喜马拉雅语音识别技术和应用介绍

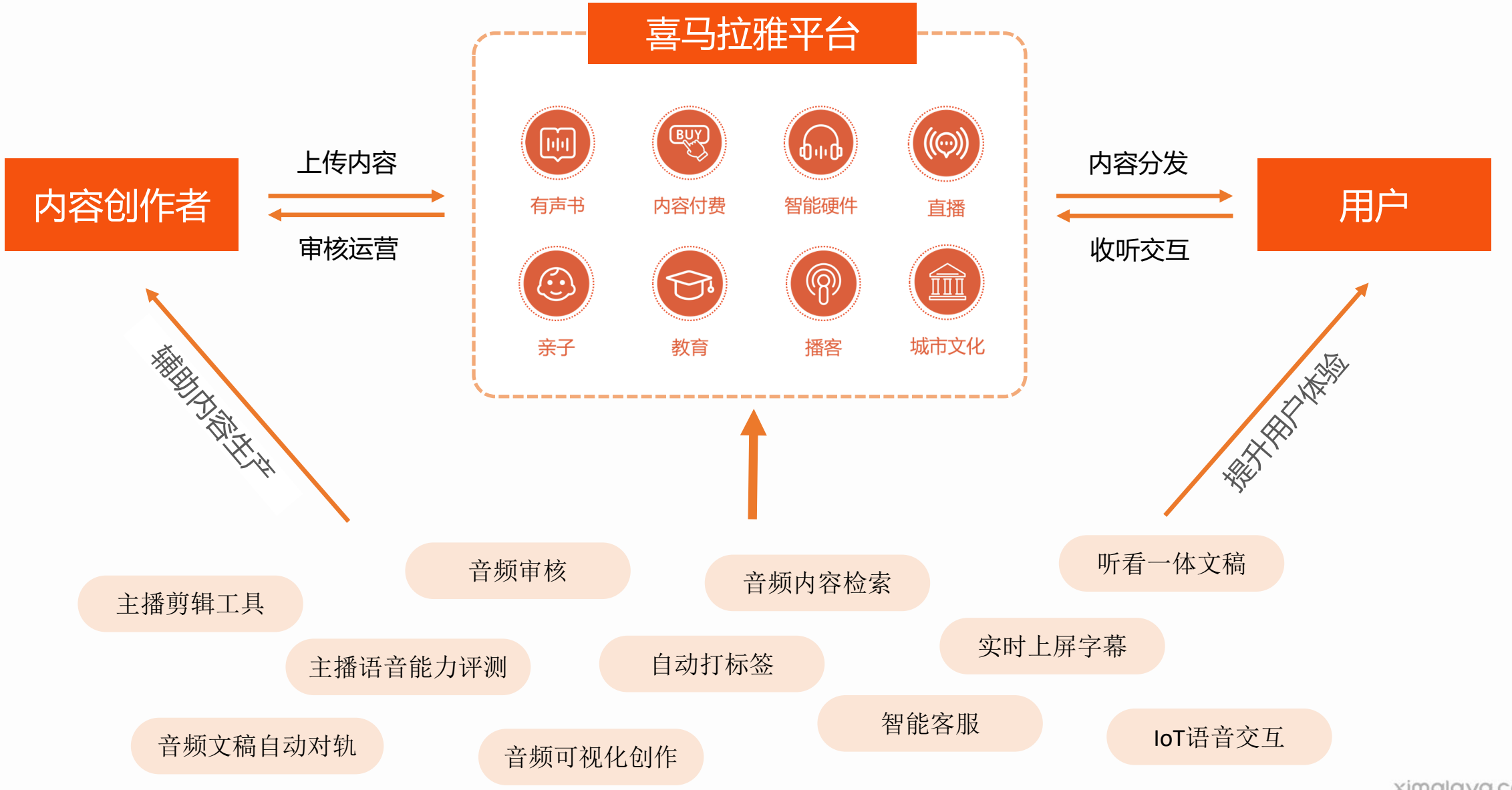
用声音分享人类智慧，用声音服务美好生活

- 01 | 喜马拉雅ASR应用场景
- 02 | HiASR算法引擎
- 03 | HiASR服务架构

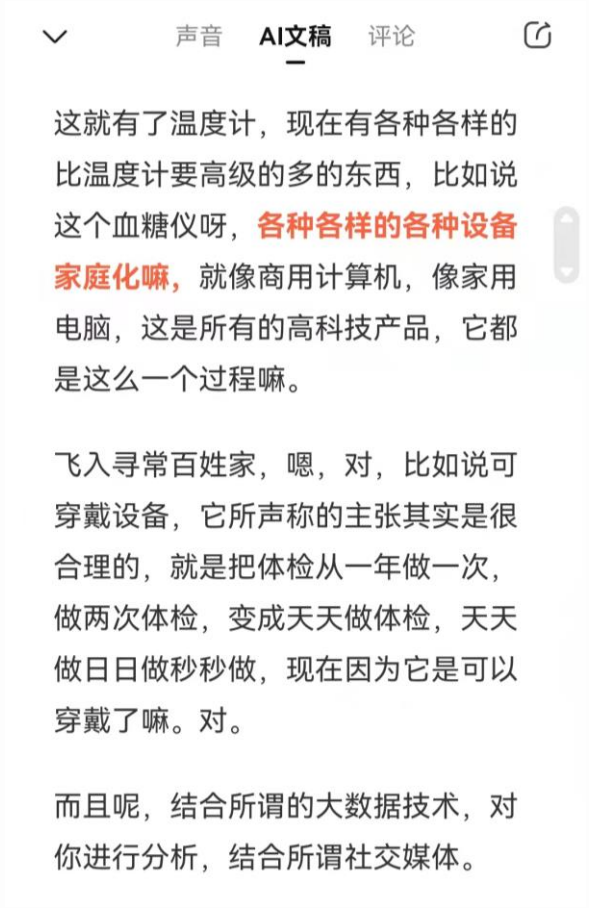
主讲人：印晶晶、吕翔

01 喜马拉雅ASR应用场景

喜马拉雅ASR应用场景



喜马拉雅ASR应用场景



听看一体文稿

音频可视化创作

主播语音能力评测

音频审核

喜马拉雅ASR应用场景

特点一：内容生态丰富

- 20+频道，100+细分品类，超2亿条声音
- 泛知识（金融、法律、历史、科技等）：专业性领域多
- 泛娱乐（影视、相声评书、脱口秀等）：背景音、风格差异大
- 信息类（头条、体育、热点资讯等）：时效性要求高
- 其他：英语、小语种、方言、戏曲、音乐等

