# AI VOICE CHATBOT - COMPLETE REQUIREMENTS

## Production-Ready System with Configurable Quality-Latency Optimization

\*\*Document Version\*\*: 3.0 FINAL CLEAN

\*\*Date\*\*: October 10, 2025

\*\*Status\*\*: ✅ COMPLETE - Ready for Implementation

---

# 📑 TABLE OF CONTENTS

\*\*PART 1: OVERVIEW & ARCHITECTURE\*\*

1. [Executive Summary](#1-executive-summary)

2. [System Architecture](#2-system-architecture)

\*\*PART 2: CORE FEATURES\*\*

3. [Quality-Latency Optimization (5 Levels)](#3-quality-latency-optimization)

4. [Voice Pipeline Components](#4-voice-pipeline-components)

\*\*PART 3: ADVANCED FEATURES\*\*

5. [Barge-In System](#5-barge-in-system)

6. [SIP Trunk Integration](#6-sip-trunk-integration)

7. [Colloquial Language & Code-Mixing](#7-colloquial-language)

8. [Voice Selection & Preview](#8-voice-selection)

9. [Noise Handling](#9-noise-handling)

10. [Background Noise Injection (Testing)](#10-background-noise-injection)

\*\*PART 4: AI & SAFETY\*\*

11. [RAG System](#11-rag-system)

12. [Guardrails](#12-guardrails)

\*\*PART 5: TECHNICAL IMPLEMENTATION\*\*

13. [Frontend Architecture](#13-frontend-architecture)

14. [Backend Architecture](#14-backend-architecture)

15. [API Specifications](#15-api-specifications)

16. [Database Schema](#16-database-schema)

\*\*PART 6: OPERATIONS\*\*

17. [Test Agent Mode](#17-test-agent-mode)

18. [Deployment & Infrastructure](#18-deployment)

19. [Cost Management](#19-cost-management)

20. [Monitoring & Analytics](#20-monitoring)

21. [Security & Compliance](#21-security)

\*\*PART 7: EXECUTION\*\*

22. [Implementation Roadmap (20 Weeks)](#22-implementation-roadmap)

23. [Success Metrics](#23-success-metrics)

24. [Appendices](#24-appendices)

---

# 1. EXECUTIVE SUMMARY

## 1.1 What We're Building

A production-ready, multilingual AI voice chatbot supporting \*\*20+ Indian languages\*\* with:

- \*\*User-controlled quality-latency tradeoff\*\* (slider from 0.7s to 4s)

- \*\*Real-time barge-in\*\* for natural interruptions

- \*\*Universal telephony integration\*\* via SIP

- \*\*Comprehensive testing environment\*\* with real APIs

- \*\*Enterprise-grade safety\*\* with multi-layer guardrails

## 1.2 Key Innovation: Quality-Latency Slider

Users choose their preferred balance via a \*\*5-level slider\*\*:

| Level | Latency | Accuracy | Use Case |

|-------|---------|----------|----------|

| \*\*Quality\*\* | 3-4s | 98% | Critical operations |

| \*\*Balanced Quality\*\* | 2-3s | 95% | Customer support |

| \*\*Balanced\*\* ⭐ | 1.5-2s | 90% | Most applications |

| \*\*Balanced Speed\*\* | 1-1.5s | 85% | Quick interactions |

| \*\*Speed\*\* | 0.7-1s | 75% | Real-time gaming |

## 1.3 Technology Stack

```yaml

Frontend: React 18 + TypeScript + Tailwind CSS

Backend: Python FastAPI + WebSocket

ASR: Sarvam AI (Fixed)

LLM: Sarvam AI (Primary) + OpenAI/Anthropic (Optional)

TTS: Sarvam AI or ElevenLabs (Configurable)

Vector DB: Pinecone/Weaviate

Cache: Redis

Database: PostgreSQL

Telephony: FreeSWITCH + Provider Adapters

```

---

# 2. SYSTEM ARCHITECTURE

## 2.1 High-Level Flow

```

┌─────────────────────────────────────────────────────┐

│ USER INPUT (Voice/Phone) │

└────────────┬────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────┐

│ 1. ASR (Sarvam) - Auto Language Detection │

│ Returns: English transcript + language code │

│ Latency: 300ms-1500ms (based on optimization) │

└────────────┬────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────┐

│ 2. RAG Query (Optional) │

│ Retrieve context from knowledge base │

│ Latency: 0ms-500ms │

└────────────┬────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────┐

│ 3. SINGLE LLM CALL (Critical: Only ONE call!) │

│ Input: System prompt + RAG context + Query │

│ Handles: Intent + Guardrails + Response │

│ Latency: 500ms-2000ms │

└────────────┬────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────┐

│ 4. Translation (If needed) │

│ English → Source language │

│ Apply colloquial/code-mixing settings │

│ Latency: 100ms-500ms │

└────────────┬────────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────┐

│ 5. TTS (Sarvam or ElevenLabs) │

│ Text → Audio with selected voice │

│ Latency: 400ms-1500ms │

└────────────┬────────────────────────────────────────┘

│

▼

AUDIO RESPONSE

```

\*\*CRITICAL\*\*: Only \*\*ONE\*\* LLM call per interaction (not two)!

