**Code Base Documentation: Memory Context Manipulation**

**System Architecture**

**Core Components**

1. app.py: Main application entry point
2. attack.py: Core attack simulation framework
3. context\_poisoning.py: Context poisoning challenge implementation
4. instruction\_overwriting.py: Instruction overwriting challenge
5. memory\_overflow.py: Memory overflow attack challenge
6. model.py: LSTM attack detection model implementation

**Key Modules**

**1. Attack Detection Model (model.py)**

Purpose: Detect conversation-based attacks using LSTM neural network

Components:

* AttackDetectionModel: LSTM neural network class
* AttackDetectionDataset: Custom dataset handler
* generate\_dataset: Creates synthetic training data
* train\_model: Model training function
* load\_model: Model loading function
* evaluate\_text: Evaluates text for attack patterns

Key Features:

* 10 input features analyzing message patterns
* 2 LSTM layers with dropout
* Binary classification output

**2. Attack Simulation Framework (attack.py)**

Purpose: Provide common functionality for all challenges

Key Functions:

* is\_repetitive\_message: Detects message spamming
* enforce\_rate\_limit: Prevents message flooding
* analyze\_conversation: Uses model to detect attacks
* generate\_system\_response: Core response generation
* check\_success: Determines challenge completion

**3. Challenge Implementations**

Context Poisoning (context\_poisoning.py)

Key Functions:

* context\_poisoning\_challenge: Main challenge interface
* enhanced\_generate\_system\_response: Custom response logic
* update\_memory\_context: Tracks conversation context
* display\_attack\_metrics: Shows security metrics

Instruction Overwriting (instruction\_overwriting.py)

Key Functions:

* instruction\_overwriting\_challenge: Main interface
* handle\_instruction\_overwriting: Response logic
* show\_attack\_tips: Displays technique suggestions

Memory Overflow (memory\_overflow.py)

Key Functions:

* memory\_overflow\_challenge: Main interface
* generate\_system\_response: Custom memory-focused logic
* update\_memory\_context: Tracks memory state
* check\_success: Memory-specific success check

**Data Flow**

1. User interacts via Streamlit UI
2. Messages processed by challenge module
3. Attack detection model analyzes conversation
4. System generates context-aware response
5. Metrics updated and displayed

Key Algorithms

**LSTM Attack Detection**

* Processes conversation features over time
* 10 input features including:
  + Message length patterns
  + Security-related keywords
  + Question/command ratios
  + Sentiment indicators

**Context Tracking**

* Maintains conversation state
* Tracks established facts and contested claims
* Adjusts security level based on interaction patterns

Configuration

**Model Parameters**

* Input size: 10 features
* Hidden size: 64
* Layers: 2
* Output: 1 (attack probability)

**Security Parameters**

* Initial security level: 5/5
* Security decay rate: Varies by challenge
* Attack thresholds: Challenge-specific

**Dependencies**

* Python 3.8+
* Streamlit
* PyTorch
* Pandas
* Matplotlib
* Scikit-learn

**Extension Points**

1. **Adding new challenge modules:**
   * Implement challenge-specific response logic
   * Add to attack.py's simulate\_attack function
   * Add navigation option in app.py
2. **Enhancing the detection model:**
   * Modify feature extraction in model.py
   * Adjust model architecture parameters
   * Add new training data patterns
3. **Customizing security parameters:**
   * Adjust security level thresholds
   * Modify decay/increase rates
   * Add new security metrics