Hackathon Prototype Roadmap: Medical Coding AI

This roadmap outlines a 7-day plan (25 total working hours) to build a functional prototype for the Medical Coding AI, focusing on demonstrating the core "clinical note-to-code conversion" using an inference-based approach.

Total Working Hours: 25 hours

- **Weekdays (Mon-Fri):** 3 hours/day (15 hours total)
- **Weekend (Sat-Sun):** 5 hours/day (10 hours total)

Phase 1: Foundation & Initial Data (Total 4-5 hours)

Day 1 (3 hours): Setup & Environment

- **Goal:** Prepare the development environment and install core tools.
- **Tasks:**
  - **Project Setup (1 hr):**
    - Create project directory.
    - Initialize Git repository (`git init`).
    - Ensure team members have access.
  - **Environment Setup (2 hrs):**
    - Create Python virtual environment (`python -m venv venv`).
    - Activate environment (`source venv/bin/activate` or `.\venv\Scripts\activate`).
    - Install libraries from `requirements.txt`: `pip install -r requirements.txt`.
    - Verify GPU setup (e.g., `import torch; print(torch.cuda.is_available())`).

Day 2 (1-2 hours): Minimal Data Preparation

- **Goal:** Create a tiny, representative dataset for immediate testing and demo.
- **Tasks:**
  - **Simulated Data Creation (1-2 hrs):**
    - Manually create **5-10 very simple, diverse clinical notes** (e.g., in a `data.csv` or a Python list of dicts).
    - For each note, manually assign 1-3 highly relevant ICD-10 and CPT codes. **This data is for demonstration and basic testing, not extensive training.**
    - *Example Note:* `"Patient has a fever and cough. Diagnosis: Acute Bronchitis."`
    - *Example Codes:* `ICD-10: J20.9`, `CPT: 99203`
  - **Load & Inspect Data (0.5 hr):** Write a simple Python script to load and print your simulated data to confirm structure.

Phase 2: Core NLP Logic - Inference Focus (Total 8-10 hours)

Day 2 (Remaining 1-2 hours): Basic Transformer Inference

- **Goal:** Verify loading and basic usage of a pre-trained transformer model.
- **Tasks:**
  - **Load Pre-trained Model (1 hr):**
    - Load `BioClinicalBERT` (or a suitable alternative like `microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext` if BioClinicalBERT proves complex for quick setup) from Hugging Face Transformers.
    - Use `AutoTokenizer` and `AutoModel`.
  - **Basic Encoding/Prediction (0.5-1 hr):**
    - Write a function that takes a raw clinical note.
    - Tokenizes the note using the loaded tokenizer.
    - Passes the tokens through the model to obtain embeddings (e.g., `model(**inputs).last_hidden_state.mean(dim=1)` for sentence embedding).

Day 3 (3 hours): Simplified Code Prediction (Inference-based)

- **Goal:** Implement a simplified, semantic similarity-based approach to predict codes.
- **Tasks:**
  - **Define Code Labels (1 hr):**
    - Create a small, fixed list of 5-10 common ICD-10/CPT codes you want to demonstrate.
    - For each code, write a concise, descriptive phrase (e.g., `{"code": "G43.901", "description": "Migraine headache, unspecified"}`).
  - **Semantic Similarity for Code Prediction (2 hrs):**
    - Encode each code's descriptive phrase using your pre-trained BioClinicalBERT model to get its embedding.
    - When a new clinical note is input:
      - Encode the note's text to get its embedding.
      - Calculate the **cosine similarity** between the note's embedding and each code phrase's embedding.
      - Select the top N codes with the highest similarity as the "predicted" codes.

Day 4 (3 hours): Basic Justification & Pre-processing

- **Goal:** Enhance the core logic with rudimentary text cleaning and "audit-ready justification."
- **Tasks:**
  - **Text Cleaning (1 hr):**
    - Implement basic text pre-processing steps for input notes: lowercasing, removing extra whitespace, punctuation, and potentially common stop words.
  - **Rudimentary Justification (2 hrs):**
    - For each predicted code, identify and highlight key medical terms or phrases from the original input note that are semantically related to that code.
    - This can be done using simple keyword matching (if "migraine" is in the note and the migraine code is predicted, highlight it) or by exploring attention mechanisms if time permits.
    - Present this as a "Key Phrases" or "Supporting Terms" list alongside the predicted codes.

Phase 3: API & UI Development (Total 6-7 hours)

Day 5 (3 hours): Backend API with Flask/FastAPI

- **Goal:** Create a RESTful API endpoint to expose your core NLP functionality.
- **Tasks:**
  - **API Skeleton (1 hr):**
    - Set up a Flask or FastAPI application.
    - Define a single POST endpoint (e.g., `/predict_codes`).
  - **Integrate NLP Logic (2 hrs):**
    - The endpoint should accept a JSON payload with a `note` field (the clinical text).
    - Call your text cleaning, semantic similarity prediction, and justification functions.
    - Return a JSON response containing `icd_codes`, `cpt_codes`, and `justification_phrases`.
  - **Test API (0.5 hr):** Use `curl` or a browser extension (e.g., Postman/Insomnia) to test the API locally.

Day 6 (5 hours): Frontend UI with Streamlit/Gradio

- **Goal:** Build a simple, interactive web interface for demonstration.
- **Tasks:**
  - **UI Layout (1-2 hrs):**
    - Create a Streamlit (`streamlit run app.py`) or Gradio (`gradio app.py`) application.
    - Design a clean layout with:
      - A large text area for "Clinical Note Input".
      - A "Predict Codes" button.
      - Clear display areas for "Predicted ICD-10 Codes", "Predicted CPT Codes", and "Audit Justification (Key Phrases)".
  - **Connect UI to API (1 hr):**
    - Write Python code within your Streamlit/Gradio app to make HTTP requests to your local Flask/FastAPI backend when the button is clicked.
    - Parse the JSON response and display the results in the respective output areas.
  - **Example Notes (0.5 hr):**
    - Add a dropdown or simple buttons to quickly load your pre-defined simulated notes into the input text area for easy demonstration.
  - **Basic Styling (0.5 hr):** Apply minimal styling for better readability and presentation within the framework's capabilities.

Phase 4: Polish & Presentation (Total 6-7 hours)

Day 7 (5 hours): Refinement, Testing & Pitch Preparation

- **Goal:** Ensure the prototype is stable, looks polished, and the pitch is compelling.
- **Tasks:**
  - **End-to-End Testing (2 hrs):**
    - Thoroughly test the entire system, from UI input to API response and output display, using all your simulated notes.

        - Identify and fix any remaining bugs or display issues.
    - **UI/UX Polish (1 hr):**
        - Make minor visual adjustments to improve clarity and user experience (e.g., clear labels, informative messages, better formatting of results).
    - **Pitch Deck & Script Finalization (1 hr):**
        - Rehearse your hackathon pitch, ensuring it's concise, impactful, and within time limits.
        - Practice the live demo flow to ensure it's smooth and highlights the core functionality effectively.
        - Prepare for potential questions from judges.
    - **Error Handling Review (0.5 hr):**
        - Ensure the UI provides user-friendly messages if the API or NLP process encounters an error.
    - **Final Code Review & Comments (0.5 hr):**
        - Add comprehensive comments to your Python code, explaining complex logic, functions, and data flows.
        - Ensure code is clean and adheres to basic best practices.
