Question Answering System - Project Documentation

**Project Structure**  
qa\_app/

│

├── app.py # FastAPI app entry point

├── app\_ui.py # Streamlit UI for interactive input/output

├── model\_utils.py # Utility functions to load model and get answers

├── download\_model.py # Downloads the fine-tuned HF model

├── requirements.txt # Dependencies

├── FineTuningQAModel.ipynb # Jupyter notebook for model fine-tuning

├── README.md # Project overview and instructions

│

├── qa\_app/

│ └── model/ # Trained QA model (config, tokenizer, pytorch\_model)

# Project Objective

The goal of this project is to:

* Fine-tune a pre-trained QA model on a subset of the SQuAD dataset
* Evaluate the model's performance
* Deploy it using FastAPI for real-time question answering
* Provide a Streamlit UI for easy frontend interaction
* Log performance with Weights & Biases

Steps Overview

# 1. Environment Setup

python -m venv qa  
source qa/bin/activate # On Windows: qa\Scripts\activate  
pip install -r requirements.txt

**2.Model Training**

* Model: distilbert-base-uncased
* Dataset: SQuAD (subset)
* Hugging Face Trainer API was used for training with a reduced dataset for faster experimentation.

Training Output:



**3.Evaluation**

* **Metric**: squad\_v2 (F1 and Exact Match)
* **Key Results:**
  + F1 Score: 30.26
  + Exact Match: 26.0
  + HasAns\_f1: 67.25
  + NoAns\_f1: 0.0

**4.FastAPI Deployment**

Download Model: Place your fine-tuned model inside:

.\qa\_app\model

Sample model\_utils.py:

from transformers import pipeline

qa\_pipeline = pipeline("question-answering", model="qa\_app/model", tokenizer="qa\_app/model")

def get\_answer(question: str, context: str) -> str:

result = qa\_pipeline(question=question, context=context)

return result['answer']

Sample app.py:

from fastapi import FastAPI

from pydantic import BaseModel

from model\_utils import get\_answer

app = FastAPI()

class QARequest(BaseModel):

question: str

context: str

@app.post("/qa/")

def qa\_endpoint(payload: QARequest):

answer = get\_answer(payload.question, payload.context)

return {"answer": answer}

Run Server:  
uvicorn app:app –reload

**5. Streamlit UI**

streamlit run app\_ui.py

* Enter context and question in the provided boxes
* Click submit to view the generated answer
* See response and score in real-time

**6. Monitoring with Weights & Biases**

* Logs:
  + Inference latency (ms)
  + Model confidence score
* wandb dashboard link is generated automatically on first run

**Example API Request**

POST /qa/  
  
Request:

{

"question": "Who wrote the theory of relativity?",

"context": "Albert Einstein developed the theory of relativity in the early 20th century."

}

Response:

{  
 "answer": "Albert Einstein"  
}

UI:  
  
A screenshot of a computer

AI-generated content may be incorrect.

**Future Improvements**

* Improve handling of unanswerable questions (boost NoAns metrics)
* Add streaming support for large real-time datasets
* Include Swagger/OpenAPI docs (already auto-generated by FastAPI at /docs)
* Add Dockerfile for containerized deployment
* Implement authentication for secure usage in production