Project Scoping Submission - Break The Bot

Team Members -

- Anjali Pai
- Atharv Talnikar
- Nitya Ravi
- Rahul Kulkarni
- Taniksha Datar
- Yashi Chawla

1. Introduction

Large Language Models (LLMs) are rapidly being deployed across products and services, but they remain vulnerable to jailbreaks and prompt-injection attacks that trick models into ignoring safety guidelines. Current safety testing practices are often manual, which makes it difficult for teams to continuously ensure safety as models evolve, prompts drift, and new vulnerabilities are discovered.

Our project aims to address this gap by building an automated MLOps pipeline for continuous LLM safety evaluation. The system will regularly run curated sets of adversarial prompts against open-source LLMs, measure attack success rates (ASR), and evaluate refusal quality using LLM-as-a-judge models. Results will be stored, visualized in dashboards, and monitored over time to detect regressions or emerging risks.

By embedding safety checks into a production-style workflow with CI/CD integration, monitoring, and alerting, this project provides a repeatable, auditable approach to LLM safety assurance. This is especially critical today as LLMs are deployed into real-world, high-stakes applications where even a single unsafe output can have significant consequences.

2. Dataset Information

We will use curated datasets of jailbreak and prompt-injection prompts along with custom red-team and attacker-LLM generated examples. Each entry stores the adversarial prompt, target model output (raw or redacted), detector scores, and labels. The dataset supports automated evaluation, safety regression testing, and detector training.

2.1 Data Card:

- **Name:** MLOps-Project-Adversarial-Prompts-v1
- **Sources:**
    1. Public benchmarks (AdvBench, JailbreakBench).
    2. Human-crafted red-team prompts.
    3. Automated attacker LLM generations (templates, paraphrases, mutations).

- **Fields:** prompt_id, prompt_text, origin, strategy, target_model_version, target_output_redacted, detector_scores, human_label, run_ts.
- **Languages:** English.
- **Intended Use:** Safety evaluation, CI/CD gating, detector model training.
- **Limitations:** Contains potentially harmful content; redacted versions provided for safe sharing.

2.2 Data Rights & Privacy:

- Respect licenses of public datasets; include attribution.
- Production data must be PII-redacted before ingestion.
- Raw harmful outputs stored under access controls; redacted outputs used for dashboards and reports.
- Dataset is internal-only by default; sanitized subsets may be shared externally with disclaimers.

3. Github Repository

Link to repo - https://github.com/yashichawla/MLOps-Project

Repository Structure:

```
📂 Repository Structure
MLOps Project/
├── data/           # seed prompts, generated prompts, adversarial examples
├── pipelines/      # evaluation and monitoring pipelines
├── runner/        # evaluator service (FastAPI, executes prompts on target
LLMs)
├── judge/          # judge service (FastAPI, scoring unsafe completions)
├── dashboards/     # Grafana dashboards, metrics visualization
├── docs/           # documentation, reports, diagrams
├── tests/          # unit tests, smoke tests, regression suites
├── .github/
│   └── workflows/  # CI/CD automation (GitHub Actions)
│
├── README.md       # project overview
├── requirements.txt# Python dependencies
├── .gitignore      # ignored files (logs,cache, etc.)
└── docker-compose.yml # container orchestration
```

4. Project Scope

4.1 Problems

- Jailbreak Vulnerabilities: LLMs can be tricked into producing unsafe or harmful outputs via jailbreaks and prompt injections.
- Ad-hoc Safety Testing: Current safety evaluations are often manual, inconsistent, and not embedded in CI/CD pipelines.
- Drift and Regression: Models change over time (new releases, fine-tunes), but teams lack automated monitoring to detect safety regressions.
- Lack of Auditability: Safety evaluations are rarely documented in a repeatable, governance-friendly way.

