**Evaluating Open-Source AI Models for Student Competence Analysis in Python**

**(FOSSEE Internship Screening Task)**

**Aim:**

To evaluate open-source AI models for student competence analysis in Python learning.

**What will I be using?**

I will be evaluating OpenAI’s GPT-based open model GPT-Neo.

* GPT- Neo is an Open-source AI driven Model made by EleutherAI. It is like Open AI’s GPT-3 but with open weights, Open training data and open code.
* EleutherAI has released several versions of the GPT-Neo such as GPT-Neo 1.3B which has about 1.3 Billion parameters and GPT-Neo 2.7B which has about 2.7 Billion parameters.
* The Key advantages of this Open-Source AI model are that it is:
* Open Source, meaning that you can download your own weights, fine-tune it, study it and customise it the way you would like to.
* Good trade-offs of cost vs performance, since people with low subscription budgets can also use it due to it being smaller and openly available.

Transparent Data and architecture, Since EleutherAI publishes its training datasets and details, it allows for auditing & researching to identify biases and understand where they may lie.

**Research Plan:**

* The evaluation process would involve testing the model’s ability to read and interpret student-written Python code, recognize errors or misconceptions, and generate questions or prompts that encourage the student to reflect on their reasoning. I would prioritize models that have been pre-trained on programming datasets, as they are more likely to provide accurate feedback on code structure, syntax, and logic.
* The assessment criteria would focus on **accuracy**, **depth of feedback**, **interpretability**, and **adaptability for educational use**. To test applicability, I would design sample student submissions with common mistakes (e.g., infinite loops, off-by-one errors, misuse of recursion) and evaluate how well the model highlights conceptual misunderstandings. Additionally, I would check whether the prompts generated are constructive and open-ended (e.g., “Why did you choose a loop here instead of recursion?”) rather than simply giving away the solution.
* A small-scale validation could be done by comparing the model’s prompts against those generated by experienced Python instructors to ensure alignment with effective teaching practices.

**Reasoning:**

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

* A suitable model should not only detect surface-level syntax errors but also interpret logic and reasoning behind a student’s code. It should generate prompts that push students toward reflection and deeper understanding. This requires exposure to both natural language and programming data during training.

2. How would you test whether a model generates meaningful prompts?

* By giving the model diverse student code samples and checking if its output encourages critical thinking (e.g., “Can you explain why this function always returns the same result?”) rather than only corrections. Comparison with educator-designed prompts would help validate quality.

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

* High-accuracy models (like large LLMs) may require significant computational resources, making them costly. Smaller open-source models are cheaper and easier to interpret but may lack depth in reasoning. There is a balance between explainability (important for education) and performance.

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

* I chose CodeT5 (Hugging Face) as a candidate because it is specifically trained on code and natural language pairs, making it strong at both code understanding and explanation. Its strength lies in detecting code-level issues and explaining them in natural language. However, its limitation is that it may sometimes produce generic feedback and may require fine-tuning on student code datasets for educational relevance.