A Unified Framework for Verification and Improvement of LLM-Based Automated Unit Test Generation

Supervisor
M r . Mudassar Adeel Ahmad

Group Members
Muhammad Abbas
Muhammad Hammad Ali
Syed Mukhtar-ul-Hassan

BACKGROUND

- Software testing ensures software quality through Unit, Integration, system, and Acceptance testing.
- Integration and Acceptance are well automated, but Unit Testing lacks reliable automation.
- Manual unit test creation is time-consuming.
- LLMs offer a new way to automate unit test generation.

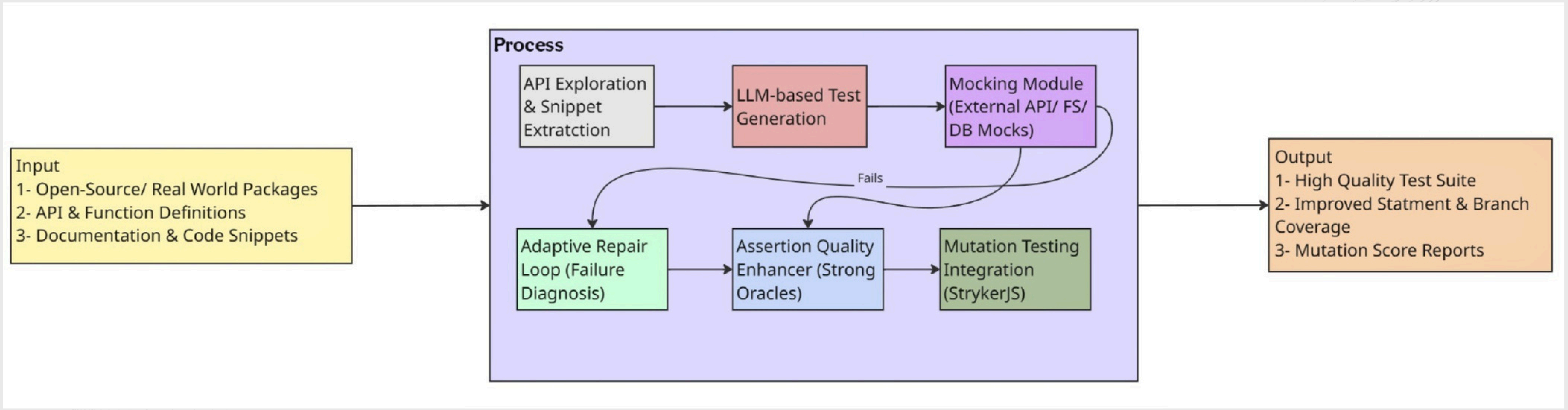
SURVEY OF RELATED RESEARCHES

#	Reference	Input	Verification	Improvement	Output
1	Tip et al. (2025)	JavaScript code	✓	✗	Mutants
2	Primbs et al. (2025)	Java Existing Tests + focal methods	✗	✓	Assertions
3	Nan et al. (2025)	Java code + test intentions	✓	✗	Unit test suite
4	Wang et al. (2025)	Project-level code (multi-language)	✓	✗	Unit test suite
5	Schäfer et al. (2024)	JavaScript code	✓	✗	Unit test suite
6	Sánchez et al. (2024)	OSS developers & projects (multi-language)	✗	✓	Developer insights
7	Alagarsamy et al. (2024)	Java code	✗	✓	Unit test suite
8	Mali et al. (2024)	Hardware designs / specs	✓	✗	Assertions
9	Olsthoorn et al. (2024)	Server-side JavaScript codebases	✓	✗	Unit test suite
10	Chen et al. (2024)	Java code snippets + context	✓	✗	Unit test suite
11	Yang et al. (2024)	Java projects	✗	✓	Evaluation metrics
12	Petrović et al. (2022)	Large-scale codebases (multi-language)	✓	✗	Mutation testing report
13	Sánchez et al. (2022)	Open-source repositories (multi-language)	✓	✗	Adoption dataset
14	Stallenberg et al. (2022)	JavaScript code (no static types)	✓	✗	Test cases
15	Park et al. (2021)	JavaScript programs / spec	✓	✗	Test programs
16	Our Proposed Solution	JavaScript code	✓	✓	Verified, Improved Test Suite

PROBLEM STATEMENT

Existing LLM-based unit test generation tools, such as TestPilot, often produce weak, inconsistent, or incomplete tests which fail to handle dependencies properly, lack strong assertions, and have limited repair capabilities, resulting in less useful tests in real projects.

