**TEST CASE GENERATION BY FINE-TUNING LARGE LANGUAGE MODELS**

A PROJECT REPORT

*Submitted By:*

**PALLAVI PANDEY (RA2011026010348)**

**ANUNAY SINGH (RA2011026010313)**

*Under the Guidance of*

**Dr. Saranya A**

Assistant Professor, Department of Computational Intelligence

*in partial fulfillment of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

with specialization in Artificial Intelligence and Machine Learning

of

**DEPARTMENT OF COMPUTATIONAL INTELLIGENCE**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

****

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR– 603 203**

**NOVEMBER 2023**

****

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**BONAFIDE CERTIFICATE**

Certified that the project report titled “TEST CASE **GENERATION BY FINE TUNING LLM’S**” is the bonafide work of **PALLAVI PANDEY (RA2011026010348) and ANUNAY SINGH (RA2011026010313)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge, the work reported here does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

### **Dr. Saranya A Ms. Sasi Rekha Sankar**

**PROJECT GUIDE PANEL HEAD**

Assistant Professor Assistant Professor

Department of Computational Intelligence Department of Computational Intelligence

**Dr. R. Annie Uthra**

**HEAD OF DEPARTMENT**

Department of Computational Intelligence



**Department of Computational Intelligences**

# **SRM Institute of Science and Technology Own Work Declaration Form**

**Degree/Course:** B.Tech in Computer Science and Engineering

S**tudent Names:** Pallavi Pandey, Anunay Singh

**Registration Number:** RA2011026010348, RA2011026010313

**Title of Work:** Test Case Generation by fine-tuning LLM’s

I/We hereby certify that this assessment compiles with the University’s Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is our own except where indicated and that we have met the following conditions:

* Clearly references/listed all sources as appropriate
* Referenced and put in inverted commas all quoted text (from books, web,etc.)
* Given the sources of all pictures, data etc that are not my own.
* Not made any use of the report(s) or essay(s) of any other student(s)either past or present
* Acknowledged in appropriate places any help that I have received from others (e.g fellow students, technicians, statisticians, external sources)
* Compiled with any other plagiarism criteria specified in the Course hand book / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

|  |
| --- |
| **DECLARATION:** |
| I am aware of and understand the University’s policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.  **Student 1 Signature:**  **Student 2 Signature:**  **Date:** |
| If you are working in a group, please write your registration numbers and sign with the date for every student in your group. |

**ACKNOWLEDGEMENT**

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T. V. Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr.R. Annie Uthra,** Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinator, **Dr. C Karthik,** Associate Professor**,** Panel Head, **Mrs. Sasi Rekha Shankar**, Assistant Professor and Panel Members, **Dr. M. Maheswari,** Assistant Professor, **Dr. R A Karthika,** Assistant Professor and **Dr. R. Udendhran,** Assistant Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. A. Maheshwari** Assistant Professor **and Dr. A. Revathi**, Assistant Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. A. Saranya**, Assistant Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computing Technologies Department, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

**Pallavi Pandey**

**(RA2011026010348)**

**Anunay Singh**

**(RA2011026010313)**

**ABSTRACT**

In the realm of software development, automating the creation of test cases emerges as a transformative solution, effectively addressing the challenges associated with manual and time-consuming processes, particularly within large software teams. This project report introduces a pioneering approach to Test Case Generation Systems (TCGS), leveraging the power of fine-tuned Large Language Models (LLMs) on a proprietary dataset. This innovative paradigm shift has the potential to reshape traditional software testing practices, optimizing resource allocation and significantly enhancing the quality and competitiveness of software products across diverse industries. In recent years, the field of software testing has undergone a remarkable transformation, propelled by advancements in artificial intelligence and machine learning. This project builds upon these advancements, demonstrating how LLMs, such as GPT-3 and its successors, can be utilized to automatically generate comprehensive and context-aware test cases. By training LLMs on a proprietary dataset encompassing various software requirements and coding patterns, we equip them with a profound understanding of the software domain, enabling them to produce high-quality test cases that cover a wide spectrum of scenarios and edge cases. This project report provides a comprehensive exploration of the methodology, outcomes, and potential implications of employing LLMs for TCGS. It underscores the significance of harnessing cutting-edge technologies to transform the software testing landscape and sets the stage for a future in which manual test case creation becomes the exception rather than the norm.

**TABLE OF CONTENTS**

**ACKNOWLEDGEMENTS 3**

**ABSTRACT 4**

**LIST OF FIGURES 7**

**ABBREVIATIONS 8**

**CHAPTER 1. 9**

**INTRODUCTION 9**

1.1 General 9

1.2 Motivation 10

1.3 Purpose of the Project 11

1.4 Objective 12

**CHAPTER 2 14**

**LITERATURE STUDY 14**

2.1 Study of Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach

14

2.2Analyzing unit test case Generation 15

2.3 Adapting LLM-Based Bug Reproduction Techniques for ETCGS………………........16

2.4Exploring the Role of LLMs in Mobile Application Test Script Generation………. …17

[2.5](https://docs.google.com/document/d/1eBcG5BaH-zKUiRE7tFGilWvdQI5FlN1bnvvFNcOFGlQ/edit?pli=1#heading=h.4i7ojhp) Transformers: A Gateway to State-of-the-Art Natural Language Processing………… …18

**CHAPTER 3 19**

**Problem Statement and Proposed Solution 19**

3.1 Problem Statement 19

3.2 Proposed Model 21

3.3Architecture Diagram……………………………………………………………………24

**CHAPTER 4 25**

**METHODOLOGY****25**

4.1 Data Collection 27

4.2 Data Splitting 27

4.3 Model Development 28

4.4 Model Evaluation 29

4.5 Model Comparison 29

4.6 Visualizations 30

4.7 Output Prediction 30

4.8 Documentation 30

**CHAPTER 5 31**

**TECHNICAL REQUIREMENTS 31**

5.1 Hardware Requirements: 31

5.2 Software Requirements 31

**CHAPTER 6 32**

**SYSTEM DESIGN 32**

**CHAPTER 7 35**

**CODING AND TESTING 35**

**CHAPTER 8 48**

**RESULT AND ANALYSIS 48**

**CHAPTER 9 49**

**CONCLUSION AND FUTURE DEVELOPMENT 49**

**REFERENCES 50**

**LIST OF FIGURES**

Fig. 3.1 Architecture Diagram of the Model

Fig. 4.1 Overview of the Model

Fig. 6.1 UML Diagram of the Model

Fig. 7.1 Output of Leaf Image and its predicted name

**ABBREVIATIONS**

1. TCGS: Test Case Generation System
2. LLM: Large Language Model
3. CI/CD: Continuous Integration/Continuous Delivery
4. NLP: Natural Language Processing
5. KPIs: Key Performance Indicators
6. IoT: Internet of Things
7. AI: Artificial Intelligence
8. RUCM: Rational Unified Process Use Case Model
9. UML: Unified Modeling Language
10. OCL: Object Constraint Language

These abbreviations are used throughout the report to refer to key concepts and terms, enhancing the readability and conciseness of the document.

**CHAPTER 1**

**INTRODUCTION**

## **1.1 GENERAL**

In the ever-evolving landscape of software development, the quest for efficiency, quality, and competitiveness has led to remarkable innovations. One such innovation, the automation of test case generation, has emerged as a transformative solution, alleviating the challenges associated with manual testing processes, particularly in the context of large software development teams. This project represents a pioneering endeavour to revolutionize Test Case Generation Systems (TCGS) by harnessing the capabilities of state-of-the-art Large Language Models (LLMs) on a proprietary dataset.

Software testing is an indispensable phase in the software development life cycle, serving as a critical gatekeeper to ensure the reliability, functionality, and security of software products. However, the manual creation of test cases is labour-intensive, time-consuming, and prone to human errors. As software systems grow in complexity, the demand for comprehensive and context-aware testing becomes increasingly pressing. In response to these challenges, the integration of LLMs into TCGS offers a promising path forward.

