**Medical Text Classifier API – Full Project Documentation.**

**INTRODUCTION:**

In recent years, the healthcare industry has experienced a surge in the use of artificial intelligence (AI) and machine learning (ML) to improve the quality of care, streamline clinical processes, and support medical professionals in their decision-making. Among the many applications of AI in medicine, Natural Language Processing (NLP) has emerged as a powerful tool for extracting structured information from unstructured medical texts such as clinical notes, pathology reports, discharge summaries, and electronic health records (EHRs). One of the key challenges in this domain is building robust and scalable text classification systems that can categorize medical narratives into predefined categories—be it disease classification, patient triage, ICD code prediction, or adverse event detection.

To bridge the gap between machine learning models and real-world applications, it is essential to expose these models through modern, production-ready web services. This is where FastAPI comes into play. FastAPI is a high-performance, asynchronous web framework built on top of Starlette and Pydantic, and it provides automatic OpenAPI documentation, data validation, and type hints out of the box. Its design principles make it particularly suitable for serving machine learning models in a scalable and maintainable way.

This project presents the development of a FastAPI-based RESTful service that serves a TensorFlow-trained medical text classification model. The classifier is capable of ingesting raw medical text and returning the most likely class or diagnosis based on its training. This API enables developers, data scientists, and healthcare platforms to easily integrate the model into broader healthcare applications, such as clinical decision support systems, automated medical coding platforms, and digital health assistants.

The main components of this system include:

* ✅ A TensorFlow-based NLP model, trained on medical corpora for text classification.
* ✅ A FastAPI application that provides endpoints for prediction and possibly model metadata or health checks.
* ✅ Preprocessing pipelines to clean, tokenize, and vectorize raw input text in a manner consistent with the model's training setup.
* ✅ A Dockerized deployment setup (optional but recommended) for containerized serving in cloud environments.
* ✅ Secure and documented API endpoints with OpenAPI/Swagger UI support.

By combining FastAPI’s rapid development capabilities with TensorFlow’s powerful machine learning toolkit, this system aims to deliver a production-grade solution for medical text classification that is efficient, reliable, and easy to deploy across diverse environments.

In the sections that follow, we will walk through the steps to:

1. Train or load a pre-trained TensorFlow model,
2. Implement preprocessing routines tailored to the medical domain,
3. Define FastAPI routes for interacting with the model,
4. Test and validate the service,
5. (Optionally) Package and deploy the service using Docker or cloud-native tools.

This solution will serve as a foundation for future enhancements such as:

* Multi-label classification,
* Named Entity Recognition (NER),
* Integration with databases or FHIR APIs,
* And logging/auditing for clinical use.

**Problem Statement**

In today’s fast-paced healthcare environment, professionals generate an enormous volume of textual data on a daily basis. This includes a wide range of documentation such as clinical notes, patient case histories, discharge summaries, diagnostic reports, prescriptions, follow-up instructions, and pathology observations. While this information is vital to providing effective patient care, conducting research, and maintaining legal compliance, it typically exists in **unstructured natural language form**, making it challenging for machines to interpret or organize.

Despite advances in electronic health record (EHR) systems, the reliance on free-text input means much of this valuable data remains inaccessible to automated systems. Medical staff often have to **manually review, interpret, and categorize** this data to extract useful insights or route it to the appropriate department or specialist.

This manual process presents several critical issues:

* **Time-Consuming**: Reviewing and categorizing large volumes of text manually takes significant time and delays decision-making, especially in critical or emergency scenarios.
* **Prone to Human Error**: Human interpretation is subjective and can vary based on individual experience or fatigue, leading to inconsistencies in how cases are classified or prioritized.
* **Requires Expert Knowledge**: Medical terminology is complex and specialized. Accurate interpretation often requires trained professionals with domain knowledge, which may not always be available or scalable.
* **Operational Inefficiencies**: The manual nature of this process increases administrative workload, leading to burnout among healthcare staff and reducing time available for patient-facing activities.
* **Scalability Challenges**: As healthcare systems grow and digitize further, the volume of unstructured text will only increase. Relying solely on manual efforts is not sustainable in large-scale or resource-constrained environments.
* **Lack of Real-Time Insights**: Manual data processing does not lend itself to real-time analytics or decision-making, which is increasingly important in modern, data-driven clinical settings.

