prone code patterns to ensure robustness in generated code.
     + **Removing Noise**: Remove irrelevant or noisy elements from the code that may hinder the code generation process. This includes removing comments, debugging statements, and other non-essential code artefacts that do not contribute to the intended functionality.
     + **Variable Renaming**: Normalize variable names and identifiers to ensure consistency and clarity. This may involve renaming variables, functions, and other identifiers to improve the readability and maintainability of the generated code.
     + **Dataset Cleaning and Augmentation**: Clean and augment the dataset used for training the language model to improve its effectiveness in code generation tasks. This may involve removing duplicate or irrelevant samples, balancing the dataset, and augmenting with additional training data to enhance model performance.
     + **Code Annotation**: Optionally, annotate the code with additional metadata or labels to provide context or guidance to the language model during code generation. This may include adding annotations for function signatures, variable types, or high-level descriptions of code functionality.
     + **Normalization and Standardization**: Normalize and standardize the input data to ensure consistency and compatibility with the language model's input requirements. This may involve pre-processing steps such as lowercasing, stemming, or lemmatization to reduce lexical variations and improve model generalization.
     + **Length Limitation**: Handle input length limitation constraints imposed by the language model by truncating or splitting long sequences into smaller segments that fit within the model's input size limitations.
* **Implementation:** Implement algorithms and methodologies for integrating LLMs into the ATCG process. The plan is to evaluate OpenAI GPT 3.5, and StarCoder LLMs.

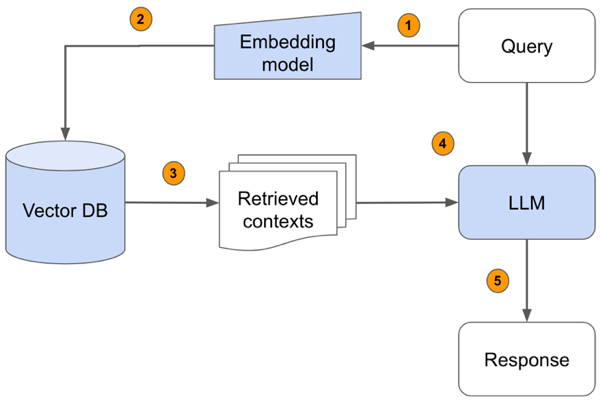


Figure 1.1: Basic Architecture to Integrate LLM with ATCG framework.

* **Evaluation:** Evaluate the generated test cases using metrics such as code coverage, fault detection rate, and execution time. Evaluation metrics for Large Language Models (LLMs) can vary depending on the distinct task and application. However, here are five commonly used evaluation metrics:
  + **Perplexity**: Perplexity assesses the efficacy of a language model in predicting a given text sample. Lower perplexity scores signify superior performance. This metric is widely employed to gauge the language model's proficiency in producing coherent and natural language.
  + **BLEU Score**: The BLEU (Bilingual Evaluation Understudy) score assesses the quality of machine-translated text by juxtaposing it with one or more reference translations. This metric gauges the resemblance between the generated text and the reference text, focusing on n-gram precision and brevity penalty.
  + **ROUGE Score**: ROUGE (Recall-Oriented Understudy for Gisting Evaluation) will be used for assessing the quality of text summarization to quantify the degree of overlap between the generated summary and the reference summary, considering n-gram recall, precision, and F1-score.
  + **Accuracy**: Accuracy quantifies the ratio of accurately generated outputs relative to the total number of samples. This metric is frequently applied in classification tasks like sentiment analysis or question answering to assess the language model's proficiency in making precise predictions.
  + **Human Evaluation**: Human evaluation entails the assessment of the generated text or outputs through human judgment. This encompasses various tasks such as evaluating fluency, assessing relevance, judging grammaticality, and providing an overall quality rating by human annotators.
* **Analysis:** Analyse the results to draw conclusions and identify insights regarding the feasibility and effectiveness of LLMs for ATCG.

# 8. Requirements Resources

The research will require access to LLMs, software artefacts for experimentation, computing resources for implementation and experimentation, and relevant software testing tools for evaluation.

# 8.1 Hardware Requirements

Large language models, especially those like GPT variants, require substantial computational resources for training and inference. These resources typically include:

* High-performance CPUs or GPUs: Modern deep learning frameworks like TensorFlow or PyTorch can efficiently utilize GPUs for accelerated training.
* Specialized hardware accelerators: Some organizations use TPUs (Tensor Processing Units) or other specialized hardware for even faster training.
* Large amounts of RAM: Language model training often requires significant memory resources to store model parameters and intermediate computations.

# 8.2 Software Requirements

The software stack for training large language models includes:

* Deep learning frameworks: TensorFlow, PyTorch, or similar frameworks are commonly used for implementing and training models.
* Efficient data processing libraries: Tools like Apache Spark, Dask, or TensorFlow Data Pipeline can help manage large datasets efficiently.
* Version control systems: Git or similar systems are crucial for collaboration and managing changes to the model codebase.
* Containerization tools: Docker or Singularity are often used to create reproducible environments for model training.

