MedJourney MVP 技术架构深度研究报告

版本: 1.0
日期: 2025年7月25日
作者: MiniMax Agent

摘要

本报告旨在为 MedJourney MVP（一款面向 Alzheimer's 患者的 AI 陪伴 Web 应用）提供一个全面、深入的技术架构研究。报告基于 2025 年的最新技术标准和最佳实践，围绕四大核心领域展开：TEN Framework 集成、Agora 实时通信、RAG 向量检索技术以及 Alzheimer's 问诊场景的特定技术需求。报告将提供每个领域的详细实现方案、技术选型建议、集成架构图、潜在挑战与解决方案，并对开发工作进行初步评估。

---

1. TEN Framework 集成研究

经研究确认，**TEN (Transformative Extensions Network) Framework** 是一个真实存在的、由 Agora 支持的开源框架，专为构建实时、多模态对话式 AI 设计。它非常适合 MedJourney 项目的需求。

1.1. TEN Framework 架构与核心功能

TEN 的核心是一个围绕“扩展”构建的模块化生态系统，允许开发者灵活地组合各种 AI 能力。

核心架构组件:

- **TEN Agent**: 这是框架的核心运行时，负责管理对话流程、状态和与各个扩展的交互。它在 Node.js 环境中运行。
- **TMAN Designer**: 一个低代码/无代码的图形化界面，用于设计和配置 AI Agent 的对话逻辑图（Graphs）。开发者可以通过拖拽节点的方式定义工作流，例如从接收音频、语音识别、语言模型处理到语音合成的整个流程。

- **扩展 (Extensions)**: 这些是实现具体功能的插件化模块。TEN 提供了多种内置扩展，并允许自定义开发。关键扩展包括：
  - **STT (Speech-to-Text)**: 支持集成 Deepgram 等第三方语音识别服务。
  - **LLM (Large Language Model)**: 支持集成 OpenAI (GPT系列)、Gemini 等任何与 OpenAI API 兼容的大语言模型服务。
  - **TTS (Text-to-Speech)**: 支持集成 ElevenLabs 等高质量的语音合成服务。
  - **VAD (Voice Activity Detection)**: 一个轻量级的流式声学活动检测模型，用于判断用户是否在说话，是实现自然对话和打断机制的关键。
  - **Turn Detection**: 更高级的模型，用于在双工通信中智能地检测对话轮次的转换点，实现更精准的打断和响应。
- **多模态能力**: 框架明确支持语音、图像和虚拟形象。例如，它可以集成 Google Gemini 的多模态能力进行实时视觉分析，或通过 Trulience 服务驱动 3D 虚拟形象。

1.2. 多模态 AI Agent 实现方式

TEN Framework 通过其灵活的扩展机制，可以便捷地实现多模态交互。

- **语音 (核心)**: 这是基础交互。流程为：`客户端 -> Agora SDK 采集音频 -> TEN Agent -> VAD/Turn Detection 判断 -> STT 扩展 -> LLM 扩展 -> TTS 扩展 -> Agora SDK 播放音频 -> 客户端`。
- **文本**: 用户也可以通过文本输入。这将跳过 STT/TTS 环节，直接将文本送入 LLM 扩展进行处理。
- **图像**: 可以添加一个图像输入模块（例如，文件上传或摄像头捕获）。图像数据可以被送入一个多模态 LLM（如 Gemini a），以实现“看图说话”或识别物体的功能。TEN 的“StoryTeller”扩展就是一个图像生成的例子，展示了其视觉处理能力。

1.3. JavaScript/Node.js 集成最佳实践

TEN Agent 本身就是基于 Node.js 构建的，因此集成是无缝的。

- **环境**: 使用 Node.js v18 (LTS) 和 Docker/Docker Compose 是官方推荐的最佳实践。
- **配置**: 所有的外部服务 API 密钥（Agora, OpenAI, Deepgram, ElevenLabs 等）都通过根目录下的 `.env` 文件进行管理，实现了代码与配置的分离。

- **部署**: 官方提供了详细的 Docker-Compose 配置，可以一键启动包括 TEN Agent 和 TMAN Designer 在内的所有服务。同时，也支持将定制好的 Agent 打包成独立的 Docker 镜像，用于云端自托管部署。
- **本地开发**: 官方提供了在本地或 GitHub Codespace 中设置开发环境的完整指南，便于快速启动和调试。

1.4. STT、LLM、TTS 插件化配置

TEN 的设计哲学是“厂商中立”，其插件化配置是核心优势。

- **实现方式**: 在 TMAN Designer 中，每个外部服务（STT/LLM/TTS）都是一个可配置的“节点”或“扩展”。开发者可以从节点市场中选择所需的服务。
- **配置过程**:
  1. 在 TMAN Designer 的画布中，拖入一个 LLM 节点（例如，`OpenAI Chat Completion`）。
  2. 点击该节点，在右侧的属性面板中，填入相应的 API Endpoint、模型名称 (e.g., `gpt-4o`) 和在 `.env` 中设置的 API 密钥变量。
  3. 对 STT 和 TTS 服务执行同样的操作。
- **优势**: 这种方式使得切换底层 AI 服务变得极其简单，无需修改任何代码。例如，如果未来出现了更先进的 TTS 提供商，只需在设计器中替换并配置新的 TTS 节点即可。

1.5. 实时交互和可打断机制 (Barge-in)

