

AI+心理关怀——从 AI Agent 出发

完成人：

- 毛周祥 (09J24102)：项目设计，实现语音部分，集成工具整合。
- 靳赫涛 (09J24103)：合作协调，实现任务调度部分。
- 周晨曦 (JS124436)：测试环境搭建，实现主动反馈心理关怀部分。
- 朱轩境 (JS324423)：项目功能细节完善，实现数据交互和可持久化部分。

摘要

本文介绍了四种与对话式人工智能相关的核心人工智能技术，基于 LLM，ASR，TTS，KWS 四种技术提出并实现了一个对话式人工智能模型。通过 Prompt Engineering 等方式，该人工智能能够**主动分析用户心理需求**，提供**心理支持和心理关怀**。避免了现有对话式人工智能 "被动"，"缺失人性" 等缺点。该人工智能**具备自闭症关怀、辅助治疗抑郁症**等多种心理场景的实用潜力。

该人工智能模型也创新性地集成了一些工具，可供普通人群使用，能够自动管理日程，提升日常工作效率，起到**缓解焦虑**的作用。

引言

对话式人工智能的发展历史

1. **起源与早期发展**：对话式人工智能的起点可以追溯到上世纪60年代。1966年，麻省理工学院的Joseph Weizenbaum开发了ELIZA，这是第一个被引用的聊天机器人，旨在模仿心理治疗师的语言模式，可以与人类用户进行简单的对话。随后，如PARRY等自然语言程序相继问世，这些系统虽然相对简单，但已经具备了初步的对话能力。
2. **统计式自然语言处理**：随着计算机科学的进步，统计自然语言处理（Statistical NLP）技术的兴起为对话式人工智能带来了新的生机。这一时期，系统开始利用大量语料库进行统计学习，通过概率模型来预测和生成自然语言文本。
3. **深度学习模型的崛起**：进入21世纪，深度学习技术的崛起使得对话式AI迈入了一个新的阶段。智能语音助手如Apple的Siri、Google的Google Assistant等开始兴起，能够识别语音指令并提供有用的信息。
4. **大模型时代的对话系统**：以 GPT 系列为代表的预训练大模型的成熟为对话式人工智能注入了新的活力。与训练模型能够处理复杂的自然语言，生成流畅连贯的文本内容，对话式人工智能达到一个前所未有的高度。

先前工作的不足

工具整合不完善

现有的对话式人工智能未能整合有效的工具，大部分任务仍然依靠大模型本身的能力完成。大模型本身的运行需要大量计算资源的支撑，直接使用大模型完成任务将导致极大的计算资源浪费。同时由于大模型能力的局限性，这些特定任务的完成效果会较差甚至无法被完成。

缺少 "人性", 被动回答

先前包括 ChatGPT 在内的对话式人工智能系统往往采取被动回答的方式，它们运行的凭据是用户输入结束，在心理学上存在很强的割裂感。AI 缺乏主动引导对话或提供额外信息的能力。

这种被动性极大限制了AI在提供个性化服务和增强用户体验方面的潜力，同时显得 AI 缺少 "人性"。

人机交互方式单一

目前相对成熟的人机交互方式有语音交互和文字交互。现有的支持语音交互的 "小度" 等 AI 没有大模型的支持，各方面能力显著较弱。而现有的大模型对话式 AI 系统普遍采用文字交互系统，人机交互方式过于单一，具体，有以下几个局限。

1. 情感和语调传达不足：

- 文字交互难以传达说话者的语调、情感和非语言提示，这可能导致误解或沟通不畅。

2. 信息输入速度较慢：

- 相比于语音输入，文字输入通常较慢，这在需要快速响应的情况下可能是一个劣势。

3. 可访问性不佳：

- 对于视觉障碍用户，仅支持文字交互的AI系统可能不够友好，他们可能需要依赖屏幕阅读器等辅助技术。

4. 多任务处理能力较弱：

- 用户在进行多任务操作时，文字交互可能不如语音交互方便，因为语音交互可以让用户在执行其他任务时同时进行。

本研究的创新点与价值

1. 工具整合

- 多模块的工具整合：针对现有对话式人工智能工具未能有效整合的问题，将多个AI模块与实际应用工具进行无缝整合，大大提高了处理问题的效率。
- 关于超级终端的尝试：通过 Prompt Engineering，同时对 AI 开放运行代码的权限，尝试将对话 AI 变成一个人机交互的超级终端。

2. 改进主动性

- 用户情感需求分析：通过长期的对话历史、情感分析、个性化偏好等来改善AI的预测与反应，使AI能根据用户的情感，偏好等作出更好的主动回应。
- 任务驱动与目标管理：让AI具备任务管理能力，主动识别未完成任务或潜在需求，并根据情况进行追踪与提醒。
- 大数据赋能对话 AI：通过进一步收集数据，AI 能够更加灵活的帮助用户工作，例如提醒用户及时完成某项工作，管理用户日程安排。

3. 附加语音功能

- 语音输出与情感交互：语音功能不仅仅是将文字转换为语音，更增强情感表达的丰富度。使AI的回答更加人性化，增强了对话的情感反馈，让用户感受到更自然的互动体验。
- 语音识别与自然语言理解：与文字输入相比，语音输入能够传达更为丰富的情感和意图信息。AI能够精确地捕捉语音中的语调变化和说话人的情感状态，进一步改善与用户的互动体验。

综述

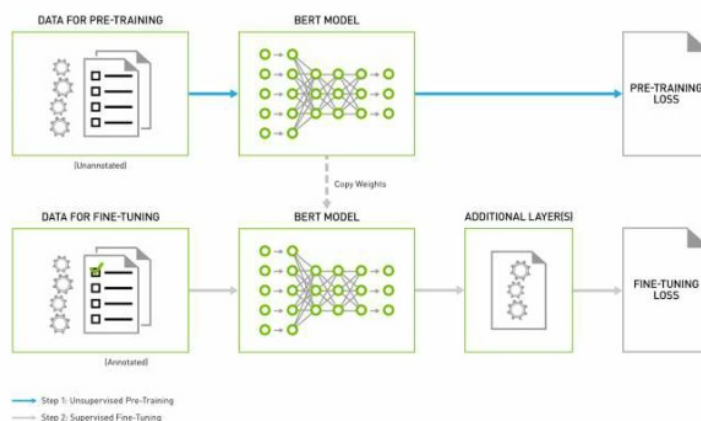
基于 Transformer 的 LLM（大型语言模型）

2017年，Transformer架构的出现，给自然语言处理（NLP）技术带来了颠覆性的变化。一种被称为大型语言模型（LLM）的深度学习模型出现了，它们表现出了非凡的理解自然语言和产生连贯反应的能力。LLM比传统神经网络复杂得多，通常包含数十个神经网络层，并具有数十亿甚至数万亿个参数。这些模型通常在大型数据集上进行训练，并基于Transformer模块构建架构。今天，LLM的多功能性使其能够执行各种任务，从文本生成和语言翻译到问答，代码生成和分析，甚至扩展到处理解释多种模式的数据，包括图像，音频和视频。这种增强使多模态大型语言模型（MLLM）能够处理可视内容，如视频编辑、图像理解和字幕生成。