RESEARCH OVERVIEW DIAGRAM


Input
1- Open-Source/ Real World Packages
2- API & Function Definitions
3- Documentation & Code Snippets
Process
API Exploration & Snippet Extratction
LLM-based Test Generation
Mocking Module (External API/ FS/ DB Mocks)
Fails
Adaptive Repair Loop (Failure Diagnosis)
Assertion Quality Enhancer (Strong Oracles)
Mutation Testing Integration (StrykerJS)
Output
1- High Quality Test Suite
2- Improved Statment & Branch Coverage
3- Mutation Score Reports

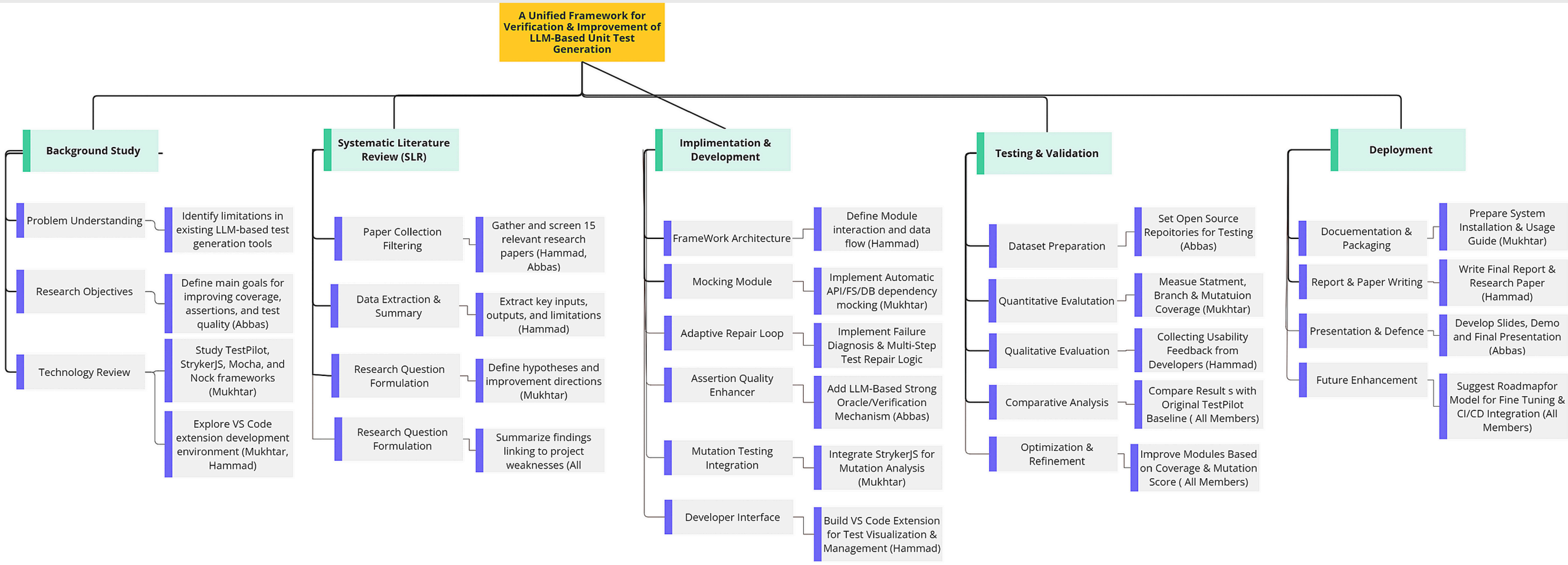
TECH STACK

Programming Language	JavaScript	JS
Framework	Mocha	MOCHA
Coverage Tool	Istanbul / nyc	
Mutation Testing Tool	StrykerJS	
Mocking Libraries	Nock, Sinon	NOCK
Interface	VS Code Extension	

