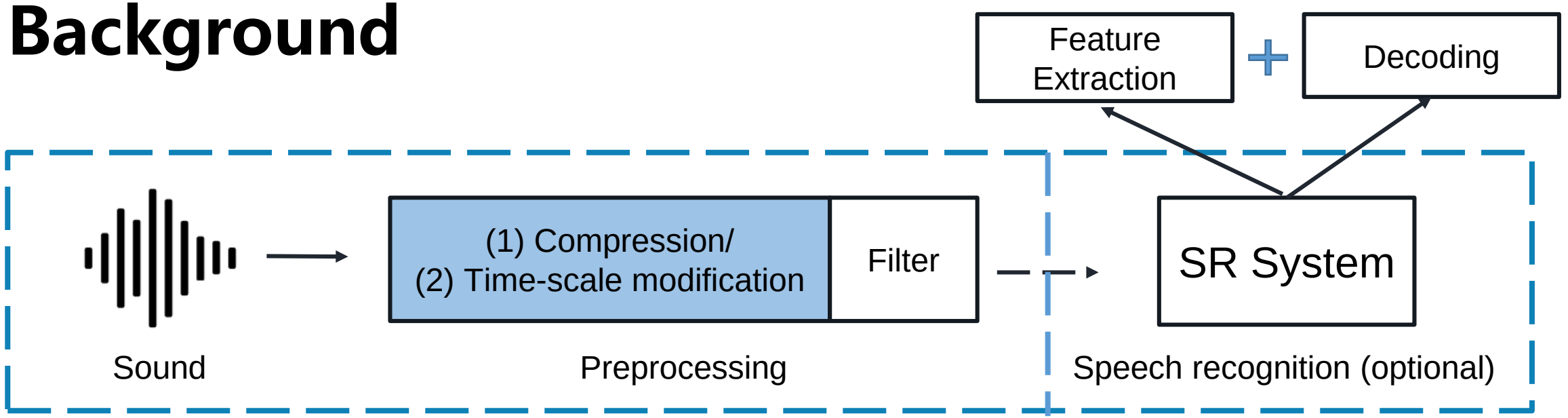
基于音频预处理过程漏洞的语音识别系统攻击

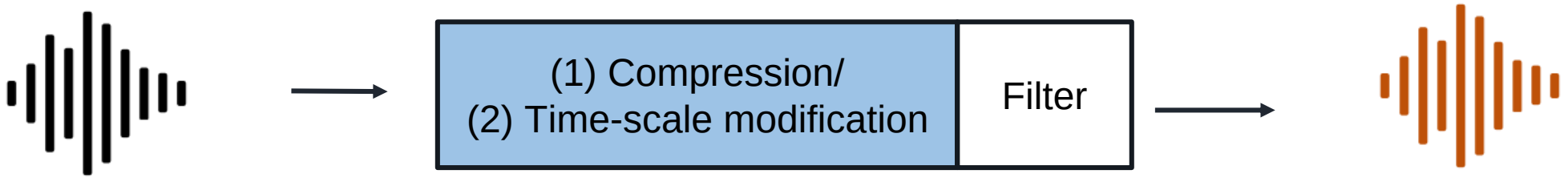
李超豪 蒋沁宏

Background



音频预处理过程

Feasibility Test – without SR



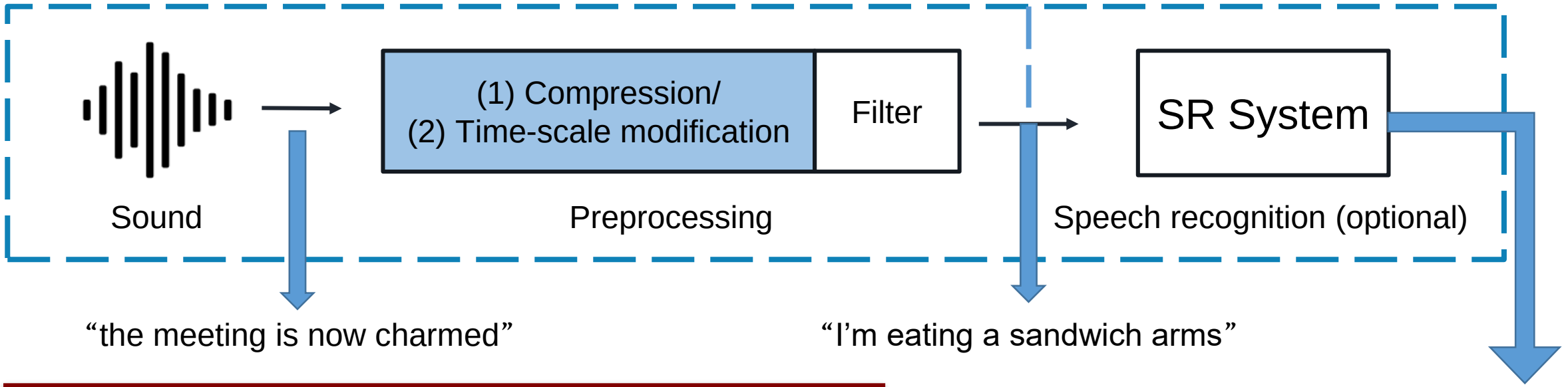
Preprocessing

“the meeting is now charmed”

“I’m eating a sandwich arms”

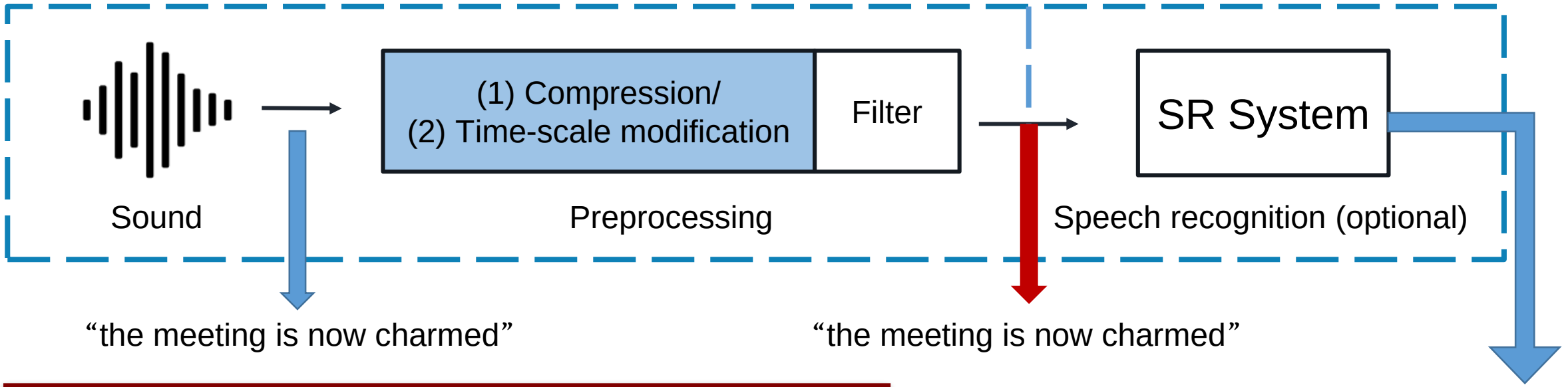
(1) Compression or (2) Time-scale modification

Advanced scenario 1 – with SR



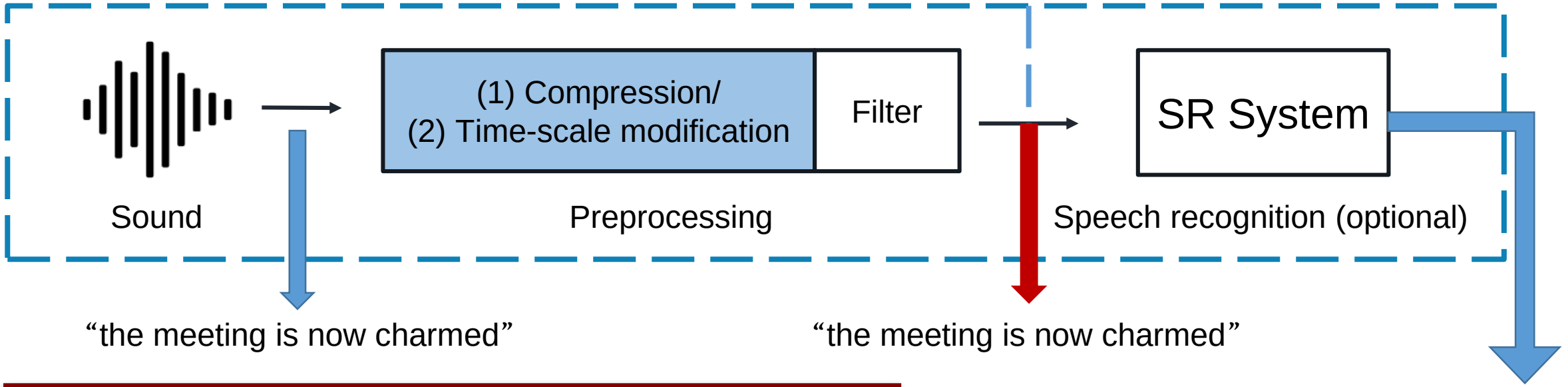
Human-perceived & SR-independent !!

Advanced scenario 2 – with SR (white box)



Human-unperceived & SR white box

Advanced scenario 3 – with SR (black box)



Human-unperceived & SR black box

Background

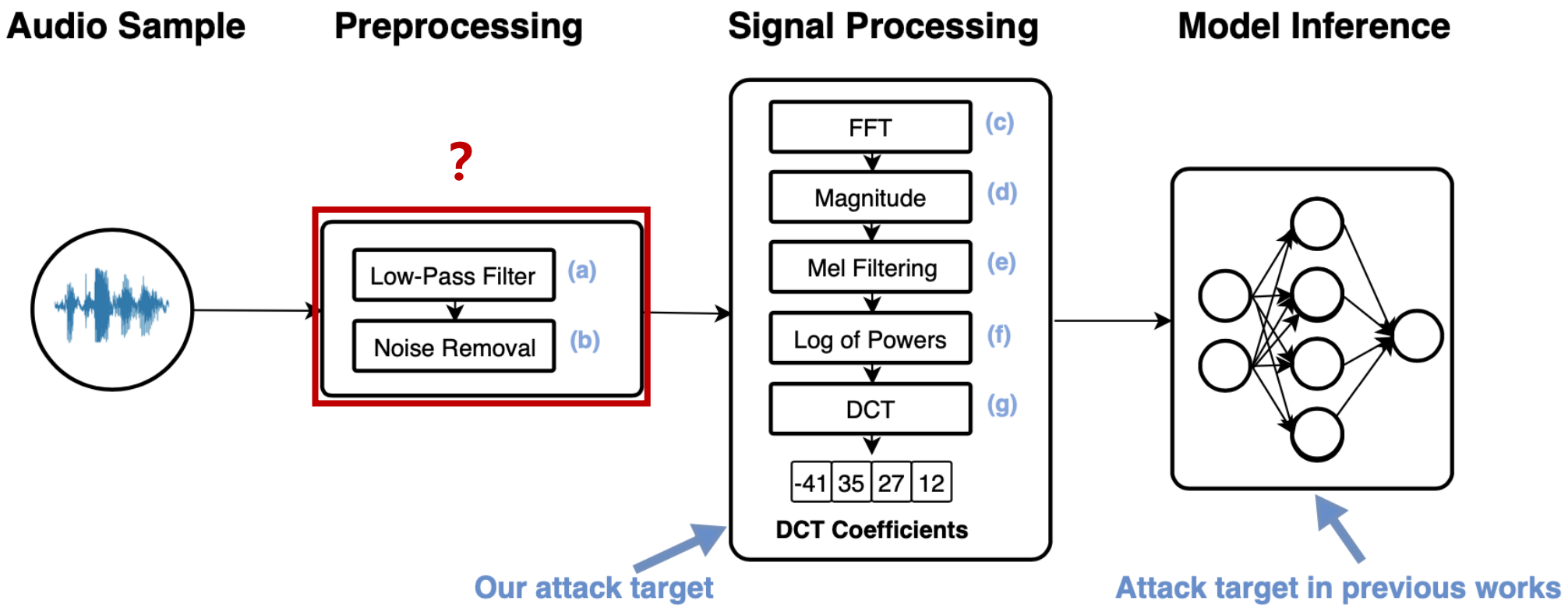
物理世界到ASR系统链路调研

- Practical Hidden Voice Attacks against Speech and Speaker Recognition Systems
- Qualcomm
- Google speech-to-text
- Nuance
- Devil’ s Whisper

