**LLM-Driven Synthetic Data Generation for API Testing using Natural Language and Data Schema**

DISSERTATION

Submitted in partial fulfillment of the requirements of the

Degree: MTech in AIML

By

**RAGURAMAN T**

2023AA05528

Under the supervision of

**RAGHU RAJENDRAN**

Principal Engineer

Optum Global Solutions

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Pilani (Rajasthan) INDIA

MAY 2025

# Abstract

Software testing is an important phase in the software development lifecycle. It ensures that the application we build achieves the purpose we intended to fulfill. With the emergence of Dev-Ops concepts and modern development methodologies, the software development lifecycle has fairly shortened in duration. Applications are built in iterations with ever varying requirements.

One of the biggest problems the developers and tester would face during their testing is the test data creation. The software needs to be tested for various scenarios and edge cases, which requires large volume of test data covering all possible scenarios. In traditional software testing, the test data is manually created, and it suffers from the following issues- time consuming and error prone due to lack of understanding of the data structure and the schema. Testers would need to spend time to understand the data schema co-relate it with the test scenario and then decide what fields are needed and what not. The quality of test data is also another concern as creating a test data which does not conforms to the schema or standards, puts the entire process at risk. In some cases, the production data is used to create the test data by data obfuscation process which has potential to leak confidential data into test environments.

The difficulties faced by the testers in test data creation are numerous and sometimes frustrating. What if there is a tool which can understand plain English and generates the test data in seconds? A tool which can understands the context, in our case the schema of the data, and generates the data based on users’ natural language input as prompts. A tool which can integrate with various other project management and SDLC tools and aids in simplifying the process of test data generation right from the plan/design phase.

Here I intend to create a tool which can be used to generate test data based on a fixed schema. A schema will be passed to the tool so set the context of the test data structure. An LLM based model will be used to understand the requirements for the test data creation passed by the user. The parsed requirement is then used to generate the test data. There are multiple approaches to generate the data, of which I intend to explore 2 approaches and chose the best to implement to achieve our goal. 1. Leveraging the pre-trained LLM to create a model by prompting, 2. Using the NLU concept of LLM to parse and understand the requirement and then generating the data based on a custom logic. Both the approaches have its own pros and cons which will be analyzed, weighed and picked for implementation.

# List of Abbreviations

# List of Tables

[Table 1 Comparison of Current Data Generation Approaches 12](#_Toc202136641)

# List of Figures

[Figure 1 High Level Architecture of the Solution 12](#_Toc202303267)

# Table of Contents

[Abstract 2](#_Toc202305670)

[List of Abbreviations 3](#_Toc202305671)

[List of Tables 4](#_Toc202305672)

[List of Figures 5](#_Toc202305673)

[Table of Contents 6](#_Toc202305674)

[Chapter 1: Introduction 8](#_Toc202305675)

[Background 8](#_Toc202305676)

[Problem 8](#_Toc202305677)

[Objectives 8](#_Toc202305678)

[Objectives Met 8](#_Toc202305679)

[Chapter 2: Research 9](#_Toc202305680)

[Current Approaches on Test Data Creation 9](#_Toc202305681)

[1. Manual Test Data Creation 9](#_Toc202305682)

[2. Production Data Masking/Obfuscation 9](#_Toc202305683)

[3. Script-Based Data Generation 9](#_Toc202305684)

[4. Record and Replay 9](#_Toc202305685)

[5. Data Generation Tools 10](#_Toc202305686)

[6. AI/ML Based Approaches 10](#_Toc202305687)

[Gaps in Current Methods 10](#_Toc202305688)

[Missing Balance 11](#_Toc202305689)

[A Balanced Approach 11](#_Toc202305690)

[Chapter 3: Literature Review 12](#_Toc202305691)

[Chapter 4: Design & Architecture 13](#_Toc202305692)

[Requirements 13](#_Toc202305693)

[Functional 13](#_Toc202305694)

[Non-Functional 13](#_Toc202305695)

[High-level Design 13](#_Toc202305696)

[Data 13](#_Toc202305697)

[Model (LLM) 14](#_Toc202305698)

[API 14](#_Toc202305699)

[Interfaces 14](#_Toc202305700)

[Scope of the project 14](#_Toc202305701)

[Chapter 5: Result Analysis 15](#_Toc202305702)

[Conclusion 16](#_Toc202305703)

# 

# Chapter 1: Introduction

## Background

Software testing plays a critical role in ensuring that applications function as intended. With the rise of DevOps and agile methodologies, development cycles have become shorter and more iterative. One of the major challenges in this fast-paced environment is generating high-quality, schema-compliant test data efficiently. Manual test data creation is time-consuming, error-prone, and often relies on testers’ deep understanding of complex data structures. This has led to a need for intelligent tools that can automate test data generation using natural language inputs and schema context.

## Problem

Manual test data creation is time-consuming, error-prone, and requires deep understanding of complex data schemas. Existing methods struggle to generate diverse, schema-compliant data quickly, especially in fast-paced DevOps environments. There is a need for an intelligent tool that can generate test data from natural language input using predefined schemas.