这是实现自然对话体验的基石，也是 TEN 和 Agora 结合的强大之处。

- **技术核心**:
    1. **低延迟音频流**: Agora 的 SD-RTN™ 网络确保了从客户端到服务器的音频流延迟极低。
    2. **持续语音识别**: STT 服务（如 Deepgram）支持流式识别，可以在用户说话的同时实时生成文字流。
    3. **VAD & Turn Detection**: TEN 的 VAD 扩展会实时分析音频流，判断用户是否仍在说话。而 Turn Detection 模型则能更智能地预测用户何时可能结束发言。
    4. **TTS 打断**: AI Agent 的 TTS 在播放语音时，会持续监听来自 VAD 的信号。如果 VAD 检测到用户开始说话（即 barge-in），TEN Agent 会立即中断正在播放的 TTS，并开始处理新的用户输入。
- **流程**: AI 正在讲话（TTS播放中） -> 用户突然插话 -> 客户端通过 Agora SDK 捕捉到音频 -> TEN Agent 的 VAD 检测到上行音频流有语音活动 -> Agent 立即停止当前 TTS 的播放 -> 开始处理用户的这段新语音 -> 形成新的回应。

1.6. 与 Web 应用的集成模式

MedJourney Web 应用（前端）将作为客户端与部署在服务器上的 TEN Agent（后端）进行通信。

- **通信协议**: 强烈建议使用 **WebSocket**。这可以确保前端和后端之间的双向、实时、低延迟通信，非常适合传输信令（如对话状态）和少量数据。音频/视频流本身由 Agora SDK 处理。

- **集成架构**:
  1. **前端 (React/Vue/Angular)**:
     - 集成 **Agora Web SDK**：负责初始化、加入/离开音视频通话频道、采集本地麦克风音频流并发布到 Agora 网络。
     - 集成 **WebSocket 客户端**: 连接到 TEN Agent 的 WebSocket 服务，用于发送文本消息、接收来自 Agent 的指令和文本响应。
  2. **后端 (TEN Agent)**:
     - **Agora RTM/RTC 服务**: TEN Agent 作为另一个“用户”加入同一个 Agora 频道，通过 Agora 的服务接收前端发送的音频流。
     - **WebSocket 服务器**: 监听来自前端的连接，处理文本消息和发送系统状态。
- **API 接口 (基于 WebSocket 消息)**:
  - `client-to-agent`:
    - `start_session`: { userId: '...' }
    - `text_message`: { text: '你好' }
    - `image_upload`: { imageUrl: '...' } // (通过其他方式上传后，将URL发给Agent)
  - `agent-to-client`:
    - `agent_status`: { status: 'listening' | 'thinking' | 'speaking' }
    - `text_response`: { text: '我正在听...' }
    - `audio_control`: { action: 'play' | 'stop', audioUrl: '...' } // (TTS的URL或数据流)

---

2. Agora 实时通信 SDK

Agora 作为业界领先的实时互动云服务商，其 Web SDK 是连接前端用户和后端 AI Agent 的桥梁，为 MedJourney 提供低延迟、高质量的音视频通信基础。

2.1. Agora Web SDK 最新版本和功能特性

根据截至 2025 年中的信息，Agora Web SDK 已迭代至 `4.2x` 系列。其核心功能与本项目高度相关：

- **超低延迟音频传输**: Agora 的核心优势在于其自建的软件定义实时网络 (SD-RTN™)，能够智能规划最优传输路径，全球端到端延迟可低至 76ms，为实现自然、实时的语音对话提供了网络保障。
- **原始音频数据访问**: SDK 允许开发者直接访问原始的音频流数据（Raw Audio Stream）。这是实现 AI 功能的关键，因为它使得将用户的实时语音流发送给后端的 STT 服务成为可能。
- **AI 降噪**: 内置了先进的 AI 驱动的环境噪声抑制功能，能在客户端层面有效过滤背景噪音，提升语音识别的准确率。
- **全平台兼容性**: 提供了对主流浏览器（Chrome, Safari, Firefox, Edge）的良好支持。

2.2. 实时音视频通信的配置和实现

在前端 Web 应用中集成 Agora SDK 的流程清晰明确：

1. **引入 SDK**: 通过 npm 安装 `agora-rtc-sdk-ng` 或通过 CDN 引入。
2. **创建客户端**: `AgoraRTC.createClient({ mode: 'rtc', codec: 'vp8' })` 初始化一个 RTC 客户端实例。
3. **获取 App ID 和 Token**: 从 Agora 控制台获取项目所需的 App ID。为保障安全，在生产环境中，Token 应由后端服务器动态生成，而不是硬编码在前端。
4. **加入频道**: `client.join(appId, channelName, token, userId)` 使用 App ID、频道名、Token 和用户 ID 加入一个通话频道。后端 TEN Agent 也将以一个独立的 userId 加入同一个频道。
5. **创建和发布音轨**: `AgoraRTC.createMicrophoneAudioTrack()` 创建一个本地麦克风的音频轨道。
6. **发布轨道**: `client.publish(localAudioTrack)` 将本地的音频轨道发布到频道中，此时频道内的其他用户（包括后端的 TEN Agent）就能接收到这个音频流。
7. **订阅远端轨道**: `client.on('user-published', ...)` 监听远端用户（即 TEN Agent）发布音视频轨道的事件，并进行订阅和播放。