---

# 3. QUALITY-LATENCY OPTIMIZATION

## 3.1 The 5-Level Slider

### 🟢 Level 1: Maximum Quality

```javascript

{

target\_latency: "3-4s",

accuracy: 98%,

techniques: {

streaming: false, // Wait for complete audio

speculation: false, // Wait for full transcript

caching: "exact\_only", // Only exact matches

rag\_depth: "deep", // 10 chunks

llm\_temperature: 0.3, // Very predictable

parallel: false // Sequential (safer)

}

}

```

### 🟡 Level 2: Balanced Quality

```javascript

{

target\_latency: "2-3s",

accuracy: 95%,

techniques: {

streaming: false,

speculation: false,

caching: "semantic", // Similarity matching

rag\_depth: "medium", // 5 chunks

llm\_temperature: 0.5,

parallel: true // Where safe

}

}

```

### 🟠 Level 3: Balanced (DEFAULT)

```javascript

{

target\_latency: "1.5-2s",

accuracy: 90%,

techniques: {

streaming: "partial", // 80% confidence

speculation: "conservative", // After 5 words

caching: "aggressive",

rag\_depth: "light", // 3 chunks

llm\_temperature: 0.7,

parallel: true

}

}

```

### 🔴 Level 4: Balanced Speed

```javascript

{

target\_latency: "1-1.5s",

accuracy: 85%,

techniques: {

streaming: "aggressive", // 60% confidence

speculation: "moderate", // After 3 words

caching: "very\_aggressive",

rag\_depth: "minimal", // 1-2 chunks

llm\_temperature: 0.8,

shortcuts: true // Fast path

}

}

```

### ⚡ Level 5: Maximum Speed

```javascript

{

target\_latency: "0.7-1s",

accuracy: 75%,

techniques: {

streaming: "very\_aggressive", // 40% confidence

speculation: "aggressive", // After 2 words

caching: "extreme",

rag\_depth: "skip", // No RAG

llm\_temperature: 0.9,

shortcuts: true,

prefetch: true,

truncate: true // 50 word max

},

warnings: ["⚠️ May sacrifice accuracy", "⚠️ Not for critical use"]

}

```

## 3.2 Frontend Implementation

```typescript

<PerformanceSettings>

<Slider

value={optimizationLevel}

onChange={setOptimizationLevel}

marks={[

{ value: 0, label: "Quality (3-4s)" },

{ value: 25, label: "Balanced Quality (2-3s)" },

{ value: 50, label: "Balanced (1.5-2s)" },

{ value: 75, label: "Speed (1-1.5s)" },

{ value: 100, label: "Max Speed (0.7-1s)" }

]}

/>

<Gauges>

<Gauge label="Accuracy" value={configs[level].accuracy} />

<Gauge label="Speed" value={100 - configs[level].accuracy} />

</Gauges>

{warnings[level].map(w => <Warning>{w}</Warning>)}

</PerformanceSettings>

```

---

# 4. VOICE PIPELINE COMPONENTS

## 4.1 ASR (Sarvam - Fixed)

```python

# Configuration

ASR\_CONFIG = {

"provider": "sarvam", # Non-configurable

"model": "saaras:v2.5",

"behavior": "auto\_detect\_language",

"output": "english\_transcript + language\_code",

"supported\_languages": 23 # All Indian languages + English

}

# Implementation

async def transcribe(audio\_file):

response = sarvam\_client.speech\_to\_text.translate(

file=audio\_file,

model="saaras:v2.5"

)

return {

"transcript": response["transcript"], # In English

"language": response["language"] # e.g., "hi-IN"

}

```

## 4.2 LLM (Configurable)

```python

# Single LLM Call Architecture

async def generate\_response(user\_transcript, rag\_context=""):

"""

ONE LLM call handles:

1. Intent understanding

2. Guardrail checking (via prompt)

3. Knowledge utilization

4. Response generation

"""

system\_prompt = f"""You are a helpful AI assistant.

GUIDELINES:

- Be concise (under 100 words for voice)

- Professional but friendly

GUARDRAILS:

- Only answer about our products/services

- Never provide medical/legal/financial advice

- Never generate PII

{f"KNOWLEDGE BASE:\n{rag\_context}" if rag\_context else ""}

"""

response = llm\_client.chat.completions(

model="sarvam-m",

messages=[

{"role": "system", "content": system\_prompt},

{"role": "user", "content": user\_transcript}

],

temperature=0.7

)

return response["choices"][0]["message"]["content"]

```

## 4.3 Translation (Colloquial-Aware)

```python

async def translate(text, target\_language, colloquial\_config):

"""Translate with colloquial/code-mixing support"""

# Determine mode based on formality level

if colloquial\_config["enabled"]:

mode = get\_mode\_for\_formality(colloquial\_config["formality\_level"])

else:

mode = "formal"

# Translate

result = sarvam\_client.text.translate(

input=text,

source\_language\_code="en-IN",

target\_language\_code=target\_language,

mode=mode

)

translated = result["translated\_text"]

# Apply code-mixing

if colloquial\_config["code\_mixing\_enabled"]:

translated = apply\_code\_mixing(translated, colloquial\_config)

return translated

