Task - 3:

Research Plan :

I will begin by identifying **open-source AI models** that are trained on both source code and natural language (e.g., CodeT5, PolyCoder, LLaMA-2). My first step will be to check whether these models can **parse Python programs** and generate meaningful feedback beyond syntax corrections.

Next, I will **collect a dataset of student-written Python code** that contains common mistakes (logic errors, misconceptions, inefficient solutions). Each model will be tested on this dataset to see how well it can:

1. Analyze the code structure and logic.
2. Generate prompts that point out conceptual misunderstandings.
3. Encourage students to reflect without simply giving the solution.

For validation, I will compare the AI-generated prompts against **teacher-written feedback** and also use **rubrics** (clarity, accuracy, depth). If possible, I would further test with a **small group of students** to see whether the AI feedback actually helps them improve their code and understanding.

I will check the feedback with two methods:

1. automatic measures like similarity with expert-written hints, and
2. review by teachers to see if the prompts are clear, helpful, and encourage students to think more deeply.

Finally, I will evaluate each model on **practical criteria** such as ease of use, hardware requirements, response time, and adaptability to educational settings. This step ensures the selected model is not just accurate, but also feasible for classroom or platform integration.

Research Gap:

Traditional feedback systems focus only on correctness (pass/fail or test-case based grading). This creates two major issues:

- Students may still hold misconceptions even when code passes test cases.
- Feedback is often generic, lacking depth or personalization.

while **LLMs (like GPT-4, LLaMA, CodeT5)** show strong potential in producing human-like feedback, most prior work relies on **manual prompt engineering** and limited evaluation methods. There is a gap in automating **high-level competence analysis** that balances accuracy, interpretability, and scalability.

Reasoning:

What makes a model suitable for high-level competence analysis?

- A good model should not just read code, but also understand what the student is trying to do.
- It should be able to notice logical or conceptual mistakes and give feedback in simple, encouraging language.
- The feedback should guide the student toward thinking critically instead of just fixing the bug.
- When combined with **educational prompt design**, these models can highlight reasoning gaps (e.g., “Why did you use recursion here? Could iteration work better?”).

How would you test whether a model generates meaningful prompts?

- I would test this by comparing the model’s prompts with hints written by teachers, and also by asking educators to rate them for clarity and usefulness.
- In addition, I would check with students to see if the prompts actually help them debug and learn better.

What trade-offs might exist between accuracy, interpretability, and cost?

- Big models may give very accurate results, but they are slow and expensive to run.
- Smaller models are cheaper and faster, but may miss some complex mistakes.
- Some models are hard to interpret, meaning it is not always clear why they gave certain feedback.
- In education, it’s important to balance these factors so that the system is affordable and reliable, while still helpful for students.

Why did you choose the model you evaluated, and what are its strengths or limitations?

- I chose CodeBERT as an example because it has been trained on both code and text, so it can understand context and also explain things in words.
- Its main strength is that it is open-source, widely used, and performs well on many code-related tasks.
- But it still needs fine-tuning for education and may not always handle very complex or unusual errors.
- Compared to very large models like Codex, it is more practical to deploy, but it may give less accurate answers in zero-shot cases.