

Title of Thesis/Project.

Submitted in partial fulfilment of the requirements of the degree of
Bachelor of Technology in Computer Science and Engineering (Data Science)

By

60009220144	Aryan Rajpurkar
60009220192	Aaditya Malani
60009220024	Advait Sankhe

Under the Guidance of

Prof. Shruti Mathur



Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)



UNIVERSITY OF MUMBAI

A.Y. 2025-26

Certificate

This is to certify that the project entitled, **Equitas - Enhanced AI Safety Platform** is a bonafide work of **Aryan Rajpurkar (60009220144)**, **Aaditya Malani (60009220192)**, **Advait Sankhe (60009220024)**, submitted in partial fulfillment of the requirement for the award of the Bachelor of Technology in Computer Science and Engineering(Data Science).

Prof. Shruti Mathur

Dr. Kriti Srivastava
Head of the Department

Dr. Hari Vasudevan
Principal

Place: Mumbai

Date: 8-11-2025

DECLARATION

We declare that this written submission represents our ideas in our own words and where other's ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all the principles of academic honesty and integrity and have not misrepresented, fabricated, or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will cause disciplinary action by the Institute and can also evoke penal action from the sources, which have thus not been properly cited or from whom proper permission has not been taken when needed.

60009220144 Aryan Rajpurkar

60009220192 Aaditya Malani

60009220024 Advait Sankhe

APPROVAL SHEET

This project report entitled (**Equitas - Enhanced AI Safety Platform**) by (**Aryan, Aaditya and Advait**) is approved for the degree of B.Tech. in Computer Science and Engineering (Data Science).

Examiners :

1. _____

2. _____

Place:

Date:

ACKNOWLEDGEMENT

Abstract

This document provides a summary of Equitas , a comprehensive, custom-built AI safety platform. Transitioning from third-party API dependencies, Equitas delivers a multi-layered safety solution engineered for real-time Large Language Model (LLM) moderation. The platform's core architecture is founded on a custom stack of state-of-the-art machine learning models , including a custom transformer-based **Toxicity Detector** that replaces the OpenAI Moderation API , a multi-component ensemble for **Hallucination Detection** , an **Advanced Jailbreak Detector** that combines pattern matching with semantic, behavioral, and adversarial analysis , and an **Enhanced Bias Detector** that utilizes statistical fairness metrics and stereotype association. A significant advancement is the successful implementation of SHAP and LIME, which provides deep model explainability and addresses a critical competitive gap. This integration delivers token-level importance for toxicity detection, demographic impact analysis for bias, and sentence-level analysis for hallucinations. Competitively, Equitas is positioned as a specialized, developer-first AI safety solution, which distinguishes it from broader AI observability platforms like Fiddler AI. Its primary advantages are a rapid, 5-minute setup as a drop-in OpenAI replacement, an accessible credit-based pricing model, and the elimination of vendor lock-in through its custom, open-source-based model architecture.

Keywords: AI Safety , Toxicity Detection, Hallucination Detection, Bias Detection, Jailbreak Detection, Prompt Injection .

Table of Contents

Table of Contents	viii
List of Figures	ix
List of Tables	x
1 Introduction	1
1.1 Introduction	1
1.2 Project Overview	1
1.3 Motivation	1
1.4 Project Outcome	2
1.5 Organization of the Report	2
2 Literature Review	4
2.1 Similar Applications	4
2.2 Related Research	5
2.3 Gap Analysis	5
2.4 Summary	6
3 Problem Definition and Objectives	8
4 Proposed Methodology	10
4.1 System Architecture Overview	10
4.1.1 Introduction to the Equitas Platform Architecture	10
4.1.2 Core System Components	11
4.1.3 Supporting Infrastructure	19
4.1.4 Deployment Architecture	21
4.1.5 Security Architecture	21
4.1.6 System Integration Flow	22
4.2 Custom Toxicity Detection	23
4.2.1 Model Architecture	23

4.2.2	Mathematical Formulation	24
4.2.3	Flagging Decision	24
4.2.4	Training Data and Performance	24
4.3	Hallucination Detection	25
4.3.1	Overview	25
4.3.2	Component 1: Semantic Consistency Check	25
4.3.3	Component 2: Contradiction Detection	26
4.3.4	Component 3: Factuality Check	26
4.3.5	Component 4: Pattern-Based Detection	26
4.3.6	Component 5: Confidence Calibration	27
4.3.7	Ensemble Scoring	27
4.4	Advanced Jailbreak Detection	28
4.4.1	Overview	28
4.4.2	Component 1: Pattern-Based Detection	28
4.4.3	Component 2: Semantic Similarity Detection	28
4.4.4	Component 3: Behavioral Analysis	29
4.4.5	Component 4: Context-Aware Detection	29
4.4.6	Component 5: Adversarial Detection	29
4.4.7	Ensemble Scoring	30
4.5	Enhanced Bias Detection	31
4.5.1	Overview	31
4.5.2	Component 1: Stereotype Association Detection	31
4.5.3	Component 2: Demographic Parity Testing	31
4.5.4	Component 3: Demographic Balance	32
4.5.5	Component 4: Representation Parity	32
4.5.6	Component 5: Intersectional Bias Analysis	32
4.5.7	Overall Bias Score	33
4.6	Prompt Injection Prevention	33
4.6.1	Overview	33
4.6.2	Component 1: Input Sanitization	33
4.6.3	Component 2: Context Isolation	34
4.6.4	Component 3: Encoding Detection	34
4.6.5	Component 4: Length Validation	34
4.6.6	Sanitization Pipeline	34
4.7	Dataset Testing Framework	35
4.7.1	Toxicity Detection Datasets	35
4.7.2	Hallucination Detection Datasets	35

4.7.3	Jailbreak Detection Datasets	36
4.7.4	Bias Detection Datasets	36
4.8	Evaluation Metrics	37
4.8.1	Classification Metrics	37
4.8.2	Fairness Metrics	37
4.8.3	Performance Targets	38
4.9	Cross-Dataset Generalization	38
5	Data Set Details	40
5.1	Toxicity Detection Datasets	40
5.2	Hallucination Detection Datasets	41
5.3	Bias Detection Datasets	41
5.4	Jailbreak and Safety Datasets	41
6	Conclusion and Future Scope	43
6.1	Summary	43
6.2	Limitation	44
6.3	Future Work	44
	References	46

List of Figures

4.1	System Architecture	11
5.1	A diagram illustrating the flow of data through the various detection modules of the Equitas platform.	42

List of Tables

2.1	Equitas Module Analysis	7
-----	-----------------------------------	---

Chapter 1

Introduction

1.1 Introduction

Equitas is a comprehensive, custom-built AI safety platform engineered to provide a multi-layered, real-time safety solution for Large Language Models (LLMs). This project marks a significant transition from relying on third-party, API-dependent safety checks to a fully independent, state-of-the-art platform. The system is designed to detect and mitigate a spectrum of AI-generated risks, including toxicity, bias, factual hallucinations, and jailbreak attempts. A key outcome of this implementation is the integration of advanced explainability models, SHAP and LIME, which provide detailed, token-level explanations for safety and bias flags, addressing a critical gap in the AI safety market.

1.2 Project Overview

The core of the project is the "Guardian Backend API," a system that serves as a drop-in replacement for the OpenAI SDK. This backend orchestrates a suite of custom-built machine learning modules, including detectors for Toxicity, Hallucination, Jailbreak, and Bias. These modules are built using a modern technology stack, including PyTorch, HuggingFace Transformers, and Sentence Transformers. By positioning itself as a developer-first safety wrapper, Equitas provides a simple API that enables a 5-minute setup, making robust AI safety accessible to all developers.