2.3. 与 TEN Framework 的协同工作方式

Agora 和 TEN 的协同是整个系统的核心。它们通过共享同一个 Agora “频道(Channel)” 来连接。

- **数据流**: 用户在前端通过麦克风说话 -> Agora Web SDK 捕获音频 -> 编码后通过 SD-RTN™ 网络传输 -> 后端的 TEN Agent（作为一个静默的 RTC 客户端）接收到原始音频流 -> TEN Agent 将此音频流实时转发给 STT 扩展进行处理。
- **信令流**: 对话的状态管理（如“正在思考”、“正在说话”）通过独立的 WebSocket 连接进行，不由 Agora 处理。这种音视频流和信令流分离的架构，清晰且高效。
- **核心优势**: 这种模式的最大优势在于，音频数据不需要经过我们的应用服务器中转，而是直接通过 Agora 高效的网络进行传输，极大地降低了延迟。TEN Agent 只需处理逻辑和与 AI 服务的交互，无需承载音视频流的传输压力。

2.4. App ID 的使用方法

`App ID: d83b679bc7b3406c83f63864cb74aa99` 是一个 Agora 项目的唯一标识符。这个 ID 告诉 Agora SDK，当前的应用是哪个注册项目下的。

- **作用**: 它是初始化 Agora 客户端和加入频道时必不可少的参数。所有使用相同 App ID 的客户端实例才能在 Agora 的网络中找到彼此并进行通信。
- **使用**: 在代码中，它通常作为 `client.join` 函数的第一个参数。在 MedJourney 项目中，前端 Web 应用和后端 TEN Agent 在初始化各自的 Agora 客户端时，都必须使用这同一个 App ID。
- **安全注意**: 虽然此 App ID 是公开的，但必须与动态生成的 Token 配合使用才能成功加入频道，从而保证了通信的安全性。

2.5. 低延迟语音交互的优化策略

除了依赖 Agora 的底层网络，我们还可以从应用层面进行多项优化：

- **选择合适的编解码器**: 对于纯语音场景，选择 **Opus** 编解码器。它在各种比特率下都表现出色，特别是在低带宽情况下也能保证较好的音质。
- **客户端性能优化**: 确保 Web 应用本身运行流畅，避免因浏览器渲染、计算等导致的性能瓶颈，从而影响音频采样的及时性。

- **快速的 STT/TTS 服务**: 选择响应速度快、支持流式处理的 STT 和 TTS 提供商至关重要。Deepgram 和 ElevenLabs 是业界的优秀选择。
- **智能 VAD 调优**: 对 TEN Framework 中的 VAD 参数进行微调（如静音阈值、判断延迟等），使其既能灵敏地检测到用户开口，又能避免因背景杂音导致的误判。
- **就近部署 TEN Agent**: 将 TEN Agent 服务部署在地理位置上靠近目标用户的云服务器区域，可以减少从 Agent 到第三方 AI 服务（LLM, TTS等）的網絡延遲。

---

3. RAG 向量检索技术

为了让 AI Agent 能够提供专业、准确的 Alzheimer's 相关信息，而不是仅仅依赖通用大模型 (LLM) 的知识，我们必须引入检索增强生成 (Retrieval-Augmented Generation, RAG) 技术。RAG 能让 Agent 在回答问题前，先从一个专业的知识库中检索相关信息，然后基于这些信息生成回答。

3.1. 适合医疗问诊场景的 RAG 架构

一个为 MedJourney 定制的 RAG 架构应包含以下流程：

1. **知识库构建 (线下)**: 收集权威的 Alzheimer's 相关医疗文本 -> 清洗和预处理 -> 使用文本分割算法切分成小块 (Chunks) -> 使用文本嵌入模型 (Embedding Model) 将文本块向量化 -> 存入向量数据库。
2. **查询处理 (线上)**: 用户提出问题 (Query)。
3. **上下文增强**: 在进行向量检索前，将原始问题与对话历史、用户画像（如病程阶段）等上下文信息结合，生成一个更具体、更丰富的“内部查询”。
4. **向量检索**: 将增强后的查询进行向量化，在向量数据库中进行相似度搜索，召回最相关的 Top-K 个知识文本块。
5. **增强提示词生成**: 将检索到的知识文本块、原始问题和对话历史一起，构建成一个内容丰富的提示词 (Prompt)。
6. **LLM 生成回答**: 将该提示词发送给 LLM，LLM 基于给定的专业知识生成最终回答，有效避免“幻觉”并提高准确性。

3.2. 向量数据库选择 (Pinecone vs. Chroma vs. FAISS)

对于 MedJourney MVP，我们需要一个易于集成、稳定可靠且能平滑扩展的向量数据库。

特性	Pinecone	Chroma DB	FAISS
类型	完全托管的云服务 (SaaS)	开源，提供自托管和托管选项	开源库，非数据库
易用性	**极高**。提供简单的 API，无需关心底层硬件和算法。	**高**。专为 RAG 应用设计，API 友好，易于上手。	**中等**。需要自行管理索引和服务器，集成成本较高。
性能/扩展性	**非常高**。专为大规模、低延迟查询设计。	**中等**。适合中小型项目，大规模部署需要更多运维工作。	**极高**。性能优异，但扩展性需开发者自行实现。
生态/集成	**成熟**。与 LangChain, LlamaIndex 等框架深度集成。	**良好**。与 LangChain 等有良好集成，社区活跃。	作为底层库，被许多其他系统集成。
成本	按使用量付费。初期可能有免费套餐。	自托管免费，托管版付费。	开源免费，但有服务器和运维成本。

选型建议:

- **MVP 阶段 (推荐): Pinecone**
  - **理由**: 对于 MVP 阶段，快速开发和验证是首要任务。Pinecone 作为完全托管的服务，可以让我们完全不用担心数据库的运维、扩展和性能调优问题，通过简单的 API 调用即可完成集成。其成熟的生态和与 LangChain.js 的无缝衔接能极大地加速开发进程。虽然有成本，但其节省下的人力和时间成本在项目初期更为宝贵。
- **长期/大规模部署: 考虑 Chroma (自托管) 或其他方案**
  - **理由**: 当应用成熟，用户量巨大时，Pinecone 的成本可能会显著增加。届时，可以考虑迁移到自托管的 ChromaDB 或其他更具成本效益的方案，但这需要投入专门的运维资源。

3.3. Alzheimer's 专业问题库的构建策略

知识库的质量直接决定了 RAG 的上限。

1. **数据源收集**:
    - **权威医疗指南**: Alzheimer's Association, National Institute on Aging (NIA), WHO 等机构发布的官方指南和事实清单。
    - **医学百科**: WebMD, Mayo Clinic 等网站上关于 Alzheimer's 的科普文章。
    - **学术论文**: 从 PubMed, Google Scholar 等平台筛选相关的综述性文章和研究论文（注意提取摘要和结论部分）。
    - **临床问答对 (FAQ)**: 整理面向患者和家属的常见问题与标准答案。
2. **数据清洗与预处理**:
    - 将 PDF、HTML 等格式统一转换为纯文本。
    - 去除广告、导航栏、页眉页脚等无关信息。
    - 处理图表和图片，用文字描述其核心内容。
3. **文本分块 (Chunking)**:
    - 这是 RAG 中最关键的步骤之一。不能简单地按固定长度切分。
    - 应采用 **递归字符文本分割器 (RecursiveCharacterTextSplitter)**，它会尝试按段落、句子、单词等语义边界进行切分，保证了每个文本块的语义完整性。
    - 块大小 (Chunk Size) 和重叠 (Overlap) 是需要实验调优的关键参数。建议初始设置为 `chunk_size=1000`，`chunk_overlap=200`。
4. **向量化与存储**:
    - **选择嵌入模型**: 使用高质量的文本嵌入模型至关重要。可以选择 OpenAI 的 `text-embedding-3-small` 或其他在 MTEB (Massive Text Embedding Benchmark) 排行榜上表现优异的模型。
    - **批处理入库**: 将所有文本块通过嵌入模型转换为向量，并批量存储到选择的向量数据库（如 Pinecone）中。

3.4. JavaScript/Node.js 环境下的实现方案

LangChain.js 是在 Node.js 环境下实现 RAG 的不二之选，它将整个流程抽象得非常清晰。

核心依赖: `@langchain/core`, `@langchain/openai` (或对应的 LLM/Embedding 提供商), `@langchain/pinecone` (或对应的向量数据库), `langchain`。

实现步骤:

1. **加载器 (Loaders)**: 使用 LangChain 的 `CheerioWebBaseLoader` 加载网页内容，或 `PDFLoader` 加载 PDF 文件。
2. **分割器 (Splitters)**: 使用 `RecursiveCharacterTextSplitter` 对加载的文档进行分块。
3. **向量存储 (Vector Stores)**: 初始化 `PineconeStore`，并使用 `OpenAIEmbeddings` 模型将分割后的文档块进行向量化并存入 Pinecone。
4. **检索器 (Retrievers)**: 从 `PineconeStore` 创建一个检索器实例 `retriever = vectorStore.asRetriever()`。它可以根据查询找到相关文档。
5. **构建链 (Chain)**: 这是 LangChain 的精髓。
    - **创建检索链 (Retrieval Chain)**: 使用 `createRetrievalChain` 创建一个链，它接收用户问题，从检索器获取相关文档，并将问题和文档传递给下一步。
    - **定义提示词模板**: 创建一个提示词模板 (PromptTemplate)，指导 LLM 如何利用上下文信息回答问题。模板中应包含 `{context}` 和 `{input}` 两个占位符。
    - **链接组件**: 将提示词模板、LLM 模型和输出解析器链接在一起，形成一个完整的 RAG 链。

上下文感知的动态提问生成:

这属于更高级的 RAG 技术。在调用检索链之前，可以先创建一个“问题重写链” (Query Rewriting Chain)。

- **输入**: 用户的当前问题 + 对话历史。
- **逻辑**: 使用一个独立的 LLM 调用，其提示词为：“根据下面的对话历史和新问题，生成一个独立的、无需上下文就能理解的查询，用于在知识库中搜索。”
- **输出**: 一个经过重构的、更适合向量检索的问题。

- **示例**:
    - 历史: “我妈妈最近记忆力不好。”
    - 新问题: “我该怎么办？”
    - 重写后的查询: “对于记忆力不好的阿尔茨海默病患者，家人应该如何应对和护理？”

这个重写后的查询将大大提升后续向量检索的准确性。LangChain.js 通过其表达式语言（LCEL）可以轻松地将这个“问题重写链”和主“RAG链”串联起来。

---

4. Alzheimer's 问诊场景技术需求

技术最终是为用户服务的。对于 MedJourney 的特殊用户群体——Alzheimer's 患者及其家属，技术实现必须以人为本，充分考虑其生理和心理上的特殊需求。