音频从物理世界到云端识别系统的链路中几乎不采用压缩算法或个别压缩算法

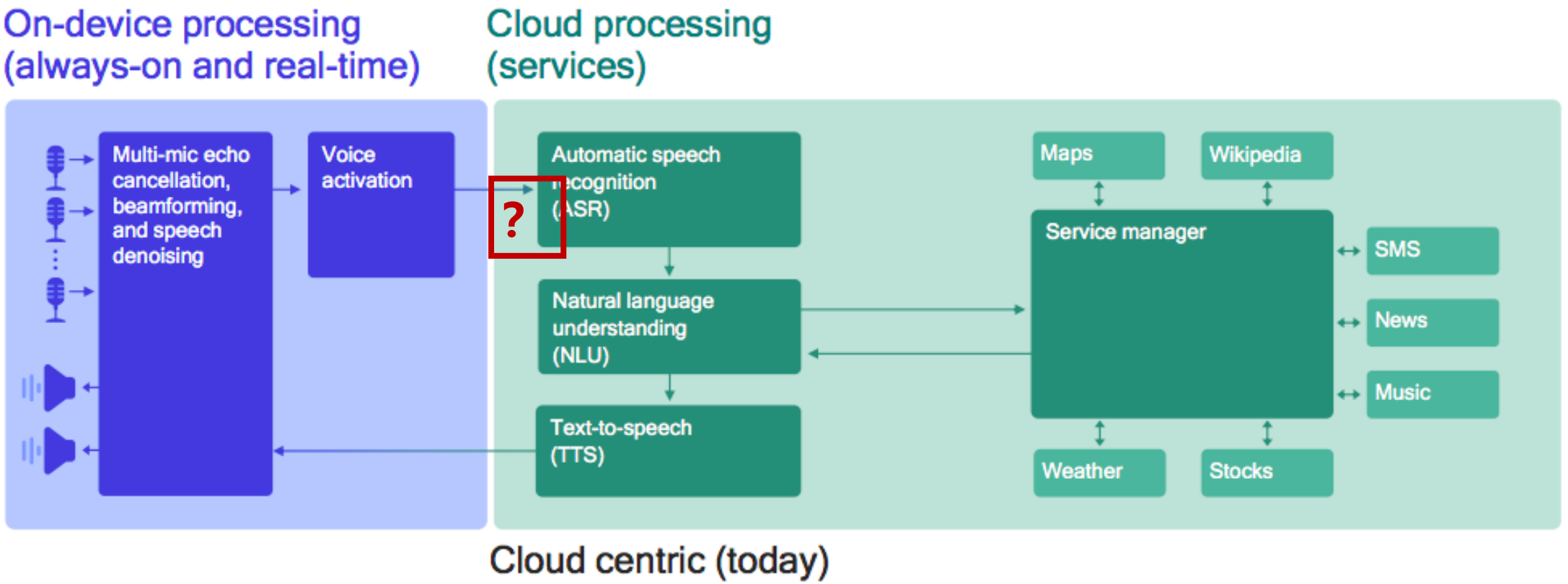
原因：不必要且会一定程度影响音频识别性能

物理世界到ASR系统链路调研 —— Practical



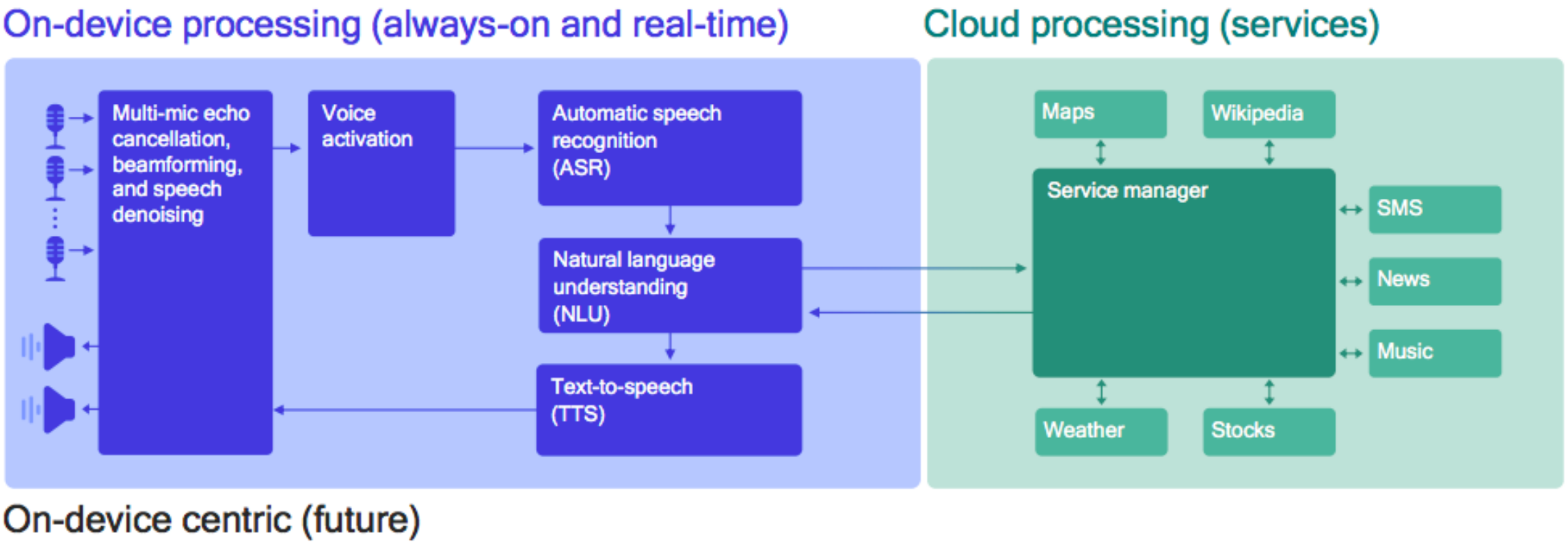
Link: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_08-1_Abdullah_paper.pdf

物理世界到ASR系统链路调研 —— Qualcomm



Link: https://www.qualcomm.com/media/documents/files/making-an-on-device-personal-assistant-a-reality.pdf

物理世界到ASR系统链路调研 —— Qualcomm



Link: https://www.qualcomm.com/media/documents/files/making-an-on-device-personal-assistant-a-reality.pdf

物理世界到ASR系统链路调研 —— Google STT

支持的音频编码

Speech-to-Text API 支持多种不同编码。下表列出了支持的音频编解码器：

编解码器	名称	无损	使用说明
MP3	MPEG 第三层音频	否	仅提供 Beta 版。如需了解详情，请参阅 RecognitionConfig 参考文档。
FLAC	免费无损音频编解码器	是	信息流要求使用 16 位或 24 位的位深
LINEAR16	线性 PCM	是	16 位线性脉冲编码调制 (PCM) 编码
MULAW	μ 律	否	8 位 PCM 编码
AMR	自适应多速率窄带	否	采样率必须为 8000 Hz
AMR_WB	自适应多速率宽带	否	采样率必须为 16000 Hz
OGG_OPUS	Ogg 容器中的 Opus 编码音频帧	否	采样率必须为 8000 Hz、12000 Hz、16000 Hz、24000 Hz 或 48000 Hz 之一
SPEEX_WITH_HEADER_BYTE	Speex 宽带	否	采样率必须为 16000 Hz

- **采样率限制**
- **OPUS\MP3\FLAC**
- **支持含有 LINEAR16 或 MULAW 编码音频的 WAV 文件**

Link：https://cloud.google.com/speech-to-text/docs/encoding

物理世界到ASR系统链路调研 —— Google STT

为达到最佳效果...	如果可能，请避免...
以 16000 Hz 或更高的采样率采集音频。	采样率较低可能会降低准确性。但是，应避免重新采样。例如，电话中的原生采样率通常为 8000 Hz，这也是应该发送到该服务的采样率。
使用无损编解码器录制和传输音频。建议使用 FLAC 或 LINEAR16。	在录制或传输过程中使用 mp3、mp4、m4a、mu-law、a-law 或其他有损编解码器可能会降低准确性。如果您的音频已经采用了不受此 API 支持的编码，请将其转码为无损 FLAC 或 LINEAR16。如果您的应用必须使用有损编解码器以节省带宽，我们建议使用 AMR_WB、OGG_OPUS 或 SPEEX_WITH_HEADER_BYTE 编解码器（排名分先后）。

Link：https://cloud.google.com/speech-to-text/docs/best-practices

物理世界到ASR系统链路调研 —— Naunce

Audio Chain Characteristics

Audio Chain Characteristic	Recommendation
Frequency Response	The front end frequency response must be flat within +/-4 dB over the range of 100 Hz to 8 kHz
Gain	Gain will be set once during initialization to a level just below clipping (2 bits headroom) when speaking with loud voice at normal distance from the microphone
Signal Resolution	16-bit A/D Converter is recommended to provide an effective dynamic range of 12 bits for the speech signal while accounting for the 2-bit headroom.
Transients	Duration of any transients shall not exceed 50 ms following command to start recording. Long-tailed transients shall settle to within noise floor limits within this period.
Low Frequency Roll-off	Maximum 6 dB attenuation at 200 Hz; minimum 24 dB attenuation 0-100 Hz
Anti-Aliasing	The attenuation of the anti-alias filter at the Nyquist frequency should be 20dB or better with a high order roll-off.
Sampling Rate	Recommended sampling rate is 16 kHz, 16-bit resolution yielding signal bandwidth of 250-7300 Hz
Compression	Several high-quality audio compression codecs are supported for Nuance Cloud Services. Please refer to the table below for recommendations.
Response Latency	Response time is a critical performance consideration in terms of user experience, and is affected by a number of variables including audio processing. For this reason, Nuance Cloud Services (NCS) is designed to process audio in real time while the user is speaking. Therefore, it required that audio is streamed to NCS in real-time. For example, 3 seconds of audio should be streamed over 3 clock seconds (1X real-time), additional latency results when audio is buffered and streamed significantly faster than real-time to catch up on streaming backlog.
Activation	Typically, start-of-speech is signaled by a manual event (button press) or wake-up word; end-of-speech is either automatically detected by CODEC/DSP or manually by a user event. (See below for additional information)

Link: https://developer.nuance.com/downloads/guidelines/Nuance%20Audio%20Input%20Specification_v11_ND.pdf

物理世界到ASR系统链路调研 —— Naunce

Nuance Developers APIs have different supported and recommended CODECs as shown in the table below.

Format	HTTP 1.0	SpeechKit	Web Sockets API
PCM 16k-16b—256 kbps	Supported	Supported	Supported
OPUS 16kHz CBR (complexity 10)	Not supported	**Recommended**	Not supported
PCM 8k-16b—128 kbps	Not supported	Supported	Not supported
SPEEX-WB—16kHz 27.8 kbps CBR	**Recommended**	Supported	**Recommended**
SPEEX NB—8kHz 24.6 kbps CBR	Supported	Supported	Supported
AMR-NB—12.2 kbps variable bit rate	Not supported	Not supported	Not supported
ADPCM—64 kbps	Not supported	Not supported	Not supported
G.711 (ulaw)—64 kbps	Not supported	Not supported	Not supported

Link: https://developer.nuance.com/downloads/guidelines/Nuance%20Audio%20Input%20Specification_v11_ND.pdf

物理世界到ASR系统链路调研 —— Devil

*Based on our experience, the changing of the **speech rate** and the **noise amplitude** is quite unique to different ASR systems, e.g., a specifically tuned audio might be decoded correctly with high confidence by one ASR system, but incorrectly by the other.*

不同ASR系统对于不同变速或噪声强度调制后的音频识别效果不同

压缩算法应用场景

-Sometimes, one may want to access to a speech recognition service from a very basic terminal which is just able to capture and transmit speech. In this case, the speech signal is transmitted from the client terminal to a distant speech recognition server. Coding of the speech signal is generally necessary to reduce transmission delays (GSM, MPEG or G7XX coding, depending on the client terminal used). *Figure 1* illustrates this client/server architecture. The problem with this architecture is that transcoding (the process of coding and decoding) modifies the spectral characteristics of the speech signal, so it is likely to have an influence on speech recognition performance.

减少传输时间（延迟），影响语音识别性能

Link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=962750

压缩算法应用场景

This is a totally different type of compression than what we have discussed so far in this book; with regard to audio, compression refers to a specific type of audio filter known as a compressor. Loud noises in a digital audio track can cause **distortion**, and likewise, quiet sounds, such as whispering, can be lost. An audio compressor can smooth out these issues in an audio track by acting as a dynamic range. By pulling down large spikes and lifting those quiet parts up, compression will ensure that the average loudness is fairly constant.

去除噪声

Link:https://books.google.com/books?id=Pa7pAwAAQBAJ&pg=PA134&lpg=PA134&dq=audio+preprocessing+compression+application&source=bl&ots=IeXGjwLNfo&sig=ACfU3U2HxovnslAT7OuZO44Aly0n_w-2VQ&hl=en&sa=X&ved=2ahUKEwin2_Hd5rnmAhXaXM0KHWAIC5gQ6AEwCHoECAoQAQ#v=onepage&q=audio%20preprocessing%20compression%20application&f=false

压缩算法应用场景

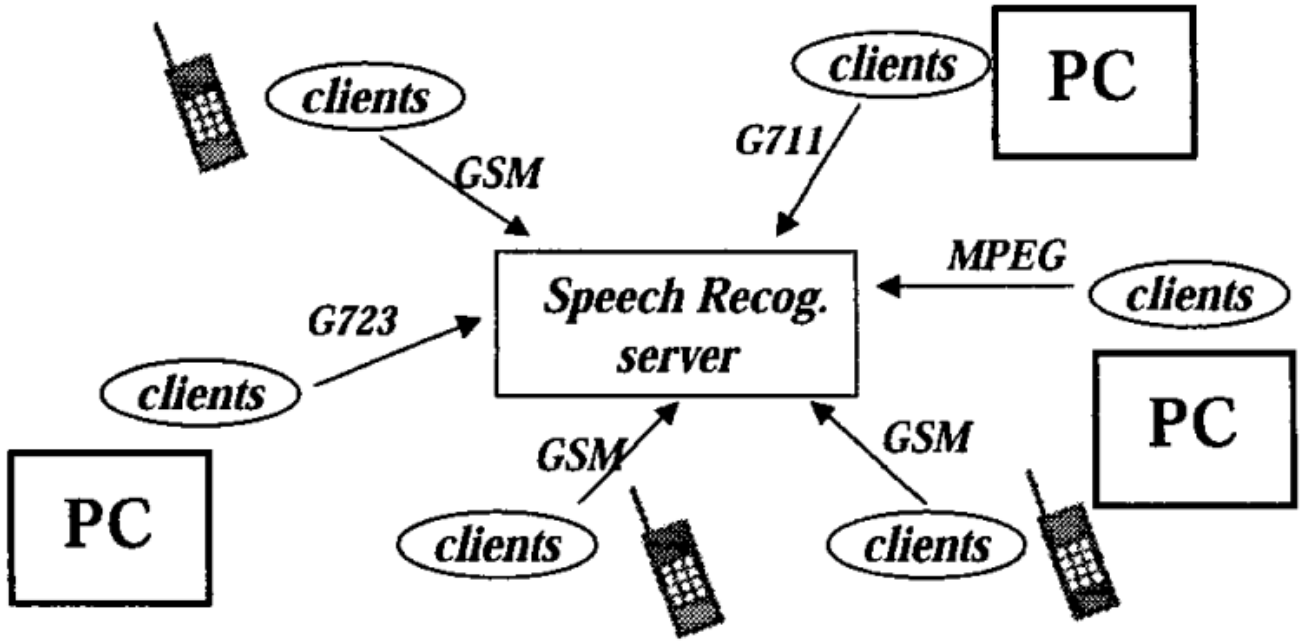


Figure 1 : speech recognition server accessed by different client terminals

各种常见编码方式

Link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=962750

压缩算法常见算法

Key Technology

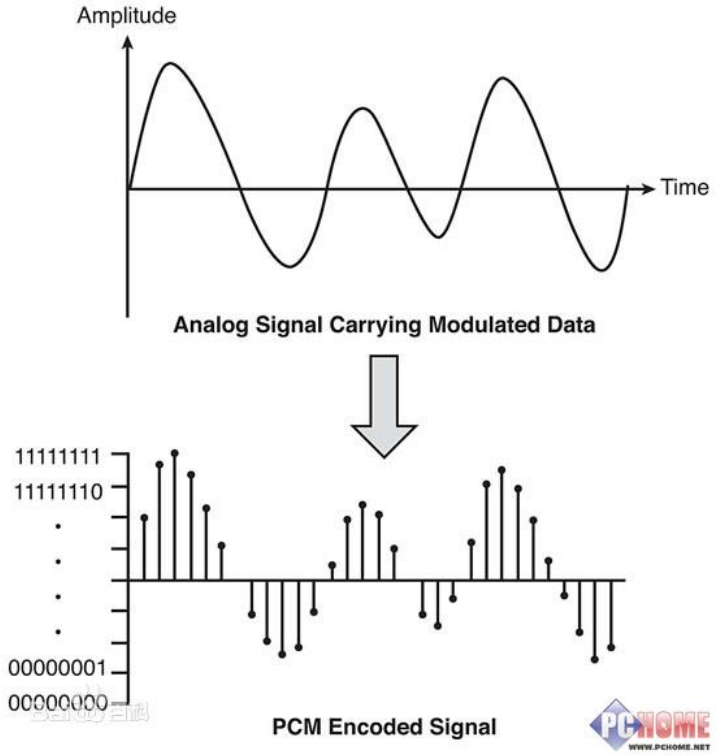
- GSM: Global System for Mobile Communications coder
- ADPCM编码
- LPC线性预测编码
- NICAM（Near Instantaneous Companded Audio Multiplex - 准瞬时压扩音频复用）

Key Technology — PCM系列

- PCM编码——脉冲编码调制
- DPCM编码——差分脉冲编码
- ADPCM编码——自适应差分脉冲编码（因子=差值的差值）
- 常用采样率为8KHz,16kHz,22.05kHz,32kHz,44.1kHz,48kHz,192kHz
- **算法复杂度低，压缩比小**（CD音质>400kbps），

  编解码延时最短（相对其它技术），声音质量一般

Q1：修改编码音频，使解码时出错?

