AI Evaluation Tool v1.0: Technical Documentation

CeRAI, IIT Madras

1 Usage & Workflows

This section describes the **detailed operational workflow of the AI Evaluation Tool**, outlining the complete lifecycle from test data ingestion to evaluation report generation.

1.1 AI Evaluation Tool Workflow

The AI Evaluation Tool follows a structured and deterministic workflow designed to ensure reproducibility, modular execution, and clear separation of responsibilities across stages.

1.1.1 Complete Workflow Overview

Figure 1 illustrates the **directed acyclic graph (DAG)** that represents the complete workflow of the AI Evaluation Tool. Each node denotes a distinct operational stage, and directed edges indicate strict execution dependencies.

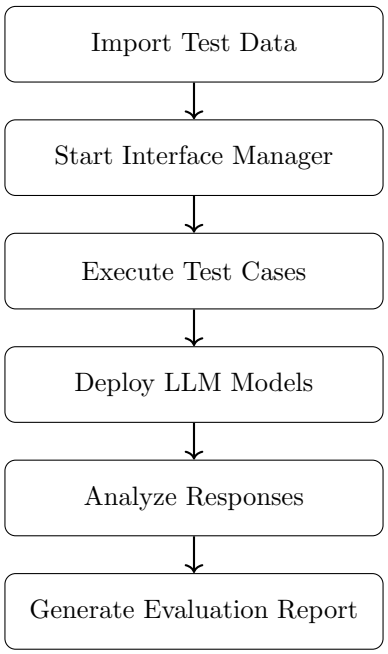

Figure 1: AI Evaluation Tool workflow

1.2 Step 1: Import Test Data into Database

Objective: This step initializes the evaluation environment by importing structured test cases and evaluation metadata from JSON sources into a persistent storage backend powered by `MariaDB` or `SQLite`. All subsequent test execution, analysis, and reporting stages depend on

the successful completion of this step.

Prerequisites:

- A supported database instance (MariaDB or SQLite) is configured and running
- A valid importer configuration file is available at `src/app/importer/config.json` (or an equivalent custom configuration file)
- Test case definitions are present in `data/DataPoints.json`

Command Execution:

```
python3 src/app/importer/main.py --config "src/app/importer/config.json
   " --orm-debug
```

Expected Console Output:

```
[INFO] Importing strategies...
[INFO] Importing test plans and metrics...
...
...
[INFO] Data import completed successfully.
```

Command-Line Arguments:

- `--config`: Path to the importer configuration file containing database connection and runtime settings (default: `config.json`)
- `--orm-debug`: Enables ORM-level debug logging for database operations when specified

Verify that test plans, metrics, and test cases have been successfully inserted into the database

1.3 Step 2: Start Interface Manager API Service

Objective: This step starts the Interface Manager API service, which acts as the communication layer between the AI Evaluation Tool and target platforms such as REST APIs, WhatsApp interfaces, and web applications.

Prerequisites:

- The Interface Manager source code is accessible in the local environment
- No other process is bound to port `8000` (or the configured service port)

Service Startup Command:

```
cd src/app/interface_manager
python main.py
```

Expected Console Output:

```
[INFO] Will watch for changes in these directories: ['.../src/app/
   interface_manager']
[INFO] Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
[INFO] Started reloader process [2060848] using WatchFiles
[INFO] Started server process [2060868]
[INFO] Waiting for application startup.
[INFO] Application startup complete.
```

Verification and Validation:

- Confirm that the service starts successfully without error messages
- Verify that the API is listening on the configured port (`8000` by default)

1.4 Step 3: Configure and Execute Test Cases

Objective: This step executes evaluation test cases against the configured target application through the Interface Manager and records the generated responses for downstream analysis.

Prerequisites:

- Interface Manager API service is running (refer Step 2)
- Executor configuration file available at `src/app/testcase_executor/config.json`
- Target application is accessible and operational
- XPath selectors and authentication credentials are configured for web or WhatsApp-based targets

1.4.1 Retrieve Available Test Plans

```
cd src/app/testcase_executor
python main.py --config "config.json" --get-plans
```

Sample Output:

Plan ID	Name	Description
1	Responsible_AI	
2	Conversational_Quality	
3	Guardrails_and_Safety	
4	Language_Support	
5	Task_Performance_Metrics	
6	Performance_and_Scalability	
7	Privacy_and_Safety	

1.4.2 Retrieve Available Evaluation Metrics

```
python main.py --config "config.json" --get-metrics
```

Sample Output:

Metric ID	Name	Description
1	Inclusivity	
2	Transparency	
3	Explainability	
4	Cultural_Sensitivity	
5	Fairness	

