# **LLM Evaluation Tool: User and Developer Manual**

This manual provides comprehensive instructions for both end-users and developers on how to use, configure, and extend the LLM Evaluation Dashboard.

## **For End-Users: Getting Started**

This section is designed for users who want to quickly evaluate LLM outputs without diving into the code.

### **1. Running the Application**

The application is designed for easy startup.

* **Download & Unzip:** Download the provided .zip file (e.g., LLMEvalTool.zip) and unzip it to a convenient location on your computer (e.g., your Desktop or C:\LLMEvalTool).
* **Launch GUI:**
  + Navigate into the unzipped llm\_eval\_tool\_dist folder.
  + **Windows:** Double-click the run\_app.bat file.
  + **macOS/Linux:** Open a terminal, navigate to the llm\_eval\_tool\_dist folder, and run bash run\_app.sh.
* **First-Time Setup:** The first time you run the application, a command prompt/terminal window will open. It will automatically:
  + Check for or install Miniconda (a lightweight Conda distribution).
  + Create a dedicated Python environment (llm\_eval\_env) with all necessary libraries.
  + Download the required Sentence-Transformer model (all-MiniLM-L6-v2).
  + **This process can take several minutes.** Please be patient and do not close the window.
* **Application Launch:** Once setup is complete, your default web browser will automatically open to the LLM Evaluation Dashboard (usually at http://localhost:8501).
* **Keep Terminal Open:** Keep the command prompt/terminal window open while using the application. Closing it will shut down the application.

### **2. Using the Graphical User Interface (GUI)**

The GUI provides a guided workflow for evaluating your LLM outputs.

#### **2.1. Navigation and Instructions**

* **"Go to Instructions" Button (Sidebar):** At any point, you can click the "💡 Go to Instructions" button in the sidebar to return to the main instruction page. This will clear any loaded data.

#### **2.2. Step 1: Upload Your Data**

* **Upload Button:** Use the "Upload your dataset (CSV or JSON)" button in the sidebar.
* **Data Format:** Your input file (CSV or JSON) **must** contain the following columns:
  + query: The input prompt or question given to your LLM.
  + llm\_output: The response generated by your LLM.
  + reference\_answer: The human-written or ground-truth answer that the LLM's output will be compared against.
  + test\_description (Optional): A brief, human-readable description of the test case (e.g., "HR leave policy query", "Financial product question").
  + test\_config (Optional): A categorical label for the test case, useful for grouping or filtering (e.g., "HR\_Policy", "Financial\_Product\_Info", "Branch\_Operations", "Sentiment").
* **Example Structure:**  
  query,llm\_output,reference\_answer,test\_description,test\_config  
  "What is the capital of France?","Paris is the capital.","The capital of France is Paris, known for its Eiffel Tower.","Basic factual question","Geography"  
  "Summarize the plot of Hamlet.","Hamlet is about a prince seeking revenge.","Hamlet, Prince of Denmark, seeks revenge on his uncle Claudius for murdering his father and marrying his mother.","Literary summarization","Literature"  
  "What is the policy for annual leave accrual?","Employees get 15 days of leave.","Employees accrue 15 days of annual leave per year, with a maximum carry-over of 5 days.","HR Policy Lookup: Annual Leave","HR\_Policy"
* **Generate Mock Data:** If you don't have your own data, click the "✨ Generate Mock Data" button in the sidebar to create a sample CSV/JSON file. A toast notification will confirm generation; then, you can manually upload this newly generated file.
* **Data Preview:** After uploading, a "📊 Uploaded Data Preview" section will appear, showing the first few rows of your dataset.

#### **2.3. Step 2 & 3: Task Type and Metric Selection (End-User Mode)**

* **Simplified View:** In end-user mode (DEVELOPER\_MODE = False in config.py), these sections are streamlined:
  + **Task Type:** It will be fixed to "RAG FAQ".
  + **Metrics:** It will be fixed to "Semantic Similarity".
* **Purpose:** This simplifies the interface, focusing on the most common evaluation scenario for RAG FAQ.

#### **2.4. Step 4: Threshold Settings**

* **"Use Custom Thresholds" Checkbox:** By default (in end-user mode), this is checked, allowing you to easily adjust the pass/fail cutoff for "Semantic Similarity."
* **Adjusting Thresholds:** Use the number input field next to "Semantic Similarity Threshold" to set your desired score (e.g., 0.75). Outputs scoring equal to or above this threshold will be marked "Pass."

#### **2.5. Run Evaluation**

* **"🚀 Run Evaluation" Button:** Once your data is uploaded and settings are configured, click this prominent button in the main content area.
* **Progress:** A spinner will indicate that the evaluation is running. This may take time for large datasets.

