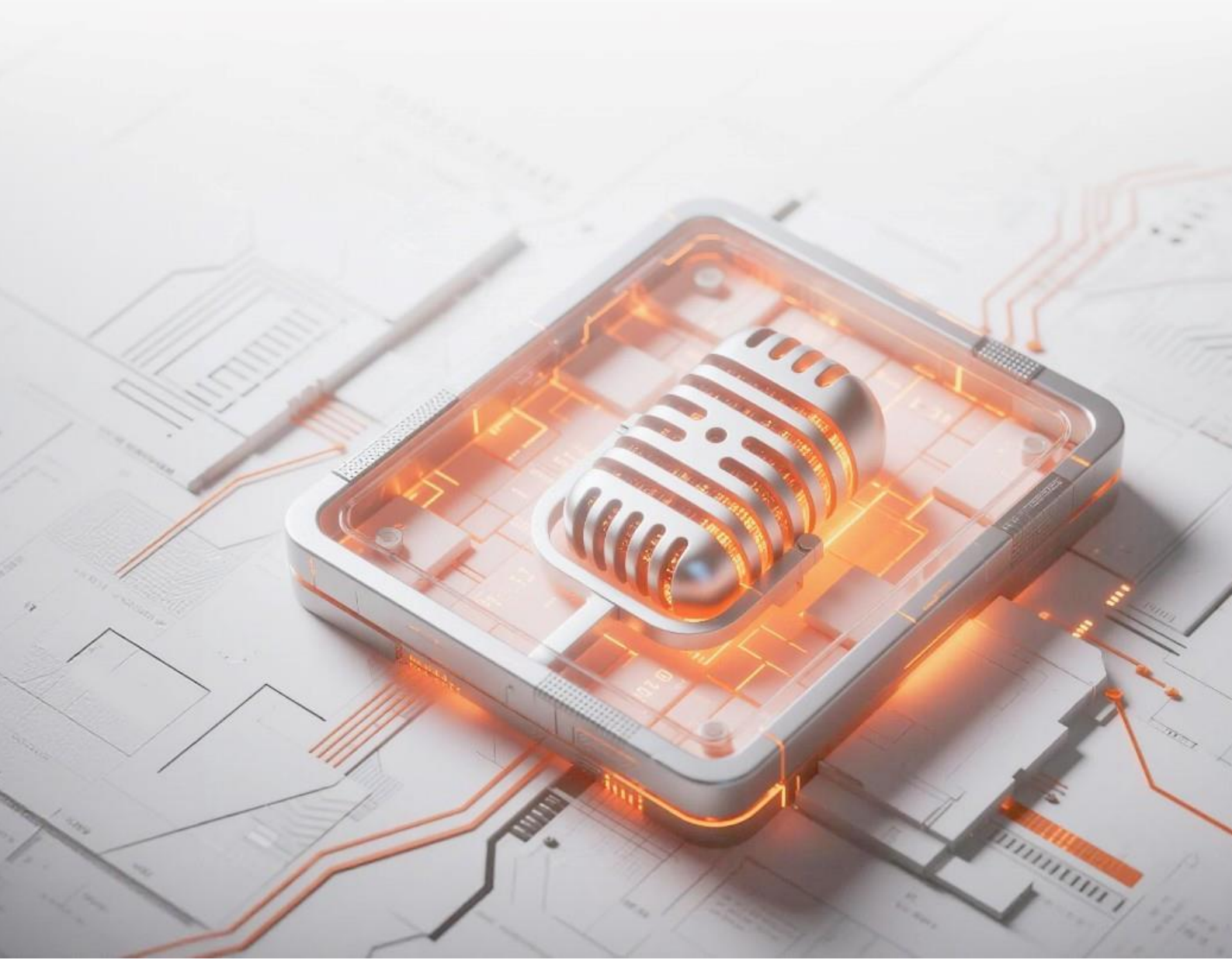
摩尔线程 MOORE THREADS | 紫光计算机 UNICOMPUTE TECHNOLOGY