This project explores how the power of LLMs, such as GPT-3 and its successors, can be harnessed to automatically generate test cases. By training these models on a carefully curated dataset that encompasses a wide range of software requirements and coding patterns, they acquire a deep understanding of the software domain. As a result, they become capable of producing high-quality test cases that not only cover common scenarios but also delve into nuanced edge cases, which are often overlooked in manual testing.

The implications of automating test case generation are profound. It promises to expedite the testing process, reduce manual effort, and optimize resource allocation. This, in turn, allows software development teams to channel their expertise into more strategic tasks, ultimately leading to improved efficiency and faster time-to-market. Additionally, the system's ability to generate test cases for a diverse array of software applications enhances the overall quality of software products, mitigating the risks associated with unnoticed defects and vulnerabilities.

Beyond these immediate benefits, the integration of LLMs into TCGS opens the door to a transformative shift in software development practices across industries. The cost-effectiveness and scalability of automated test case generation make it a compelling option for organizations of all sizes. By embracing this innovative approach, software companies can streamline their development processes, reduce costs, and deliver more reliable and competitive products, thus meeting the ever-increasing demands of the digital age.

This project aims to provide a comprehensive exploration of the methodology, results, and potential implications of employing LLMs for TCGS. It underscores the importance of harnessing cutting-edge technologies to redefine the software testing landscape, setting the stage for a future where manual test case creation becomes the exception rather than the rule.

## **1.2 MOTIVATION**

Our decision to embark on the Test Case Generation Systems (TCGS) project is driven by several practical considerations and opportunities that promise to benefit software development. At its core, this project aims to streamline the process of creating test cases, which are essential for ensuring that software functions correctly. The following key motivations underline our commitment to this project.

First and foremost, we are motivated by the prospect of boosting efficiency in software development. Traditionally, crafting test cases manually can be a time-consuming endeavor, potentially causing delays in project timelines. By leveraging intelligent computer programs, we can significantly expedite the test case generation process. This, in turn, allows software developers and testers to direct their efforts towards more crucial aspects of their work, ultimately speeding up project delivery.

Another compelling factor is the potential for improving the quality of software through better testing. Human testers, while diligent, may inadvertently overlook certain aspects of software functionality, leading to hidden problems that may emerge later. By training smart computer programs, we can generate test cases that comprehensively examine the software, even delving into complex and less obvious scenarios. This approach enhances the overall quality of software by uncovering defects and vulnerabilities that might otherwise go undetected.

Cost-effectiveness and resource optimization represent an additional driving force for this project. The manual creation of test cases often demands a significant allocation of time, effort, and human resources. By introducing computer assistance into the process, we anticipate cost savings for project stakeholders. Fewer human resources are required for manual testing, and the risk of undetected issues is minimized, reducing the need for costly post-release bug fixes.

Lastly, the project's motivation lies in its potential to empower software development teams, enabling them to remain competitive in the fast-paced software industry. In a world where speed to market is a significant advantage, using computer-generated test cases can expedite the development process. This, in turn, enhances the competitive edge of organizations, as they can deliver products more swiftly, responding to the ever-increasing demands of the digital age.

In conclusion, the TCGS project is motivated by the pursuit of enhanced efficiency, superior testing quality, resource savings, and competitiveness within the software development landscape. It revolves around the notion of harnessing intelligent computer programs to simplify and expedite the test case creation process, thereby ensuring that software is more reliable and cost-effective, and that development teams can deliver products in a timely manner.

## **1.3 PURPOSE OF THE PROJECT**

The purpose of this project is to transform and enhance the conventional process of creating test cases in software development by leveraging advanced Large Language Models (LLMs) and intelligent algorithms. The primary objectives of this project are as follows:

Efficiency Improvement: The project seeks to significantly improve the efficiency of software development by automating the generation of test cases. By doing so, we aim to expedite the testing process, reduce time-to-market, and enable software development teams to focus on core tasks.

**Quality Assurance**: One of the central purposes of this project is to elevate the quality and reliability of software products. By using LLMs to create test cases, we can ensure a more comprehensive examination of software functionality, uncovering defects and vulnerabilities that manual testing might miss.

**Resource Optimization:** A key purpose is to optimize resource allocation in software development. Through automation, the project aims to reduce the human resources and effort required for test case creation, resulting in cost savings and efficient resource management.

**Competitive Edge:** The project strives to position software development teams and organizations for greater competitiveness. The ability to accelerate the testing phase and deliver software products more quickly can provide a significant competitive advantage in a rapidly evolving industry.

**Innovation in Testing:** This project is motivated by the aspiration to innovate and explore new testing methodologies. By incorporating intelligent computer programs, it opens up new possibilities for how software can be rigorously tested and validated.

**Cost Reduction:** The project's purpose includes cost reduction, as it can lead to savings in both time and expenses. Fewer human testers are needed, and the likelihood of post-release bug fixes is minimized, reducing the overall cost of software development.

In summary, the primary purpose of this project is to revolutionize the way test cases are generated in software development. By integrating advanced technology and intelligent algorithms, we aim to make testing more efficient, cost-effective, and comprehensive. Ultimately, the project's purpose is to enhance software quality, reduce resource overhead, and enable organizations to remain competitive in a dynamic and demanding software industry.

## **1.4 OBJECTIVE**

The objective of this project is to develop and implement an automated Test Case Generation System (TCGS) that harnesses the capabilities of Large Language Models (LLMs) and intelligent algorithms to create high-quality, comprehensive test cases for software applications. The specific goals of this project include:

**Automation of Test Case Generation:** Develop a system that can automatically generate test cases for software applications, reducing the manual effort and time required for this critical phase of software development.

**Enhanced Test Coverage:** Ensure that the automated system produces test cases that provide extensive coverage of software functionality, including both common and edge cases, to identify defects and vulnerabilities effectively.

**Resource Optimization:** Optimize resource allocation by reducing the human effort and cost associated with manual test case creation. The project aims to demonstrate the cost-effectiveness of automated test case generation.

**Improved Software Quality:** Enhance the quality and reliability of software products by using the automated system to uncover hidden defects and vulnerabilities, resulting in a higher level of assurance and overall product quality.

**Time-to-Market Acceleration:** Expedite the software development process by automating test case generation, enabling faster time-to-market for software products, which can provide a competitive edge.

**Innovation in Software Testing:** Innovate in the field of software testing by integrating advanced technologies, such as Large Language Models, into the test case generation process. Explore new methodologies and approaches to software testing.

The overarching objective is to create a sophisticated automated system that not only accelerates the testing phase but also ensures comprehensive coverage and high-quality testing. This project aims to make software testing more efficient, cost-effective, and innovative, ultimately benefiting software development teams and organizations in delivering reliable and competitive software products.

# **CHAPTER 2**

# **LITERATURE STUDY**

**2.1 Study of Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach [2]**

The research paper titled "Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach" by Chunhui Wang, Fabrizio Pastore, Arda Goknil, and Lionel C. Briand [2] presents a groundbreaking approach to automating the generation of acceptance test cases from natural language requirements, specifically focusing on use case specifications. This paper offers valuable insights and methodologies that can be adapted for our project, which aims to transform Test Case Generation Systems (TCGS) by utilizing Large Language Models (LLMs) and intelligent algorithms to create test cases efficiently and comprehensively.

The authors [2] of the paper recognize the challenges associated with manual test case generation, particularly in the context of acceptance testing, where the goal is to validate software systems against functional requirements. These requirements are often provided in natural language, such as use case specifications, and can be extensive and complex. The paper acknowledges the expense and error-prone nature of manual test case creation in such scenarios.