#### **2.6. View Results**

After evaluation, the main dashboard will display:

* **Summary Report:**
  + **Metric Pass/Fail Rates:** Shows the percentage of test cases that passed, failed, or encountered errors for each metric.
  + **Average Metric Scores:** Displays the average score for each metric across all test cases.
* **Metric Insights and Performance Summary:**
  + **Overall Performance at a Glance:** Provides a quick interpretation (Excellent, Good, Review needed) of your LLM's performance for each metric based on its pass rate.
  + **Individual Metric Insights:** Explains what each metric measures and what its score signifies.
* **Detailed Results:**
  + A table showing the query, llm\_output, reference\_answer, test\_description, test\_config, and the calculated score and pass/fail status for each selected metric for every test case.
  + **Color Coding:**
    - **Scores:** Cells are colored with a gradient from light red (low score) to light green (high score), with dark text for readability.
    - **Pass/Fail:** "Pass" cells are light green, "Fail" cells are light red, and "Error" cells are light yellow.
  + **Hidden Columns:** Columns like ref\_facts and ref\_key\_points are hidden by default for a cleaner view.
* **Download Results:** Click the "Download Results as CSV" button to save the full evaluation results table to your computer.

### **3. Troubleshooting**

* **"Error loading metric models" / App doesn't start:**
  + Ensure your internet connection is stable, especially on the first run, as models need to be downloaded.
  + **Crucial:** Close the command prompt/terminal window completely and restart the run\_app.bat (or run\_app.sh) script. This clears Streamlit's cache and forces a fresh start.
* **"Missing required columns" error:**
  + Check your input CSV/JSON file to ensure it has at least query, llm\_output, and reference\_answer columns.
  + Verify column names are spelled correctly (case-sensitive).
* **App stuck on "Uploaded Data Preview" after clicking "Go to Instructions":**
  + This issue should be resolved with the latest updates. If it persists, ensure you have the absolute latest streamlit\_app.py and llm\_eval\_package/ui/sidebar\_view.py files. A full restart is always recommended.
* **"Mock data generated" message disappears too quickly:**
  + The message now persists for 3 seconds. Streamlit's st.toast is designed to be transient. Automatic upload of mock data directly into the file uploader is not supported by Streamlit for security reasons.

## **For Developers: Customization and Extension**

This section is for developers who want to understand, modify, or extend the LLM Evaluation Tool.

### **1. Project Structure (llm\_eval\_tool\_dist/)**

The application is organized into a Python package llm\_eval\_package for modularity:

* streamlit\_app.py: The main entry point for the Streamlit GUI.
* main.py: The entry point for the command-line interface (CLI).
* environment.yml: Defines the Conda environment dependencies.
* llm\_eval\_package/: The core Python package.
  + config.py: Centralized configuration for the entire application (metrics, thresholds, UI toggles, model paths).
  + data/: Handles data loading and mock data generation.
    - loader.py: Responsible for reading input data files.
    - generator.py: Generates synthetic evaluation data.
  + core/: Contains the main evaluation logic.
    - engine.py: The Evaluator class, which orchestrates metric computation.
    - reporting.py: The Reporter class, for generating summary reports.
  + ui/: Contains all Streamlit UI components (views).
    - data\_view.py: Displays the uploaded data preview.
    - results\_view.py: Renders the evaluation results table and summaries.
    - sidebar\_view.py: Manages the sidebar controls (upload, metric selection, thresholds).
    - tutorial\_view.py: Displays the "How to Use" instructions.
  + metrics/: Contains individual metric implementations.
    - base.py: BaseMetric abstract class. All new metrics must inherit from this.
    - completeness.py, conciseness.py, fluency\_similarity.py, safety.py, trust\_factuality.py: Concrete metric implementations.
  + tasks/: Defines task types and their associated properties.
    - registry.py: Maps task types to relevant metrics and primary columns.
  + utils.py: General utility functions, including ModelDownloader.
* data/: External folder for input/output data files.
* models/: External folder for downloaded Sentence-Transformer models.

