Design Document: Continuous Training and Deployment Pipeline for Transformer-based Text Classification Model

1. Overview

This design outlines a continuous training and deployment pipeline for a transformer-based text classification model. The pipeline can be deployed either locally or on AWS and supports both live and batch inference. Retraining is triggered automatically based on a defined staleness metric (accuracy drop), ensuring the model remains effective over time.

---

2. Architecture Components

A. Monitoring Model Staleness

- **Metric**: Accuracy is used to determine model staleness. If the model's accuracy drops below a threshold of 80% (or another specified value), retraining is triggered.
- **Local Setup**: MLflow monitors performance metrics, logging each training session. Accuracy degradation is assessed based on recent MLflow logs.
- **AWS Setup**: Model accuracy metrics are sent to CloudWatch. A Lambda function compares current accuracy with the threshold, triggering retraining if necessary.

B. Data Handling and Preprocessing

- **Dataset**: Text data labeled for sentiment classification or any other categorization.
- **Preprocessing**: Text tokenization is handled using Hugging Face's `AutoTokenizer`, which prepares the dataset for the transformer model by padding and truncating text sequences.

C. Training and Retraining Pipeline

- **Model**: Uses a transformer model (e.g., BERT or DistilBERT) for text classification.
- **Training Logic**:
    - **Local**: The model is trained in Python, and training parameters/metrics are logged in MLflow. When retraining is triggered, the process is repeated.
    - **AWS**: A `train.py` script is executed on AWS SageMaker, with data pulled from and saved to S3. Model artifacts are versioned and stored in S3, and the model can be retrained if accuracy drops.

D. Inference

- **Live Inference**:

    - **Local**: Served via a Flask API.
    - **AWS**: Deployed through a SageMaker endpoint.
- **Batch Inference**:
    - **Local**: A batch inference script loads data, predicts, and stores results.
    - **AWS**: Batch inference is handled through SageMaker's batch transform jobs, saving results in S3.

---

3. Infrastructure Overview

Local Deployment

- **MLflow**: Tracks model parameters, metrics, and artifacts locally.
- **Flask API**: Provides real-time predictions.
- **Email Alerts**: Sends alerts through Gmail if retraining is required.

AWS Deployment

- **AWS S3**: Stores data, model checkpoints, and inference outputs.
- **SageMaker**: Manages both training and deployment.
- **Lambda and CloudWatch**: Monitors model accuracy and triggers retraining.
- **SNS**: Sends alerts to notify stakeholders if model retraining is triggered.

---

4. Cost Optimization Strategies

- **Local**:
    - **Batch Inference**: Only runs as needed to reduce computational load.
    - **Retraining Threshold**: Limits retraining frequency based on a significant accuracy drop.
- **AWS**:
    - **Instance Selection**: Uses cost-effective `ml.m5.large` instances for inference and `ml.m5.xlarge` for training.
    - **On-Demand Batch Processing**: Limits the usage of SageMaker resources to when batch inference is necessary.
    - **S3 Data Storage**: Data lifecycle policies clean up old models and datasets to reduce storage costs.