语音识别全栈国产化技术实践白皮书



目录

一、 白皮书简介

《语音识别全栈国产化技术实践白皮书》阐述了一套基于基于紫光计算机集群和摩尔线程全功能 GPU 进行训练与推理的语音识别解决方案。该方案包含两大核心服务：适用于实时、低延迟场景的流式语音识别，以及适用于长音频、批量转写场景的离线语音转写。每个服务均自带完整音频预处理与后处理能力（VAD、ASR、标点生成与文本归一化 ITN），同时该方案支持热词增强与垂直领域语言模型定制，输出可选字级时间戳与句子级时间戳，满足实时转写、检索索引和高精度后处理需求。该系统已通过国家工业信息安全发展研究中心的检测（即“国检”），并持有相应的检测报告与证书，验证了其在多种噪声环境下的稳定性与识别准确性。

二、 应用场景

1. 电话客服质检与话单转写：支持大批量录音离线转写与准实时的通话监控，便于质检、关键词抽取与话单归档。
2. 会议与访谈记录：长时音频离线转写、自动切句和时间索引，便于内容检索和二次编辑。
3. 客服机器人/IVR 打点与实时转写：流式模型低延迟输出，结合热词和垂类 LM 提升行业用语识别率。
4. 媒体转写与字幕生成：自动标点、数字与格式归一化（ITN）保证字幕可读性与同步性。
5. 行业定制场景：金融、电力、医疗等需要垂直语言模型适配的场景，通过语言模型定制提升专业词识别能力。

三、 技术介绍

本产品采用先进的端到端对齐与预测融合架构，针对流式和非流式两类推理场景做了统一设计与专项优化。系统训练和推理均在摩尔线程全功能 GPU 上完成，结合国产 GPU 推理架构在算力与吞吐上实现协同提升。

(一) 主要功能

1. 支持 8k、16k 单声道音频请求。
2. 支持中文普通话、中英混合识别、英文识别。
3. 全链路语音处理能力

- 系统支持从原始音频到结构化文本的完整处理流程，支持对返回结果进行灵活的配置，包括：
  a) 语音活动检测（VAD）：自动识别语音段落，降低无效计算和错误触发概率。
  b) 端到端语音识别（ASR）：支持流式识别与离线识别两种形式，满足实时场景和长音频转写需求。
  c) 自动标点恢复：智能恢复句读符号，提升阅读体验和文本可用性。
  d) 文本归一化（ITN）：支持数字、金额、单位、字母等表达方式的规范化输出，便于业务处理与检索。
- 对于实时（流式）与离线（非流式）两类场景，系统均可输出：
  a) 字级时间戳：快速定位字和词对应时间节点。
  b) 句级时间戳：便于做质检、语音检索、字幕对齐等业务扩展。

4. 行业定制与增强能力

- 系统为实际业务落地提供多种效果增强能力：

  a) 热词自定义（Custom Hotwords）：支持业务专有名词（如企业名称、产品型号）快速增强识别。

  b) 垂类语言模型定制（Domain LM Adaptation）：可针对客服、金融、电力等领域进行深度 LM 微调，提高专业词识别率。

  c) 多场景适配：涵盖日常生活对话、电话客服、会议访谈等真实语音场景，支持中英文混合、数字与字母混读等复杂表达方式。

5. 多线程调用

- 支持多线程、多并发请求。

6. 高识别准确率

- 本产品已通过国家工业信息安全发展研究中心（以下简称“国检”）的权威检测与评估，检测报告与认证证书可作为第三方资质证明。采用关键词识别、连续语音识别、数字识别等多维度指标进行评估，在底噪、高噪环境下，上述测试的准确率均大于 90%。
- 国检结果表明，本产品在低噪与高噪两类典型业务环境下均表现出优异的识别准确性与鲁棒性，尤其在客服通话与日常对话场景对数字、字母及关键词的识别具有显著优势。