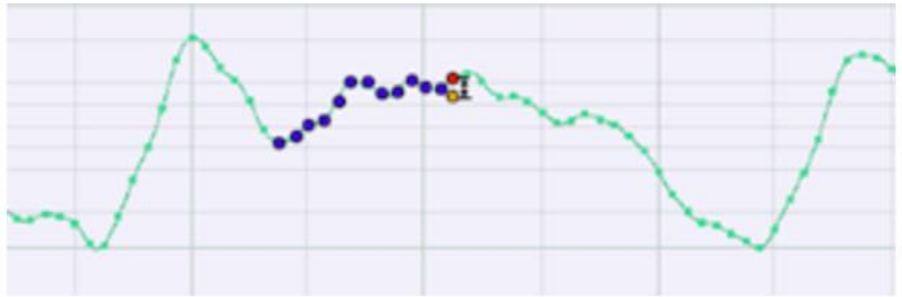


PCM编码

抽样 -> 量化-> 编码

Key Technology — LPC系列

- LPC：Linear Predictive Coding, 线性预测编码
- 一个语音的抽样能够用过去**若干个语音抽样**的线性组合来逼近。通过使实际语音抽样和线性预测抽样之间**差值的平方和达到最小**，能够决定**唯一**的一组预测系数。
- **压缩比大，计算量大，音质不高，廉价**



$$\tilde{s}(n)=\sum_{i=1}^{p}\alpha_i s(n-i) \qquad e(n)=s(n)-\tilde{s}(n)$$

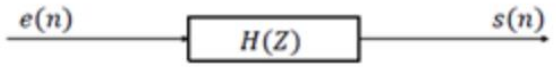
$$e(n)=s(n)-\sum_{i=1}^{p}\alpha_i s(n-i)$$

$$\Downarrow$$

$$e(Z)=S(Z)-\sum_{i=1}^{p}\alpha_i S(Z)Z^{-i}$$

$$\Downarrow$$

$$H(Z)=\frac{S(Z)}{e(Z)}=\frac{1}{1-\sum_{i=1}^{p}\alpha_i Z^{-i}}$$



Link: 1) https://blog.csdn.net/hhaowang/article/details/88776140
2) https://www.cnblogs.com/jizhiyuan/p/3748221.html

Key Technology — CELP

- CELP：Code Excited Linear Prediction,码激励线性预测编码
- 1）利用一个线性预测（LP）模型模拟声道
- 2）使用（自适应的和固定的）密码本条目作为LP模型的输入（激励）
- 3）在“感知加权域”执行闭合搜索

Key Technology - 时域压缩

- 针对音频PCM码流的样值进行处理，通过**静音检测**、**非线性量化**、**差分**等手段对码流进行压缩
- **算法复杂度低**，**声音质量一般**，**压缩比小**（CD音质> 400kbps），编解码**延时最短**（相对其它技术）
- G.711、ADPCM、LPC、CELP，以及发展起来的块压扩技术如NICAM、子带ADPCM（SB-ADPCM）技术

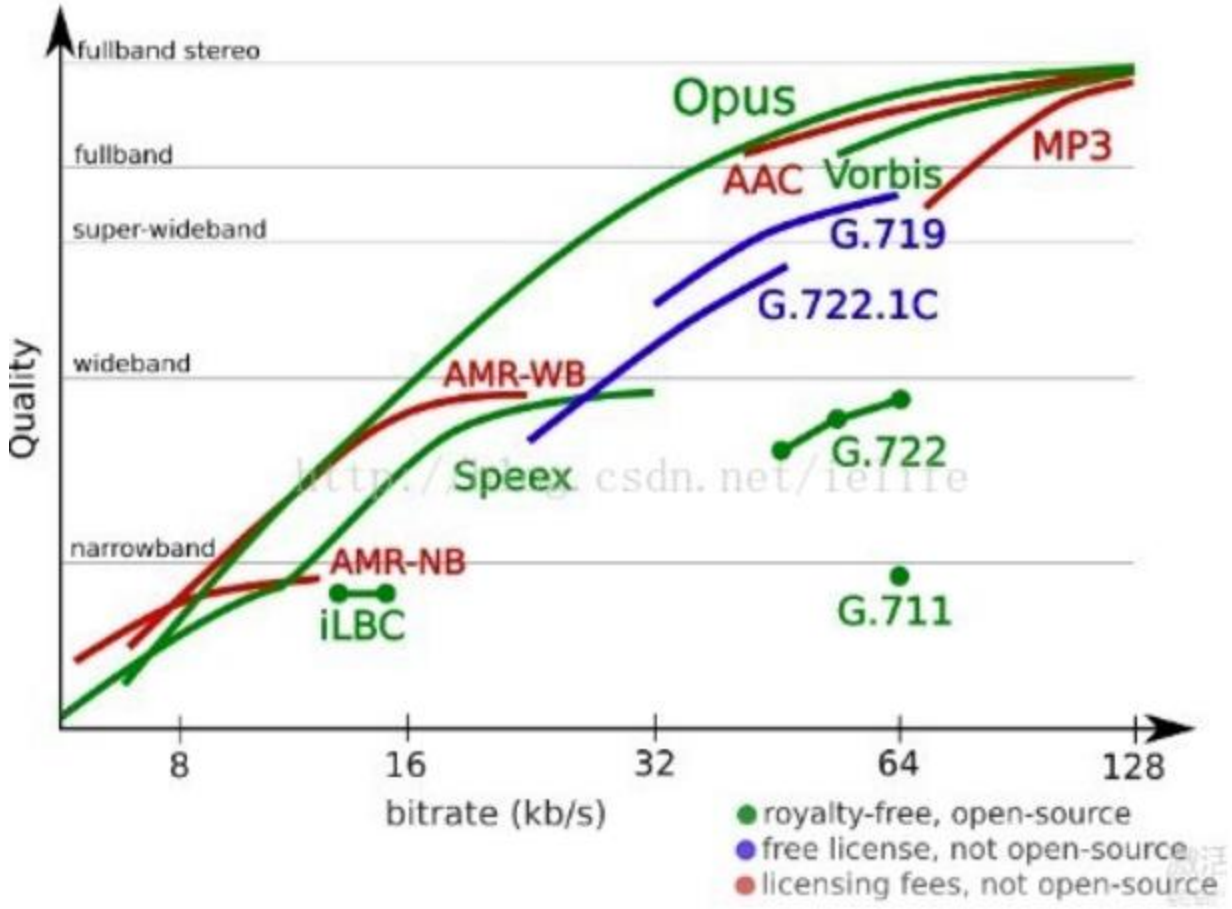
Key Technology – 子带压缩

- 将信号分解为若干**子频带**内的分量之和，然后对各子带分量根据其不同的分布特性采取**不同的压缩策略**以降低码
- 根据人对声音信号的**感知模型**（心理声学模型）
- 通过对**信号频谱的分析**来决定子带样值或频域样值的量化阶数和其它参数选择的
- 带编码的复杂度要略低于变换编码，编码延时也相对较短
- 感知型（Perceptual）压缩编码

AutoEncoder

- Machine learning
- https://ezgif.com/video-to-gif

Key Technology



Key Technology

- **Dynamic Range Compression**

This is a totally different type of compression than what we have discussed so far in this book; with regard to audio, compression refers to a specific type of audio filter known as a compressor. Loud noises in a digital audio track can cause **distortion**, and likewise, quiet sounds, such as whispering, can be lost. An audio compressor can smooth out these issues in an audio track by acting as a dynamic range. By pulling down large spikes and lifting those quiet parts up, compression will ensure that the average loudness is fairly constant.

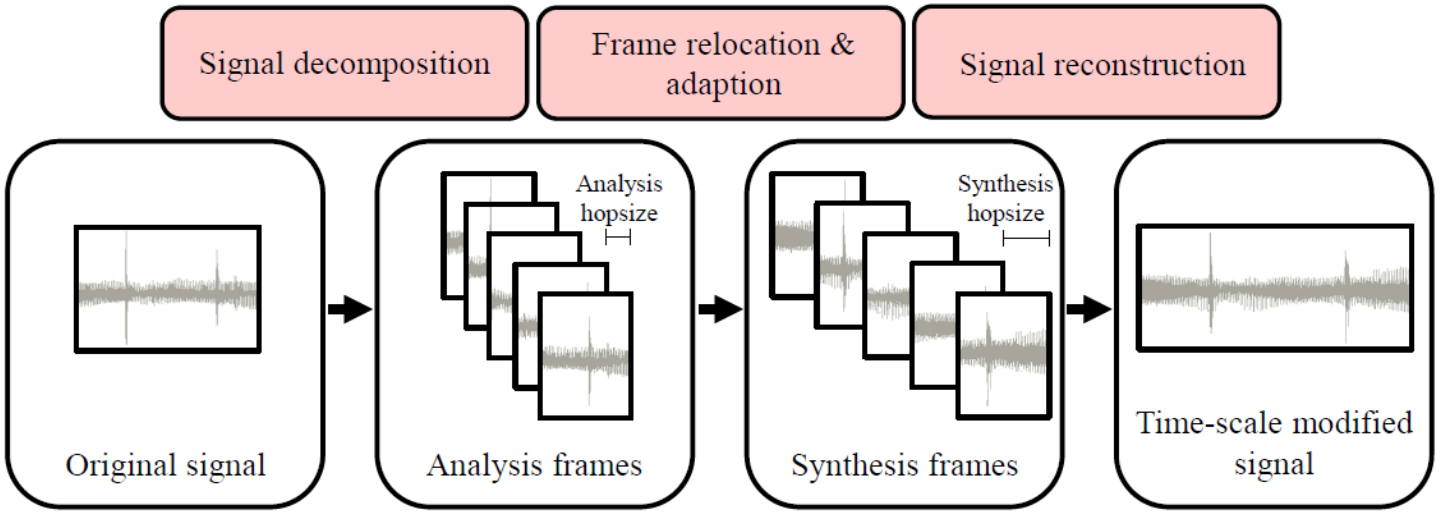
https://books.google.com/books?id=Pa7pAwAAQBAJ&pg=PA134&lpg=PA134&dq=audio+preprocessing+compression+application&source=bl&ots=IeXGjwLNfo&sig=ACfU3U2HxovnslAT7OuZO44Aly0n_w-2VQ&hl=en&sa=X&ved=2ahUKEwin2_Hd5rnmAhXaXM0KHWAIC5gQ6AEwCHoECAoQAQ#v=onepage&q=audio%20preprocessing%20compression%20application&f=false

高幅值压缩/低幅值压缩

倍速算法资料整理

Overview

- Reference：**A Review of Time-Scale Modification of Music Signals**
- **帮助理解：**https://zhuanlan.zhihu.com/p/110278983
- 1）按帧分解；2）重新定位；3）重新合成



Link：https://www.mdpi.com/2076-3417/6/2/57/pdf

Key Technology

- OLA (Overlap-Add)
- WSOLA (Waveform Similarity-based Overlap-Add)
- Phase Vocoder

OLA

- 叠加算法
- $\partial = H_s/H_\partial$
- Ha：analysis hopsize
- Hs：synthesis hopsize

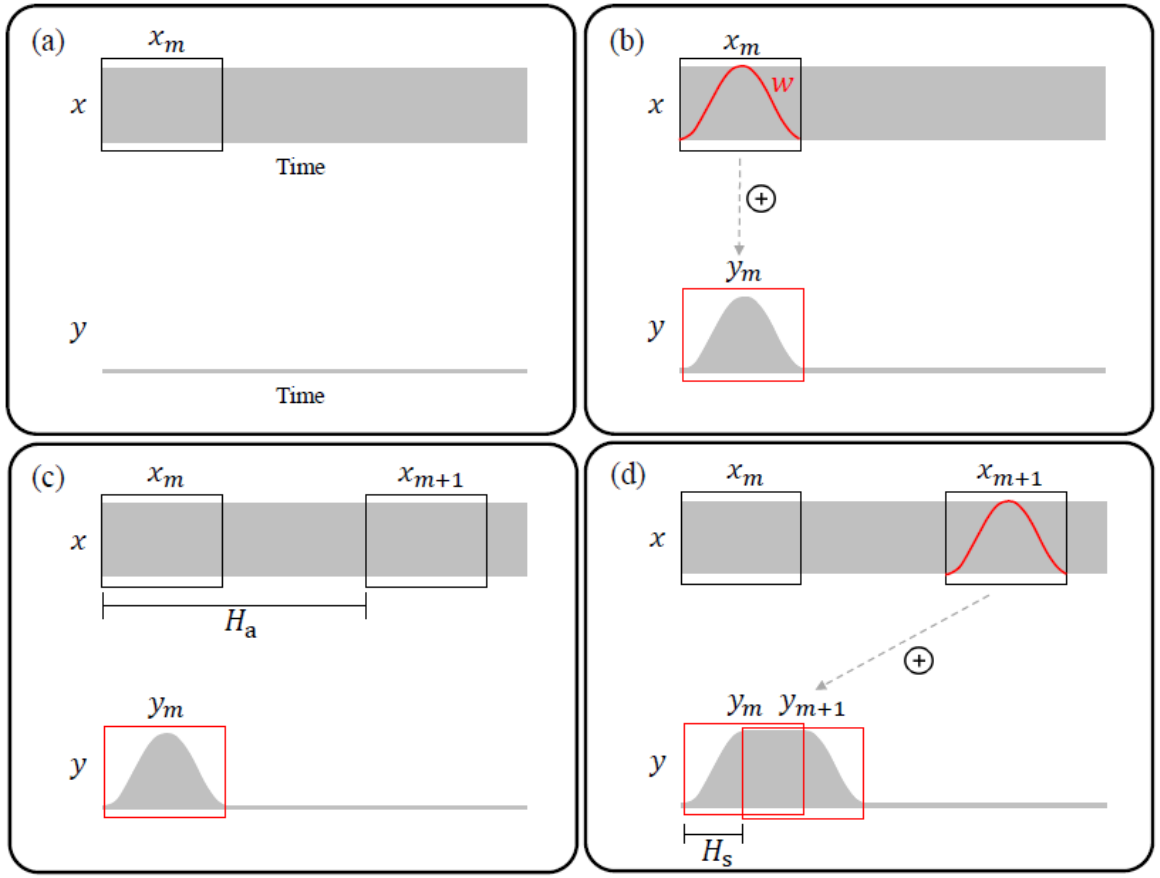
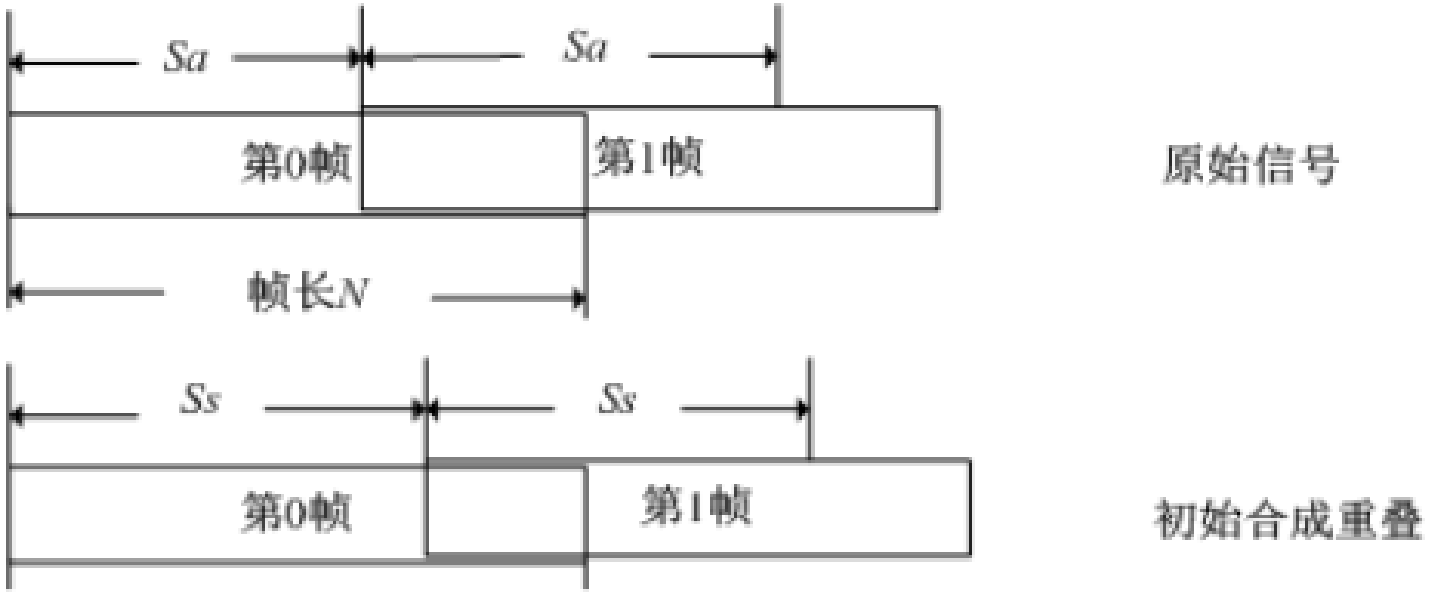


Figure 3. The principle of TSM based on overlap-add (OLA). (a) Input audio signal x with analysis frame x_m. The output signal y is constructed iteratively; (b) Application of Hann window function w to the analysis frame x_m resulting in the synthesis frame y_m; (c) The next analysis frame x_{m+1} having a specified distance of H_a samples from x_m; (d) Overlap-add using the specified synthesis hopsize H_s.