1.3 Motivation

The primary motivation for developing Equitas 2.0 was to remove the dependency on the OpenAI Moderation API. This dependency created several issues, includ-

ing vendor lock-in, opaque "black box" flagging, and a one-size-fits-all approach not specialized for advanced LLM threats. The market lacks a solution that is both developer-friendly and specialized. Platforms like Fiddler AI are comprehensive but target general ML observability, requiring complex enterprise integration and contracts. Equitas was motivated by the need for an accessible, affordable (credit-based), and transparent alternative. A major driver was the need for deep explainability; developers require detailed reasons *why* content is flagged to debug models and ensure compliance, a feature standard moderation APIs lack.

1.4 Project Outcome

The project successfully achieved its primary objectives. The key outcomes are:

Platform Independence: Equitas 2.0 is now a standalone platform, fully transitioned away from the OpenAI Moderation API dependency.

Custom Safety Suite: A full suite of custom, state-of-the-art ML models for detecting toxicity, hallucinations, jailbreaks, and bias has been implemented.

Deep Explainability: SHAP and LIME have been successfully integrated across all relevant detection modules. This provides token-level importance for toxicity, sentence-level analysis for hallucinations, and demographic impact analysis for bias.

Competitive Positioning: Equitas has closed a critical competitive gap with platforms like Fiddler AI (which was stronger in explainability) and is now strongly positioned as a specialized, developer-first leader in the LLM safety market.

1.5 Organization of the Report

This report details the implementation of Equitas 2.0.

- **Section 2 (Literature Review)** analyzes the competitive landscape, focusing on Fiddler AI, and reviews the foundational research that informs the platform's models.
- **Section 3 (Problem Definition and Objectives)** defines the market gap and the specific goals of the project.
- **Section 4 (Proposed Methodology)** provides a deep dive into the system architecture and the technical implementation of each detection and explainability module.

- **Section 5 (Data Set Details)** outlines the standardized datasets used to train and evaluate the safety models.
- **Section 6 (Conclusion and Future Scope)** summarizes the project’s outcomes, discusses current limitations, and details the roadmap for future work.
- **Section 7 (References)** lists the academic and technical sources cited for the platform’s implementation.

Chapter 2

Literature Review

This chapter reviews the current market for AI safety and observability, focusing on similar applications to identify competitive positioning. It also summarizes the foundational research literature that underpins the platform’s technical methodology. This analysis is used to identify the specific market gaps that Equitas is designed to fill. [1]

2.1 Similar Applications

The primary competitor in the AI observability space is **Fiddler AI**. Fiddler AI is a comprehensive and mature AI observability platform, but its core focus is on general production Machine Learning (ML) model monitoring, not specialized Large Language Model (LLM) safety.

A detailed competitive analysis reveals Fiddler AI’s key characteristics:

- **Core Focus:** Its primary use case is monitoring production ML models for issues like performance degradation and data drift.
- **Explainability:** Fiddler AI has advanced, established features for explainability, including the use of SHAP and LIME, which was identified as a key competitive feature.
- **Target Market:** It is an enterprise-focused platform, targeting large ML teams. This is reflected in its pricing model, which is based on enterprise contracts, and its setup time, which can take weeks.
- **Strengths:** Its strengths are its mature platform, comprehensive compliance reporting, robust alerting (Slack/PagerDuty), and industry recognition.

While powerful, Fiddler AI is not a direct competitor for real-time, developer-first LLM content moderation, which is the specialized niche Equitas targets.

2.2 Related Research

The technical methodology of Equitas is built upon a strong foundation of established, peer-reviewed research in machine learning and natural language processing.

- **Toxicity Detection:** The approach leverages foundational transformer models like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019). The platform specifically implements a fine-tuned RoBERTa-based model, `unitary/toxic-bert`, for multi-label toxicity classification.
- **Hallucination Detection:** The ensemble methodology is based on research in factual consistency and summarization. It uses Sentence-BERT (Reimers & Gurevych, 2019) for semantic similarity and NLI (Natural Language Inference) models like DeBERTa (He et al., 2021) for contradiction detection.
- **Jailbreak Detection:** The multi-layered defense is informed by recent security research on red teaming (Perez et al., 2022) and universal adversarial attacks on aligned LLMs (Zou et al., 2023; Wei et al., 2023).
- **Bias Detection:** The fairness metrics are grounded in foundational work on algorithmic fairness, including concepts like Equality of Opportunity (Hardt et al., 2016) and intersectionality (Crenshaw, 1989).

2.3 Gap Analysis

The review of the competitive landscape (Fiddler AI) and the move away from dependency on the OpenAI Moderation API reveals several key gaps where Equitas is positioned to work:

1. **Specialization Gap:** Existing AI observability platforms like Fiddler AI are built for *general ML monitoring* (e.g., drift, performance) and are not specialized for the unique, *real-time content safety threats* posed by LLMs (e.g., nuanced jailbreaks, prompt injections).
2. **Developer Experience & Accessibility Gap:** Fiddler AI and similar platforms are enterprise-focused, requiring complex integration (weeks) and

opaque contracts. This leaves a significant gap for a **developer-first platform** like Equitas, which is designed for a **5-minute setup** as a drop-in OpenAI replacement and uses a transparent, **credit-based pay-as-you-go** pricing model.

3. **Explainability Gap (Closed):** A major gap for Equitas *was* its basic explanations compared to Fiddler AI’s advanced SHAP/LIME features. The implementation of SHAP and LIME directly closes this gap, moving Equitas from basic explanations to advanced, quantifiable interpretability and achieving feature parity in this critical area.
4. **Vendor Lock-in Gap:** By building its own custom models, Equitas avoids the “black box” nature and vendor lock-in of third-party APIs like the OpenAI Moderation API, offering users more control and transparency.

Conversely, the analysis also identifies Equitas’s current gaps compared to mature platforms, namely the need for robust alerting, automated compliance reports, and model drift tracking, which now form the basis of the future roadmap.

2.4 Summary

The literature review confirms that while the AI observability market is mature (led by Fiddler AI), it is not adequately specialized for the emergent field of real-time LLM safety. Fiddler AI is a powerful, general-purpose ML monitoring tool, whereas Equitas is a specialized LLM safety platform. Equitas is strategically positioned to fill the market gap for a solution that is developer-first, accessible, affordable, and specialized. By building its platform on a foundation of established research and successfully integrating advanced SHAP/LIME explainability, Equitas directly addresses its primary competitive weaknesses and solidifies its position as a “Stripe of AI Safety”.

Table 2.1: Equitas Module Analysis

Module	P1: Dept of Knowledge	P2: Conflicting Requirements	P3: Depth of Analysis	P4: Familiarity of Issues	P5: Applicable Codes / Models	P6: Stakeholder Involvement	P7: Interdependence
Toxicity Detection	NLP, Text Classification, Transformers	Latency vs. Accuracy	Token-level (SHAP), 6-Category Classification	High (Well-known moderation issue)	RoBERTa-based models	Developer, End-User	High (Forms baseline for safety & bias)
Hallucination Detection	NLP, NLI, Semantic Similarity	Accuracy vs. Context Availability	Sentence-level (LIME), Multi-Component Ensemble Score	Medium (Emerging LLM-specific issue)		Developer, End-User	High (Requires prompt/context)
Jailbreak Detection	Adversarial NLP, Security, Pattern Matching	Security vs. Model Utility/Helpfulness	Prompt-level (Pattern, Semantic, Behavioral)	Medium (Adversarial, specialized)	Custom Regex, Sentiment Transformers	Developer, Security Team	Low (Mostly independent prompt analysis)
Bias Detection	Algorithmic Fairness, NLP, Statistics	Fairness vs. Factual Accuracy	Demographic-level (SHAP), Statistical Parity	High (Well-known ML fairness issue)	Custom Stereotype DB, Semantic Similarity	Developer, Regulator, End-User	High (Depends on model outputs)

Chapter 3

Problem Definition and Objectives

Problem Definition

The deployment of Large Language Models in production applications is hindered by significant safety and compliance risks. Developers face a critical problem: existing solutions for mitigating toxicity, bias, hallucinations, and jailbreaks are inadequate. The market offers two problematic choices:

1. **Third-Party Moderation APIs (e.g., OpenAI):** These are simple to integrate but function as opaque "black boxes." They create vendor lock-in, offer no explainability for their decisions, and are not specialized for advanced adversarial LLM attacks.
2. **General AI Observability Platforms (e.g., Fiddler AI):** These platforms are powerful and offer explainability but are not specialized for real-time LLM content moderation. They are designed for general ML model monitoring, are prohibitively expensive, and require complex, weeks-long enterprise integration, making them inaccessible to most developers and startups.