Metric ID	Name	Description
6	Robustness	
7	Truthfulness	
8	Bias_Assessment	
9	Topic_Drift_Rate	
10	Dialogue_Coherence	
11	Grammatical_Correctness_Rate	
12	Lexical_Diversity	
13	Relevance_and_Information	
14	Logical_flow_and_Discourse_Structure	
15	BLEU	
16	ROUGE	
17	METEOR	
18	NER_performance_for_the_relevant_entities	
19	Intent_Recognition	
20	Fluency_Score	
21	Response_out_of_scope	
22	Toxicity_Level	
23	Rejection_Rate	
24	Efficacy_of_content_filtering	
25	Detection_of_Harmful_Inputs	
26	Inappropriate_Content_Detection_Rate	
27	Hallucination_Rate	
28	Bias_Detection	
29	Robustness_against_Adversarial_Attacks	
30	Language_Coverage	
31	Ability_to_handle_multiple_Indian_languages_in_one_context	
32	Transliterated_Language_Handling	
33	Accuracy_per_Language	
34	Fluency_in_Indian_Languages	
35	Task_Completion_Rate	
36	Accuracy	
37	Turn_Around_Time	
38	Uptime	
39	Error_Rate	
40	Mean_Time_Between_Failures	
41	Transactions_per_minute	
42	Message_Volume_Handling	
43	Misuse	
44	Jailbreak	
45	Exaggerated_Safety	
46	Privacy_Awareness_Query	
47	Privacy_Leakage	
48	Privacy_Confidence_Agreement	

1.4.3 Retrieve Existing Test Runs

Before executing new test cases, existing test runs can be inspected to determine whether an execution should be continued or a new run should be created.

Command:

```
python main.py --config "config.json" --get-runs
```

Sample Output:

Run ID	Name	Status
1	doodle-accepting-pascal-nibh	RUNNING

Usage Note: The retrieved run name can be reused with the `--run-continue` flag to resume an existing execution.

1.4.4 Retrieve Available Target Applications

The following command lists all target applications configured in the system, including APIs, WhatsApp-based interfaces, and web applications.

Command:

```
python main.py --config "config.json" --get-targets
```

Sample Output:

Target ID	Name	Type	Domain	URL
1	Gooey AI	WhatsApp	Agriculture	https://www.help.gooey.ai/f...
2	August AI	WhatsApp	Healthcare	https://wa.me/8738030604
3	Vaidya AI	WhatsApp	Healthcare	https://wa.me/8828808350
4	Jivi AI	WhatsApp	Healthcare	https://wa.me/8121839444
5	FarmSawa	WhatsApp	Healthcare	https://wa.me/+254704582362
6	CPGRAMS	WebApp	Government Services	https://cpgramsaichatbot.com/
7	OPENWEB-UI	WebApp	General	http://localhost:3000
8	Gemini-2.5-Flash	API	General	https://generativelanguage...
9	Gemma3n:e2b	API	General	http://localhost:11434
10	llama3.2:3b	API	General	http://localhost:11434

Usage Note: The selected target application determines the execution interface, domain constraints, and protocol used during test case execution.

1.4.5 Execute Test Cases

```
python main.py \
  --config "config.json" \
  --testplan-id <testplan-id> \
  --testcase-id <testcase-id> \
  --metric-id <metric-id> \
  --max-testcases <max-testcases> \
  --run-name <run-name> \
  --execute
```

Execution Constraint: The `--testplan-id` parameter is mandatory for all executions. Optionally, either a `--testcase-id` or a `--metric-id` may be provided to further scope the execution. At minimum, one of the following combinations must be specified: `testplan-id`, `testplan-id + testcase-id`, or `testplan-id + metric-id`.

Command-Line Arguments:

- `--config (-c)`: Path to the configuration file containing database connection and target application details
- `--get-config-template (-T)`: Outputs a template configuration file for reference
- `--get-plans (-P)`: Retrieves all available test plans
- `--get-metrics (-M)`: Retrieves all supported evaluation metrics
- `--get-testcases (-C)`: Retrieves test cases for a specific test plan or all test cases
- `--get-targets (-G)`: Retrieves all configured target applications
- `--get-runs (-N)`: Retrieves historical test execution runs
- `--testplan-id (-p)`: Specifies the test plan identifier to execute
- `--testcase-id (-t)`: Specifies a single test case identifier to execute
- `--metric-id (-m)`: Specifies the evaluation metric to apply
- `--max-testcases (-n)`: Maximum number of test cases to execute (default: 10)
- `--run-name (-r)`: Custom identifier assigned to the test execution run
- `--run-continue (-R)`: Continues execution of an existing run with the specified run name
- `--execute (-e)`: Triggers execution mode for test plans or test cases
- `--verbosity (-v)`: Controls logging verbosity level (0–5, default: 5)
- `--language-strict (-l)`: Enables strict language matching during test case selection
- `--domain-strict (-d)`: Enables strict domain matching during test case selection

