Challenges:

1. Time Consuming
2. Might not be always accurate

Solution to (1):

Preprocessing the input:

Using linters, type checkers and formatters. We would need to identify the language which we can do by the file name (for example: file1.py) and use the specific linter for it.

For python:

* **flake8 (Linter) :**Detects syntax issues, undefined variables, and PEP 8 violations.
* mypy (Type Checker)
* **black (Formatter) :**Formats the code automatically to conform to PEP 8.

**Similarly, C and C++: clang-tidy**

**JS: eslint**

**Java: checkstyle**

And then use the llm

Instead of asking the LLM to find **everything**, let it focus on **specific errors**.

* Pass **checker results** as part of the LLM prompt.
* This makes the LLM focus only on logical and runtime errors.

Reduce overall processing time by **running LLM only on files with errors**.

* **Sort files by errors**: Those with **more errors** go first.

Python and some other languages require indentation checks also which might get missed by an llm

Solution to (2):

Fine-tuning the model, but it requires a dataset.

Prompt engineering: trying different kinds of prompts.

Using models specifically related to code like codellama, deepseek-code.

Postprocessing the code outputs: Validate using checks if the code is running fine, else repeat the process.

Few Shot Learning : We provide with some examples in the prompt but the issue is enormous types of codes and errors.