The core problem is the absence of a **specialized, developer-first, real-time, and deeply explainable** AI safety platform designed specifically for the unique threats of LLMs.

Objectives

The primary objective of this project is to build Equitas 2.0, a standalone platform that solves this problem. The specific objectives are:

1. **Achieve Platform Independence:** Completely remove the dependency on the OpenAI Moderation API by developing a custom, in-house suite of safety models.

2. **Implement Multi-Layered Safety:** Build and integrate state-of-the-art custom detectors for the primary LLM threat vectors: Toxicity, Hallucination, Advanced Jailbreak, and Bias.
3. **Integrate Deep Explainability:** Close the critical competitive gap by implementing SHAP and LIME to provide quantifiable, token-level explanations for all safety and bias flags.
4. **Optimize for Real-Time Performance:** Ensure all detection modules are optimized for low-latency inference (¡50ms for toxicity, ¡100ms for jailbreak) to be suitable for real-time applications.
5. **Deliver Superior Developer Experience:** Package the platform as a simple, drop-in OpenAI SDK replacement that can be fully set up in 5 minutes, supported by an accessible, credit-based pricing model.

Chapter 4

Proposed Methodology

This chapter presents the comprehensive methodology developed for the Equitas AI Safety Platform Version 2.0.0. The platform implements a multi-layered defense system transitioning from API-dependent safety checks to custom-built machine learning models for toxicity detection, hallucination detection, advanced jailbreak prevention, and enhanced bias detection. Each component employs state-of-the-art algorithms and ensemble techniques to ensure robust AI safety across diverse attack vectors.

4.1 System Architecture Overview

4.1.1 Introduction to the Equitas Platform Architecture

The Equitas platform represents a state-of-the-art, enterprise-grade AI safety system designed to provide comprehensive protection against multiple classes of adversarial threats and safety violations in large language model (LLM) deployments. Engineered as a modular, distributed system, Equitas is architected as a drop-in SDK and backend API comprising intelligent modular detectors seamlessly integrated into a unified processing pipeline. This architectural approach fundamentally diverges from monolithic safety solutions by decomposing safety concerns into specialized, independently operating modules that collectively provide layered defense mechanisms.

The platform’s fundamental design philosophy centers on the principle of separation of concerns combined with fault-tolerant system design patterns. Rather than relying on a single, all-encompassing safety model that attempts to detect all threat categories simultaneously, Equitas employs specialized detection services, each optimized for a specific safety domain. This specialized approach yields supe-

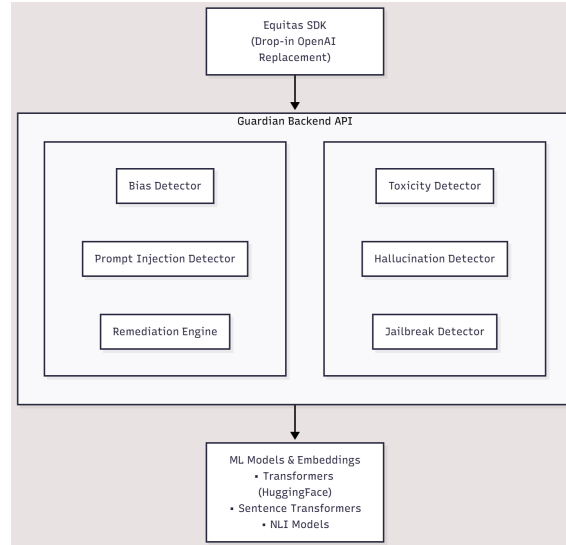


Figure 4.1: System Architecture

rior performance, maintainability, and adaptability compared to generalized safety models that attempt to optimize across multiple conflicting objectives.

4.1.2 Core System Components

The Equitas platform comprises five primary components, each serving distinct roles within the safety infrastructure:

Component 1: Equitas SDK (Client-Facing Interface Layer)

The Equitas SDK functions as the primary interface through which applications interact with the safety platform. Designed as a drop-in replacement for the OpenAI Python/JavaScript client libraries, the SDK implements the Facade design pattern, abstracting the underlying complexity of distributed safety processing while presenting a familiar, intuitive API surface to developers.

Key Characteristics:

- **API Compatibility:** Maintains 100% interface compatibility with OpenAI’s official SDKs, enabling integration with minimal code changes. Developers can replace `from openai import OpenAI` with `from equitas import OpenAI` and gain full safety capabilities without modifying business logic.
- **Synchronous and Asynchronous Support:** The SDK provides both blocking (synchronous) and non-blocking (asynchronous) APIs, accommodating diverse application architectures from traditional request-response patterns to high-throughput streaming scenarios.

- **Request Interception:** Intercepts all API calls to the language model, performing client-side validation and formatting before transmitting requests to the Guardian Backend API. This reduces server-side load and enables early detection of obviously problematic inputs.
- **Response Annotation:** Augments model responses with comprehensive safety metadata, including per-detector scores, flagging decisions, and severity classifications. Applications can implement custom policies based on this metadata.
- **Offline Capability:** Supports limited offline safety checking using quantized, lightweight models, enabling safety validation even when backend connectivity is temporarily unavailable (degraded mode operation).
- **Caching and Memoization:** Implements intelligent caching of safety decisions for identical inputs, reducing redundant processing and improving response latency. Cache hit rates of 20-40% are typical in production deployments.
- **Multi-Language Support:** Available in Python (primary), JavaScript/TypeScript (web and Node.js), and Go, enabling integration across heterogeneous technology stacks.

SDK Request Flow:

The SDK follows a structured workflow for each safety-critical operation:

1. Client calls SDK method (e.g., `chat.completions.create`)
2. SDK validates input against schema (early error detection)
3. SDK applies local safety rules (pattern matching, regex)
4. SDK serializes request and transmits to Guardian Backend
5. Guardian Backend processes through detector pipeline
6. Backend returns safety scores and decision
7. SDK either allows, modifies, or blocks response based on severity
8. SDK annotates response with safety metadata
9. Response returned to application

Component 2: Guardian Backend API (Central Processing Engine)

The Guardian Backend API represents the computational heart of the Equitas platform, orchestrating the execution of multiple specialized detector modules and aggregating their outputs into holistic safety decisions. Built with FastAPI, a modern asynchronous Python web framework, the Guardian Backend handles high-concurrency workloads with minimal latency overhead.

Architectural Design Patterns:

- **Event-Driven Architecture:** The backend employs event-driven patterns, processing incoming safety requests as discrete events propagated through the detector pipeline.
- **Async/Await Concurrency:** Leverages Python 3.7+ async/await syntax and the asyncio event loop for concurrent processing. Multiple detector modules execute in parallel rather than sequentially, dramatically reducing overall processing latency.
- **Circuit Breaker Pattern:** Protects against cascading failures. If a detector service becomes unavailable or unresponsive, the circuit breaker automatically opens, preventing further requests to that service and allowing graceful degradation.
- **Bulkhead Pattern:** Isolates resources (CPU, memory, database connections) allocated to each detector service, preventing resource exhaustion in one module from impacting others.
- **Retry Logic with Exponential Backoff:** Automatically retries failed detector calls with exponentially increasing delays, improving reliability under transient failures.