Link：https://www.mdpi.com/2076-3417/6/2/57/pdf

OLA

- 叠加算法
- $\partial = H_s/H_\partial$
- Ha：analysis hopsize
- Hs：synthesis hopsize



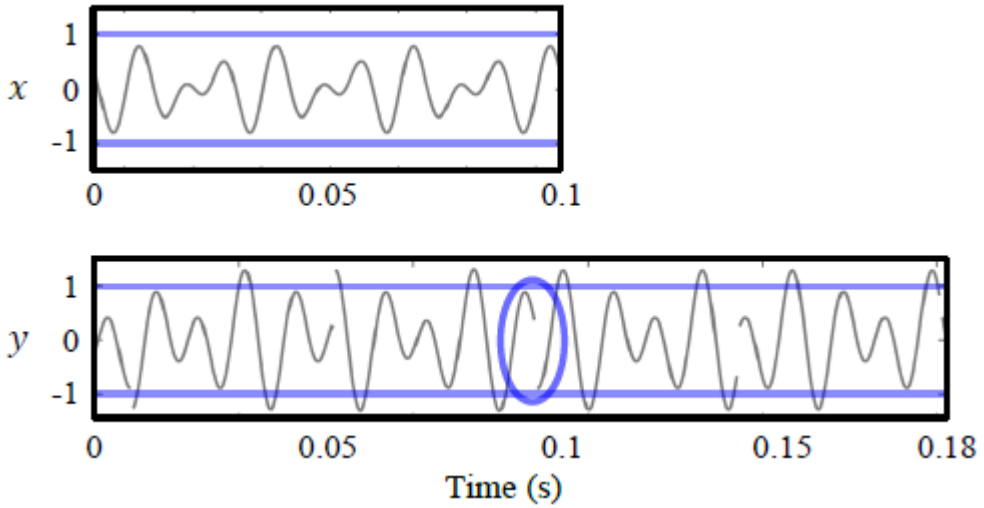
原始信号

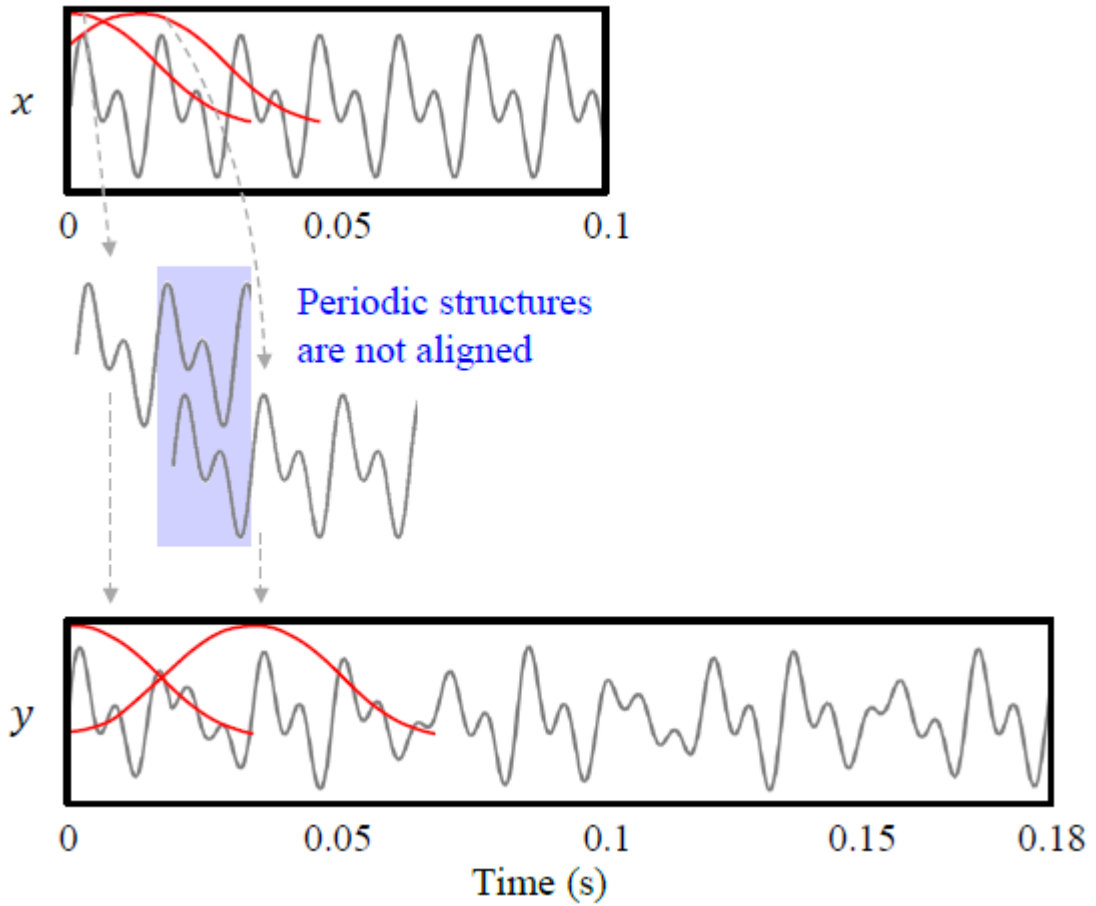
初始合成重叠

Link：https://www.mdpi.com/2076-3417/6/2/57/pdf

OLA

- 音频不连续
- 叠加部分信号幅值改变





Link：https://www.mdpi.com/2076-3417/6/2/57/pdf

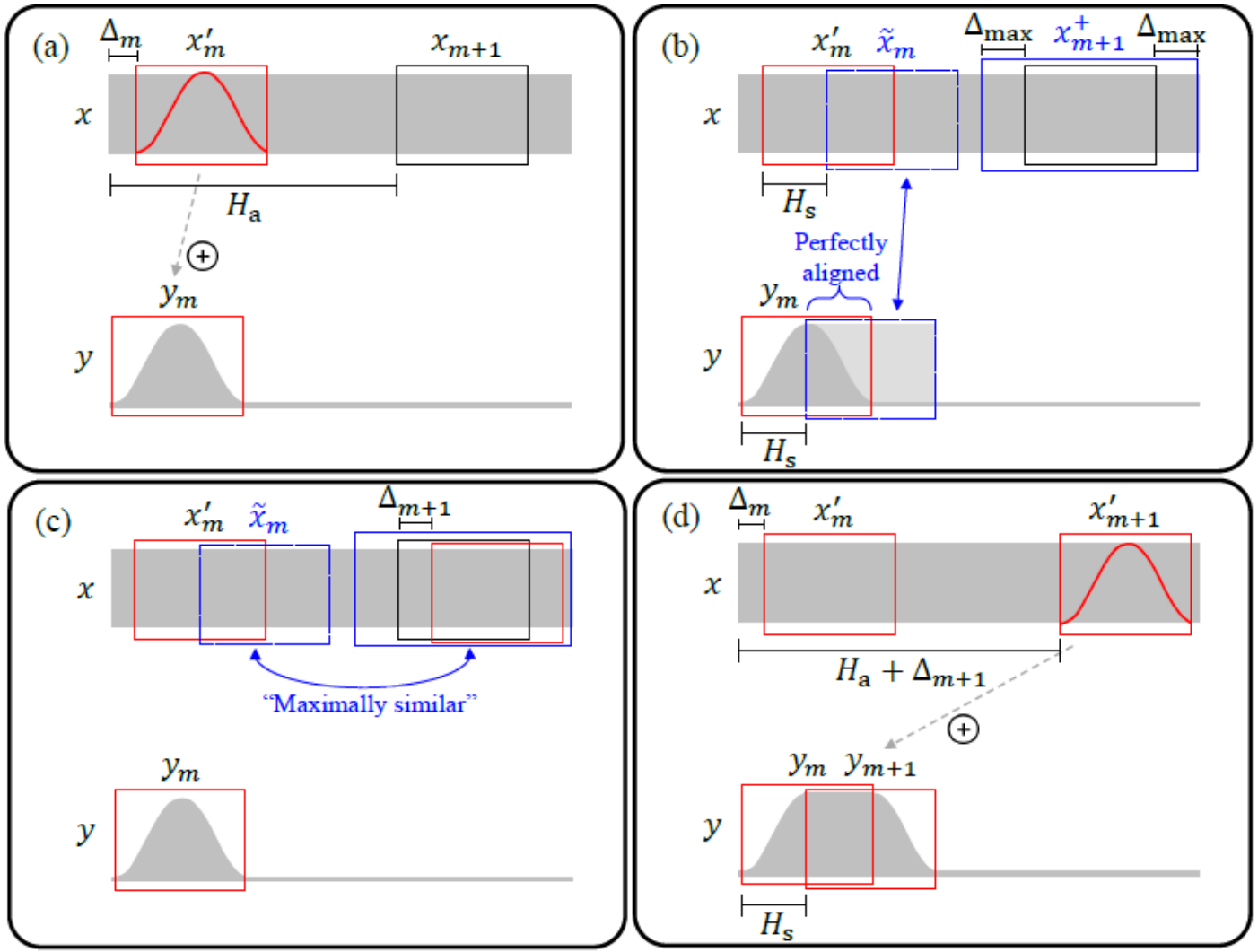
WSOLA

- Waveform Similarity Overlap-Add
- Cross-correlation（“自相关”）

$$\Delta_m \in [-\Delta_{\max} : \Delta_{\max}]$$

$$x'_m(r) = \begin{cases} x(r + mH_a + \Delta_m), & \text{if } r \in [-N/2 : N/2 - 1], \\ 0, & \text{otherwise.} \end{cases}$$

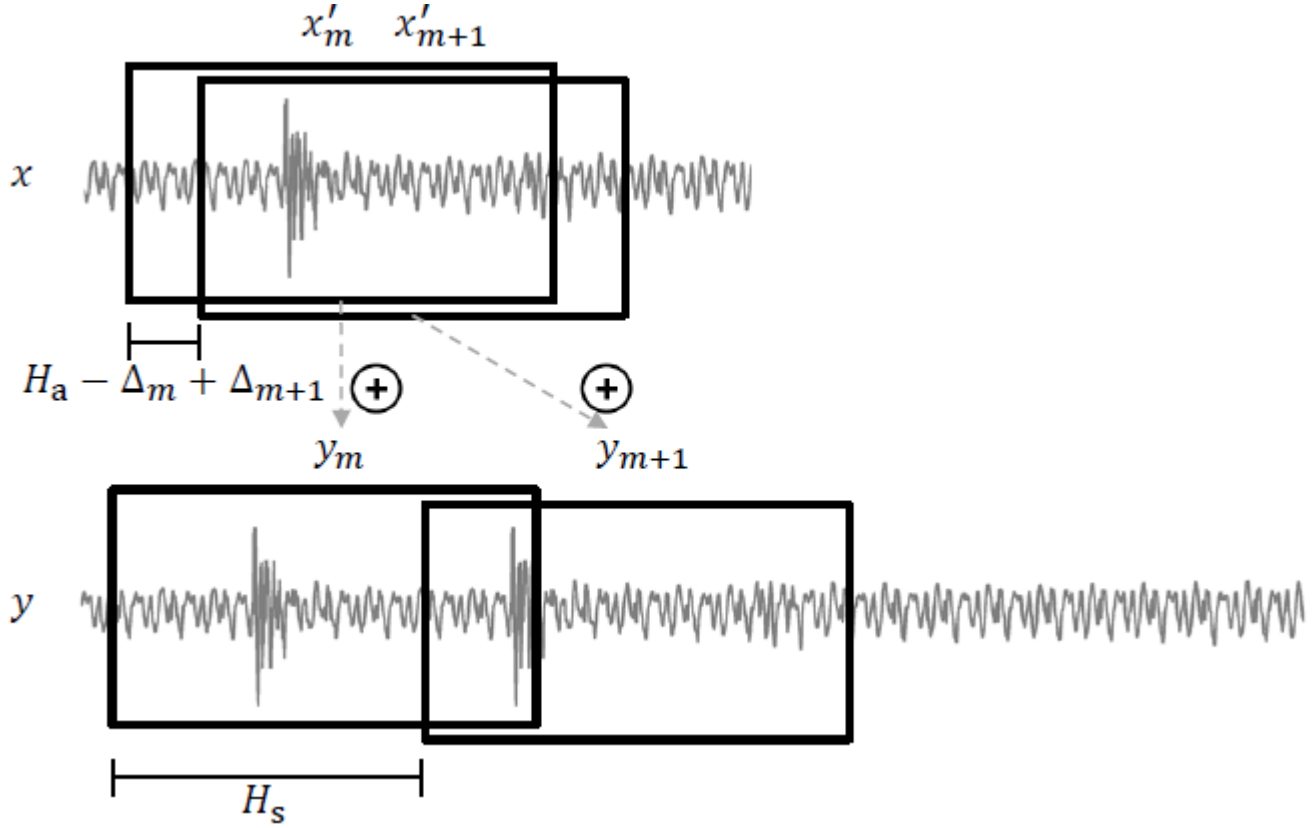
$$c(q, p, \Delta) = \sum_{r \in \mathbb{Z}} q(r)\, p(r + \Delta)$$



Link：https://www.mdpi.com/2076-3417/6/2/57/pdf

WSOLA

- L1：拉长变口吃
- L2：缩短变吞音
- L3：拉长缩短交响乐的时候只会保留主要部分



Link：https://www.mdpi.com/2076-3417/6/2/57/pdf

Phase Vocoder

m-1和m分别是连续的两帧,时间间隔为 $\triangle t$ ， $\varphi_1\varphi_2$ 分别是两帧启始相位,那么理想情况下应该有

$$\varphi_2 = \varphi_1 + w\Delta t$$

但实际上因为震荡(oscillates),可以看到,红色的正弦线和绿色的正弦线在交界处是相位不连续的,因此:

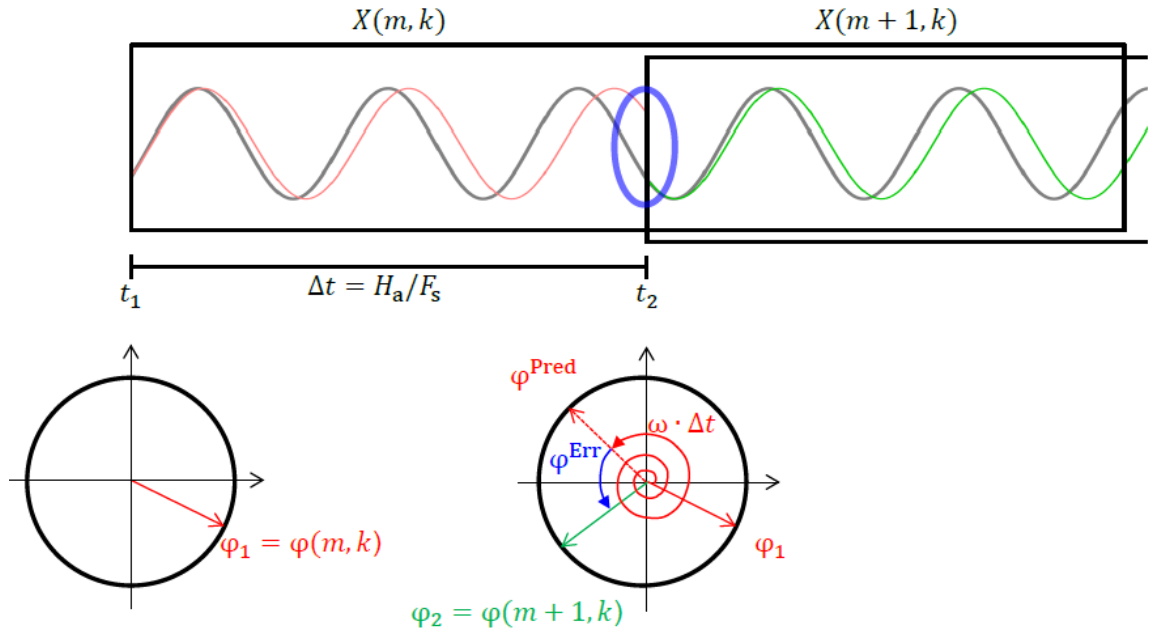
$$\varphi_1 \neq \varphi_2$$

我们用 $\varphi_1\varphi_2$ 的相位差 φ^{err} 来表示,就有

$$\varphi_2 = \varphi_1 + w\Delta t + \varphi^{err}$$

那么估算的瞬时频率应该是

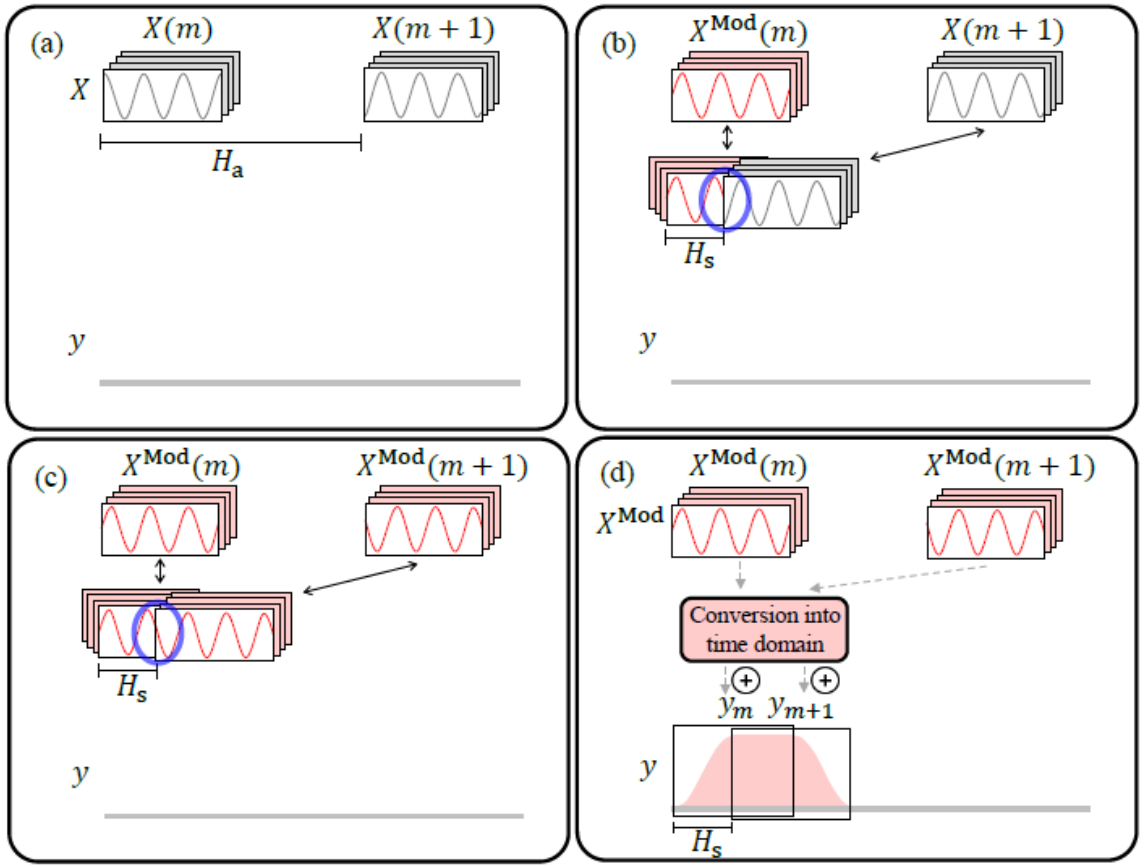
$$IF(w) = w + \frac{\varphi^{err}}{\Delta t}$$



Link: https://www.mdpi.com/2076-3417/6/2/57/pdf

Phase Vocoder

- a.先对原音频信号进行分帧处理
- b.通过STFT,对两个帧进行处理,计算其相位差,并依照瞬时频率估计章节的内容对其进行瞬时频率估算。
- c.通过瞬时频率的计算后,我们可以重构出和m+1帧没有相位跳变的信号。
- d.ISTF,也就是逆变换,重构时域信号



Link: https://www.mdpi.com/2076-3417/6/2/57/pdf

Phase Vocoder

- Transients are often smeared
- Phasiness：the loss of vertical phase coherence

Link：https://www.mdpi.com/2076-3417/6/2/57/pdf

实验目标与内容

实验目标与内容

- 观察不同ASR对于同一语音的识别效果
- 观察不同ASR对于同一语音不同倍速处理后的语音识别效果
- （level 2）对一段语音部分音节做不同倍速处理后，观察不同ASR的识别效果

实验Setup

实验流程 —— ASR识别系统

生成基础语音库
(10条)

倍速处理
(TSM库/会声会影库)

Speech recognition

基础语音库	倍速处理	ASR
• Okay Google, take a picture. • Okay Google, turn off the light. • ……	• TSM开源算法（已完成） • 会声会影商用算法（正在做）	• CMU Sphinx • Google STT • Baidu STT • 科大讯飞 STT

环境搭建 —— ASR识别系统

- CMU Sphinx
- Google STT
- Baidu STT
- 科大讯飞 STT
- Kaldi (环境还没搭好)
- DeepSeech (环境还没搭好)

Sphinx&Google:
https://github.com/Uberi/speech_recognition.

科大讯飞

- APPID：5e4936be
- APISecret：0c54ef03a106903edf9b9fce4e82cbc9
- APIKey：a1d59fcb877819cf203e7ce804d248a4

百度

- AppID：18493239
- API Key：T5sA7FUN2803vZfVURRG8Fz0
- Secret Key：KHG7i6cS8Dksy2oSIDSGl0k1rHbC1L8L

DeepSpeech：https://github.com/mozilla/DeepSpeech

Kaldi：http://kaldi-asr.org.

环境搭建 —— 其他

- Google TTS文本转换：https://github.com/pndurette/gTTS
- 倍速处理TSM：https://github.com/Muges/audiotsm
- 实验室服务器：10.14.103.254 用户名：usslab 密码：db2013

实验初步测试结果

测试结果

指令名称	倍速速率	Sphinx	GSTT	科大讯飞	百度
Okay Google, take a picture.	1				
	0.5				
	1.5				
	2				
Okay Google, navigate to my	1				
	0.5				

详见表格

部分结论 —— 实际干扰因素

- GTTS生成语音不经过倍速处理也不能被ASR正确识别
- **唤醒词**识别率很低
- Google识别率很低（潜在因素可能包括翻墙限流、生成语音质量低等）
- 不同ASR对于同一倍速音频识别效果不同
- 不同倍速算法对ASR识别效果影响不同
- 不同速率对ASR识别效果影响不同（**低速率与高速率影响都很大**）

部分结论 —— 困难与接下来工作

- 国外大部分API都需要信用卡注册
- 国内国外API免费的也存在限流的情况
- 目前完全实现的ASR包括GSTT与Sphinx
- **探究科大讯飞、百度（80%）等黑盒ASR与DeepSpeech（10%）等开源白盒ASR在不同测试环境下的识别效果**
- **挖掘Google ASR识别率低的原因，改进现在程序效率**
- **引入评估参数对现有和未来数据做定量分析**

2月23号讨论Memo

未来工作

- 倍速算法自身机制对于语音识别影响程度的机理
- 随着倍速偏离正常值程度的增加，ASR识别影响程度变化的规律（线性or指数）
- 倍速算法在语义学或语言学上对ASR识别影响的机理（单词重叠或过慢）
- 细粒度执行评估实验（倍速0.5 1等变化为0.5、0.75、1、1.25、1.5、1.75、2）
- 挖掘Google ASR识别率低的原因，改进现在程序效率
- 引入评估参数对现有和未来数据做定量分析

Ps. 测试样本先不加唤醒词

4月1号讨论Memo

讨论内容

- **Paper1：** “Speech Coding and Audio Preprocessing for Mitigating and Detecting Audio Adversarial Examples on Automatic Speech Recognition”
- **Paper2：** “DOMPTEUR: Taming Audio Adversarial Examples with Psychoacoustic Compression”
- **结论1**：压缩算法对于未考虑此类预处理过程的攻击有影响，但是对于未经过此类预处理过的语料库训练的ASR的正常功能也有影响。因此此类防御方法必须要求ASR的训练样本为经过预处理过程后的数据。
- **结论2**：这两篇文章不管是研究的AE攻击方式，还是研究的预处理过程算法，考虑得都不全面。实验做的也不solid。

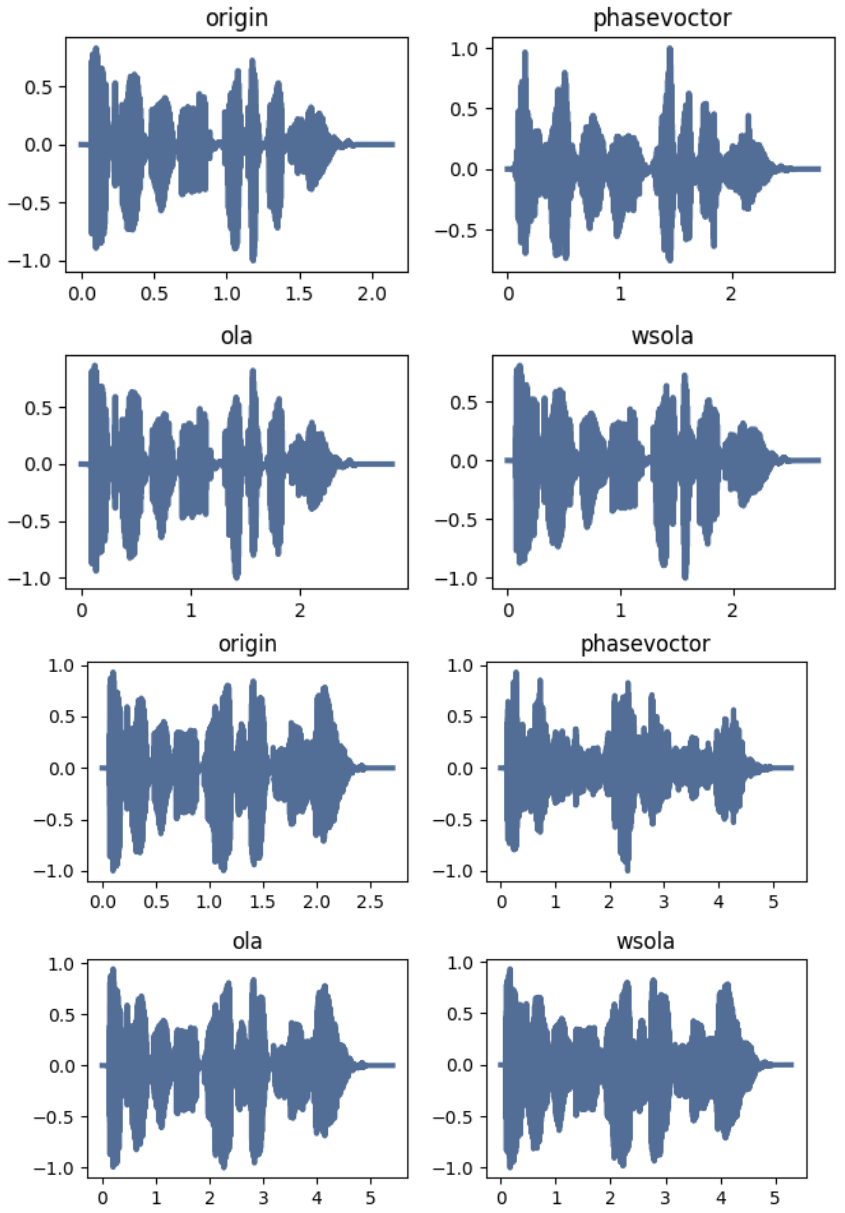
讨论内容

- **思考1**：基于预处理算法的防御方法的本质是找一个“全能滤波器”。1）此“全能滤波器是否存在？”；2）如何验证攻击不成功（悖论）？
- **思考2**：基于预处理算法的攻击方法要比防御更直接，更不容被质疑。
- **Link**： https://www.youtube.com/watch?v=ZncTqqkFipE（carlini演讲）

波形图时域细节

- **发现1**：PV有放大突出部分的感觉
- **发现2**：OLA时域上没什么变化
- **发现3**：WSOLA时域上类似于平滑滤波

- Ps. 准备绝对值



细粒度实验初步结果

- 正常无处理样本可做到100%
- 太慢或太快容易识别不出来
- 粗看OLA效果最好，OLA>WSOLA>PV，例如Okay->Ok
- 科大讯飞好于百度

4月10号 汇报PPT

最近进展

- **环境搭建**：1）开源模型环境搭建（沁宏详细介绍下现在卡在哪里）；2）百度、科大讯飞正常功能debug完成，确定google api国内不可用。
- **思路讨论**：基于预处理的防御方法详见P53-55
- **音频频谱图与波形图处理：**1）同一音频，同一速度，不同倍速算法；2）同一音频，同一倍速算法，不同速度。
- **倍速算法预处理机理研究：**详见P33-P40
- **细粒度探究实验:** 0.25:2.75:0.25，总结见P57
- **Github文件整理**

未来工作

- **量化实验（代码加输出）**：WER、MFCC相似度等
- **白盒开源模型调试完成**
- **部分单词倍速测试**

语义学

音节

- 音节（英语：syllable）是构成语音序列的单位，也是语音中最自然的语音结构单位
- 一个音节通常都包含一个音节核（syllable nucleus，通常由元音充当），此外还可能有音节起首和结尾的界音（margin，通常由辅音充当）
- 一个**元音音素**（音素不是字母）可构成一个音节，一个元音音素和一个或几个辅音音素结合也可以构成一个音节
- 元音音素可以构成音节，辅音音素不响亮，不能构成音节。