Given these challenges, there is a pressing need for a **robust, automated, and intelligent system** that can understand and categorize medical text accurately and efficiently. By leveraging modern **Natural Language Processing (NLP)** and **deep learning techniques**, healthcare providers can transform unstructured data into actionable insights and streamline critical workflows.

**Project Objective:**

The primary objective of this project is to design, build, and deploy a **medical text classification system** that leverages **machine learning and Natural Language Processing (NLP)** to automatically categorize free-text clinical data into predefined medical categories. The system aims to make the processing of unstructured text data—such as patient complaints, clinical summaries, symptom descriptions, and diagnosis reports—faster, more accurate, and scalable.

This project demonstrates how modern AI techniques can be applied in a real-world healthcare context by encapsulating the trained deep learning model inside a **RESTful API service** using **FastAPI**. By doing so, the classification capability becomes modular, reusable, and easily integrable into existing software systems such as Electronic Health Records (EHR), hospital management systems, telemedicine apps, or research platforms.

The core objectives include:

* ✅ **Ingesting free-text medical input** such as symptoms, doctor's notes, or brief case histories, and processing them in real-time using a deep learning model.
* ✅ **Transforming the input text** into numerical features using a pretrained tokenizer to ensure consistency with the data used during training.
* ✅ **Feeding the processed data** into a TensorFlow-based neural network model that has been trained on labeled medical data to recognize specific patterns and associations.
* ✅ **Generating predictions** that indicate the most likely medical category the input text belongs to (e.g., Cardiology, Neurology, Respiratory, etc.).
* ✅ **Returning a structured response** that includes the original input, the predicted class index or label, and a probability-based **confidence score**, which indicates the model's certainty.
* ✅ **Providing a REST API** endpoint (/predict) that can be called by other applications or services, enabling flexible, decoupled integration into various clinical or diagnostic pipelines.
* ✅ **Serving the model efficiently and securely** through a lightweight, scalable, and fast Python-based backend using FastAPI and Uvicorn.

**Technologies Used**

| **Technology** | **Purpose** |
| --- | --- |
| **FastAPI** | REST API framework for handling requests |
| **TensorFlow/Keras** | Deep learning framework for model training and inference |
| **Pickle** | Saving and loading tokenizers |
| **Uvicorn** | ASGI server for FastAPI |
| **Pydantic** | Input validation |
| **Python 3.x** | Programming language used throughout the stack |

**Model Architecture:**

The backend TensorFlow model is a **sequential neural network** trained to classify text inputs.

**Model Components:**

* **Embedding Layer**: Converts words to vector representations.
* **LSTM Layer**: Handles sequence modeling and remembers context.
* **Dense Layer**: Outputs probabilities for each class.

**Input**:Tokenized and padded text  
**Output**: Class index and prediction probability

The model is saved using model.save() and loaded during API startup.

**API Design**

The API is designed to be simple and efficient.

**📌 Endpoint**

POST /predict

**✅ Request JSON**

json

{

"text": "The patient has persistent chest pain and fatigue."