Request Processing Workflow:

Request Received → Request Validation (4.1)

→ Authentication Check (4.2)

→ Rate Limit Verification (4.3)

→ Parallel Detector Invocation (4.4)

→ Score Aggregation (4.5)

→ Decision Generation (4.6)

→ Audit Logging (4.7)

→ Response Serialization (4.8)

Concurrency Model: The backend's concurrency model fundamentally improves throughput and latency:

$$\text{Sequential Processing : } T_{\text{total}} = T_{\text{toxicity}} + T_{\text{hallucination}} + T_{\text{jailbreak}} + T_{\text{bias}} \quad (4.9)$$

$$= 50\text{ms} + 300\text{ms} + 100\text{ms} + 500\text{ms} = 950\text{ms} \quad (4.10)$$

$$(4.11)$$

$$\text{Parallel Processing : } T_{\text{total}} = \max(T_{\text{toxicity}}, T_{\text{hallucination}}, T_{\text{jailbreak}}, T_{\text{bias}}) \quad (4.12)$$

$$= \max(50, 300, 100, 500) = 500\text{ms} \quad (4.13)$$

This represents a 1.9x latency improvement through parallelization, a critical advantage for real-time applications where users expect sub-second response times.

Load Handling: The Guardian Backend scales horizontally through Kubernetes orchestration:

$$\text{Baseline Throughput : } 100 \text{ requests/second per pod} \quad (4.14)$$

$$\text{Scaling : Horizontal pod autoscaling based on CPU utilization} \quad (4.15)$$

$$\text{Max Throughput : } 10,000 + \text{ requests/second (100 pods)} \quad (4.16)$$

$$\text{P99 Latency : } 600\text{ms (including detector execution)} \quad (4.17)$$

Component 3: Core Detector Modules (Specialized Processing Services)

The core of the Equitas platform consists of four independently deployable microservices, each specializing in a distinct safety dimension:

Toxicity Detector Service:

- **Model Architecture:** RoBERTa-based transformer with 125 million parameters, fine-tuned on 159,000+ labeled comments
- **Classification Task:** Multi-label binary classification across six toxicity categories
- **Latency:** 50ms on GPU, 200ms on CPU
- **Accuracy:** 95% on held-out test set, 0.92 F1-score
- **Deployment:** 2-4 GPU instances in production, with CPU fallback

Hallucination Detector Service:

- **Model Ensemble:** Five-component ensemble combining semantic consistency, NLI contradictions, factuality checking, pattern analysis, and confidence calibration
- **Primary Models:** Sentence Transformers (all-MiniLM-L6-v2), NLI cross-encoder (DeBERTa)
- **Latency:** 300-500ms (multi-component processing)
- **Deployment:** 2-4 instances with load balancing

Jailbreak Detector Service:

- **Detection Methods:** Pattern matching, semantic similarity, behavioral analysis, adversarial detection, context-aware analysis
- **Threat Coverage:** Instruction override, role manipulation, system prompt injection, encoding tricks, prompt smuggling
- **Latency:** 100-200ms
- **False Negative Rate:** Target $\leq 5\%$ (emphasizing recall for security)

Bias Detector Service:

- **Analysis Dimensions:** Stereotype associations, demographic parity, representation balance, intersectional effects
- **Demographic Groups:** Gender, race, age, religion, profession, and intersections
- **Latency:** 500-800ms (includes demographic variant testing)
- **Deployment:** 2 instances (lower throughput due to computational intensity)

Service Characteristics:

All detector services share common architectural characteristics:

- **Stateless Design:** Services maintain no session state between requests. This enables horizontal scaling without shared state synchronization overhead or sticky session requirements.
- **Containerized Deployment:** Each service runs in Docker containers with resource limits and health checks.

- **RESTful API Contracts:** Services expose standard REST endpoints with Swagger/OpenAPI documentation.
- **Health Monitoring:** Expose `/health` and `/ready` endpoints for infrastructure monitoring.
- **Metrics Export:** Export Prometheus metrics for monitoring latency, throughput, errors, and model performance.
- **Graceful Shutdown:** Support drain requests during shutdown, preventing abrupt termination during active processing.

Component 4: Machine Learning Models (Core Intelligence)

The intelligence of the Equitas platform derives from carefully selected and fine-tuned machine learning models:

Transformer Models:

- **RoBERTa-Base:** 12-layer transformer with 125M parameters, effective for sequence classification tasks
- **DeBERTa-Base:** 12-layer transformer with disentangled attention, superior performance on NLI tasks
- **DistilBERT:** 6-layer distilled model for latency-critical applications (mobile/edge)
- **Sentence Transformers:** Specialized models for semantic similarity computation via dense embeddings

Embedding Systems:

- **Dense Embeddings:** Sentence-level embeddings (384-768 dimensions) enable semantic similarity searches
- **Sparse Embeddings:** Complementary sparse representations for interpretability and keyword-based searches
- **Cached Embeddings:** Pre-computed embeddings for 100,000+ reference phrases (jailbreak patterns, stereotypes, toxic comments)

NLI (Natural Language Inference) Models:

- **Classification Task:** ENTAILMENT, CONTRADICTION, NEUTRAL

- **Application:** Detecting contradictions within LLM responses indicating hallucination
- **Model:** cross-encoder-nli-deberta-v3-base, fine-tuned on MNLI dataset

Statistical Fairness Models:

- **Demographic Parity:** Measure equal decision rates across demographic groups
- **Equalized Odds:** Ensure equal true positive and false positive rates across groups
- **Calibration Metrics:** Verify that predicted probabilities match empirical frequencies

Component 5: Technology Stack (Infrastructure Foundation)

PyTorch and HuggingFace Transformers:

PyTorch provides the deep learning framework enabling efficient training and inference of transformer models:

- **Dynamic Computation Graphs:** Enable flexible model architectures and debugging
- **Distributed Training:** Support for distributed training across multiple GPUs and nodes
- **Inference Optimization:** Export models to ONNX and TensorRT for production optimization
- **Ecosystem:** Extensive libraries for data loading, preprocessing, and augmentation

HuggingFace Transformers provides pre-trained models and training utilities:

- **Model Hub:** 50,000+ pre-trained models covering diverse NLP tasks
- **Training Recipes:** Standardized fine-tuning scripts for various tasks
- **Evaluation Metrics:** Built-in implementations of BLEU, ROUGE, F1, and custom metrics

- **Tokenizers:** Efficient tokenization with support for special tokens and truncation strategies

FastAPI Framework:

FastAPI is the chosen backend framework due to superior performance and modern Python capabilities:

Throughput : 10,000 + requests/second per instance (4.18)

P99 Latency : 50ms (excluding detector overhead) (4.19)

Memory Overhead : 100 – 200MB per instance (4.20)

Key advantages:

- **Async/Await Native Support:** Non-blocking I/O for high concurrency
- **Automatic API Documentation:** OpenAPI/Swagger generation eliminates manual documentation burden
- **Type Safety:** Pydantic models provide runtime validation and IDE auto-completion
- **Dependency Injection:** Built-in support for clean, testable code architecture
- **Middleware Support:** Request/response interceptors for logging, authentication, and observability

SQLAlchemy ORM:

Provides database abstraction enabling flexible data persistence:

- **Database Agnostic:** Support for PostgreSQL, MySQL, SQLite without code changes
- **Connection Pooling:** Efficient connection management for high-concurrency environments
- **Schema Migration:** Alembic integration for reproducible schema evolution
- **Query Building:** Pythonic query construction with compile-time safety checking

AsyncIO Framework:

Python's built-in asynchronous I/O library enables concurrent processing:

- **Event Loop:** Single-threaded event loop coordinating concurrent tasks
- **Coroutines:** Lightweight concurrency mechanism without OS thread overhead
- **Futures and Tasks:** Promise-based concurrency patterns for complex workflows
- **Performance:** Enables 1000x more concurrent tasks than traditional threading

pytest Testing Framework:

Comprehensive testing ensures reliability and prevents regressions:

- **Unit Tests:** Test individual detector components with synthetic data
- **Integration Tests:** Verify end-to-end processing through the full pipeline
- **Adversarial Tests:** Specifically test against known attack patterns and edge cases
- **Performance Tests:** Verify latency and throughput meet SLAs
- **Regression Tests:** Ensure model updates do not degrade on reference datasets
- **Coverage Analysis:** Track code coverage with target of ≥90% coverage