1.4.6 Execution Examples

Basic Execution:

```
python main.py --config "config.json" --testplan-id 1 --execute
```

Custom Run Name:

```
python main.py --config "config.json" --testplan-id 1 --run-name "
   custom_run" --execute
```

Strict Filtering Enabled:

```
python main.py --config "config.json" --testplan-id 1 --language-strict
    --domain-strict --execute
```

Continue Existing Run:

```
python main.py --config "config.json" --testplan-id 1 --run-continue --
    run-name "previous_run_name" --execute
```

1.4.7 Expected Execution Output

```
[INFO] Starting the Testcase Executor...
[INFO] Test Plan: Performance_and_Scalability (ID: 6)
[INFO] Metric: Transactions_per_minute (ID: 41)
[INFO] Target: Vaidya AI (WhatsApp Web)
[INFO] Executing 1 test case...
[INFO] Run Name: Created new run with ID 10 and name 'group-chewy-swamp
    -iPhone'
[PROGRESS] 1/1 - Test Case P1201
[SUCCESS] Test execution completed in 1 minute 28 seconds
[INFO] Responses stored in database
[INFO] Run ID: 11
[SUCCESS] Execution of test plan completed successfully.
```

Important: The generated **Run Name** must be preserved, as it is required for response analysis and report generation in subsequent workflow steps.

Common Execution Issues:

- Connection timeouts indicate target application unavailability
- Selenium-related failures typically result from ChromeDriver and browser version mismatches
- XPath resolution errors require updates to `xpaths.json`

1.5 Step 4: Deploy LLM Models

Objective: This step provisions and launches all Large Language Models (LLMs) required for automated evaluation, including the default *LLM-as-Judge*. These models are used for response generation, safety analysis, translation, and qualitative judgment during the evaluation pipeline.

1.5.1 Required Models

The following models must be available prior to evaluation:

- `sarvamai/sarvam-2b-v0.5` — Text generation and classification
- `google/shieldgemma-2b` — Safety and harmful content detection
- `sarvamai/sarvam-translate` — Multilingual translation
- `qwen3:32b` — Default LLM-as-Judge

Note:

- Ollama uses a fixed default port `11434`
- All other services (e.g., Sarvam AI) use configurable ports and must be assigned free ports

1.5.2 Deployment Options

LLM serving can be performed using one of the following approaches.

A. Local Serving This configuration is recommended when all services are hosted on the same machine.

Start Sarvam AI Service:

```
cd src/app/sarvam_ai
python main.py --port <free-port-local>
```

Pull and Serve LLM-as-Judge:

```
ollama pull qwen3:32b
ollama serve
```

Note: The `ollama serve` process may start automatically and might not require manual execution.

B. Remote GPU Serving (Port Forwarding) This configuration is recommended when models are hosted on a remote GPU-enabled machine.

Start Services on Remote GPU Machine:

```
cd src/app/sarvam_ai
python main.py --port <free-port-gpu>
```

```
ollama pull qwen3:32b
```

Forward Remote Ports to Local Machine:

```
ssh gpu_machine_cred@machineIP \
  -L localhost:11434:localhost:<free-port-local> \
  -L localhost:<free-port-gpu>:localhost:<free-port-local>
```

Command-Line Arguments:

- `--port (-p)`: Port on which the FastAPI service is exposed (default: 16000)
- `--host (-H)`: Host address for the FastAPI service (default: 0.0.0.0)
- `--verbosity (-v)`: Logging verbosity level in the range 0–5 (default: 5)
- `--translator-model (-t)`: Translation model identifier (default: sarvamai/sarvam-translate)
- `--generator-model (-g)`: Text generation model identifier (default: sarvamai/sarvam-2b-v0.5)
- `--safety-model (-s)`: Safety evaluation model identifier (default: google/shieldgemma-2b)
- `--force-cpu`: Forces CPU execution for the translator model

1.5.3 Expected Service Output

```
[INFO] Loading models...
[INFO] Loading sarvamai/sarvam-2b-v0.5...
[INFO] Loading google/shieldgemma-2b...
[INFO] Loading sarvamai/sarvam-translate...
[INFO] All models loaded successfully
[INFO] Sarvam AI Service running on http://0.0.0.0:8000
[INFO] Ready to serve evaluation requests
```