4.1. 老年患者友好的交互设计原则

针对认知能力下降的老年用户，Web 应用的界面和交互 (UI/UX) 必须遵循以下原则：

- **简洁至上**: 界面布局必须清晰、简单，避免信息过载。每个页面只承载一个核心功能。使用大号字体、高对比度的颜色方案（如白底黑字）。
- **一致性**: 交互元素（按钮、图标）的设计和位置在整个应用中必须保持高度一致，以降低用户的学习成本和记忆负担。
- **明确的视觉焦点**: 使用大尺寸、色彩鲜明的按钮来引导用户的注意力，避免使用需要精确点击的小链接或图标。
- **减少输入**: 尽可能使用选择（按钮、卡片）而非文本输入。对于必须输入的内容，应提供语音输入作为替代。
- **即时反馈**: 用户的每一次操作（点击、语音指令）都应立即给予清晰的视觉或听觉反馈（如按钮按下效果、提示音），让他们确信系统正在响应。
- **容错性设计**: 用户可能会误操作。系统应提供简单明了的“撤销”或“返回”功能，避免用户因犯错而感到沮丧。
- **语音优先**: 将语音交互作为核心，允许用户通过对话完成主要任务，最大程度减少手动操作。

4.2. 情感陪伴和心理支持的技术实现

AI Agent 不应仅仅是一个问答机器，更要成为一个有“温度”的陪伴者。

- **情绪识别**:
  - **技术方案**: 可以在 TEN Framework 中加入一个“情感分析”扩展。该扩展可以在 STT 将语音转为文本后，对文本内容进行情感倾向分析（正面、负面、中性）。更进一步，可以分析语音语调（如音高、语速），但这需要更专业的声学模型。
  - **实现**: 调用成熟的情感分析 API（如 OpenAI API 本身在一定程度上也能理解情感，或使用专门的服务），将分析结果（如 `emotion: 'sadness'`）作为对话状态的一部分。
- **共情回应**:
  - **技术方案**: 在 RAG 的提示词工程中，明确指示 LLM 根据识别到的用户情绪，生成富有同理心和支持性的回应。
  - **实现**: **提示词示例**: `"背景：用户当前的情绪是“悲伤”。任务：请生成一句安慰和鼓励的话，并温和地询问原因。请使用关怀、亲切的语气。"`
- **个性化记忆**:
  - **技术方案**: 为每个用户建立一个独立的档案（可以存在简单的 NoSQL 数据库如 MongoDB 中），记录下对话中提到的关键信息（如家人姓名、喜欢的歌曲、重要的往事）。
  - **实现**: 在对话过程中，Agent 可以通过一个函数调用 (Function Calling) 将识别到的关键实体（如 `entity: 'granddaughter'`, `name: 'Lucy'`）存入该用户的档案。在后续对话中，Agent 可以主动提及这些信息（“今天想听听您孙女 Lucy 喜欢的歌吗？”），营造长期、专属的陪伴感。

4.3. 健康数据收集和分析算法

应用可以在用户无感知的情况下，收集有价值的健康相关数据，但必须在用户知情同意的前提下进行。

- **数据收集**:
  - **对话主题**: 记录和分类用户经常谈论的话题，分析其关注点的变化。
  - **情绪波动**: 长期记录用户的情绪变化趋势，识别异常的、持续的负面情绪。
  - **语言复杂度**: 分析用户语言使用的变化，如词汇量、句子长度、语速等，这些都可能是认知变化的指标。
  - **反应时间**: 记录用户回答问题的平均响应时长。
- **分析算法**:
  - **技术方案**: 后端可以设置一个定期的批处理任务（如每晚执行）。该任务读取当天收集的原始数据，进行聚合分析，计算出各项指标的统计数据（如正面/负面情绪比例、平均词汇量等）。
  - **实现**: 使用 Node.js 的定时任务库（如 `node-cron`）触发分析脚本。分析结果可以结构化地存储在数据库中，用于后续的可视化展示。

4.4. 家属简报和医生报告的数据可视化

将收集和分析的数据以直观、易懂的方式呈现给家属和医生，是实现应用价值闭环的关键。

- **技术选型**:
  - **Chart.js (推荐)**: 轻量级、易于上手，提供了多种常用的图表类型（折线图、条形图、饼图），非常适合 MVP 阶段快速构建数据看板。它与 React/Vue 等前端框架有良好的集成库。
  - **D3.js**: 功能极其强大，可以实现任何你想要的定制化、交互式图表。但学习曲线陡峭，开发成本高，更适合在项目后期有复杂可视化需求时引入。

- **可视化方案**:
  - **家属看板**: 应侧重于情感和活动的宏观趋势。例如：
    - 用 **饼图** 展示近一周的情绪分布。
    - 用 **折线图** 展示认知评估得分的变化趋势。
    - 用 **词云** 展示最近的热门谈话主题。
  - **医生报告**: 应更专业、更数据化。例如：
    - 用 **多条折线图** 对比语言复杂度、反应时间等多个认知指标的变化。
    - 提供详细的对话日志，并标注出情绪异常或认知困难的关键节点。