4.1.3 Supporting Infrastructure

Data Storage and Caching**PostgreSQL (Primary Database):**

- **Use Cases:** Audit logs, user accounts, API keys, model metadata
- **Scalability:** Connection pooling supports 500+ concurrent connections
- **Replication:** Primary-replica replication for high availability
- **Backup:** Automated daily backups with PITR (Point-in-Time Recovery)

Redis (In-Memory Cache):

- **Cached Data:** Pre-computed embeddings, model outputs for repeated queries
- **Session Storage:** Temporary request context and intermediate results
- **Rate Limiting:** Atomic counters for per-user/per-IP rate limit enforcement
- **Performance:** Sub-millisecond access latency, 100,000+ ops/second throughput

Vector Database (e.g., Pinecone, Weaviate):

- **Purpose:** Semantic similarity searches for hallucination and bias detection
- **Scale:** Indexes 1 million+ reference embeddings
- **Latency:** 10-50ms semantic search latency
- **Maintenance:** Automatic index updates as new reference data arrives

Monitoring and Observability

Prometheus Metrics:

- **Request Latency Histograms:** Per-service latency distributions
- **Error Rates:** Track detector failures and edge case errors
- **Resource Utilization:** CPU, memory, GPU utilization per service
- **Model Performance:** Per-detector accuracy, precision, recall on recent predictions

Distributed Tracing (Jaeger):

- **Request Tracing:** Track complete request path through all services
- **Latency Attribution:** Identify performance bottlenecks
- **Error Tracking:** Correlate errors across service boundaries

Logging (ELK Stack):

- **Centralized Logging:** Aggregate logs from all services
- **Search and Analysis:** Elasticsearch enables rapid searching of 100GB+ logs
- **Alerting:** Automatic alerts for error spikes and anomalies

4.1.4 Deployment Architecture

Containerization

Each component is containerized independently with specific resource allocations:

Equitas SDK : 500MB image, 200MB runtime memory (4.21)

Guardian Backend : 1GB image, 500MB runtime memory (4.22)

Toxicity Detector : 2.5GB image (PyTorch + RoBERTa), 4GB runtime memory (4.23)

Hallucination Detector : 3.5GB image, 6GB runtime memory (4.24)

Jailbreak Detector : 2GB image, 3GB runtime memory (4.25)

Bias Detector : 2GB image, 4GB runtime memory (4.26)

Kubernetes Orchestration

Kubernetes manages container deployment and scaling:

- **Horizontal Pod Autoscaling:** Automatically scales services based on CPU and memory metrics
- **Service Discovery:** Kubernetes DNS enables dynamic service lookup
- **Configuration Management:** ConfigMaps and Secrets for environment-specific configuration
- **Health Management:** Liveness probes restart unhealthy pods; readiness probes route traffic only to healthy instances
- **Resource Limits:** CPU and memory limits prevent resource exhaustion
- **Rolling Updates:** Zero-downtime deployments with automatic rollback on failure

4.1.5 Security Architecture

Inter-Service Communication

- **mTLS (Mutual TLS):** All service-to-service communication encrypted with mutual authentication

- **Network Policies:** Kubernetes network policies restrict communication to explicitly allowed paths
- **Service Mesh:** Istio provides fine-grained traffic management and security policies

API Authentication and Authorization

- **API Key Authentication:** Each client receives unique API key scoped to specific operations
- **OAuth 2.0:** Support for OAuth 2.0 authorization for web applications
- **RBAC:** Role-Based Access Control for different user types (admin, developer, auditor)
- **Rate Limiting:** Per-user rate limits (e.g., 1000 requests/day) prevent abuse

Data Protection

- **Encryption at Rest:** AES-256 encryption for stored models and embeddings
- **Encryption in Transit:** TLS 1.3 for all network communication
- **Key Management:** HashiCorp Vault for centralized secret management
- **Audit Logging:** Immutable logs of all API access and safety decisions for compliance

4.1.6 System Integration Flow

The complete data flow through the Equitas platform follows this comprehensive path:

$$\text{Client Application} \xrightarrow[\text{HTTP/REST}]{\text{OpenAI-compatible}} \text{Equitas SDK} \quad (4.27)$$

$$\text{Equitas SDK} \xrightarrow[\text{validation}]{\text{Input}} \text{API Gateway} \quad (4.28)$$

$$\text{API Gateway} \xrightarrow[\text{authentication}]{\text{Rate limit check}} \text{Guardian Backend} \quad (4.29)$$

$$\text{Guardian Backend} \xrightarrow{\text{Parallel invocation}} \left\{ \begin{array}{l} \text{Toxicity Detector} \\ \text{Hallucination Detector} \\ \text{Jailbreak Detector} \\ \text{Bias Detector} \end{array} \right. \quad (4.30)$$

$$\text{Detectors} \xrightarrow{\text{Scores}} \text{Guardian Backend} \quad (4.31)$$

$$\text{Guardian Backend} \xrightarrow{\text{Aggregation}} \text{Score Aggregator} \quad (4.32)$$

$$\text{Score Aggregator} \xrightarrow{\text{Decision}} \text{API Gateway} \quad (4.33)$$

$$\text{API Gateway} \xrightarrow{\text{Audit logging}} \text{PostgreSQL} \quad (4.34)$$

$$\text{API Gateway} \xrightarrow{\text{Response}} \text{Equitas SDK} \quad (4.35)$$

$$\text{Equitas SDK} \xrightarrow{\text{Annotated response}} \text{Client Application} \quad (4.36)$$

This comprehensive architecture ensures reliability, scalability, security, and maintainability across all components of the AI safety platform.

4.2 Custom Toxicity Detection

4.2.1 Model Architecture

The toxicity detection module replaces OpenAI Moderation API with a custom transformer-based fine-tuned model. The primary architecture specifications are:

$$\text{Model : Unitary Toxic-BERT (RoBERTa-based)} \quad (4.37)$$

$$\text{Parameters : 125M} \quad (4.38)$$

$$\text{Input : Text sequences, max 512 tokens} \quad (4.39)$$

$$\text{Output : Multi-label binary classification, 6 categories} \quad (4.40)$$

The six toxicity categories are:

1. Toxic: General toxicity
2. Severe Toxic: Severe toxicity
3. Obscene: Obscene language
4. Threat: Threatening language
5. Insult: Insulting language
6. Identity Hate: Identity-based hate

4.2.2 Mathematical Formulation

Given input text x , the RoBERTa model outputs logits $\mathbf{z} \in \mathbb{R}^6$:

$$\mathbf{z} = f_{\text{RoBERTa}}(x) \in \mathbb{R}^6 \quad (4.41)$$

Sigmoid activation is applied to obtain probabilities for each toxicity category:

$$p_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}, \quad \forall i = 1, \dots, 6 \quad (4.42)$$

The toxicity score is computed as the maximum probability across all categories:

$$\text{toxicity_score} = \max_i p_i \quad (4.43)$$

4.2.3 Flagging Decision

Text is flagged as toxic if the toxicity score exceeds a configurable threshold:

$$\text{flagged} = \begin{cases} 1 & \text{if toxicity_score} \geq \tau_t \\ 0 & \text{otherwise} \end{cases} \quad (4.44)$$

where $\tau_t = 0.7$ is the default configurable threshold.

4.2.4 Training Data and Performance

The model is trained on three comprehensive datasets:

- Jigsaw Toxic Comment Classification Dataset (Kaggle)
- Civil Comments Dataset (Conversation AI)
- Wikipedia Toxic Comments Dataset

Performance Metrics

$$\text{Accuracy : 95\% on held-out test set} \quad (4.45)$$

$$\text{F1-Score (macro-averaged) : 0.92} \quad (4.46)$$

$$\text{Latency : 50ms on GPU, 200ms on CPU} \quad (4.47)$$

$$\text{Throughput : 100 requests/second (batch processing)} \quad (4.48)$$

4.3 Hallucination Detection

4.3.1 Overview

Hallucination detection employs a multi-component ensemble system to identify factual errors, contradictions, and unsupported claims. Five distinct components are integrated with weighted scoring.