```

## 4.4 TTS (Dual Provider)

```python

async def synthesize(text, language, provider, voice\_settings):

"""TTS with automatic fallback"""

try:

if provider == "elevenlabs":

if language in ELEVENLABS\_SUPPORTED:

return await elevenlabs\_tts(text, voice\_settings)

else:

# Fallback to Sarvam for unsupported languages

return await sarvam\_tts(text, language, voice\_settings)

else:

return await sarvam\_tts(text, language, voice\_settings)

except Exception as e:

# Always fallback to Sarvam

return await sarvam\_tts(text, language, voice\_settings)

```

---

# 5. BARGE-IN SYSTEM

## 5.1 How It Works

```

User interrupts bot → VAD detects speech → Send INTERRUPT signal →

Cancel TTS → Preserve context → Ready for new input

```

## 5.2 Configuration

```javascript

BARGE\_IN\_CONFIG = {

enabled: true,

vad\_sensitivity: 0.7, // 0-1

min\_speech\_duration\_ms: 300,

sustained\_duration\_ms: 500, // Prevent false positives

interruption\_delay\_ms: 100, // Small delay

play\_acknowledgment: true,

resume\_on\_false\_positive: true

}

```

## 5.3 Frontend UI

```typescript

<BargeInSettings>

<Toggle label="Enable Barge-In" value={enabled} onChange={setEnabled} />

{enabled && (

<>

<Slider

label="Voice Detection Sensitivity"

value={vadSensitivity}

min={0} max={1}

leftLabel="Less Sensitive"

rightLabel="More Sensitive"

/>

<Slider

label="Interruption Delay"

value={delay}

min={0} max={500}

unit="ms"

/>

<Toggle

label="Resume After False Trigger"

value={resumeAfterFalse}

/>

</>

)}

</BargeInSettings>

```

---

# 6. SIP TRUNK INTEGRATION

## 6.1 Supported Providers

- ✅ Twilio

- ✅ Vonage (Nexmo)

- ✅ Bandwidth

- ✅ Custom SIP Server

## 6.2 Configuration Example (Twilio)

```javascript

SIP\_CONFIG = {

provider: "twilio",

connection: {

account\_sid: "ACxxxxxxxx",

auth\_token: "your\_token",

phone\_number: "+15551234567"

},

webhooks: {

on\_call\_start: "https://yourapp.com/webhooks/call-start",

on\_call\_end: "https://yourapp.com/webhooks/call-end"

},

ivr: {

enabled: true,

greeting: "Press 1 for English, Press 2 for Hindi"

}

}

```

## 6.3 Admin UI

```typescript

<SIPConfiguration>

<ProviderSelector

options={["twilio", "vonage", "bandwidth", "custom"]}

value={provider}

onChange={setProvider}

/>

{provider === "twilio" && (

<TwilioConfig>

<Input label="Account SID" value={sid} onChange={setSid} />

<PasswordInput label="Auth Token" value={token} onChange={setToken} />

<Input label="Phone Number" value={phone} onChange={setPhone} />

<TestButton onClick={testConnection}>Test Connection</TestButton>

</TwilioConfig>

)}

{provider === "custom" && (

<CustomSIPConfig>

<Input label="SIP Domain" value={domain} />

<Input label="Username" value={username} />

<PasswordInput label="Password" value={password} />

<Select label="Transport" options={["UDP", "TCP", "TLS"]} />

</CustomSIPConfig>

)}

</SIPConfiguration>

```

---

# 7. COLLOQUIAL LANGUAGE

## 7.1 Two Main Controls

### A) Formality Slider (0-100)

```

0: Very Formal → "आपका ऑर्डर संसाधित किया जा रहा है"

50: Conversational → "Aapka order process ho raha hai"

100: Very Informal → "Aapka order chal raha hai boss"

```

### B) Code-Mixing Toggle + Ratio

```

English Ratio: 30%

"Aapka order process ho raha hai" (30% English, 70% Hindi)

English Ratio: 70%

"Your order process ho raha hai" (70% English, 30% Hindi)

```

## 7.2 Frontend UI

```typescript

<LanguageSettings>

{/\* Formality \*/}

<Toggle label="Use Colloquial Language" value={colloquialEnabled} />

{colloquialEnabled && (

<Slider

label="Formality Level"

value={formalityLevel}

min={0} max={100}

marks={[

{ value: 0, label: "Very Formal" },

{ value: 50, label: "Conversational" },

{ value: 100, label: "Very Informal" }

]}

/>

)}

{/\* Code-Mixing \*/}

<Toggle label="Mix English with Local Language" value={codeMixingEnabled} />

{codeMixingEnabled && (

<>

<Slider

label="English Mix Ratio"

value={englishRatio}

min={0} max={100}

leftLabel="More Local Language"

rightLabel="More English"

/>

<CheckboxGroup label="Keep in English:">

<Checkbox label="Technology Terms" value={domains.tech} />

<Checkbox label="Business Terms" value={domains.business} />

</CheckboxGroup>

</>

)}

</LanguageSettings>