→ 数据 + 算法

特点二：引擎需求多样

- 音频播客：离线长音频转写
- 直播/语音搜索：流式转写
- 智能客服：多说话人识别
- 智能硬件：语音交互、远场识别

特点三：调用链路复杂

- 识别的前处理、后处理
- 说话人分割与分离
- 语音评测、副语言信息
- 声音事件检测、文本检测

→ 服务架构

HiASR（Himalaya ASR）整体能力建设

业务应用

喜马拉雅APP | A+ | 奇迹文学 | 主播APP | 直播 | 审核 | 客服 | 小雅 | ……

服务能力

HAS（Himalaya Audio Service） | 微服务架构 | AI云平台 | CPU/GPU部署

算法能力

信号处理	长音频切分	E2E-ASR	Kaldi-ASR	后处理
• mp3/m4a/aac等音频解码 • 双通道转单通道 • 降噪或语音分离 • 特征提取、数据增强 • 副语言信息分析	• VAD（cnn/tdnn） • 说话人分割 • 音频事件检测	• LAS、CTC、Transformer、Conformer model • 中文模型、中英文模型 • 推理优化、剪枝、蒸馏 • 结合TLG	• Chain model • ngram领域语言模型 • NNLM rescore • 热词功能	• 自动加标点 • 段落划分 • ITN • 置信度 • 敏感词检测