模型原理

- 自注意力机制：**Transformer模型的核心是自注意力机制，它允许模型同时处理整个输入序列，而不是像循环神经网络（RNN）那样按顺序逐步处理。这种并行化处理方式显著提高了模型在处理长序列数据时的效率和速度，允许模型在训练过程中同时考虑序列中的所有元素，从而加快了学习过程并提高了模型的可扩展性。
- 自监督学习：**LLM通常采用预训练和微调的策略，这使得模型能够在大规模无标签数据上进行预训练，学习通用的语言表示。预训练阶段，模型通过自监督学习任务（如掩码语言模型或下一句预测）来捕捉语言的统计特性和语义信息。之后，在特定任务的有限数据上进行微调，模型能够快速适应新任务，同时保持了模型的泛化能力。这种策略不仅提高了模型的效率，还使得模型能够灵活地应用于各种不同的NLP任务。

Pre-Training vs. Fine-Tuning

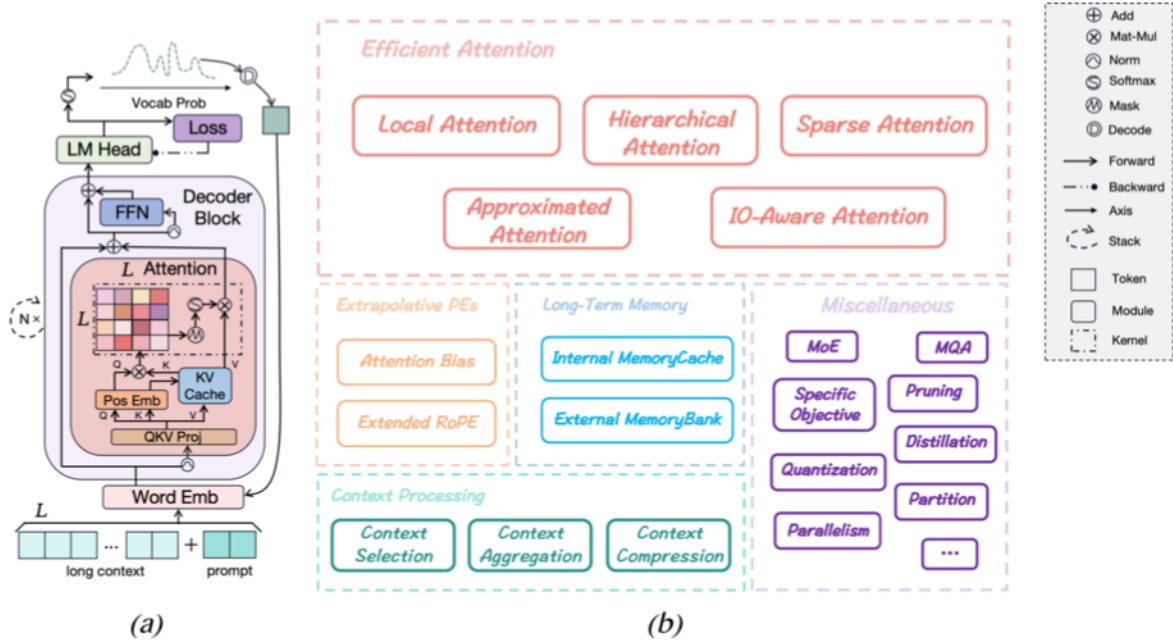


CSDN @镰刀韭菜

模型优势

- 模型解释性：**尽管深度学习模型通常被认为是“黑箱”，但Transformer模型的自注意力机制提供了一定程度的可解释性。通过分析注意力权重，研究人员可以观察到模型在做出决策时关注了输入序列的哪些部分，这有助于理解模型的行为和预测。例如，当模型被要求识别句子中的关键信息或解释其预测时，注意力图可以提供直观的证据，展示模型是如何聚焦于特定的词或短语的。这种可解释性对于提高模型的透明度、建立用户信任以及调试模型错误都非常重要。

2. **长距离依赖捕捉**：Transformer通过自注意力机制能够捕捉输入序列中任意两个位置之间的依赖关系，无论它们相隔多远。这种能力对于理解语言中的长距离依赖至关重要，因为语言中的语义和语法关系往往跨越很长的距离。例如，在解析复杂的句子结构或理解上下文中的指代关系时，Transformer能够更准确地建模这些关系，从而提高语言理解的准确性。



基于 Conformer 模型的 ASR（实时语音听写）

基于Conformer模型的ASR（自动语音识别）是一种结合了Transformer和卷积神经网络（CNN）优势的端到端语音识别技术，它有效地捕捉音频序列的局部和全局依赖关系，实现了参数高效的建模。

模型结构

Conformer模型由四个模块堆叠而成：前馈模块、自注意力模块、卷积模块和第二个前馈模块。自注意力模块集成了相对正弦位置编码，提升了模型对不同输入长度的泛化能力。卷积模块则利用门控机制和一维深度卷积，有效地捕获局部特征。

相关性能测试

在LibriSpeech基准测试中，Conformer模型展现了卓越的性能，未使用语言模型时的词错误率（WER）为2.1%/4.3%，使用外部语言模型时降至1.9%/3.9%。即使是参数量较小的模型（仅10M参数），也表现出了2.7%/6.3%的竞争力性能。

模型优势

- 参数效率**：Conformer以参数高效的方式建模音频序列的局部和全局依赖，相较于之前的Transformer和CNN模型，Conformer显著提升了性能。
- 小模型的竞争力**：即使在模型参数仅为10M时，Conformer也表现出了2.7%/6.3%的竞争力性能。
- 泛化能力**：Conformer模型在不同长度的输入上展现出良好的泛化能力，特别是在集成了相对正弦位置编码的多头自注意力模块中。

模型缺陷

- 对训练数据的依赖：**Conformer模型的性能在很大程度上依赖于训练数据的质量和多样性。在数据分布不均匀或数据量不足的情况下，模型的性能可能会受到影响。
- 模型部署的挑战：**由于模型的复杂性，Conformer在实际部署时可能面临计算资源和延迟的挑战，尤其是在资源受限的设备上。

基于 Diffusion 算法的 TTS

文本转语音（TTS）技术作为人工智能领域的一个重要分支，近年来取得了显著的进展。特别是基于深度学习的TTS模型，能够生成接近人类自然语言的语音，极大地推动了语音合成技术的发展。Fish Diffusion算法作为一种新兴的TTS框架，以其独特的优势和应用前景，引起了学术界和工业界的广泛关注。