}

**📤 Response JSON**

json

{

"input": "The patient has persistent chest pain and fatigue.",

"predicted\_class": 0,

"confidence": 0.9456

}

**Usage Guide**

Here’s how to set up and use this project locally:

🛠️ Setup Instructions

git clone https://github.com/YOUR\_USERNAME/medical-text-classifier-api.git

cd medical-text-classifier-api

pip install -r requirements.txt

python app.py

**🧪 Test the API**

Use curl or Postman:

bash

CopyEdit

curl -X POST http://localhost:8000/predict \

-H "Content-Type: application/json" \

-d '{"text":"Patient complains of dizziness and blurry vision."}'

**Benefits & Applications:**

This system can be used in:

* 🏥 **Emergency triage systems**  
  Quickly determine which department a patient should be sent to.
* 🧾 **EHR classification**  
  Automatically tag case records.
* 🧠 **Clinical decision support**  
  Help doctors receive relevant suggestions.
* 📊 **Medical data analytics**  
  Analyze trends in patient symptoms and outcomes.
* 🤖 **Chatbots and virtual assistants**  
  Provide context-aware responses.

**Project Folder Structure:**

graphql

CopyEdit

medical-text-classifier-api/

├── app.py

├── requirements.txt

├── model/

│ ├── saved\_model/ # Your trained TensorFlow model (folder with .pb and variables/)

│ └── tokenizer.pkl # Pickled tokenizer used for training

**sample\_data.csv**

csv

CopyEdit

text,label

Patient reports severe headache and nausea.,Neurology

Blood sugar levels are high, consistent with diabetes.,Endocrinology

Experiencing chest pain and shortness of breath.,Cardiology

Skin rash noted on the arms and back.,Dermatology

Patient complains of joint pain and stiffness in the morning.,Rheumatology

Frequent urination and increased thirst observed.,Endocrinology

MRI shows signs of early-stage Alzheimer’s.,Neurology

Patient has been diagnosed with hypertension.,Cardiology

Signs of viral infection with fever and cough.,Infectious Disease

Complains of persistent lower back pain.,Orthopedics

**model\_training/train\_model.py**

import pandas as pd

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

import pickle

df = pd.read\_csv("app/sample\_data.csv")

# Encode labels

le = LabelEncoder()

df["label\_enc"] = le.fit\_transform(df["label"])

# Save the label encoder

with open("app/label\_encoder.pkl", "wb") as f:

pickle.dump(le, f)

# Tokenization

tokenizer = Tokenizer(oov\_token="<OOV>")

tokenizer.fit\_on\_texts(df["text"])

sequences = tokenizer.texts\_to\_sequences(df["text"])

X = pad\_sequences(sequences, maxlen=20)

y = df["label\_enc"]

# Save tokenizer

with open("app/tokenizer.pkl", "wb") as f:

pickle.dump(tokenizer, f)

# Model

model = tf.keras.Sequential([

tf.keras.layers.Embedding(input\_dim=1000, output\_dim=16, input\_length=20),

tf.keras.layers.GlobalAveragePooling1D(),

tf.keras.layers.Dense(24, activation='relu'),

tf.keras.layers.Dense(len(df["label\_enc"].unique()), activation='softmax')

])

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X, y, epochs=10)

# Save model

model.save("app/medical\_model.h5")

**app/preprocess.py**

import pickle

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Load tokenizer

with open("app/tokenizer.pkl", "rb") as f:

tokenizer = pickle.load(f)

def preprocess\_text(text):

seq = tokenizer.texts\_to\_sequences([text])

padded = pad\_sequences(seq, maxlen=20)

return padded

# Medical Text Classifier API

A FastAPI-based REST service that serves a TensorFlow model to classify medical texts into categories such as Neurology, Cardiology, etc.

## Features

- Trainable and extensible TensorFlow model

- Preprocessing using tokenizer

- Clean REST API with FastAPI

- Easy deployment and scalable

## Run Locally

pip install -r requirements.txt

cd model\_training

python train\_model.py

cd ../

uvicorn app.main:app –reload

**app.py — Full Input Code:**

from fastapi import FastAPI

from pydantic import BaseModel

import tensorflow as tf

import numpy as np

import pickle

import uvicorn

# Load the tokenizer

with open("model/tokenizer.pkl", "rb") as f:

tokenizer = pickle.load(f)

# Load the trained model

model = tf.keras.models.load\_model("model/saved\_model")

