AIBoMGen: Hugging Face Training API with AIBoM gen

Overview

AIBoMGen is a distributed system for training AI models with a focus on **trustability**, integrity, reproducibility, and artifact management. The system ensures that the training process is fully auditable and tamper-proof by generating an **AI Bill of Materials (AIBoM)**, which includes cryptographic attestations of the training environment, inputs, and outputs.

Key features:

- **Dataset Integrity Verification**: Hashing datasets and inputs to ensure consistency and prevent tampering.
- **AIBoM Generation with Attestations**: Captures metadata, input/output hashes, and environment details, signed with a private key to ensure authenticity and trustability.
- **Tamper-Proof Workflow**: The AIBoM generation process is automated and cannot be altered by the user, ensuring the integrity of the training pipeline.
- **Secure API**: The system uses a controlled API to prevent arbitrary code execution, ensuring that users cannot tamper with the training process, shut down containers, or bypass the AIBoM generation.
- **Containerized Workers**: Training jobs are executed in isolated Docker containers, providing an additional layer of security and ensuring that the system remains stable and safe from user interference.
- **Distributed Training**: Uses Celery workers for scalable training jobs.
- **MinIO Integration**: Handles file storage and retrieval for datasets, models, and logs.

Requirements

Prerequisites (You Need to Install)

- **Docker**: For containerized deployment.
- **Docker Compose**: To orchestrate the containers.
- **Environment Variables**: Create a `.env` file in the root directory to configure the system (see **Setup** for details).
- **Private Key**: A valid `private-key.pem` file must be placed in the worker container/directory. This key is used to cryptographically sign the AIBoM as an authority of the platform (e.g., the EU for compliance with the EU AI Act).

Handled by Docker (API and Worker Containers)

API Container

- **Python 3.9**: For FastAPI and Celery integration.
- **FastAPI**: For building the API.
- **Uvicorn**: For running the FastAPI application.
- **Celery**: For task scheduling and distributed execution.
- **Pydantic**: For data validation and settings management.
- **Requests**: For making HTTP requests.

- **Python-Multipart**: For handling file uploads.
- **Python-Dotenv**: For loading environment variables.
- **Boto3**: For interacting with AWS S3-compatible storage (e.g., MinIO).

Worker Container

- **Python 3.9**: For TensorFlow and Celery integration.
- **Celery**: For task scheduling and distributed execution.
- **Flower**: For monitoring Celery workers.
- **Python-Dotenv**: For loading environment variables.
- **TensorFlow**: For model training.
- **PyYAML**: For parsing dataset definitions.
- **Pandas**: For dataset preprocessing.
- **Cryptography**: For signing the AIBoM.
- **CycloneDX-Python-Lib**: For generating CycloneDX SBOMs.
- **Boto3**: For interacting with AWS S3-compatible storage (e.g., MinIO).

Repository Structure

```
AIBoMGen/
├── api/
│   ├── Dockerfile          # API service Docker configuration
│   ├── requirements.txt    # API dependencies
│   ├── app.py              # FastAPI application
│   └── routes/             # API route definitions
├── worker/
│   ├── Dockerfile          # Worker service Docker configuration
│   ├── requirements.txt    # Worker dependencies
│   ├── tasks.py            # Celery tasks for training
│   ├── celery_config.py    # Celery configuration
│   └── aibom_generator.py  # AIBoM generation and signing logic
├── shared/
│   └── minio_utils.py      # MinIO helper functions
├── utils                   # Contains python scripts for generating a
│                           - private key and generating test models and datasets
├── docker-compose.yml      # Docker Compose configuration
└── README.md               # Project documentation
```

Setup

1. Clone the Repository

```
git clone <repository-url>
cd AIBoMGen
```

2. Start Docker Daemon

- **Linux**:

  ```
  sudo systemctl start docker
  ```

- **Windows**: Start Docker Desktop.

3. Configure Environment Variables

Create a `.env` file in the root directory with the following variables:

```
RABBITMQ_USER=rmq_user
RABBITMQ_PASSWORD=rmq_password
FLOWER_BASIC_AUTH=admin:admin
CELERY_BROKER_URL=amqp://rmq_user:rmq_password@rabbitmq:5672//
CELERY_RESULT_BACKEND=rpc://
MINIO_ROOT_USER=minio_user
MINIO_ROOT_PASSWORD=minio_password
MINIO_ENDPOINT=http://minio:9000
MINIO_BUCKET_NAME=aibomgen
```

4. Add a Private Key for Signing

Place a valid `private-key.pem` file in the `worker/` directory or ensure it is mounted into the worker container. This key is used to cryptographically sign the AIBoM, ensuring trustability and compliance with regulations (e.g., the EU AI Act). The utils directory has a python script `generate_private_key.py` to let you generate one for testing purposes.