4.5. 记忆和认知功能评估的技术方案

将标准的认知评估工具游戏化、对话化，可以降低患者的抵触情绪，并实现持续、无压力的评估。

- **技术核心**: **对话式 MMSE (Mini-Mental State Examination)**
  - **MMSE 简介**: MMSE 是一个广泛使用的认知功能速查工具，包含定向力、记忆力、注意力、计算能力、语言能力等多个维度的评估。
  - **数字化实现**: 我们可以将 MMSE 的题目改编成 AI Agent 的提问。Agent 在一个特定的“评估模式”下，通过对话引导用户完成测试。
    - **定向力**: “早上好！您知道今天大概是几号，星期几吗？”
    - **记忆力**: “我告诉您三样东西：苹果、桌子、硬币。请您记住。过一会儿我会再问您。”
    - **注意力/计算**: “我们来做个小游戏吧。从100开始，每次减掉7，您能告诉我第一个答案是多少吗？”
    - **语言能力**: Agent 可以展示一张图片（如手表），然后问：“您能告诉我这是什么吗？”
- **评分与记录**:
  - **技术方案**: Agent 根据用户的回答，通过 LLM 的理解能力或预设的规则进行自动评分。例如，对于记忆力问题，LLM 可以判断用户的回答是否包含了之前提到的三个词。
  - **实现**: 将评分结果与测试时间一同记录在数据库中，用于生成前述的认知变化趋势图。

- **优势**: 这种方式将枯燥的测试融入到日常的陪伴和对话中，数据收集的频率可以更高，结果也更能反映患者的真实状态。

---

5. 集成架构图和实现步骤

5.1. 系统集成架构图

下图清晰地展示了 MedJourney MVP 的各个技术组件是如何协同工作的。

MedJourney MVP 技术架构图

5.2. 核心实现步骤

1. **环境搭建**: 部署一个 Node.js v18+ 和 Docker 环境。克隆 TEN Framework 仓库，并根据文档配置好 `.env` 文件，填入所有必要的 API 密钥（Agora, OpenAI, Deepgram, ElevenLabs, Pinecone）。
2. **后端 Agent 初始化**: 运行 `docker compose up -d` 启动 TEN Agent 和 TMAN Designer。访问 TMAN Designer，搭建一个基础的对话流：`Agora In -> STT -> LLM -> TTS -> Agora Out`，并确保其能正常工作。
3. **前端应用初始化**: 创建一个 React 或 Vue 前端项目。集成 Agora Web SDK，实现加入/离开频道、采集和发布本地麦克风音频流、订阅和播放远端音频流的功能。
4. **RAG 知识库构建**: 编写脚本，使用 LangChain.js 的 Loaders 和 Splitters 处理收集到的 Alzheimer's 文档，并通过 OpenAIEmbeddings 将其向量化，存入 Pinecone 数据库。
5. **后端 RAG 集成**: 在 TMAN Designer 中，将基础对话流修改为 RAG 流程。在 STT 和 LLM 之间插入一个 RAG 节点。该节点负责调用 LangChain.js 的 RAG 链，用知识库检索到的内容来增强 LLM 的提示词。
6. **前端后端连接**: 在前端应用中，实现 WebSocket 客户端，用于连接 TEN Agent 的 WebSocket 服务。实现发送文本消息、接收 Agent 状态和响应的逻辑。
7. **数据看板开发**: 在前端应用中开辟一个新的路由页面作为家属/医生看板。使用 Chart.js，根据从后端获取的分析数据，渲染出情绪分布、认知评估趋势等图表。

8. **部署**: 将定制好的 TEN Agent 打包成 Docker 镜像，部署到云服务器（如 AWS EC2 或 GCP a）。将前端应用部署到静态网站托管服务（如 Vercel, Netlify）。

---

6. 潜在的技术挑战和解决方案

- **挑战 1: 实时对话的延迟**
  - **描述**: 即便各组件都很快，从用户说话到听到 AI 回应的全链路延迟仍可能过高，影响自然感。
  - **解决方案**:
    1. **选择最快的服务**: 严格测试并选择延迟最低的 STT/LLM/TTS 服务。
    2. **流式处理**: 确保从 STT 到 LLM 再到 TTS 全程采用流式处理，即不等一整句话说完就开始处理和响应。
    3. **就近部署**: 将 TEN Agent 部署在离用户最近的云数据中心。
- **挑战 2: RAG 知识库的质量和更新**
  - **描述**: 知识库内容可能不全面或过时，导致回答不准确。分块策略不佳也可能影响检索效果。
  - **解决方案**:
    1. **持续迭代**: 建立一个定期的知识库更新机制，定期爬取和处理新的医疗指南。
    2. **分块策略调优**: 投入时间实验不同的分块大小和重叠参数，找到最适合医疗文本的组合。
    3. **混合检索**: 未来可引入关键词检索与向量检索结合的混合模式，提高召回率。

- **挑战 3: 对老年用户方言、口音和语速的适应性**
    - **描述**: STT 模型可能难以准确识别老年用户的非标准普通话、缓慢的语速或模糊的发音。
    - **解决方案**:
        1. **选择强大的 STT**: Deepgram 等领先的 STT 服务在口音适应性上通常有更好的表现。
        2. **微调模型 (长期)**: 如果有足够的标注数据，未来可以考虑对 STT 模型进行微调，以更好地适应目标用户群体。
        3. **界面引导**: 在界面上给予用户明确的提示，鼓励他们尽量清晰、大声地说话。