4.2 Current Solutions

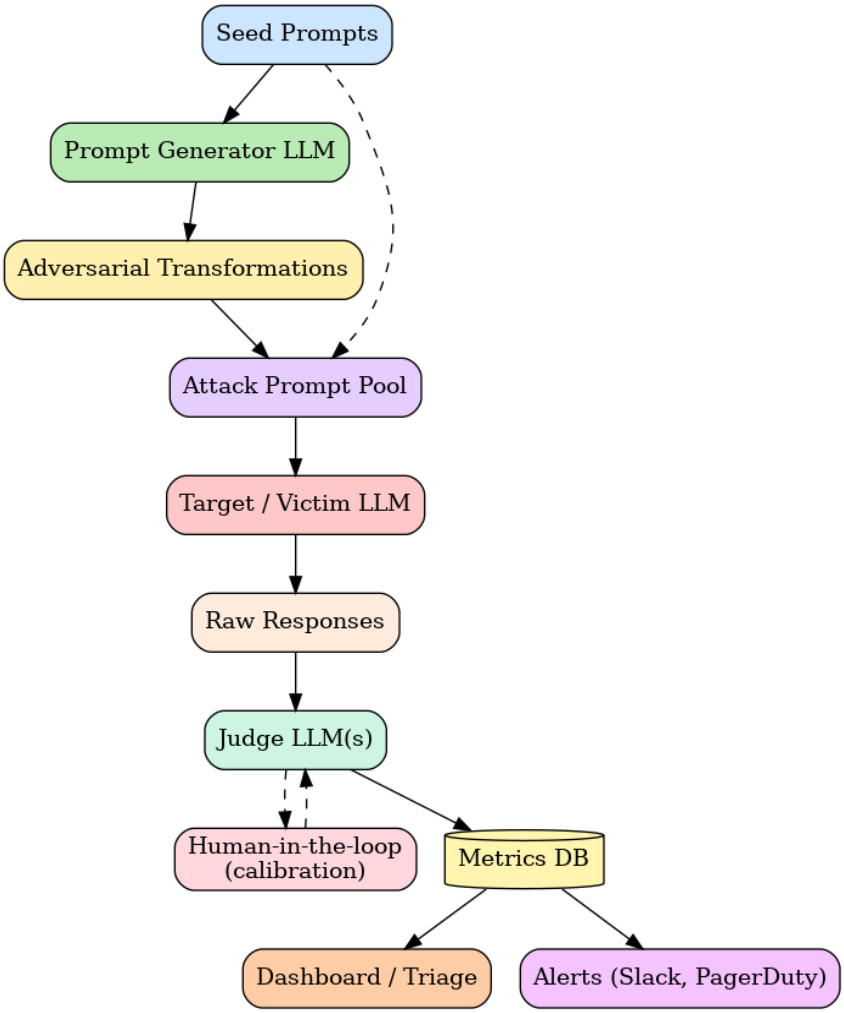
- Manual Red-Teaming: Human testers attempt jailbreaks before deployment, but this is time-consuming, inconsistent, and non-scalable.
- Benchmark Datasets: Provide standardized attack prompts, but results are often static and not tracked over time.
- Closed-Source Safety Evaluations: Proprietary models (e.g., OpenAI's evals) have internal systems, but these are not accessible, auditable, or applicable to open-weight models.

4.3 Proposed Solutions

- Automated Safety Pipeline: Build an MLOps system that continuously evaluates open-source LLMs against curated adversarial datasets.
- LLM-as-a-Judge Evaluation: Use judge models to automatically score refusal quality and detect unsafe completions.
- Monitoring & Alerts: Store results, visualize safety metrics (ASR, refusal quality), and trigger alerts when regressions or drift occur.
- Governance Framework: Document safe handling of sensitive data, provide reproducible reports, and ensure transparency in safety evaluations.

5. Current Approach Flowchart and Bottleneck Detection

5.1 Flowchart/ Current Approach



5.2 Bottlenecks

- Compute & Latency: Running large prompt sets and judge-model scoring can be expensive and slow.
- Dataset Freshness: Static attack sets quickly become outdated; new adversarial prompts must be added.
- Judge Reliability: LLM-as-a-judge may misclassify edge cases or introduce bias into safety scores.
- Drift & Monitoring: Shifts in prompts or model behavior can reduce evaluation accuracy; continuous monitoring is needed.
- Interpretability: High-level metrics (e.g., ASR) need detailed failure examples to explain *why* a model failed.

6. Metrics, Objectives, Business Goals

6.1 Metrics

- Attack Success Rate (ASR): % of adversarial prompts that bypass safety.
- Refusal Quality Score: Rated by LLM-as-a-judge for clarity, and completeness of refusals.
- Coverage Metrics: Number and diversity of prompts tested across attack categories.