Phonemes

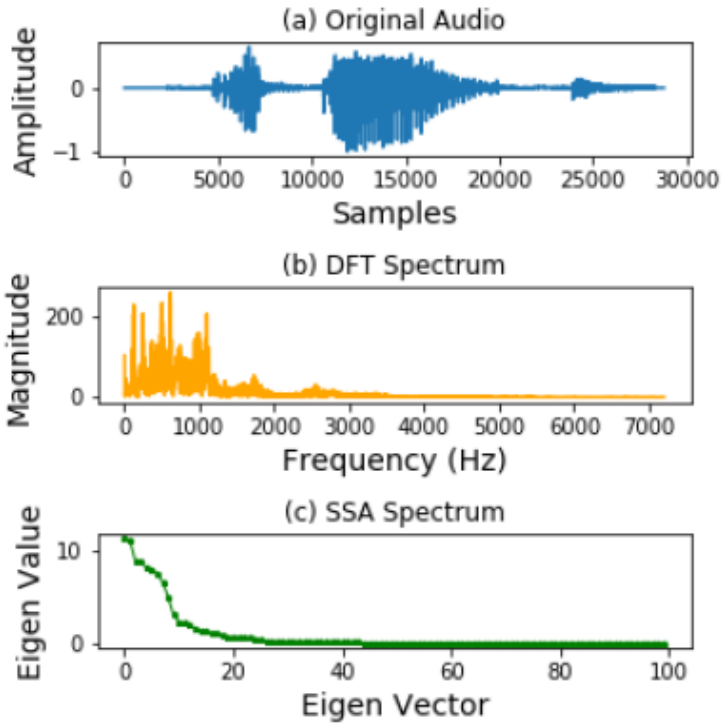
- English is made up of **44** phonemes
- **Vowels(元音)**
- **Fricatives(摩擦音)**
- Stops
- Affricates
- Nasal
- Glides

连读

- **参考资料：**https://zhuanlan.zhihu.com/p/45053115
- **发音规则一**：辅音+元音
- **发音规则二**：元音+元音：拼读成“元音+ [j] 或 [w] +元音”
- **发音规则三**：省略【h】的连读
- **发音规则四：**爆破音+爆破音=失去爆破
- **发音规则五：**爆破音[t]和[d]+鼻辅音[m]和[n]
- **发音规则六：**爆破音[t]和[d]+舌边音[l]
- **发音规则七：**爆破音+摩擦音/破擦音=失去爆破

关键问题

- **DFT**: Data-Independent Transforms:
- **SSA**: Singular Spectrum Analysis
- 参考资料： Hear “No Evil”, See “Kenansville”*: Efficient and Transferable Black-Box Attacks on Speech Recognition and Voice Identification Systems
- **To do list**



关键问题1：如何拆解句子与音素?

关键问题

- 这个还没想出来

关键问题2：如何规范化以上规则并建模?

关键问题

- PESQ
- User Study
- ……

关键问题3：如何衡量倍速之后的用户可察觉性?

思路Overview

- **Goal**：targeted attack/ 最优untargeted attack
- **L1**：攻击语句不固定
- **L2**：攻击语句固定

思路Overview

- **L1**：攻击语句不固定
- **G**：**根据目标语句搜索最优可行性攻击语句**
- **PS**：搭建基于连读等规则的攻击库
- **S1**：分解目标语句，提取关键音素
- **S2**：遍历连读等可被倍速影响的规则，并以最小人为察觉性对优化目标，修改攻击语句部分语句播放速度

思路Overview

- **L2**：攻击语句固定
- **G**：**根据攻击语句搜索最优恶意语句**
- **PS**：搭建基于连读等规则的攻击库
- **S1**：分解攻击语句，提取关键音素
- **S2**：遍历连读等可被倍速影响的规则，并以最大恶意程度为优化目标，修改攻击语句部分语句播放速度

DeepSpeech

测试结果

- 正常识别有点问题

text	DeepSpeech origin
Okay Google, take a picture.	okay google take an sure
Okay Google, navigate to my home.	okay google navigate the my home
Okay Google, turn off the light.	okay google turn off the light
Okay Google, turn on wireless hot spot.	okay google turn on wireless taught pot
Okay Google, restart phone now.	okay google burton now
Okay Google, read mail.	okay google read mail
Hey Cortana, open the website.	take were tana opened the website
Hey Cortana, make it warmer.	take were tana made it warmer
Echo, turn off the computer.	to turn off the computer
Echo, call my wife.	go call my wife

5月20日

ARPABET

- ARPABET (also spelled ARPAbet) is a set of phonetic transcription codes developed by Advanced Research Projects Agency (ARPA) as a part of their Speech Understanding Research project in the 1970s. It represents phonemes and allophones of General American English with distinct sequences of ASCII characters.

ARPABET

Vowels[2]

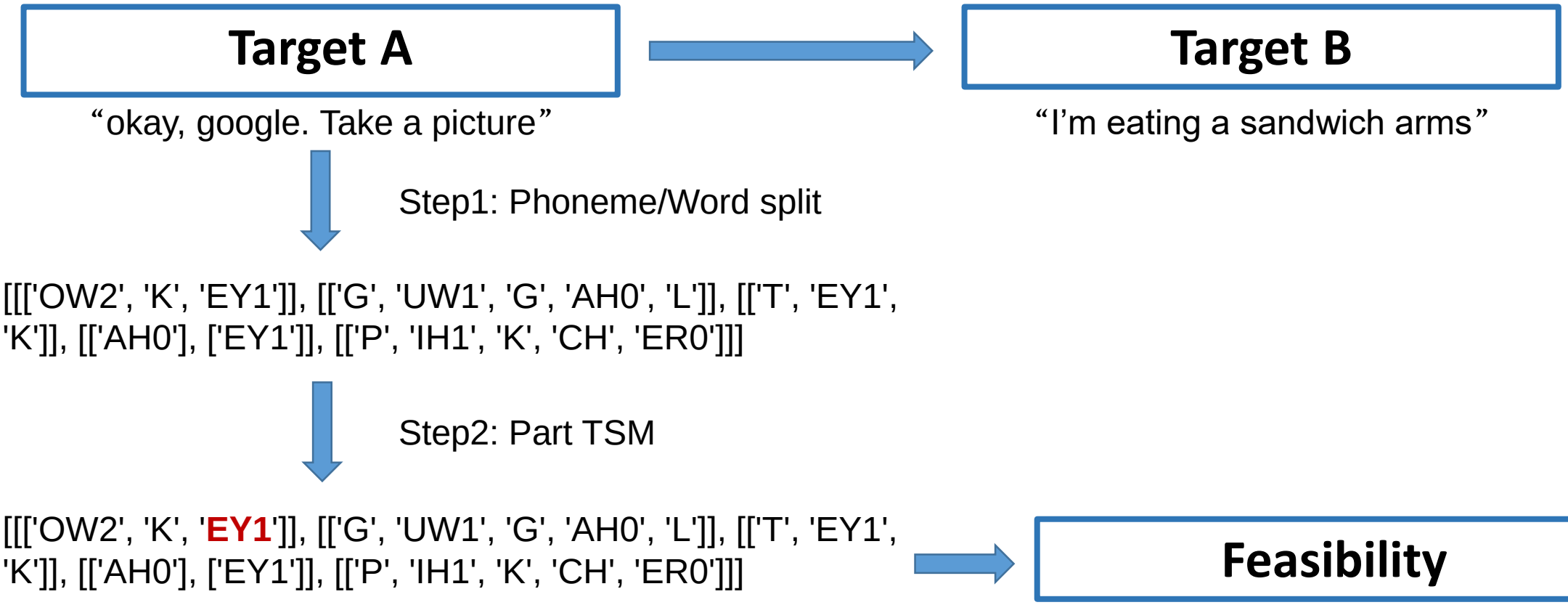
ARPABET		IPA	Example(s)
1-letter	2-letter		
a	AA	ɑ	balm, bot
@	AE	æ	bat
A	AH	ʌ	butt
c	AO	ɔ	story
W	AW	aʊ	bout
x	AX	ə	comma
N/A	AXR[3]	ɚ	letter
Y	AY	aɪ	bite
E	EH	ɛ	bet
R	ER	ɝ	bird
e	EY	eɪ	bait
I	IH	ɪ	bit
X	IX	ɨ	roses, rabbit
i	IY	i	beat
o	OW	oʊ	boat
O	OY	ɔɪ	boy
U	UH	ʊ	book
u	UW	u	boot
N/A	UX[3]	ʉ	dude

Consonants[2]

ARPABET		IPA	Example
1-letter	2-letter		
b	B	b	buy
C	CH	tʃ	China
d	D	d	die
D	DH	ð	thy
F	DX	ɾ	butter
L	EL	l̩	bottle
M	EM	m̩	rhythm
N	EN	n̩	button
f	F	f	fight
g	G	g	guy
h	HH *or* H[3]	h	high
J	JH	dʒ	jive
k	K	k	kite
l	L	l	lie
m	M	m	my
n	N	n	nigh
G	NX *or* NG[3]	ŋ	sing
N/A	NX[3]	ɾ̃	winner
p	P	p	pie
Q	Q	ʔ	uh-oh

5月26日

Model


Target A
Target B
“okay, google. Take a picture”
“I’m eating a sandwich arms”
Step1: Phoneme/Word split
[[['OW2', 'K', 'EY1']], [['G', 'UW1', 'G', 'AH0', 'L']], [['T', 'EY1', 'K']], [['AH0'], ['EY1']], [['P', 'IH1', 'K', 'CH', 'ER0']]]
Step2: Part TSM
[[['OW2', 'K', 'EY1']], [['G', 'UW1', 'G', 'AH0', 'L']], [['T', 'EY1', 'K']], [['AH0'], ['EY1']], [['P', 'IH1', 'K', 'CH', 'ER0']]]
Feasibility

Contributions

- 不改变语句的音素结构，从倍速的角度切入进行target攻击（但是无法保证一定能成功）

Challenge

- Audio phoneme split (Chaohao)：也是ASR的关键
- Transferability matrics (Chaohao)
- 如何自动化分析WER等指标和音素的关系
- 如何根据输入音频文本的音素进行拆解与倍速
- Model与black box的区别性（子集）

讨论点

- 在音频音素无法分解情况下，如何关联结果文本与ASR系统的关联关系，即输出文本的音素是否可以代表ASR识别结果的音素（详见P81关于ASR调研）
- Correlation (WER and the structure of phoneme)
- 识别结果不同是否是由于ASR神经网络训练的不同，即不同ASR对于同一音频的音素拆解不同。
- 部分音频识别出来无结果

ASR

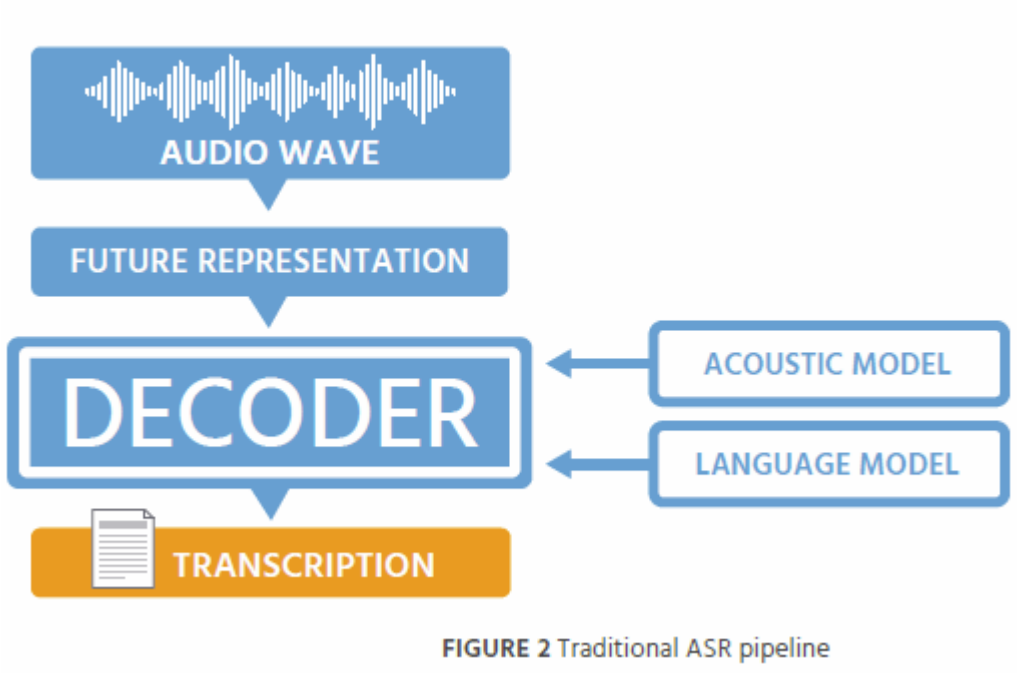


FIGURE 2 Traditional ASR pipeline

传统模型(Kaldi)