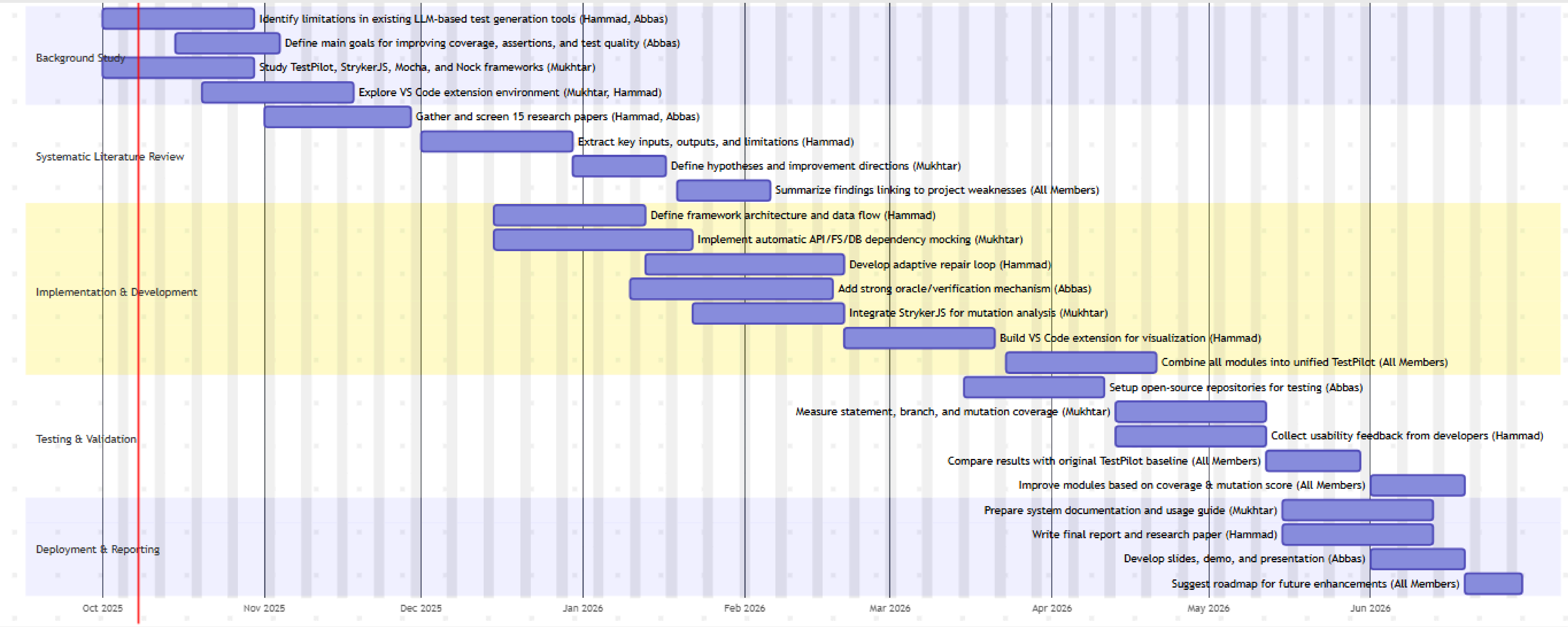
BUSINESS MODEL CANVAS

Problem	Solution	Unique Value Proposition	Unfair Advantage	Customer Segments
Existing LLM-based tools struggle with dependencies, weak assertions, and lack proper verification.	A unified framework to verify, repair, and improve LLM-generated tests using mocking, mutation testing, and an interactive interface.	Generates verified and optimized tests, improving reliability, fault detection, and developer trust.	Combines verification and improvement into one end-to-end system.	Software developers, QA teams, software houses, and research groups focusing on automated testing.
Existing Alternatives	**Key Metrics**	**High-Level Concept**	**Channels**	**Early Adopters**
TestPilot, Nessie, Diffblue Cover, and EvoSuite lack strong assertions, mocking, and effective repair.	Measured by code coverage, mutation score, assertion quality, repair success, and extension adoption.	An intelligent assistant that generates, verifies, and improves unit tests automatically	Open-source release, CI/CD integration, and academic publication.	Research labs, AI coding startups, and open-source LLM contributors.

WORK BREAKDOWN STRUCTURE


A Unified Framework for Verification & Improvement of LLM-Based Unit Test Generation
Background Study
Problem Understanding
Identify limitations in existing LLM-based test generation tools
Research Objectives
Define main goals for improving coverage, assertions, and test quality (Abbas)
Technology Review
Study TestPilot, StrykerJS, Mocha, and Nock frameworks (Mukhtar)
Explore VS Code extension development environment (Mukhtar, Hammad)
Systematic Literature Review (SLR)
Paper Collection Filtering
Gather and screen 15 relevant research papers (Hammad, Abbas)
Data Extraction & Summary
Extract key inputs, outputs, and limitations (Hammad)
Research Question Formulation
Define hypotheses and improvement directions (Mukhtar)
Research Question Formulation
Summarize findings linking to project weaknesses (All
Implimentation & Development
FrameWork Architecture
Define Module interaction and data flow (Hammad)
Mocking Module
Implement Automatic API/FS/DB dependency mocking (Mukhtar)
Adaptive Repair Loop
Implement Failure Diagnosis & Multi-Step Test Repair Logic
Assertion Quality Enhancer
Add LLM-Based Strong Oracle/Verification Mechanism (Abbas)
Mutation Testing Integration
Integrate StrykerJS for Mutation Analysis (Mukhtar)
Developer Interface
Build VS Code Extension for Test Visualization & Management (Hammad)
Testing & Validation
Dataset Preparation
Set Open Source Repoitories for Testing (Abbas)
Quantitative Evalutation
Measue Statment, Branch & Mutatuion Coverage (Mukhtar)
Qualitative Evaluation
Collecting Usability Feedback from Developers (Hammad)
Comparative Analysis
Compare Result s with Original TestPilot Baseline (All Members)
Optimization & Refinement
Improve Modules Based on Coverage & Mutation Score (All Members)
Deployment
Docuementation & Packaging
Prepare System Installation & Usage Guide (Mukhtar)
Report & Paper Writing
Write Final Report & Research Paper (Hammad)
Presentation & Defence
Develop Slides, Demo and Final Presentation (Abbas)
Future Enhancement
Suggest Roadmapfor Model for Fine Tuning & CI/CD Integration (All Members)

PROJECT TIMELINE



THANK YOU