```

---

# 8. VOICE SELECTION

## 8.1 Available Voices

### Sarvam Voices

```

Female: anushka, manisha, vidya, arya

Male: abhilash, karun, hitesh

Languages: 20+ Indian languages

```

### ElevenLabs Voices

```

Female: Rachel, Bella

Male: Adam

Languages: English (limited Indian language support)

```

## 8.2 Frontend UI with Preview

```typescript

<VoiceSettings>

<ProviderSelector value={ttsProvider} onChange={setProvider}>

<Option value="sarvam">Sarvam AI (20+ languages)</Option>

<Option value="elevenlabs">ElevenLabs (Premium)</Option>

</ProviderSelector>

<VoiceGallery>

{voices.map(voice => (

<VoiceCard key={voice.id} selected={selectedVoice === voice.id}>

<VoiceName>{voice.name}</VoiceName>

<VoiceMetadata>{voice.gender} • {voice.characteristics}</VoiceMetadata>

{/\* Standard Preview \*/}

<PreviewButton onClick={() => playPreview(voice.id)}>

▶ Preview

</PreviewButton>

{/\* Custom Text Preview \*/}

<CustomPreview>

<Input

placeholder="Type custom text..."

value={customTexts[voice.id]}

onChange={(e) => setCustomText(voice.id, e.target.value)}

/>

<PlayButton onClick={() => playCustom(voice.id, customTexts[voice.id])}>

▶

</PlayButton>

</CustomPreview>

</VoiceCard>

))}

</VoiceGallery>

{/\* Voice Tuning (Provider-Specific) \*/}

{ttsProvider === "sarvam" && (

<VoiceTuning>

<Slider label="Pitch" value={pitch} min={-0.75} max={0.75} />

<Slider label="Speed" value={speed} min={0.3} max={3} />

<Slider label="Volume" value={volume} min={0} max={3} />

</VoiceTuning>

)}

</VoiceSettings>

```

---

# 9. NOISE HANDLING

## 9.1 Three Layers

```

Layer 1: Client (WebRTC) → Noise suppression, echo cancellation

Layer 2: Backend Processing → High-pass filter, Sarvam preprocessing

Layer 3: Quality Monitoring → Real-time SNR, clipping detection

```

## 9.2 Frontend UI

```typescript

<AudioProcessing>

{/\* Quick Presets \*/}

<PresetButtons>

<Preset onClick={() => applyPreset("quiet")}>Quiet</Preset>

<Preset onClick={() => applyPreset("moderate")}>Moderate</Preset>

<Preset onClick={() => applyPreset("noisy")}>Noisy</Preset>

</PresetButtons>

{/\* Individual Controls \*/}

<Toggle label="Noise Suppression" value={noiseSuppression} />

{noiseSuppression && (

<Select value={level} options={["low", "medium", "high", "very\_high"]} />

)}

<Toggle label="Echo Cancellation" value={echoCancellation} />

<Toggle label="Auto Volume" value={autoGain} />

<Toggle label="Sarvam AI Preprocessing" value={sarvamPreprocessing} badge="Premium" />

{/\* Real-Time Quality \*/}

<QualityMonitor>

<QualityBar value={audioQuality} />

<QualityLabel>

{audioQuality > 80 ? "Excellent" :

audioQuality > 60 ? "Good" : "Poor"}

</QualityLabel>

{audioQuality < 50 && (

<Warning>

Low quality. Try:

• Quieter location

• Closer to mic

• Enable noise suppression

</Warning>

)}

</QualityMonitor>

</AudioProcessing>

```

---

# 10. BACKGROUND NOISE INJECTION

## 10.1 For Testing Only

Inject realistic noise to test bot performance in challenging conditions.

## 10.2 Noise Types

```

1. Call Center → Chatter, typing, phone rings

2. Café → Ambient chatter, coffee machine, music

3. Street → Traffic, horns, footsteps

4. Office → AC, computers, distant voices

5. Custom → Upload your own

```

## 10.3 Frontend UI

```typescript

<NoiseInjection>

<InfoBanner>Add realistic noise to test performance</InfoBanner>

<NoiseTypeGrid>

<NoiseCard value="none">🔇 No Noise</NoiseCard>

<NoiseCard value="call\_center">📞 Call Center</NoiseCard>

<NoiseCard value="cafe">☕ Café</NoiseCard>

<NoiseCard value="street">🚗 Street</NoiseCard>

<NoiseCard value="office">🏢 Office</NoiseCard>

</NoiseTypeGrid>

{noiseType !== "none" && (

<>

<Slider

label="Noise Volume"

value={noiseVolume}

min={0} max={1}

/>

{noiseType === "call\_center" && (

<CallCenterOptions>

<Toggle label="Background Chatter" value={chatter} />

<Toggle label="Keyboard Typing" value={typing} />

<Toggle label="Phone Ringing" value={phoneRings} />

</CallCenterOptions>

)}

<PreviewButton onClick={previewNoise}>▶ Preview Noise</PreviewButton>

</>

)}

</NoiseInjection>

