LLM Evaluation Pipeline for AI Tutor System

Comprehensive System Design & Process Documentation

Repository: `AmanPawar9/llm-eval-assignment`

November 13, 2025

Contents

1 Executive summary

This document describes the implementation and design of the *LLM Evaluation Pipeline for AI Tutor System*. The codebase implements a production-oriented evaluation service composed of:

- A FastAPI backend implementing dataset management, evaluation job creation, CSV export, and job-case querying.
- An RQ-based worker that performs evaluation tasks asynchronously, executes rubric-based judge calls, writes NDJSON audit logs, and persists structured results in a SQL database.
- A set of judge adapters (mock / Ollama / HuggingFace) for LLM-as-judge evaluation.
- A Streamlit demo frontend for launching evaluations and visualizing results.
- Optional Docker Compose infrastructure providing Redis, Postgres, MinIO, backend, and worker services.

The system is modular, scalable, and suitable for real deployments or research workflows evaluating educational LLMs.

2 Concrete project structure

`backend/app/main.py`	FastAPI application entrypoint, router wiring and middleware.
`backend/app/api/`	Routers for dataset APIs, evaluation APIs, engine selection and metrics routes.
`backend/app/orchestrator/`	Core orchestration logic: batching, validator calls, and enqueueing tasks into RQ.
`backend/app/workers/`	RQ worker execution path, NDJSON writing, storage of case results.
`backend/app/db/`	SQLAlchemy models and DB helpers. Defaults to SQLite when DATABASE_URL is not set.
`frontend/streamlit_app.py`	Streamlit frontend for job creation, monitoring and visual summaries.
`infra/docker-compose.yml`	Full-stack compose for Redis, Postgres, MinIO, backend, and worker.
`config.yaml` / `.env.example`	Runtime configuration and environment variables.

3 Dataset and input format

The evaluation pipeline expects datasets in structured JSON form. Each test case looks like:

```
{
  "id": "test_001",
  "student_query": "Explain mitosis",
  "grade_level": "high_school",
  "subject": "biology",
  "expected_concepts": ["cell division","chromosomes","phases"],
```

```
  "ground_truth_answer": "optional reference answer",
  "type": "concept_explanation"
}
```

Datasets typically span multiple grade levels, subjects, and question types. The system supports tenant-scoped dataset uploads and synchronous or asynchronous processing.

4 Evaluation metrics and rubric-driven scoring

The pipeline evaluates each tutor response across five core metrics:

1. Clarity
2. Completeness
3. Accuracy
4. Appropriateness
5. Long-term Retention (memory consistency)

Each metric is scored on a 1–5 scale using an LLM-as-judge rubric. A typical judge prompt requests:

- the score for a metric,
- a short justification,
- strict JSON output.

The worker aggregates metric scores, generates an overall score, and stores all results in both NDJSON audits and SQL tables.

5 System architecture

The pipeline consists of a frontend, orchestrator, judge model adapters, job queues, and data persistence layers.

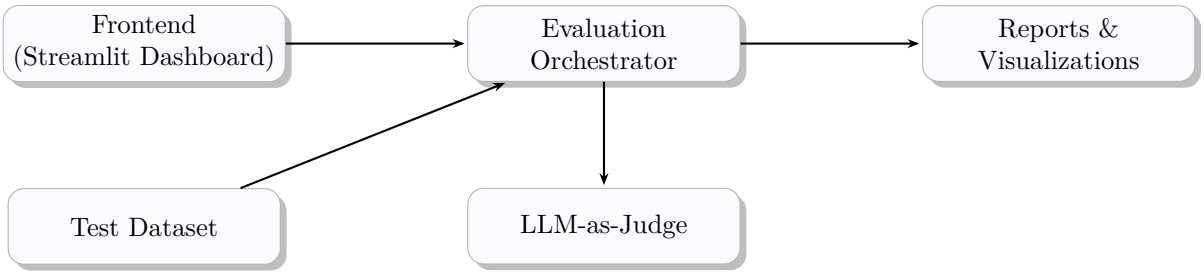

Figure 1: Compact, A4-safe architecture of the LLM Evaluation Pipeline.

6 End-to-end evaluation flow

1. A client requests an evaluation job through the API.
2. The orchestrator validates the dataset and splits evaluation cases.
3. Cases are queued in Redis RQ or executed synchronously.
4. The worker retrieves tasks, constructs rubric-based judge prompts, receives responses, and stores:
   - metric-level scores,
   - JSON rationales,
   - aggregated scores,
   - audit logs (NDJSON),
   - SQL DB rows.
5. The API exposes job metadata, per-case results, status, CSV downloads, and summary statistics.

7 Running the system

Local development

```
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Start the FastAPI app:

```
uvicorn backend.app.main:app --reload --port 8000
```

Start the worker in another terminal:

```
python backend/scripts/start_rq_worker.py
```

For synchronous debugging:

```
python backend/scripts/run_sync_evaluation.py
```

Full stack via Docker Compose

```
docker-compose -f infra/docker-compose.yml up --build
```

This launches Redis, Postgres, MinIO, backend, and workers.

8 Configuration and storage

- Configuration is managed through `config.yaml` and environment variables.
- The database layer uses SQLAlchemy and defaults to SQLite when no external DB is configured.

- NDJSON audit files store detailed evaluation traces.
- CSV exports are available through the evaluation results API.

9 Developer notes and extension points

- Judge adapters are pluggable; new adapters can be added for custom models or evaluation backends.
- Authentication currently uses development API keys; production systems can integrate OAuth or JWT.
- Tests can be added under `tests/` using pytest for API, worker, and adapter validation.
- Additional metrics or rubric styles can be integrated by extending the judge prompt templates.