Nice — this is a fun project. I'll give you a complete, ready-to-run prototype that follows the diagram you showed, but uses a Groq-based LLM + TTS (placeholders you can wire to Groq), Whisper for transcription, and a Python web UI for fast iteration.

I recommend a single-process **Streamlit** app for the prototype (fast to build, deployable, lets you upload/play audio, show transcriptions, etc.). The app will:

- Accept an uploaded voice note (or record locally and upload).
- Chunk the audio (configurable).
- Transcribe each chunk with Whisper (OpenAI API or local whisper — I'll show the OpenAI/whisper client path and a local whisper fallback comment).
- Send the concatenated transcription to a Groq LLM via a simple HTTP wrapper (placeholder).
- Maintain a conversational chain (simple memory stored in session state).
- Ask Groq to produce a response text.
- Send response text to Groq TTS (placeholder) to create the response audio.
- Return/play response audio (Ava's note) and display text.

Below you'll find:

1. File structure
2. `requirements.txt`
3. Streamlit app code (`app.py`)
4. `audio_utils.py` for chunking + WAV conversion
5. Dockerfile (optional)
6. How to replace ElevenLabs calls with Groq TTS + Groq LLM — where to add real endpoints/keys.
7. Deployment notes.

---

1 — File structure

```
voice-assistant-groq/
├─ app.py
├─ audio_utils.py
├─ requirements.txt
├─ Dockerfile
├─ README.md
```

---

2 — requirements.txt

```graphql
streamlit
pydub
requests
```

```
numpy
openai        # if using OpenAI Whisper API
soundfile
python-dotenv
ffmpeg-python
```

Note: install system `ffmpeg` (apt/yum/brew) for pydub conversions.

---

3 — audio_utils.py

Save this as `audio_utils.py`. It handles chunking and conversion to WAV (16k) expected by many transcription APIs.

```python
python

# audio_utils.py
import os
from pydub import AudioSegment
import math
import uuid

def ensure_dir(p):
    os.makedirs(p, exist_ok=True)
    return p

def load_audio(file_path):
    return AudioSegment.from_file(file_path)

def convert_to_wav_16k(input_path, out_path=None):
    audio = load_audio(input_path)
    audio = audio.set_frame_rate(16000).set_channels(1).set_sample_width(2)
    if out_path is None:
        out_path = f"/tmp/{uuid.uuid4().hex}.wav"
    audio.export(out_path, format="wav")
    return out_path

def chunk_audio_wav(input_wav_path, chunk_length_ms=8000, out_dir="/tmp/chunks"):
    """
    Split a wav file into chunk_length_ms (milliseconds) segments.
    Returns list of chunk file paths.
    """
    ensure_dir(out_dir)
    audio = AudioSegment.from_wav(input_wav_path)
    duration_ms = len(audio)
    num_chunks = math.ceil(duration_ms / chunk_length_ms)
    chunk_paths = []
    for i in range(num_chunks):
        start = i * chunk_length_ms
        end = min((i + 1) * chunk_length_ms, duration_ms)
        chunk = audio[start:end]
        chunk_name = os.path.join(out_dir, f"chunk_{i}_{uuid.uuid4().hex}.wav")
        chunk.export(chunk_name, format="wav")
        chunk_paths.append(chunk_name)
    return chunk_paths
```

---

4 — app.py (Streamlit app)

Save as `app.py`. This is the main UI and orchestration. Replace placeholder Groq endpoints with your real endpoints.