```

---

# 11. RAG SYSTEM

## 11.1 Architecture

```

Documents → Chunking (1000 tokens) → Embedding (OpenAI) →

Vector DB (Pinecone) → Retrieval (top 5) → LLM Context

```

## 11.2 Configuration

```python

RAG\_CONFIG = {

"enabled": True,

"chunking": {"size": 1000, "overlap": 200},

"embedding": {"provider": "openai", "model": "text-embedding-3-small"},

"vector\_db": {"provider": "pinecone", "index": "chatbot-kb"},

"retrieval": {"top\_k": 5, "threshold": 0.7}

}

```

## 11.3 Implementation

```python

async def retrieve\_context(query):

# Embed query

query\_embedding = await openai.embed([query])

# Search vector DB

results = await pinecone.query(

vector=query\_embedding[0],

top\_k=5,

filter={"active": True}

)

# Format context

context = "\n\n".join([

f"[Source: {r.metadata['source']}]\n{r.metadata['text']}"

for r in results.matches if r.score > 0.7

])

return context

```

---

# 12. GUARDRAILS

## 12.1 Three-Layer System

### Layer 1: Pre-LLM (Fast Checks)

```python

# Block keywords

BLOCKED = ["medical advice", "legal advice", "how to make bomb"]

def pre\_check(user\_input):

for keyword in BLOCKED:

if keyword in user\_input.lower():

return {"blocked": True, "response": "I can only help with product questions."}

return {"blocked": False}

```

### Layer 2: LLM Prompt (Instructions)

```python

system\_prompt = """

STRICT RULES:

- Only answer about our products/services

- Never provide medical/legal/financial advice

- Never generate harmful content

- Never share PII

"""

```

### Layer 3: Post-LLM (Validation)

```python

def post\_check(response):

# Check for PII

if re.search(r"\d{4}[\s-]?\d{4}[\s-]?\d{4}[\s-]?\d{4}", response):

return {"valid": False, "reason": "PII detected"}

# Check length

if len(response.split()) > 150:

return {"valid": False, "reason": "Too long"}

return {"valid": True}

```

---

# 13. FRONTEND ARCHITECTURE

## 13.1 Tech Stack

```

React 18 + TypeScript

Tailwind CSS

Redux Toolkit (state)

Socket.IO (WebSocket)

WebRTC (audio)

Recharts (charts)

```

## 13.2 Component Structure

```

src/

├── components/

│ ├── chat/

│ │ ├── ChatInterface.tsx

│ │ └── MicButton.tsx

│ ├── settings/

│ │ ├── PerformanceSettings.tsx

│ │ ├── LanguageSettings.tsx

│ │ └── VoiceSettings.tsx

│ └── admin/

│ └── Dashboard.tsx

├── hooks/

│ ├── useVoiceChat.ts

│ └── useAudioProcessing.ts

└── services/

├── audio/

│ └── AudioProcessor.ts

└── api/

└── chatbotApi.ts

```

---

# 14. BACKEND ARCHITECTURE

## 14.1 Tech Stack

```

Python FastAPI

WebSocket (real-time)

PostgreSQL (data)

Redis (cache)

Pinecone (vectors)

```

## 14.2 Structure

```

backend/

├── main.py

├── services/

│ ├── asr\_service.py

│ ├── llm\_service.py

│ ├── tts\_service.py

│ ├── rag\_service.py

│ └── guardrail\_service.py

├── models/

│ └── schemas.py

└── utils/

└── optimization.py

```

---

# 15. API SPECIFICATIONS

## 15.1 Main Endpoint

```

POST /api/v1/chat/voice

Headers:

Authorization: Bearer {api\_key}

Body (multipart/form-data):

audio\_file: <binary WAV/MP3>

config: {

optimization\_level: "balanced",

tts\_provider: "sarvam",

enable\_rag: true

}

Response:

{

"status": "success",

"data": {

"user\_transcript": "What are your hours?",

"detected\_language": "hi-IN",

"response\_text": "हमारे घंटे...",

"response\_audio": "base64\_audio",

"metadata": {

"asr\_latency\_ms": 850,

"llm\_latency\_ms": 1200,

"tts\_latency\_ms": 900,

"total\_latency\_ms": 2950

}

}

}

```

---

# 16. DATABASE SCHEMA

```sql

-- Users

CREATE TABLE users (

user\_id VARCHAR(50) PRIMARY KEY,

email VARCHAR(255) UNIQUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Conversations

CREATE TABLE conversations (

conversation\_id VARCHAR(50) PRIMARY KEY,

user\_id VARCHAR(50) REFERENCES users(user\_id),

started\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

language\_code VARCHAR(10),

optimization\_level VARCHAR(20),

total\_latency\_ms INTEGER

);

-- Messages

CREATE TABLE messages (

message\_id VARCHAR(50) PRIMARY KEY,

conversation\_id VARCHAR(50) REFERENCES conversations(conversation\_id),

role VARCHAR(20),

transcript TEXT,

asr\_latency\_ms INTEGER,

llm\_latency\_ms INTEGER,

tts\_latency\_ms INTEGER,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Knowledge Base Documents

CREATE TABLE kb\_documents (

doc\_id VARCHAR(50) PRIMARY KEY,

title VARCHAR(255),

chunk\_count INTEGER,

uploaded\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Guardrail Rules

CREATE TABLE guardrail\_rules (

rule\_id VARCHAR(50) PRIMARY KEY,

name VARCHAR(255),

type VARCHAR(50),

enabled BOOLEAN DEFAULT TRUE,

keywords TEXT[]

);

-- Cost Tracking

CREATE TABLE cost\_tracking (

id SERIAL PRIMARY KEY,

date DATE,

service VARCHAR(50),

cost\_usd DECIMAL(10, 6)

);

