The Postcondition Auditor: A Rigorous Analysis of LLM Prompting Strategies

Team Number 32
Team Members Naveen Mishra (2025201084), Radha Krishna Nallagatla (2025201069), Sriram Paruchuri (2025201074), Vipin Yadav (2025201013), Mohd Shahid Kaleem (2025204008)

About Project

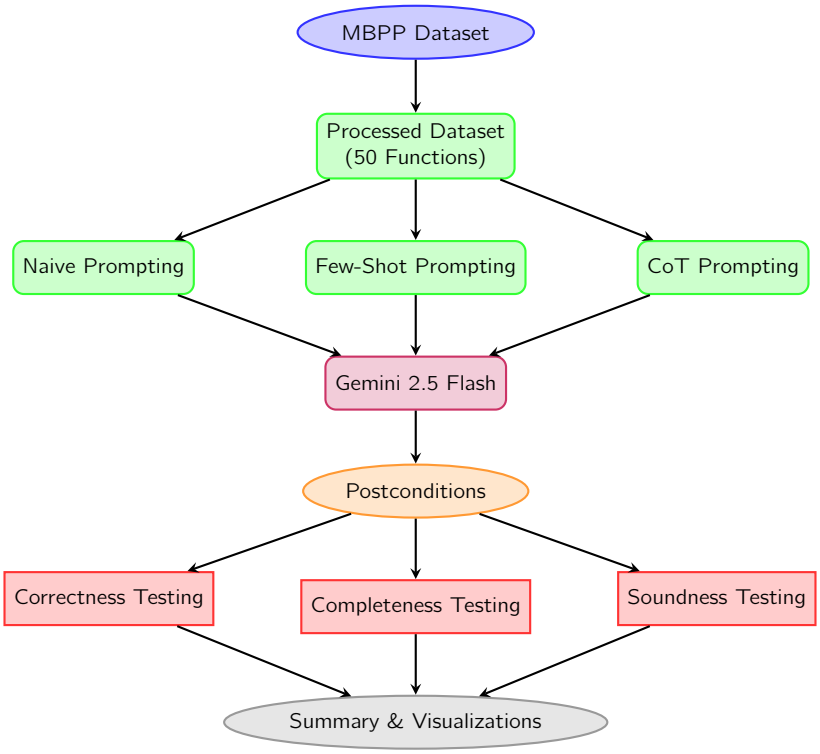
Problem Statement: Large Language Models (LLMs) are increasingly used for automated code generation and verification tasks. However, there is limited quantifiable evidence on how different prompting strategies affect the quality of LLM-generated formal specifications, specifically postconditions. This project investigates whether naive, few-shot, or chain-of-thought prompting produces more *correct*, *complete*, and *sound* postconditions.

1. **Naive Prompt**: A direct, zero-shot instruction where we give the function signature and description and ask the LLM for the postcondition.
2. **Few-shot Prompt**: Includes three hand-curated examples of correct postconditions for analogous functions along with the function signature and description and then ask the LLM for the postcondition.
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 for the provided function signature and description.

Solution: We developed an automated evaluation framework—the **Postcondition Auditor**—that rigorously evaluates LLM-generated postconditions across three dimensions using a **Tripartite Evaluation Framework**:

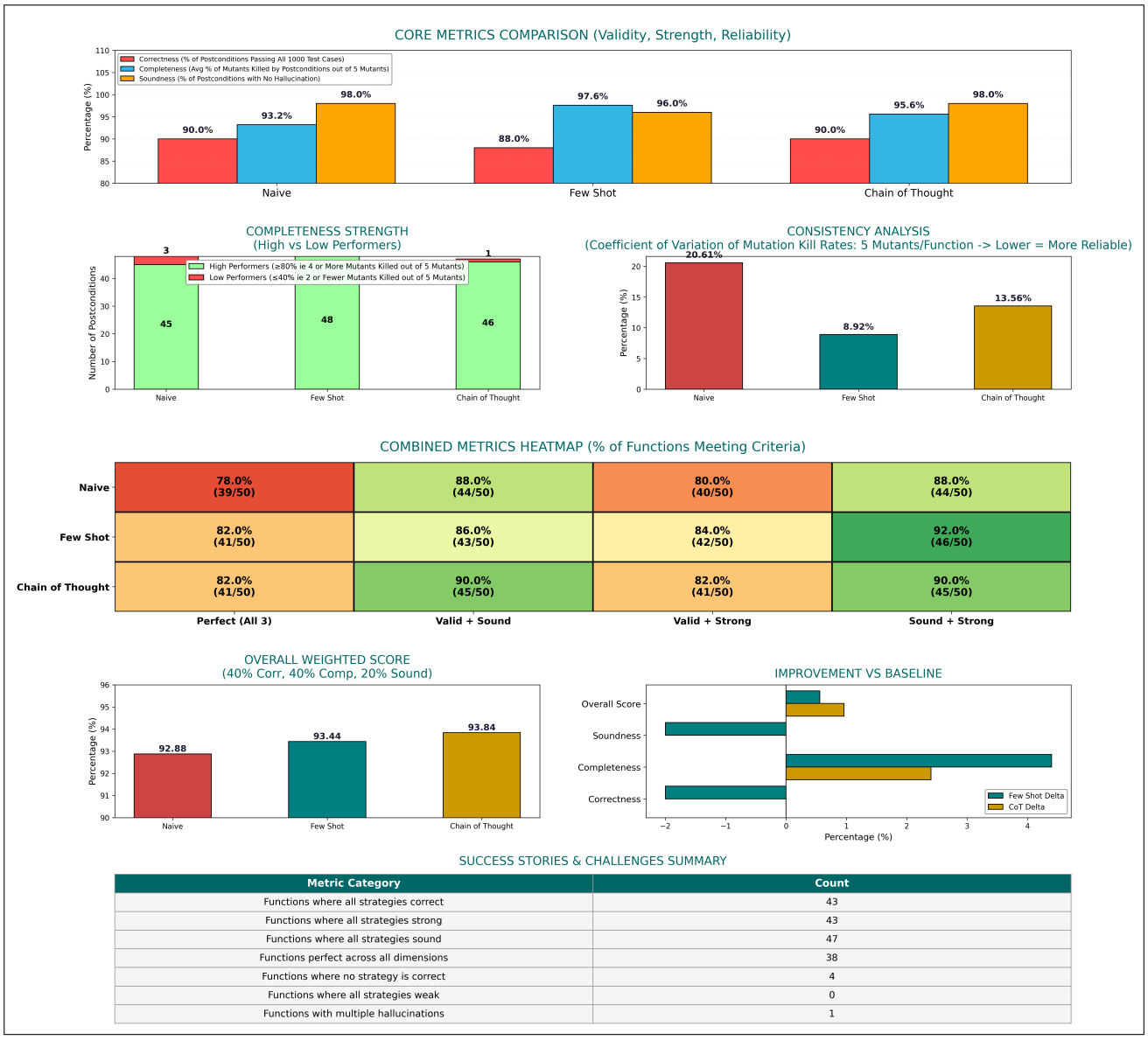
1. **Correctness (Validity):** Property-based testing using Hypothesis with 1000 test cases per function
2. **Completeness (Strength):** Mutation analysis with 5 mutants per function using 6 standardized mutation operators (ROR, AOR, LOR, CRP, UOI, RSM)
3. **Soundness (Reliability):** Static analysis via AST parsing to detect hallucinated variables

Pipeline Architecture:



Summary Dashboard & Visualizations

Below is the comprehensive dashboard showing all the metrics across Tripartite Framework: Correctness, Completeness, Soundness:



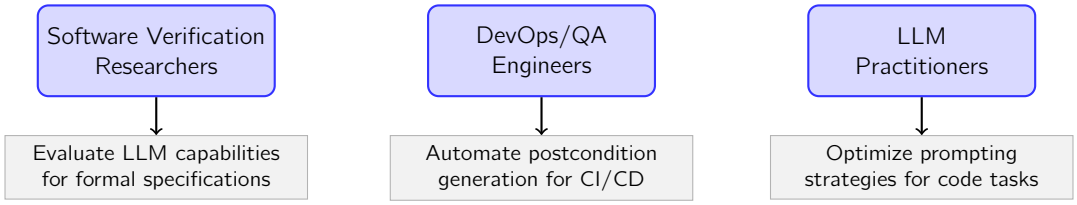
Metric Category	Count
Functions where all strategies correct	43
Functions where all strategies strong	43
Functions where all strategies sound	47
Functions perfect across all dimensions	38
Functions where no strategy is correct	4
Functions where all strategies weak	0
Functions with multiple hallucinations	1

Key Finding: Chain-of-Thought prompting achieves the best overall performance with a weighted score of 93.84, outperforming naive (92.88) and few-shot (93.44) strategies.

