```python
import streamlit as st
import speech_recognition as sr
import pyttsx3
from googletrans import Translator
import tempfile
import os
import soundfile as sf
import librosa
import noisereduce as nr
import matplotlib.pyplot as plt
import librosa.display
import pandas as pd
import numpy as np
import nltk
import google.generativeai as genai
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity
from gtts import gTTS
from pydub import AudioSegment

# NLTK downloads
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Google Gemini setup
genai.configure(api_key="YOUR_GEMINI_API_KEY")
gemini_model = genai.GenerativeModel("models/gemini-1.5-pro")

# Ensure directory exists
os.makedirs("audio_input", exist_ok=True)

# Globals
r = sr.Recognizer()
engine = pyttsx3.init()
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
translator = Translator()

# Load FAQ data and embeddings
faq_df = pd.read_csv("data/FAQ.csv")
faq_df['processed_questions'] = faq_df['Question'].apply(lambda text: [lemmatizer.lemmatize(word) for word in word_tokenize(text.lower()) if word.isalpha() and word no
sentences = faq_df['processed_questions'].tolist()
w2v_model = Word2Vec(sentences=sentences, vector_size=100, window=5, min_count=1, workers=4)

def get_word2vec_vector(words):
    vectors = [w2v_model.wv[word] for word in words if word in w2v_model.wv]
    return np.mean(vectors, axis=0) if vectors else np.zeros(100)

faq_df['word2vec_vectors'] = faq_df['processed_questions'].apply(get_word2vec_vector)

glove_model = {}
with open("data/glove.6B.100d.txt", "r", encoding="utf-8") as f:
    for line in f:
        parts = line.split()
        word = parts[0]
        vector = np.array(parts[1:], dtype='float32')
        glove_model[word] = vector

def get_glove_vector(words):
    vectors = [glove_model[word] for word in words if word in glove_model]
    return np.mean(vectors, axis=0) if vectors else np.zeros(100)

faq_df['glove_vectors'] = faq_df['processed_questions'].apply(get_glove_vector)

# Streamlit interface
st.title("Multilingual Voice-Based Farmer Assistant рџЊѕрџ—ЈпёЏрџЊЌрџ‘ЁвЂЌрџЊѕ")
st.sidebar.header("Instructions")
st.sidebar.write("Select language, speak into mic, and get translated query + smart answer.")

languages = {
    'hi': 'Hindi', 'mr': 'Marathi', 'gu': 'Gujarati', 'bn': 'Bengali',
    'ta': 'Tamil', 'te': 'Telugu', 'kn': 'Kannada', 'ml': 'Malayalam',
    'or': 'Oriya', 'pa': 'Punjabi', 'ur': 'Urdu'
}
selected_lang_code = st.selectbox("Select Language", options=list(languages.keys()), format_func=lambda x: languages[x])

if st.button("рџЋ™пёЏ Start Recording"):
    with sr.Microphone() as source:
        st.write("Listening...")
        r.adjust_for_ambient_noise(source)
        audio_data = r.listen(source, phrase_time_limit=15)

    raw_path = "audio_input/input.wav"
    clean_path = "audio_input/cleaned.wav"

    with open(raw_path, "wb") as f:
        f.write(audio_data.get_wav_data())

    y, sr_orig = librosa.load(raw_path, sr=None)
    y_denoised = nr.reduce_noise(y=y, sr=sr_orig)
    y_resampled = librosa.resample(y_denoised, orig_sr=sr_orig, target_sr=16000)
    y_norm = y_resampled / np.max(np.abs(y_resampled))
    sf.write(clean_path, y_norm, 16000)

    query = r.recognize_google(audio_data, language=selected_lang_code)
    st.success(f"You said: {query}")

    translated_query = translator.translate(query, src=selected_lang_code, dest='en').text
    st.info(f"Translated to English: {translated_query}")

    with open("audio_input/translation_log.txt", "a", encoding="utf-8") as f:
        f.write(f"Language: {languages[selected_lang_code]} ({selected_lang_code})/n")
        f.write(f"User said: {query}/n{translated_query}/n")

    st.audio(raw_path, format='audio/wav')
    st.audio(clean_path, format='audio/wav')

    # Visualizations
    def plot_audio(audio_path, title):
        y, sr = librosa.load(audio_path, sr=None)
        fig, ax = plt.subplots(2, 1, figsize=(12, 6))
        librosa.display.waveshow(y, sr=sr, ax=ax[0])
        ax[0].set(title=f"Waveform - {title}")
        D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
        img = librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='hz', ax=ax[1], cmap='magma')
        ax[1].set(title=f"Spectrogram - {title}")
        fig.colorbar(img, ax=ax)
        st.pyplot(fig)
```

```
plot_audio(raw_path, "Original")
plot_audio(clean_path, "Cleaned")

# Similarity-based response
tokens = [lemmatizer.lemmatize(word) for word in word_tokenize(translated_query.lower()) if word.isalpha() and word not in stop_words]
