# **LLM Evaluation Tool: Detailed Deployment & Usage Guide**

This document provides comprehensive instructions for deploying, using, and understanding the LLM Evaluation Tool, catering to both end-users and developers.

## **1. Project Overview**

The LLM Evaluation Tool is a Python application designed to help evaluate Large Language Model (LLM) outputs against reference answers using various metrics. It provides both a user-friendly Streamlit Graphical User Interface (GUI) and a Command-Line Interface (CLI) for automated evaluations.

### **1.1. Key Features**

* **GUI:** Interactive Streamlit dashboard for data upload, metric selection, threshold adjustment, and results visualization.
* **CLI:** Command-line tool for batch evaluations, ideal for automation and integration into CI/CD pipelines.
* **Modular Design:** Codebase organized into a Python package (llm\_eval\_package) for easy maintenance and extension.
* **Extensible Metrics:** Easily add new evaluation metrics by inheriting from a BaseMetric class.
* **Configurable Tasks:** Define different LLM task types (e.g., RAG, Summarization) and associate specific metrics with them.
* **Developer/End-User Modes:** Toggle features for simplified end-user experience or full developer control.
* **Local Model Management:** Automatically downloads and uses Sentence-Transformer models locally.

### **1.2. Project Structure**

The project is organized into a main distribution folder (llm\_eval\_tool\_dist) with a core Python package (llm\_eval\_package):

llm\_eval\_tool\_dist/ # This is the folder you will zip and distribute  
├── streamlit\_app.py # Main Streamlit GUI entry point  
├── main.py # CLI entry point  
├── environment.yml # Conda environment definition  
├── run\_app.bat # Windows batch file to install/run GUI  
├── run\_cli.bat # Windows batch file to install/run CLI  
├── llm\_eval\_package/ # The core Python package  
│ ├── \_\_init\_\_.py  
│ ├── config.py # Centralized application configurations  
│ ├── data/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── loader.py # Handles data loading  
│ │ └── generator.py # Generates mock data  
│ ├── core/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── engine.py # The main evaluation engine (Evaluator)  
│ │ └── reporting.py # Handles report generation (Reporter)  
│ ├── ui/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── data\_view.py # Data preview UI  
│ │ ├── results\_view.py # Results display UI  
│ │ ├── sidebar\_view.py # Sidebar controls UI  
│ │ └── tutorial\_view.py # Tutorial/Instructions UI  
│ ├── metrics/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── base.py # Base metric class  
│ │ ├── completeness.py  
│ │ ├── conciseness.py  
│ │ ├── fluency\_similarity.py  
│ │ ├── safety.py  
│ │ └── trust\_factuality.py  
│ ├── tasks/  
│ │ ├── \_\_init\_\_.py  
│ │ └── registry.py # Task definitions and metric mappings  
│ └── utils.py # General utility functions (e.g., ModelDownloader)  
├── data/ # External directory for generated/uploaded data files  
│ └── llm\_eval\_mock\_data\_generated.csv  
│ └── ...  
└── models/ # External directory for downloaded Sentence-BERT models  
 └── all-MiniLM-L6-v2/  
 └── ... (model files)

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

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

### **2.1. System Requirements**

* **Operating System:** Windows 10/11, macOS, or Linux.
* **Internet Connection:** Required for the initial setup to download Miniconda and the Sentence-Transformer model.
* **Disk Space:** Approximately 1-2 GB for the Conda environment and model files.

### **2.2. Running the Application (GUI)**

The application is designed for easy startup using a batch file.

1. **Download & Unzip:**
   * Download the provided .zip file (e.g., LLMEvalTool.zip) from your distribution source.
   * Unzip the file to a convenient location on your computer (e.g., C:\LLMEvalTool on Windows, or your Desktop on macOS/Linux). This will create a folder named llm\_eval\_tool\_dist.
2. **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.
3. **First-Time Setup (Important!):**
   * 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 (5-15 minutes depending on internet speed and system performance).** Please be patient and **do not close the window** while it's running.
4. **Application Launch:** Once setup is complete, your default web browser will automatically open to the LLM Evaluation Dashboard (usually at http://localhost:8501).
5. **Keep Terminal Open:** Keep the command prompt/terminal window open while using the application. Closing it will shut down the application.

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

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

#### **2.3.1. 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 and evaluation results, providing a fresh start.

#### **2.3.2. 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\_FAQ", "Financial\_Product\_Info", "Branch\_Operations\_Info", "Sentiment\_Analysis").
* **Example Structure (CSV):**  
  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"  
  "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: Annual Leave","HR\_Policy\_FAQ"  
  "Explain compound interest.","It's interest on interest.","Compound interest is interest calculated on the initial principal and accumulated interest.","Finance Concept: Compound Interest","Financial\_Product\_Info"  
  "Operating hours of main branch?","9 AM to 5 PM weekdays.","The main branch operates 9:00 AM-5:00 PM Mon-Fri, closed weekends.","Branch Info: Main Branch Hours","Branch\_Operations\_Info"
* **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 (llm\_eval\_mock\_data\_generated.csv/.json) in your data/ folder. A toast notification will appear. After generation, you can then manually upload this newly created file using the "Upload your dataset" button.
* **Data Preview:** After uploading, a "📊 Uploaded Data Preview" section will appear, showing the first few rows of your dataset.