Fish Diffusion 算法概述

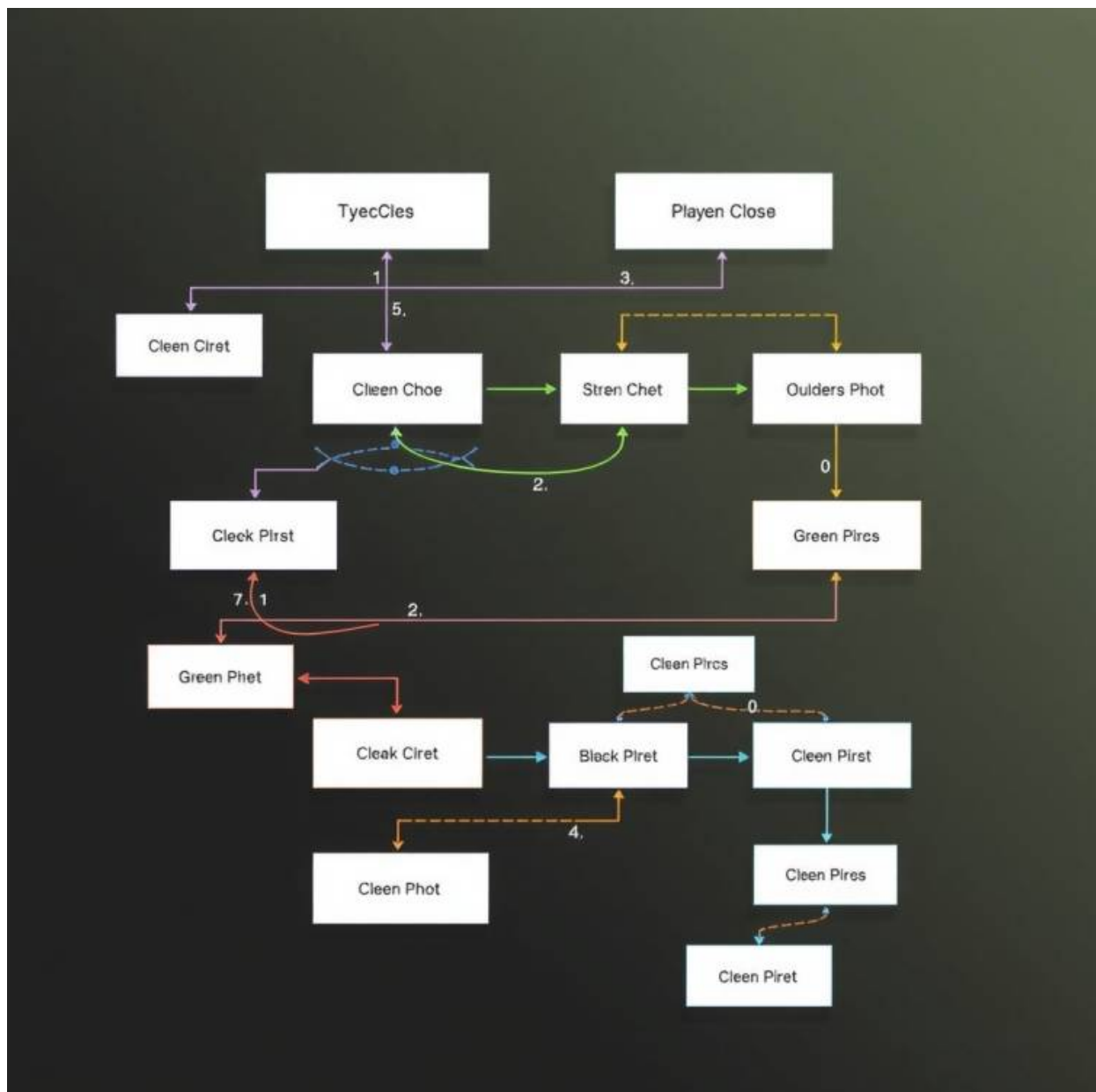
Fish Diffusion是一个基于扩散模型的语音生成框架，旨在解决文本转语音（TTS）等多种语音任务。与传统方法相比，Fish Diffusion采用了最新的扩散模型技术，能够生成更加自然、高质量的语音。

Diffusion 算法原理

Diffusion算法的核心思想是模拟物理扩散过程，通过逐步减少噪声来生成数据。这个过程可以被看作是时间反转的扩散过程，其中模型学习如何从噪声数据中恢复出原始信号。以下是Diffusion算法的几个关键步骤：

- 初始化：**从一个简单的噪声分布开始，通常是一个高斯分布。
- 逐步扩散：**通过一系列确定性的步骤逐渐增加噪声，直到数据完全转化为噪声。
- 学习反转：**模型学习如何逆转扩散过程，从噪声中恢复出原始数据。
- 生成数据：**给定一个噪声样本，模型逐步减少噪声，最终生成目标数据。

Diffusion算法的优势在于其生成数据的质量和多样性。由于其逐步减少噪声的特性，Diffusion算法能够生成更加平滑和连贯的数据，这对于语音合成尤其重要。此外，Diffusion算法的高适应性使其能够处理各种不同的数据类型和任务。



技术分析

Fish Diffusion利用了最新的扩散模型，以其强大的噪声消除能力和对数据集的高适应性而闻名。扩散模型是一种生成模型，它通过逐步减少噪声来生成数据，与传统的生成对抗网络（GAN）和变分自编码器（VAE）相比，扩散模型在生成质量和多样性上展现出了显著的优势。

技术优势

- 生成质量：**Diffusion算法能够生成高质量的语音数据，因为它通过逐步减少噪声来恢复出原始信号，这种方法能够生成更加平滑和连贯的语音，减少了传统方法中常见的不自然跳跃和失真。
- 多样性和灵活性：**由于Diffusion算法的逐步噪声减少过程，它能够生成具有高度多样性的语音样本，这对于模拟不同说话人的声音特征尤为重要。
- 数据集适应性：**Diffusion模型能够适应各种不同的数据集，包括小规模和非标准数据集，这使得它在数据资源有限的情况下也能表现出色。
- 训练效率：**Fish Diffusion支持多机多卡训练和半精度训练，这大大提升了训练速度，同时减少了内存和计算资源的需求，使得大规模训练成为可能。

- 5. **模型泛化能力**：Diffusion算法的生成过程不依赖于特定的数据分布，因此具有很好的泛化能力，可以应用于多种不同的语音合成任务。
- 6. **模型迁移和配置灵活性**：Fish Diffusion支持多种配置和模型迁移，这使得研究人员和开发者可以根据自己的需求进行实验和优化，提高了模型的适用性和灵活性。

主要特点

- 1. **多说话人支持**：Fish Diffusion实现了多说话人的语音生成，提高了模型的灵活性和应用范围。
- 2. **简洁易懂的代码结构**：所有模块都经过解耦，易于理解和修改。
- 3. **高质量声码器**：集成了44.1kHz的声码器，确保生成的语音具有优秀的音质。
- 4. **高效训练**：支持多机多卡训练和半精度训练，提升了训练速度并节省了内存资源。

