## Project 2: The Postcondition Auditor - A Rigorous Analysis of LLM Prompting Strategies

**1. Context & Motivation:** Research at the **Software Engineering Research Centre (SERC), IIIT Hyderabad**, investigates the automation of software verification. This project constitutes a controlled experiment to evaluate the efficacy of Large Language Models (LLMs) as postcondition generators. Your team's mandate is to move beyond demonstration and provide **quantifiable, evidence-based conclusions** on how prompting strategies impact the **correctness, completeness, and reliability** of LLM-generated postconditions.

**2. Primary Objective:** To design a novel evaluation framework and conduct a definitive comparative analysis of three distinct LLM prompting strategies for postcondition generation, measuring their performance against rigorous, pre-defined metrics.

**3. Core Input:** A curated subset of 50 functions from the **Most Basic Programming Problems (MBPP)** dataset. Each function is provided with its signature, implementation, and a set of input-output test cases.

**4. Core Output & Deliverables:** Your team must produce a **software system** and a **comprehensive project report** that includes:

**A. The Experimental Setup:**

- A reproducible pipeline that programmatically processes each of the 50 functions.
- For each function, the pipeline must generate postconditions using **three distinct, documented prompting strategies**:
  1. **Naive Prompt:** A direct, zero-shot instruction.
  2. **Few-Shot Prompt:** Includes three hand-curated examples of correct postconditions for analogous functions.
  3. **Chain-of-Thought (CoT) Prompt:** Explicitly instructs the LLM to reason about the function's purpose, invariants, and edge cases before generating the postcondition.

**B. The Tripartite Evaluation Framework (The Core Challenge):** Your framework must execute and report on three parallel evaluation tracks:

1. **Correctness (Validity) - Measured by Property-Based Testing:**

   - **Implementation:** For each generated postcondition, automatically generate a `hypothesis` test.
   - **Metric:** The **Percentage of Valid Postconditions** for each strategy. A postcondition is valid only if its corresponding test passes for 1000 generated inputs without a single failure.

2. **Completeness (Strength) - Measured by Mutation Analysis:**

   - **Implementation:** For each function, generate a set of 5 plausible mutants (e.g., changing a > to a >=, altering a loop boundary, introducing an off-by-one error).
   - **Metric:** The **Mutation Kill Score (%)**. A postcondition's score is the percentage of mutants for which it correctly fails (i.e., the mutant causes the postcondition to be violated). The **Average Mutation Kill Score** is the key metric for comparing strategy completeness.

3. **Soundness (Reliability) - Measured by Hallucination Audit:**
    - **Implementation:** A static analysis module that parses the generated postcondition string and the function's abstract syntax tree (AST). It must flag any identifier in the postcondition that is not a function parameter, the `result` keyword, or a built-in.
    - **Metric:** The **Hallucination Rate (%)** for each strategy. A single hallucinated variable invalidates a postcondition for this metric.

**5. References & Resources:**

- **Link:** Google Research MBPP GitHub Repository
    - The dataset file is typically named `mbpp.jsonl` or `mbpp.json` in this repository.

**6. Technology Stack:**

- **Language:** Python 3.10+
- **LLM:** A single, specified API (e.g., DeepSeek-Coder).
- **Testing:** `pytest`, `hypothesis`
- **Mutation Testing:** `mutmut`
- **Static Analysis:** `ast` library
- **Analysis:** `pandas`, `seaborn`/`matplotlib`

**7. Team Size:** 5

**8. Point of Contact:** Abhishek Singh (abhishek.singh@iiit.ac.in)