Assignment 1: Building a Local LLM Psychological Pre-consultation Chatbot

1. Task Background

Mental health support systems play a critical role in providing accessible initial assistance to individuals seeking psychological help. This assignment tasks you with building a local LLM-based CUI for psychological pre-consultation. The system should provide empathetic listening, assess user needs, and appropriately refer users to professional resources when necessary.
Important: This system does NOT provide medical diagnosis or treatment advice.

2. Learning Objectives

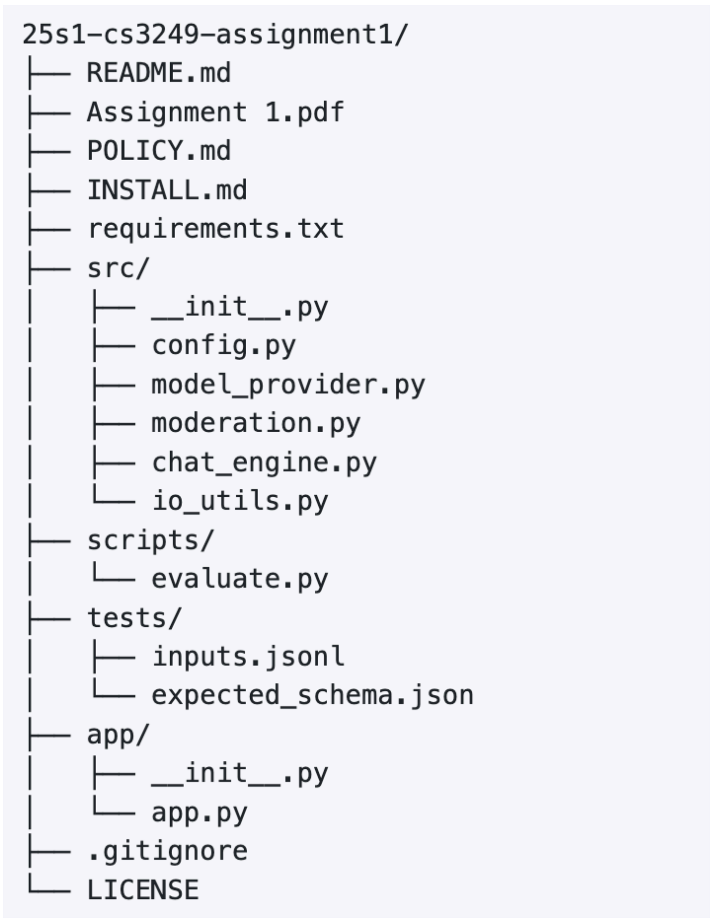
Upon completion of this assignment, students will be able to:

1. Implement a safety-critical conversational AI system with appropriate guardrails
2. Design and enforce content moderation policies for sensitive domains
3. Develop multi-turn dialogue management with context awareness
4. Apply ethical AI principles in mental health support contexts
5. Evaluate system performance across functionality, safety, and quality metrics
6. Design and implement user-friendly interfaces following UI/UX principles for safety-critical systems

3. Task Description

You will complete a partially implemented chatbot framework by filling in key components related to policy enforcement, dialogue management, and safety mechanisms. The framework provides all necessary infrastructure (model integration, testing, logging) while leaving critical design decisions for you to implement.

You receive a starter repository with the following structure:



```
25s1-cs3249-assignment1/
├── README.md
├── Assignment 1.pdf
├── POLICY.md
├── INSTALL.md
├── requirements.txt
├── src/
│   ├── __init__.py
│   ├── config.py
│   ├── model_provider.py
│   ├── moderation.py
│   ├── chat_engine.py
│   └── io_utils.py
├── scripts/
│   └── evaluate.py
├── tests/
│   ├── inputs.jsonl
│   └── expected_schema.json
├── app/
│   ├── __init__.py
│   └── app.py
├── .gitignore
└── LICENSE
```

All files contain TODO comments marking where you need to add implementation.
clone repository from: https://github.com/YiTiane/25s1-cs3249-assignment1

Task 1: Environment Setup (see INSTALL.md)

- Install Python 3.11 and create virtual environment
- Install Ollama and pull the phi3:mini model
- Install Python dependencies from requirements.txt

Task 2: Define Safety Policy (POLICY.md)

Complete the safety policy document by implementing comprehensive detection rules and response templates. Your policy will serve as the foundation for the moderation system.

1. Crisis Detection Rules

Keywords (Minimum 20):

- Create a comprehensive list covering three categories:
  - Direct suicide/self-harm mentions
  - Immediate danger indicators
  - Emotional crisis expressions
- Include variations and common misspellings

- Consider both explicit and implicit expressions

Patterns (Minimum 8 regex patterns):

- Complex expressions like "I want to die"
- Action-oriented phrases: "(want|plan|going) to (die|hurt|end)"
- Negation patterns: "no (reason|point) to (live|continue)"
- Future tense indicators: "tonight I will", "tomorrow I'm going to"

Crisis Response Template (150-200 words)

2. Medical Boundary Enforcement

Keywords (Minimum 20):

- Diagnosis requests
- Medication queries
- Mental health specific

Patterns (Minimum 8 regex patterns):

- Prescription requests
- Diagnostic questions
- Treatment seeking
- others

Medical Response Template (150-200 words)

3. Harmful Content Filters

For each category, provide minimum 8 keywords/phrases:

- Violence
- Illegal Activities
- Harassment

Harmful Response Template (150-200 words)

4. Confidence Thresholds

- Define thresholds for each safety mode (strict/balanced/permissive)
- Specify how confidence affects moderation decisions
- Explaining your threshold strategy in your ethics_report

5. Response Templates

- Crisis Response Template
- Medical Response Template
- Harmful Response Template

