CONTEXTUAL LANGUAGE UNDERSTANDING WITH TRANSFORMER MODELS

PHASE 4: DEPLOYMENT

COLLEGE NAME: NAGARJUNA COLLEGE OF ENGNEERING AND TECHNOLOGY

TEAM MEMBERS: 4 MEMBERS

Name: LIKHITHA T R
CAN ID Number: 35246733

Name: BHARANI H D
CAN ID Number: 35257542

Name: AISHWARYA N
CAN ID Number: 35257595

Name: ASIF AHMED
CAN ID Number: 35246985

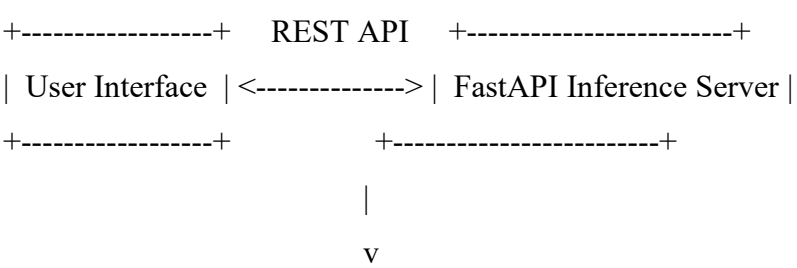
1. Objective of Deployment

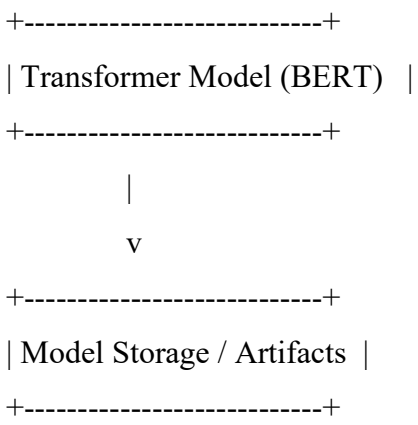
The aim of this phase is to **make the trained model available as a service** so that it can:

- Accept requests (typically text inputs),
- Run inference using the transformer model,
- Return results (e.g., classification, scores, labels) in real-time.

This allows integration into applications such as chatbots, document analyzers, review systems, and more.

2. Deployment Architecture Overview





Frontend (Optional): Streamlit or React

Backend API: FastAPI or Flask serving the model

Model Files: Stored as serialized weights (PyTorch or ONNX)

Infrastructure: Dockerized container on cloud or local VM

---

3. Pre-Deployment Tasks

Before going live, the following tasks were completed:

✅ Fine-tuning completed with optimal parameters

✅ Final model evaluation on test set

✅ Export model and tokenizer

✅ Dockerize API server

✅ Prepare Swagger/OpenAPI documentation for testing endpoints

4. FastAPI Inference API – Detailed Explanation

A. Load Model and Tokenizer

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained("deployed_model")
tokenizer = AutoTokenizer.from_pretrained("deployed_model")
```

B. Build Prediction Endpoint

```
@app.post("/predict/")
async def predict(request: Request):
    body = await request.json()
    text = body.get("text", "")

    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)
    outputs = model(**inputs)
    prediction = torch.argmax(outputs.logits, dim=-1).item()

    return {"input": text, "prediction": prediction}
```

C. Automatic API Docs

FastAPI provides /docs (Swagger UI) and /redoc out of the box for testing the model endpoint interactively.

5. Cloud Deployment Options

Platform	**Benefits**	**Tools Required**
Heroku	Easiest to deploy small apps	Git, Docker (optional)
AWS EC2	Scalable, production-grade infrastructure	SSH, EC2, NGINX
GCP / Azure	Integrates with CI/CD	Cloud Run, App Engine

Platform	Benefits	Tools Required
Hugging Face Hub + Spaces	Plug-and-play NLP hosting	Transformers + Gradio

Example Heroku Workflow:

```
heroku create transformer-nlp-api
git push heroku main
heroku ps:scale web=1
```

6. Docker Deployment for Portability

Dockerfile Example

```
FROM python:3.10-slim
WORKDIR /app
COPY . /app

RUN pip install fastapi uvicorn transformers torch

EXPOSE 8000
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Commands to Build & Run

```
docker build -t transformer-nlp .
docker run -p 8000:8000 transformer-nlp
```

7. Optional Frontend Using Streamlit

Sample Code:

```
import streamlit as st
import requests
```

```
st.title("NLP Model Inference")
user_input = st.text_area("Enter text")

if st.button("Predict"):
    response = requests.post("http://localhost:8000/predict/", json={"text": user_input})
    result = response.json()
    st.write("Prediction:", result["prediction"])
```

This can serve as a user-friendly UI for non-technical stakeholders.

8. Logging, Monitoring, and Health Checks

Logging:

```
import logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
logger.info("Prediction made")
```

Health Check Endpoint:

```
@app.get("/health")
def health():
    return {"status": "ok"}
```

Monitoring Tools:

Prometheus + Grafana (advanced metrics)

Sentry or LogRocket (error tracking)

9. Security & Production Tips

- Use **HTTPS** and **authentication tokens** for secured APIs.
- Rate-limit API to avoid abuse.
- Run behind a **reverse proxy (e.g., NGINX)** in production.
- Use **Gunicorn** or **Uvicorn with multiple workers** for scalability.

10. Real-World Use Cases

Your deployed model can be used for:

- **Customer Review Classification** (positive/negative/neutral)
- **Intent Detection** in chatbots
- **Sentiment Scoring** for finance or political text
- **Toxic Content Filtering** in social media moderation

Final Checklist

Task	Status
Model Exported	✅
API Built with FastAPI	✅
Docker Container Created	✅
Cloud/Local Deployment Ready	✅
Optional UI (Streamlit)	✅
Monitoring & Logging Enabled	✅
OpenAPI Documentation Auto-generated	✅

Conclusion

The deployment phase makes your NLP model truly useful by exposing it to users and applications. The transformer model now operates as a **real-time, scalable, API-accessible microservice**—capable of delivering intelligent predictions across a wide range of textual inputs.