---

7. 开发时间评估和优先级建议

假设投入 2-3 名熟练的全栈开发者，MVP 版本的开发周期预估如下：

阶段	核心任务	预估时间	优先级
第一周：技术验证 (PoC)	- 搭建 TEN Agent 基础环境		
- 实现 Agora 音视频基础通信			
- 验证核心对话流 (STT->LLM->TTS)	1 周		**最高**
第二至三周：核心功能开发	- 构建 RAG 知识库 (v1.0)		
- 在 Agent 中集成 RAG 链			
- 开发基础的前端交互界面	2 周		**最高**
第四周：核心体验优化	- 实现并调优语音打断机制		
- 设计并实现老年友好的 UI/UX (v1.0)	1 周		**高**
第五周：数据与支持功能	- 开发情感识别与共情回应逻辑		
- 开发数据收集与分析后台任务			
- 开发家属/医生数据看板 (v1.0)	1 周		**中**
第六周：集成与测试	- 开发对话式认知评估功能 (MMSE)		
- 进行端到端的集成测试和 Bug 修复			
- 准备部署	1 周		**中**

总计预估：约 6 周

优先级建议: MVP 的核心是验证 **“通过 RAG 增强的、可实时打断的 AI 语音对话”** 这一核心模式是否可行并能被用户接受。因此，应优先保证 **TEN + Agora + RAG** 这条主干链路的稳定和流畅。情感分析、数据看板、认知评估等功能虽然重要，但可以在核心体验打磨好之后再逐步完善。

8. 关键代码示例与安全合规

为了使本研究报告更具实践指导意义，特补充关键代码示例和安全合规两大模块。

8.1. 关键代码示例 (Boilerplate Code)

以下代码片段旨在为开发者提供一个快速启动的模板。

8.1.1. 前端: Agora Web SDK 初始化与原始音频流获取 (JavaScript)

```
// 安装: npm install agora-rtc-sdk-ng
import AgoraRTC from "agora-rtc-sdk-ng";

const client = AgoraRTC.createClient({ mode: "rtc", codec:
"opus" });

// 监听远端用户发布事件，用于接收 Agent 的音频
client.on("user-published", async (user, mediaType) => {
  await client.subscribe(user, mediaType);
  if (mediaType === "audio") {
    const remoteAudioTrack = user.audioTrack;
    remoteAudioTrack.play();
  }
});

async function joinChannel(appId, channelName, token) {
  const uid = await client.join(appId, channelName, token, null);

  // 创建麦克风音频轨道
  const localAudioTrack = await
AgoraRTC.createMicrophoneAudioTrack();

  // *** 核心：获取原始音频流 ***
  // 通过 getMediaStreamTrack() 可以获取原始的 MediaStreamTrack
  // 这个 streamTrack 可以被用于 Web Audio API 进行进一步处理，
  // 或通过 WebSocket 发送到后端（但推荐由后端 Agent 直接在 Agora 频道内接
收）
  const mediaStreamTrack = localAudioTrack.getMediaStreamTrack();
  console.log("Raw MediaStreamTrack for STT processing:",
mediaStreamTrack);

  // 将本地音频发布到频道，供后端 TEN Agent 接收
  await client.publish([localAudioTrack]);

  console.log(`Successfully joined channel <span class="math-
```

```
inline" style="display: inline;"><math xmlns="http://www.w3.org/
1998/Math/MathML" display="inline"><mrow><mrow><mi>c</mi><mi>h</
mi><mi>a</mi><mi>n</mi><mi>n</mi><mi>e</mi><mi>l</mi><mi>N</
mi><mi>a</mi><mi>m</mi><mi>e</mi></mrow><mi>w</mi><mi>i</mi><mi>t</
mi><mi>h</mi><mi>U</mi><mi>I</mi><mi>D</mi></mrow></math></
span>{uid}`);
}

// 使用示例
// const AGORA_APP_ID = "d83b679bc7b3406c83f63864cb74aa99";
// const CHANNEL_NAME = "medjourney_session_1";
// const TOKEN = "<Generated from your server>";
// joinChannel(AGORA_APP_ID, CHANNEL_NAME, TOKEN);
```

8.1.2. 后端: LangChain.js 最小化 RAG 链 (Node.js)

```
// 安装: npm install langchain @langchain/openai @langchain/
pinecone pinecone-client
import { ChatOpenAI, OpenAIEmbeddings } from "@langchain/openai";
import { PineconeStore } from "@langchain/pinecone";
import { Pinecone } from "@pinecone-database/pinecone";
import { ChatPromptTemplate } from "@langchain/core/prompts";
import { createStuffDocumentsChain } from "langchain/chains/
combine_documents";
import { createRetrievalChain } from "langchain/chains/retrieval";

// 1. 初始化服务
const pinecone = new Pinecone({ apiKey:
process.env.PINECONE_API_KEY });
const pineconeIndex = pinecone.index("medjourney-knowledge-base");
const embeddings = new OpenAIEmbeddings();
const llm = new ChatOpenAI({ modelName: "gpt-4o" });

// 2. 初始化向量数据库
const vectorStore = await
PineconeStore.fromExistingIndex(embeddings, {
  pineconeIndex,
});
const retriever = vectorStore.asRetriever();

// 3. 创建提示词模板
const prompt = ChatPromptTemplate.fromTemplate(`
  你是一个专业的医疗助手。请根据下面提供的上下文信息来回答用户的问题。
  如果上下文中没有相关信息，请礼貌地说明你无法回答，不要编造答案。

  上下文:
  {context}

  问题: {input}
`);
```

```
// 4. 创建文档处理链 (将检索到的文档组合进提示词)
const combineDocsChain = await createStuffDocumentsChain({ llm,
prompt });

// 5. 创建检索链 (核心 RAG 链)
const retrievalChain = await createRetrievalChain({
  retriever,
  combineDocsChain,
});

// 6. 调用链进行问答
async function ask(question) {
  const result = await retrievalChain.invoke({
    input: question,
  });
  console.log(result.answer);
  return result.answer;
}

// 使用示例
// ask("阿尔茨海默病早期有哪些症状？");
```