### **2. Conda Environment Setup**

The environment.yml file defines the exact Conda environment.

name: llm\_eval\_env  
channels:  
 - defaults  
 - conda-forge  
dependencies:  
 - python=3.9 # Or your specific Python version  
 - pip  
 - pandas  
 - numpy  
 - streamlit  
 - tqdm  
 - scikit-learn  
 - pip:  
 - sentence-transformers

To set up the environment manually (e.g., for development):

conda env create -f environment.yml  
conda activate llm\_eval\_env

### **3. Running the Application (Development)**

* **GUI:** From the project root: streamlit run streamlit\_app.py
* **CLI:** From the project root: python main.py --help (to see options)
  + Example: python main.py --input\_file data/llm\_eval\_mock\_data\_generated.csv --output\_file cli\_results.csv --metrics "Semantic Similarity,Completeness"

### **4. Customization and Extension**

#### **4.1. Feature Toggles (llm\_eval\_package/config.py)**

* **DEVELOPER\_MODE**: Set to True for full control over task and metric selection in the GUI. Set to False to simplify the UI for end-users (fixed RAG FAQ task, fixed Semantic Similarity metric).
* ENABLE\_TASK\_SELECTION: Controls visibility of task type selection.
* ENABLE\_METRIC\_SELECTION: Controls visibility of metric selection.

#### **4.2. Adding New Metrics**

1. **Create a New Metric File:** In llm\_eval\_package/metrics/, create a new Python file (e.g., my\_new\_metric.py).
2. **Inherit BaseMetric:**  
   from llm\_eval\_package.metrics.base import BaseMetric  
     
   class MyNewMetric(BaseMetric):  
    def \_\_init\_\_(self):  
    super().\_\_init\_\_("My New Metric Name") # Use a unique display name  
     
    def compute(self, llm\_output: str, reference\_answer: str = None, query: str = None, \*\*kwargs) -> float:  
    # Implement your metric's logic here.  
    # Return a float score (e.g., 0.0 to 1.0).  
    # Access kwargs for any custom data (e.g., sensitive\_keywords).  
    return 0.5 # Placeholder score  
     
    def get\_score\_description(self, score: float) -> str:  
    # Provide a human-readable description for the score.  
    return "Description of what this score means."
3. **Register the Metric:**
   * In llm\_eval\_package/config.py, add your new metric to AVAILABLE\_METRICS:  
     AVAILABLE\_METRICS = {  
      # ... existing metrics ...  
      "My New Metric Name": "MyNewMetric",  
     }
   * In llm\_eval\_package/core/engine.py, import your new metric class:  
     from llm\_eval\_package.metrics.my\_new\_metric import MyNewMetric
   * In llm\_eval\_package/core/engine.py, initialize your new metric in \_get\_cached\_metric\_instances\_internal():  
     metrics\_instances["My New Metric Name"] = MyNewMetric()
4. **Update Task Mappings (Optional):** In llm\_eval\_package/tasks/registry.py, add your new metric to TASK\_METRICS for relevant task types if you want it preselected.
5. **Add Insight (Optional):** In llm\_eval\_package/config.py, add an entry to INTERPRETATION\_CONFIG for your new metric's insights.

#### **4.3. Adding New Task Types**

1. **Define New Task Type:** In llm\_eval\_package/tasks/registry.py, add your new task type constant.
2. **Map Display Name:** In TASK\_TYPE\_MAPPING.
3. **Define Metrics:** In TASK\_METRICS, specify which metrics are relevant for your new task type.
4. **Define Primary Columns:** In PRIMARY\_REFERENCE\_COLUMNS and PRIMARY\_PREDICTION\_COLUMNS.

#### **4.4. Model Management**

* **llm\_eval\_package/utils.py (ModelDownloader):** This class handles downloading Sentence-Transformer models. If you need to use different models, you can modify SENTENCE\_BERT\_MODEL in llm\_eval\_package/config.py.
* **Pre-downloading:** For faster first-time startup, you can manually run the ModelDownloader script (as described in Part 3 of the main response) to download models before distributing the application.

#### **4.5. UI Customization**

* **llm\_eval\_package/ui/ files:** Modify these files to change the layout, add/remove widgets, or adjust styling.
* **Styling:** Streamlit allows some styling via st.markdown with unsafe\_allow\_html=True for custom CSS, as seen in streamlit\_app.py for headers and buttons, and in results\_view.py for table cell coloring.

#### **4.6. Adding Guided Data Entry (Advanced)**

* As discussed, this feature was not in the original code. To add it, you would:
  1. Create llm\_eval\_package/ui/manual\_data\_entry\_view.py.
  2. Implement Streamlit input widgets (st.text\_area, st.text\_input) to collect query, llm\_output, reference\_answer, test\_description, test\_config for one or more rows.
  3. Use st.button to "Add Row" to a st.session\_state list of dictionaries.
  4. Provide "Download as CSV/JSON" and "Clear Data" buttons for the manually entered data.
  5. Integrate this view into streamlit\_app.py using st.tabs or st.radio to switch between "Upload File" and "Manual Entry."
  6. Ensure the data from manual entry is correctly formatted into a pandas.DataFrame and passed to the DataLoader or directly to the Evaluator.

This manual provides a solid foundation for understanding and working with your LLM Evaluation Tool.