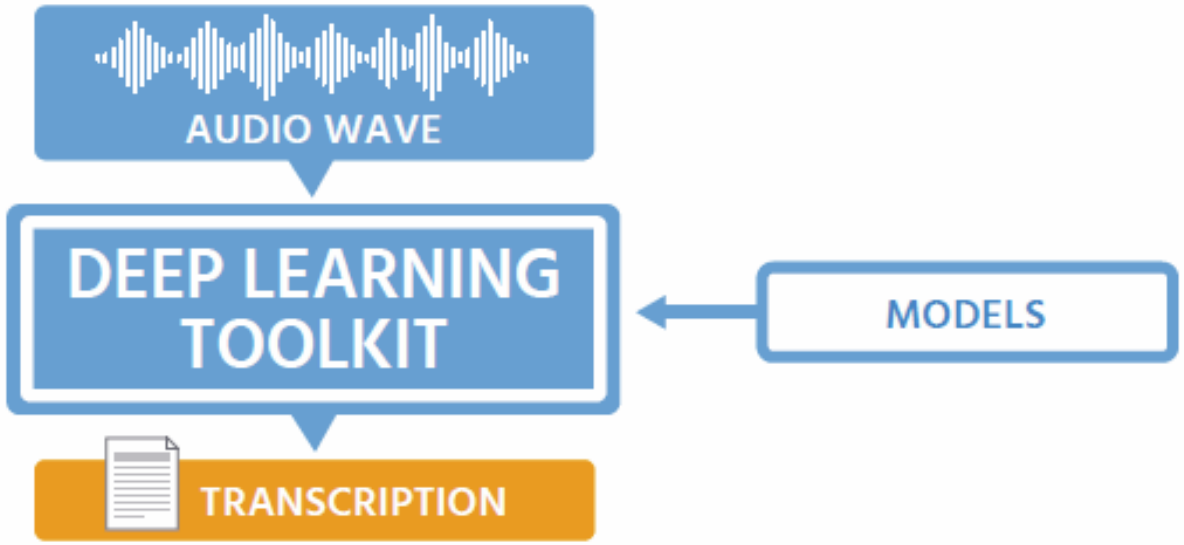


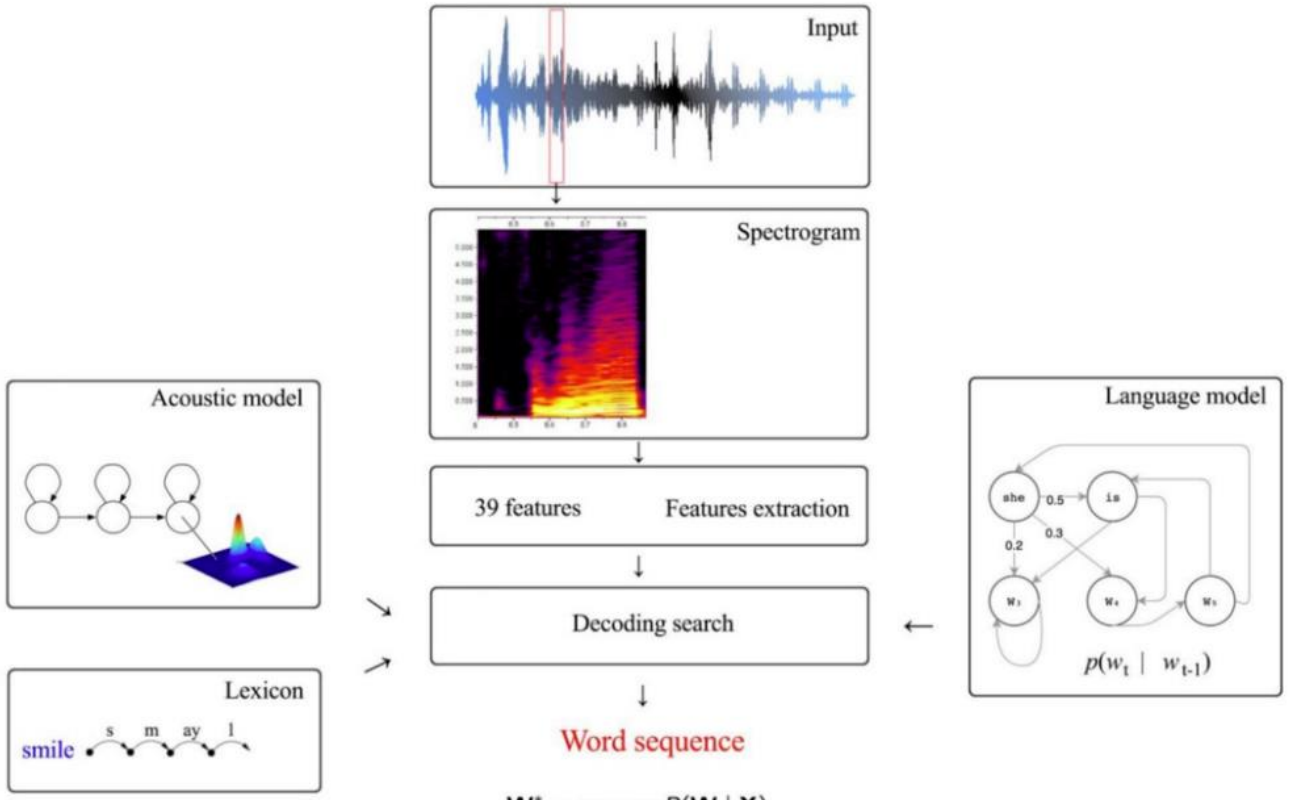
FIGURE 3 Deep learning ASR pipeline

现代模型(DeepSpeech)

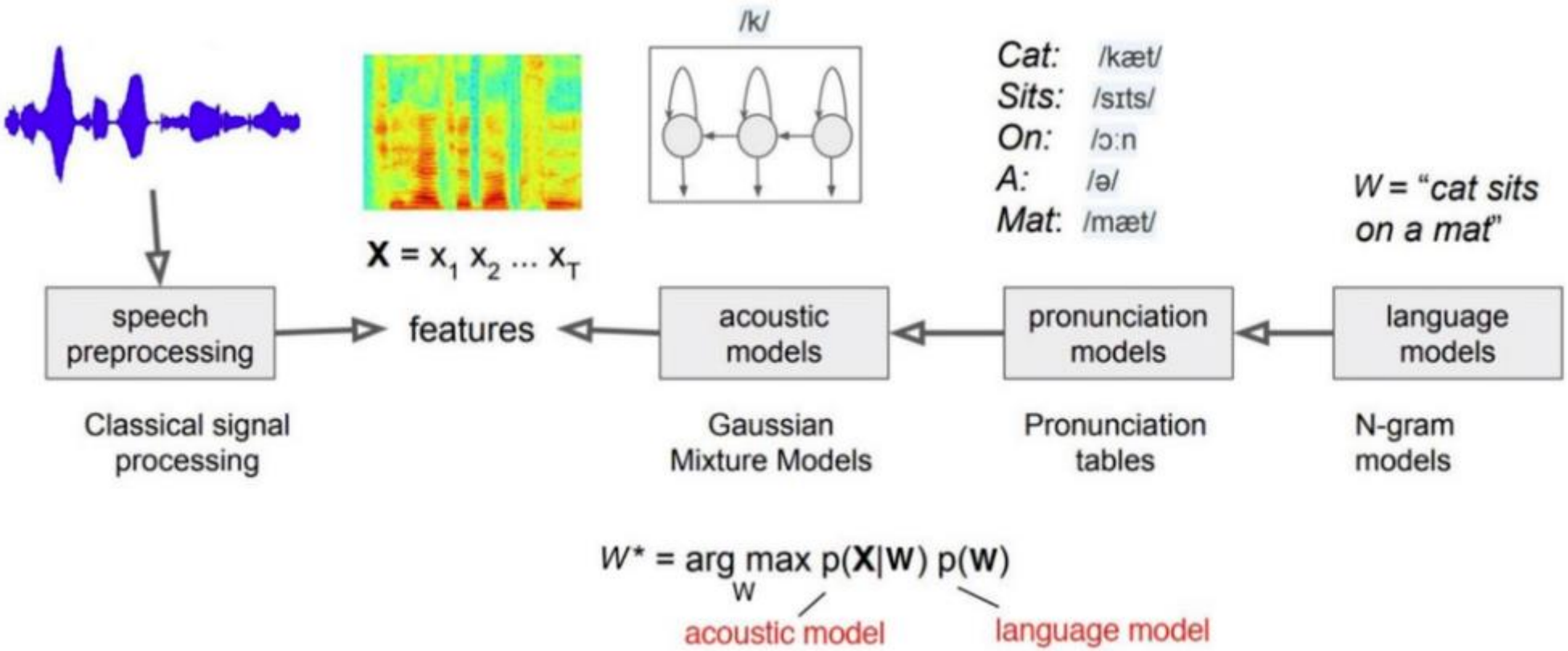
ASR —— Traditional

- **Create features from the sound file**: this may require **windowing** (a form of filtering and aggregation) and performing transformations on the windows, such as **Fourier transformations and connectionist temporal classification (CTC).**
- **Apply an acoustic model to match the phonemes**
- **Apply a language model** that uses probability distributions to predict words from the phonemes and then the sequences of words from the phonemes.
- **Kaldi, CMU Sphinx and HTK**

