**Goal:**

Create a basic framework that allows multiple models (Claude, Gemini 2.5 Pro, and ChatGPT) to interact, share data, and collaborate. This will involve sending messages between models, storing shared state data, and ensuring communication is consistent.

---

**Feature List for the Simple MCP (Model Control Protocol)**

1. **Message Passing Between Models**
    - **Objective**: Models should be able to send and receive messages to each other.
    - **Task**: Set up basic communication protocols between models, using text-based messages to simulate interaction.
2. **Shared Data Management via SQLite**
    - **Objective**: Keep track of all interactions and data shared between models.
    - **Task**: Use an SQLite database to store messages, states, and responses from each model.
3. **Model Coordination (Sequential Execution)**
    - **Objective**: The models should work in sequence—one model communicates, then the next processes the data, and so on.
    - **Task**: Implement a simple sequence of operations that models follow to avoid them running in parallel or clashing.
4. **Simple Interface for Interactions**
    - **Objective**: Provide a basic interface to control and initiate communication.
    - **Task**: Set up a simple Python interface (e.g., command-line input) that allows for triggering model interactions.
5. **Data Integrity (Basic Data Logging)**
    - **Objective**: Ensure no message gets lost during the exchange.
    - **Task**: Log messages and responses in the SQLite database to track the interaction history.

---

**Backwards Implementation Guide with Analogies**

**1. Final Goal (Models Collaborating and Interacting)**

**Analogy**: Imagine three friends trying to solve a puzzle together. They pass pieces of the puzzle back and forth to each other, but they need to take turns. They must keep track of which pieces have been placed where, and they store their progress in a shared notebook.

- **What needs to happen**:
    - Models exchange data and messages in a sequence.
    - All interactions are logged, and data is stored so that each model can "remember" what the others have done.
    - Each model "takes turns" based on a predefined order.

---

**2. Store Data (SQLite Database)**

- **Objective**: Ensure all interactions are recorded for later reference.

**Task Breakdown**:

- Set up an SQLite database with two main tables:
    1. **messages**: Store model communication and state.
    2. **model_interactions**: Track the sequence of interactions.

**Steps**:

- Install SQLite and set up the database schema.
- Create two tables:
    - messages: id, model_id, message_type, message_content, timestamp.
    - model_interactions: interaction_id, model1_id, model2_id, message_id, timestamp.

**Analogy**: It’s like creating a logbook where each message exchanged between friends is written down, so no one forgets what happened.

---

**3. Basic Communication (Message Passing)**

- **Objective**: Allow the models to send and receive messages.

**Task Breakdown**:

- Write Python functions for **send_message** and **receive_message**.
- Messages are simple text strings but can include basic instructions or data requests.

**Steps**:

- Implement a basic loop where each model sends a message, and the next receives and processes it.
- Store each message in the SQLite messages table.
- The models will just print the message received for simplicity.

**Analogy**: Like writing a letter, passing it to your friend, who reads it and writes a response. You keep track of all the letters in your notebook.

---

**4. Simple Sequential Coordination (Turn-based Execution)**

- **Objective**: Ensure the models work one at a time, not simultaneously.

**Task Breakdown**:

- Implement a function that allows models to take turns interacting with each other.
- Define a set of rules for how each model communicates. E.g., Model 1 always starts, Model 2 responds, then Model 3 does the next task.

**Steps**:

- Create a simple function to **execute_turn**. This function will handle one cycle of message passing (Model 1 sends message → Model 2 responds → Model 3 responds).
- Set a flag to indicate when a model is "ready" to send its message.

**Analogy**: Like playing a turn-based board game, where each player (model) takes one move in sequence.

---

**5. Interface for Control (User Input for Starting Communication)**

- **Objective**: Set up a basic interface that lets a user start the interaction.

**Task Breakdown**:

- Provide a command-line input that allows the user to trigger the interaction process.
- The user can control when each model sends its message and moves forward.

**Steps**:

- Implement a **start_communication** function where the user can initiate the sequence of messages.
- A simple prompt asks the user when to start and continue the interaction.

**Analogy**: Like someone starting the game by rolling the dice and then prompting the next player when it's their turn.

---

**Task Breakdown by Role**

**Who Does What:**

1. **Modeler/Lead Developer** (You):
    - Design the **message passing** system (implementing the basic logic).
    - Set up the **SQLite database** schema.
    - Build the **sequential execution logic** (turn-based communication).
    - Write and test the Python **functions** for handling model interactions.
2. **Assistant Developer** (if available):
    - Set up the **interface** for triggering communication (the command-line tool).
    - Implement **basic logging** of messages into the SQLite database.
    - Test the overall flow of interaction, ensuring no messages are lost and all data is recorded.
3. **Collaborating AI Models (Claude, Gemini 2.5 Pro, ChatGPT)**:
    - Each model will simply follow the predefined rules: sending messages to the next model in line and logging their responses into the system.

---

**Simplified Step-by-Step Flow:**

1. **Initialize SQLite Database**: Set up tables for tracking messages and interactions.
2. **Create Communication Functions**: Implement functions for sending/receiving messages between models.
3. **Implement Sequential Execution**: Ensure models take turns sending and receiving messages, with logging.
4. **Design Command-Line Interface**: Allow the user to initiate and control model interactions.
5. **Test & Refine**: Make sure all interactions are logged correctly, and the sequence works without errors.

---

**Summary**

This simple bridge allows the models to interact through basic message passing, using SQLite to store interactions, and a command-line interface to control the flow. The implementation is basic but lays the groundwork for future, more complex systems. It's like building a rudimentary communication pipeline between friends and tracking the conversation in a shared notebook.