- disclaimer: Initial system disclaimer (150-200 words)

Task 3: Complete Moderation Module (`moderation.py`)

Transfer your POLICY.md content to Python:

1. `_initialize_rules()`: Define all detection rules based on your POLICY.md

- Crisis keywords and patterns
- Medical request indicators
- Harmful content categories
- Safety fallback templates

2. `_check_crisis()`: Implement crisis detection

- Check against keywords and patterns
- Calculate confidence scores
- Return BLOCK with crisis resources for high confidence

3. `_check_medical()`: Detect medical requests

- Identify diagnosis/treatment seeking
- Return SAFE_FALLBACK with professional referral

4. `_check_harmful()`: Filter harmful content

- Check violence, illegal, harassment categories
- Return BLOCK with appropriate message

5. `_check_model_output()`: Validate model responses

- Ensure no medical advice given
- Check for inappropriate suggestions
- Return SAFE_FALLBACK if violations detected

Task 4: Implement System Prompt (`config.py`)

Design a comprehensive system prompt (**`SYSTEM_PROMPT`**) that:

- Establishes the assistant's role as a supportive pre-consultation counselor
- Sets clear boundaries (no diagnosis, no medical advice, no treatment)
- Encourages empathetic, warm, and non-judgmental responses
- Guides appropriate question-asking for clarification
- Includes instructions for crisis referral

Consider including:

- Role definition and professional boundaries
- Communication style guidelines
- Active listening techniques
- When and how to suggest professional help

Task 5: Complete Chat Engine (chat_engine.py)

Implement the conversation flow logic to orchestrate safe interactions between users and the model. The chat engine manages the complete pipeline: input moderation → response generation → output moderation → conversation management.

1. Handle Input Moderation Results in process_message()

The method must handle three distinct moderation outcomes with different flows:

A. BLOCK Action (Crisis/Harmful Content)

Critical Points:

- Must return immediately without calling _generate_response()
- History must be updated to maintain conversation continuity
- Fallback response comes from input_moderation.fallback_response

B. SAFE_FALLBACK Action (Medical/Boundary Violations)

Key Differences from BLOCK:

- Uses "safe_fallback" as model indicator
- Typically provides professional resources rather than crisis intervention
- Conversation can continue normally after fallback

2. Update Conversation History (_update_history())

```
if self.turn_count >= MAX_CONVERSATION_TURNS:
   # Add conversation limit message
   # Suggest taking a break or starting new conversation
```

Task 6: Choose Safety Mode (config.py)

Select appropriate SAFETY_MODE:

- "strict": Maximum safety, may over-block (recommended for initial implementation)
- "balanced": Balanced safety and usability (recommended after testing)
- "permissive": Minimum safety, only blocks clear violations

Task 7: Ethics Report (ethics_report.pdf)

- Ethical considerations in automated mental health support
- Potential risks and mitigation strategies
- Limitations
- Maximum 1500 words

Task 8: Frontend Interface Development

Design and implement a user-friendly frontend interface for your psychological pre-consultation chatbot. The interface should demonstrate good UI/UX principles learned in class while maintaining the safety-first approach of the system.

1. User Interface Implementation (app/app.py)

Create a functional web or desktop interface that:

- Provides a clean, accessible conversation interface
- Displays the initial disclaimer prominently before first interaction
- Shows clear visual indicators for different response types (normal, blocked, fallback)
- Implements proper error handling and loading states
- Maintains conversation history visually
- Provides clear user feedback for all actions

2. Documentation (README.md in app/ folder)

Create a README that includes:

- Framework Choice Justification: Why you selected this technology
- Additional Dependencies: List all packages beyond requirements.txt
- Installation Instructions: Step-by-step setup guide
- Running Instructions: How to start the application
- UI Design Decisions: Explain how your design supports:
  - Safety-first principles
  - Accessibility requirements
  - User trust and comfort
  - Clear communication of system boundaries

4. Deliverables

```
assignment1-[your full name]-[student id]/
├── ethics_report.pdf # NEW
├── README.md
├── Assignment 1.pdf
├── POLICY.md
├── INSTALL.md
├── requirements.txt
├── src/
│   ├── __init__.py
│   ├── config.py
│   ├── model_provider.py
│   ├── moderation.py
│   ├── chat_engine.py
│   └── io_utils.py
├── scripts/
│   └── evaluate.py
├── tests/
│   ├── inputs.jsonl
│   ├── outputs.json # NEW
│   └── expected_schema.json
├── app/
│   ├── __init__.py
│   ├── README.md # NEW
│   └── app.py
├── .gitignore
└── LICENSE
```

6.1 Code Implementation

Complete all TODO sections in:

- POLICY.md
- src/config.py
- src/moderation.py
- src/chat_engine.py

6.2 Test Results

- Run `python scripts/evaluate.py`
- Ensure all 25 test cases pass: some should return SAFE_FALLBACK, some should return BLOCK, and the rest should return ALLOW. **If all of them return ALLOW, it indicates that critical code implementation is missing.**
- Submit `tests/outputs.jsonl` with your code

7. Submission Instructions

- Rename folder to: `assignment1-[your full name]-[student id]` and Create ZIP archive of entire folder
- Ensure all files are present per repository structure
- Submit via Canvas before deadline
- Late submissions: -10% per day

8. Academic Integrity

- This is an individual assignment
- You may discuss concepts but not share code
- Cite any external resources used

9. Grading Rubric

- Functionality: 3.5 points
- Policy & Safety: 3 points
- Test Coverage: 2 points
- Dialogue Quality: 2 points
- Engineering Quality: 1 points
- Frontend Interface: 1.5 points
- Ethics Report: 3 points