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

* **Simplified View:** By default, for end-users, 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. Developers can enable full control (see Section 3.1).

#### **2.3.4. 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."

#### **2.3.5. 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.

#### **2.3.6. 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.

### **2.4. Using the Command-Line Interface (CLI)**

The CLI allows for automated evaluation without the GUI.

1. **Launch CLI Setup Script:**
   * Navigate into the llm\_eval\_tool\_dist folder.
   * **Windows:** Double-click the run\_cli.bat file.
   * **macOS/Linux:** Open a terminal, navigate to the llm\_eval\_tool\_dist folder, and run bash run\_cli.sh.
   * This script will perform the initial Conda environment setup (similar to run\_app.bat), then provide instructions on how to use the main.py script.
2. **Activate Environment:** Open a **new** command prompt/terminal window and activate your Conda environment:  
   conda activate llm\_eval\_env
3. **Run main.py:** Navigate to your llm\_eval\_tool\_dist directory if you're not already there. Then, use python main.py with arguments:  
   # Basic usage: Evaluate a CSV with default RAG FAQ metrics, save to CSV  
   python main.py --input\_file data/llm\_eval\_mock\_data\_generated.csv --output\_file my\_cli\_results.csv  
     
   # Evaluate a JSON file with specific metrics and custom thresholds, save as JSON  
   python main.py --input\_file data/llm\_eval\_mock\_data\_generated.json \  
    --output\_file my\_cli\_results.json \  
    --metrics "Semantic Similarity,Completeness" \  
    --custom\_thresholds "Semantic Similarity=0.8,Completeness=0.7" \  
    --report\_format json  
     
   # Evaluate with the Safety metric and custom sensitive keywords  
   python main.py --input\_file data/llm\_eval\_mock\_data\_generated.csv \  
    --output\_file safety\_results.csv \  
    --metrics "Semantic Similarity,Safety" \  
    --sensitive\_keywords "profanity,hate speech,unsafe\_word"  
     
   # Get help for all available arguments  
   python main.py --help  
     
   **Key Arguments:**
   * --input\_file <path> (Required): Path to your input data (CSV or JSON).
   * --output\_file <path> (Default: llm\_evaluation\_results.csv): Path to save the output.
   * --task\_type <type> (Default: rag\_faq): The task type (e.g., rag\_faq, summarization, classification).
   * --metrics <list>: Comma-separated list of metric names (e.g., "Semantic Similarity,Conciseness"). Overrides defaults.
   * --custom\_thresholds <dict\_string>: Comma-separated MetricName=Value pairs (e.g., "Semantic Similarity=0.8").
   * --sensitive\_keywords <list>: Comma-separated keywords for the Safety metric.
   * --report\_format <format> (Default: csv): csv or json.

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

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

### **3.1. 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

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

* **GUI:** From the project root (llm\_eval\_tool\_dist):  
  streamlit run streamlit\_app.py
* **CLI:** From the project root (llm\_eval\_tool\_dist):  
  python main.py --help  
    
  (See Section 2.4 for example CLI commands).

### **3.3. Customization and Extension Points**

#### **3.3.1. 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 in GUI.
* ENABLE\_METRIC\_SELECTION: Controls visibility of metric selection in GUI.

#### **3.3.2. 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:**  
   # llm\_eval\_package/metrics/my\_new\_metric.py  
   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.  
    # Access kwargs for any custom data (e.g., sensitive\_keywords).  
    return 0.5 # Placeholder score (0.0 to 1.0)  
     
    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.

#### **3.3.3. Adding New Task Types**

1. **Define New Task Type:** In llm\_eval\_package/tasks/registry.py, add your new task type constant (e.g., MY\_NEW\_TASK = "my\_new\_task").
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.

#### **3.3.4. Model Management (llm\_eval\_package/utils.py)**

The ModelDownloader class in llm\_eval\_package/utils.py handles downloading Sentence-Transformer models. If you need to use different models, you can modify SENTENCE\_BERT\_MODEL in llm\_eval\_package/config.py.

To manually pre-download models (e.g., before zipping for distribution):

# Ensure your Conda environment is activated: conda activate llm\_eval\_env  
# Navigate to your project's root directory (llm\_eval\_tool\_dist)  
python -c "from llm\_eval\_package.utils import ModelDownloader; from llm\_eval\_package.config import SENTENCE\_BERT\_MODEL, MODEL\_DIR; md = ModelDownloader(); md.download\_and\_save\_model(SENTENCE\_BERT\_MODEL, MODEL\_DIR)"

#### **3.3.5. 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. Alternative Deployment Solutions**

While Conda packing with batch/shell scripts is robust, other solutions offer different trade-offs in terms of ease-of-use for end-users, portability, and complexity for developers.

### **4.1. PyInstaller (Creating a Single Executable)**

**Concept:** PyInstaller bundles a Python application and all its dependencies into a single executable file (e.g., .exe for Windows, .app for macOS). End-users simply double-click the executable, with no Python or Conda installation required.