总结

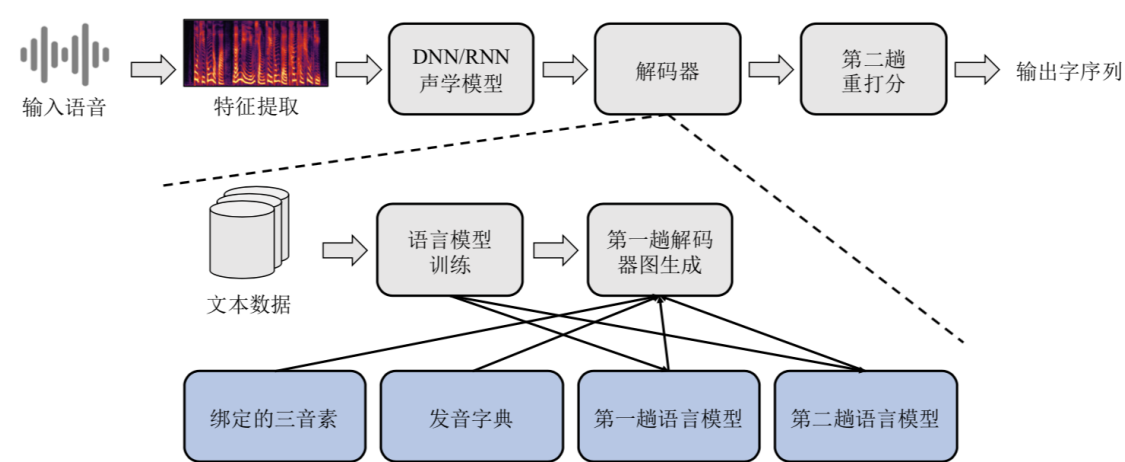
Fish Diffusion作为一个易于理解和使用的TTS框架，为语音合成技术的发展注入了新的活力。它不仅为研究人员提供了一个强大的实验平台，也为开发者和创作者提供了实现创意的工具，推动了TTS技术在各个领域的应用和发展。

基于端到端神经网络的 KWS（关键词唤醒）

关键词唤醒技术（KWS）

关键词唤醒技术（KWS）旨在实时检测音频流中的特定语音命令以唤醒大模型。传统的KWS系统依赖于声学模型、特征提取与语言模型的组合，但随着深度学习的发展，基于端到端神经网络的KWS方法逐渐成为研究热点。端到端神经网络能够通过一个统一的模型直接从原始音频信号中学习特征，简化了传统方法中复杂的特征提取和后处理步骤。

- 传统语音识别语言模型训练



<https://blog.csdn.net/u014365862>

基于端到端神经网络的音频处理

传统的KWS方法包括多阶段的处理流程：音频信号预处理、特征提取（如梅尔频率倒谱系数MFCC）、声学模型（如隐马尔可夫模型HMM）和解码器。这些方法依赖于手工设计的特征，计算复杂且对噪声较为敏感。相比之下，端到端神经网络通过单一的神经网络模型处理音频数据，能够自动学习到最优特征，提升了系统的整体性能。

神经网络架构

端到端KWS系统常用的神经网络架构包括卷积神经网络（CNN）、循环神经网络（RNN）及其混合模型。CNN能够有效提取局部频谱特征，适用于短时信号的特征学习；RNN，尤其是长短期记忆网络（LSTM），能够处理音频中的时序信息，适合捕捉关键词在时间上的变化。CNN和RNN的混合架构常被用于结合局部特征提取和全局时序建模，提升KWS的准确性。近年来，基于自注意力机制的Transformer模型也被引入KWS任务，因其在长序列处理中的优势，展现了广阔的应用前景。

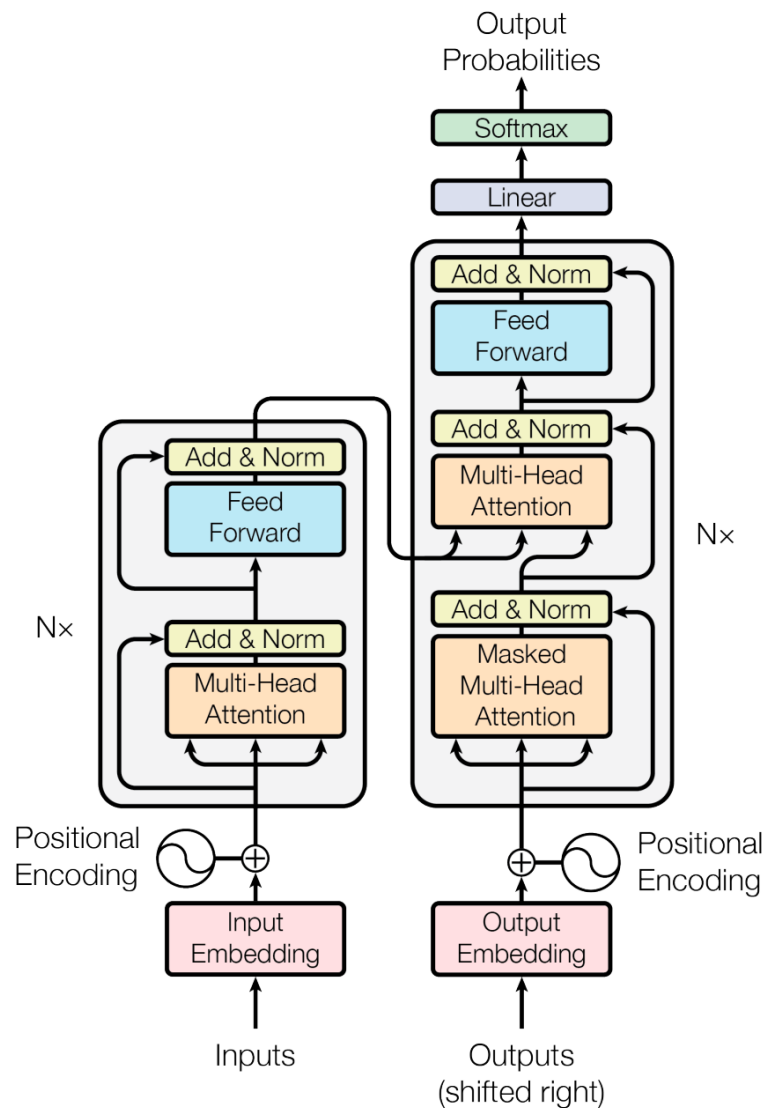


Figure 1: The Transformer - model architecture.

CSDN @AI架构师易筋

系统架构设计

语音处理

KWS 部分

本 AI 系统可以通过关键词 "小妍小妍" 唤醒。

关键词经过音节成分的充分考量，利于 AI 识别的同时可以减少日常其它交流的误唤醒。

我们使用了科大讯飞提供的 SDK（Software Development Kit）简化了开发流程。

针对现有 KWS 技术的优缺点，我们在工程方面做了以下努力：

1. **本地化部署**：在计算资源受限的条件下进行本地化部署在工程上具有一定的挑战性。为此我们选取了较小的唤醒模型，在牺牲一定精度的前提下完成了本地化部署，同时使得唤醒部分的功耗得到有效控制。
2. **HTTP进程通信**：科大讯飞的 SDK 套件采用纯 C++ 实现，为了减少 ASR 部分的计算负载，我们采用唤醒——侦听机制来激活**流式语音转文本**。因此采用 HTTP 作为跨进程信息传输方式。

ASR 部分

在接收 KWS 部分的唤醒操作后，ASR 部分开始流式侦听。

调用系统麦克风录音后将音频流按照 40ms 切分开来分批发送到 ASR 识别部分。

不同段的音频流会进行略有重叠的拼接并调用识别语音的模块，语音识别模块采用了动态更新机制，识别结果可能会根据后续的完整发音发生一定改变。

这样的设计使得流式侦听可以进行断句，使得语义传达更加精确。

在断句后，与 ASR 部分集成的大模型会额外进行语义检查并纠正发音相同导致的错音字。

任务调度

系统概述

该系统通过一个 `UserTask` 类提供了一个接口，用于与后端数据库交互，管理任务的生命周期。系统的核心功能包括：

- **任务添加**：允许用户通过自然语言指令添加新任务。
- **任务删除**：允许用户通过自然语言指令删除现有任务。
- **任务查询**：允许用户查询所有任务或特定任务。