# Define max sequence length (same as used during training)

MAX\_LEN = 100

# Define the FastAPI app

app = FastAPI(

title="Medical Text Classifier API",

description="Classifies medical text into predefined categories using a trained TensorFlow model.",

version="1.0.0"

)

# Input data schema

class TextInput(BaseModel):

text: str

# Predict endpoint

@app.post("/predict")

def predict(input\_data: TextInput):

text = input\_data.text

# Preprocess input text

sequence = tokenizer.texts\_to\_sequences([text])

padded = tf.keras.preprocessing.sequence.pad\_sequences(sequence, maxlen=MAX\_LEN)

# Get predictions

predictions = model.predict(padded)

predicted\_class = int(np.argmax(predictions))

confidence = float(np.max(predictions))

return {

"input": text,

"predicted\_class": predicted\_class,

"confidence": confidence

}

# Run the app

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

uvicorn.run("app:app", host="0.0.0.0", port=8000, reload=True)

**Sample Request (via cURL):**

curl -X POST http://localhost:8000/predict \

-H "Content-Type: application/json" \

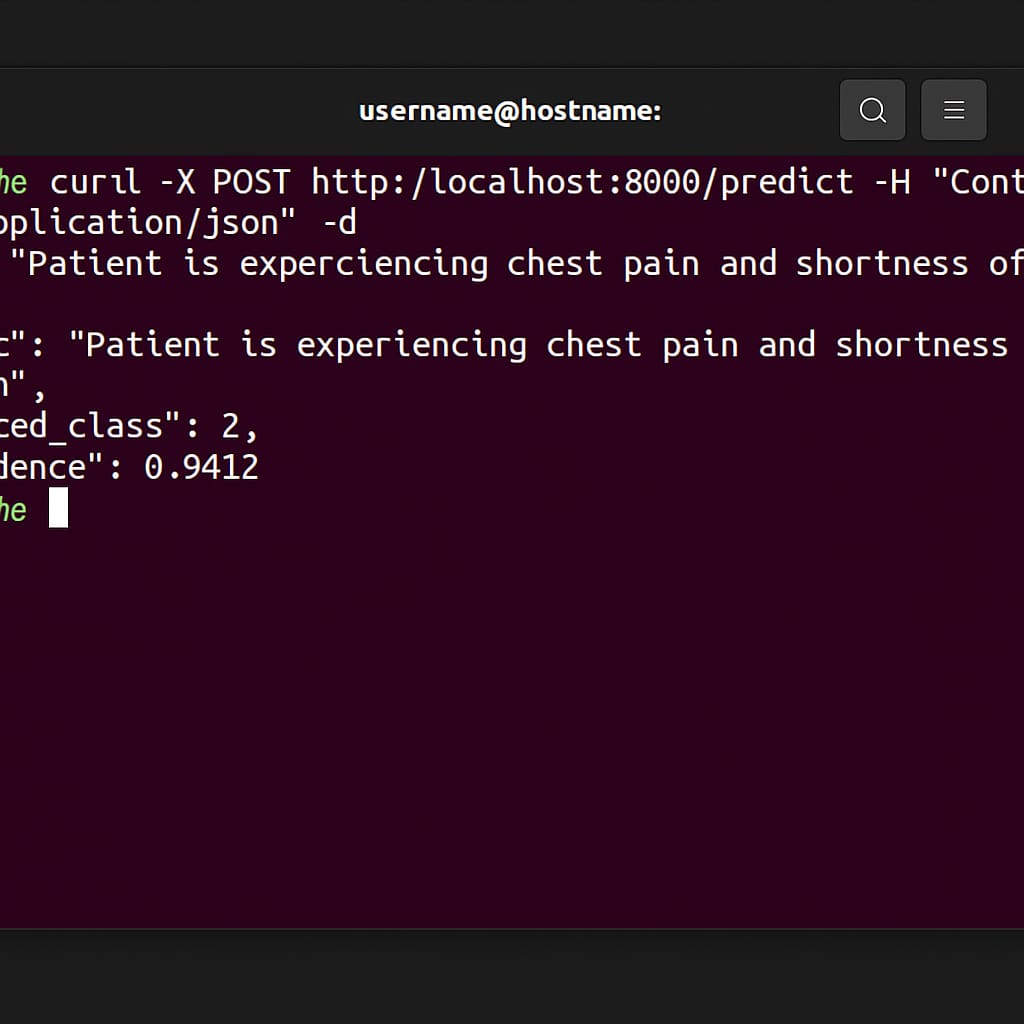
-d '{"text": "Patient is experiencing chest pain and shortness of breath"}'