## Objectives

Objectives of this project are

1. Creating a tool which generates test data
2. Tool allows us to set the context of the test data by passing a schema document
3. Tool allows to customize the test data using user prompts

## Objectives Met

<TBU>

# Chapter 2: Research

## Current Approaches on Test Data Creation

I did a thorough research on how the test data is generated in the industry at present to gain an in depth understanding of the test data generation methods and their strengths and weaknesses.

### Manual Test Data Creation

Testers create test data by manually entering values that fulfill specific test case requirements. This often requires a strong understanding of the data schema, business logic, and test objectives.

#### ****Challenges:****

* Time-consuming and labor-intensive.
* Error-prone, especially with complex or large datasets.
* Low scalability for large or frequently updated applications.
* Testers must have deep domain and schema knowledge.
* Difficult to maintain consistency and coverage across tests.

### Production Data Masking/Obfuscation

In this method, real-world data from production systems is copied and then sanitized or anonymized before being used in test environments. The goal is to retain the realism and complexity of actual data—like realistic formats, patterns, and relationships—while removing or altering sensitive and personally identifiable information (PII) to avoid data privacy violations.

#### ***Challenges***:

* Data privacy and security risks, especially if masking is incomplete.
* Compliance issues with GDPR, HIPAA, or other data protection regulations.
* High complexity in correctly masking sensitive fields without corrupting data relations.
* Not ideal for all testing scenarios, especially edge cases or error conditions.

### Script-Based Data Generation

Script-based test data generation involves writing custom code (usually in scripting or programming languages like SQL, Python, Java, or Shell) to create and insert test data into a database or data storage system. These scripts are crafted based on the application’s data schema, business logic, and test case requirements.

The approach allows testers or developers to control the kind of data generated, ensuring it meets specific needs such as format, constraints, and relationships among entities.

#### Challenges:

* Requires skilled developers or testers to write and maintain scripts.
* Scripts may break if the underlying schema changes.
* Limited flexibility without frequent updates.
* Difficult to handle dynamic or highly variable test requirements.

### Record and Replay

Record and Replay is a test data reuse strategy where actual data (or requests/responses) from previous test runs or production-like environments is captured ("recorded") and then used again ("replayed") in subsequent test cycles. This method is commonly used in integration testing, API testing, UI automation, and performance testing scenarios. This method is sometimes referred as A/B testing in the industry.

Rather than generating synthetic data each time, this approach uses real interaction data that has been known to work (or fail), helping verify consistent behaviour over time.

#### Challenges:

* Does not cover new or corner-case scenarios.
* Risk of data staleness as the application evolves.
* Limited flexibility to adapt or scale data to different test scenarios.
* Possibility of repeating hidden defects due to unchanged data.

### Data Generation Tools

Data generation tools are specialized software applications designed to automatically create synthetic test data based on predefined rules, schema definitions, and business constraints. These tools help QA teams, developers, and data engineers generate large volumes of realistic, valid, and varied data that closely mimics real-world datasets—without using actual production data.

#### Challenges:

* May require learning curve and licensing costs.
* Generated data can lack realism unless properly configured.
* Some tools have limited support for complex schema relationships.
* Integration issues with custom or legacy systems.

### AI/ML Based Approaches

Uses machine learning models or large language models (LLMs) to learn patterns from existing data or user instructions, then generates realistic or context-aware test data.

#### Challenges:

* Still an emerging field, with ongoing research and limited tools.
* Generated data may lack control and precision for specific test cases.
* Requires fine-tuning models and handling large compute resources.
* Explainability and traceability of generated data can be limited.
* Risk of bias or security issues if training data isn't well curated.

## Gaps in Current Methods

The above section detailed the various testing and data generation techniques exist in the current industry. They all have their own advantages and disadvantages. The below table shows each methods and their trade-offs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Approach | Time Consumption | Flexibility | Skillset needed | Privacy |
| Manual | High | Good | High | Poor |
| Masking | Low | Poor | Low | Good |
| Tool Based | Medium | Good | High | Good |
| Record and Replay | Low | Poor | Low | Poor |
| Script based | High | Good | High | Moderate |

Table 1 Comparison of Current Data Generation Approaches

A problem addressed in one technique is not covered in the other. And most of them had the below common issues that needs attention

* Time consuming
* Need of deeper understanding of the data or generation method
* Privacy and compliance
* Lack of flexibility

### Missing Balance

While each of the methods of data generation or preparation addresses a different issue discussed above, there is a no methodology that addresses all of them. For example,

Masking or Obfuscation based method reduces time by sourcing the data from the production but it lacks flexibility in covering edge cases as most majority of production data falls into a straight forward case.

Data Generation Tools may be designed to cover the edge cases but they lacks realism and needs a testers to be technically strong to customize, update and maintain the tool.

Manual generation of data does not need technical understanding but a deeper understanding of data formats

There is clearly a lack of balance between these approaches and there is a need to develop new solutions that addresses all these short comings.

