Since the release of ChatGPT, Large Language Models (LLMs) have gained popularity due to their ability to answer questions in detail, ensure grammatical correctness, and generate software unit tests.

Problem Statement: Faults in software unit tests are problematic as they can affect a piece of software's quality, reliability, and security. Software quality must remain high to ensure comprehensive test coverage of essential logic while also maintaining clear and readable code comments and unit tests for maintainability. Software reliability must remain high to ensure that unit tests compile and run correctly, allowing failures to be distinguished between actual bugs in the input program and incorrect expectations in the tests, while also validating expected behavior at all times. Software security must remain high to ensure changes in software code are detected to not introduce vulnerabilities that may expose sensitive data to hackers. As companies increasingly adopt LLMs for generating software unit tests, it's important to recognize that the outputs may not always be fully accurate due to the nature of the prompt used or limitations of LLMs. Therefore, software unit tests generated by LLMs must be carefully evaluated.

Research Objective: The objective of this research is to ensure software unit tests written by LLMs are valid, effective, and readable. Software unit tests are valid if they compile, run successfully, and properly test the input program with correct expectations. Software unit tests are effective if they execute majority of the input program as well as detect any changes to the input program. Software unit tests are readable if their reading grade level is low and their Halstead metrics indicate low complexity, ensuring clarity, ease of understanding, and maintainability.

Evaluation: I propose an evaluation framework that dictates whether software unit tests written by LLM's are valid, effective, and readable enough to replace unit tests written by software engineers into six metrics: build \& success rate, software unit test failure explanation \& evaluation, line coverage, mutation score, readability of code comments, and readability of the code itself. Build \& Success rate measures whether the unit tests are able to be compiled and run successfully without human intervention. Software unit test failure explanation \& evaluation measures whether unit test failures are due to the LLM creating a faulty unit test or actually identifying a bug in the input program. Line coverage measures how much of the input program is being executed when unit tests are being executed. Mutation score measures how many faults a unit test can detect if the faults are purposely inserted into the input program. Readability of code comments measures how easy and effective they are for other software engineers to understand the purpose and functionality of the code. Readability of code measures how easily a software engineer can understand and interact with the code through maintenance and modification. To calculate these metrics, I used IntelliJ, PiTest, the Gunning Fog Index calculator, and the Halstead Metrics calculator. IntelliJ provided the environment for running tests and PiTest. PiTest measured line coverage and mutation scores. The Gunning Fog Index calculator assessed comment readability, while Halstead Metrics measured code readability.

Roadmap: The organization of the rest of the paper is as follows: Chapter 2 discusses related work and other research papers that provided the motivation for this study and inspired the selection of six metrics to evaluate software unit tests generated by LLMs. Chapter 3 discusses my proposed evaluation framework to determine if unit tests written by LLM's are valid, effective, and readable. Chapter 4 discusses how the unit tests written by LLM's are evaluated with experiments and the tools utilized to acquire the results. Chapter 5 presents the results of my experiments. Chapter 6 discusses and analyzes the results of my experiments. Finally, Chapter 7 provides a conclusion of my study of whether software unit tests generated by LLMs are valid, effective, and readable enough to replace those manually written by software engineers as well as future research directions.