**Logical Order of File Creation**

To ensure the custom GPT is built efficiently and logically, we’ll start with foundational files that define the structure and logic, then move toward result integration and user progress tracking. The logical order of file creation is as follows:

**Step 1: Foundational Test Logic**

1. **rules.json**:
   * Defines scoring rules, tie-resolution logic, and adaptive testing guidelines.
   * Serves as the backbone for dynamic test logic and execution.
2. **questions.json**:
   * Stores baseline questions, dynamic scenarios, and initial test flow.
   * Ensures balanced exposure to all communication styles.
3. **test\_logic.py**:
   * Implements test execution, including dynamic question delivery, scoring, and adaptive adjustments.
   * Leverages rules.json and questions.json for core functionality.

**Step 2: Results Framework**

1. **results.json**:
   * Defines the structure of test results, including style descriptions, Bible verses, and formatting templates.
   * Links style definitions to visual and textual elements.
2. **chart.py**:
   * Generates radar charts based on user scores.
   * Ensures visual representation matches the design framework.
3. **results\_gen.py**:
   * Compiles and formats results, integrating textual summaries, radar charts, and Bible verses.
   * Pulls data from results.json and chart.py.

**Step 3: Adaptive Testing and Progress Tracking**

1. **progress.py**:
   * Tracks user progress and stores historical data.
   * Dynamically adjusts future testing scenarios based on user growth.
2. **feedback.json**:
   * Stores user feedback on test clarity, relevance, and satisfaction.
   * Prepares data for quality assurance.

**Step 4: Quality Assurance**

1. **qa\_metrics.py**:
   * Processes feedback from feedback.json.
   * Analyzes confidence and reliability metrics to refine test quality.

**Order Summary**

1. **rules.json**: Define scoring and adaptive logic.
2. **questions.json**: Baseline questions and scenarios.
3. **test\_logic.py**: Core test execution.
4. **results.json**: Results formatting and descriptions.
5. **chart.py**: Radar chart generation.
6. **results\_gen.py**: Results compilation and integration.
7. **progress.py**: User progress tracking.
8. **feedback.json**: Collect user feedback.
9. **qa\_metrics.py**: Analyze feedback and quality metrics.