python

```
# app.py
import streamlit as st
import tempfile, os, requests, json
from audio_utils import convert_to_wav_16k, chunk_audio_wav
from pathlib import Path
from dotenv import load_dotenv
import openai    # optional, only if using OpenAI Whisper

load_dotenv()

# CONFIG
CHUNK_MS = 8000  # 8 sec chunks (adjust)
WORK_DIR = "/tmp/voice_assistant"
Path(WORK_DIR).mkdir(parents=True, exist_ok=True)

# API KEYS (set in .env or environment)
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")   # optional for Whisper
GROQ_API_KEY = os.getenv("GROQ_API_KEY")       # placeholder for Groq LLM/TTS

# ---------- Helpers ----------
def transcribe_with_openai_whisper(audio_file_path):
    """
    Uses OpenAI whisper api (if you have access), otherwise replace with local whisper code.
    """
    openai.api_key = OPENAI_API_KEY
    # The OpenAI REST endpoint or openai-python client may vary; this is an example.
    # If your account doesn't support whisper via OpenAI or you prefer local, swap in your
pipeline.
    with open(audio_file_path, "rb") as f:
        # NOTE: adjust to your installed openai version's interface
        resp = openai.Audio.transcribe("whisper-1", f)
    return resp.get("text", "")

def transcribe_chunks(chunk_paths):
    texts = []
    for p in chunk_paths:
        # Choose your transcription method:
        if OPENAI_API_KEY:
            txt = transcribe_with_openai_whisper(p)
        else:
            # Fallback: naive local transcription placeholder (requires a local model)
            txt = f"[transcript placeholder for {os.path.basename(p)}]"
        texts.append(txt)
    return " ".join(texts)

# ---------------- Groq placeholders ----------------
def groq_llm_query(prompt, conversation_history=None):
    """
    Replace this with your Groq LLM request.
    Example: a POST to your Groq endpoint with headers including GROQ_API_KEY.
    Return a text response body.
    """
    # Placeholder pseudo-call:
    if not GROQ_API_KEY:
        # return a mock response for prototyping
        return "Hello — this is a mock Groq response to: " + (prompt[:200])
    url = "https://api.groq.example/v1/generate"  # <- REPLACE with real Groq LLM endpoint
    headers = {"Authorization": f"Bearer {GROQ_API_KEY}", "Content-Type": "application/json"}
    payload = {
        "prompt": prompt,
        "max_tokens": 512,
        # other model-specific params...
    }
    r = requests.post(url, headers=headers, json=payload)
    r.raise_for_status()
    data = r.json()
    # Adjust extraction per Groq response shape
    return data.get("text") or data.get("output") or json.dumps(data)

def groq_tts_synthesize(text, out_path):
    """
    Replace with Groq TTS endpoint. This stub writes silence or TTS placeholder.
    If GROQ_API_KEY present, do real HTTP request to Groq TTS.
    """
    if not GROQ_API_KEY:
        # generate placeholder beep using pydub
        from pydub.generators import Sine
        tone = Sine(440).to_audio_segment(duration=800)
        tone.export(out_path, format="wav")
        return out_path

    url = "https://api.groq.example/v1/tts"  # <- REPLACE
    headers = {"Authorization": f"Bearer {GROQ_API_KEY}", "Content-Type": "application/json"}
    payload = {"text": text, "voice": "ava", "format": "wav"}
    r = requests.post(url, headers=headers, json=payload, stream=True)
    r.raise_for_status()
    # write binary stream to out_path
    with open(out_path, "wb") as f:
        for chunk in r.iter_content(chunk_size=8192):
            if chunk:
                f.write(chunk)
    return out_path

# ---------------- Streamlit UI ----------------
st.set_page_config(page_title="Ava — Groq voice assistant prototype", layout="wide")

st.title("Ava — Voice assistant prototype (Groq LLM + TTS)")

st.markdown("Upload a voice note; the app will chunk → transcribe → query LLM → synthesize
response audio.")

# session memory for convo chain
if "conversation" not in st.session_state:
    st.session_state.conversation = []

uploaded_file = st.file_uploader("Upload voice note (mp3/wav/m4a/ogg)", type=
["mp3","wav","m4a","ogg"])

col1, col2 = st.columns(2)

with col1:
    st.subheader("Transcription & LLM")
    if uploaded_file:
        tmp_in = os.path.join(WORK_DIR, uploaded_file.name)
        with open(tmp_in, "wb") as f:
            f.write(uploaded_file.getbuffer())

        wav16 = convert_to_wav_16k(tmp_in)
        chunk_paths = chunk_audio_wav(wav16, chunk_length_ms=CHUNK_MS,
out_dir=os.path.join(WORK_DIR, "chunks"))
        st.write(f"Split into {len(chunk_paths)} chunk(s).")

        with st.spinner("Transcribing chunks..."):
            transcription = transcribe_chunks(chunk_paths)
        st.text_area("Transcription", value=transcription, height=200)

        # Build conversation prompt
        # Memory: store last N turns
        memory_text = "\n".join(st.session_state.conversation[-10:])
        prompt = f"System: You are Ava, friendly assistant.\nMemory:\n{memory_text}\n\nUser
said: {transcription}\n\nReply as Ava in short, friendly voice:"
        st.code(prompt[:1000] + ("..." if len(prompt)>1000 else ""))

        if st.button("Send to Groq LLM"):
            with st.spinner("Querying Groq LLM..."):
                llm_resp = groq_llm_query(prompt,
conversation_history=st.session_state.conversation)
            st.session_state.conversation.append(f"User: {transcription}")
            st.session_state.conversation.append(f"Ava: {llm_resp}")
            st.success("Got response")
            st.write("**Ava (text):**")
            st.write(llm_resp)

with col2:
    st.subheader("Synthesized audio (Ava)")
    if "llm_resp" in locals():
        out_audio_path = os.path.join(WORK_DIR,
f"ava_{os.path.basename(uploaded_file.name)}.wav")
        with st.spinner("Generating TTS..."):
            tts_path = groq_tts_synthesize(llm_resp, out_audio_path)
        st.audio(open(tts_path, "rb").read(), format="audio/wav")
        st.download_button("Download Ava's note", data=open(tts_path,"rb").read(),
file_name="ava_note.wav", mime="audio/wav")

st.sidebar.header("Session Memory")
st.sidebar.write("Last conversation turns:")
for t in st.session_state.conversation[-10:]:
    st.sidebar.write(t)

st.sidebar.markdown("---")
st.sidebar.write("Env keys used:")
st.sidebar.write(f"OPENAI_API_KEY={'set' if OPENAI_API_KEY else 'NOT set'}")
st.sidebar.write(f"GROQ_API_KEY={'set' if GROQ_API_KEY else 'NOT set'}")
```

