Technical Assessment: Medical Text Summarization Prototype

Time Expectation: 4-6 hours
Submission Format: GitHub repository with documentation

Overview

Create a proof-of-concept medical text summarization system that could help clinicians quickly understand patient cases. Your solution should demonstrate both technical competence and creative thinking about improving clinical workflows.

Core Requirements

1. Basic Functionality

- Create a Python-based application that takes medical text input (e.g., clinical notes) and generates concise, structured summaries
- Use a large language model of your choice (OpenAI API key will be provided)
- Include basic error handling and input validation

2. Technical Implementation

- Implement a modular, production-ready code structure
- Include unit tests for core functionality
- Provide a simple API (e.g. FastAPI) to interact with the system (POST /summarize, GET /health, POST /feedback (optional))

  Note on API Response: The response structure can range from a simple string with metadata to complex nested JSON objects - design the format that best serves your summarization approach and clinical users' needs.
- Include basic logging (including i/o tokens per request) and performance monitoring
- Use Docker for containerization

3. Innovation Component

Implement an innovative feature to make your summary stand out and improve the user experience. For example:

- Customizable summary formats based on different clinical roles

- Highlighting of critical findings
- Linking information from summary to a source document

Evaluation Criteria

1. Code Quality & Technical Architecture (50%)
    - Clean, well-structured code
    - Proper error handling
    - Effective use of design patterns
    - Quality of tests
    - System design choices
    - API design
    - Production readiness
2. Summary Quality & User Experience (30%)
    - Novel approach to solving clinical needs
    - User experience considerations
    - Creative features that add value
    - Documentation of future enhancement ideas
3. Speed, Latency, and Costs (20%)
    - Efficient use of API calls and tokens
    - Response time optimization

Clear consideration of cost-performance tradeoffs

Caching and optimization strategies

Deliverables

1. Git repository containing:
    - Source code
    - README.md with:
        - Setup instructions

        - Architecture overview
    - Requirements.txt or equivalent
    - Unit tests
    - Dockerfile
2. Brief documentation (max 1 page) including:
    - Link to the Git repo
    - Your approach to the problem
    - Technical decisions made
    - Challenges faced and solutions
    - Ideas for enhancing and scaling the system

Sample Data

You will be provided a few sample clinical notes for testing (./notes).

Note: do not include provided sample data in your Git repo.

Notes

- Feel free to use any open-source libraries and frameworks
- Document any assumptions made
- Focus on demonstrating both technical skills and creative problem-solving
- There are million ways to create a summary but there is no right way
- Code should be production-quality but doesn't need to be perfect
- You don’t have to implement everything that you can think of. Include comments about potential improvements or scaling considerations
- Simple UI for interacting with API is welcomed but NOT required