In response to these challenges, the paper introduces the Use Case Modeling for System-level, Acceptance Tests Generation (UMTG) approach. UMTG leverages natural language processing (NLP) techniques to automate the generation of acceptance test cases from use case specifications and domain models. The primary goal of UMTG is to reduce manual effort while ensuring comprehensive requirements coverage. Unlike existing approaches, UMTG does not impose strict limitations on the expressiveness of use case specifications, making it a flexible and adaptable solution [2]

UMTG's key innovations include the automatic identification of test scenarios and the generation of formal constraints that capture conditions triggering these scenarios. This, in turn, enables the automated generation of test data. In two industrial case studies, UMTG demonstrated its effectiveness by accurately translating a significant portion of use case specification steps into formal constraints for test data generation. Furthermore, it successfully generated test cases that not only covered all test scenarios manually implemented by experts but also identified critical scenarios that were previously overlooked.

The lessons and methodologies presented in this research paper hold substantial value for our TCGS project, which shares similar goals and challenges. Specifically, we can draw upon the following insights:

1. **Leveraging NLP Techniques:** The success of UMTG in automating test case generation highlights the potential of NLP techniques in handling natural language requirements. We can adapt NLP tools and methodologies to extract essential information from requirements documents and use it for automated test case generation.
2. **Flexibility in Use Case Specifications:** UMTG's approach of not imposing strong restrictions on the expressiveness of use case specifications aligns with our project's aim to accommodate various types of input documents. We can design our system to work with diverse natural language sources to ensure versatility.
3. **Comprehensive Test Data Generation:** UMTG's ability to generate formal constraints for test data is a critical component in ensuring thorough test coverage. Our project can incorporate similar mechanisms to identify and capture test scenarios and their corresponding conditions, enhancing the quality of automated test case generation.
4. **Validation Through Case Studies:** The industrial case studies conducted by UMTG underscore the practical applicability and effectiveness of the approach. Similarly, our project can benefit from real-world testing to validate and fine-tune the automated test case generation system.

In conclusion, the paper on UMTG provides a valuable reference for our TCGS project. It demonstrates the feasibility and advantages of using NLP-based automation to generate test cases from natural language requirements. By adapting and building upon the methodologies and insights presented in this research, our project can make substantial progress in achieving more efficient, comprehensive, and cost-effective automated test case generation for software development.

**2.2 Analyzing unit test case generation that respects temporal constraints within software classes[3]**

The research paper titled "Call Me Maybe: Using NLP to Automatically Generate Unit Test Cases Respecting Temporal Constraints" by Arianna Blasi, Alessandra Gorla, Michael D. Ernst, and Mauro Pezzè [3] introduces an innovative approach to unit test case generation that respects temporal constraints within software classes. This paper presents valuable insights and methodologies that can be adapted for our project, which aims to revolutionize Test Case Generation Systems (TCGS) by leveraging Large Language Models (LLMs) and intelligent algorithms for efficient and comprehensive test case creation.

The authors of the paper acknowledge the importance of adhering to temporal constraints in software classes, as violations of these constraints can lead to runtime exceptions that may not necessarily expose software faults. In response to this challenge, the paper introduces "CallMeMaybe," a novel technique that utilizes natural language processing (NLP) to analyze Javadoc comments to identify and understand temporal constraints within software classes. CallMeMaybe then uses this information to guide a test case generator in executing sequences of method calls that respect these temporal constraints.

Key insights and methodologies presented in this research paper can be adapted for our TCGS project in the following ways:

1. **Leveraging Natural Language Processing (NLP):** The success of CallMeMaybe in utilizing NLP to analyze Javadoc comments aligns with our project's objectives. We can adapt NLP techniques to extract and understand relevant information from natural language requirements, code comments, or documentation, thus enhancing our understanding of the software's intended behavior.
2. **Temporal Constraint Identification:** The paper's emphasis on identifying and respecting temporal constraints offers valuable guidance for our project. By incorporating similar capabilities into our TCGS system, we can ensure that generated test cases consider temporal requirements and constraints, reducing the likelihood of unnecessary exceptions.
3. **Enhanced Test Case Precision:** CallMeMaybe's success in achieving a precision of 83% and a recall of 70% when translating temporal constraints into Java expressions underscores the potential for improving the precision of automated test case generation. We can apply similar evaluation methods and enhancements to our project to ensure the quality of generated test cases[3].
4. **Integration with Existing Tools:** The integration of CallMeMaybe with Randoop to identify false alarms and enrich correctly failing test cases demonstrates the importance of compatibility with existing testing tools. Our project can benefit from similar integrations, ensuring a seamless fit into the software development workflow.

In summary, the paper on "Call Me Maybe" provides valuable insights into how NLP can be used to understand and respect temporal constraints during unit test case generation.

**2.3 Adapting LLM-Based Bug Reproduction Techniques for Enhanced Test Case Generation [4]**

The research paper titled "Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction" by Sungmin Kang, Juyeon Yoon, and Shin Yoo [4] explores an innovative approach to automated test case generation using Large Language Models (LLMs) for the purpose of reproducing bugs. This paper offers significant insights and methodologies that can be adapted for our Test Case Generation Systems (TCGS) project, which aims to enhance the efficiency and effectiveness of test case creation in software development.

The paper acknowledges the importance of reproducing bugs as a crucial objective in software testing. Reproducing bugs from bug reports is essential for identifying and resolving software issues effectively. However, existing test generation techniques often fall short of achieving this goal, as they tend to focus on other objectives such as increasing test coverage or generating exploratory inputs.

In response to the challenge of automating test generation from bug reports, the paper presents "LIBRO," a framework that leverages LLMs[4]. LLMs have shown their capabilities in performing code-related tasks, making them a valuable resource for test case generation. LIBRO focuses on post-processing steps to determine when LLMs are effective in generating bug-reproducing test cases and ranks the produced tests according to their validity.

Key insights and methodologies from this research paper can be adapted for our TCGS project in the following ways:

1. **Utilizing Large Language Models:** The success of LIBRO in using LLMs for test case generation aligns with our project's objectives, which involve the use of LLMs to automate test case creation. We can adapt similar techniques and post-processing steps to ensure the effectiveness of LLMs in generating relevant and valid test cases.
2. **Bug Reproduction:** The paper emphasizes the importance of reproducing bugs from bug reports, which is an essential aspect of software testing. Our project can incorporate bug reproduction as a specific objective, ensuring that the generated test cases are designed to replicate the reported issues effectively.
3. **Validity Ranking:** LIBRO's approach to rank the produced tests according to their validity is a valuable concept that can enhance the quality of test case generation. We can adapt similar mechanisms to prioritize and rank the generated test cases based on their relevance and effectiveness.
4. **Enhancing Developer Efficiency:** The paper's results demonstrate LIBRO's potential to significantly enhance developer efficiency by automatically generating tests from bug reports. Our project can aim for a similar outcome, providing developers with an automated and efficient solution for test case generation.

In summary, the paper on LLM-based bug reproduction provides valuable insights into how Large Language Models can be used to automate the generation of bug-reproducing test cases from general bug reports. By adapting and building upon the methodologies presented in this research, our TCGS project can make substantial progress in achieving the efficient and effective automated test case generation, with a specific focus on reproducing reported bugs. This adaptation can significantly enhance the overall quality and reliability of software testing and development processes.

**2.4 Exploring the Role of Large Language Models in Mobile Application Test Script Generation: Insights for Enhanced Test Case Generation [5]**

The research paper titled "LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities" by Shengcheng Yu, Chunrong Fang, Yuchen Ling, Chentian Wu, and Zhenyu Chen [5] explores the application of Large Language Models (LLMs) in the context of mobile application test script generation. This paper presents valuable insights and methodologies that can be adapted for our Test Case Generation Systems (TCGS) project, which aims to enhance the efficiency and effectiveness of test case creation in software development, including mobile applications.