---

5 — Dockerfile (optional)

If you want to containerize:

```
dockerfile


FROM python:3.11-slim

WORKDIR /app
COPY requirements.txt .
RUN apt-get update && apt-get install -y ffmpeg build-essential && pip install --no-cache-dir -r
requirements.txt
COPY . .
EXPOSE 8501
CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

Build & run:

```
arduino


docker build -t voice-assistant-groq .
docker run -p 8501:8501 -e OPENAI_API_KEY=... -e GROQ_API_KEY=... voice-assistant-groq
```

---

6 — Where to swap out ElevenLabs for Groq

- **`groq_tts_synthesize()`** is your TTS replacement. Replace the **`url`** and request/response decoding with Groq TTS endpoints (if Groq provides TTS). If Groq doesn't provide TTS, you can:
    - Use Groq LLM to produce SSML or phoneme-friendly text and use another TTS provider that you prefer.
    - Or integrate any other TTS engine (coqui TTS, local models, or paid vendor).
- **`groq_llm_query()`** is where you POST the prompt to Groq LLM. Replace the placeholder URL/headers with the real Groq HTTP API shape. Map the returned JSON to extract response text.

Important: I kept the code modular so swapping providers is just changing those two functions.

---

7 — WhatsApp voice note integration (optional)

If you want to accept voice notes directly from WhatsApp (diagram shows WhatsApp), you'll need a webhook (Twilio WhatsApp or Meta WhatsApp Business API). The flow:

1. WhatsApp user sends voice note → webhook receives media URL.
2. Your server downloads media → saves → passes to the same chunk/transcribe pipeline.
3. Reply back to WhatsApp with generated audio via Twilio/WhatsApp API.

This adds complexity (webhooks + auth). For a prototype, uploading via the Streamlit UI is enough.

---

8 — Notes, caveats & next steps

- **Groq API specifics:** I included placeholders for Groq endpoints. I don't hardcode any vendor-specific calls because exact endpoints/payloads differ and change — plug in your Groq API docs there.
- **Transcription:** If you prefer local transcription, you can integrate `whisperx` or `faster-whisper` instead of OpenAI.
- **Chunking:** I used fixed-size chunks (8s). For long audio, consider VAD (voice activity detection) to chunk on silence.
- **Conversational chain (memory):** currently stored in session state; for persistence use a DB (SQLite, Redis) or vector DB (Pinecone, Milvus) for semantic memory.
- **Latency & cost:** Consider batching or summarizing chunk transcripts before sending to LLM if audio is long.
- **Privacy & security:** If handling real user voice notes (WhatsApp), ensure consent and secure storage (encryption, retention policy).

---

9 — Example `.env` (local)