```

---

# 17. TEST AGENT MODE

## 17.1 Key Principle

\*\*NO MOCKS, NO DUMMIES - Always use real production APIs\*\*

## 17.2 Features

- ✅ Real Sarvam APIs (ASR, LLM, TTS)

- ✅ Scenario library with automated execution

- ✅ Real-time metrics collection

- ✅ Automated evaluation (ASR accuracy, latency, relevance)

- ✅ Audio recording of all tests

## 17.3 Test Scenario Format

```json

{

"id": "order\_status",

"name": "Order Status Flow",

"settings": {

"optimization\_level": "balanced",

"language": "en-IN"

},

"steps": [

{

"step": 1,

"user\_speaks": "What's my order status?",

"expected": {

"detected\_language": "en-IN",

"asr\_latency\_max\_ms": 1500

}

},

{

"step": 2,

"bot\_responds": true,

"expected": {

"response\_contains": ["order", "number"],

"llm\_latency\_max\_ms": 2000

}

}

],

"success\_criteria": {

"all\_steps\_passed": true,

"avg\_latency\_max\_ms": 3000

}

}

```

## 17.4 Frontend UI

```typescript

<TestAgentInterface>

{/\* Split Layout \*/}

<SplitLayout>

{/\* Left: Live Agent \*/}

<AgentPanel>

<LiveAgent

realAPIs={true} // ALWAYS TRUE

onMetrics={updateMetrics}

/>

<ConversationDisplay messages={messages} />

<TestControls>

<Button onClick={startTest}>Start</Button>

<Button onClick={stopTest}>Stop</Button>

<NoiseControl>

<Select label="Noise" value={noiseType}

options={["none", "call\_center", "cafe"]} />

<Slider label="Volume" value={noiseVolume} />

</NoiseControl>

</TestControls>

</AgentPanel>

{/\* Right: Metrics \*/}

<MetricsPanel>

<RealtimeMetrics>

<Metric label="ASR" value={metrics.asrLatency} target={1500} />

<Metric label="LLM" value={metrics.llmLatency} target={2000} />

<Metric label="Total" value={metrics.totalLatency} target={5000} />

<Metric label="Accuracy" value={metrics.accuracy} target={90} />

</RealtimeMetrics>

<Chart title="Latency Breakdown">

<StackedBar data={metrics.breakdown} />

</Chart>

<TestResults>

<Stat label="Tests" value={results.total} />

<Stat label="Passed" value={results.passed} color="green" />

<Stat label="Failed" value={results.failed} color="red" />

</TestResults>

</MetricsPanel>

</SplitLayout>

{/\* Bottom: Scenario Library \*/}

<ScenarioPanel>

{scenarios.map(s => (

<ScenarioCard

key={s.id}

scenario={s}

onClick={() => runScenario(s.id)}

/>

))}

</ScenarioPanel>

</TestAgentInterface>

```

---

# 18. DEPLOYMENT

## 18.1 Docker

```dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

```

## 18.2 Kubernetes

```yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: voice-chatbot

spec:

replicas: 3

template:

spec:

containers:

- name: app

image: voice-chatbot:latest

ports:

- containerPort: 8000

env:

- name: SARVAM\_API\_KEY

valueFrom:

secretKeyRef:

name: api-secrets

key: sarvam-key

resources:

requests:

memory: "1Gi"

cpu: "500m"

limits:

memory: "4Gi"

cpu: "2000m"

---

apiVersion: v1

kind: Service

metadata:

name: voice-chatbot

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 8000

selector:

app: voice-chatbot

---

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: voice-chatbot-hpa

spec:

scaleTargetRef:

kind: Deployment

name: voice-chatbot

minReplicas: 3

maxReplicas: 20

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 70

```

---

# 19. COST MANAGEMENT

## 19.1 Cost Per Conversation

```

Sarvam Stack (All Sarvam):

- ASR: $0.002

- LLM: $0.005

- Translation: $0.001 (cached: $0.0007)

- TTS: $0.003 (cached: $0.0024)

- Total: ~$0.011

With Caching (30% hit rate):

- Total: ~$0.009 per conversation

Monthly (10,000 conversations):

- API: $90

- Infrastructure: $150

- Total: ~$240/month

```

## 19.2 Budget Alerts

```python

BUDGET\_CONFIG = {

"monthly\_limit": 500, # USD

"alert\_thresholds": [0.5, 0.75, 0.9, 1.0],

"actions": {

0.9: "send\_warning\_email",

1.0: "rate\_limit\_non\_critical\_requests"

}

}

