

语音驱动的人机交互入口 —— 智能电脑语音控制应用

姓名：张迪 应聘职位：上海-产品经理

摘要

本项目《语音驱动的人机交互入口（Voice Computer Assistant）》以七牛智能语音 API 为核心，构建了一个可直接执行系统动作的语音助手 MVP。用户只需一句自然语言指令（如“打开微信”“调高亮度”“截图”“开启专注模式”），系统即可自动完成从语音识别→意图理解→本地执行→语音反馈的全闭环。

与传统语音助手不同，本项目在本地侧实现了动作执行引擎（Executor），能在 macOS 上直接操作系统功能（音量、亮度、应用控制等），同时具备跨平台兼容性（Windows 可运行核心链路），实现“真执行，而非对话模拟”。

1 产品背景与核心用户场景

随着多任务办公越来越普遍，传统的人机交互方式（键盘+鼠标）在复杂任务、多窗口切换和高频操作中效率逐渐受限。同时，LLM 的出现让语音具备了“理解上下文 + 执行复杂指令”的能力，为语音成为下一代人机交互入口提供了现实可能。

我认为语音并非要替代键鼠，而是作为更自然的“第二入口”，尤其在多任务并行、双手被占用或需要快速执行系统操作时更具效率优势。

在高频多任务办公场景中，键鼠组合已成为效率瓶颈：当用户同时处理 IM 消息、查资料、写文档、切应用、查日程时，频繁的界面跳转与重复性操作会打断思路、降低专注度。同时，语音交互技术在智能设备中已较为普及，但现有语音助手普遍存在三类问题：

现状问题	影响
可控性差，只能做“简单问答”	不能满足办公需求
指令理解浅、不连贯	无法承接多步任务
响应链路不透明、执行有限	停留在“玩具级系统”

因此，办公场景需要的并不是“聊天式语音助手”，而是一个可控、可执行、能承担办公任务链路的语音入口。

我将用户按 ToB 视角分层如下：

用户层级	需求特征	语音价值
多任务办公人群（核心）	需要在不中断当前动作下完成系统操作	语音缩短操作路径，提高专注效率
效率党/重度电脑用户（次核心）	有清晰目标、追求最短操作链	以语音提升流程速度
特殊人群/双手行动不便者（衍生价值）	传统交互成本高	降低使用门槛，提高无障碍体验

典型办公用户故事：
用户正在写 PPT，手在键盘上输入内容，此时需要快速调出日历查看明天会议时间、并打开飞书对应群文件。若使用键鼠，需要切出窗口→查日历→切回→打开 IM→搜索→点击进入。

语音路径：一句“打开飞书并搜索项目日报群文件，同时打开日历”→系统完成→用户不被打断。

这个故事直接说明：语音=降低操作成本+保持专注，这是办公场景的真实价值。

还有在高频办公与多任务环境中，鼠标与键盘操作的重复性高、上下文切换频繁。

例如：
音量调节、亮度设置、截图等系统操作需要多次点击；
打开微信、文档、浏览器等应用时被迫中断当前思路；
无障碍用户、录音转写、会议记录等场景对语音交互有强需求。

目标：通过一句话语音命令实现系统级操作，让语音成为新的操作入口。
这不仅是效率工具，更是让电脑理解人类自然语言的第一步。

2 目标用户与价值假设

目标用户：

用户类型	核心需求	价值体现
高效办公人群	降低操作切换成本	一句语音完成系统控制
创意生产者	双手操作受限但需频繁切换	提高多任务效率
无障碍用户	用语音代替键鼠交互	实现真正的可访问性
技术从业者	探索语音与系统 API 的结合	可复用、可扩展的语音入口架构

基于上述角色，我的产品价值假设是：

1. 语音是新的桌面入口：通过一句话执行复合指令（宏指令），打破 GUI 界限；
2. 减少操作摩擦：无需切换窗口、查找菜单；
3. 即时反馈：结合 TTS 播报，实现自然回馈；
4. 开放性架构：支持未来扩展更多系统动作或自定义模式（如会议模式、影像模式）。

