AI Lab Notebook for Experiments

Payal Saha (24200)
Megha Patidar (24758)
Paras Raina (24170)
Himanshu Ranjan (24077)

Abstract

Reproducibility is an enduring bottleneck in machine-learning research: experiments are often run ad-hoc, metadata is scattered, and comparing results across runs is time-consuming. We propose an "AI Lab Notebook" - a lightweight ResearchOps assistant that automatically logs datasets, hyperparameters, metrics and artifacts to Weights & Biases (W&B) while synchronizing a compact local index for fast querying. A LangChain-driven agent provides natural-language access to the experiment store, composes concise run summaries (best configs, loss curves, anomalies), and supports simple agentic actions (e.g., flag a run for publishing to Hugging Face). The system combines three technologies: W&B for experiment telemetry, Hugging Face for model/artifact management, and LangChain for agent orchestration and retrieval-augmented generation (RAG). Expected outcomes are (1) a working end-to-end prototype that logs runs automatically, (2) a Streamlit UI for a human-friendly view of runs and comparisons, and (3) an LLM agent that answers queries like "Compare today's experiment with last week's" without hallucinating facts. This project demonstrates how agentic tooling can materially improve reproducibility and researcher productivity.

1 Introduction

Background and Context

Modern ML research relies on iterative experimentation: many hyperparameter sweeps, dataset variants, and model checkpoints are produced while only a small fraction are documented well. Without systematic tracking, reproducing or comparing experiments becomes costly, slowing progress and undermining scientific rigor.

Objectives and Significance

This project aims to deliver a compact, demonstrable ResearchOps tool that:

- Automatically logs experiment metadata and artifacts.
- Summarizes runs and surfaces anomalies.
- Enables natural-language queries over past experiments and performs constrained agentic actions (e.g., flagging runs).

The significance lies in improving reproducibility, reducing manual bookkeeping, and accelerating iteration for researchers and small labs.

2 Methodology

Planned Approach

We build a minimal pipeline integrating three components:

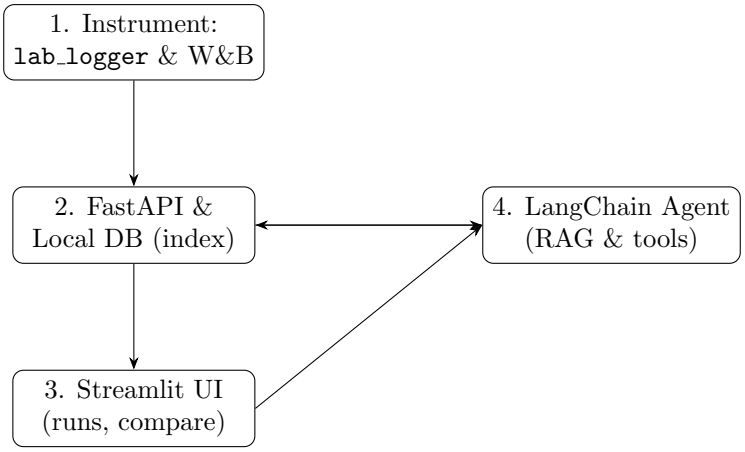
1. **Instrumentation:** a small Python utility `lab_logger` (decorator/CLI) that captures hyperparameters, metrics, artifacts, git commit, and environment specs and writes to W&B. It also emits a compact manifest to a local SQLite index.
2. **Backend & Indexing:** a lightweight FastAPI service that syncs selected W&B summaries into a local DB and exposes endpoints for listing runs, retrieving run details, comparing two runs, and flagging runs for publish.
3. **Agent & Retrieval:** a LangChain-based agent that follows a tool-first design: parse intent → call backend API tools → retrieve structured facts → compose a concise, grounded natural-language response. Retrieval-augmented generation (RAG) prevents hallucination by constraining the LLM to DB/W&B facts.
4. **Frontend / UX:** a Streamlit app (MVP) that shows runs, loss/metric plots (fetched from W&B or local artifacts), a compare modal, and an assistant chat box that sends queries to the agent.

Algorithms and Tools

- **Tools:** Weights & Biases (experiment telemetry), Hugging Face (model hosting), LangChain (agent orchestration), FastAPI (backend), Streamlit (UI), SQLite (local index for fast queries).
- **Core algorithms / heuristics:**
  - Run summarizer: select best checkpoint by validation metric, report final metrics, and produce short bullet summaries.
  - Anomaly detection: simple heuristics (sudden spikes, NaNs, flat loss) and threshold-based alerts for the demo.
  - Comparison/diff: compute numeric differences in hyperparameters and overlay metric curves.
  - Agent parsing: few-shot prompt templates to map NL queries into supported intents (list, compare, best, flag).

Workflow / Implementation Pipeline

1. Developer installs `lab_logger` and sets W&B API key.
2. Training script annotated with `@log_run` or calls logger API; on run completion, metadata and artifacts are pushed to W&B and a manifest is written to local DB.
3. A FastAPI worker syncs W&B summaries into local DB and indexes run manifests.
4. The Streamlit UI displays runs and provides plotting and comparison.
5. The LangChain agent receives natural-language queries, uses backend "tools" (API calls) to fetch facts, and composes grounded answers with links to runs and plots.



Evaluation

We evaluate the system on: correctness of agent responses (no hallucinations on a test set of queries), fidelity of run summaries (matches W&B), and demonstration of a reproducible rerun (manifest leads to same command and env).

3 Individual Member Contribution

- **Himanshu Ranjan (Instrumentation & Experiments)**
  Implement the decorator/utility that logs hyperparameters, metrics, artifacts, git hash and environment to W&B and writes the manifest. Responsible for running and tagging the short experiments used in the demo. Will publish the manifest schema and usage examples for other components to rely on.
- **Paras Raina (Backend & Indexing)**
  Development of the local DB (SQLite) schema and the ingestion pipeline that indexes run manifests. Implement FastAPI endpoints for listing runs, fetching run details, comparing runs, and flag-for-publish. Will publish the API contract (endpoints and JSON formats) for the frontend and agent to consume.
- **Payal Saha (Agent & Retrieval)**
  Design prompt templates, implement the RAG workflow, and build tool wrappers that invoke FastAPI endpoints. Responsible for the agentic action (flag-for-publish), evaluation of prompt correctness, and safety constraints to avoid hallucination. Will share example prompts and the agent's output schema.
- **Megha Patidar (Frontend & Integration)**
  Implement the runs table, run detail page, compare modal, and chatbox connected to the agent. Integrate API responses and W&B plots. Responsible for final demo flow and UI polish.

Each member is individually responsible for documentation, tests for their component, and contributing to the final demo. Coordination tasks (API specifications, manifest formats, and identifiers for demonstration) will be communicated clearly by the responsible member to ensure smooth integration across components.