4.3.2 Component 1: Semantic Consistency Check

The semantic consistency component measures alignment between prompt and response using cosine similarity:

$$\text{consistency_score} = 1 - \cos(\text{embed}(p), \text{embed}(r)) \quad (4.49)$$

where:

- p : prompt text
- r : response text
- $\text{embed}(\cdot)$: Sentence Transformer encoder (all-MiniLM-L6-v2)

Cosine similarity is computed as:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} \quad (4.50)$$

Threshold

Low consistency ($\text{consistency_score} < 0.5$) indicates potential hallucination.

4.3.3 Component 2: Contradiction Detection

The Natural Language Inference (NLI) model classifies sentence pairs from responses against prompts:

$$\text{NLI}(s_i, s_j) \in \{\text{ENTAILMENT}, \text{CONTRADICTION}, \text{NEUTRAL}\} \quad (4.51)$$

For response r with sentences $S = \{S_1, S_2, \dots, S_n\}$, the contradiction score is:

$$\text{contradiction_score} = \frac{1}{n(n-1)/2} \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{I}[\text{NLI}(S_i, S_j) = \text{CONTRADICTION}] \quad (4.52)$$

where $\mathbb{I}[\cdot]$ is the indicator function.

Model and Threshold

- Model: cross-encoder-nli-deberta-v3-base (DeBERTa-based)
- Threshold: Contradiction rate ≥ 0.2 indicates high hallucination risk

4.3.4 Component 3: Factuality Check

Factuality is assessed via semantic similarity to a knowledge base:

$$\text{factuality_score} = 1 - \max_{k \in K} \cos(\text{embed}(r), \text{embed}(k)) \quad (4.53)$$

where $K = \{k_1, k_2, \dots, k_m\}$ is the knowledge base with pre-computed embeddings.

Low similarity to trusted sources indicates high hallucination risk.

4.3.5 Component 4: Pattern-Based Detection

Overconfident language patterns are identified through predefined indicators:

$$\text{pattern_score} = \min \left(\frac{\# \text{ matches patterns}}{0.2}, 1.0 \right) \quad (4.54)$$

Detected Patterns

Patterns include definitive language without evidence:

- “definitely”, “absolutely”, “100%”, “guaranteed”
- “true”, “false”, “proven”
- “studies show”, “research proves”, “experts agree” (without citations)

4.3.6 Component 5: Confidence Calibration

Confidence calibration analyzes linguistic markers:

$$\text{confidence_score} = \frac{N_{\text{high}}}{N_{\text{high}} + N_{\text{low}}} \quad (4.55)$$

where:

- N_{high} : count of high-confidence markers (“definitely”, “proven”, “fact”)
- N_{low} : count of low-confidence markers (“maybe”, “perhaps”, “according to”)

High confidence score (≥ 0.6) combined with low factuality indicates hallucination risk.

4.3.7 Ensemble Scoring

The final hallucination score aggregates all component scores with configurable weights:

$$\text{hallucination_score} = \frac{1}{n} \sum_{i=1}^n w_i s_i \quad (4.56)$$

where:

- s_i : component scores (consistency, contradiction, factuality, pattern, confidence)

- w_i : component weights (default: equal weights, $w_i = 1/5$)
- $n = 5$: number of components

Flagging Decision

$$\text{flagged} = \begin{cases} 1 & \text{if hallucination_score} \geq 0.6 \\ 0 & \text{otherwise} \end{cases} \quad (4.57)$$

4.4 Advanced Jailbreak Detection

4.4.1 Overview

The jailbreak detection module implements a multi-layered defense combining pattern matching, semantic analysis, behavioral indicators, and adversarial detection.

4.4.2 Component 1: Pattern-Based Detection

Regular expression matching against known jailbreak patterns:

$$\text{pattern_score} = \min \left(\frac{\# \text{ matches patterns}}{0.3}, 1.0 \right) \quad (4.58)$$

Detected Patterns

- Instruction override: “ignore previous”, “all instructions”, “commands”
- Role manipulation: “pretend”, “act”, “roleplay”, “you are”
- System prompt injection: “system”, “INST”

4.4.3 Component 2: Semantic Similarity Detection

Measures similarity to known jailbreak examples:

$$\text{semantic_score} = \max_{j \in J} \cos(\text{embed}(x), \text{embed}(j)) \quad (4.59)$$

where:

- x : input text

- J : set of known jailbreak examples with pre-computed embeddings

Threshold

Similarity ≥ 0.7 indicates high jailbreak risk.

4.4.4 Component 3: Behavioral Analysis

Detection of behavioral indicators suggesting jailbreak attempts:

$$\text{behavioral_score} = \min\left(\frac{N_{\text{indicators}}}{\text{indicators threshold}}, 1.0\right) \quad (4.60)$$

where $\text{threshold} = 0.25$, and $N_{\text{indicators}}$ is the count of detected behavioral indicators.

Behavioral Indicators

- “bypass”, “override”, “exploit”
- “no restrictions”, “do anything”

4.4.5 Component 4: Context-Aware Detection

Historical analysis of user attempts:

$$\text{context_score} = \min\left(\frac{N_{\text{previous attempts}}}{\text{threshold}}, 0.8\right) \quad (4.61)$$

where $\text{threshold} = 0.2$.

Users with previous jailbreak attempts are classified as higher risk.

4.4.6 Component 5: Adversarial Detection

Detection of encoding tricks and obfuscation techniques:

$$\text{adversarial_score} = \min\left(\sum_{t \in T} w_t \cdot \mathbb{K}[\text{detect}_t(x)], 1.0\right) \quad (4.62)$$

where:

- T : set of encoding techniques (URL encoding, hex encoding, Unicode tricks, excessive whitespace)
- w_t : weight for encoding type t

Encoding Detection Weights

$$w_{\text{URL}} = 0.3 \quad (4.63)$$

$$w_{\text{hex}} = 0.3 \quad (4.64)$$

$$w_{\text{unicode}} = 0.2 \quad (4.65)$$

$$w_{\text{whitespace}} = 0.2 \quad (4.66)$$

4.4.7 Ensemble Scoring

Final jailbreak score combines all components with fixed weights:

$$\text{jailbreak_score} = \sum_{i=1}^5 w_i s_i \quad (4.67)$$

where component weights are:

$$w_{\text{pattern}} = 0.3 \quad (4.68)$$

$$w_{\text{semantic}} = 0.3 \quad (4.69)$$

$$w_{\text{behavioral}} = 0.2 \quad (4.70)$$

$$w_{\text{context}} = 0.1 \quad (4.71)$$

$$w_{\text{adversarial}} = 0.1 \quad (4.72)$$

Flagging Decision

$$\text{flagged} = \begin{cases} 1 & \text{if } \text{jailbreak_score} \geq 0.6 \\ 0 & \text{otherwise} \end{cases} \quad (4.73)$$

4.5 Enhanced Bias Detection

4.5.1 Overview

Comprehensive bias detection employs statistical fairness metrics, stereotype association analysis, and demographic parity testing across multiple demographic groups.

4.5.2 Component 1: Stereotype Association Detection

Measures semantic similarity to known stereotype embeddings:

$$\text{stereotype_score} = \max_{g \in G} \max_{s \in S_g} \cos(\text{embed}(r), \text{embed}(s)) \quad (4.74)$$

where:

- r : response text
- G : set of demographic groups (gender, race, age, religion, profession)
- S_g : set of stereotype phrases for group g

Pre-computed Stereotypes

- Gender: “women are emotional”, “men are aggressive”, “women are bad at math”
- Race: “Black people are criminals”, “Asian people are good at math”
- Age: “old people are slow”, “young people are lazy”

Threshold

Similarity ≥ 0.7 indicates stereotype reinforcement.

