Research plan

Paragraph 1 — approach to identifying & evaluating models.

I will begin by scoping freely available models that have explicit code-understanding capabilities (e.g., StarCoder / CodeGen / CodeT5 variants and lightweight LLMs that can be fine-tuned) and filter by license, token/context length, and local/federated deployment feasibility. For each candidate I'll collect model artifacts (weights, tokenizer, sample prompts) and toolchain requirements (inference RAM/VRAM, latency, and cost to run). Evaluation will use a curated dataset of student Python submissions (a mix of short exercises: functions, small projects, common bugs) plus human-annotated competence labels and pedagogical prompts. I'll run two pipelines: (A) automated probe — ask the model to *generate diagnostic prompts* for each student submission (e.g., Socratic questions, targeted misconceptions checks), and (B) critique — ask the model to *identify likely misconceptions* and provide minimal hints. I'll log outputs, compute quantitative metrics (alignment with expert prompts, coverage of Bloom's taxonomy levels, hallucination rate), and run a small human-in-the-loop validation where educators rate usefulness and non-spoiling quality.

Paragraph 2 — how to test/validate applicability.

Validation will combine offline and live testing. Offline: compare generated prompts against a gold set created by experienced instructors and measure precision@k for relevant conceptual cues, plus diversity of prompts across Bloom levels. Live: a small classroom A/B test where one group receives model-generated Socratic prompts and another receives standard static hints; measure downstream signals (revision quality, time-to-correct, and a short conceptual quiz). I'll also perform failure-mode analysis (cases where the model gives wrong diagnoses or over-specified hints) and measure cost/latency for realistic deployment (e.g., per-student interactive session). Final deliverables will include a README with setup + inference scripts, a reproducible evaluation notebook, example prompts, and a short recommendation on whether the model is fit-for-purpose or needs fine-tuning / prompt engineering / retrieval augmentation.

Reasoning (required)

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

• Strong code semantic understanding: ability to parse and summarise program intent, control/data-flow, and common idioms.

• Flexible natural-language generation: can produce tuned Socratic, diagnostic, and scaffolded prompts without revealing full solutions.

• Calibrated confidence & interpretability: outputs that can be traced to code regions (e.g., explain why it suspects a misconception).

• Practicality: permissive license, reasonable inference cost, and sufficient context window to handle typical student submissions.

• Amenability to fine-tuning or tool-augmentation (e.g., retrieval of rubric/examples) to reduce hallucination.

How would you test whether a model generates meaningful prompts?

1. Create a gold standard set: for each student program, gather 3–5 instructor-written prompts across Bloom's levels.

2. Automatic metrics: measure overlap / semantic similarity between model prompts and gold prompts (embedding cosine), uniqueness/diversity, and proportion of prompts that target the real bug/concept.

3. Human evaluation: educators rate prompts on (a) non-spoiling quality, (b) relevance to student misconception, and (c) potential to elicit deeper reasoning.

4. Learning signal: run a small study to measure whether students who received model prompts improve more on a short concept test and on subsequent code revisions.

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

• Accuracy vs. cost: larger models (better accuracy) require more compute / latency — expensive for real-time interactive use.

• Interpretability vs. capability: highly capable models often behave as black boxes; adding explicit explainers or attention visualizations helps interpretability but adds complexity.

• Cost vs. safety: cheaper models may hallucinate or give incorrect hints, harming learning; safer behavior often requires retrieval augmentation or fine-tuning (extra cost).

• Delivery trade-off: run a smaller on-device model (lower cost, higher privacy) vs. server-side large model (higher accuracy but operational cost & latency).

Why I chose the model I evaluated (example: StarCoder) — strengths & limitations.

Choice rationale: StarCoder (BigCode) is a strong open-source starting point for code tasks: permissive license, optimized tokenization for programming languages, public checkpoints, and a community focused on code understanding/generation. It supports large context windows and can be fine-tuned on pedagogical corpora.

Strengths: good at parsing code, producing short explanations, and being fine-tuned or combined with retrieval to reduce hallucination; available for local deployment which helps privacy for student data.

Limitations: out-of-the-box it is not trained specifically for pedagogy — it may either give solution-level hints or miss subtle misconceptions; it still hallucinates and can be overconfident. It may also need substantial compute (GPU) for interactive latency, and instructor-in-the-loop is necessary for quality control. Practical deployment will likely require: prompt engineering, constrained-output templates (to avoid revealing full solutions), small fine-tuning on annotated student-code→prompt pairs, and a lightweight reasoning pipeline that highlights code locations that motivated each prompt.

Practical next steps (what to include in the GitHub submission)

- `README.md`: short summary, model choices, quickstart for running evaluation.

- `environment.yml` / `requirements.txt`: reproducible setup.

- `inference/` scripts: run model on sample student submissions and save generated prompts.

- `evaluation/` notebook: automatic metrics, embedding similarity, and educator rating collection template.

- `examples/` : 10 student problems, gold prompts, and sample model outputs.

- Short `privacy_and_limitations.md`: notes on student data handling and pedagogical risks.