teraleads
grow your clinic

Name : *Sugandha Sharma*
Assignment: *Teraleads Prompt Engineer Assessment*

Section 1: Voice Prompt Engineering

Do:

Write one concise voice prompt covering name, contact, and appointment preference.

Add notations for tone and pauses (keep it minimal but meaningful).

Provide a short sample conversation (5–6 exchanges max).

Write a 200-word explanation, focused on tone, clarity, and recovery.

Keep focus on:

Natural phrasing

Voice UX (pace, pause, re-prompts)

Basic error handling (“Sorry, I didn’t catch that…”)

Answer:

Voice Prompt

<friendly tone>
"Hey! I’ll help you book your appointment. <pause>
What’s your name, your phone number, and when would you like to come in?"

Sample Conversation

Bot:
"Hey! I’ll help you book your appointment. <pause> What’s your name, your phone number, and when would you like to come in?"

User:
"Hi, I'm Riya. My number's 555-0921."

Bot:
"Thanks, Riya! What day or time works best for you?"

User:
"Saturday morning, if that's free?"

Bot:
"Cool, lemme check real quick..."

Bot:
"Alright, I've got a 10 AM slot on Saturday. Should I book it?"

200-Word Explanation

I wanted this to feel like a natural chat—not a customer service script. The idea is, you're talking to a helpful person who gets straight to the point, but still sounds nice. That's why I started with a casual "Hey!" and used phrases like "lemme check" and "cool."

The tone is super chill and clear. No jargon, nothing awkward. The prompt hits the three things it needs: name, number, and appointment time—but asks it in a way that's easy to answer.

I added short pauses so users aren't rushed, and if they miss something or don't answer fully, the bot can casually say something like, "Oops, didn't catch that—mind repeating your number?"

That's the basic error handling piece—simple, not annoying.

In voice UX, it's all about making the user feel like they're just having a quick convo, not filling out a form. So I kept the flow short and familiar, like something you'd actually say.

Section 2: Python & JSON Integration

Do:

Write a simple Python script:

Hardcode a small JSON string (2–3 appointments).

Parse and generate 2–3 simulated TTS messages.

Annotate with comments: “<friendly tone>”, “<pause>”, etc.

Add a short paragraph on how this could plug into TTS systems.

Focus on:

JSON parsing

Modularity (use a generate_reminder() function)

Readable message output with tone guidance

ANSWER:

For this section, the goal was to simulate how appointment reminders can be prepared in a voice-friendly format using Python and JSON. Below are the steps I followed to complete this part of the assessment:

Step 1: Define Appointment Data in JSON Format

I started by creating a simple JSON string that contains 3 sample appointments. Each entry includes a person’s name, the date, and the time of their appointment.

```
import json

# Hardcoded JSON string containing 3 appointments
appointments_json = '''
[
    {
        "name": "Riya",
        "date": "2025-04-20",
        "time": "10:00 AM"
    },
    {
        "name": "Sam",
        "date": "2025-04-21",
        "time": "2:30 PM"
    },
    {
        "name": "Aisha",
        "date": "2025-04-22",
        "time": "4:00 PM"
    }
]
'''
```

Step 2: Convert the JSON string to a Python list of dictionaries

```
appointments = json.loads(appointments_json)
```

Step 3: Function to generate a basic reminder message with simulated voice annotations

```
def generate_reminder(appointment):
    name = appointment["name"]
    date = appointment["date"]
    time = appointment["time"]

    # Message with placeholder tone and pause notes
    message = (
        f"<friendly tone> Hey {name}, <pause> "
        f"just a quick reminder about your appointment on {date} at {time}. "
        f"<pause> Let us know if you need to reschedule!"
    )
    return message
```

Step 4: Function to generate an SSML-ready version of the reminder message

```
print("🔊 Simulated Voice Prompts:\n")
for appt in appointments:
    print(generate_reminder(appt))
    print("-" * 60)

print("\n🗣️ SSML-Formatted Messages (for real TTS engines):\n")
for appt in appointments:
    print(generate_ssml_reminder(appt))
    print("-" * 60)
```