**Pros:**

* **Extremely Easy for End-Users:** True "double-click and run" experience.
* **No Dependencies:** Users don't need Python, Conda, or any other runtime installed.
* **Self-Contained:** All libraries are bundled.

**Cons:**

* **Platform-Specific:** You need to build a separate executable for each operating system (Windows, macOS, Linux).
* **Large File Size:** Executables can be very large (hundreds of MBs) as they include a full Python interpreter and all libraries.
* **Complexity for Streamlit:** Streamlit applications involve a web server (uvicorn or tornado) and a browser. PyInstaller needs to correctly bundle the server and ensure the browser can open to the correct local port. This can be tricky and often requires custom PyInstaller hooks or spec files.
* **Antivirus Flags:** Sometimes, executables built with PyInstaller can be flagged as suspicious by antivirus software due to their bundled nature.
* **Model Files:** Large models (like Sentence-Transformer) often need to be bundled separately or downloaded on first run, as including them directly can make the executable enormous.

**General Steps (High-Level):**

1. **Install PyInstaller:** pip install pyinstaller (in your Conda environment).
2. **Create a spec file:** pyinstaller --name "LLMEvalTool" streamlit\_app.py (this generates a .spec file).
3. **Edit the spec file:** This is where the complexity lies for Streamlit. You'd need to:
   * Include hidden imports for Streamlit's internal dependencies (e.g., uvicorn, watchdog, tornado).
   * Ensure data files (like environment.yml, your llm\_eval\_package files, data/ folder, models/ folder) are correctly copied into the bundle using datas and binaries in the spec file.
   * Potentially add hook-streamlit.py to handle Streamlit's specific requirements.
4. **Build the Executable:** pyinstaller LLMEvalTool.spec

**Recommendation:** PyInstaller is the *easiest for end-users* but the *most complex for developers* to set up correctly for a Streamlit application, especially with external models. If your primary target is Windows users and you want the simplest possible end-user experience, it's worth exploring, but be prepared for significant debugging during the build process.

### **4.2. Docker (Containerization)**

**Concept:** Docker packages your application and its entire environment (including Python, all libraries, and even the operating system dependencies) into a lightweight, portable container. Users need Docker Desktop installed.

**Pros:**

* **Ultimate Portability:** "Build once, run anywhere." The container runs identically on any system with Docker.
* **Environment Isolation:** Complete isolation from the host system.
* **Reproducibility:** Guarantees the exact same runtime environment.
* **Scalability:** Easy to deploy in cloud environments or scale with container orchestration.

**Cons:**

* **Docker Installation Required:** End-users need to install Docker Desktop, which can be a barrier for non-technical users.
* **Learning Curve:** Users need basic Docker commands (docker build, docker run).
* **Image Size:** Container images can be large, especially with large models bundled inside.

**General Steps (High-Level):**

1. **Create a Dockerfile:**  
   # Dockerfile  
   FROM python:3.9-slim-buster # Or a specific Python version and base OS  
   WORKDIR /app  
   COPY environment.yml .  
   RUN pip install --no-cache-dir conda-build # Install conda-build if needed for env creation  
   RUN conda env create -f environment.yml # Create conda env  
   ENV PATH="/opt/conda/envs/llm\_eval\_env/bin:$PATH" # Add env to PATH  
   COPY . . # Copy your entire project  
   # Download models during build (or on first run if not bundled)  
   RUN python -c "from llm\_eval\_package.utils import ModelDownloader; from llm\_eval\_package.config import SENTENCE\_BERT\_MODEL, MODEL\_DIR; md = ModelDownloader(); md.download\_and\_save\_model(SENTENCE\_BERT\_MODEL, MODEL\_DIR)"  
   EXPOSE 8501  
   CMD ["streamlit", "run", "streamlit\_app.py", "--server.port", "8501", "--server.enableCORS", "false", "--server.enableXsrfProtection", "false"]
2. **Build the Image:** docker build -t llm-eval-tool .
3. **Run the Container:** docker run -p 8501:8501 llm-eval-tool

**Recommendation:** Docker is excellent for robust, reproducible deployments, especially if your users are comfortable with developer tools or if you plan to deploy to a server. For truly non-technical users, installing Docker Desktop might be more challenging than just running a batch file.

## **5. Troubleshooting (Common Issues)**

* **"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). Use the "Example CSV/JSON structure" in the GUI's instructions as a reference.
* **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.
* **ModuleNotFoundError: No module named 'llm\_eval\_package':**
  + This usually means streamlit\_app.py (or main.py) is not in the project's root directory, or the llm\_eval\_package folder is not directly inside the root. Ensure your directory structure matches the "Project Structure for Distribution" section.
  + Ensure all \_\_init\_\_.py files are present in all package and sub-package directories.
* **Conda environment creation/activation issues:**
  + Ensure you have enough disk space.
  + Check your internet connection.
  + If on Windows, ensure Miniconda was installed with "Add to PATH" selected, or restart your command prompt after installation.

This detailed guide should provide all the information needed for both deployment and usage of your LLM Evaluation Tool.