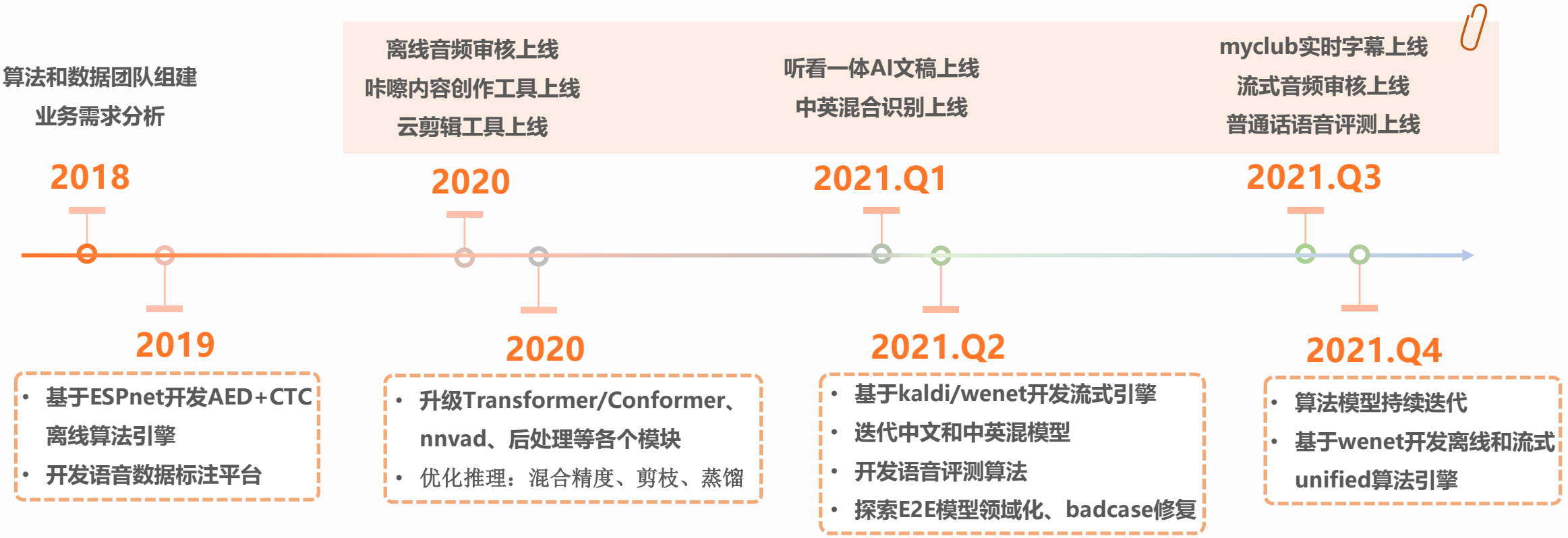
数据能力

喜马频道数据 | 业务场景数据 | 反馈数据 | 采购数据 | HiCrowd数据标注平台 | 错误数据反馈链路

02 HiASR 算法引擎

HiASR算法迭代路径

喜马拉雅以通用转写需求为主 非常适合采用E2E模型



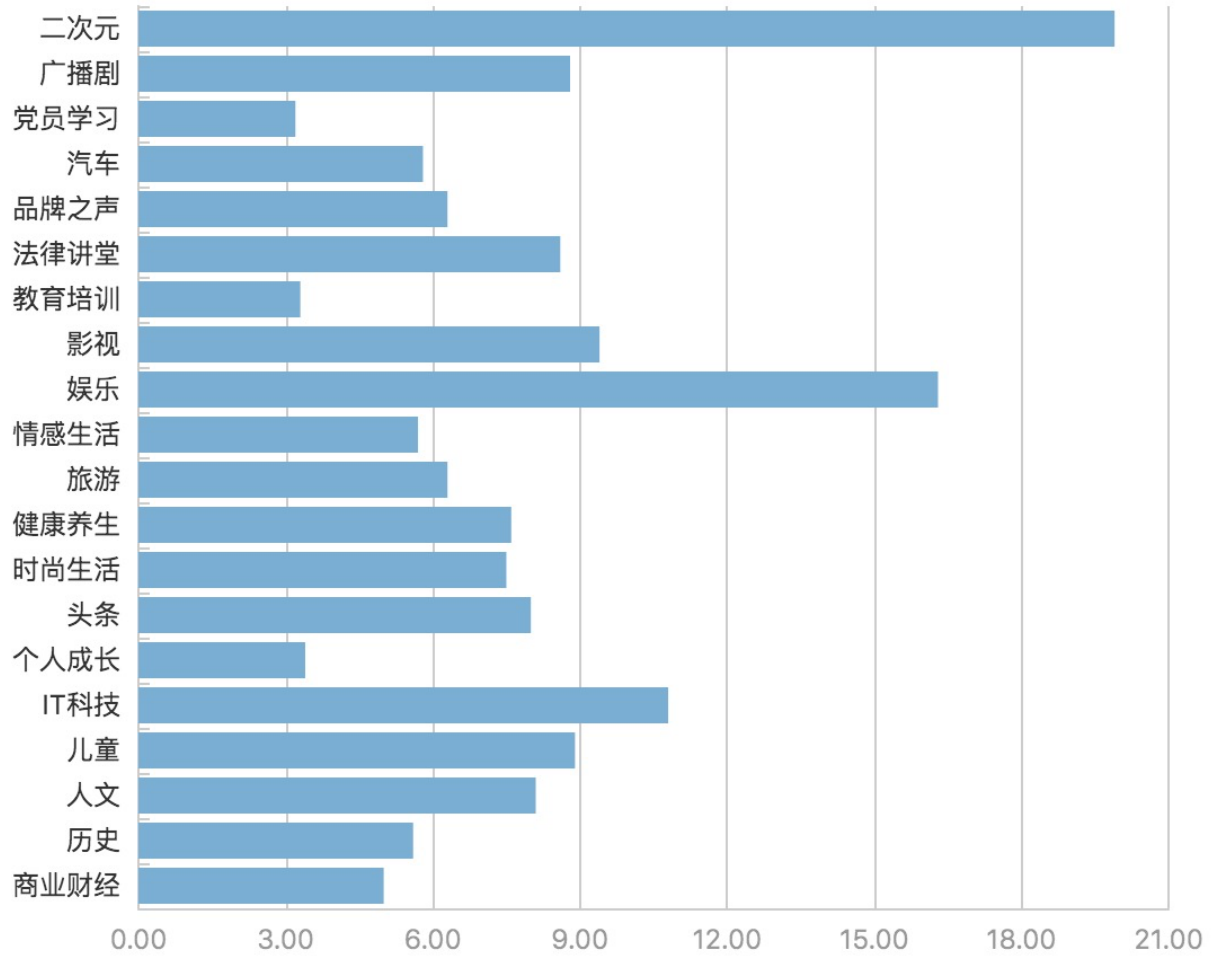
HiASR-E2E模型

数据情况

- 整体：按业务频道需求的优先级以及算法迭代的准确率挑选待处理长音频
- 预处理：音频切分、双模型预识别进行交叉筛选
- 训练数据：数万小时
- 测试数据：20+频道130多小时
- 专有名词、人名等特殊词汇，单独收集

整体识别率

- 大部分场景准确率>93%
- 部分复杂场景，例如较多背景音乐、太口语化、语速快、口音重、内容领域专业术语多，准确率可能较低，尤其直播娱乐类场景



HiASR-E2E模型

框架和模型选择

- 框架：尝试了ESPnet、fairseq和WeNet，前两个不支持流式识别，且需要自行开发推理引擎
- 模型：以AED自回归方案为主，识别率较高
- RNN_Attention->Transformer->Conformer
- 加入CTC进行联合训练可以有效提升识别率

训练策略

- 数据增强：变速、SpecAug、加噪声处理
- 模型迁移和增量训练：基于已经训好的大模型的中间checkpoint进行初始化
- fp16混合精度：可以提升训练和推理速度
- warmup、label smoothing、dropout等有助于收敛和防止过拟合

CER	cn7k model			某第三方
	rnn_att	transformer	conformer	
商业财经	8.1%	5.59%	5.08%	5.4%
历史	7.6%	5.94%	5.46%	6.2%
人文	9.7%	7.94%	7.54%	8.2%
儿童	11.2%	9.28%	8.51%	7.0%
IT科技	13.0%	11.11%	10.48%	8.9%
个人成长	4.9%	3.88%	3.91%	4.9%
头条	7.8%	7.90%	7.41%	6.1%
娱乐	19.9%	17.31%	16.57%	21.2%
教育培训	5.0%	3.51%	3.28%	3.3%
广播剧	10.2%	9.62%	8.20%	8.4%

HiASR-E2E模型

中英混合模型优化

- 混合建模单元：中文汉字、英文子词（BPE）
- 训练语料：大量两种单语数据 + 部分句内code-switching数据
- 采用bi-encoder：可以有效降低整体MER，并且中文CER和英文WER都会降低
  - encoder分别用单语模型初始化
  - 卷积层共享，encoder层部分拆分部分共享

cn15k + en8k model	中英混 MER (CER/WER)
baseline transformer	14.8 (10.49/33.58)
+ cnen0.8k	12.4 (9.18/26.15)
+ cnen0.9k + cn3k	11.0 (8.31/20.93)
+ biencoder	10.4 (7.64/19.86)