7. 高识别速度

- 摩尔线程的 MUSA 推理架构与算子级工程优化，在单张 MTT S4000 显卡上，非流式离线转写在 1 并发时的实时因子（RTF）仅为 0.02，在 10 并发时 RTF 为 0.06；流式实时识别在 1 并发时 RTF 为 0.08，在 10 并发时 RTF 为 0.23。

(二) 核心技术特色

1. 端到端统一架构

- 采用先进的端到端预测融合式结构，使得模型可以同时支持流式和非流式推理，确保延迟可控的同时保持高准确率。

2. 自注意力机制建模

- 结合摩尔线程 MUSA 推理架构，使用自注意力机制建模技术，提高识别的内容的准确性。

3. 大规模数据驱动与领域自适应

- 结合质检大模型技术，生成高质量的伪标签数据，用于流式/非流式识别模型的训练。
- 使用 NST（noise student training）技术迭代质检大模型，用于提升目标领域的伪标签标注效果。
- 训练使用数十万小时通用语料 + 数万小时电话客服垂类语料，确保模型在广谱场景与行业场景均有扎实基础。
- 通过专门的垂类语言模型微调与深度适配，实现对专业术语、企业名与业务表达的高命中率。

4. 分阶段课程表学习与多任务训练

- 采用课程表（curriculum learning）分阶段训练策略，从易到难逐步增加样本复杂度与任务难度，提升模型收敛速度与稳定性，提高噪音环境、长音频的识别准确率。
- 实施 multitask 多任务联合训练，促使模型在边界检测、时序对齐与后处理上具备一体化能力。

5. 强化学习技术

- 通过强化学习技术，增强标注模型对于原始长音频、含噪音频的解码能力，增强对关键词的识别能力。

6. 中英混识别能力

- 通过自研语音合成大模型生成大量领域的中英混音频数据，用于训练语音识别模型，增强 PC 领域频繁出现的中英混识别问题。

7. 全链路工程化后处理能力

- 内置 VAD、自动标点恢复、ITN（文本归一化：数字/单位/字母/金额等）、以及字级/句级时间戳输出，支持字幕生成、质检索引与精确对齐等下游任务。
- 支持热词注入与偏置、以及垂直语言模型在线/离线融合，便于落地快速适配客户专用词表与业务。

