**Python Screening Task 3: Evaluating Open Source Models for Student Competence Analysis**

**Research Plan**

Alright, so my first step would be to get a lay of the land. I'd start by digging through places like GitHub and AI research papers to find models tagged for code intelligence or educational NLP. I'm not just looking for any code model; I need ones that can explain and reason, not just complete code. My top candidates would probably be instruction-tuned models like Code Llama-Instruct or Star Coder, because they're built to follow commands and have a conversation about code, which is exactly what we need for generating prompts.

To actually test them, I'd build a small benchmark of real, beginner-to-intermediate Python code snippets. I've seen my colleagues write stuff with classic mistakes like off-by-one errors, misunderstood scope, or inefficient loops. For each snippet, I'd create a specific prompt asking the model to act like a tutor to generate a question that makes us think about why our code works a certain way, without giving the answer away. The real validation would come from having actual Python instructors review the model's outputs.

**Reasoning**

**1. What makes a model suitable for high-level competence analysis?**  
It has to go way beyond checking if the code runs. It needs to understand the intent behind the code and the student's thought process . A suitable model can identify the underlying concept a student is struggling with (like recursion or mutability) and reason about it on a conceptual level, not just a syntactic one. It's more about being a teaching assistant than a code validator.

**2. How would you test whether a model generates meaningful prompts?**  
The only way to know for sure is with expert human review. I'd have experienced teachers rate the model's output prompts based on a few rules: Does the question target the specific misconception? Is it open-ended and make the student reflect? And crucially, does it guide them to discover the answer themselves? If instructors consistently say "Yeah, I'd actually use that question in class," then it's meaningful.

**3. What trade-offs might exist between accuracy, interpretability, and cost?**  
This is the big dilemma. The most accurate models (like GPT-4) are incredibly expensive to run and are complete black boxes—you can't see how they got their answer. Free, open-source models are cheaper and more transparent, but their accuracy isn't as perfect. You might save money but then have to spend more time carefully prompt-engineering and checking the model's work to catch its mistakes. There's no free lunch; you're always trading one thing for another.

**4. Why did you choose the model you evaluated, and what are its strengths or limitations?**  
I'd start with a model like **CodeLlama-13B-Instruct**. I chose it because it's small enough that a school could potentially run it on a single powerful GPU (making it cost-effective and keeping student data private), but it's also large enough and, more importantly, designed for having conversations about code.