```

---

# 20. MONITORING

## 20.1 Key Metrics

```javascript

METRICS = {

// Performance

avg\_latency\_ms: "gauge",

p95\_latency\_ms: "histogram",

error\_rate: "gauge",

// Quality

asr\_accuracy: "gauge",

user\_satisfaction: "gauge",

// Business

total\_conversations: "counter",

active\_users: "gauge",

// Cost

total\_cost\_usd: "counter",

cost\_per\_conversation: "gauge"

}

```

## 20.2 Dashboards

- Real-time performance

- Quality trends over time

- Cost tracking

- Language distribution

---

# 21. SECURITY

## 21.1 Data Encryption

```python

ENCRYPTION = {

"at\_rest": "AES-256-GCM",

"in\_transit": "TLS 1.3",

"key\_rotation": "90 days"

}

```

## 21.2 GDPR Compliance

```python

class GDPRService:

async def export\_user\_data(self, user\_id):

"""Right to access"""

return {

"conversations": await get\_conversations(user\_id),

"settings": await get\_settings(user\_id),

"exported\_at": datetime.now()

}

async def delete\_user\_data(self, user\_id):

"""Right to erasure"""

await delete\_from\_database(user\_id)

await delete\_audio\_files(user\_id)

await delete\_from\_cache(user\_id)

return {"status": "deleted"}

```

---

# 22. IMPLEMENTATION ROADMAP

## Phase 1: MVP (Weeks 1-4)

\*\*Goal\*\*: Working voice chatbot

### Week 1-2: Core Infrastructure

- [ ] Backend API (FastAPI)

- [ ] Database setup (PostgreSQL)

- [ ] Sarvam integration (ASR, LLM, TTS)

- [ ] Basic WebSocket

### Week 3-4: Basic UI

- [ ] React frontend

- [ ] Audio recording

- [ ] Simple chat interface

- [ ] Sequential pipeline

\*\*Deliverable\*\*: Basic working chatbot

---

## Phase 2: Optimization (Weeks 5-8)

\*\*Goal\*\*: <2s latency with quality controls

### Week 5-6: Performance

- [ ] Quality-Latency slider (5 levels)

- [ ] Redis caching layer

- [ ] Connection pooling

- [ ] Parallel processing where safe

### Week 7-8: Quality Features

- [ ] Colloquial language controls

- [ ] Code-mixing implementation

- [ ] Voice selection UI

- [ ] Voice preview system

- [ ] Audio preprocessing (WebRTC)

\*\*Deliverable\*\*: Optimized chatbot with <2s latency

---

## Phase 3: Advanced Features (Weeks 9-12)

\*\*Goal\*\*: Full-featured with RAG and safety

### Week 9-10: Interaction

- [ ] Barge-in system (VAD)

- [ ] Noise handling (multi-layer)

- [ ] Background noise injection

- [ ] Audio quality monitoring

### Week 11-12: AI & Safety

- [ ] Vector database (Pinecone)

- [ ] Document processing

- [ ] RAG implementation

- [ ] 3-layer guardrails

- [ ] Content filtering

\*\*Deliverable\*\*: Full-featured chatbot

---

## Phase 4: Enterprise (Weeks 13-16)

\*\*Goal\*\*: Production-ready with telephony

### Week 13-14: SIP Integration

- [ ] FreeSWITCH setup

- [ ] Twilio adapter

- [ ] Vonage adapter

- [ ] Custom SIP adapter

- [ ] IVR system

- [ ] Call recording

### Week 15-16: Admin & Testing

- [ ] Admin dashboard

- [ ] Analytics views

- [ ] Test Agent Mode (real APIs)