6.2 Objectives

- Automate red-teaming and safety evaluations in a reproducible pipeline.
- Provide transparent, auditable safety metrics to stakeholders.

6.3 Business Goals

- Reduce the risk of unsafe outputs in deployed applications.
- Improve trustworthiness and compliance of LLM-based systems.
- Enable scalable safety monitoring for future model iterations.

7. Failure Analysis

7.1 Definition of Failures:

- Unsafe Completions: The model outputs harmful or policy-violating responses (e.g., toxic language, PII leakage, successful jailbreaks).
- Weak Refusals: The model refuses to respond but does so ineffectively (e.g., vague refusal, refusal where a safe answer was expected).

7.2 Categorization of Failures:

- PII Leakage: unintended disclosure of personal information.
- Toxic/Harassment: generation of offensive or unsafe content.
- Compliance Refusal Errors: refusing safe queries or misclassifying benign requests.
- Jailbreak Successes: system prompts or safety rules bypassed.
- Hallucinated Refusals: false claim that a safe request is disallowed.

7.3 Analysis Workflow:

- Logging: All failed prompt-response pairs are logged in the metrics database with metadata (model version, timestamp, failure category, judge score).

- Audit Trail: High-severity examples are stored as case studies for compliance reviews and improvement planning.

7.4 Outputs of Failure Analysis:

- Failure Rate by Category: (e.g., 12% jailbreak bypass, 8% weak refusal).
- Trend Analysis: monitoring whether failure rates increase after model updates.
- Recommendations: actionable next steps, such as counter-training, prompt filtering, or refinement of the refusal template.

8. Deployment Infrastructure

8.1 Objectives

- Reproducible, local-first deployment for the evaluator + judges.
- Versioned data and models; auditable runs.

8.2 Environments

- **Local Dev:** docker-compose for all services; CPU-friendly models (GGUF via llama.cpp) to keep costs and setup low.

8.3 Components & Tools

- **Containers:** Docker images for evaluator-api, judge-api.
- **Artifact & Data Store: MinIO** (S3-compatible) for Parquet data, MLflow artifacts, and reports.
- **Experiment Tracking: MLflow** server (SQLite backend is fine).
- **Data Versioning: DVC** (remote = MinIO) for prompt/eval snapshots.
- **Observability (core): Prometheus** client metrics from services + **Grafana** basic dashboard.

8.4 Topology

- evaluator-api (FastAPI) → orchestrates runs against containerized **Model(s) Under Test**.
- judge-api (FastAPI) → safety/refusal judges as a separate service (lets you scale/swap judges).
- mlflow, minio (+ mc), prometheus, grafana.

8.5 CI/CD

- **GitHub Actions**: run unit tests + Great Expectations checks + tiny **smoke suite** (e.g., 50 prompts) on PR; build and push images to GHCR.

8.6 Secrets & Governance

- .env → Kubernetes secrets (if using K8s); never commit raw harmful strings or API keys.
- Content-handling policy: store **hash IDs + safe summaries**, not raw risky generations.

8.7 Storage Layout

- MinIO buckets: dvc/, mlflow/, models/, reports/.

8.8 Performance Notes

- Start with **7–9B** open LLMs quantized (GGUF) on CPU; set small concurrency env vars; keep prompts short.

9. Monitoring Plan

9.1 Service SLOs

- **Evaluator/judge latency:** p95 < **150 ms** on small prompts (CPU).
- **Error rate:** < **2%** 5-min rolling window during runs.

9.2 Safety & Quality KPIs

- **Attack Success Rate (ASR)** overall and by category.
- **Refusal precision/recall** on a benign control set.

9.3 Data Quality Monitors

- **Great Expectations**: schema, nulls, duplicates, category balance on each suite.

9.4 Drift & Regression Rules

- **ASR regression gate:** block release if ASR increases by **> 2 percentage points** vs last accepted run (overall or in any critical category).
- **Refusal precision floor:** block if < **90%** on the benign control set.

9.5 Observability Setup

- **Prometheus**: request rate/latency/error metrics from both APIs; job-level counters for prompts processed, judge outcomes.
- **Grafana**: one “Safety Overview” dashboard (ASR, refusal precision/recall, release annotations) + one “Ops” dashboard (latency, errors, MinIO health).