## A Balanced Approach

The previous section emphasized on the missing balance between various data preparation approaches. There is a clear need for newer solutions that try to bridge the gap in terms of balancing the issues each of the approaches suffering from.

The solution should mainly address the followings

**Time Consumption**: Time taken by a tester or developer to generate/prepare data should be less without compromising the quality

**Flexibility**: The solution is flexible enough to generate data with all possible cases and edge cases

**Knowledge of data**: The solution should take over the knowledge of data structures and generation techniques requiring the testers to possess lesser knowledge on data structure and nuances.

**Security**: The solution should synthesize the data and not use the actual data

# Chapter 3: Literature Review

# Chapter 4: Design & Architecture

## Requirements

This section will describe the requirements of the solution considering the scope of the project and real world usecase.

### Functional

The system should have the following functional requirements

1. Solution should be in the form of a chat interface but should be extensible to other forms via APIs.
2. Solution should allow the user to describe the data required in natural language
3. Solution should accept a schema of the data and generate the data conforming to the schema structure
4. Solution should be capable of generating domain specific data
5. The data generated should be a valid format, e.g., JSON, XML, etc.

### Non-Functional

Following non-functional requirements should be met.

1. Time taken to generate the data should be under 10 seconds.
2. System should be available for 99% of the time.

## High-level Design

The figure below shows a high-level design of the solution.

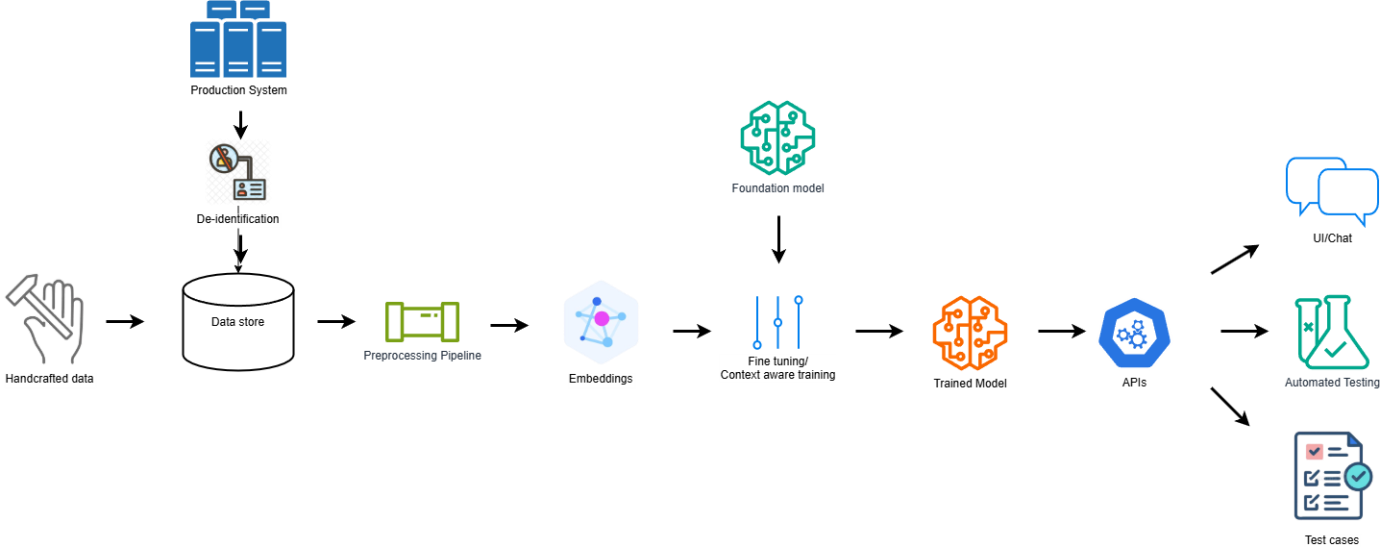


Figure 1 High Level Architecture of the Solution

### Data

The training data for this design can be prepared in two ways

1. Sourced directly from the production systems after de-identifying the user information if any.
2. Hand crafted data covering all possible scenarios

While production data can give various real world scenarios, hand crafted data can provide edge cases and balance of data classes. The solution will support both.

### Model (LLM)

LLM will be used for understanding the user input and reasoning. It will also be fine-tuned with LoRA technique to enable it generate domain specific content. The model will be chosen considering its capability to generate structured data.

### API

The trained model will be served through REST APIs. APIs are the current industry standards when it comes to providing a service. With micro-service architecture, and various options to implement, APIs will be created to expose the model which will be consumed by the end interfaces.

### Interfaces

Once the model is available as an API, the options for interface are endless. The most obvious and meaningful choice will be a chat interface where a user can input the requirement and get the model generate the data.

## Scope of the project

The above design is a aimed at solving the problem with many features included, the project implementation will be limited to proof of concept. For example, the data will only be handcrafted and no production data will be used in this project. The final interface will be chat based UI and no integration to other products will be implemented even though the solution will support it.

# Chapter 5: Technical Implementation

This chapter covers the technical choices and the detailed implementation of the solution.

# Chapter 6: Result Analysis

# Conclusion