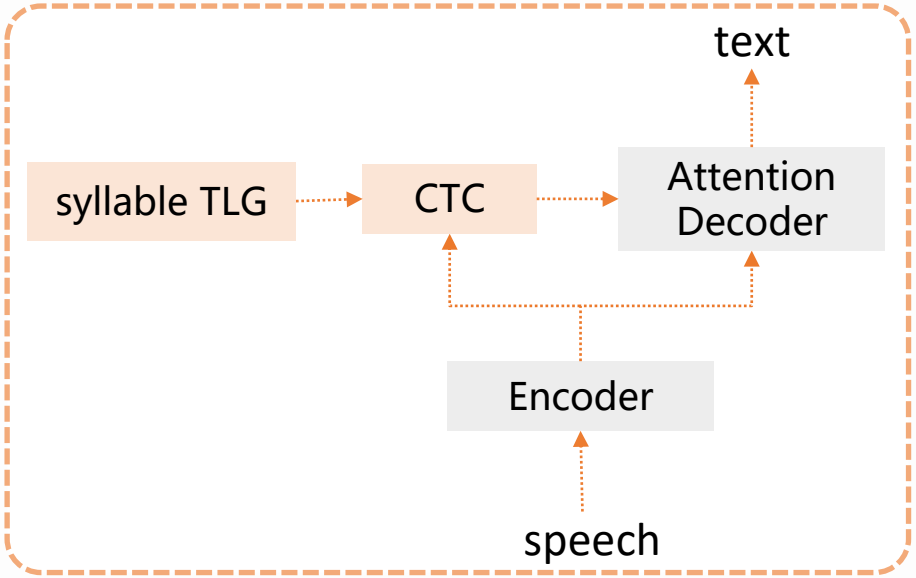
HiASR-E2E模型

模型领域化

- shallow fusion：加入语言模型的方式，小数据集上可以获得一定的效果，在大数据量上效果不明显
- 语音合成+语音转换：可用于修复badcase
- 探索syllable TLG：

  - 基于multi-cn数据集合进行实验验证， syllable TLG相比char TLG获得CER相对10.7%的提升

  - syllable TLG在喜马测试集进行诗词类数据的领域化，也获得了相对11.6%的提升



multi-cn	aishell	aidatatang	magicdata	thchs	avg
conformer	4.85%	4.75%	3.08%	13.90%	6.65%
+ char TLG	4.26%	3.91%	2.44%	14.43%	6.26%
+ syllable TLG	4.22%	3.72%	2.31%	12.12%	5.59%

HiASR-推理引擎

模型压缩

- 模型剪枝：利用剪枝削减模型的参数量，减少显存占用和推理耗时，其中CNN部分效果较好
- 模型蒸馏：用conformer蒸馏transformer，最终达到与conformer接近的性能，而推理速度更快，但训练时间也会比较长

$$L = \alpha \cdot CE(y,p) + \beta \cdot KL(p,q) \qquad q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

cn7k model	params	aishell_test	cctv_news
conformer	93M	3.72%	1.62%
transformer	49M	3.94%	1.72%
probs_t2_alpha0.5	49M	5.38%	2.44%
probs_t2_alpha0.8	49M	5.36%	2.47%
probs_t1_alpha0.5	49M	3.77%	1.62%

推理优化

- 相对阈值裁剪：进入下个step时，先对新路径做相对裁剪，再全局beam裁剪
- 动态batchsize：多个长音频分句后，根据句子时长排序重新组batch，充分利用显存
- 单GPU多进程部署：提升显卡利用率
- python/C++混合开发：line_profile工具分析，将推理中耗时较多的模块用C++改写
- 基于TensorRT/CUDA：模型网络改写

离线 rtf	fbank	vad	transf.	单进程 all	多进程 all
2080ti	2332	550	783	290	>400
3090	2636	763	882	360	>1000

HiASR-推理引擎

原始方案

E2E离线引擎

- transformer model，识别率高
- 无法支持流式识别
- GPU推理：强调高吞吐量
- 优化后2080ti rtf>400，3090 rtf>1000
- 少量GPU服务器一天可处理几十万小时
- 热词和领域化需要自己开发

kaldi流式引擎

- chain model，支持流式识别
- 可以额外训练超大规模语言模型
- 方便进行热词更新和领域化定制
- CPU推理：强调高并发量低延迟
- 实测单核并发>2路，平均尾包延迟0.014s

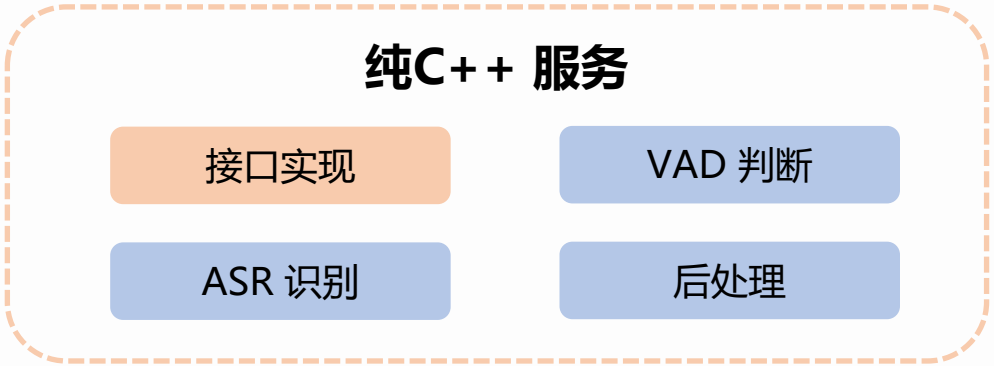
基于 WeNet Unified 方案

- unified流式和非流式，可以迭代一套模型复用到不同场景
- 支持多种解码方式，可以根据准确率或速度要求灵活配置
- 支持GPU推理和CPU推理，可以根据吞吐量或并发量需求进行部署
- 支持语言模型、热词功能、字级时间戳，可以实现功能的快速定制
- 具有完善的引擎和服务的框架，可以实现更加快速的工业化落地

cn15k model	中文 test1	中文 test2	中文 test3	avg
transformer (non-streaming)	6.81%	8.89%	7.96%	7.5%
unified_conformer (non-streaming)	6.52%	7.96%	7.42%	7.0%
unified_conformer (streaming)	7.10%	8.72%	8.09%	7.7%

