**🏗️ System Architecture Overview**

**1. Data Ingestion Layer**

* **Component:** Claim Data Loader
* **Description:** Extracts structured and unstructured claims data from CMS SynPUF and synthetic datasets. Supports batch ingestion from CSV, JSON, or SQL sources.
* **Tech Stack:** pandas, SQLAlchemy, MySQL Connector, Airflow (for scheduling)

**2. Data Preprocessing & Tokenization**

* **Component:** Claim Preprocessor
* **Description:** Cleans, normalizes, and tokenizes claim fields (CPT, ICD, provider ID, dates, notes). Converts structured fields into embeddings and prepares unstructured text for transformer input.
* **Tech Stack:** spaCy, scikit-learn, HuggingFace Tokenizers, NumPy

**3. Embedding & Feature Engineering**

* **Component:** Claim Embedding Generator
* **Description:** Converts each claim into a high-dimensional vector using domain-adapted transformer models (e.g., ClinicalBERT, RoBERTa). Includes positional and provider-aware embeddings.
* **Tech Stack:** HuggingFace Transformers, PyTorch, sentence-transformers

**4. Transformer-Based Detection Engine**

* **Component:** Fraud Detection Model
* **Submodules:**
  + **Duplicate Claims Detector**
  + **Upcoding Detector**
  + **Phantom Billing Detector**
  + **Temporal Pattern Analyzer**
* **Description:** Fine-tuned transformer models trained to detect anomalies using semantic similarity, time-series modeling, and adversarial perturbation.
* **Tech Stack:** PyTorch Lightning, Transformers, Optuna (for hyperparameter tuning)

**5. Mathematical Analysis & Interpretability**

* **Component:** Model Auditor
* **Description:** Applies PAC-Bayes bounds, eigenvalue analysis of attention weights, and perturbation testing to ensure robustness and interpretability.
* **Tech Stack:** SciPy, NumPy, Matplotlib, Captum (for model interpretability)

**6. Evaluation & Simulation**

* **Component:** Performance Evaluator
* **Description:** Computes ROC-AUC, F1, precision-recall, and simulates cost savings. Includes bootstrapping and cross-validation.
* **Tech Stack:** scikit-learn, Seaborn, Matplotlib, pandas

**7. Database & Storage**

* **Component:** Claims Repository
* **Description:** Stores raw, preprocessed, and labeled claims data. Also stores model outputs and audit logs.
* **Tech Stack:** MySQL, SQLAlchemy, Alembic (for migrations)

**8. API & Interface Layer**

* **Component:** Fraud Detection API
* **Description:** RESTful API for submitting claims and retrieving fraud scores or explanations.
* **Tech Stack:** FastAPI, Uvicorn, Pydantic

**9. Dashboard & Visualization**

* **Component:** Monitoring Dashboard
* **Description:** Visualizes fraud trends, model performance, and flagged claims. Supports drill-down into attention weights and similarity scores.
* **Tech Stack:** Streamlit or Dash, Plotly, Grafana (optional for metrics)

**10. Orchestration & Deployment**

* **Component:** Pipeline Orchestrator
* **Description:** Manages end-to-end workflows from data ingestion to model inference and reporting.
* **Tech Stack:** Airflow, Docker, Kubernetes (for scaling), MLflow (for experiment tracking)

Here’s a **complete set of generic scripts** and a **deployment guideline** for your Transformer-Based Healthcare Payment Integrity System. These scripts are modular and aligned with the architecture you provided.