一句总结用户驱动力：
语音不是替代输入法，而是补全“操作意图”的更快路径。

3 核心功能设计与优先级规划

系统支持语音触发的完整电脑操作链路，并扩展了“宏能力（Macro Orchestration）”，可实现多步场景化操作。

优先级	MVP 能力范围（本次实现）	场景目的
Must	唤醒、语音识别 (ASR)、打开/切换应用、意图解析 (LLM)、系统功能控制（截屏/音量/搜索）、播放音乐、TTS 反馈	最小可用路径（执行闭环）
Should	多轮语音对话执行任务、文件搜索、网页信息查询	自然度提升
Could	文本生成、邮件草稿整理、摘要记录	生产力拓展
Won't	跨设备联动、任务自动化编排、语音插件生态	Pro 阶段再做

3.1 基础能力清单 (MVP 核心)

类型	示例语音	实际执行
打开类	“打开微信 / Safari”	调用 <code>open_app()</code> 打开指定应用
系统类	“调大音量 / 静音 / 截屏”	调用 <code>system_control()</code> 实际修改系统状态
搜索类	“搜索周杰伦演唱会”	自动打开浏览器执行搜索
亮度类	“调亮一点 / 亮度设置为 60”	调用亮度控制（macOS 可用）
组合类	“开启专注模式 / 演示模式”	执行多步宏 (Macro) 指令序列

3.2 亮点能力：宏编排 (Macro Orchestration)

背景：

普通语音助手只能执行“单步命令”。

而本项目的 LLM 能生成结构化 JSON，自动组合多步动作，实现“一个自然语言 → 多动作协同”的智能宏。

3.2.1 宏示例一：专注模式

用户体验描述：

说：“开启专注模式”。

系统回复：“我已为你开启专注模式。”

同时电脑自动调低音量、打开备忘录、并搜索 Lo-fi 音乐播放列表帮助提升专注度。

设计价值：

让 LLM 不仅理解单一意图，更能生成“动作序列”，实现任务级智能；用户体验上相当于“语音自动化 workflow”。

3.2.2 宏示例二：演示模式

用户体验描述：

说：“开启演示模式。”

系统回应：“演示模式已就绪。”

音量自动静音、打开浏览器并进入指定演示网页。

设计价值：

此功能展示 LLM 的任务规划与执行能力结合，证明了“非 Agent 架构下，也能实现自主多步执行的场景智能”。

4 系统架构与执行器设计

4.1 总体架构流程

用户语音 → 录音 → 七牛 ASR → LLM (意图识别/宏生成) → Executor (系统执行器) → 七牛 TTS → 播报反馈

模块划分：
ASR 层：将语音转文字；
LLM 层：解析自然语言意图并输出 JSON；
Executor 层：按意图调用本地动作；
TTS 层：生成自然语音反馈；
App 层：统一调度控制，完成闭环。

4.2 执行器 (Executor) 结构说明

本地执行器被拆为多个子模块，具备模块化与可扩展性：

模块	功能	对应文件
app_control	打开或聚焦应用（open -a / os.startfile）	executor/app_control.py
system_control	控制音量、亮度、截屏、锁屏等系统动作	executor/system_control.py
search_control	浏览器搜索或访问网页	executor/search_control.py
macro_orchestrator	解析并执行 LLM 生成的多步动作序列	集成于 app.py

模块间通过统一的 intent 字段协同，设计简洁、复用性强。

4.3 跨平台策略

功能	macOS	Windows
音量控制	AppleScript 实现	NirCmd.exe 调用
截屏	screencapture	PowerShell
应用打开	open -a	os.startfile
宏执行	完整支持	降级执行
TTS/ASR/LLM	七牛云	七牛云

工程策略：
macOS 实现全功能链；
Windows 实现核心链路（ASR + LLM + TTS + 系统动作音量/截图）。
如缺 nircmd.exe 时，系统仍能正常运行“识别+播报”闭环。

4.4 模块规格说明

本系统采用模块化设计，每个模块具备清晰的输入、输出与职责边界，以保证可维护性与可扩展性。系统核心模块规格如下：

模块	职责说明	输入	输出	被 谁 调用
ASR Client	将录音上传并完成语音转写	WAV URL	Text	app.py
LLM Client	对文本进行意图解析与宏规划	Text	Intent JSON	app.py
Executor – App Control	打开/切换应用	Intent JSON	执行结果	app.py
Executor – System Control	控制音量/亮度/截屏等系统能力	Intent JSON	执行结果	app.py
Executor – Search Control	搜索或打开 URL	Intent JSON	执行结果	app.py
TTS Client	将反馈文本合成为音频	Text	WAV Bytes	app.py