The paper acknowledges the significance of test script generation in the domain of software testing, highlighting its role in automating repetitive test tasks efficiently and reliably. However, it also identifies challenges in existing test script generation approaches, particularly when dealing with diverse devices, platforms, and applications. Differences in screen sizes, input methods, platform behaviors, API inconsistencies, and application architectures can complicate the accurate and comprehensive generation of test scripts.

The primary objective of the paper is to explore how LLMs can be leveraged to address these challenges and serve as a versatile tool for test automation. The investigation encompasses several key aspects:

1. **Adaptability to Diverse Devices and Systems:** The paper explores how well LLMs can adapt to a variety of devices and systems, ensuring that they can accurately capture and generate test scripts. This adaptability is essential for ensuring the robustness of test automation across different mobile platforms.
2. **Cross-Platform Generation Capabilities:** The paper evaluates the LLMs' cross-platform generation capabilities, assessing their ability to handle variations in operating systems and platform-specific behaviors. This is crucial for achieving consistent and reliable test automation across different mobile platforms.
3. **Cross-App Migration:** The research investigates the application of LLMs in cross-app migration, where they generate test scripts for different applications and software environments based on existing scripts. This can significantly streamline the testing process for multiple applications.
4. **User Interface and Interaction Adaptability:** The paper analyzes LLMs' adaptability to various user interfaces, app architectures, and interaction patterns. This adaptability ensures that LLMs can accurately generate test scripts that mimic user interactions and app behaviors, enhancing compatibility and reliability.

The findings of this research contribute to a deeper understanding of LLMs' capabilities in test automation, particularly in the context of mobile applications. By leveraging LLMs, this research aims to enhance software testing practices, ultimately empowering app developers to achieve higher levels of software quality and development efficiency.

**2.5 Transformers: A Gateway to State-of-the-Art Natural Language Processing[8]**

The paper titled "Transformers: State-of-the-Art Natural Language Processing" by Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush [8] , introduces an innovative and comprehensive approach to Natural Language Processing (NLP). This paper provides valuable insights and methodologies that can inform and enhance our understanding of state-of-the-art NLP techniques for various applications.

The paper acknowledges the significant progress made in NLP, attributing it to advancements in model architecture and pretraining. Transformer architectures have played a pivotal role in enabling the development of high-capacity models, while pretraining has made it feasible to leverage this capacity across a wide range of NLP tasks. This has led to the development of the open-source library, "Transformers," with the overarching goal of democratizing these advances within the broader machine learning community [8].

Key elements of the "Transformers" library include:

1. **State-of-the-Art Transformer Architectures:** The library encompasses carefully engineered Transformer architectures that represent the cutting edge in NLP. These architectures are made available through a unified API, making them accessible for researchers, practitioners, and developers.
2. **Pretrained Models:** The library includes a curated collection of pretrained models that are not only created by experts but are also made available for the wider community. This collection empowers researchers and practitioners to utilize state-of-the-art models without the need for extensive training.
3. **Extensibility:** "Transformers" is designed to be extensible, allowing researchers to build upon existing models and adapt them for specific applications. This adaptability is a key aspect of enabling innovation and the development of domain-specific NLP solutions.
4. **User-Friendly Interface:** The library's simplicity and unified API aim to make it approachable and usable for practitioners. It is built with the intention of enabling a wide range of developers to harness the power of advanced NLP models without a steep learning curve.
5. **Industrial Deployment:** The library emphasizes speed and robustness for industrial deployments, acknowledging the importance of practical, real-world applications of NLP models.

In summary, this paper introduces the "Transformers" library as a comprehensive and accessible resource for leveraging state-of-the-art NLP techniques. It empowers researchers, practitioners, and developers to harness the capabilities of advanced NLP models for various applications. The extensibility, user-friendly interface, and focus on industrial deployment make "Transformers" a valuable tool in advancing the field of NLP and expanding its practical applications.

A green chart with white text

Description automatically generated

Fig 2.1 Literature Survey

A green chart with white text

Description automatically generated

Fig. 2.2 Literature Survey

# **CHAPTER 3**

# **PROBLEM STATEMENT AND PROPOSED SOLUTION**

**3.1 PROBLEM STATEMENT**

The project focuses on Test Case Generation Systems (TCGS) and aims to address several challenges faced in the software development and testing process. The core problem revolves around the limitations of traditional manual test case generation methods, which are often time-consuming, error-prone, and less effective in uncovering software defects and vulnerabilities. These limitations pose significant challenges in terms of resource allocation, quality assurance, and time-to-market for software products.

**Challenges Faced:**

1. **Manual Test Case Generation:** Traditional manual test case generation methods require significant human effort, which can be resource-intensive and slow down the software development process. This approach may result in incomplete test coverage and the possibility of overlooking critical test scenarios.
2. **Efficiency and Time-to-Market:** The time required for manual test case creation can lead to delays in software release cycles. This delay can be particularly problematic in fast-paced industries where timely product delivery is essential to remain competitive.
3. **Quality Assurance:** Manual test case generation may miss potential software defects and vulnerabilities, leading to post-release bug fixes and potentially damaging the reputation of the software. Ensuring the quality and reliability of software products is a key challenge.
4. **Resource Optimization:** Allocating human resources to manual test case generation can be costly. This challenge includes not only the financial cost but also the need for skilled testers and the allocation of their time to manual testing tasks.
5. **Test Coverage:** Achieving comprehensive test coverage is challenging, especially when test cases need to account for various use cases, boundary conditions, and error scenarios. Manual methods might not cover all aspects of the software's functionality.

**3.2 PROPOSED SOLUTION**

To address the challenges mentioned above, the project proposes the development of an advanced Test Case Generation System (TCGS) that leverages the capabilities of Large Language Models (LLMs) and intelligent algorithms. The following are the proposed solutions to mitigate these challenges:

1. **Automation of Test Case Generation:** The project aims to automate the process of test case generation using LLMs and intelligent algorithms. This approach significantly reduces the manual effort required for test case creation, expediting the testing process.
2. **Enhanced Test Coverage:** By using LLMs, the project aims to generate test cases that provide comprehensive coverage of software functionality. These automated test cases can explore various scenarios, boundary conditions, and error cases, ensuring a more thorough examination of the software.
3. **Resource Optimization:** Automation reduces the human resources and effort required for test case creation. This results in cost savings, efficient resource management, and the ability to allocate skilled testers to more complex testing tasks.
4. **Improved Software Quality:** The project seeks to enhance the quality and reliability of software products by using LLMs to uncover hidden defects and vulnerabilities that manual testing might miss. This approach leads to more robust software.
5. **Time-to-Market Acceleration:** Automation allows for faster test case generation, leading to expedited software development and shorter time-to-market. This advantage provides a competitive edge in rapidly evolving industries.
6. **Innovation in Testing:** The project encourages the exploration of innovative testing methodologies. By incorporating LLMs and intelligent algorithms, it opens up new possibilities for how software can be rigorously tested and validated.
7. **Cost Reduction:** By automating test case generation, the project reduces both time and expenses. Fewer human testers are needed, and the likelihood of post-release bug fixes is minimized, ultimately reducing the overall cost of software development.

In conclusion, the project's proposed solution involves the development of an advanced TCGS system that leverages LLMs and intelligent algorithms to automate test case generation, enhance test coverage, optimize resource allocation, improve software quality, accelerate time-to-market, drive innovation in testing, and reduce costs. This comprehensive approach aims to overcome the challenges associated with manual test case generation, ultimately resulting in more efficient and effective software development and testing processes.