通过自然语言指令，用户可以轻松地与系统交互，实现任务的自动化处理。

工作流程

用户可以用自然语言描述任务，通过系统解析提取关键信息，并通过API添加到数据库中。类似地，用户可以通过自然语言指令指定要删除的任务，系统解析该指令，确定要删除的任务，并调用API将其从数据库中删除。用户还可以请求查看所有任务或特定任务，系统调用API获取任务列表并将其呈现给用户。关键信息通常是任务的描述，通过这些关键信息可以对任务进行分类，从而大大优化了任务列表的获取效率。

技术原理

1. **上下文理解与任务属性提取**：提示中包含的上下文信息帮助AI模型理解任务的背景，从而在执行任务时考虑到相关因素。例如，如果用户提到“下周的会议”，提示可以被设计来询问“您是指哪一天的会议？”以确保系统能够准确捕捉任务的时间属性。这种上下文理解能力对于任务管理至关重要，因为它允许系统在执行任务时考虑到所有相关细节。
2. **个性化响应与用户指导**：根据用户的偏好和历史行为，提示可以被个性化，以提供更符合用户期望的响应。同时，提示也可以提供指导，帮助用户更有效地与系统交互。例如，如果系统识别出用户经常在周末安排任务，提示可以是“您是否希望在本周末添加新任务？”这样的个性化提示可以提高用户满意度并减少错误。

技术优势

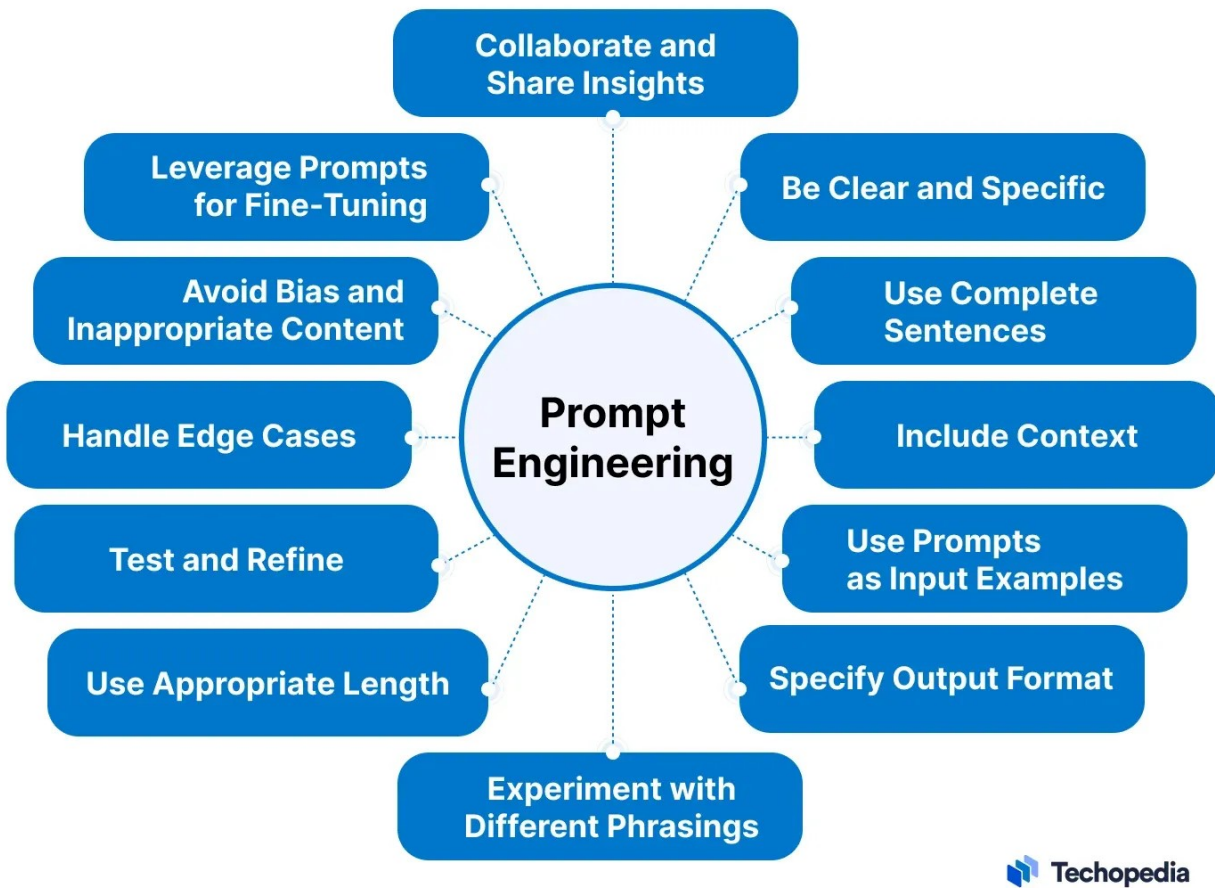
1. **增强的搜索和检索**：分类系统使用户能够快速搜索和检索特定类型的任务，这对于管理大量任务特别有效。
2. **支持资源分配**：任务分类有助于识别时间、人力、物力等资源需求，从而更有效地规划和分配资源。
3. **易于监控和报告**：分类任务可以更容易地监控和报告，这对于项目管理和绩效评估至关重要。
4. **减少认知负担**：通过自动分类，用户不需要花额外的精力去记忆或组织任务，这有助于减轻认知负担，让用户专注于任务本身。
5. **增强用户体验**：自然语言交互的设计大大降低了使用门槛，用户无需学习特定的命令或语法，即可以最自然的方式与系统交互，极大地提高了用户满意度。
6. **灵活性和可扩展性**：系统的设计允许轻松扩展新功能，如任务优先级，截止日期提醒等，以满足不同用户的需求，这种灵活性是传统任务管理工具难以比拟的。

主动反馈

总体架构设计分为：提示词（prompt），数据库查询与数据处理，决策与决策结果处理三个部分。

提示词

- 提示词是主动反馈决策流程中的核心，起到了桥梁作用。它不仅规定了智能体当前的任务和决策要求，还决定了决策依据的来源（即数据库中的数据）、决策方式、以及决策后的应答格式。通过精心设计提示词，我们可以确保智能体在执行决策时获得最准确、最相关的上下文信息，帮助其做出更加合适的响应。
- 在这一过程中，prompt engineering（提示工程）的关键作用在于如何有效地引导智能体关注到合适的的数据、任务背景和决策依据。合理构建提示词，可以让智能体理解当前情境，从而在多变的输入中保持高效和准确的决策执行。例如，通过优化提示中的任务描述、情境背景和历史信息，智能体能够更好地识别用户需求，做出灵活且及时的回应。