# 8.3 Dataset Specifications

Large language models require vast amounts of text data for training. For this research, domain-specific data will be used which will contain ServiceNow-specific code taken from ServiceNow's Code Snippets community repository, designed for the ServiceNow Dev Program, and managed by the Developer Program and the sndevs community.

An additional dataset, known as "The Stack," sourced from the Hugging Face library, is also under consideration. This dataset encompasses a vast expanse of licensed source code files spanning 358 programming languages, totalling over 6TB.

# 8.4 Language Model Training Details

Training a large language model involves several steps and considerations:

* Pre-processing: Text data must be tokenized, normalized, and possibly filtered before training.
* Model architecture: Choose an appropriate architecture (e.g., Transformer-based) and size for the language model.
* Hyperparameters: Tuning hyperparameters such as learning rate, batch size, and dropout rate is crucial for effective training.
* Training procedure: Training often involves using techniques like gradient accumulation, learning rate schedules, and early stopping to improve convergence and stability.
* Evaluation: Models should be evaluated on validation datasets to monitor performance and prevent overfitting.
* Fine-tuning: After initial training, models may be fine-tuned on specific tasks or domains to improve performance.

These resources and considerations are essential for successfully training and deploying large language models.

# 9. Research Plan

Here's a breakdown of a research plan spanning 22 weeks from January 17th, 2024, when research topics were approved:

* **Week 1-2: Research Proposal Development**
  + **Week 1:** Define research objectives, scope, and methodology.
  + **Week 2:** Draft the research proposal, including the literature review and theoretical framework.
* **Week 3-4: Proposal Review and Revision**
* **Week 3:** Submit the proposal for review by advisors or peers.
* **Week 4:** Revise the proposal based on feedback and finalize it for approval.
* **Week 5-8: Data Collection and Analysis**
* **Week 5-7:** Collect data through surveys, experiments, or other methods.
* **Week 8:** Begin data analysis using appropriate statistical or qualitative techniques.
* **Week 9-12: Data Analysis and Interpretation**
* **Week 9-10:** Continue data analysis and refine interpretations.
* **Week 11-12:** Draft the results section of the research report.
* **Week 13-16: Writing and Revision**
* **Week 13-14: Write the discussion and conclusion sections.**
* **Week 15-16: Revise the entire research report based on feedback.**
* **Week 17-18: Presentation Preparation**
* **Week 17:** Prepare visual aids and presentation materials.
* **Week 18:** Practice the presentation for conferences or seminars.
* **Week 19-22: Presentation and Finalization**
* **Week 19:** Present the research findings at conferences or seminars.
* **Week 20-22:** Finalize the research report based on feedback and submit it for publication or assessment.

Here is the Gantt chart for the project timeline:



Figure 2. Project Plan

This plan provides a structured timeline for completing each phase of the research process within 22 weeks, ensuring adequate time for proposal development, data collection, analysis, writing, presentation, and finalization. Adjustments can be made as needed based on the specific requirements and progress of the research project.

# References

K, B., 2000. *Extreme Programming Explained: Embrace Change..* s.l.:Addison-Wesley.

Siddiqui, S., 2021. *Learning Test-Driven Development.* s.l.:O’Reilly.

J. Shore and S. Warden, 2021. *The Art of Agile Development, Sebastopol.* CA, USA: O’Reilly.

E. Daka and G. Fraser, "., 2014. *A survey on unit testing practices and problems.* s.l.:s.n.

M. M. Almasi, H. H. G. F. A. A. a. J. B., n.d. *An industrial evaluation of unit test generation: Finding real faults in a financial application.* s.l.:s.n.

Zalewski, M., 2023. *American fuzzy lop.* s.l.:s.n.

G. Grano, S. S. H. C. G. a. R. O., 2018. *An empirical investigation on the readability of manual and generated test cases.* s.l.:s.n.

A. Panichella, S. P. G. F. A. A. S. a. V. J. H., 2020. *Revisiting test smells in automatically generated tests: Limitations pitfalls and opportunities.* s.l.:s.n.

E. Daka, J. C. G. F. J. D. a. W. W., 2015. *Modeling readability to improve unit tests.* s.l.:s.n.

al., B. C. e., 2022. *CodeT: Code Generation with Generated Tests.* s.l.:s.n.

S. K. Lahiri et al., 2022. *Interactive code generation via test-driven user-intent formalization.* s.l.:s.n.

E. Arteca, S. H. M. P. a. F. T., 2022. *"Nessie: Automatically testing JavaScript APIs with asynchronous callbacks", Proc..* s.l.:44th IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE), pp. 1494-1505,..

al., T. B. B. e., 2020. *"Language models are few-shot learners".* s.l.:Available: https://arxiv.org/abs/2005.14165..

Anon., n.d. *"OpenAI LLMs: Deprecations", Nov. 2023, [online] Available: https://platform.openai.com/docs/deprecations..* s.l.:s.n.

"Starcoder: A state-of-the-art LLM for code" , Nov. 2023. s.l.:[online] Available: https://huggingface.co/blog/starcoder..