```ini
OPENAI_API_KEY=sk-...
GROQ_API_KEY=groq-...
```

If you want, I can:

- Fill `groq_llm_query()` and `groq_tts_synthesize()` with exact Groq HTTP requests if you paste Groq's API docs or an example request/response.
- Add a Twilio WhatsApp webhook example to ingest voice notes automatically.
- Convert the app to a FastAPI backend + React frontend if you want a production architecture (better separation of UI & API).

Which of those three follow-ups would you like me to do right now? (I can directly implement the Groq integration for you if you paste the Groq API spec or sample requests/response formats.)

Perfect 👍 Since you want **only Groq** (both **Whisper transcription** + **Llama LLM response** + **TTS** if Groq supports it), we'll drop OpenAI/ElevenLabs entirely.

I'll give you a **Streamlit webapp** that:

1. Takes an audio file (voice note).
2. Chunks → sends to **Groq Whisper (whisper-large-v3)** for transcription.
3. Sends the transcript to **Groq LLaMA-3 70B** for response.
4. Uses **Groq TTS** (if exposed in their API, otherwise placeholder — you can plug in once you confirm endpoint).
5. Plays and downloads Ava's generated audio.

🔹 Requirements

```txt
streamlit
pydub
requests
soundfile
python-dotenv
ffmpeg-python
```

System dependency: `ffmpeg`

🔹 `.env`

```ini
GROQ_API_KEY=your_groq_api_key_here
```

🔹 `app.py`

```python
import os
import requests
import streamlit as st
from dotenv import load_dotenv
from pydub import AudioSegment
import uuid

load_dotenv()
GROQ_API_KEY = os.getenv("GROQ_API_KEY")

GROQ_HEADERS = {
    "Authorization": f"Bearer {GROQ_API_KEY}"
}

# -------- Helpers --------
def convert_to_wav_16k(uploaded_file):
    """Convert uploaded audio to 16kHz mono wav"""
    temp_path = f"/tmp/{uuid.uuid4().hex}.wav"
    audio = AudioSegment.from_file(uploaded_file)
    audio = audio.set_frame_rate(16000).set_channels(1).set_sample_width(2)
    audio.export(temp_path, format="wav")
    return temp_path

def groq_transcribe(file_path):
    """Transcribe audio using Groq Whisper"""
    url = "https://api.groq.com/openai/v1/audio/transcriptions"
    with open(file_path, "rb") as f:
        files = {"file": f}
        data = {"model": "whisper-large-v3"}
        r = requests.post(url, headers=GROQ_HEADERS, data=data, files=files)
    r.raise_for_status()
    return r.json()["text"]

def groq_llm(prompt):
    """Chat with Groq LLaMA"""
    url = "https://api.groq.com/openai/v1/chat/completions"
    payload = {
        "model": "llama-3.1-70b-versatile",
        "messages": [
            {"role": "system", "content": "You are Ava, a friendly helpful AI."},
            {"role": "user", "content": prompt}
        ]
    }
    r = requests.post(url, headers=GROQ_HEADERS, json=payload)
    r.raise_for_status()
    return r.json()["choices"][0]["message"]["content"]