vec_w2v = get_word2vec_vector(tokens).reshape(1, -1)
vec_glove = get_glove_vector(tokens).reshape(1, -1)

sims_w2v = cosine_similarity(np.vstack(faq_df['word2vec_vectors'].values), vec_w2v).flatten()
sims_glove = cosine_similarity(np.vstack(faq_df['glove_vectors'].values), vec_glove).flatten()
faq_df['similarity'] = (sims_w2v + sims_glove) / 2

best_match = faq_df.loc[faq_df['similarity'].idxmax()]
st.subheader("Best Match from FAQ")
st.markdown(f"**Q:** {best_match['Question']}/n/n**A:** {best_match['Answer']}")

def is_farming_related(text):
    # Define keywords common in farming-related queries
    keywords = [
    # Core farming & agriculture
    'crop', 'soil', 'pesticide', 'fertilizer', 'weather', 'rain', 'market', 'wheat',
    'rice', 'harvest', 'plant', 'farm', 'irrigation', 'insect', 'yield', 'disease',
    'sowing', 'ploughing', 'organic', 'livestock', 'milk', 'tractor', 'seed',
    'paddy', 'maize', 'barley', 'agriculture', 'agronomy', 'drought', 'manure',
    'horticulture', 'poultry', 'farming', 'crop rotation', 'drip irrigation',
    'sprayer', 'greenhouse', 'weather forecast', 'pest', 'farm income', 'commodity prices',
    'barn', 'fodder', 'mulching', 'vermicompost', 'polyhouse', 'thresher',
    'soil health', 'crop insurance', 'farm loan', 'farm subsidy', 'climate change',
    'germination', 'nursery', 'land preparation', 'intercropping', 'weeding',
    'biopesticide', 'biofertilizer', 'water logging', 'crop disease', 'yield prediction',
    'precision farming', 'kharif', 'rabi', 'zayed', 'tillage', 'sprinkler irrigation',
    'irrigation canal', 'farmer', 'veterinary', 'agri input', 'market rate',

    # Government schemes and support
    'pm kisan', 'pm-kisan', 'pm fasal bima yojana', 'pradhan mantri fasal bima yojana',
    'soil health card', 'rashtriya krishi vikas yojana', 'e-nam', 'kisan credit card',
    'kcc', 'pmksy', 'pradhan mantri krishi sinchayee yojana', 'national food security mission',
    'paramparagat krishi vikas yojana', 'msp', 'minimum support price', 'agricultural subsidy',
    'dbt agriculture', 'farmers welfare', 'crop loan waiver', 'subsidy scheme', 'farmer pension scheme',
    'gramin bhandaran yojana', 'kisan samman nidhi', 'agrimarket', 'kisan call center',
    'agmarknet', 'rural employment', 'nrega', 'mnrega', 'rural development'
    'agriculture technology', 'agri-tech', 'digital agriculture', 'smart farming',
    'agriculture innovation', 'agriculture research', 'agriculture extension', 'agriculture education',]

    # Tokenize and lowercase user query
    tokens = [word.lower() for word in word_tokenize(text)]

    # Return True if any keyword is found
    return any(keyword in tokens for keyword in keywords)

# Gemini response conditional
if is_farming_related(translated_query):
    query_for_gemini = translated_query + " Can you summarize this in 5 simple lines as a paragraph that a farmer can easily understand?"
    gemini_response = gemini_model.generate_content(query_for_gemini)
    st.subheader("Gemini AI Summary")
    st.write(gemini_response.text)

    # Translate back
    native_summary = translator.translate(gemini_response.text, src='en', dest=selected_lang_code).text
    st.subheader(f"Translated Summary ({languages[selected_lang_code]})")
    st.write(native_summary)

    # Convert translated summary to speech
    tts = gTTS(text=native_summary, lang=selected_lang_code)
    temp_mp3 = "audio_input/temp_output.mp3"
    output_wav = "audio_input/output.wav"
    tts.save(temp_mp3)
    sound = AudioSegment.from_mp3(temp_mp3)
    sound.export(output_wav, format="wav")

    st.subheader("Spoken Summary in Native Language")
    st.audio(output_wav, format='audio/wav')
    plot_audio(output_wav, "Generated Output (TTS)")

else:
    # Fallback for non-farming queries
    st.subheader("Gemini AI Summary")
    fallback_msg = "Please ask a question related to farming."
    st.warning(fallback_msg)

    # Translate fallback
    fallback_translation = translator.translate(fallback_msg, src='en', dest=selected_lang_code).text
    st.subheader(f"Translated Summary ({languages[selected_lang_code]})")
    st.write(fallback_translation)

    # Convert fallback to speech
    tts = gTTS(text=fallback_translation, lang=selected_lang_code)
    temp_mp3 = "audio_input/temp_output.mp3"
    output_wav = "audio_input/output.wav"
    tts.save(temp_mp3)
    sound = AudioSegment.from_mp3(temp_mp3)
    sound.export(output_wav, format="wav")

    st.subheader("Spoken Summary in Native Language")
    st.audio(output_wav, format='audio/wav')
    plot_audio(output_wav, "Generated Output (TTS)")
```