**3.3 ARCHITECTURE DIAGRAM**

A diagram of a test case

Description automatically generated

Fig. 3.1 Architecture Diagram

# 

# **CHAPTER 4**

# **METHODOLOGY**

The methodology for the TCGS consists of several steps, from requirement analysis to fine-tuning, which can be elaborated as follows:

* **Project Initiation:** At the outset, the project begins with the definition of clear objectives, scope, and goals. An essential component of project initiation is the assembly of a dedicated project team with expertise in software testing, natural language processing, and machine learning. This team will be responsible for guiding the project through its various stages.
* **Requirements Analysis:** In this phase, the specific testing needs and challenges within your organization are identified and thoroughly analyzed. It involves a detailed examination of the types of software to be tested, the critical functionality that needs to be covered, and the overarching objectives of the TCGS.
* **Data Collection and Preparation:** Effective data management is a foundational step. Relevant data, including software documentation, requirements, user stories, and existing test cases, is collected and systematically organized. To make this data suitable for use by Large Language Models (LLMs), preprocessing and cleaning are essential steps.
* **LLM Selection and Fine-Tuning:** The selection of an appropriate LLM, such as GPT-4 or BERT, is a crucial decision. The selected LLM is fine-tuned on a proprietary dataset that includes your organization's software-related text data. This fine-tuning process ensures that the LLM can effectively understand and generate content specific to your software projects.
* **Test Scenario Identification:** Test scenarios are the building blocks of effective test case generation. The project involves the development of algorithms and processes that can identify potential test scenarios from software documentation and requirements. LLMs play a significant role in extracting and understanding test scenarios presented in natural language.
* **Test Case Generation:** This phase involves the design of algorithms and workflows for the automated generation of test cases based on the identified test scenarios. LLMs are employed to automatically generate test cases in natural language. The aim is to create a reliable and efficient system for test case creation.
* **Test Case Evaluation:** Generated test cases are evaluated for quality and coverage. A system is developed to ensure that test cases adhere to predefined criteria and effectively cover essential software functionality. This step ensures the generated test cases are comprehensive and reliable.
* **Test Execution and Monitoring:** An execution framework is implemented to run the generated test cases on the target software. This phase also involves active monitoring of test results, with the recording of any issues or failures detected during the testing process.
* **Feedback Loop:** A feedback loop is established to continuously improve the LLM-based test case generation process. This includes an analysis of test results and making necessary adjustments to the LLM and associated algorithms. The feedback loop ensures the system evolves in response to changing requirements.
* **Documentation and Reporting:** Comprehensive test case documentation is generated, including detailed descriptions, input data, and expected outcomes. Additionally, test reports and summaries are prepared for various stakeholders, highlighting test coverage and identified issues.
* **Integration with Testing Workflow:** To maximize the impact of the automated test case generation system, it is essential to integrate it seamlessly with your organization's existing testing workflow and tools. This integration ensures compatibility with popular testing frameworks and tools, such as Selenium and JUnit.
* **Validation and Testing:** Validation testing is conducted to verify the effectiveness and efficiency of the automated test case generation system. The system is rigorously tested across a range of software projects to ensure its adaptability and performance in various real-world scenarios.
* **Training and Documentation:** To facilitate the adoption of the automated test case generation system, training is provided to testing teams and stakeholders. User documentation and tutorials are created to guide users in effectively utilizing the system.
* **Continuous Improvement:** The project emphasizes the importance of regular updates and fine-tuning of the LLM based on user feedback and evolving software requirements. Continuous monitoring and optimization of the test case generation process are critical for maintaining efficiency and effectiveness.
* **Deployment and Maintenance:** Upon successful development, the automated test case generation system is deployed within your organization. Ongoing maintenance and support are provided to ensure its reliable operation and address any issues that may arise.
* **Evaluation and Metrics:** Key performance indicators (KPIs) are defined to measure the success and impact of the automated test case generation system. The system's performance is evaluated based on metrics such as test coverage, time savings, and defect detection.
* **Scaling and Expansion:** The project also considers opportunities for scaling the system for use in additional software projects and domains. Exploration of integration with continuous integration and continuous delivery (CI/CD) pipelines is also part of the plan, ensuring the system's adaptability to evolving software development practices.

This methodology is designed to guide the implementation of an automated Test Case Generation System using Large Language Models, emphasizing adaptability, integration, and continuous improvement to enhance software testing processes.

A diagram of a model

Description automatically generated

Fig. 4.1 Overview of the model

**4.1 DATA COLLECTION AND PREPARATION**

Data collection for the Test Case Generation System using Large Language Models (LLMs) project is a pivotal stage, central to its effectiveness. Initially, it involves identifying key data sources within the organization, encompassing software documentation, requirements, user stories, and existing test cases. Once identified, a systematic approach is adopted to gather, organize, and structure the data meticulously. Cleaning and preprocessing steps are implemented to ensure data quality and compatibility with LLMs. Notably, a proprietary dataset is created, tailored to your organization's software-related text data, serving as the foundation for fine-tuning the chosen LLM. This dataset is subject to version control to keep it aligned with evolving software requirements. Security and privacy measures are diligently enforced, and comprehensive documentation of the data collection process is maintained.

Data collection is an ongoing process, requiring regular maintenance to keep the dataset current and relevant to software projects. The data's quality and relevance directly influence the LLM's performance in generating test cases and comprehending software-related content. It's imperative to establish robust data management practices, as this forms the cornerstone of the project's success.

In our project, we undertook the task of creating a custom dataset comprising approximately 300 entries. This dataset exhibits a simple yet essential structure, consisting of two columns. The first column is dedicated to recording the names of various products, while the second column serves as a repository for the associated feature lists for which we aim to generate test cases.

This manual dataset creation process was a deliberate and methodical endeavor, undertaken to ensure that the data is finely tuned to our project's specific requirements. By populating the first column with the product names, we establish a clear reference point for the items under consideration. Simultaneously, the second column accommodates the detailed feature lists associated with each product, which will serve as the basis for our test case generation efforts.

**4.2 MODEL DEVELOPMENT**

Model development in the context of the Test Case Generation System using Large Language Models (LLMs) is a critical phase that underpins the entire project. This phase focuses on creating, fine-tuning, and optimizing the LLMs that will be at the heart of the automated test case generation process. Here's an overview of what model development entails:

1. Model Selection:

The process begins with careful consideration and selection of the appropriate LLM for the project. Choices may include popular models like GPT-4, BERT, or other variants. The selection is based on the specific requirements, capabilities, and compatibility with the project's objectives.

1. Fine-Tuning:

Once the LLM is chosen, it is subjected to a fine-tuning process. This step is critical in aligning the model with the project's unique needs. Fine-tuning involves training the LLM on a proprietary dataset that includes software-related text data specific to the organization. The goal is to make the LLM proficient in understanding and generating content relevant to the software projects.

1. Algorithm Design:

Concurrently, algorithms and workflows are designed to facilitate the automated generation of test cases based on the identified test scenarios and feature lists. These algorithms incorporate the LLM's capabilities and are optimized for reliability and efficiency in generating test cases.

1. Iterative Development:

Model development is an iterative process, often involving multiple rounds of fine-tuning and algorithm refinement. The development team continually refines the LLM's understanding of software-related content and its ability to generate contextually accurate test cases.

1. Evaluation:

The effectiveness of the developed model is rigorously evaluated, both in isolation and as an integrated part of the automated test case generation system. Evaluation criteria include the quality of generated test cases, coverage of essential software functionality, and overall efficiency.

Model development is central to the success of the Test Case Generation System, as the LLM's capabilities and efficiency directly impact the quality and comprehensiveness of the generated test cases. It's a dynamic process that requires constant adaptation and refinement to meet the evolving needs of software testing within the organization.

**4.3 MODEL EVALUATION**

Evaluating the model in the context of the Test Case Generation System using Large Language Models (LLMs) is essential to ensure its effectiveness and reliability. The evaluation process is multifaceted and involves various criteria to assess the model's performance. Here's an overview of how the model will be evaluated:

**1. Test Case Quality**: One of the primary evaluation criteria is the quality of the test cases generated by the model. The generated test cases should accurately reflect the intended functionality and requirements of the software under test. A systematic review of the test cases is conducted to ensure they are well-structured, logically sound, and effectively cover the software features.