- [ ] Scenario library

- [ ] Automated evaluation

\*\*Deliverable\*\*: Production-ready system

---

## Phase 5: Launch (Weeks 17-20)

\*\*Goal\*\*: Secure, documented, launched

### Week 17-18: Security

- [ ] Security audit

- [ ] GDPR compliance

- [ ] Data encryption

- [ ] API key rotation

- [ ] Secrets management

### Week 19-20: Documentation & Launch

- [ ] API documentation

- [ ] User guides

- [ ] Admin tutorials

- [ ] Load testing

- [ ] Production deployment

- [ ] Monitoring setup

\*\*Deliverable\*\*: Launched product ✅

---

# 23. SUCCESS METRICS

## 23.1 Technical Targets

| Metric | Target | Measurement |

|--------|--------|-------------|

| \*\*Avg Latency\*\* | < 2s | P50 total time |

| \*\*P95 Latency\*\* | < 3s | 95th percentile |

| \*\*ASR Accuracy\*\* | > 90% | Word Error Rate |

| \*\*Uptime\*\* | 99.5% | Monthly availability |

| \*\*Error Rate\*\* | < 2% | Failed/total requests |

| \*\*Cache Hit Rate\*\* | > 30% | Cached/total |

## 23.2 Business Targets

| Metric | Target |

|--------|--------|

| \*\*User Satisfaction\*\* | > 80% |

| \*\*Completion Rate\*\* | > 85% |

| \*\*Cost/Conversation\*\* | < $0.015 |

| \*\*Concurrent Users\*\* | 100+ |

| \*\*Languages Supported\*\* | 20+ |

---

# 24. APPENDICES

## Appendix A: Environment Variables

```bash

# Core

ENVIRONMENT=production

DEBUG=false

# API Keys

SARVAM\_API\_KEY=your\_sarvam\_key

OPENAI\_API\_KEY=your\_openai\_key

ELEVENLABS\_API\_KEY=your\_elevenlabs\_key

PINECONE\_API\_KEY=your\_pinecone\_key

# Database

DATABASE\_URL=postgresql://user:pass@localhost:5432/chatbot

REDIS\_URL=redis://localhost:6379/0

# Pinecone

PINECONE\_ENVIRONMENT=us-west1-gcp

PINECONE\_INDEX\_NAME=chatbot-kb

# Security

SECRET\_KEY=your\_secret\_key\_32\_chars\_min

ALLOWED\_ORIGINS=https://yourdomain.com

# Features

DEFAULT\_OPTIMIZATION\_LEVEL=balanced

DEFAULT\_TTS\_PROVIDER=sarvam

ENABLE\_RAG=true

ENABLE\_GUARDRAILS=true

# Monitoring

PROMETHEUS\_ENABLED=true

SENTRY\_DSN=your\_sentry\_dsn

```

## Appendix B: Supported Languages (23 total)

```

Hindi (hi-IN), Bengali (bn-IN), Kannada (kn-IN), Malayalam (ml-IN),

Marathi (mr-IN), Odia (od-IN), Punjabi (pa-IN), Tamil (ta-IN),

Telugu (te-IN), Gujarati (gu-IN), English (en-IN), Assamese (as-IN),

Bodo (brx-IN), Dogri (doi-IN), Konkani (kok-IN), Kashmiri (ks-IN),

Maithili (mai-IN), Manipuri (mni-IN), Nepali (ne-IN), Sanskrit (sa-IN),

Santali (sat-IN), Sindhi (sd-IN), Urdu (ur-IN)

```

## Appendix C: Quick Start Commands

```bash

# Clone repository

git clone https://github.com/yourorg/voice-chatbot.git

cd voice-chatbot

# Setup environment

cp .env.template .env

# Edit .env with your API keys

# Start with Docker Compose

docker-compose up -d

# Or run locally

pip install -r requirements.txt

uvicorn main:app --reload

# Frontend

cd frontend

npm install

npm run dev

```

## Appendix D: Testing Commands

```bash

# Run unit tests

pytest tests/

# Run integration tests

pytest tests/integration/

# Run load tests

locust -f load\_test.py --users 100

# Test specific scenario

python test\_agent.py --scenario order\_status

```

## Appendix E: Common Issues & Solutions

### Issue: High Latency

\*\*Solutions:\*\*

- Lower optimization level (move slider right)

- Enable caching

- Check network latency to Sarvam APIs

- Review database query performance

### Issue: Poor ASR Accuracy

\*\*Solutions:\*\*

- Enable noise suppression

- Enable Sarvam preprocessing

- Check audio quality (SNR > 10dB)

- Ensure 16kHz sample rate

### Issue: TTS Sounds Robotic

\*\*Solutions:\*\*

- Try different voice

- Adjust pace/pitch settings

- Use ElevenLabs for English

- Enable Sarvam preprocessing

---

# FINAL CHECKLIST

## Before Implementation

- [ ] Review all 24 sections

- [ ] Confirm API keys available

- [ ] Approve budget ($250-500/month for 10k conversations)

- [ ] Assign team (2 backend, 1 frontend, 1 QA)

- [ ] Setup infrastructure (AWS/GCP/Azure)

## Before Launch

- [ ] Security audit passed

- [ ] Load testing completed (100 concurrent users)

- [ ] All success metrics defined

- [ ] Monitoring dashboards live

- [ ] Documentation complete

- [ ] Rollback plan ready

---

# DOCUMENT SUMMARY

\*\*Total Sections\*\*: 24

\*\*Total Pages\*\*: ~80

\*\*Implementation Time\*\*: 20 weeks

\*\*Estimated Cost\*\*: $250-500/month (10k conversations)

\*\*All requirements documented in ONE place:\*\*

✅ Quality-Latency optimization (5 levels)

✅ Barge-in system

✅ SIP trunk integration (all providers)

✅ Test agent mode (real APIs only)

✅ Colloquial language + code-mixing

✅ Voice selection with preview

✅ Multi-layer noise handling

✅ Background noise injection

✅ RAG system

✅ Multi-layer guardrails

✅ Complete frontend architecture

✅ Complete backend architecture

✅ API specifications

✅ Database schema

✅ Deployment guide

✅ Security & compliance

✅ Cost management

✅ Monitoring & analytics

✅ 20-week roadmap

---

\*\*STATUS\*\*: ✅ \*\*COMPLETE & READY FOR IMPLEMENTATION\*\*

\*\*Version\*\*: 3.0 FINAL CLEAN

\*\*Date\*\*: October 10, 2025

\*\*For questions or clarifications, contact the project team.\*\*

---

\*\*END OF DOCUMENT\*\*