Sentiment Analysis Project Documentation

Chapter 3: From Training to Serving

3.1 Why I Chose FastAPI

FastAPI was chosen over Flask for several reasons:

- Speed: FastAPI is asynchronous and built on Starlette and Pydantic, making it highly performant for serving ML models.

- Type Hinting: Offers automatic validation and OpenAPI documentation from Python type hints.

- Ease of use: Rapid to prototype and scale into production-level API endpoints.

🔍 Compared to Flask, FastAPI allows us to handle multiple requests efficiently—crucial when doing inference on large datasets or concurrent web requests.

3.2 Building the Local Server

Steps:

1. Load the trained model and tokenizer.

```
model = BertForSequenceClassification.from_pretrained("./model")
tokenizer = BertTokenizer.from_pretrained("./model")
```

2. Create the prediction endpoint.

```
@app.post("/predict")
async def predict(input: InputText):
    tokens = tokenizer(input.text, return_tensors="pt")
    outputs = model(**tokens)
    prediction = torch.argmax(outputs.logits, dim=1)
    return {"sentiment": label[prediction.item()]}
```

3. Test locally using http://127.0.0.1:8000/docs (Swagger UI).

3.3 Deploying with GitHub + Railway

- **GitHub + CI/CD Integration**: The entire backend was pushed to a GitHub repository to enable automated deployment pipelines via Railway. CI/CD (Continuous Integration/Continuous Deployment) ensures that every code push is automatically tested and deployed to production. This was achieved using Railway's GitHub integration, which triggers builds upon every git push, simplifying deployment management.

- **railway.toml Explained**: The railway.toml file defines how Railway should build and run the app. It can include build commands, start commands, and variables specific to the Railway environment.

  Example:

  ```
  [build]
  command = "pip install -r requirements.txt"

  [start]
  command = "uvicorn main:app --host 0.0.0.0 --port 8000"
  ```

  This file ensures that the platform knows exactly how to prepare the runtime environment.

- **pyproject.toml Explained**: pyproject.toml is a modern Python packaging file that can define your project dependencies and build system. In our case, it was used to prevent build errors with pip, especially on Railway's build environment. For Example:

  ```
  [build-system]
  requires = ["setuptools", "wheel", "pip>=21"]
  build-backend = "setuptools.build_meta"
  ```

  This file resolved issues related to legacy pip versions and helped maintain reproducibility across deployments.

- **Dockerfile for Custom Environment**:

  ```
  FROM python:3.10
  WORKDIR /app
  COPY . .
  RUN pip install -r requirements.txt
  CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
  ```

  This helped ensure consistent behavior across development and deployment environments.

- **Debugging Challenges Faced**:
  - Railway build failures due to missing build configurations.
  - Errors caused by outdated pip versions, resolved using pyproject.toml.
  - Long troubleshooting sessions with .toml syntax errors or directory path mismatches.
  - Unexpected Docker caching issues that required forcing clean builds.
- **Large File Handling**:
  - Used Git LFS or .gitattributes and .gitignore to manage large model files.
  - Removed unnecessary files like cache folders and raw datasets to reduce build time.
- **Limitations**:
  - GitHub restricts file size to 100MB per file.
  - Repository cloning with large models can slow down the build process.
  - Railway's free-tier containers had memory constraints during model loading.

3.4 Model Hosting Challenges

Problems:

- Cold start delays (30–45s)

- BERT model reloaded per request

- Memory/timeouts on Railway free-tier

Fixes:

- Cached model in memory

- Used .safetensors to reduce footprint

- Timed and optimized latency

3.5 Hugging Face Model Hub to the Rescue

Why switch to Hugging Face?

- Uploading large model files (with no 100MB GitHub limit)

- Enabled the **Inference API** to serve predictions with GPU support
- Hugging Face Spaces for deploying visual demo using Gradio

```
transformers-cli login
transformers-cli repo create SentimentAnalysisV2
transformers-cli upload ./model/
```

Benefits:

- Fast inference using free GPU backend
- Simplified model management
- Public API with version control and auto-deployment

3.6 Lessons Learned

Platform	Pros	Cons
Railway	Simple FastAPI deployment	Cold start, memory limits
GitHub	Great for version control	100MB file limit
Hugging Face	GPU inference + easy hosting	Less custom frontend flexibility

Takeaways:

- Optimize model loading speed

- Use .safetensors for memory optimization

- Plan ahead for **model size**, **CI/CD integration**, and **hosting cost**.

- Hugging Face = best GPU-backed serving option

🚀 Switching from Railway to Hugging Face significantly improved inference speed and developer experience.