4.5.3 Component 2: Demographic Parity Testing

Paired prompt testing across demographic variants:

$$\text{parity_score} = \text{Var}(f(r_1), f(r_2), \dots, f(r_n)) \quad (4.75)$$

where:

- f : outcome function (e.g., positive sentiment, recommendation score)
- r_i : responses generated from demographic variants

Equalized Odds

$$\text{equalized_odds} = \max_{d, d' \in D} |\text{TPR}_d - \text{TPR}_{d'}| + |\text{FPR}_d - \text{FPR}_{d'}| \quad (4.76)$$

where:

- TPR_d : True Positive Rate for demographic d
- FPR_d : False Positive Rate for demographic d
- D : set of demographic variants

Ideal equalized odds = 0 (perfect fairness).

4.5.4 Component 3: Demographic Balance

Measures variance in demographic mentions:

$$\text{balance_score} = \text{Var}(c_d \mid d \in D) \quad (4.77)$$

where c_d is the count of mentions of demographic d in the response.

High variance indicates imbalanced representation and potential bias.

4.5.5 Component 4: Representation Parity

Measures equality of demographic representation:

$$\text{representation_parity} = 1 - \frac{\max_{d \in D} c_d - \min_{d \in D} c_d}{\sum_{d \in D} c_d} \quad (4.78)$$

Ideal representation parity = 1 (equal representation).

4.5.6 Component 5: Intersectional Bias Analysis

Detects compounding demographic mentions:

$$\text{intersectional_score} = \frac{N_{\text{intersectional mentions}}}{N_{\text{total mentions}}} \quad (4.79)$$

High intersectional mentions (≥ 3) may indicate compounding bias.

4.5.7 Overall Bias Score

$$\text{bias_score} = \max(\text{stereotype_score}, \text{parity_score}, \text{balance_score}) \quad (4.80)$$

Flagging Decision

$$\text{flagged} = \begin{cases} 1 & \text{if bias_score} \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \quad (4.81)$$

4.6 Prompt Injection Prevention

4.6.1 Overview

Multi-layered prevention system combining input sanitization, context isolation, encoding detection, and length validation to prevent prompt injection attacks.

4.6.2 Component 1: Input Sanitization

Pattern removal while preserving semantic meaning:

$$x_{\text{sanitized}} = \text{removePatterns}(x, P) \quad (4.82)$$

where P is the set of injection patterns.

Patterns Removed

- System markers: “system”, “INST”, “system:”
- Separators: “—”, “'”, “||”
- Role markers: “user:”, “system:”, “assistant:”

4.6.3 Component 2: Context Isolation

Strict separation of user input from system prompt:

$$x_{\text{isolated}}, s_{\text{clean}} = \text{isolate}(x, s) \quad (4.83)$$

where s is the system prompt and s_{clean} removes system prompt markers.

4.6.4 Component 3: Encoding Detection

Detection of obfuscation techniques:

$$\text{encoding_score} = \sum_{t \in T} w_t \cdot \mathbb{K}[\text{detect}_t(x)] \quad (4.84)$$

where T includes URL encoding, hex encoding, and Unicode tricks.

Encoding Weights

$$w_{\text{URL}} = 0.3 \quad (4.85)$$

$$w_{\text{hex}} = 0.3 \quad (4.86)$$

$$w_{\text{unicode}} = 0.2 \quad (4.87)$$

4.6.5 Component 4: Length Validation

Detection of excessive length:

$$\text{length_flag} = \begin{cases} 1 & \text{if } \text{length}(x) > 10000 \\ 0 & \text{otherwise} \end{cases} \quad (4.88)$$

4.6.6 Sanitization Pipeline

Complete sanitization process:

$$x_{\text{final}} = \text{normalize}(\text{decode}(\text{removePatterns}(x))) \quad (4.89)$$

where:

- removePatterns: removes injection markers
- decode: decodes encoding tricks
- normalize: normalizes Unicode and whitespace

4.7 Dataset Testing Framework

4.7.1 Toxicity Detection Datasets

Jigsaw Toxic Comment Classification

Size : 159,000 comments (4.90)

Categories : Toxic, Severe Toxic, Obscene, Threat, Insult, Identity Hate (4.91)

Split : 80% train, 10% validation, 10% test (4.92)

Civil Comments Conversation AI

Size : 2M comments (4.93)

Categories : Toxicity, Identity attacks, Insults, Threats (4.94)

Metadata : Demographics, identity groups (4.95)

Wikipedia Toxic Comments

Size : 180,000 comments (4.96)

Categories : Toxic, Severe toxic, Obscene, Threat, Insult, Identity hate (4.97)

4.7.2 Hallucination Detection Datasets

FEVER (Fact Extraction and VERification)

Size : 185,000 claims (4.98)

Labels : SUPPORTED, REFUTED, NOT ENOUGH INFO (4.99)

Use : Factuality checking (4.100)

XSum (BBC News Summaries)

Size : 226,000 summaries (4.101)

Use : Consistency checking (4.102)

TruthfulQA

Size : 800 questions (4.103)

Focus : Truthfulness and hallucinations (4.104)

4.7.3 Jailbreak Detection Datasets**Custom Jailbreak Prompts Dataset**

Size : 500 jailbreak examples (4.105)

Categories : Instruction override, Role manipulation, Adversarial (4.106)

HarmfulQA Safety Benchmarks

Size : 1,000 harmful prompts (4.107)

Categories : Violence, Self-harm, Illegal activities (4.108)

4.7.4 Bias Detection Datasets**BOLD (Bias in Open-Ended Language Generation)**

Size : 23,000 prompts (4.109)

Demographics : Gender, Race, Religion, Profession (4.110)

StereoSet

Size : 17,000 sentences (4.111)

Focus : Stereotype detection (4.112)

CrowS-Pairs

Size : 1,500 sentence pairs (4.113)

Focus : Social bias across demographics (4.114)

4.8 Evaluation Metrics**4.8.1 Classification Metrics****Accuracy**

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (4.115)$$

Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (4.116)$$

Recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (4.117)$$

F1-Score

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4.118)$$

Area Under ROC Curve

$$\text{AUC-ROC} = \int_0^1 \text{TPR}(\text{FPR}) d\text{FPR} \quad (4.119)$$

4.8.2 Fairness Metrics**Demographic Parity**

$$\text{DP} = P(\hat{Y} = 1 \mid A = a) - P(\hat{Y} = 1 \mid A = b) \quad (4.120)$$

where A is the protected attribute.

Equalized Odds

$$\text{EO} = P(\hat{Y} = 1 \mid A = a, Y = y) - P(\hat{Y} = 1 \mid A = b, Y = y) \quad (4.121)$$

Calibration

$$\text{Calibration Error} = \mathbb{E}[Y \mid P = p] - p \quad (4.122)$$

Ideal calibration error = 0 (perfect calibration).

4.8.3 Performance Targets**Latency Targets**

$$\text{Toxicity Detection : 50ms (GPU), 200ms (CPU)} \quad (4.123)$$

$$\text{Hallucination Detection : 300ms (multi-component)} \quad (4.124)$$

$$\text{Jailbreak Detection : 100ms} \quad (4.125)$$

$$\text{Bias Detection : 500ms (with demographic testing)} \quad (4.126)$$

Throughput

$$\text{Batch Processing : 100 requests/second} \quad (4.127)$$

$$\text{Concurrent Requests : 50 parallel requests} \quad (4.128)$$

$$\text{GPU Utilization : 80\% under load} \quad (4.129)$$

Accuracy Targets

$$\text{Toxicity F1 : 0.92} \quad (4.130)$$

$$\text{Hallucination AUC : 0.85} \quad (4.131)$$

$$\text{Jailbreak Recall : 0.95 (high recall for security)} \quad (4.132)$$

$$\text{Bias Detection F1 : 0.80} \quad (4.133)$$

4.9 Cross-Dataset Generalization

To evaluate model generalization across diverse evaluation sets:

$$\text{GeneralizationScore} = 1 - \frac{1}{n} \sum_{i=1}^n \sqrt{(F_{\text{model}, D_i} - \bar{F})^2} \quad (4.134)$$

where:

- D_i : evaluation dataset i
- F_{model, D_i} : F1-score on dataset i
- \bar{F} : average F1-score across all datasets
- n : number of datasets

Chapter 5

Data Set Details

The robustness and reliability of the Equitas AI safety and monitoring platform are fundamentally dependent on the quality and diversity of the datasets used for training its various detection modules. [file:1] This chapter details the datasets leveraged for developing the toxicity, hallucination, bias, and jailbreak detection models, outlining the rationale for their selection and their specific characteristics.

