✍️ Final Project Documentation: Guardian-NLP

1. Project Overview

"Guardian-NLP" is an advanced, end-to-end MLOps project that demonstrates the complete lifecycle of a machine learning application. The system is designed to detect toxic speech within video or audio files via its frontend interface, "AudioTox Guard." It functions by accepting a media file, transcribing the spoken content to text using a sophisticated speech-to-text model, and then analyzing that text for toxicity with a custom-trained classifier. This document outlines the entire workflow, showcasing a professional-grade process that includes a multi-stage ML pipeline, a microservices-based application architecture, data and model versioning, containerization, local Kubernetes orchestration, and a full CI/CD pipeline for automated builds and publishing.

---

2. System Architecture

The project is built on a modern microservices architecture designed for scalability, portability, and automation. The architecture is composed of several distinct layers:

1. **ML Pipeline**: The core logic involves a two-stage process. An audio/video file is first processed by a **Speech-to-Text (STT)** model (OpenAI's Whisper) to generate a text transcript. This text is then fed into a custom-trained **toxic text classification** model to get a prediction.
2. **Application Layer**: The system is split into two distinct services that communicate over a network:
    - **Backend API**: A **Flask** application that exposes the toxicity classification model via a /predict REST endpoint. It is the "brain" that performs the analysis.
    - **"AudioTox Guard" Frontend**: An interactive **Streamlit** web application that provides a user-friendly interface for uploading files, viewing transcriptions, and seeing the final analysis results.
3. **Containerization**: Both the backend API and the frontend UI are individually containerized using **Docker**. A **Docker Compose** file is used to define, manage, and

network these containers for a cohesive local development experience.

4. **Orchestration**: The containerized application is designed for a production-like environment using **Kubernetes**. Configuration files are provided to deploy both services to a local cluster (Minikube), managing scaling and networking automatically.
5. **CI/CD Automation**: A **GitHub Actions** workflow provides a fully automated pipeline. On every push to the GitHub repository, the pipeline automatically builds fresh Docker images for both the backend and frontend and pushes them to **Docker Hub**.

---

3. Technology Stack

This project utilizes a comprehensive stack of modern tools standard in the MLOps and cloud-native software development industries.

- **Machine Learning & Audio Processing**
  - **Python**: The core programming language for all development.
  - **Scikit-learn**: Used for training the foundational text classification model (LogisticRegression) and for text feature extraction (TfidfVectorizer).
  - **Pandas**: For efficient data loading and manipulation of the initial training dataset.
  - **OpenAI Whisper**: A state-of-the-art, pre-trained model used for high-accuracy speech-to-text transcription.
  - **PyAV**: A Python library used for audio and video processing, specifically for extracting audio from video files.
  - **FFmpeg**: A critical system-level dependency used by PyAV and Whisper to decode and handle various media formats.
- **Application & API Development**
  - **Flask**: A lightweight web framework used to build the robust, standalone backend API.
  - **Streamlit**: A modern Python framework used to rapidly create the interactive and user-friendly "AudioTox Guard" web UI.
- **Versioning & Code Management**
  - **Git**: The distributed version control system for tracking all source code.
  - **DVC (Data Version Control)**: Used to version large model artifacts (e.g., .pkl files) that are too large for a Git repository, ensuring reproducibility.
- **Containerization & Orchestration**
  - **Docker**: The platform used to package the frontend and backend applications into portable, self-contained containers.
  - **Docker Compose**: A tool for defining and running multi-container Docker applications, used here to manage the local development environment.
  - **Kubernetes (Minikube)**: The container orchestration platform used to deploy, scale, and manage the containerized application in a production-like local cluster.

- **CI/CD & Hosting**
    - **GitHub**: The central repository for hosting source code and managing the project.
    - **GitHub Actions**: The automation engine used to create the CI/CD pipeline, automatically building and testing code on every push.
    - **Docker Hub**: The public container registry used to store and distribute the versioned Docker images built by the CI/CD pipeline.

---

4. Model Development Phase

4.1. Text Classifier Training

The project's analytical foundation is a toxic text classifier trained on the "Jigsaw Toxic Comment Classification" dataset. A LogisticRegression model was trained on TF-IDF features, achieving **95.66% accuracy** on the validation set.

4.2. Model Versioning with DVC

The trained model (model.pkl) and the TF-IDF vectorizer (vectorizer.pkl) were versioned using DVC to ensure reproducibility. This allows the project to track large model files without bloating the Git repository. A local remote was configured for development.

Bash

```bash
# DVC Commands
git init
dvc init
dvc remote add -d localremote "path/to/local/storage"
dvc add model.pkl vectorizer.pkl
git commit -m "feat: track models with dvc"
dvc push
```

5. Application Layer

5.1. Backend API (Guardian-NLP)

A Flask API was built to serve the text classification model. It loads the versioned model files and exposes a single /predict endpoint.

Python

```python
# Code Snippet: app.py (abridged)
from flask import Flask, request, jsonify
# ... loading model and vectorizer from .pkl files ...
app = Flask(__name__)
@app.route('/predict', methods=['POST'])
def predict():
    # ... logic to get text, vectorize, predict, and return JSON ...
    pass
```

5.2. "AudioTox Guard" UI (Frontend)

An interactive UI was built with Streamlit. This application allows users to upload video/audio files, which it then processes.

Python

```python
# Code Snippet: ui.py (abridged)
import streamlit as st
import whisper
import requests
import av

# ... UI layout and file upload logic ...

# Transcribe audio with Whisper
result = model.transcribe(audio_path)
transcribed_text = result["text"]

# Call the backend API
api_url = "http://backend:5000/predict"
response = requests.post(api_url, json={"comment": transcribed_text})
```

---

6. Containerization & Orchestration

6.1. Dockerization

Separate Dockerfile and Dockerfile.ui files were created to containerize the backend and frontend services. A docker-compose.yml file was created to orchestrate both services in a networked environment for local development.

YAML

```yaml
# Code Snippet: docker-compose.yml
version: '3.8'
services:
  backend:
    build:
      context: .
      dockerfile: Dockerfile
    ports: ["5000:5000"]
  frontend:
    build:
      context: .
      dockerfile: Dockerfile.ui
    ports: ["8501:8501"]
    depends_on: [backend]
```

6.2. Kubernetes Deployment

Kubernetes configuration files (k8s-backend.yaml and k8s-frontend.yaml) were written to define the deployment on a Kubernetes cluster.

Bash

```
# Kubernetes Commands
minikube start --driver=docker
minikube image load guardian-nlp-api guardian-nlp-frontend
kubectl apply -f k8s-backend.yaml
kubectl apply -f k8s-frontend.yaml
```

---

7. CI/CD Automation with GitHub Actions

A CI/CD pipeline was created in .github/workflows/main.yaml. The workflow contains two parallel jobs, build-backend and build-frontend. On every push to main, these jobs automatically build the respective Docker images and push them to separate Docker Hub repositories.

YAML

```yaml
# Code Snippet: main.yaml (abridged)
jobs:
  build-backend:
    # ... steps to build and push backend image to docker hub ...
  build-frontend:
    # ... steps to build and push frontend image to docker hub ...
```

Congratulations on completing this extensive project!