8. 推理优化与高吞吐低延迟

- 针对摩尔线程 MUSA 推理栈做算子融合、显存调度与流水线并发优化，显著降低单样本延时与并发下的 RTF。

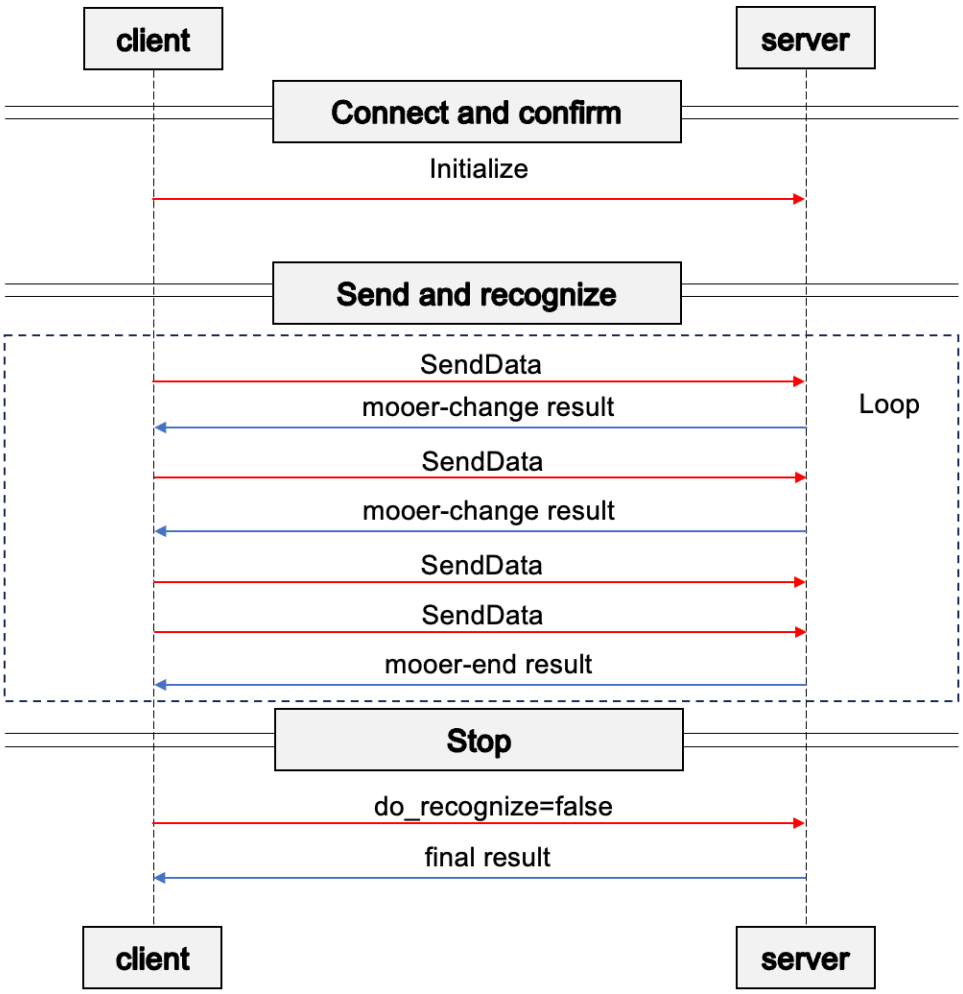
9. 评测与行业基线

- 在电话客服等行业数据集上使用 CER（字错误率）、NIST（National Institute of Standards and Technology） 等标准评测指标进行衡量，并保持行业领先的准确率与稳定性，适配企业级质检与合规要求。

(三) 接口协议

1. 流式语音识别接口

1) 交互流程图



2) 接口清单

服务名	url

流式语音识别 Websocket 接口	ws://${host}:${port}

3）API 调用说明

流式初始化消息

字段名	类型	说明	可选值	必填	示例
mooer-type	str	识别模式	mooer-change	是	mooer-change
sample_rate	int	采样率	16000, 8000	是	16000
wav_name	str	自定义的输入名，仅作请求标识		否	unique_wav_name
do_recognize	bool	开始发送音频的标志	true	是	true
itn	bool	是否打开 itn	true / false	否	true
punc	bool	是否打开标点	true / false	否	true

流式发送音频

按照 0.1s 一包发送 pcm binary 数据

4）流式结束发送消息

字段名	类型	说明	可选值	必填	示例
do_recognize	bool	结束发送音频的标志	false	是	false

5）流式接收片段结果

字段名	类型	说明	可能值	示例	备注
finished_recognize	bool	是否完成识别	false	false	
mooer-type	str	识别结果类型	mooer-change	mooer-change	当前句子的识别结果，用于实时显示和上屏（非最终结果），会实时改变
text	str	识别的片段结果		这是前半句话	
wav_name	str	与初始化的wav_name		unique_wav_name	

		对应，仅作请求标识			

6）流式接收断句结果

字段名	类型	说明	可能值	示例	备注
finished_recognize	bool	是否完成识别，如果是 true 表示所有识别已完成，可以断开连接	false / true	false	整条完整音频完成识别
mooer-type	str	识别结果类型	mooer-end	mooer-end	当前句子经过判停之后的最终识别结果，不会再发生改变
text	str	识别的片段结果		这是前半句话	

wav_name	str	与初始化的wav_name对应，仅作请求标识		unique_wav_name	
word_time_stamps	str	每个文本片段的起止时间		[[19750,19330]]	
utt_time_stamp	List[StampSent]	每个短句的时间戳			

utt_time_stamp

字段名	类型	说明	可能值	示例
utt_start	int	开始时间		19750
utt_end	int	结束时间		19930
utt_text	str	识别的文本片段结果，用空格分割		这 是 前 半 句 话
utt_word_time_stamps	List[(int, int)]	每个文本片段的起止时间		[[19750, 19930], [19930, 20270]]