数据库查询与数据处理

- 通过web端数据库的接口依次查询 用户与智能体的对话记录，用户目前的任务安排，智能体目前已经作出的决策记录。并根据目前所查询到的数据的数量对数据进行不同的处理。例如：在数据较少时会将数据完全添加至本次传递给AI的提示词当中，而当数据数量较大时，会对不同类型的数据做不同的处理，例如，对于聊天记录会采取摘要获取的形式，而任务列表和决策记录则会采取获取一定数量最近有效记录的形式。
- 这一过程对于prompt engineering同样至关重要，因为提示词的设计需要决定哪些数据最为关键，如何将其简洁而有效地传达给智能体，避免冗余或无关信息的干扰。

决策与决策结果处理

在prompt中规定，智能体在主动决策之中有三种决策结果，唤醒（向决策记录中添加一条主动唤醒决策），沉默（对于目前的主动决策不进行任何的调整），删除（根据用户临时的日程变动，删除之前已经作出的，由于日程变动而变得不再合适的主动唤醒决策），不同的结果会返回不同的结果标记，而唤醒决策会额外返回下一次唤醒时的时间与下一次唤醒时主动回复的内容，删除决策会返回需要删除的主动唤醒决策在数据库中的键值）。

可持久化

概要

可持久化通过在web维持一个关系型数据库实现历史记录查询，来进行智能体优化与用户个性化的实现

关系型数据库

关系型数据库的优点

1. **数据结构化**：数据以表格的形式存储，每张表由行和列组成，这种结构化使得数据的存储和查询变得直观和方便。
2. **数据完整性**：通过使用主键、外键和约束（如唯一性约束、非空约束等），关系型数据库能够保证数据的完整性和准确性。
3. **事务性**：支持事务处理，确保数据的一致性和可靠性。事务具有原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）的特性，即ACID特性。
4. **可扩展性**：可以通过添加更多的表和字段来轻松扩展数据库结构，以适应不断变化的业务需求。
5. **灵活性**：支持复杂的查询，包括多表连接、子查询、分组、排序等操作，使得数据检索非常灵活。

关系型数据库的实现

通过导入Python flask_sqlalchemy库中的SQLAlchemy模块，实现基本的关系型数据库的操作，其中包括创建、上传、修改、查询、删除等操作。每次执行代码时会创建一个新的数据库或打开已有的数据库，用户在使用数据库时须先创建一个表，每个表中的每组数据都有唯一的key来作为标签key的值由用户确定，并在上传数据时一并传入。如果用户后续需要操作某组数据，只须提供表单名称和key值即可。用户上传的每组数据元素数不超过五个，且默认为：任务名称、开始时间、结束时间、描述、附加信息，例如['成绩', '20240905', '20240905', '94', 'English']用户也可以根据自己的需要自己定义任意个（不超过五个）元素所代表的意义，例如['难过']。

web端维护数据库

通过导入Python flask库中的Flask, request, jsonify模块，实现主机端服务器创建，处理用户端操作请求，返回格式化的数据等操作。用户可通过url访问服务器的数据库，且上传、修改、查询这些操作会以字典的形式返回表单，key值和数据方便用户保存整理提取的数据。对于数据这个数组，他不会返回值为空的元素，所以对于数据['难过']实际上是以['难过',' ',' ',' ']'的形式存储，这方便了用户的修改，减少索引错误的风险。

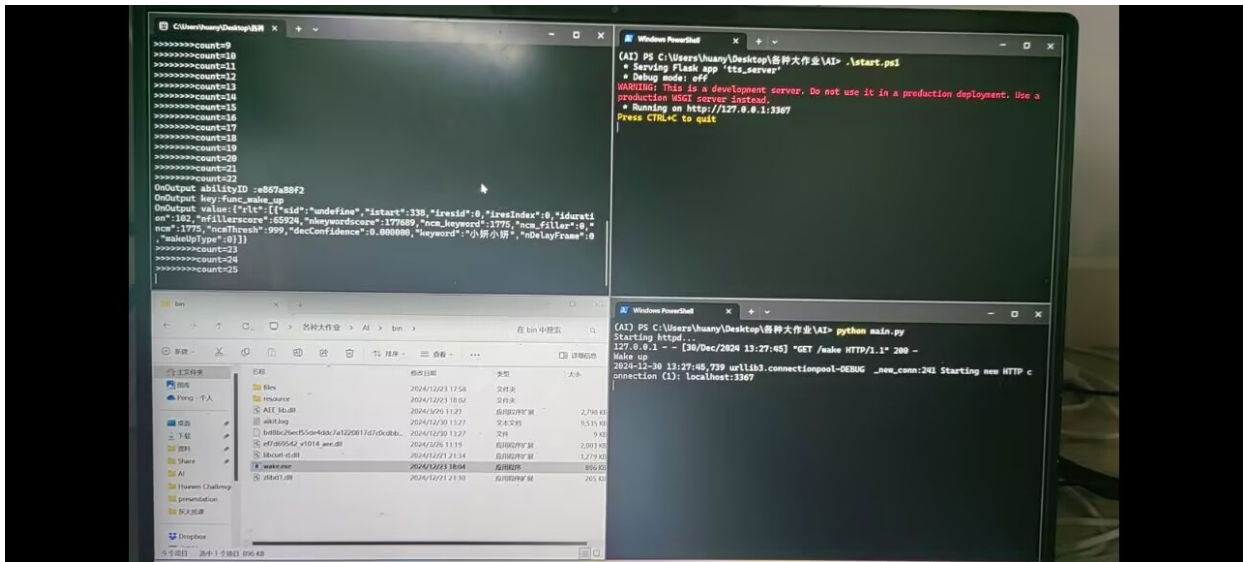
实验

环境搭建

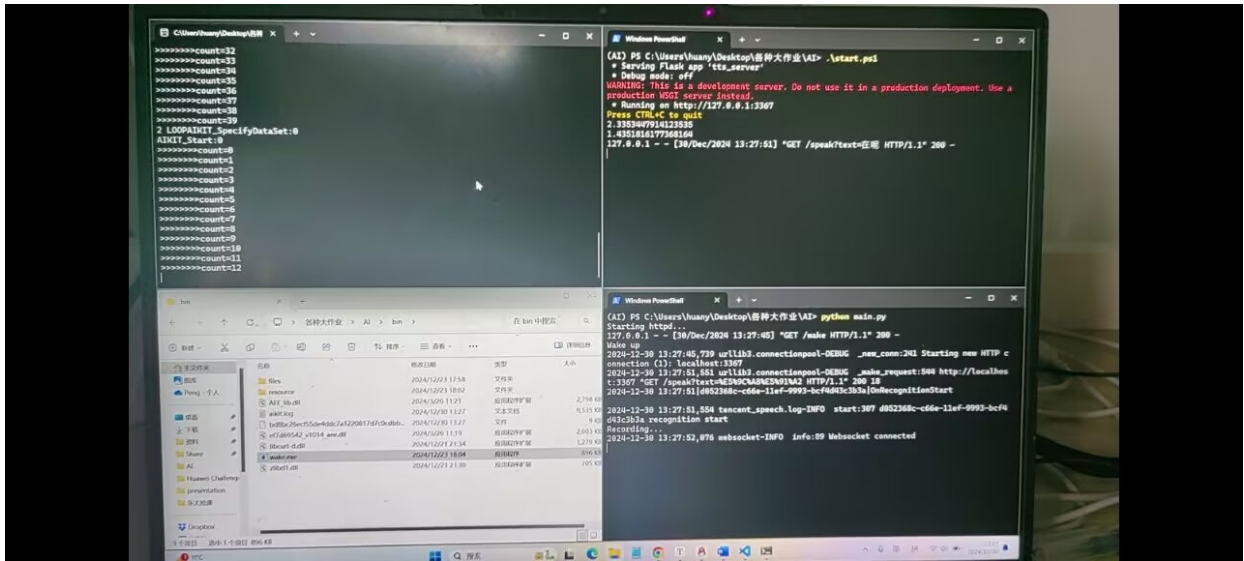
本实验的复现使用 Windows11 操作系统，按照附录中开源项目地址中的引导完成环境配置，依次激活 KWS, TTS, main 三个部分后开始正常运行。