03 HiASR 服务架构

典型流式语音识别服务架构



Pros

- ✓ 高效率
- ✓ 健壮性

Cons

- ✗ 应用库少
- ✗ 开发难度大

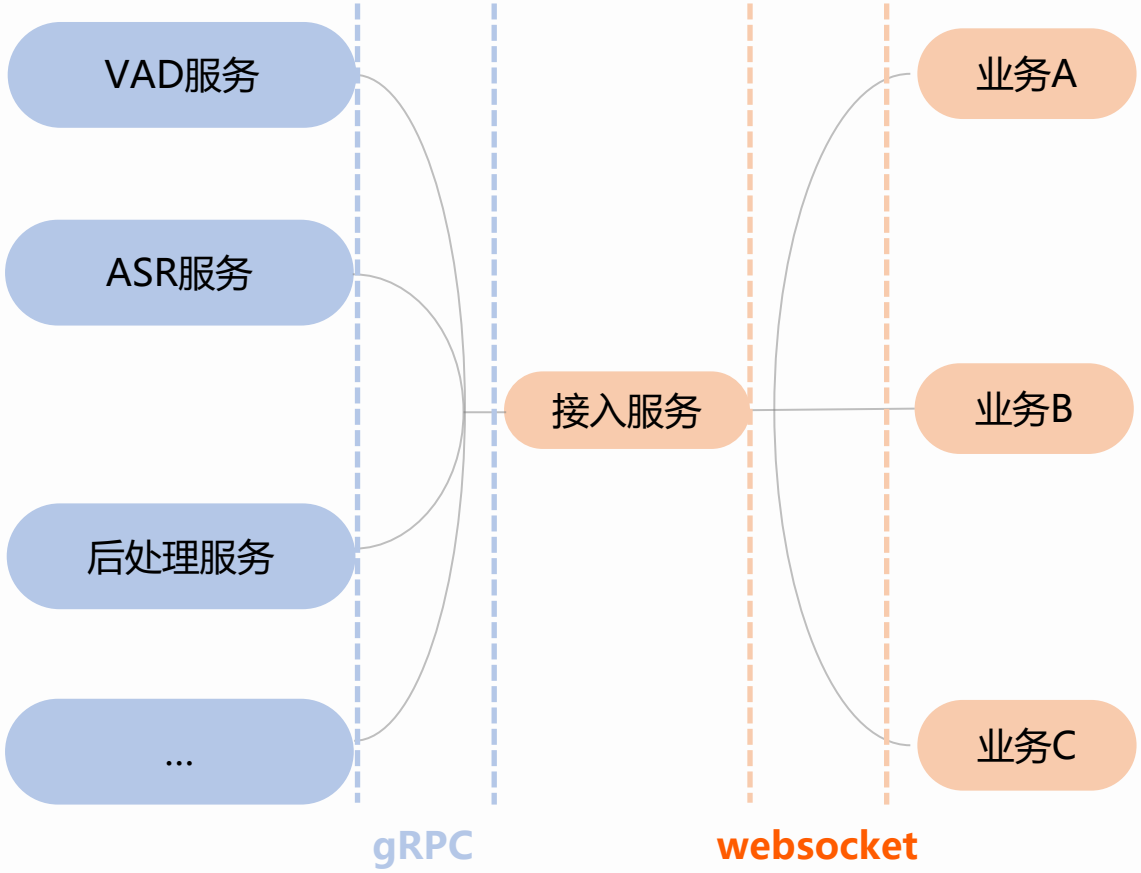
Pros

- ✓ 可用库丰富
- ✓ 开发难度/效率均衡

Cons

- ✗ JNI调用问题排查
- ✗ 应用库少
- ✗ 开发难度大

基于微服务/gRPC 的云原生架构




```
service ASR {
  rpc Recognize (stream Request) returns (stream Response) {}
}
```

```
message Request {
  message DecodeConfig {
    int32 nbest_config = 1;
    bool continuous_decoding_config = 2;
  }
  oneof RequestPayload {
    DecodeConfig decode_config = 1;
    bytes audio_data = 2;
  }
}
```

```
message Response {
  message OneBest {
    string sentence = 1;
    repeated OnePiece wordpieces = 2;
  }
  message OnePiece {
    string word = 1;
    int32 start = 2;
    int32 end = 3;
  }
  enum Status {
    ok = 0;
    failed = 1;
  }
  enum Type {
    server_ready = 0;
    partial_result = 1;
    final_result = 2;
    speech_end = 3;
  }
  Status status = 1;
  Type type = 2;
  repeated OneBest nbest = 3;
}
```