def groq_tts(text, out_path="ava_note.wav"):
    """
    Placeholder for Groq TTS.
    If Groq exposes TTS endpoint, replace URL+payload below.
    """
    url = "https://api.groq.com/v1/audio/speech"
    payload = {"model": "groq-tts", "voice": "ava", "input": text}
    r = requests.post(url, headers=GROQ_HEADERS, json=payload, stream=True)
    r.raise_for_status()
    with open(out_path, "wb") as f:
        for chunk in r.iter_content(1024):
            f.write(chunk)
    return out_path

# -------- Streamlit UI --------
st.set_page_config(page_title="Ava (Groq Voice Assistant)", layout="wide")
st.title("🎙️ Ava — Groq Voice Assistant")

uploaded = st.file_uploader("Upload a voice note", type=["wav","mp3","m4a","ogg"])

if uploaded:
    wav_path = convert_to_wav_16k(uploaded)

    with st.spinner("🔎 Transcribing with Groq Whisper..."):
        transcript = groq_transcribe(wav_path)
    st.text_area("Transcription", transcript, height=150)

    with st.spinner("🤖 Asking Groq LLaMA..."):
        response = groq_llm(transcript)
    st.success("Got response from Ava")
    st.write("**Ava (Text Response):**")
    st.write(response)

    with st.spinner("🗣️ Generating Ava’s voice..."):
        audio_path = groq_tts(response)
    st.audio(audio_path)
    st.download_button("Download Ava's Note", data=open(audio_path,"rb").read(),
                       file_name="ava_note.wav", mime="audio/wav")
```

🔹 Run

```bash
pip install -r requirements.txt
streamlit run app.py
```

Open http://localhost:8501↗.

⚡ Next steps:

- If Groq doesn't have direct **TTS** yet, I'll add **Coqui TTS** fallback.
- If you plan **WhatsApp integration**, we can wrap this into a FastAPI webhook that handles WhatsApp voice notes.

👉 Mr. ArkS, do you want me to also wire this for **WhatsApp voice notes** (so you get Ava replies directly in WhatsApp), or keep it as a webapp for now?

Got it 🚀 Mr. ArkS — let's extend your Groq-powered Ava into a **WhatsApp bot**.

To build a WhatsApp bot you'll need three parts:

🔹 WhatsApp Bot Architecture

1. **User** sends a voice note on WhatsApp.
2. **Meta WhatsApp Business API** (or **Twilio WhatsApp API**) delivers the voice file URL to your webhook.
3. Your **server (FastAPI/Flask/Node)**:
    - Downloads the voice file.
    - Converts → sends to **Groq Whisper** for transcription.
    - Sends transcript to **Groq LLaMA** for response.
    - Sends response text to **Groq TTS** for voice.
    - Uploads/returns generated voice back to WhatsApp user.

🔹 Option 1 — Meta WhatsApp Business Cloud API (recommended, direct)

Steps:

1. **Create a Meta Developer Account** → developers.facebook.com↗.
2. **Create a WhatsApp Business App** → get sandbox phone number & API token.
3. **Set up a Webhook server** (FastAPI/Flask). Meta will forward messages to your webhook.
4. **Respond back via `POST /messages` API** (send text or audio).

🔹 Example (FastAPI WhatsApp Bot with Groq)

bot.py

```python
from fastapi import FastAPI, Request
import requests, os
from pydub import AudioSegment
import uuid

app = FastAPI()

WHATSAPP_TOKEN = os.getenv("WHATSAPP_TOKEN")
WHATSAPP_PHONE_ID = os.getenv("WHATSAPP_PHONE_ID")
GROQ_API_KEY = os.getenv("GROQ_API_KEY")