Output:

```
🔊 Simulated Voice Prompts:

<friendly tone> Hey Riya, <pause> just a quick reminder about your appointment on 2025-04-20 at 10:00 AM. <pause> Let us know if you need to reschedule!
------------------------------------------------------------
<friendly tone> Hey Sam, <pause> just a quick reminder about your appointment on 2025-04-21 at 2:30 PM. <pause> Let us know if you need to reschedule!
------------------------------------------------------------
<friendly tone> Hey Aisha, <pause> just a quick reminder about your appointment on 2025-04-22 at 4:00 PM. <pause> Let us know if you need to reschedule!
------------------------------------------------------------

🗣️ SSML-Formatted Messages (for real TTS engines):

<speak>
        <prosody rate="medium">Hey Riya,</prosody>
        <break time="500ms"/>
        Just a quick reminder about your appointment on 2025-04-20 at 10:00 AM.
        <break time="500ms"/>
        Let us know if you need to reschedule.
    </speak>
------------------------------------------------------------
<speak>
        <prosody rate="medium">Hey Sam,</prosody>
        <break time="500ms"/>
        Just a quick reminder about your appointment on 2025-04-21 at 2:30 PM.
        <break time="500ms"/>
        Let us know if you need to reschedule.
    </speak>
------------------------------------------------------------
<speak>
        <prosody rate="medium">Hey Aisha,</prosody>
        <break time="500ms"/>
        Just a quick reminder about your appointment on 2025-04-22 at 4:00 PM.
        <break time="500ms"/>
        Let us know if you need to reschedule.
    </speak>
------------------------------------------------------------
```

How This Could Plug Into a TTS System

These SSML messages can be sent directly to a TTS API (like Polly or Google Cloud) to generate real audio files or voice calls. Most modern TTS services accept SSML as input, making this an easy integration point for voice-based systems or customer reminder bots.

For example, if using Amazon Polly:

- *The SSML string is passed as input*
- *Polly returns an MP3 of the spoken reminder*
- *That audio can then be used in a call or voice interface*

This exercise shows how even a simple Python script can:

- *Work with structured JSON data*
- *Be modular (each reminder is generated by a function)*
- *Be voice-friendly (annotations + SSML)*
- *Easily scale to plug into real-world voice systems*

Section 3: Automation Workflow Design

Do:

Create a basic flowchart using Draw.io, Whimsical, or even hand-drawn (scan it).

Trigger: Booking confirmation

Steps: Data to JSON → Create voice prompt → Send via API → Log outcome

Add a 250-word description explaining:

Each step

Integration with TTS

Fallbacks (e.g., if no answer, send SMS or retry)

teraleads
grow your clinic

Focus on:

Simplicity

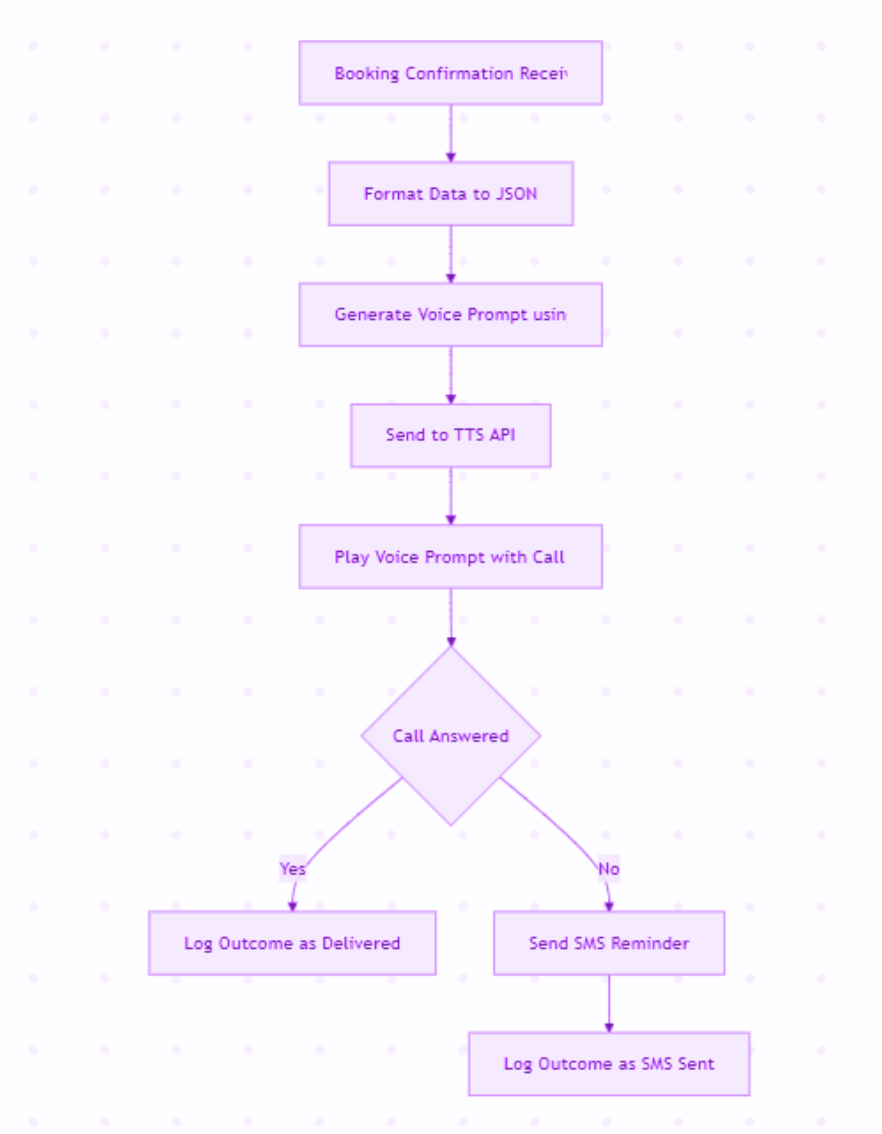
Clear data movement

Voice-specific handling (TTS, delays, retries)