Pros

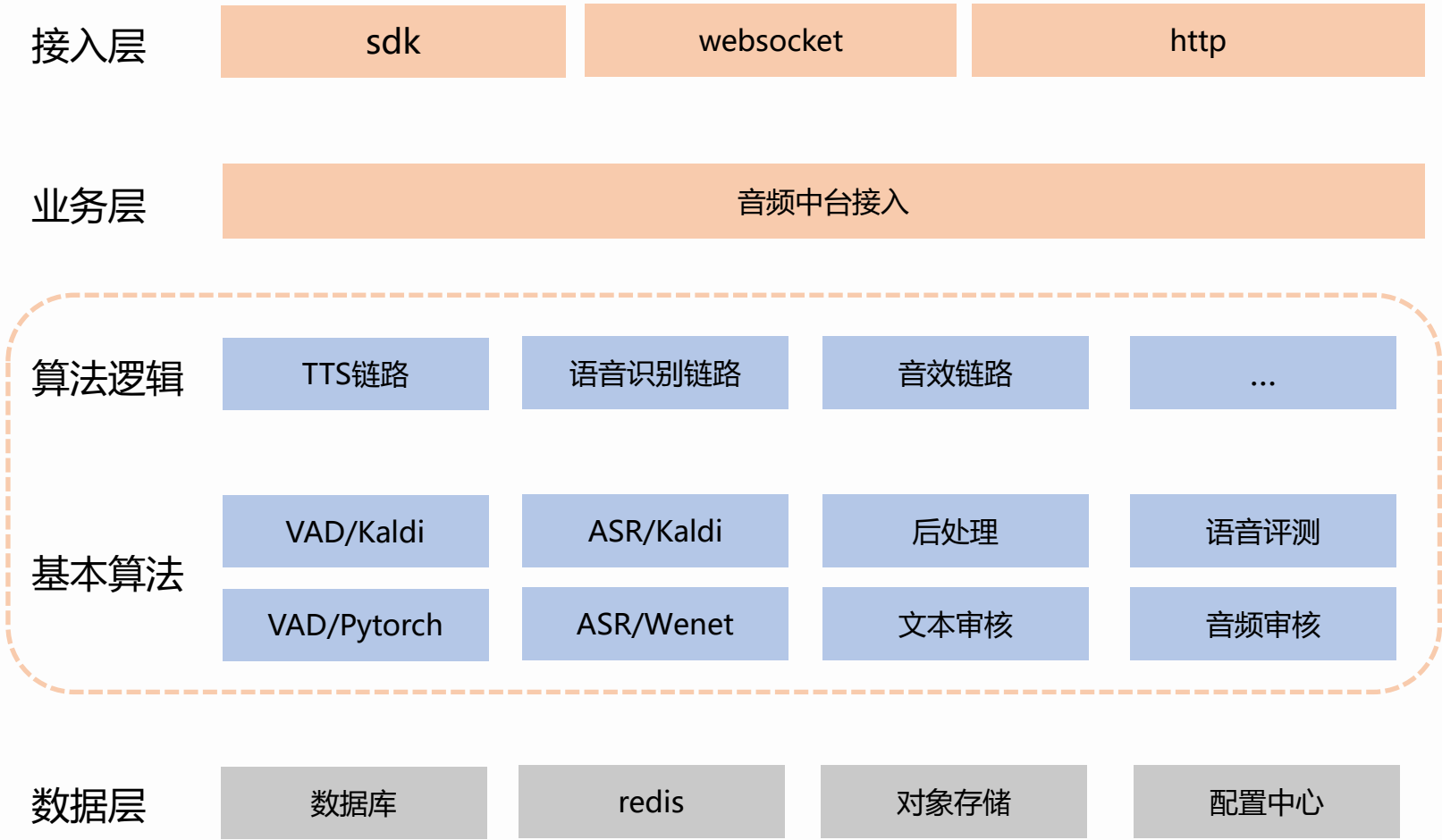
- ✓ 模块解耦，独立迭代
- ✓ 多路复用
- ✓ 双向流式
- ✓ 自带队列机制