示例代码

```
import asyncio
import websockets
import json

audio_file = "audio_8k.pcm"
sample_rate = 8000
host = ""
port = ""
async def asr_client():
    uri = f"ws://{host}:{port}"
    try:
        async with websockets.connect(uri, ping_interval=10) as websocket:

            # 发送初始化消息
            init_message = {
                "mooer-type": "mooer-change",
                "sample_rate": sample_rate,
                "wav_format": "pcm",
                "do_recognize": True,
                "itn": True,
                "punc": True,
            }

            await websocket.send(json.dumps(init_message))
```

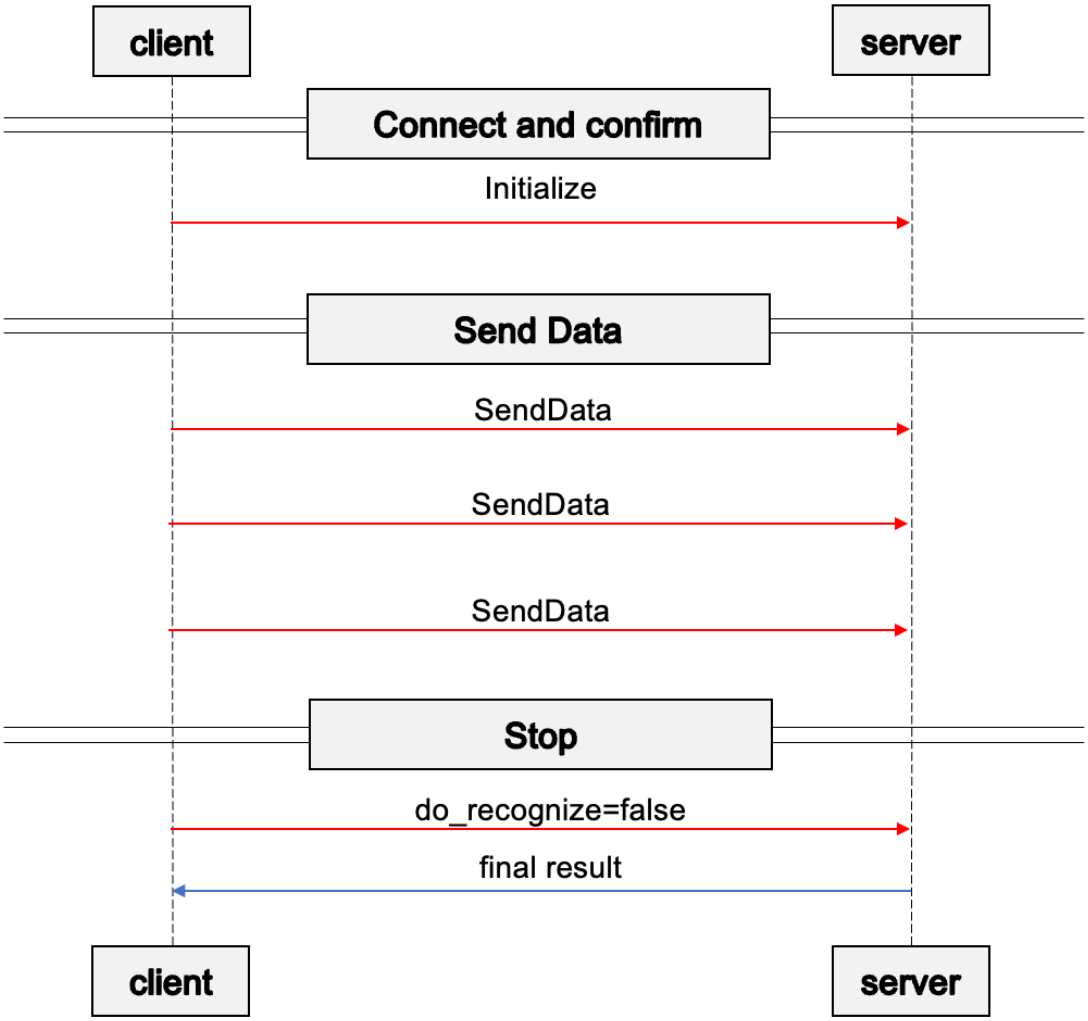
```
# 创建接收消息的任务
async def receive_messages():
    result_count = 0
    t = ""
    async for message in websocket:
        result_count += 1
        print(f"\n--- 消息 #{result_count} ---")
        print(f"原始消息: {message}")
        try:
            result = json.loads(message)
            mooer_type = result.get("mooer-type", "")
            if mooer_type == "mooer-change":
                t += result.get("text", "")
                print(f"单句中间结果: {t}")
            elif mooer_type == "mooer-end":
                t = result.get("text", "")
                print(f"单句最终结果: {t}")
                t = ""
            if result.get("finished_recognize", False):
                print("\n=== 识别完成 ===")
                return True
        except:
            pass
    return False
```

```
# 创建发送音频数据的任务
async def send_audio():
    """发送音频文件数据"""
    try:
        with open(audio_file, 'rb') as f:
            audio_data = f.read()


        chunk_size = sample_rate * 2 // 10
        total_chunks = len(audio_data) // chunk_size + (1 if len(audio_data) % chunk_size else 0)



        for i in range(0, len(audio_data), chunk_size):
            chunk = audio_data[i:i + chunk_size]
            await websocket.send(chunk)
            await asyncio.sleep(0.1)


    except Exception as e:
        print(f"发送音频文件时出错: {e}")


    # 发送结束消息
    end_message = {"do_recognize": False}
    await websocket.send(json.dumps(end_message))


# 并发执行发送和接收任务
```

```
            receive_task = asyncio.create_task(receive_messages())
            send_task = asyncio.create_task(send_audio())

            # 等待接收任务完成（识别完成）
            await receive_task
            # 取消发送任务（如果还在运行）
            send_task.cancel()

    except Exception as e:

        print(f"连接或处理过程中出错: {e}")

if __name__ == "__main__":
    asyncio.run(asr_client())
```

