Here's a comprehensive set of **project guidelines** for building a **Continuous ML Training, A/B Testing, and Rollback Pipeline**. These guidelines are broken into phases, detailing core components and best practices across **data science**, **DevOps**, and **software engineering** domains.

Problem Statement: Create a pipeline for continuous training, A/B testing, and automated rollbacks of machine learning models. This project showcases your ability to bridge the gap between data science, programming and operations.
Key features:

- Implement automated model training and validation
- Set up A/B testing for model performance comparison
- Implement automated rollbacks if model accuracy drops
- Design a scalable inference API

Bonus: Implement model versioning and experiment tracking

Project Guidelines: Continuous ML Training, A/B Testing, and Rollbacks

Objective

Build a robust, production-grade ML pipeline that:

- Trains models automatically
- A/B tests new models against current production models
- Automatically rolls back underperforming models
- Serves predictions via a scalable inference API
- (Bonus) Tracks experiments and versions models systematically

1. Environment Setup

Tools & Technologies (suggested)

- **Language:** Python 3.10+
- **ML Framework:** scikit-learn, TensorFlow, or PyTorch
- **Workflow Orchestration:** Apache Airflow / Prefect / MLflow
- **Containerization:** Docker
- **Model Serving:** FastAPI or Flask + Gunicorn/Uvicorn
- **Deployment:** Kubernetes (GKE/EKS), or Docker Compose for local dev
- **CI/CD:** GitHub Actions / GitLab CI / Jenkins
- **Monitoring:** Prometheus + Grafana, Sentry
- **Versioning:** DVC, MLflow, or custom S3 + metadata
- **Storage:** S3/GCS for artifacts and datasets

2. Automated Model Training & Validation

Steps

- Automate data preprocessing (include data validation checks)
- Schedule recurring training jobs (via Airflow/Prefect)
- Train models using latest production data
- Evaluate on a validation set with multiple metrics (e.g., accuracy, F1, precision)
- Store trained models with metadata (training time, data hash, metrics)

Deliverables

- `train.py` script or notebook
- `evaluate.py` with clear metric logging
- Automated training pipeline
- Trained model artifacts

3. A/B Testing Infrastructure

Strategy

- Route inference traffic between current (Model A) and candidate (Model B)
- Use feature flags or load-balancing (e.g., Envoy, Istio, NGINX)
- Log user responses, predictions, and ground truth
- Compare live performance based on actual user behavior

Deliverables

- Load balancer config for A/B routing
- Traffic logging mechanism (e.g., Kafka or logging API)
- A/B evaluation script for analyzing live metrics

4. Automated Rollbacks

Implementation

- Set performance thresholds (e.g., drop in accuracy > 5%)
- Trigger rollback if Model B underperforms consistently
- Alert stakeholders (via email, Slack webhook)
- Revert deployment to last known good model (via version tag)

Deliverables

- `rollback.py` with threshold rules
- Alert/notification module
- Deployment scripts with rollback logic

5. Inference API (Model as a Service)

Requirements

- Build a RESTful API
- Support health checks, batch predictions, and logging
- Load model version dynamically based on env var or config

6. Bonus: Model Versioning & Experiment Tracking

Tools

- Use MLflow or DVC for tracking experiments
- Maintain a versioned model registry (e.g., MLflow Model Registry or custom S3 tags)
- Auto-tag model versions with git hash, training data hash, and metrics

Deliverables

- Versioned metadata schema
- Experiment dashboard

7. Testing & Validation

- Unit tests for training, evaluation, and inference
- Integration tests for API endpoints
- Load tests (e.g., using Locust or k6) for inference scalability
- Continuous testing in CI/CD pipeline

8. Project Structure (Suggested)

```
ml_pipeline_project/
│
├── data/                     # Input data (or reference to S3)
├── models/                   # Trained and versioned models
├── pipelines/
│   ├── train_model_dag.py    # Airflow DAG or Prefect flow
│   └── rollback.py           # Rollback logic
├── inference/
│   ├── inference_api.py      # FastAPI serving logic
│   └── Dockerfile
├── scripts/
│   ├── train.py
│   ├── evaluate.py
│   ├── compare_ab.py
│   └── track_experiments.py
├── tests/
│   ├── test_train.py
│   ├── test_inference.py
│   └── test_rollback.py
├── configs/
│   └── config.yaml           # Thresholds, paths, model versions
├── .github/workflows/        # CI/CD definitions
├── README.md
└── requirements.txt
```

Final Notes

- Ensure **observability**: Log all predictions, model versions, and feature inputs
- Use **feature stores** if production data and training data diverge
- Enforce **reproducibility**: Pin package versions, record dataset versions
- Maintain **security best practices**: Secure API endpoints and artifacts
- 9 week project divided into 3 checkpoints at 3 week points.
- Mentors will be assessing at each checkpoint.

Happy Hacking!