侦听功能测试

当用户念出 "小妍小妍" 时，侦听器成功检测到唤醒，并将信息通过 http 方式发送给主程序。



主程序启动后发布 TTS 任务，通过云函数得到响应后生成回复音频，再由主程序在本地播放。



此时播放预设的回复 "在呢"，同时激活流式语音检测函数，日志中打印 "Recording" 记录中信息。

任务管理测试

唤醒后告诉小妍 "明天" 的体育上午的理论考试，通过在 Prompt 中加入系统时间，小妍自动推算出精确的绝对时间。基于该精确时间后自动设置提醒任务，一般会在当晚进行提醒。

语音提取：明天上午有一个体育理论考试，你帮我记一下，然后记得提醒我

```
加载个人及系统配置文件用了 1744 毫秒。
PS C:\Users\huany\Desktop\各种大作业\AI> .\start.ps1
* Serving Flask app 'tts_server'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:3367
Press CTRL+C to quit
2.2762608528137207
1.665248155593872
127.0.0.1 -- [01/Jan/2025 10:22:17] "GET /speak?text=在呢 HTTP/1.1" 200 -
3.1908810138702393
7.093207597732544
127.0.0.1 -- [01/Jan/2025 10:22:54] "GET /speak?text=好的，明天上午的体育理论考试已经帮你记下了。我会在考试前提醒你的。
HTTP/1.1" 200 -
```

二次唤醒，告诉小妍 "周二"（和前文的 "明天" 指同一天）上午九点，小妍识别出这是一条安排任务的指令后调用集成式工具查询数据库检测，检测后存在冲突，提醒用户在两个任务之间做出安排。

```
回复 好的，你周二上午9点和同学约了出去吃饭。我会帮你检查一下这个安排和明天上午的体育理论考试是否有冲突。

明天是星期二，你的考试和吃饭安排都在星期二。考试是上午，吃饭也是上午9点。所以这两个安排有冲突，你需要调整一下时间。

我建议你可以把吃饭时间改到中午或者下午，这样就不会影响你的考试了。你觉得呢？
```

用户完成决策后，小妍确认并加入数据库。

```
回复 好的，我明白了。你打算将吃饭时间改到晚上。现在我会帮你给你的同学发短信。

短信内容如下：

亲爱的同学，
我们原定明天（周二）上午9点的聚餐，因为明天上午我有体育理论考试，所以我想将聚餐时间改到明天晚上。具体时间我们稍后再定。希望这个变动不会给你带来不便，期待晚上与你相聚。
祝好，
[你的名字]

你觉得这样的短信内容可以吗？如果没问题，我就帮你发送了。
```

暂未集成发送短信的部分，上图仅供展示。

总结与展望

实验结论

通过采用 AI Agent，能够进一步改进对话式人工智能的能力。定制化的 AI Agent 能够在垂直度更高的领域发挥作用。

存在的问题与改进方向

- 集成的功能较少且不稳定，所有功能都需要提前手工编写代码，距离实际应用仍有很大的距离。
- Prompt Engineering 部分多采用大模型自监督确认，算力需求大，难以脱离云端计算资源运行。
- ASR 受到背景杂音干扰严重，会影响模型回复的准确性，同时这类对话历史容易污染后续 Prompt。

未来技术延拓

以自编程的形式拓展功能

现有大模型已经具备充分且可靠的代码编写能力，通过接入具有可靠编码能力的大模型，AI 可以自编程完成集成工具并加入到自身源代码中，为用户提供更加全面的服务。

但开放完全的系统权限可能会导致一些不可预料的问题，AI 的代码能否完全信任？AI 的自编程是否会导致原有系统崩溃等，这些都需要进一步的实验。

语音抗干扰

语音抗干扰有两个主要方向。

一方面可以通过 ASR 部分完成说话人识别以排除背景杂音，但这样会导致某些记录场景（例如期望 AI 发挥书记员作用记录重要谈话）无法完成预期功能。

另一方面，大模型可以根据上下文自动分拣语音文本，但这类下游任务需要更进一步的微调。

大数据

提供更全面立体的数据供 AI 进行决策分析，能够有效提升 AI 回复的准确度，提升用户体验。

附录

部分核心代码

流式侦听

```
try:
    def record_audio(callback):
        CHUNK = 1024
        FORMAT = pyaudio.paInt16
        CHANNELS = 1
        RATE = 16000
        RECORD_SECONDS = 120

        p = pyaudio.PyAudio()

        stream = p.open(format=FORMAT,
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        frames_per_buffer=CHUNK)

        print("Recording...")

        frames = []

        for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
            if time.time() - last_action_time > INTERVAL_LIMIT:
                break
            data = stream.read(CHUNK)
            frames.append(data)
            callback(data)

        print("Recording finished.")

        stream.stop_stream()
        stream.close()
        p.terminate()

    # 记录日志
    wf = wave.open(f"temp/{time.time()}.wav", 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(p.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    wf.writeframes(b''.join(frames))
    wf.close()
```



```

def process_audio_data(data):
    recognizer.write(data)

recognizer.start()
record_audio(process_audio_data)
try:
    while True:
        time.sleep(0.1)
        action_time_lock.acquire()
        if time.time() - last_action_time > INTERVAL_LIMIT:
            action_time_lock.release()
            print("Recive End")
            voices = []
            while voice_text_queue.qsize() > 0:
                print("GET TEXT")
                voices.append(json.loads(voice_text_queue.get())["result"]
["voice_text_str"])
            return "\n".join(voices)
            action_time_lock.release()

    except KeyboardInterrupt:
        pass
except Exception as e:
    print(e)
finally:
    recognizer.stop()

```

任务调度

```

from api.chat import get_kimi_api_response
import requests

TABLE_NAME = "dialogue"
CHECK_DEL = "请分析以下文本是否包含删除任务的指令，并根据以下规则处理：\n"
CHECK_DEL += "1. 如果不是指令，返回 None。 \n"
CHECK_DEL += "2. 如果是指令，检查数据库中是否存在对应的任务。 \n"
CHECK_DEL += "    - 如果任务存在，执行删除并返回 (True, 删掉的任务, AI回复)。 \n"
CHECK_DEL += "    - 如果任务不存在，返回 (False, None, AI回复)。 \n"
CHECK_ADD = "请分析文本以下是否包含添加任务的指令，并根据以下规则处理：\n"
CHECK_ADD += "1. 如果不是指令，返回 None。 \n"
CHECK_ADD += "2. 如果是指令，检查数据库中是否存在冲突的任务。 \n"
CHECK_ADD += "    - 如果没有冲突，执行添加并返回 (True, 增加的任务, AI回复)。 \n"
CHECK_ADD += "    - 如果存在冲突，返回 (False, None, AI回复)。 "
CHECK_KEY = "以下任务的时间是什么时候（以年-月-日形式输出）"
UPDATE_HISTORY = "在任务列表中加入以下任务\n"

class UserTask:
    def __init__(self):
        pass

    def add_task(self, input_task):