**✅ Scripts Overview**

**1. Data Ingestion Layer (data\_ingestion.py)**

import pandas as pd

from sqlalchemy import create\_engine

def load\_claims\_data(db\_url: str, table\_name: str) -> pd.DataFrame:

engine = create\_engine(db\_url)

df = pd.read\_sql\_table(table\_name, con=engine)

return df

**2. Data Preprocessing & Tokenization (preprocessing.py)**

from transformers import AutoTokenizer

import pandas as pd

tokenizer = AutoTokenizer.from\_pretrained("roberta-base")

def preprocess\_claims(df: pd.DataFrame):

texts = df["notes"].fillna("").tolist()

return tokenizer(texts, padding=True, truncation=True, return\_tensors="pt")

**3. Embedding Generator (embedding\_generator.py)**

from transformers import AutoModel

class EmbeddingModel:

def \_\_init\_\_(self, model\_name="roberta-base"):

self.model = AutoModel.from\_pretrained(model\_name)

def generate\_embeddings(self, tokenized\_inputs):

with torch.no\_grad():

outputs = self.model(\*\*tokenized\_inputs)

return outputs.last\_hidden\_state.mean(dim=1)

**4. Fraud Detection Engine (fraud\_detection.py)**

import torch

from torch.nn.functional import cosine\_similarity

def detect\_duplicate\_claims(embeddings\_a, embeddings\_b, threshold=0.9):

scores = cosine\_similarity(embeddings\_a, embeddings\_b)

return scores > threshold

**5. Model Auditor (model\_auditor.py)**

from captum.attr import LayerIntegratedGradients

def interpret\_model(model, inputs):

lig = LayerIntegratedGradients(model, model.embeddings)

attributions = lig.attribute(inputs["input\_ids"], target=0)

return attributions

**6. Evaluation & Simulation (evaluation.py)**

from sklearn.metrics import roc\_auc\_score, f1\_score

def evaluate(y\_true, y\_pred):

return {

"roc\_auc": roc\_auc\_score(y\_true, y\_pred),

"f1\_score": f1\_score(y\_true, y\_pred)

}

**7. API Layer (api.py)**

from fastapi import FastAPI

from pydantic import BaseModel

app = FastAPI()

class Claim(BaseModel):

provider\_id: str

procedure\_code: str

date: str

notes: str

@app.post("/predict")

def predict(claim: Claim):

# Placeholder for model inference

return {"fraud\_score": 0.85}

**8. Dashboard (dashboard.py)**

import streamlit as st

import pandas as pd

def run\_dashboard():

st.title("Healthcare Fraud Detection Dashboard")

df = pd.read\_csv("results.csv")

st.dataframe(df)

if \_\_name\_\_ == "\_\_main\_\_":

run\_dashboard()

**9. Orchestration (pipeline.py)**

from data\_ingestion import load\_claims\_data

from preprocessing import preprocess\_claims

from embedding\_generator import EmbeddingModel

from fraud\_detection import detect\_duplicate\_claims

def run\_pipeline():

df = load\_claims\_data("mysql+pymysql://user:pass@localhost/claims\_db", "claims")

tokenized = preprocess\_claims(df)

model = EmbeddingModel()

embeddings = model.generate\_embeddings(tokenized)

# Add detection logic here

if \_\_name\_\_ == "\_\_main\_\_":

run\_pipeline()

**🚀 Deployment & Testing Guide**

**🔧 Environment Setup**

1. **Install dependencies**
2. pip install -r requirements.txt
3. **MySQL Setup**
   * Create a database claims\_db
   * Import synthetic claims data into a table claims
4. **Model Setup**
   * Download pretrained models (e.g., RoBERTa, ClinicalBERT)
   * Fine-tune on labeled claims data if available

**🧪 Testing**

* **Unit Tests**: Use pytest for each module
* **API Testing**: Use curl or Postman to test /predict endpoint
* **Dashboard Testing**: Run streamlit run dashboard.py

**🐳 Docker Deployment**

**Dockerfile**

FROM python:3.10

WORKDIR /app

COPY . /app

RUN pip install -r requirements.txt

CMD ["python", "pipeline.py"]

**Build & Run**

docker build -t fraud-detector .

docker run -p 8000:8000 fraud-detector

**☁️ Cloud Deployment**

* Use **Kubernetes** for scaling
* Use **MLflow** for model tracking
* Use **Airflow** for orchestration

Would you like me to generate this as a downloadable zip or GitHub-ready repo structure?