5. Start Docker Compose

```
docker-compose up --build
```

Usage

API Endpoints

The FastAPI service runs on http://localhost:8000. Below are the key endpoints:

1. Submit a Training Job

- **Endpoint**: `POST /submit_job_by_model_and_data`

- **Request Body**:

```
{
  "model": <binary file>,  // Upload the model file (e.g., .keras)
  "dataset": <binary file>,  // Upload the dataset file (e.g., .csv)
  "dataset_definition": <binary file>  // Upload the dataset definition file
(e.g., .yaml)
}
```

- Testing files can be generated using the provided python scripts in the `utils/` directory.
- **Example dataset_definition.yaml**:

```
columns:
   feature1: float
   feature10: float
   feature11: float
   feature12: float
   feature13: float
   feature2: float
   feature3: float
   feature4: float
   feature5: float
   feature6: float
   feature7: float
   feature8: float
   feature9: float
   label: int
input_shape:
- 13
label: label
output_shape:
- 1
preprocessing:
   clip:
   - 0.0
   - 1.0
   normalize: true
   scale: 1.0
```

- **Response**:

```
{
  "job_id": "123e4567-e89b-12d3-a456-426614174000",
  "status": "Training started",
  "unique_dir": "6251847e-710f-47a3-b809-070518e4b1b9"
}
```

2. Check Job Status

- **Endpoint**: `GET /job_status/{job_id}`
- **Response**:

```
{
  "job_id": "123e4567-e89b-12d3-a456-426614174000",
  "status": "completed",
  "result": {
    "training_status": "training job completed",
    "unique_dir": "6251847e-710f-47a3-b809-070518e4b1b9",
    "job_id": "123e4567-e89b-12d3-a456-426614174000",
    "message": "Training completed successfully and AIBoM generated.",
    "output_artifacts": [
      "trained_model.keras",
      "metrics.json",
      "logs.txt",
      "aibom.json",
      "aibom.sig"
    ]
  }
}
```

3. Retrieve Job Artifacts

- **Endpoint**: `GET /job_artifacts/{job_id}`
- **Response**:

```
{
  "job_id": "123e4567-e89b-12d3-a456-426614174000",
  "artifacts": [
    ".../trained_model.keras",
    ".../etrics.json",
    ".../logs.txt",
    ".../aibom.json",
    ".../aibom.sig"
  ]
}
```

4. Download a Specific Artifact

- **Endpoint**: `GET /job_artifacts/{job_id}/{artifact_name}`
- **Response**:

```
{
  "artifact_name": "trained_model.keras",
  "url": "http://minio:9000/aibomgen/6251847e-710f-47a3-b809-
070518e4b1b9/trained_model.keras?X-Amz-Algorithm=..."
}
```

UNDER DEVELOPMENT

AIBoM Generation

The **AI Bill of Materials (AIBoM)** is a JSON document that captures:

- **Environment Details**: Python version, TensorFlow version, and system metadata.
- **Input Files**: Hashes of datasets and model files used for training.
- **Output Files**: Hashes of generated artifacts (e.g., trained model, metrics).
- **Configuration**: Training parameters such as epochs, batch size, and learning rate.

Example AIBoM

```json
{
  "environment": {
    "python_version": "3.9",
    "tensorflow_version": "2.12.0",
    "request_time": "2025-04-16 17:00:00",
    "job_id": "123e4567-e89b-12d3-a456-426614174000"
  },
  "inputs": {
    "dataset.csv":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
    "model.h5": "d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2"
  },
  "outputs": {
    "trained_model.keras":
"d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2",
    "metrics.json":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
  },
  "config": {
    "epochs": 10,
    "batch_size": 32,
    "learning_rate": 5e-5
  }
}
```

Trustability Features

1. **Automated AIBoM Generation**:

    - The AIBoM is generated automatically by the system during the training process. Users cannot modify or bypass this step.

2. **Cryptographic Signing**:

- The AIBoM is signed using a private key, creating a digital signature that ensures the authenticity and integrity of the document.

3. **Verification**:

   - The generated AIBoM and its signature can be verified using the corresponding public key, ensuring that the training process was not tampered with.

4. **Controlled API**:

   - Users interact with the system exclusively through a secure API. This prevents arbitrary code execution, ensuring that users cannot tamper with the training process, bypass the AIBoM generation, or interfere with the system's infrastructure.

5. **Containerized Workers**:

   - Training jobs are executed in isolated Docker containers. This ensures that each job runs in a secure and controlled environment, preventing users from accessing or modifying other parts of the system.

Dashboards

The system provides several dashboards for monitoring and managing the components of AIBoMGen:

1. **API (FastAPI)**:

   - **URL**: http://localhost:8000/docs
   - The FastAPI dashboard provides interactive API documentation using Swagger UI. You can test endpoints, view request/response formats, and explore the API.

2. **RabbitMQ**:

   - **URL**: http://localhost:15672
   - RabbitMQ's management UI allows you to monitor message queues, exchanges, and worker activity. Use the credentials defined in your `.env` file (`RABBITMQ_USER` and `RABBITMQ_PASSWORD`) to log in.

3. **Flower (Celery)**:

   - **URL**: http://localhost:5555
   - Flower provides a real-time monitoring dashboard for Celery workers. You can view task statuses, worker activity, and task execution details.

4. **MinIO**:

   - **URL**: http://localhost:9001
   - MinIO's web interface allows you to manage datasets, models, and artifacts stored in the object storage. Use the credentials defined in your `.env` file (`MINIO_ROOT_USER` and `MINIO_ROOT_PASSWORD`) to log in.