8.2. 安全与合规性强化 (Security & Compliance)

处理个人健康信息 (PHI) 是 MedJourney 项目的最高优先级，必须在架构设计之初就融入安全与合规性考量。

8.2.1. 数据加密

- **端到端加密 (E2EE)**: Agora SDK 原生支持端到端加密。必须在客户端和 TEN Agent 初始化时启用此功能，确保用户与 AI Agent 之间的音视频流在传输过程中是完全加密的，即使是 Agora 服务器也无法解密。
- **传输中加密 (In-Transit)**: 所有 API 调用（与第三方 LLM, STT, TTS 服务的通信）和 WebSocket 连接都必须使用 TLS 1.2 或更高版本进行强制加密。

- **静态加密 (At-Rest)**: 存储用户画像、对话历史和分析结果的数据库（如 MongoDB）必须启用静态加密功能，确保数据在磁盘上是加密存储的。Pinecone 等托管服务通常默认提供静态加密。

8.2.2. 隐私合规 (HIPAA & GDPR)

虽然 MVP 阶段可能不直接面向市场，但必须按照合规标准构建，以备未来之需。

- **HIPAA (美国健康保险流通与责任法案)**:
  - **业务合作协议 (BAA)**: 必须与所有处理 PHI 的云服务提供商（包括 Agora, OpenAI, Pinecone, 以及云托管平台 AWS/GCP）签订 BAA 协议。这是一个法律合同，规定了服务商保护 PHI 的责任。
  - **最小必要原则**: 系统在收集、使用和披露 PHI 时，必须遵循“最小必要”原则。例如，仅收集与病情评估和陪伴相关的必要信息。
  - **访问控制**: 必须建立严格的基于角色的访问控制（RBAC），只有授权的开发者或医疗人员才能访问敏感数据。
- **GDPR (欧盟通用数据保护条例)**:
  - **用户同意**: 必须在用户注册时，通过清晰、明确的语言获取用户对于数据收集和处理的同意书。
  - **数据主体权利**: 系统必须有能力响应用户的请求，包括访问、更正、删除（“被遗忘权”）其个人数据的权利。
  - **数据处理协议 (DPA)**: 与 BAA 类似，需要与第三方服务商签订 DPA。

8.2.3. 数据匿名化与去标识化

在将数据用于分析或模型训练之前，应尽可能进行去标识化处理。

- **方案**: 可以在 TEN Agent 中增加一个“PHI Scrubber”扩展。在将任何对话文本发送给非合规的 LLM 或用于日志记录之前，该扩展使用正则表达式或命名实体识别 (NER) 技术，识别并替换掉姓名、地址、电话号码等敏感信息。
- **示例**: `"用户说：我叫张三，住在xx路"` -> `"用户说：我叫[PERSON_NAME]，住在[ADDRESS]"`。

8.2.4. 安全最佳实践

- **API 密钥管理**: 严禁在代码中硬编码任何 API 密钥。所有密钥都必须存储在安全的环境变量或专用的密钥管理服务中（如 AWS Secrets Manager, HashiCorp Vault）。
- **依赖项安全**: 定期使用 `npm audit` 或 Snyk 等工具扫描项目依赖，及时修复已知的安全漏洞。
- **日志记录**: 日志中绝对不能包含任何原始的 PHI。所有敏感信息在记入日志前都必须经过上述的“去标识化”处理。

(原第7节现调整为第9节)

9. 开发时间评估和优先级建议

...(内容不变)...

9. 开发时间评估和优先级建议

假设投入 2-3 名熟练的全栈开发者，MVP 版本的开发周期预估如下：

阶段	核心任务	预估时间	优先级
第一周：技术验证 (PoC)	- 搭建 TEN Agent 基础环境		
- 实现 Agora 音视频基础通信			
- 验证核心对话流 (STT->LLM->TTS)	1 周	**最高**	
第二至三周：核心功能开发	- 构建 RAG 知识库 (v1.0)		
- 在 Agent 中集成 RAG 链			
- 开发基础的前端交互界面	2 周	**最高**	
第四周：核心体验优化	- 实现并调优语音打断机制		
- 设计并实现老年友好的 UI/UX (v1.0)	1 周	**高**	
第五周：数据与支持功能	- 开发情感识别与共情回应逻辑		
- 开发数据收集与分析后台任务			
- 开发家属/医生数据看板 (v1.0)	1 周	**中**	
第六周：集成与测试	- 开发对话式认知评估功能 (MMSE)		
- 进行端到端的集成测试和 Bug 修复			
- 准备部署	1 周	**中**	
第七周：安全加固	- 实施端到端加密和数据库加密		
- 开发PHI去标识化模块			
- 梳理并检查第三方服务的BAA/DPA协议	1 周	**高**	

总计预估：约 7 周

优先级建议: MVP 的核心是验证 **“通过 RAG 增强的、可实时打断的 AI 语音对话”** 这一核心模式是否可行并能被用户接受。因此，应优先保证 **TEN + Agora + RAG** 这条主干链路的稳定和流畅。情感分析、数据看板、认知评估等功能虽然重要，但可以在核心体验打磨好之后再逐步完善。**安全与合规性应与核心功能同步进行**，而不是事后添加。