**2. Test Coverage:** Test coverage measures the extent to which the test cases exercise different parts of the software. The model is evaluated based on its ability to provide comprehensive coverage, ensuring that all critical functionalities, code paths, and edge cases are tested. High test coverage indicates the model's effectiveness in identifying essential test scenarios.

**3. Accuracy:** The model's accuracy in understanding natural language, software-related content, and generating contextually relevant test cases is a crucial evaluation metric. Accuracy is assessed by comparing the generated test cases to the actual software requirements and functionality to determine how closely they align.

**4. Efficiency:** Efficiency is evaluated by considering the time and resources required for the model to generate test cases. An efficient model should be able to produce test cases in a reasonable timeframe, contributing to faster testing processes and reduced testing costs.

**5. Robustness:** The model's robustness is tested by subjecting it to various software projects and scenarios. It should consistently generate reliable test cases across different projects, adapting to changes in software requirements, and demonstrating stability in diverse testing environments.

Evaluating the model is an ongoing process, with regular assessments and updates to ensure that it continues to meet the evolving needs of software testing. The feedback loop is vital, as user feedback and testing results drive continuous improvement and optimization of the model.

**4.4 MODEL COMPARISON**

Model comparison within the Test Case Generation System using Large Language Models (LLMs) is a crucial process that involves selecting and evaluating potential LLM candidates, such as GPT-4 and BERT. These candidates are assessed against predefined evaluation criteria, including test case quality, coverage, accuracy, efficiency, and robustness. User feedback is incorporated to gauge usability and overall satisfaction with the generated test cases. A comparative analysis against traditional manual methods helps quantify the benefits introduced by LLMs, such as improved test case quality and enhanced testing efficiency. The LLM that consistently excels across all evaluation criteria is selected as the most suitable for the project. Moreover, the model comparison process remains dynamic, with ongoing monitoring and reassessment to ensure the selected LLM continues to align with evolving project requirements, thereby optimizing the test case generation process effectively.

In summary, model comparison involves rigorous evaluation of LLM candidates, user feedback incorporation, and benchmarking against traditional methods to select the best-fit LLM. Continuous monitoring and reassessment guarantee its alignment with evolving project needs, ultimately enhancing the efficiency of test case generation.

**4.5 DOCUMENTATION AND REPORTING**

Documenting and reporting within the Test Case Generation System using Large Language Models (LLMs) are essential aspects of maintaining clarity and transparency throughout the testing process. This comprehensive documentation encompasses a variety of elements, including detailed descriptions of generated test cases, input data, expected outcomes, and test steps. These records serve as invaluable references for testers and stakeholders, ensuring that the purpose and execution of each test case are well-documented and easily accessible.

Moreover, test reports are pivotal in summarizing the overall testing activities and results. These reports provide insights into the extent of test coverage achieved, the number and distribution of generated test cases across different software functionalities, and the project's progress and effectiveness. User documentation is designed to guide testing teams and stakeholders in effectively utilizing the LLM-based testing system, offering step-by-step instructions and best practices. Summarized reports tailored for stakeholders offer high-level overviews of the testing process, key findings, test coverage, and identified issues. In addition, custom reports can be generated to focus on specific areas of interest, such as defect tracking or efficiency improvements. The comprehensive documentation and reporting efforts contribute to enhanced understanding, communication, and the continuous improvement of the automated test case generation system using LLMs.

# **CHAPTER 5**

# **TECHNICAL REQUIREMENTS**

**5.1 HARDWARE REQUIREMENTS**

* Computing Resources: High-performance servers or workstations with multi-core processors (e.g., Intel Core i7 or equivalent) for running LLMs and other software components efficiently.
* Memory (RAM): Adequate RAM capacity (e.g., 16GB or more) to handle large datasets and LLM model fine-tuning.
* Storage: Fast and reliable SSD storage for storing project-related data, LLM models, and generated test cases.
* Graphics Processing Unit (GPU): Depending on the LLM and its requirements, a dedicated GPU (e.g., NVIDIA GeForce or Quadro) with sufficient memory can significantly accelerate LLM-related tasks.
* Network Connectivity: Stable and high-speed internet access for downloading LLM models and updates, as well as for collaborating with remote team members.
* Backup and Data Redundancy: A robust backup and data redundancy system to ensure data integrity and recovery in case of hardware failures.
* Peripheral Devices: Standard peripherals like monitors, keyboards, and mice for workstations or servers.

**5.2 SOFTWARE REQUIREMENTS**

* Operating System: A compatible operating system, such as Windows, Linux, or macOS, depending on the team's preferences.
* Programming Language: Programming languages like Python or a language suitable for developing software components and algorithms.
* Integrated Development Environment (IDE): A development environment, such as PyCharm, Visual Studio Code, or Jupyter Notebook, for coding and debugging.

# **CHAPTER 6**

# **SYSTEM DESIGN**

**System Components:** The system comprises several key components, starting with the User Interface (UI) that acts as the front-end. This UI provides users with a means to configure test case generation parameters, manage datasets, and monitor the testing process. On the backend, a dedicated server hosts the core logic of the system, handling requests from the UI and coordinating the entire process. Integrated into the system are the LLMs, which are the central engines responsible for processing natural language requirements, extracting relevant test scenarios, and generating test cases. The specific LLMs, such as GPT-4 or BERT, are carefully selected based on the project's requirements.

**Data Management and Preprocessing:** A robust database management system (DBMS) is used to store and manage datasets, test case templates, and the generated test cases. The DBMS ensures data integrity and provides efficient data retrieval, essential for organizing and accessing the large volume of project-related data. Data preprocessing modules play a vital role in cleaning and formatting input data, making it suitable for LLM processing. These modules involve tasks like text cleaning, tokenization, and data transformation, ensuring that the input data is in a format that the LLMs can effectively work with.

**Test Scenario Identification and Generation:** The heart of the system lies in its ability to identify test scenarios from natural language requirements. This involves the development of algorithmic components that process textual data and extract relevant test scenarios. Subsequently, the Test Case Generation module takes over, employing the LLMs to automatically generate test cases in natural language based on the identified scenarios. The primary goal is to create a reliable and efficient system for test case creation that minimizes manual effort.

**Validation, Testing, and Execution:** The system is equipped with a Validation and Testing module to evaluate the quality and coverage of the generated test cases. This module ensures that test cases adhere to predefined criteria and effectively cover essential software functionality. Additionally, an execution framework is implemented to run the generated test cases on the target software. During the testing process, the system actively monitors test results and records any issues or failures detected, providing valuable feedback to the testing teams.

**Integration, Scalability, and Security:** The system is designed to integrate seamlessly with external tools and APIs to enhance its capabilities. Integration with version control systems, cloud services for data storage, and continuous integration platforms is considered to streamline project management and testing processes. Scalability is achieved through cloud deployment, allowing for resource allocation based on project demands. Security measures are paramount, incorporating access control, encryption, and secure data transfer protocols to protect project data and ensure compliance with data privacy and security standards. The comprehensive system design emphasizes modularity, integration, scalability, and security to enhance software testing processes.