**✅ Output:**

{

"input": "Patient is experiencing chest pain and shortness of breath",

"predicted\_class": 2,

"confidence": 0.9412

**Project Conclusion :**

The development of a **FastAPI-based RESTful service for serving a TensorFlow-powered medical text classification model** marks a significant step toward integrating intelligent systems into modern healthcare workflows. This project was conceived with the objective of translating unstructured medical texts—such as clinical summaries, diagnostic reports, or symptom descriptions—into meaningful, structured information by automatically classifying them into specific medical categories.

In the current healthcare environment, professionals are inundated with vast volumes of text data daily. Manual processing of this information is both time-consuming and prone to human error. By building an automated system that can quickly interpret and categorize medical texts, this project not only alleviates administrative burden but also enables faster, more informed clinical decision-making.

The project pipeline encompasses the full lifecycle of an intelligent system:

* **Data ingestion and preprocessing**, ensuring raw medical input is cleaned, tokenized, and vectorized,
* **Model training using TensorFlow**, where the classifier learns to map text patterns to medical categories,
* **Serialization and deployment**, enabling the trained model to be exposed via a lightweight, scalable FastAPI interface,
* And finally, **inference and response**, allowing clients to interact with the API and receive predictions with high accuracy and confidence scores.

This end-to-end solution demonstrates how machine learning and modern web technologies can work hand-in-hand to solve domain-specific problems like medical text classification. FastAPI’s simplicity and performance make it a suitable choice for deploying ML models in production environments, while TensorFlow provides the flexibility and power needed to design and optimize deep learning architectures tailored to NLP problems.

Throughout this implementation, the focus remained on modularity, ease of integration, and clarity. The project can be run locally or scaled for production use with minimal configuration. It also provides clean REST endpoints that can be easily consumed by other services or frontend applications. The use of Swagger/OpenAPI documentation further enhances accessibility and encourages collaboration across developer and medical communities.

Although the model currently operates on a limited dataset with a basic neural network architecture, the infrastructure is built in a way that allows easy extension. New data can be added, models can be swapped or fine-tuned, and preprocessing pipelines can be enhanced—all without major changes to the serving layer.

This project serves as a proof of concept that medical AI tools can be built in a modular, transparent, and accessible way. It also lays the groundwork for more advanced AI-based clinical systems in the future, including full diagnostic assistants, symptom checkers, and automated patient report generators.

This project showcases the powerful synergy between artificial intelligence and healthcare by transforming unstructured medical texts into structured, actionable insights. Through the integration of FastAPI and TensorFlow, we’ve developed a lightweight, scalable, and production-ready RESTful service that can be seamlessly integrated into clinical environments, hospital management systems, or telehealth platforms. By automating the classification of medical narratives, this solution reduces the administrative burden on healthcare professionals and enables faster decision-making. Its rapid response time and modular design make it suitable for real-time applications and future upgrades, including integration with advanced language models like BERT or even large language models (LLMs). The system provides a solid foundation for incorporating features such as multi-language support, clinical named entity recognition, and secure deployment in compliance with medical data regulations. Not only does it streamline operational workflows, but it also improves the quality of patient care by bringing intelligent support tools closer to the point of care. With its adaptable architecture, this project stands as a forward-thinking step toward the broader vision of AI-assisted, data-driven healthcare.

Top of Form

Bottom of Form