Cons

- ✗ 联调难度增大

https://mp.weixin.qq.com/s/xPOp0y3VPYsdI7seXH9TXg

喜马拉雅流式云原生架构



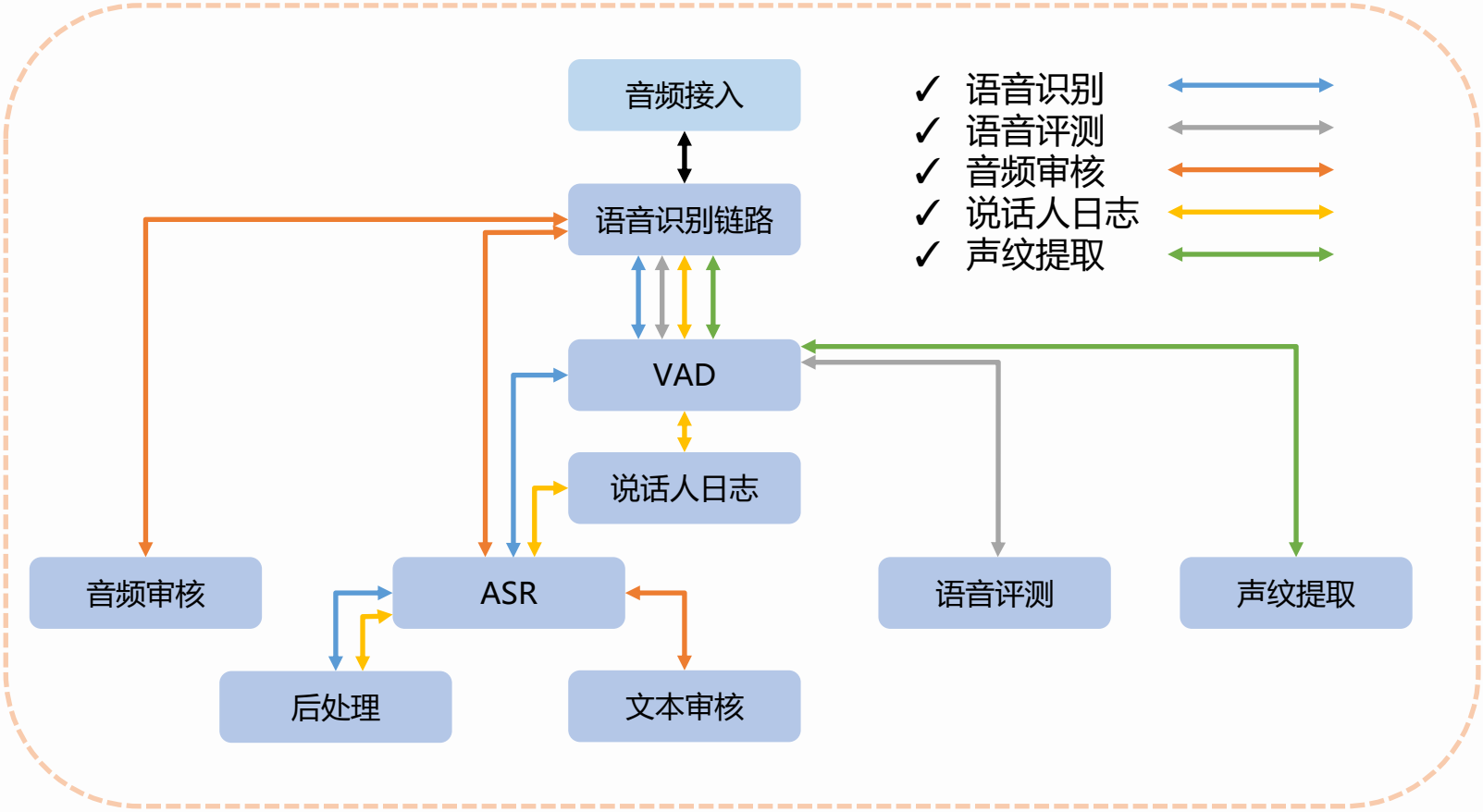
Pros

- ✓ 鉴权/流量控制
- ✓ 接口统一
- ✓ 算法独立迭代

Cons

- ✗ 联调难度增加
- ✗ 服务数量增加

喜马拉雅流式云原生架构



Wenet 算法服务具体实现

Wenet service

gRPC server

Proto request/response

client

```
==202275== 69,664 bytes in 1 blocks are possibly lost in loss record 85,117 of 85,126
==202275==    at 0x483C855: malloc (vg_replace_malloc.c:380)
==202275==    by 0xF34D0DC: mm_account_ptr_by_tid (in /root/miniconda3/lib/libmkl_core.so.1)
==202275==    by 0xF34C3D9: mkl_serv_malloc (in /root/miniconda3/lib/libmkl_core.so.1)
==202275==    by 0xD5EBA90: mkl_serv_domain_get_max_threads (in /root/miniconda3/lib/libmkl_g
==202275==    by 0x5A0EF10: at::init_num_threads() (in /opt/e2e-streamingASR/runtime/server/x
==202275==    by 0x61EB2A5: THFloatTensor_equalImpl(c10::TensorImpl*, c10::TensorImpl*) (in /
==202275==    by 0x61EB34F: THFloatTensor_equal(c10::TensorImpl*, c10::TensorImpl*) (in /opt/
==202275==    by 0x6088041: at::native::legacy::cpu::_th_equal(at::Tensor const&, at::Tensor
==202275==    by 0x7A4D972: torch::autograd::VariableType::equal(at::Tensor const&, at::Tenso
==202275==    by 0x8046555: torch::jit::(anonymous namespace)::tensorEqual(at::Tensor const&,
==202275==    by 0x804920B: torch::jit::(anonymous namespace)::attributesEqualCSE(torch::jit:
ch_cpu.so)
==202275==    by 0x8049D0E: torch::jit::EqualNode::operator()(torch::jit::Node const*, torch:
==202275==
==202275== LEAK SUMMARY:
==202275==    definitely lost: 0 bytes in 0 blocks
==202275==    indirectly lost: 0 bytes in 0 blocks
==202275==      possibly lost: 69,920 bytes in 2 blocks
==202275==    still reachable: 19,084,554 bytes in 132,331 blocks
==202275==         suppressed: 0 bytes in 0 blocks
==202275== Reachable blocks (those to which a pointer was found) are not shown.
==202275== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==202275==
==202275== For lists of detected and suppressed errors, rerun with: -s
==202275== ERROR SUMMARY: 6 errors from 6 contexts (suppressed: 0 from 0)
```

原始代码结果

```
==210829== 18,048 bytes in 47 blocks are possibly lost in loss record 72,120 of 72,370
==210829==    at 0x4841478: calloc (vg_replace_malloc.c:1117)
==210829==    by 0x40149CA: allocate_dtv (dl-tls.c:286)
==210829==    by 0x40149CA: _dl_allocate_tls (dl-tls.c:532)
==210829==    by 0x485E322: allocate_stack (allocatestack.c:622)
==210829==    by 0x485E322: pthread_create@@GLIBC_2.2.5 (pthread_create.c:660)
==210829==    by 0xAE6FDEA: ??? (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
==210829==    by 0xAE678E0: GOMP_parallel (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
==210829==    by 0x8BAB402: dnnl::impl::cpu::x64::jit_uni_reorder_t::execute(dnnl::impl::
==210829==    by 0x804B46E: dnnl_primitive::execute(dnnl::impl::exec_ctx_t&) const (in /o
==210829==    by 0x804B7E4: dnnl_primitive_execute (in /opt/e2e-streamingASR/runtime/serv
==210829==    by 0x5C0ED1D: dnnl::primitive::execute(dnnl::stream const&, std::unordered_
) const (in /opt/e2e-streamingASR/runtime/server/x86/fc_base/libtorch-src/lib/libtorch_cp
==210829==    by 0x5C0F487: ideep::tensor::reorder_if_differ_in(ideep::tensor::desc const
==210829==    by 0x5C14399: void ideep::convolution_forward::do_compute<true>(ideep::conv
pt/e2e-streamingASR/runtime/server/x86/fc_base/libtorch-src/lib/libtorch_cpu.so)
==210829==    by 0x5C0B3E1: at::native::_mkldnn_conv2d(ideep::tensor const&, ideep::tenso
 /opt/e2e-streamingASR/runtime/server/x86/fc_base/libtorch-src/lib/libtorch_cpu.so)
==210829==
==210829== LEAK SUMMARY:
==210829==    definitely lost: 0 bytes in 0 blocks
==210829==    indirectly lost: 0 bytes in 0 blocks
==210829==      possibly lost: 55,698 bytes in 86 blocks
==210829==    still reachable: 274,448,332 bytes in 409,934 blocks
==210829==                       of which reachable via heuristic:
==210829==                         newarray           : 1,088 bytes in 2 blocks
==210829==         suppressed: 0 bytes in 0 blocks
==210829== Reachable blocks (those to which a pointer was found) are not shown.
==210829== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==210829==
==210829== For lists of detected and suppressed errors, rerun with: -s
==210829== ERROR SUMMARY: 41 errors from 41 contexts (suppressed: 0 from 0)
```

使用**eigen**替换**mkl**编译结果

```
void TorchAsrDecoder::Rescoring() {
  // Do attention rescoring
  Timer timer;
  AttentionRescoring();
  LOG(INFO) << "Rescoring cost latency: " << timer.Elapsed() << "ms.";
  mkl_free_buffers();
}
```

```
==111961== LEAK SUMMARY:
==111961==    definitely lost: 0 bytes in 0 blocks
==111961==    indirectly lost: 0 bytes in 0 blocks
==111961==      possibly lost: 0 bytes in 0 blocks
==111961==    still reachable: 792,039 bytes in 12,527 blocks
==111961==         suppressed: 0 bytes in 0 blocks
```

