Problem Statement

Background

Our company is building intelligent coding assistants that require robust small models capable of understanding, generating, and retrieving code. To test your ability to adapt and fine-tune small transformer models under limited compute (Google Colab T4 or similar), we ask you to perform three related tasks:

- **Extended Pre-Training** – Continue training a small **code language model** (e.g., TinyLlama, Qwen-Coder, CodeGemma, etc.) on a **small curated code corpus** (100–300 files).
  This simulates how models are adapted to a new codebase or organization's style.All we want to understand is how you can take a small repo and perform continued pre-training over it.
- **Instruction Tuning (SFT)** – Fine-tune a small model on synthetic input-output pairs for coding tasks.
  Decide a series of various instructions you can use to create those pairs.
- **Embedding Fine-Tuning** – Adapt a small embedding model for text → code retrieval and evaluate its retrieval quality.
  You can use any vector DB of your choice.

NOTE : Hey, you can tackle this assignment in a bunch of ways. We're just curious to see your unique approach. If you've got a better idea than ours, totally let us know!

Task

Perform the following three tracks (at least **two** must show measurable improvement to pass):

- **Track A – Extended Pre-Training:**
  Run a few epochs of **Extended Pre-Training** on a curated code dataset.
  Report baseline perplexity vs after-training perplexity.

  Masking Notes
  - Use the standard **next-token prediction** objective.

    - **Causal masking** (preventing the model from seeing future tokens) happens automatically in decoder LMs — no manual work needed.
    - Apply **loss masking** to ignore padding or special tokens during loss computation (handled automatically if you use `DataCollatorForLanguageModeling(mlm=False)`).
- **Track B – Instruction Tuning (SFT):**
  Perform small-scale fine-tuning (≈ 300 – 600 synthetic pairs).
  Evaluate on a mini test set of coding tasks. (Use pass@k where k can be 1-3)
- **Track C – Embedding Fine-Tuning:**
  Fine-tune a small embedding model (e5-small / BGE-small / GTE-small) on text–code pairs.
  Evaluate retrieval metrics (MRR@10, nDCG@10, Recall@10).

Brownie Points

If you document a **Data Card** for your synthetic data (prompt templates, seeds, filtering rules, distributions, edge cases), we'd love that!
It'll help us understand how well you'd work with our machines collaboratively.

(Seriously though, if you're not using the src for the code files, we won't let you use our machines. LOL.)

Challenge Focus

This task specifically tests your ability to:

- Handle data preparation with synthetic LLM generation.
- Adapt small transformer models for pre-training, fine-tuning, and retrieval under resource limits.
- Evaluate models with meaningful metrics beyond accuracy.
- Balance engineering clarity and research rigor inside a Colab notebook.

Examples of Instruction Prompts

- "Explain what the following function does."
- "Write a docstring for this function."
- "Fix the bug in this snippet."

- "Suggest improvements to this piece of code."
- "Generate unit tests for this function."

Examples of Retrieval Queries

- Query: "function to add two numbers (think of a different function type for a given repo)" → Code: `def add(a,b): return a+b`
- Query: "how to open custom_file.ts (imagine a custom file in a codebase)" → Code: `with open("file.txt") as f:`
- Query: "sort a list in ascending order (again, for a custom repo)" → Code: `arr.sort()`

Requirements

Data Preparation

- Curate or generate a tiny code corpus (100 – 300 files) for Extended Pre Training.
- Generate 300 – 600 instruction–response pairs for SFT.
- Prepare 200 – 400 text–code pairs for retrieval.
- Document all generation steps in a **Data Card**.

Model Development

- Use a small **causal LM** (TinyLlama-1B, Qwen-coder (recommended), or any choice of yours) for **Extended Pre-Training**.
- Use the same or similar small model for instruction tuning.
- Use a small embedding model (e5-small / BGE-small / GTE-small) for retrieval.
- Keep training to ≈ 2-4 epoch per track (Colab-friendly).

Evaluation

- **Extended Pre Training:** Perplexity before vs after training.
- **SFT:** Pass-rate of unit-test stubs, style score (docstring/PEP8), hallucination rate.
- **Embeddings:** MRR@10, nDCG@10, Recall@10 on held-out set.
- Provide error analysis (10 – 20 failure cases).

Code Quality

- Write clean, well-documented code inside a single Colab notebook.
- Use Transformers, PyTorch, PEFT, Sentence-Transformers as needed.

- Organize with clear markdown headers (Track A, B, C).

Deliverables

- **Notebook:** `ML_Assignment_Part2.ipynb` with complete solution and analysis.
- **HTML Export:** for quick review.
- **Metrics File:** JSON summary of results.
- **Data Card:** included within the notebook.

Evaluation Criteria

Your solution will be evaluated on:

- How well you interpreted the problem and built your pipelines.
- Quality of data generation and Data Card documentation.
- Code clarity, readability, and reproducibility.
- Depth of evaluation (metrics + failure analysis).
- Handling of compute constraints (must run on Colab T4).

NOTE: We're not expecting state-of-the-art numbers. We want to see your methodology — how you thought, structured, and executed the task.
Using AI tools for coding is fine, as long as your approach and understanding are clear.

Time Limit

You have 3 days to complete the deliverable.
All the very best!

We understand that you might use AI tools for writing the scripts, but your methodology is what truly matters to us.