A diagram of software testing

Description automatically generated

Fig. 6.1 System Design UML Diagram

# **CHAPTER 7**

# **CODING AND TESTING**

1. Import the Libraries

!pip install transformers

!pip install accelerate>=0.20.1

import torch

from transformers import GPT2Tokenizer, GPT2LMHeadModel, TrainingArguments, Trainer

from torch.utils.data import Dataset

1. Fine-tune the model and save the model

trainer.train()

trainer.save\_model()

1. Import torch

import torch

print(torch.cuda.is\_available())

!pip install transformers==4.10.0

!pip install accelerate>=0.20.1

!pip install torch==1.9.0

import torch

from transformers import GPT2Tokenizer, GPT2LMHeadModel, TrainingArguments, Trainer

from torch.utils.data import Dataset

1. Define the paths to your dataset and save the directory

dataset\_path = '/content/fine\_tuning.csv'

output\_dir = 'fine\_tuned\_gpt2\_model'

1. Load the dataset

import pandas as pd

df = pd.read\_csv(dataset\_path)

prompts = df['Device'].tolist() # Convert to a list of strings

targets = df['Result'].tolist() # Convert to a list of strings

1. Initialize the GPT-2 tokenizer and model

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

model = GPT2LMHeadModel.from\_pretrained("gpt2")

1. Add a padding token to the tokenizer

tokenizer.add\_special\_tokens({'pad\_token': '[PAD]'})

1. Tokenize the prompts and targets

tokenized\_prompts = tokenizer(prompts, padding=True, truncation=True, max\_length=128, return\_tensors="pt", add\_special\_tokens=True)

tokenized\_targets = tokenizer(targets, padding=True, truncation=True, max\_length=128, return\_tensors="pt", add\_special\_tokens=True)

1. Combine prompts and targets into a dataset

class CustomDataset(Dataset):

def \_\_init\_\_(self, prompts, targets):

self.prompts = prompts

self.targets = targets

def \_\_len\_\_(self):

return len(self.prompts.input\_ids)

def \_\_getitem\_\_(self, idx):

return {

'input\_ids': self.prompts.input\_ids[idx],

'attention\_mask': self.prompts.attention\_mask[idx],

'labels': self.targets.input\_ids[idx],

}

train\_dataset = CustomDataset(tokenized\_prompts, tokenized\_targets)

1. Define training arguments

training\_args = TrainingArguments(

output\_dir=output\_dir,

overwrite\_output\_dir=True,

num\_train\_epochs=3, # Adjust as needed

per\_device\_train\_batch\_size=8,

save\_steps=10\_000,

save\_total\_limit=2,

)

1. Initialise the trainer

trainer = Trainer(

model=model,

args=training\_args,

data\_collator=None, # We're using a custom dataset and data collator

train\_dataset=train\_dataset,

)

# Fine-tune the model

trainer.train()

# Save the fine-tuned model

trainer.save\_model()

1. Models to be used

import torch

from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load the fine-tuned GPT-2 model

output\_dir = 'fine\_tuned\_gpt2\_model' # Update this with your fine-tuned model's directory

model = GPT2LMHeadModel.from\_pretrained(output\_dir)

tokenizer = GPT2Tokenizer.from\_pretrained(output\_dir)

# Define a function to generate results based on device name

def generate\_result(device\_name):

# Construct a prompt with the device name

prompt = f"{device\_name} API Description:"

# Generate text using the model

input\_ids = tokenizer.encode(prompt, return\_tensors="pt")

output = model.generate(input\_ids, max\_length=100, num\_return\_sequences=1, no\_repeat\_ngram\_size=2)

generated\_text = tokenizer.decode(output[0], skip\_special\_tokens=True)

# Extract the result from the generated text

result\_start = generated\_text.find("API Description:") + len("API Description:")

result\_end = generated\_text.find("Utility:")

result = generated\_text[result\_start:result\_end].strip()

return result

# Get user input for the device name

device\_name = input("Enter the device name: ")

# Generate and print the result

result = generate\_result(device\_name)

print("Result:", result)

1. Train and evaluate each model

!pip install transformers==4.10.0

!pip install accelerate>=0.20.1

!pip install torch==1.9.0

import torch

from transformers import GPT2Tokenizer, GPT2LMHeadModel, TrainingArguments, Trainer

from torch.utils.data import Dataset

# Define the paths to your dataset and save directory

dataset\_path = '/content/fine\_tuning.csv'

output\_dir = 'fine\_tuned\_gpt2\_model'

# Load the dataset

import pandas as pd

df = pd.read\_csv(dataset\_path)

prompts = df['Device'].tolist() # Convert to a list of strings

targets = df['Result'].tolist() # Convert to a list of strings

# Initialize the GPT-2 tokenizer and model

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

model = GPT2LMHeadModel.from\_pretrained("gpt2")

# Add a padding token to the tokenizer

tokenizer.add\_special\_tokens({'pad\_token': '[PAD]'})

# Tokenize the prompts and targets

tokenized\_prompts = tokenizer(prompts, padding=True, truncation=True, max\_length=128, return\_tensors="pt", add\_special\_tokens=True)

tokenized\_targets = tokenizer(targets, padding=True, truncation=True, max\_length=128, return\_tensors="pt", add\_special\_tokens=True)

# Combine prompts and targets into a dataset

class CustomDataset(Dataset):

def \_init\_(self, prompts, targets):

self.prompts = prompts

self.targets = targets

def \_len\_(self):

return len(self.prompts.input\_ids)

def \_getitem\_(self, idx):

return {

'input\_ids': self.prompts.input\_ids[idx],

'attention\_mask': self.prompts.attention\_mask[idx],

'labels': self.targets.input\_ids[idx],

}

train\_dataset = CustomDataset(tokenized\_prompts, tokenized\_targets)

1. Fine-tune and save the model

# Define training arguments

training\_args = TrainingArguments(

output\_dir=output\_dir,

overwrite\_output\_dir=True,

num\_train\_epochs=3, # Adjust as needed

per\_device\_train\_batch\_size=8,

save\_steps=10\_000,

save\_total\_limit=2,

)

# Initialize the Trainer

trainer = Trainer(

model=model,

args=training\_args,

data\_collator=None, # We're using a custom dataset and data collator

train\_dataset=train\_dataset,

)

# Fine-tune the model

trainer.train()

# Save the fine-tuned model

trainer.save\_model()

Output:

A screenshot of a computer

Description automatically generated

# **CHAPTER 8**

# **RESULT AND ANALYSIS**

The results and analysis of the Test Case Generation System using Large Language Models (LLMs) project are essential for evaluating its performance and impact. Below are key aspects of the results and the analysis of the project:

**Test Case Generation Efficiency:** One of the primary outcomes of the project is the efficiency of the test case generation process. The system successfully automated the creation of test cases using LLMs, reducing the time and effort required for manual test case development. This led to a significant improvement in test case generation speed.

**Test Coverage and Quality:** The analysis of the generated test cases focused on their coverage and quality. The system demonstrated the ability to cover a wide range of test scenarios based on natural language requirements. The test cases met predefined criteria and effectively covered essential software functionality, ensuring comprehensive testing.

**Reduction in Manual Effort:** A notable result of the project was the substantial reduction in manual effort required for test case generation. Testing teams experienced a reduced workload, allowing them to allocate more time to other critical testing activities. This reduction in manual effort was instrumental in enhancing overall testing efficiency.

**Bug Detection and Reporting:** The automated test cases generated by the system were effective in detecting software bugs and issues. The analysis of test results revealed a notable increase in the detection of defects, including both common and critical issues. The system provided detailed bug reports and summaries to assist developers in addressing these problems.

**Resource Savings:** The project's analysis also highlighted resource savings in terms of both time and workforce. By automating the test case generation process, organizations saved time that would have been spent on manual test case development. Additionally, fewer testing team members were required for test case creation, leading to cost savings.

**Adaptability and Scalability:** The system demonstrated its adaptability to various software projects and domains. Through the analysis of its performance across different projects, it was evident that the LLM-based approach could be applied to a wide range of scenarios. The system's scalability was also assessed, showcasing its ability to handle projects of varying sizes.

**User Feedback and Continuous Improvement:** User feedback played a crucial role in the analysis of the project's success. The feedback loop established in the project allowed for continuous improvements based on user suggestions and evolving software requirements. This iterative approach ensured that the system evolved to meet changing needs effectively.