1.5.4 Additional Supporting Models

In addition to the primary models, the following lightweight models are automatically downloaded and used for specialized evaluation strategies:

- `amedvedev/bert-tiny-cognitive-bias`
- `LibrAI/longformer-harmful-ro`
- `vectara/hallucination_evaluation_model`
- `thenlper/gte-small`
- `all-MiniLM-L6-v2`
- `nicholasKluge/ToxiGuardrail`
- `sentence-transformers/paraphrase-multilingual-mpnet-base-v2`
- `google/flan-t5-large`
- `holistic-ai/bias_classifier_albertv2`
- `Human-CentricAI/LLM-Refusal-Classifier`
- `cross-encoder/nli-deberta-base`

1.6 Step 6: Analyze Responses

Objective: This step evaluates the responses generated during test execution by applying a combination of rule-based heuristics and model-driven evaluation strategies. The resulting scores quantify model behavior across multiple quality, safety, and performance dimensions and are persisted for reporting.

Prerequisites:

- Test case execution has completed successfully (refer Step 3)
- A valid **Run Name** is available from execution logs or the database
- Sarvam AI service is running and fully initialized (refer Step 5.1)
- LLM-as-Judge model service is running (refer Step 5.2)
- All required models have completed loading (allow 2–3 minutes after service startup)

1.6.1 Retrieve Run Name from Database

If the run name is not available from execution logs, it can be retrieved directly from the database.

Testcase Executor:

```
python main.py --config "config.json" --get-runs
```

SQLite:

```
sqlite3 AIEvaluationData.db \
  "SELECT run_id, run_name, created_at FROM test_runs ORDER BY
     created_at DESC LIMIT 5;"
```

MariaDB:

```
mysql -u aiet_user -p aievaluationtool \
  -e "SELECT run_id, run_name, created_at FROM test_runs ORDER BY
     created_at DESC LIMIT 5;"
```

1.6.2 Execute Response Analysis

Command:

```
cd src/app/response_analyzer
python analyze.py --config "config.json" --run-name "doodle-accepting-
   pascal-nibh"
```

Command-Line Arguments:

- `--config`: Path to the response analyzer configuration file (default: `config.json`)
- `--get-config-template (-T)`: Outputs a template configuration file for reference
- `--verbosity (-v)`: Controls logging verbosity level (0–5, default: 5)
- `--run-name (-r)`: Name of the test execution run to be analyzed
- `--force (-f)`: Forces re-evaluation of runs that have already been analyzed

1.6.3 Expected Analysis Workflow

The response analysis pipeline proceeds through the following deterministic stages:

1. **Load Responses:** Retrieve all test responses associated with the specified run from the database
2. **Apply Strategies:** Execute rule-based checks and model-based evaluators (including LLM-as-Judge)
3. **Compute Metrics:** Aggregate and compute scores for each evaluation metric
4. **Store Results:** Persist computed scores and metadata back into the database

Operational Note: Each analysis run is idempotent by default. The `--force` flag must be explicitly provided to overwrite existing evaluation results for a given run.

1.7 Generate Evaluation Report

Objective: This step generates a consolidated evaluation report for a completed analysis run. The report aggregates metric-level scores across test plans and provides a structured summary of evaluation outcomes.

Prerequisites:

- Response analysis has been completed successfully (refer Step 6)
- A valid **Run Name** corresponding to the analyzed execution is available
- Evaluation results for the specified run exist in the database

1.7.1 Generate Report

Command:

```
cd src/app/response_analyzer
python report.py --config "path to config file" --run-name <run-name>
```

Report Generator Command-Line Arguments:

- `--config`: Path to the report generator configuration file (default: `config.json`)
- `--get-config-template (-T)`: Outputs a template configuration file for reference
- `--verbosity (-v)`: Controls logging verbosity level (0–5, default: 5)
- `--get-runs (-N)`: Retrieves all available test execution runs
- `--run-name (-r)`: Name of the analysis run for which the report is generated
- `--force (-f)`: Forces report generation even if a report already exists for the specified run

Operational Notes:

- The `--run-name` must match the run name used during the analysis stage
- Report generation is non-destructive by default; existing reports are preserved unless overridden with `--force`

1.7.2 Example Report Structure

The generated report presents evaluation scores in a tabular format grouped by test plan and metric.

Plan Name	Metric Name	Score
<Plan_Name>	<Metric_Name>	<Numeric_Score>
<Plan_Name>	<Metric_Name>	<Numeric_Score>
⋮	⋮	⋮