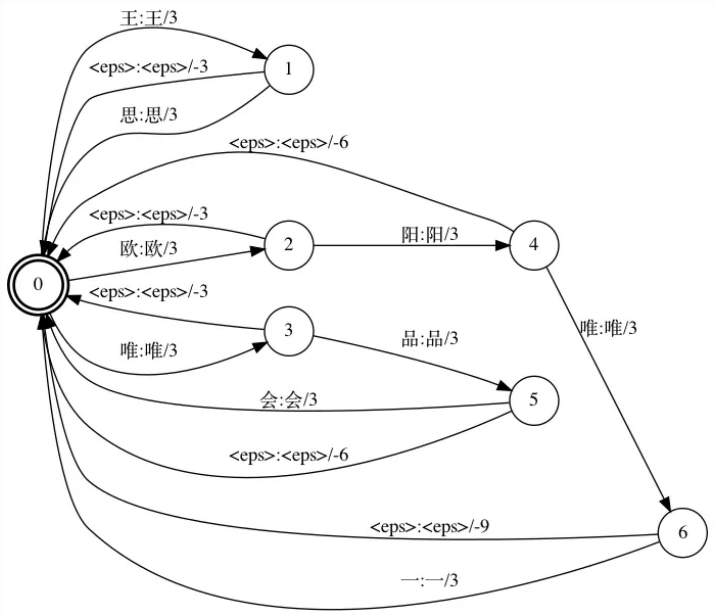
```
==28566== LEAK SUMMARY:
==28566==    definitely lost: 0 bytes in 0 blocks
==28566==    indirectly lost: 0 bytes in 0 blocks
==28566==      possibly lost: 3,840 bytes in 10 blocks
==28566==    still reachable: 303,244,294 bytes in 714,015 blocks
==28566==                       of which reachable via heuristic:
==28566==                         newarray           : 1,088 bytes in 2 blocks
==28566==         suppressed: 0 bytes in 0 blocks
```

每次识别结束释放缓存结果

https://github.com/wenet-e2e/wenet/blob/main/runtime/core/bin/grpc_server_main.cc

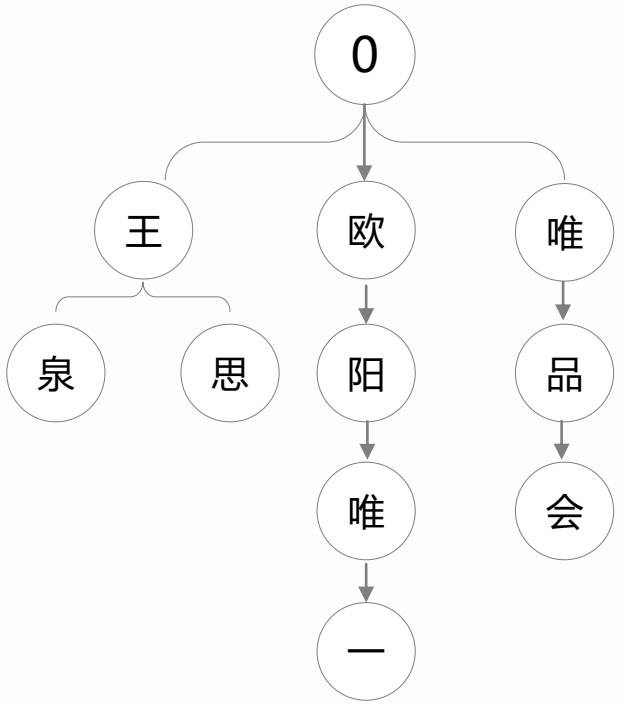
Wenet 算法服务具体实现

基于fst的热词方案




```
if (context_graph_) {
  if (arc.olabel == 0) {
    e_next->val->context_state = tok->context_state;
  } else {
    e_next->val->context_state = context_graph_->GetNextState(
      tok->context_state, arc.olabel, &context_score,
      &is_start_boundary, &is_end_boundary);
    graph_cost -= context_score;
  }
}
```

基于字典树的热词方案 （不依赖fst）




```
if (config_.trie != NULL)
{
  std::tie(next_trie_, history_hot_words_) = UpdateByHotword(tok, arc.olabel, tot_cost, ac_cost, graph_cost);
}
```

```
{"nbest":[{"sentence":"神智出现交易几乎停滞的情况"}],"status":"ok",'
```

https://mp.weixin.qq.com/s/d0rBp8F1Ln_wuW4vSpPdQA

Kaldi/Python 算法服务具体实现

Kaldi/Python service

gRPC server

Proto request/response

client

Kaldi VAD

后验概率阈值
有效语音帧
有效静音帧

Kaldi ASR

尾包延迟低于**wenet**
领域语言模型
领域热词

Kaldi 评测

限定解码空间
考虑跳读/漏读
考虑最优/次优**phone**区分性

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig/
LDLIBS += -Wl,--as-needed -L/usr/local/lib -lgrpc++_reflection \
`pkg-config --libs grpc++ libglog protobuf`
```

```
message Recognize_Request{
  string biz_id = 1;
  string session_id = 2;
  string config_json = 3;
  bytes audio_data = 4;
}

message Recognize_Response{
  string response_json = 1;
  int32 error_code = 2;
}
```

Python后处理

正则ITN
循环神经网络加标点

Python文本检测

构建敏感词库字典树，耗时**80**ms->**5**ms
bert蒸馏textcnn，速度提升**5**倍，精度保持**99%**

Python声纹

pytorch tdnn->onnx推理，速度提升**6**倍

Python说话人日志

cosine+sc+vb

优劣分析

经济学十大原理第一条

人们面临权衡取舍

无意于强调架构的优劣，合适的才是最好的

Pros

- ✓ 算法/代码，自由选择
- ✓ 离线/实时统一，业务逻辑层兼容处理
- ✓ 各个模块单独迭代，增加服务可靠性
- ✓ 提升工作效率

Cons

- ✗ 尾包延时增加，100ms以内尚可接受
- ✗ 机器利用效率降低
- ✗ 负载均衡处理，需要L7级负载均衡
- ✗ 无状态处理

Q & A