2. 离线语音转写接口

1）交互流程图



2）接口清单

服务名	url
非流式语音识别 Websocket 接口	ws://{host}:{port}

3）API 调用说明

非流式初始化消息

字段名	类型	说明	可选值	必填	示例
mooer-type	str	识别模式	offline	是	offline
sample_rate	int	采样率	16000, 8000	是	16000
wav_name	str	自定义的输入名，仅作请求标识		否	unique_wav_name
do_recognize	bool	开始发送音频的标志	true	是	true
itn	bool	是否打开 itn	true / false	否	true
punc	bool	是否打开标点	true / false	否	true

非流式发送音频

发送 pcm binary 数据

4）非流式结束发送消息

字段名	类型	说明	可选值	必填	示例

do_recognize	bool	结束发送音频的标志	false	是	false

5）非流式接收断句结果

字段名	类型	说明	可能值	示例
mooer-type	str	识别结果类型	offline	offline
text	str	识别的片段结果		这是前半句话
wav_name	str	与初始化的 wav_name 对应，仅作请求标识		unique_wav_name
word_time_stamps	str	每个文本片段的起止时间		[[19750,19330]]
utt_time_stamp	List[StampSent]	每个短句的时间戳		

utt_time_stamp

字段名	类型	说明	可能值	示例
utt_start	int	开始时间		19750
utt_end	int	结束时间		19930

utt_text	str	识别的文本片段结果，用空格分割		这 是 前 半 句 话
utt_word_time_stamps	List[(int, int)]	每个文本片段的起止时间		[[19750, 19930], [19930, 20270]]