```

```

        url = f'http://127.0.0.1:5000/{TABLE_NAME}'
        payload = {'key': get_kimi_api_response(CHECK_KEY + input_task), 'value':
input_task}
        response = requests.post(url, json=payload)
        return response.json()

    def delete_task(self, input_task):
        url = f'http://127.0.0.1:5000/delete/{input_task}'
        response = requests.delete(url)
        return response.status_code

    def get_all_data(tablename):
        url = f'http://127.0.0.1:5000/by_table/{tablename}'
        response = requests.get(url)
        return response.json()

def try_del_task(text):
    table = UserTask()
    history = ""
    list = table.get_all_data(TABLE_NAME)
    for task in list:
        history += task[3] + "\n"
    response = get_kimi_api_response(UPDATE_HISTORY + history + CHECK_DEL +
text).split()
    if response.length() != 3:
        return None
    if response[0] == True:
        table.delete_task(response[1])
    return [response[0], response[1], response[2]]

def try_add_task(text):
    table = UserTask()
    history = ""
    list = table.get_all_data(TABLE_NAME)
    for task in list:
        history += task[3] + "\n"
    response = get_kimi_api_response(UPDATE_HISTORY + history + CHECK_ADD +
text).split()
    if response.length() != 3:
        return None
    if response[0] == True:
        table.add_task(response[1])
    return [response[0], response[1], response[2]]

```

主动反馈

```
from api.chat import get_kimi_api_response as api
import time
import requests
from datetime import datetime
url = "http://127.0.0.1:5000/by_table/"
delete_url="http://127.0.0.1:5000/delete/decides/"
prompt = """
    任务要求： 你需要根据现在的时间，
    用户目前的行程安排和之前已经做出的主动交流决定，
    判断现在是否需要再加一次主动交流或暂时不需要更多的主动交流
    又或是用户的行程安排发生了改变，需要删除之前作出的主动交流决定
    主动交流内容包括对用户目前的任务做时间提醒，日常问候，或者情感关怀
    若不需要更多的主动交流，你只需要返回 false 这个单词就好
    若需要新增一次主动交流，你需要返回两块内容：
    首先，是下一次主动交流的时间，
    请将时间以相同于2024-11-23 08:55:29的格式给出，
    然后，是主动交流的内容。
    在两块内容之间，以一个 | 符号作为分隔
    若需要删除某个标号所对应的，尚未执行的主动交流决定
    你需要返回 delete 这个单词和决定的标号，中间用|分割
    注意：若后方的日程安排,聊天记录或主动交流决定后面没有记录，
    表示目前暂无内容，并非异常情况
    尽量避免在同一个时间点作出多次相同的主动交流
    作为ai伴侣，尽量主动一些，也可以根据用户反馈作出调整，
    下方为一些回复示例：
    eg1.（调用时恰好是中午）-> 2024-12-06 12:00:00|喵喵喵，主人记得吃饭哦。
    eg2.（调用时发现某个 Task 的开始时间快到了）-> 2024-12-06 14:00:00|主人记得去开会哦，今天下午五点
    eg3.（调用时发现最近的对话信息中用户情绪不对劲）-> 2024-12-07 22:00:00|有什么难过的事情吗？小妍可以听听吗？一个人憋着很难受吧...
    eg4.（调用时发现没什么可说的/之前主动交流决定记录中以及有过类似问候了）-> false
    eg5.（调用时发现标号为 12 的,目前 尚未执行 的主动交流决定因为日程变动而不再合适 -> delete|12
    """"

def process(returned):
    message = ''
    if(isinstance(returned,list)):
        for item in returned:
            message += item['key']
            message += ' '
            for val in item['value']:
                message += val
                message += ' '
            message += '\n'
    else:
        if (returned['error']=='Table is not existed'):
            return message
        message += returned['key']
        message += ' '
        for val in returned['value']:
```

```

        message += val
        message += ' '
    message += '\n'

    return message

def next_active():

    message = ""
    response=[]
    message += prompt
    chain = []
    dic = {}
    dic["role"] = "system"

    current_time = datetime.now()
    current_time = str(current_time.strftime("%Y-%m-%d %H:%M:%S"))
    message += "当前时间是："
    message += current_time

    dialogue = requests.get(url+"dialogue")
    dialogue = dialogue.json()
    message += "\n此前的对话记录是\n"

    message += process(dialogue)

    task = requests.get(url+"Task")
    message += "\n默认的任务属性为 任务名称 开始时间 结束时间 描述 附加信息 但是不一定如此\n"
    message += "当前用户的日程安排是:\n"
    task = task.json()
    message += "\nTask: \n"

    message += process(task)

    decides = requests.get(url+"decides")
    message += "\n此前作出过的主动交流决定时间及其内容为:\n"
    decides = decides.json()

    message += process(decides)
    #print(message)
    dic["content"] = message
    chain.append(dic)

    reply = api(chain)
    print(reply)
    time.sleep(30)

    if(reply=="false"):
        response=[False,"",current_time]
        return response

    reply = reply.split("|")

```

```

if(reply[0]=="delete"):
    requests.delete(delete_url+reply[1])
    response=[False,"",current_time]
    return response

response.append(True)
response.append(reply[1])
response.append(reply[0])

key = hash(reply[1])
key = str(abs(key))
data={}
data["table"] = "decides"
data["key"] = key
data["value"] = [reply[1],reply[0]]

url_post = f'http://127.0.0.1:5000/{data["table"]}'
url_create =f'http://127.0.0.1:5000/creattable/{data["table"]}'
requests.post(url_create,json=data)
requests.post(url_post,json=data)

return response

if(__name__=="__main__"):
    print(next_active())

```

参考文献

- Wallace, E., Feng, S., Downey, D., et al. (2021). Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021), 72–85.
- Li, J., Zhang, S., & Zheng, X. (2021). End-to-End Keyword Spotting with Transformer. Proceedings of the 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2021), 2959–2963. <https://doi.org/10.1109/ICASSP39728.2021.9414261>
- Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is All You Need. Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017), 5998–6008. <https://doi.org/10.5555/3295222.3295344>
- Ho J, Jain A. Denoising Diffusion Probabilistic Models. *Advances in Neural Information Processing Systems*, 2020, 33: 6840-6851.
- Zhang Y, Wang Y. Deep Learning Based Text-to-Speech Synthesis: A Comprehensive Review. *arXiv preprint arXiv:1909.05550*, 2019.
- Valin J M, Skoglund J. Lattice-based speech codec with less than 1 bit per sample. *IEEE Transactions on Audio, Speech, and Language Processing*, 2007, 15(3): 1032-1044.

开源项目地址

本实验所需的全部代码于 github 开源, [代码仓库地址](#)