ANSWER

Here is the Flowchart for Booking confirmation. Steps it follows is:

Data to JSON → Create voice prompt → Send via API → Log outcome



Explanation (Workflow Description – ~250 words)

This automation starts when a **booking confirmation** is triggered — either through a form, CRM update, or API call. The goal is to send out a voice reminder in a fully automated, voice-friendly way.

Step 1: Convert Data to JSON

As soon as a booking is confirmed, the system formats key appointment details (like name, date, and time) into a structured JSON format. This ensures the data is clean and standardized for downstream services.

Step 2: Generate a Voice Prompt

Using the JSON, a custom voice message is created — either using a Python function or templating logic. This message is annotated using SSML (Speech Synthesis Markup Language), which controls how the voice sounds (tone, pauses, speed).

Step 3: Send to TTS API

The SSML message is sent to a Text-to-Speech API such as Amazon Polly, Google Cloud TTS, or OpenAI's voice engine. These services convert the text into natural-sounding speech.

Step 4: Deliver via Voice Call

The audio is played through a voice call to the user. If the call is answered, the outcome is logged as **delivered**.

Step 5: Fallback Handling

If the user doesn't answer the call, a **fallback** mechanism kicks in — either retrying the call after a delay or sending an SMS with the same message. This ensures users don't miss critical appointment info.

Final Logging

All outcomes — whether successful or fallback — are logged for tracking and system reliability.

Section 4: Voice RAG System + Use Case

Do:

Use a standard RAG diagram template with:

User query → Retriever → Vector DB → Language Model → Voice Output

Write a 300–350-word explanation:

Focus on voice UX (clarity, truncation, session memory)

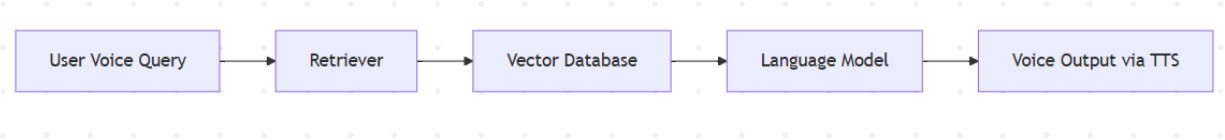
Simulate one voice FAQ use case (e.g., "What are dental implants?")

teraleads
grow your clinic

ANSWER

Explanation (~320 words)

This flow represents a **RAG (Retrieval-Augmented Generation)** system tailored for **voice-based FAQs**. It's designed to provide accurate, spoken responses in a natural way — ideal for clinics, customer support, or any domain where users ask questions out loud.



Step 1: User Voice Query

The user starts by speaking a question, like:
"What are dental implants?"
Their voice is transcribed to text using a Speech-to-Text (STT) engine (e.g., Whisper or Google STT).

Step 2: Retriever

Next, the system identifies key terms ("dental implants") and uses a retriever module (like FAISS or Elasticsearch) to fetch relevant documents from a knowledge base.

Step 3: Vector Database

The retriever matches the query against a **Vector DB**, which contains embeddings of FAQ documents. This lets the system understand semantic similarity — not just keywords — so even slightly rephrased queries get good results.

Step 4: Language Model (LLM)

The LLM takes both the user's question and the retrieved context to generate a fluent, accurate response. This step benefits from prompt tuning to maintain short, informative replies suited for voice.

Step 5: Voice Output

Finally, the response is passed to a **Text-to-Speech (TTS)** engine. The output is optimized with SSML to improve pacing, add pauses, and ensure clarity.

Voice UX Tips:

- **Clarity**: Responses should be short and structured clearly.
- **Truncation**: Avoid overly long answers — keep it ~2–3 sentences.
- **Memory**: For multi-turn dialogs, maintain light session memory (e.g., remember what "that procedure" refers to in the next question).
- **Re-prompts**: If unclear, the voice bot can say:
  "Would you like me to repeat that or give more details?"

Sample Voice Use Case:

User: "What are dental implants?"

Bot: "Dental implants are artificial roots placed in your jaw to support replacement teeth. They look and function like natural teeth. <pause> Would you like to know the procedure or cost?"