SmartQA+: Natural Language Question Answering API

Built by Sumesh (2025)

Goal

The goal of this project is to build a simple, intelligent Question Answering (QA) system that can answer natural language questions about member data retrieved from a public API.

Example Questions

- "When is Layla planning her trip to London?"
- "How many cars does Vikram Desai have?"
- "What are Amira's favorite restaurants?"

The system provides a `/ask` API endpoint that accepts a natural-language question and returns an inferred answer based on the messages dataset.

```
!pip install -q fastapi uvicorn pyngrok sentence-transformers transformers requests
```

Environment Setup

The following libraries are installed for:

- **FastAPI** – to create the API service
- **Uvicorn** – ASGI web server
- **Sentence Transformers** – for semantic search
- **Transformers** – for question-answering pipeline
- **Pyngrok** – to create a public endpoint for Colab testing

```
from fastapi import FastAPI, Query
from pyngrok import ngrok
from sentence_transformers import SentenceTransformer, util
from transformers import pipeline
import requests, torch, nest_asyncio, uvicorn, threading, logging, datetime
```

```
logging.basicConfig(
    format="%(asctime)s - %(levelname)s - %(message)s",
    level=logging.INFO
)

# STEP 4: Initialize FastAPI
app = FastAPI(
    title="SmartQA+",
    description="Enhanced Natural-Language Question Answering API built over member messages",
    version="2.0"
)
```

🧠 Model Initialization

We use two models to build SmartQA+:

1. **SentenceTransformer (`all-MiniLM-L6-v2`)** for finding the most semantically relevant messages.
2. **RoBERTa (`deepset/roberta-base-squad2`)** for extracting precise answers from the retrieved message context.

```
logging.info("Loading models...")
embedder = SentenceTransformer("all-MiniLM-L6-v2")
qa_pipeline = pipeline("question-answering", model="deepset/roberta-base-squad2")
logging.info("Models loaded successfully ✅")

def fetch_messages():
    """Retrieve member messages from the provided API."""
    try:
        resp = requests.get("https://november7-730026606190.europe-west1.run.app/messages", timeout=15)
        resp.raise_for_status()
        data = resp.json()
        return data.get("items", [])
    except Exception as e:
```

```
        logging.error(f"Data fetch error: {e}")
        return []
```

```
Device set to use cpu
```

🔍 System Workflow

1. Fetch all member messages from the provided API.
2. Encode both the user question and the messages into embeddings.
3. Perform semantic search to identify top relevant messages.
4. Use a QA model to extract the most probable answer from the best contexts.
5. Return a structured JSON response.

```
@app.get("/ask")
def ask(question: str = Query(..., description="Enter a natural language question")):
    start_time = datetime.datetime.utcnow().isoformat()
    try:
        messages = fetch_messages()
        if not messages:
            return {"answer": "No messages found in the dataset."}

        texts = [m["message"] for m in messages if "message" in m]
        if not texts:
            return {"answer": "Dataset contains no valid message text."}

        # Compute embeddings for semantic similarity
        q_emb = embedder.encode(question, convert_to_tensor=True)
        m_embs = embedder.encode(texts, convert_to_tensor=True)

        # Retrieve top 3 relevant messages
        hits = util.semantic_search(q_emb, m_embs, top_k=3)[0]
        top_contexts = [texts[h["corpus_id"]] for h in hits]
        combined_context = " ".join(top_contexts)  # combine multiple relevant snippets

        # Extract answer using the QA pipeline
        qa_result = qa_pipeline(question=question, context=combined_context)
        answer = qa_result.get("answer", "").strip()
        confidence = round(float(qa_result.get("score", 0)), 3)

        # Construct structured, professional JSON output
        return {
            "question": question,
            "answer": answer or "No clear answer found.",
            "confidence": confidence,
            "context_used": top_contexts,
            "total_messages_scanned": len(messages),
            "timestamp_utc": start_time
        }

    except Exception as e:
        logging.exception("Error during Q&A processing")
        return {"error": str(e)}
```

```
@app.get("/")
def home():
    return {
        "message": "Welcome to SmartQA+ 🚀",
        "description": "A semantic question-answering API over member messages.",
        "usage_example": "/ask?question=When%20is%20Layla%20planning%20her%20trip%20to%20London",
        "docs": "/docs",
        "developer": "Built by Sumesh - 2025"
    }
```

```
nest_asyncio.apply()
```

🧪 Testing Instructions

After the model loads, a public URL is generated using **ngrok**.

Example:

```
ngrok.set_auth_token("35FWHZLswIjkQVh2Yjnji0x2yfg_EpwmiUYTz3T9KynrWi4E")

public_url = ngrok.connect(8000)
```

```
print(f"\n🌍 Public SmartQA+ URL: {public_url.public_url}")
print(f"➡️ Swagger Docs: {public_url.public_url}/docs")
print(f"➡️ Example: {public_url.public_url}/ask?question=When%20is%20Layla%20planning%20her%20trip%20to%20Londor

# Run FastAPI server in a thread
def run_server():
    uvicorn.run(app, host="0.0.0.0", port=8000)

thread = threading.Thread(target=run_server)
thread.start()
```

```
🌍 Public SmartQA+ URL: https://crowned-ilene-comely.ngrok-free.dev
➡️ Swagger Docs: https://crowned-ilene-comely.ngrok-free.dev/docs
➡️ Example: https://crowned-ilene-comely.ngrok-free.dev/ask?question=When%20is%20Layla%20planning%20her%20trip%20
```

You can:

- Open the Swagger UI (`/docs`) to test questions interactively.
- Or directly call the `/ask` endpoint from a browser or Postman.