GROQ_HEADERS = {"Authorization": f"Bearer {GROQ_API_KEY}"}

def groq_transcribe(file_path):
    url = "https://api.groq.com/openai/v1/audio/transcriptions"
    with open(file_path, "rb") as f:
        files = {"file": f}
        data = {"model": "whisper-large-v3"}
        r = requests.post(url, headers=GROQ_HEADERS, data=data, files=files)
    r.raise_for_status()
    return r.json()["text"]

def groq_llm(prompt):
    url = "https://api.groq.com/openai/v1/chat/completions"
    payload = {
        "model": "llama-3.1-70b-versatile",
        "messages": [{"role":"system","content":"You are Ava."},
                     {"role":"user","content":prompt}]
    }
    r = requests.post(url, headers=GROQ_HEADERS, json=payload)
    r.raise_for_status()
    return r.json()["choices"][0]["message"]["content"]

def groq_tts(text, out_path="ava_note.ogg"):
    url = "https://api.groq.com/v1/audio/speech"   # confirm Groq TTS endpoint
    payload = {"model": "groq-tts", "voice": "ava", "input": text}
    r = requests.post(url, headers=GROQ_HEADERS, json=payload, stream=True)
    with open(out_path, "wb") as f:
        for chunk in r.iter_content(1024):
            f.write(chunk)
    return out_path

def send_whatsapp_audio(user_id, audio_file):
    url = f"https://graph.facebook.com/v20.0/{WHATSAPP_PHONE_ID}/messages"
    files = {"file": open(audio_file, "rb")}
    headers = {"Authorization": f"Bearer {WHATSAPP_TOKEN}"}
    data = {"messaging_product": "whatsapp", "to": user_id, "type": "audio"}
    r = requests.post(url, headers=headers, data=data, files=files)
    r.raise_for_status()

@app.post("/webhook")
async def webhook(request: Request):
    data = await request.json()

    # check if message contains audio
    try:
        entry = data["entry"][0]["changes"][0]["value"]["messages"][0]
        user_id = entry["from"]
        if entry["type"] == "audio":
            media_id = entry["audio"]["id"]
            # get audio file url
            media_url = f"https://graph.facebook.com/v20.0/{media_id}"
            media_resp = requests.get(media_url, headers={"Authorization":f"Bearer
{WHATSAPP_TOKEN}"})
            download_url = media_resp.json()["url"]
            # download file
            audio_file = f"/tmp/{uuid.uuid4().hex}.ogg"
            a = requests.get(download_url, headers={"Authorization":f"Bearer {WHATSAPP_TOKEN}"})
            with open(audio_file, "wb") as f:
                f.write(a.content)

            # convert to wav
            wav_file = f"/tmp/{uuid.uuid4().hex}.wav"
            audio = AudioSegment.from_file(audio_file)
            audio.set_frame_rate(16000).set_channels(1).export(wav_file, format="wav")

            # process with Groq
            transcript = groq_transcribe(wav_file)
            response = groq_llm(transcript)
            response_audio = groq_tts(response)

            # send back
            send_whatsapp_audio(user_id, response_audio)
    except Exception as e:
        print("Error:", e)

    return {"status":"ok"}
```

Run:

```bash
uvicorn bot:app --host 0.0.0.0 --port 8000
```

Expose with **ngrok** → set webhook URL in Meta Dashboard.

🔹 Option 2 — Twilio WhatsApp API (simpler, but paid after trial)

1. Sign up at Twilio ↗.
2. Get sandbox number.
3. Webhook → same as above (FastAPI or Flask).
4. Use Twilio's `twilio` Python SDK to send/receive messages.

🔹 Deployment

- Host on **Render, Railway, or AWS EC2**.
- For testing, use **ngrok**:

```bash
ngrok http 8000
```

- Set the **ngrok URL** in WhatsApp Business App → Webhooks → Messages.

⚡ Now you'll have a **WhatsApp Ava Bot**:

- User: sends voice note →
- Ava (Groq): replies back with synthesized voice.

👉 Do you want me to make this **production-ready with both text + voice handling** (so Ava can also reply in text if voice TTS is unavailable), or keep it **voice-only like your diagram**?