Persona and User Journey

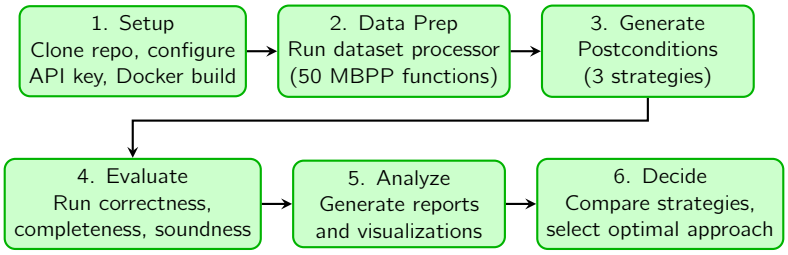
Stakeholders

The Postcondition Auditor serves three primary stakeholder groups, each with distinct goals and workflows:



User Journey Diagram

The following diagram illustrates the end-to-end workflow for using the Postcondition Auditor:



Detailed Stakeholder Workflows

Software Verification Researcher:

1. Configures experiment parameters (number of functions, test cases)
2. Runs full pipeline to generate comparative data
3. Analyzes mutation kill scores to assess postcondition strength
4. Publishes findings on LLM formal specification capabilities

DevOps/QA Engineer:

1. Integrates postcondition generator into CI/CD pipeline
2. Uses Chain-of-Thought prompting (best performer) for production
3. Monitors correctness scores to catch regression bugs
4. Automates contract verification for new code commits

LLM Practitioner:

1. Studies prompting strategy effectiveness across metrics
2. Extracts prompt templates for similar code generation tasks
3. Fine-tunes prompts based on soundness analysis (hallucination detection)
4. Applies learnings to other LLM-assisted development workflows

Software/Hardware

Technology Stack:

- **Language:** Python 3.12+
- **LLM:** Google Gemini 2.5 Flash (via `google-generativeai`)
- **Testing:** Hypothesis (property-based testing)
- **Mutation Testing:** Custom Mutmut-style operators (ROR, AOR, LOR, CRP, UOI, RSM)
- **Static Analysis:** Python `ast` module
- **Visualization:** Matplotlib
- **Containerization:** Docker

Dependencies: `google-generativeai`, `python-dotenv`, `hypothesis`, `matplotlib`, `seaborn`

Contribution

- **Sriram (2025201074):** Dataset processing pipeline, random sampling implementation and visualization dashboard
- **Vipin (2025201013):** Postcondition generation module, LLM prompting strategies design
- **Naveen (2025201084):** Correctness evaluation, Hypothesis integration, test case generation, project coordination
- **Radha Krishna (2025201069):** Completeness evaluation, mutation operator implementation
- **Shahid (2025204008):** Soundness evaluation, AST analysis and summary generation

Conclusion

Our comprehensive evaluation of 50 MBPP functions across three prompting strategies yielded significant insights into LLM-generated postcondition quality:

- **Chain-of-Thought prompting emerged as the best overall strategy** with a weighted score of 93.84, outperforming Few-Shot (93.44) and Naive (92.88) approaches.
- **Correctness:** Both Naive and Chain-of-Thought achieved 90% validity, with Few-Shot slightly lower at 88%. All strategies demonstrate high correctness with minimal variation.
- **Completeness:** Few-Shot prompting excelled with a 97.6% average mutation kill score, demonstrating that example-guided generation produces stronger postconditions. Chain-of-Thought (95.6%) also outperformed Naive (93.2%).
- **Soundness:** Naive and Chain-of-Thought both achieved 98% soundness (2% hallucination rate), while Few-Shot showed slightly higher hallucination at 4%.
- **Consistency:** Few-Shot demonstrated the most consistent performance with the lowest coefficient of variation (8.92%), compared to Chain-of-Thought (13.56%) and Naive (20.61%).
- **Perfect Postconditions:** 82% of functions achieved valid, strong, and sound postconditions with both Few-Shot and Chain-of-Thought strategies, compared to 78% with Naive prompting.

Key Takeaways: The results demonstrate that structured prompting strategies (Chain-of-Thought and Few-Shot) consistently outperform naive approaches for formal specification generation. While Few-Shot excels in completeness and consistency, Chain-of-Thought provides the best balance across all three evaluation dimensions. For production use, we recommend Chain-of-Thought prompting when overall quality is prioritized, and Few-Shot when mutation-resistant specifications are critical.

References

1. Google Research MBPP Dataset: `https://github.com/google-research/google-research/tree/master/mbpp`
2. Google Gemini API: `https://ai.google.dev/gemini-api/docs`
3. Hypothesis Library (Property-Based Testing): `https://hypothesis.readthedocs.io/`
4. Mutmut Mutation Testing: `https://mutmut.readthedocs.io/`