**Key Performance Indicators (KPIs):** Several KPIs were defined to measure the success and impact of the automated test case generation system. Metrics such as test coverage, time savings, and defect detection rates were evaluated to gauge the system's performance.

In summary, the project's results and analysis indicated a significant improvement in test case generation efficiency, test coverage, and bug detection. The system effectively reduced manual effort, resulting in resource savings, and proved adaptable and scalable for various software projects. Continuous improvement based on user feedback and well-defined KPIs contributed to the project's success in enhancing software testing processes.

# **CHAPTER 9**

# **CONCLUSION AND FUTURE DEVELOPMENT**

The Test Case Generation System using Large Language Models (LLMs) project has demonstrated its potential to revolutionize software testing processes. The project's conclusions highlight several key takeaways:

1. **Efficiency Enhancement:** The project has successfully automated the test case generation process, significantly reducing manual effort and saving time for testing teams. This efficiency enhancement is a major benefit, allowing organizations to streamline their testing workflows.
2. **Adaptability and Scalability:** The system's adaptability to various software projects and domains showcases its versatility. It can be applied to a wide range of scenarios, making it a valuable asset for organizations with diverse software testing needs.
3. **User Feedback-Driven Improvement:** The establishment of a feedback loop ensures that the system evolves in response to user feedback and changing software requirements. Continuous improvement is a key feature of the project.

**Future Developments:**

The project sets the stage for future developments and enhancements in the following areas:

1. **Advanced LLM Integration:** Future development efforts may focus on integrating the latest advancements in Large Language Models, such as newer versions of GPT or other state-of-the-art models. This ensures that the system stays up-to-date with cutting-edge natural language processing capabilities.
2. **Integration with CI/CD Pipelines:** Exploring tighter integration with continuous integration and continuous delivery (CI/CD) pipelines is a logical next step. This would enable seamless test case generation as part of the software development pipeline.
3. **Extension to Different Domains:** The project can be extended to different domains beyond traditional software testing. For example, it can be applied to testing in the fields of robotics, IoT, or other emerging technologies, broadening its applicability.

In conclusion, the project marks a significant milestone in automating test case generation using LLMs. Its success and user feedback provide a solid foundation for future developments that aim to further advance software testing processes and adapt to evolving industry demands.

# 

# **REFERENCES**

[1] I. S. Association et al., “Systems and software engineering vocabulary iso/iec/ieee 24765: 2017,” pp. 1–536, 2017.

[2] Radio Technical Commission for Aeronautics-RTCA, “DO-178C, software considerations in airborne systems and equipmen certification,” 2011.

[3] International Organization for Standardization-ISO, “ISO-26262: Road vehicles – functional safety.”

[4] B. Anda, K. Hansen, and G. Sand, “An investigation of use case quality in a large safety-critical software development project,” Information and Software Technology, vol. 51, no. 12, pp. 1699–1711, 2009.

[5] C. Nebut, F. Fleurey, Y. Le-Traon, and J.-M. Jezequel, “Automatic test generation: a use case driven approach,” IEEE Transactions on Software Engineering, vol. 32, no. 3, pp. 140–155, 2006.

[6] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, “Extracting domain models from natural-language requirements: Approach and industrial evaluation,” in MODELS’16, 2016, pp. 250–260.

[7] L. Provenzano and K. Hanninen, “Specifying software require- ¨ ments for safety-critical railway systems: An experience report,” in REFSQ’17, 2017, pp. 363–369.

[8] M. J. Escalona, J. J. Gutierrez, M. Mej´ıas, G. Aragon, I. Ramos, ´ J. Torres, and F. J. Dom´ınguez, “An overview on test generation from functional requirements,” Journal of Systems and Software, vol. 84, no. 8, pp. 1379–1393, 2011.

[9] M. Shirole and R. Kumar, “UML behavioral model based test case generation: a survey,” ACM SIGSOFT Software Engineering Notes, vol. 38, no. 4, pp. 1–13, 2013.

[10] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang, “Generating test cases from UML activity diagram based on gray-box method,” in APSEC’04, 2004, pp 35

[11] J. Ryser and M. Glinz, “A scenario-based approach to validating and testing software systems using statecharts.” in ICSSEA’99, 1999.

[12] T. Yue, L. C. Briand, and Y. Labiche, “Facilitating the transition from use case models to analysis models: Approach and experiments,” ACM Transactions on Software Engineering and Methodology, vol. 22, no. 1, 2013.

[13] T. Yue, S. Ali, and L. Briand, “Automated transition from use cases to UML state machines to support state-based testing,” in ECMFA’11, 2011, pp. 115–131.

[14] T. Yue, L. Briand, and Y. Labiche, “An automated approach to transform use cases into activity diagrams,” in ECMFA’10, 2010, pp. 337–353.

[15] M. Zhang, T. Yue, S. Ali, H. Zhang, and J. Wu, “A systematic approach to automatically derive test cases from use cases specified in restricted natural languages,” in SAM’14, 2014, pp. 142–157.

[16] E. Sarmiento, J. C. Leite, E. Almentero, and G. S. Alzamora, “Test scenario generation from natural language requirements descriptions based on petri-nets,” Electronic Notes in Theoretical Computer Science, vol. 329, pp. 123–148, 2016.

[17] E. Sarmiento, J. C. S. d. P. Leite, and E. Almentero, “C&L: Generating model based test cases from natural language requirements descriptions,” in RET’14, 2014, pp. 32–38.

[18] J. Gutierrez, M. J. Escalona, M. Mej ´ ´ıas, and J. Torres, “A case study for generating test cases from use cases,” in RCIS’08, 2008, pp. 83–96.

[19] A. L. L. de Figueiredo, W. L. Andrade, and P. D. L. Machado, “Generating interaction test cases for mobile phone systems from use case specifications,” SIGSOFT Software Engineering Notes, vol. 31, no. 6, pp. 1–10, 2006.

[20] G. Carvalho, D. Falcao, F. Barros, A. Sampaio, A. Mota, L. Motta, ˜ and M. Blackburn, “NAT2TEST SCR: Test case generation from natural language requirements based on SCR specifications,” Science of Computer Programming, vol. 95, pp. 275–297, 2014.

[21] “IEE (International Electronics & Engineering) S.A.” <http://www.iee.lu>.

[22] C. Larman, Applying UML and Patterns: An Introduction to ObjectOriented Analysis and Design and the Unified Process. Prentice Hall Professional, 2002.

[23] J. Zhou, Y. Lu, K. Lundqvist, H. Lonn, D. Karlsson, and B. Liwang, “Towards feature-oriented requirements validation for automotive systems,” in RE’14, 2014, pp. 428–436.

[24] T. Yue, S. Ali, and M. Zhang, “RTCM: A natural language based, automated, and practical test case generation framework,” in ISSTA’15, 2015, pp. 397–408.

[25] I. Hajri, A. Goknil, L. C. Briand, and T. Stephany, “Applying product line use case modeling in an industrial automotive embedded system: Lessons learned and a refined approach,” in MODELS’15, 2015, pp. 338–347.

[26] ——, “PUMConf: a tool to configure product specific use case and domain models in a product line,” in ESEC/FSE’16, 2016, pp. 1008–1012.

[27] ——, “Configuring use case models in product families,” Software & Systems Modeling, vol. 17, no. 3, pp. 939–971, 2018.

[28] ——, “Change impact analysis for evolving configuration decisions in product line use case models,” Journal of Systems and Software, vol. 139, pp. 211–237, 2018.

[29] ——, “Incremental reconfiguration of product specific use case models for evolving configuration decisions,” in REFSQ’17, 2017, pp. 3–21.

[30] I. Hajri, A. Goknil, and L. C. Briand, “A change management approach in product lines for use case-driven development and testing,” in Poster and Tool Track at REFSQ’17, 2017.