9.6 Runbooks

- Short markdown runbooks: **“ASR Spike”**, **“Refusal Precision Drop”**, **“Service Error Burst.”**

10. Success and Acceptance Criteria

10.1 Success Criteria

- A fully functional pipeline that automatically evaluates LLMs against adversarial prompts.
- Safety metrics (e.g., Attack Success Rate, Refusal Quality) are computed, logged, and visualized.
- The system operates reproducibly, with clear documentation and governance guidelines.

10.2 Acceptance Criteria

- Dashboards clearly display trends in ASR and refusal quality over time.
- CI/CD integration blocks deployment if safety metrics fall below defined thresholds.
- Stakeholders (e.g., instructors, evaluators) can audit results and reproduce evaluations using the provided framework.

11. Timeline Planning

Week 1 – Foundations

- Set up GitHub repository with structure (`/data`, `/pipelines`, `/runner`, `/judge`, `/dashboards`).
- Define governance and PII redaction policy.
- Collect & version initial seed prompts.
- Configure DVC for dataset versioning.

Week 2 – Prompt Generator LLM

- Build generator service (FastAPI/Python).
- Generate paraphrased and roleplay-based variants of seed prompts.
- Store outputs in prompt pool with metadata (seed_id, generator model, transformations applied).

Week 3 – Adversarial Transformations & Prompt Pool

- Implement obfuscation, encoding, context injection, roleplay personas.
- Create Attack Prompt Pool (versioned, tagged) in Postgres/DVC.
- Add prompt categories (info disclosure, toxic, compliance bypass, etc.).

Week 4 – Runner / Target LLM Integration

- Build evaluator service: pull prompts → call target LLM → log responses.
- Store raw responses in encrypted MinIO/S3.
- Log metadata (model version, parameters) in Metrics DB.

Week 5 – Judge LLM

- Deploy judge service with JSON schema: `{violation, category, severity, exploitability, confidence}`.
- Run first judge calibration with ~200 labeled samples.
- Log judge scores alongside raw responses.

Week 6 – Human-in-the-loop & Calibration

- Build annotation UI (Streamlit/Label Studio).
- Label 500+ prompt-response pairs for calibration.
- Evaluate judge precision/recall against human gold labels.
- Route low-confidence outputs to human annotators.

Week 7 – Metrics DB & Dashboard Prototype

- Implement Postgres schema (runs, prompts, responses, judge_scores, human_labels).
- Build Grafana dashboards:
    - Safety Overview (ASR, severity trends, categories).
    - Ops Dashboard (latency, error rates).

Week 8 – Failure Analysis, Clustering & Regression Tests

- Identify recurring jailbreak strategies.

- Add regression tests to CI.

Week 9 – Scaling, Infra & CI Gates
- Dockerize runner + judge APIs.
- Add Prometheus monitoring for throughput/latency.

Week 10 – Final Validation & Reporting
- Run hidden test set for unbiased evaluation.
- Validate regression suite & CI/CD gates.
- Produce final dashboards, runbooks, ethics checklist.
- Deliver final presentation & project report.

12. Additional Information

12.1 Ethical and Responsible Use

- Purpose Limitation: The pipeline is designed strictly for *evaluation and safety monitoring*, not for building or disseminating harmful prompts.
- Content Handling: All adversarial data will be stored in secure, access-restricted locations. Publicly shared results will use redacted or summarized forms.
- Transparency: Reports and dashboards will document methodology and evaluation governance, enabling auditability without exposing harmful content.

12.2 Future Extensions

- Multilingual Expansion: Incorporate adversarial prompts in other languages (e.g., Hindi, Spanish, Mandarin) to cover global deployment scenarios.
- Adaptive Attacks: Integrate attacker-LLMs that evolve prompt obfuscations over time, simulating real-world adversaries.
- Cross-Model Benchmarking: Extend pipeline to evaluate both open-source and closed-weight APIs (e.g., GPT-4, Claude) for comparative safety metrics.

12.3 Long-Term Vision

This project aims not just to test models today, but to create a repeatable, auditable framework for continuous LLM safety assurance. Over time, it could evolve into a plug-and-play evaluation service for organizations deploying LLMs in sensitive domains such as finance, healthcare, and education.