本系统的跨平台策略遵循“核心逻辑统一，系统能力按平台适配”的原则。在 macOS 环境实现完整能力，在 Windows 环境维持 ASR→LLM→TTS→Executor 的链路闭环，并通过 NirCmd 等方式作为系统执行器适配层，从而确保系统可运行、可验证、可扩展。

本系统跨平台策略遵循“识别与意图链路统一、执行能力按平台适配”的原则。在 macOS 环境下提供完整能力（ASR→LLM→TTS→系统执行器→播报闭环），在 Windows 环境下通过 NirCmd 作为执行器适配层，维持语音控制链路 with 可展示能力一致，使系统具备：

- 可运行性 (Runability)：两端均可跑通链路验证；
- 可扩展性 (Scalability)：执行器能力可独立替换；
- 可复现性 (Reproducibility)：任意平台均能完成 ASR→LLM→TTS→动作演示。

通过这种“核心统一 + 层次适配”的方式，系统保持了稳健性，同时避免了跨平台系统调用碎片化带来的不可维护问题。

5 技术挑战与应对策略

挑战类别	解决方案
ASR 识别错误	16kHz 单声道采集· 指令短句约束 + 关键词槽位校验
LLM 输出不规范 JSON	实现_safe_json() 清洗函数, 自动替换中文标点、清除 json 标签
意图模糊或不可执行	LLM Schema 限制 (intent/slots/say, 兜底 natural response
执行失败 (动作侧)	“微信/WeChat/Weixin” 三重映射, 错误可见且可回退
延迟过长 (体验卡顿)	异步播放 + 局部流式播报, 提升交互流畅度
TTS 异常或无反馈	优先 WAV, 失败回退 MP3, 本地播放失败→文本兜底
误唤醒	指令模式 (非唤醒词模式, 明确“执行型语音”语料边界
多步执行容错	宏执行时逐步记录日志, 失败步骤不中断整体执行
跨平台差异	检测类型, 动态加载不同实现模块

核心策略一句话:
“短链路、强确定性、强反馈” → 让语音像键盘一样可信赖

6 模型选型与能力对比

模型链路由三部分组成: ASR (听) + LLM (懂) + TTS (说)

项目使用 七牛云开放平台 提供的:
ASR (语音识别) : /voice/asr
LLM (意图解析) : /chat.completions
TTS (语音合成) : /voice/tts

模块	技术实现	说明
ASR	七牛 /voice/asr	语音转文本, 中文识别精度高
LLM	七牛 OpenAI 兼容接口 (Qwen-plus)	将自然语言解析为结构化意图
TTS	七牛 /voice/tts	中文音色丰富, 支持 WAV 播放
存储	七牛 Kodo	临时音频上传, 获得公网 URL
播放	本地播放器 (simpleaudio/pydub)	实时合成播放语音反馈

对比:

能力类型	备选方案	实际采用	选型理由
ASR（语音识别）	讯飞/阿里/七牛	七牛 /voice/asr	接口简洁、识别稳定、中文优化好
LLM（语义理解）	GPT-3.5/Qwen-plus	Qwen-plus	兼容 OpenAI 接口、中文理解强、延迟低
TTS（语音合成）	Edge/七牛 TTS	七牛 /voice/tts	中文音色丰富、支持 WAV、低延迟
存储服务	OSS / COS / Kodo	七牛 Kodo	上传速度快、API 集成度高、易维护

说明:

七牛方案在接口设计与数据流整合上具备天然一致性, 避免多平台依赖导致的授权和延迟问题。

尤其在比赛限制禁止第三方 Agent 下, 此组合完全合规且性能优越。

我将 LLM 的角色定位为“语义解析+指令规划器”, 而不是“无限对话者”, 技术链路短、成功率高。

7 方案整体设计与架构概览

本系统基于七牛智能语音开放接口构建, 通过 ASR、LLM、TTS 三大模块形成端到端闭环: 听得懂、想得明白、做得出来、说得回去”。

A[麦克风输入 16k WAV] --> B[Kodo 上传, 获取公网 URL]

B --> C[Qiniu ASR, 语音识别]

C --> D[Qiniu LLM, 意图解析 / qwen-plus]

D --> E[本地执行, open_app / system / search]

E --> F[Qiniu TTS, 语音合成反馈]

F --> G[扬声器输出语音反馈]

系统核心逻辑:

用户语音 → 本地录音 → 上传至七牛 Kodo;

Kodo 公网 URL 传入 ASR 接口 /voice/asr;

LLM 解析语义结构 {intent, slots, say};

本地执行器匹配动作 (打开应用 / 控制系统 / 搜索) ;

执行结果调用 TTS /voice/tts 播报反馈;

若任一环节失败, 系统自动回退或以文本提示。

该架构充分体现了“短链路 + 高鲁棒 + 全闭环”的设计理念。

8 未来能力规划

阶段	目标	功能演进
短期	完善宏定义与命令学习	支持自定义语音宏、导出宏脚本
中期	增强对话自然度	引入上下文记忆、语音打断机制
长期	向桌面侧 AGI 助手演进	连续对话、多模态感知 (视觉 + 声音)

人机交互入口的价值不在“语音输入本身”，而在从执行单动作 → 自动化任务 → AI 接管部分工作流的升级路径上，它可以真正提升办公效率，这是长期粘性来源。

规划原则：先闭环 → 再智能 → 再生态

核心演化逻辑：从“可控”迈向“可协作”。

保持轻量、稳定，不走“虚拟人”或冗长代理路线；

最终目标是打造一个：

“语音即桌面入口”的智能操作系统交互层，让电脑从“被动工具”进化为“主动协作者”。

9 Demo 效果呈现

示例 1：打开微信

体验线：

我说：“打开微信”

系统立即响应：“好的，我马上为你打开微信。”

技术线：

ASR 输出：打开微信

LLM 输出：{"intent":"open_app","slots":{"app":"微信"},"say":"好的，我马上为你打开微信"}

Executor: open -a "WeChat"

TTS 播报：“好的，我马上为你打开微信”

成功启动 WeChat 应用并语音反馈。

“一句指令 = 五个系统动作；语音让专注不中断。”

示例 2：专注模式 (Macro)

体验线：

我说：“进入专注模式”

系统：“已为你开启专注模式。”

技术线：

见上方宏日志。系统依次调低音量→打开备忘录→播放 Lo-fi 音乐。

执行完成，全程无卡顿。

10 总结

本项目在仅使用 LLM / ASR / TTS 的约束下，实现了一个真正可运行的语音控制系统，完成了从语音输入 → 语义理解 → 系统执行 → 语音反馈的完整闭环。

与传统语音助手不同，它并非炫技性的 Demo，而是一款可在日常办公中落地使用的语音生产力工具。

项目遵循“短链路、强可控、可落地”的核心设计原则，具备以下特征：

1. 闭环真实可运行 —— 语音→理解→执行→反馈，延迟低、执行可靠；
2. 架构可扩展 —— 模块化设计 (ASR / LLM / TTS / Executor)，支持宏编排和场景化任务；
3. 智能可解释 —— LLM 自动生成结构化 JSON，执行过程透明、可追踪；
4. 跨平台可部署 —— macOS 全功能，Windows 核心链路完整，具备稳定演示能力；
5. 用户价值明确 —— 显著缩短多任务操作链路，提升专注与 workflow 效率。

它验证了语音驱动系统控制的可行性，并为未来的人机交互形态提供了一个可持续演进的原型：

从语音指令 → 自动化流程 → 个性化生产力助手。

这不仅是一次技术实验，更是语音成为新一代人机入口的实践样本。

附录

1. 支持指令类型示例

类型 示例指令

打开类 打开微信 / 打开浏览器

系统类 调大音量 / 音量设置到 30 / 截屏一下/锁屏/休眠

搜索类 搜索周杰伦演唱会

宏场景 开启专注模式 / 演示模式

2. 跨平台兼容说明

macOS: 支持全部功能（音量/亮度/截屏/宏）；

Windows: 支持 ASR + LLM + TTS + 音量控制 + 截屏；

兜底方案：即便系统层指令失败，仍可执行完整语音识别与播报流程；

nircmd.exe: 用于系统调用，无需额外依赖；

demo_cli 模式：可纯展示逻辑编排输出（适合无系统权限环境）。

3. 开发说明与独立完成声明

本项目由本人独立完成，包括需求分析、架构设计、模型调用链路构建、核心功能开发（ASR / LLM / TTS / Macro / Executor）、跨平台适配、测试与文档撰写。项目完整开发过程均有 Git commit 记录可查，符合本赛事对“完备过程可追溯性”的要求。