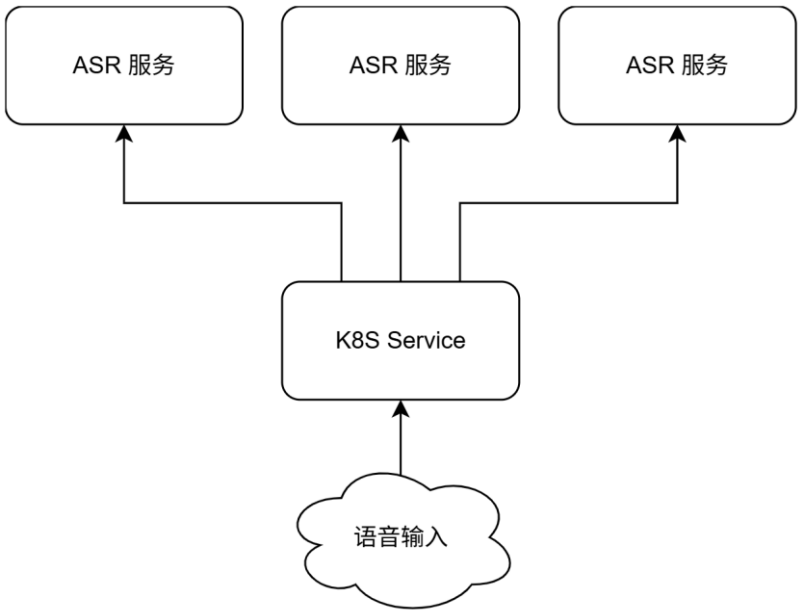
示例代码

```
import asyncio
import websockets
import json


audio_file = "audio_8k.pcm"
sample_rate = 8000
host = ""
port = ""
async def asr_client():
    uri = f"ws://{host}:{port}"
    async with websockets.connect(uri, subprotocols=["binary"]) as websocket:
        # 发送初始化消息
        init_msg = {
            "mooer-type": "offline",
            "sample_rate": sample_rate,
            "wav_format": "pcm",
```

```
            "wav_name": "test_audio",
            "do_recognize": True,
            "itn": False,
            "punc": True,
        }
        await websocket.send(json.dumps(init_msg))

        # 发送 PCM 音频数据（示例）
        with open(audio_file, "rb") as f:
            audio_data = f.read()
        await websocket.send(audio_data)
        await websocket.send(json.dumps({
          "do_recognize": False
        }))
        # 接收识别结果
        async for message in websocket:
            result = json.loads(message)
            print(result)
            break

asyncio.run(asr_client())
```

四、 系统部署

(一) 部署架构



(二) 配置要求

硬件	配置
GPU	MTT S4000
CPU	Intel(R) Xeon(R) Gold 6430
内存	1 TB
硬盘	15 TB

五、 公司介绍

关于摩尔线程

摩尔线程以全功能 GPU 为核心，致力于向全球提供加速计算的基础设施和一站式解决方案，为各行各业的数智化转型提供强大的 AI 计算支持。我们的目标是成为具备国际竞争力的 GPU 领军企业，为融合人工智能和数字孪生的数智世界打造先进的加速计算平台。我们的愿景是为美好世界加速。

关于紫光计算机

紫光计算机科技有限公司依托自主创新的研发实力和本土洞察，与产业链生态合作伙伴紧密协作，提供国产全栈式 AI 产业解决方案，赋能行业数字化转型与发展。作为“数字经济”建设的重要参与者，紫光计算机紧随国家经济发展方向，在“新智复合体”战略的引领下，现已拥有 PC 终端、智算一体机、智算中心部署实施、智算中心运维&算力租售四大核心业务板块，并逐步形成覆盖算力基础、行业应用、生产制造的立体架构，探索构建紫光计算机智能科技新生态。