5.1 Toxicity Detection Datasets

To train the toxicity detection module, a combination of established and novel datasets was employed to ensure broad coverage of different types of toxic content.

- **Jigsaw Toxic Comment Classification Challenge:** This foundational dataset from a Kaggle competition provides a large corpus of Wikipedia comments labeled for various types of toxicity, including toxic, severe toxic, obscene, threat, insult, and identity hate. [web:23] It is a crucial resource for training a baseline model for general-purpose toxicity detection. [web:23]
- **DeToxy:** The first publicly available toxicity annotated dataset for the English language. [web:15] Sourced from various openly available speech databases, DeToxy consists of over 2 million utterances. [web:15]
- **Multilingual Toxicity Dataset:** This dataset expands the scope of toxicity detection beyond English, with data in several languages including Russian, Italian, French, and more. [web:27] It is essential for developing a more globally applicable toxicity detection model. [web:27]

5.2 Hallucination Detection Datasets

Detecting hallucinations in LLMs is a nuanced task that requires specialized datasets. The following datasets were chosen for this purpose:

- **DefAn: Definitive Answer Dataset:** This benchmark dataset includes over 75,000 prompts across eight domains, designed to elicit concise and factual answers, making it ideal for identifying when an LLM deviates from known facts. [web:20]
- **HalluRAG Dataset:** This dataset is specifically designed for detecting closed-domain hallucinations, where the LLM is given context to answer a question. [web:28] It helps in training models that can identify when the LLM generates information not present in the provided context. [web:28]
- **PsiloQA:** A large-scale dataset for multilingual span-level hallucination detection. [web:24] It provides fine-grained supervision across 14 languages, enabling the training of detectors with strong cross-lingual generalization capabilities. [web:24]

5.3 Bias Detection Datasets

Addressing bias in LLMs is critical for ensuring fairness. The following datasets are instrumental in training and evaluating the bias detection module:

- **Equity-Eval:** This dataset is designed to evaluate social and demographic biases in language models. It contains prompts that are likely to elicit biased responses, allowing for the quantification and mitigation of such biases.
- **BOLD (Bias in Open-Ended Language Generation):** This dataset provides a large number of prompts for evaluating bias across various demographic groups, such as gender, race, and religion. [web:17] It is a key resource for understanding and addressing stereotypical and other harmful biases in LLM outputs.

5.4 Jailbreak and Safety Datasets

To defend against adversarial attacks and "jailbreaks," the following datasets are used to train the safety and moderation models:

- **WildJailbreak:** A large-scale safety training dataset with 262,000 training examples. [web:11] It is a significant update to prior public safety training resources and is crucial for training models to recognize and block jailbreak attempts. [web:11]
- **AEgis 2.0:** A diverse AI safety dataset that helps train "guard" models against a new taxonomy of harm categories. [web:22]
- **YAIR (Youth AI Risk):** The first benchmark dataset designed to evaluate and improve the safety of youth-LLM interactions. [web:14] It includes 12,449 annotated conversation snippets, each labeled using a three-tier risk taxonomy. [web:14]

Figure 5.1: A diagram illustrating the flow of data through the various detection modules of the Equitas platform.

Chapter 6

Conclusion and Future Scope

This chapter synthesizes the accomplishments of the Equitas project, offers a transparent assessment of its current limitations, and outlines a strategic roadmap for its future development. [file:1] The discussion aims to consolidate the project's contributions to AI safety and chart a course for continued innovation and impact. [file:1]

6.1 Summary

The Equitas project has culminated in the creation of a sophisticated, independent AI safety platform engineered for the real-time moderation of Large Language Models (LLMs). [file:1] This initiative successfully transitioned from a dependency on third-party moderation APIs, like OpenAI's, to a proprietary, multi-layered solution capable of identifying and mitigating a wide spectrum of AI-generated risks. [file:1] The core of the system is the "Guardian Backend API," which functions as a seamless, drop-in replacement for the OpenAI SDK, orchestrating a suite of custom-built detection modules for Toxicity, Hallucination, Jailbreak attempts, and Bias. [file:1]

A significant technical achievement of this project is the deep integration of explainability models, specifically SHAP and LIME. [file:1] These tools provide crucial token-level importance scores and detailed analytics for safety flags, demystifying model decisions and addressing a critical transparency gap in the AI safety market. [file:1] The platform is architected with a modern technology stack including PyTorch, HuggingFace Transformers, and Sentence Transformers, ensuring state-of-the-art performance. [file:1] By positioning itself as a developer-first safety wrapper with a rapid, 5-minute setup, Equitas makes robust, explainable AI safety accessible, offering a compelling alternative to existing solutions through its custom architecture and flexible pricing model. [file:1]

6.2 Limitation

Despite its advanced capabilities, the Equitas platform has several inherent limitations that must be acknowledged. [file:1] The efficacy of the detection modules is fundamentally tied to the diversity and scope of their training data, which means novel or "zero-day" adversarial attacks and nuanced forms of harmful content may go undetected. [file:1] The rapidly evolving nature of jailbreak techniques presents a continuous challenge, requiring constant vigilance and model retraining to maintain robustness. [web:5]

Furthermore, while the custom models for toxicity, hallucination, and bias are highly effective, they may not yet possess the sheer scale and breadth of coverage offered by large, commercially-funded systems that have been trained on vastly larger datasets over many years. [file:1] The computational overhead of running an ensemble of deep learning models in a real-time pipeline can also introduce latency and increase operational costs, which may be a constraint for high-throughput applications. [file:1] A primary current limitation is the platform's focus on the English language, which restricts its utility in global and multilingual environments. [file:1]

6.3 Future Work

The future roadmap for Equitas is focused on addressing its current limitations and expanding its capabilities to meet emerging challenges in AI safety. [file:1] A key priority is to enhance the platform's resilience and adaptability through a more dynamic and incremental development approach. [web:6]

Key strategic initiatives for future development include:

- **Cross-Lingual Generalization:** A primary goal is to extend the platform's capabilities beyond English. [file:1] This will involve training new models on multilingual datasets and exploring cross-lingual transfer learning techniques to build a truly global safety solution. [web:4]
- **Advanced Adversarial Defense:** To counter the evolving threat of jailbreaks and other adversarial attacks, we will invest in proactive risk assessment and red-teaming. [web:5] Future work will involve implementing more sophisticated adversarial training regimes and developing dedicated detectors for subtle attack patterns, such as those using encoding or context manipulation. [file:1][web:5]
- **Enhanced Fairness and Bias Mitigation:** We will move beyond detec-

tion to proactive bias mitigation. [file:1] This includes developing tools for intersectional bias analysis and integrating techniques to ensure demographic parity and fair representation in model outputs, aligning with emerging industry standards for responsible AI. [file:1][web:3]

- **Multimodal Safety Assurance:** The next frontier for AI safety involves extending protections to multimodal inputs and outputs (e.g., images, audio). [web:6] We plan to research and develop new modules capable of detecting risks in non-textual data, ensuring comprehensive safety across different modes of interaction.
- **Collaboration and Standardization:** We will seek to align with global AI safety standards and frameworks, such as those proposed by NIST and other international bodies. [web:9][web:2] Engaging in partnerships with the broader AI community will be crucial for sharing insights, contributing to harmonized safety protocols, and ensuring the platform remains at the forefront of responsible AI development. [web:6]

References

- [1] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.