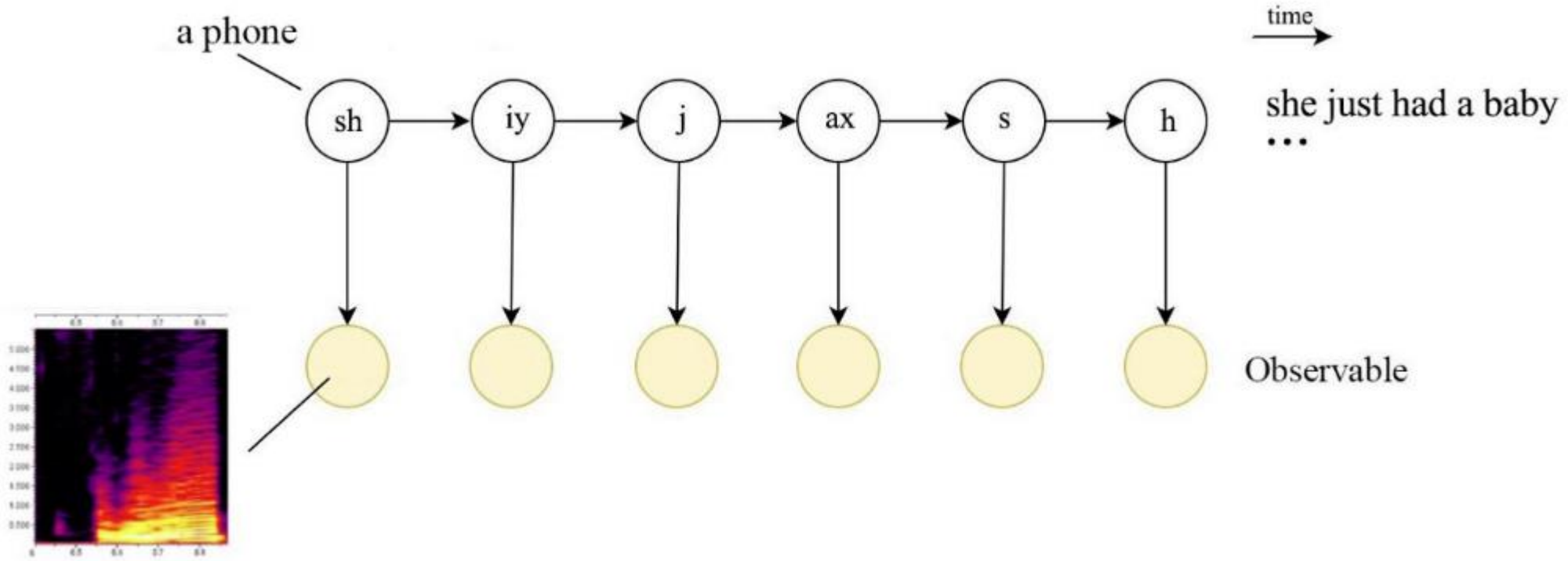
ASR —— Traditional



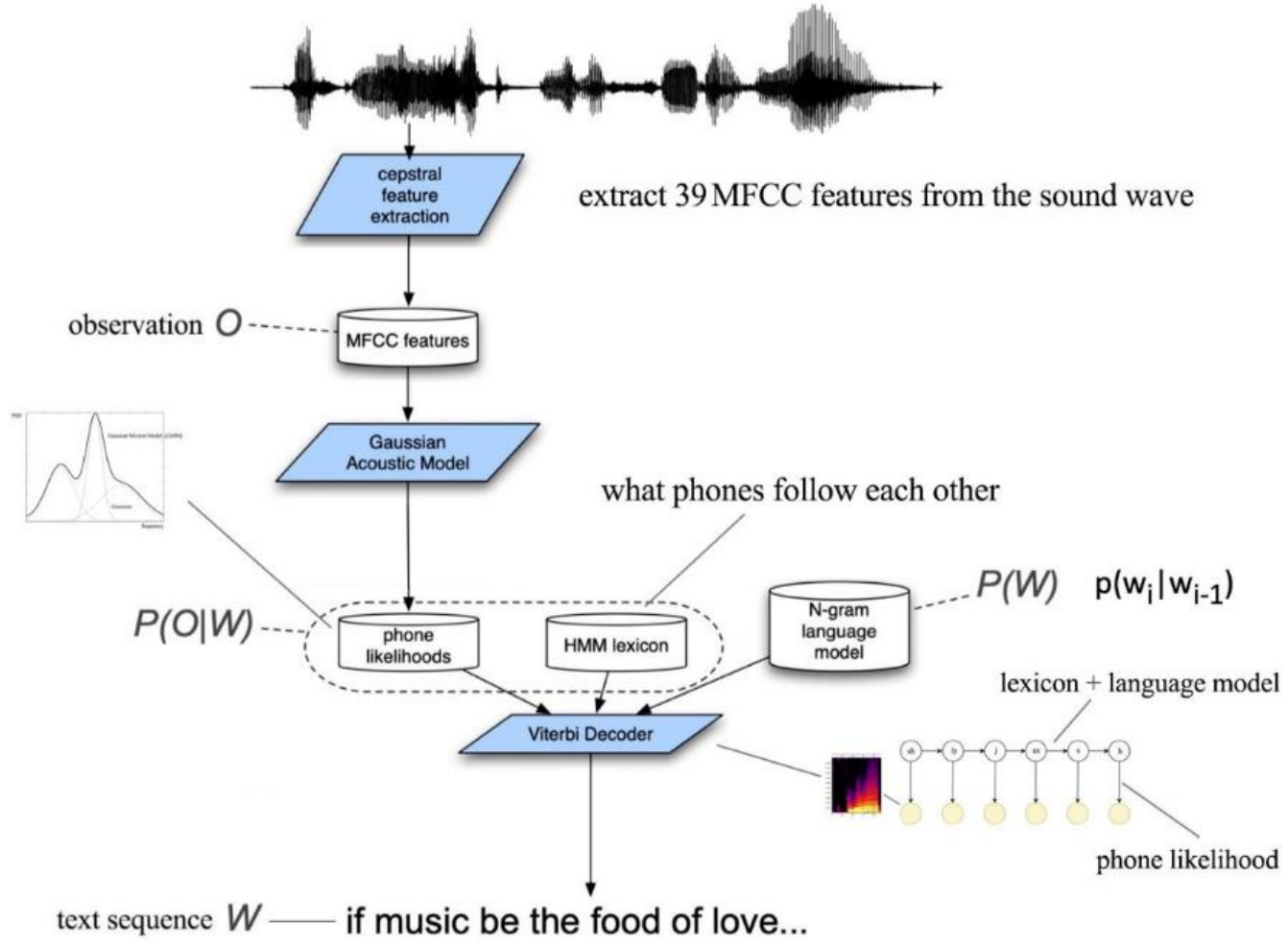
ASR —— Traditional



ASR —— Traditional

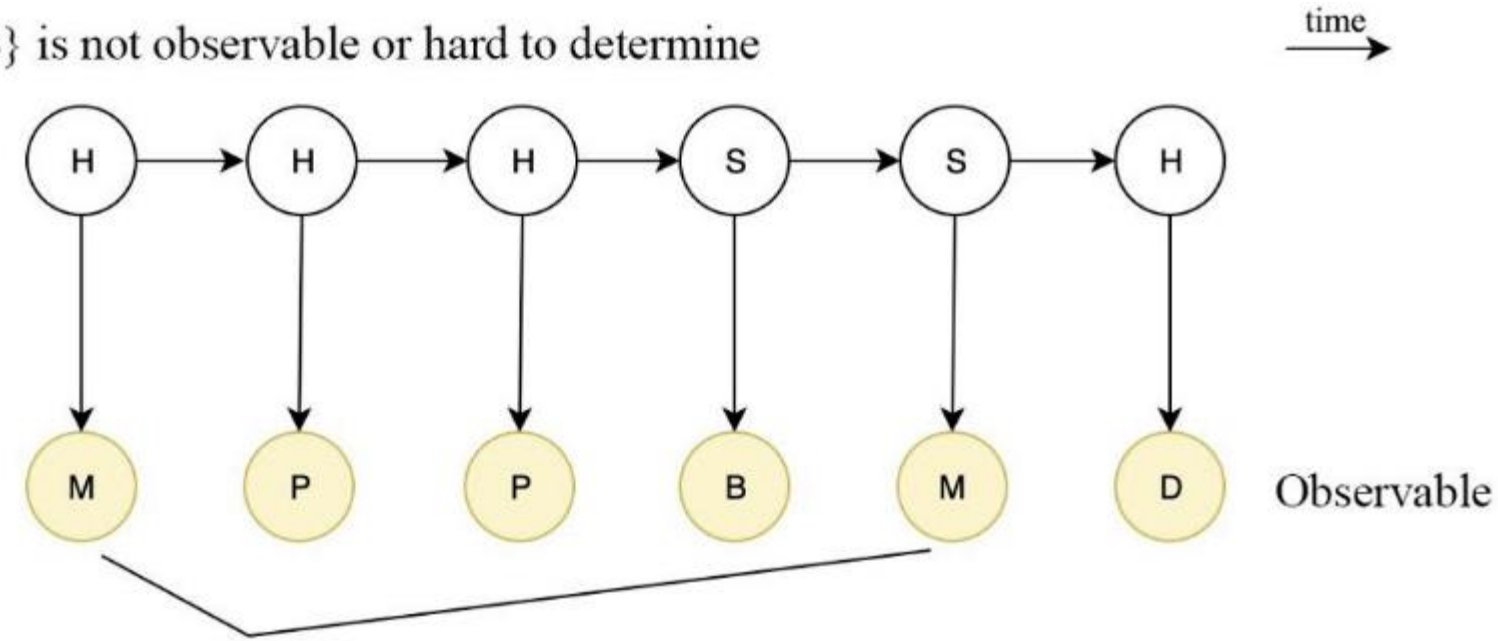


ASR —— Traditional



ASR —— HMM

internal state {H, S} is not observable or hard to determine

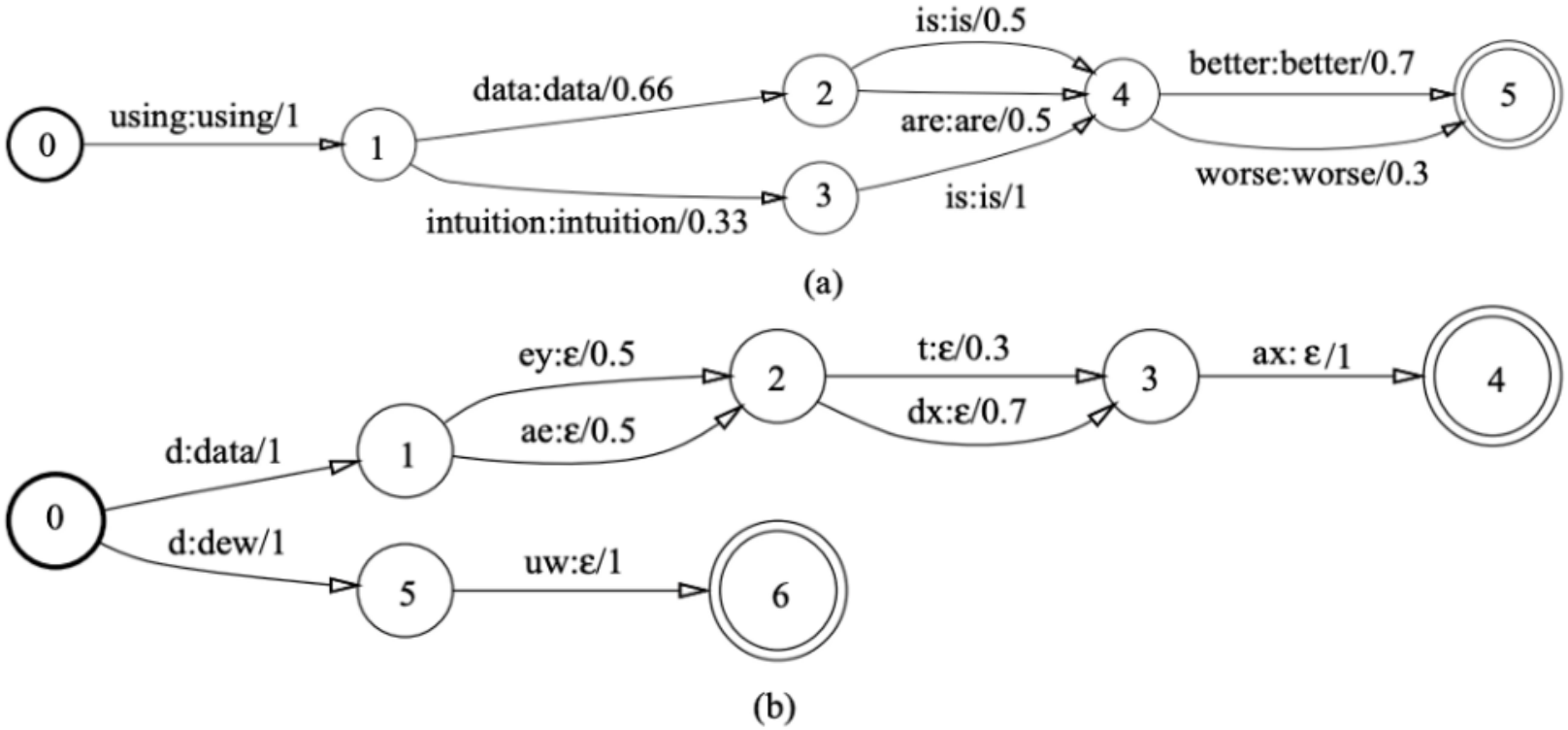


0.2 chance that I go to movie when I am happy.

0.4 chance that I go to movie when I am sad.

ASR —— Kaldi

- OpenFST：constructing and searching weighted finite-state transducers (WFST)

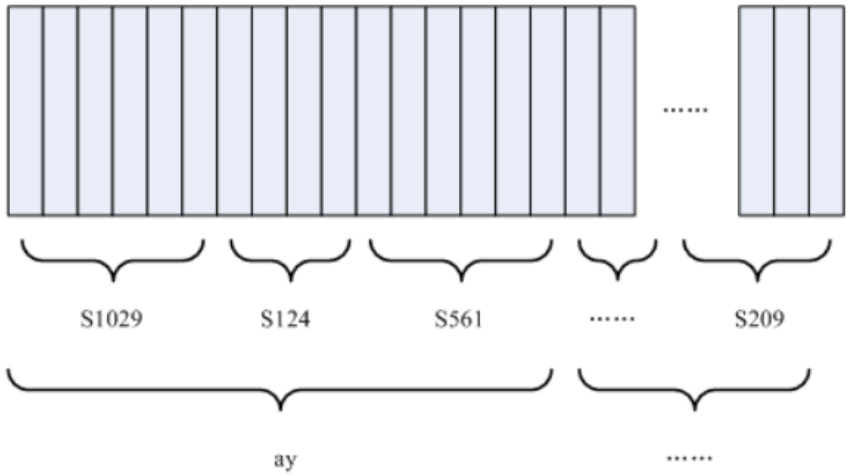


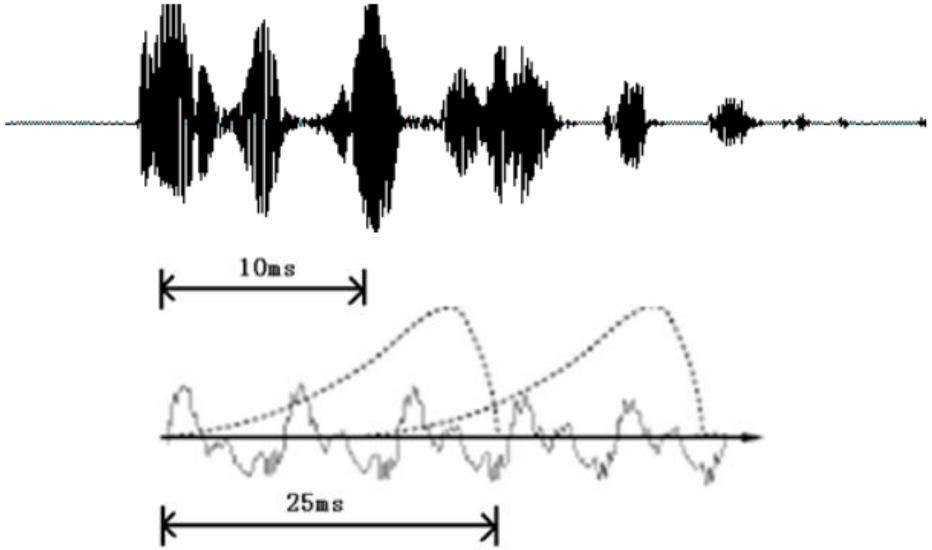
ASR —— Deep Learning

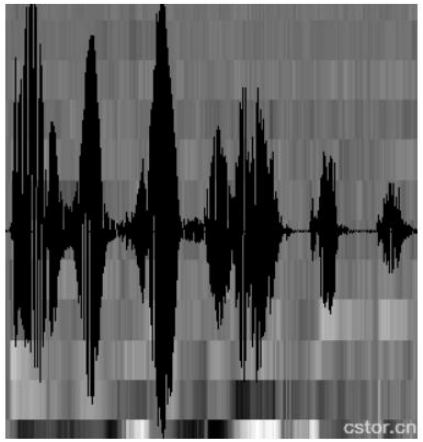
- Replace the intermediate steps with **one algorithm**
- **DeepSpeech, PyTorch-Kaldi and CNT**

ASR

- 第一步：把帧识别成状态（难点）
- 第二步：把状态组合成音素
- 第三步：把音素组合成单词
- 帧 – 状态号 – 状态 – 音素 – 单词

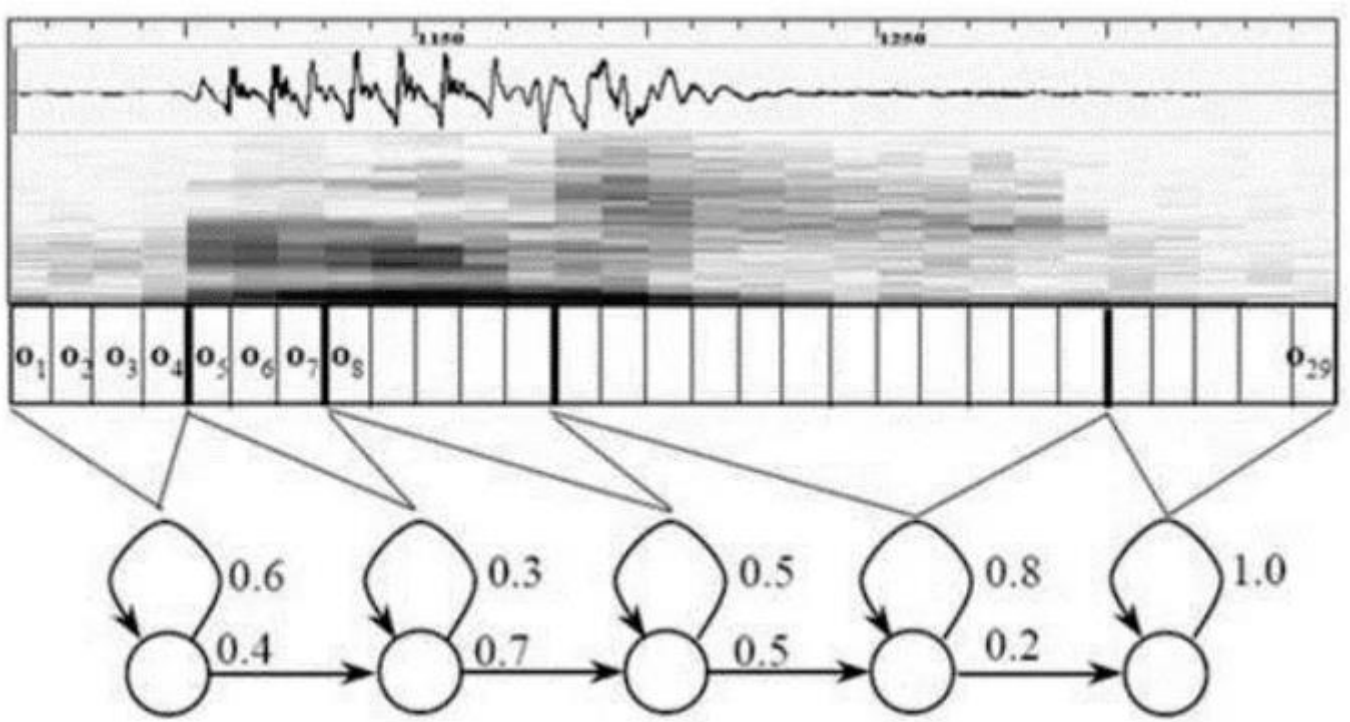






ASR

- 第一步：把帧识别成状态（难点）
- 第二步：把状态组合成音素
- 第三步：把音素组合成单词

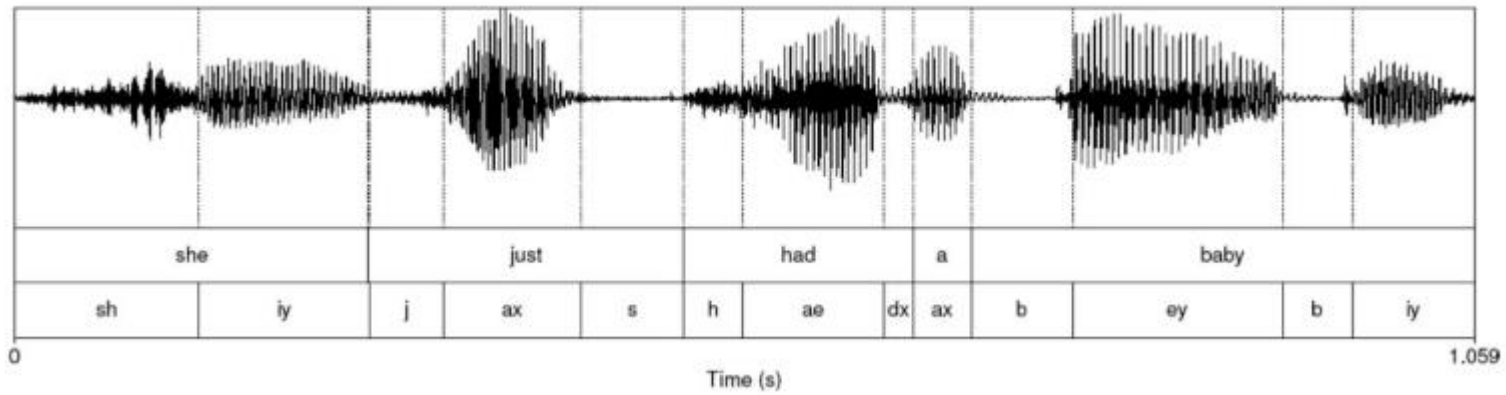


VAD

- 第一步：把帧识别成状态（难点）

Phonemes

- 第一步：把帧识别成状态（难点）



Interval

- This 25ms width is large enough for us to capture enough information and yet the features inside this frame should remain relatively stationary. If we speak 3 words per second with 4 phones and each phone will be sub-divided into 3 stages, then there are 36 states per second or **28 ms** per state. So the 25ms window is about right.
- Each slid window is about **10ms** apart so we can capture the dynamics among frames to capture the proper context.
- Starting from an audio clip, we slide windows of 25 ms width and 10 ms apart to extract MFCC features.

ASR

- 对于DeepSpeech之类的，输出文本等效为ASR识别出来的音素
- 输入音频的拆解等效于ASR的功能，无法实现

WER结论

- 0.25倍速，会出现wer>1
- Phasevecotor > 1.5, wsola>1.75, ola>2.0, 倍速意义不大，高速情况ola优于wsola
- Phasecotor<0.75, ola<0.5, wsola<0.5， 倍速意义不大，低速情况wsola优于ola

MFCC相似度结论

- 1倍速，Ola的距离值最低
- 整体上，距离值大于11000，wer大于0，识别效果差

phasevector										
speed	sentence1	sentence2	sentence3	sentence4	sentence5	sentence6	sentence7	sentence8	sentence9	sentence10
0.25	39889.98	47270.7	43178.2	59820.82	49354.63	36470.51	43480.38	36385.68	41470.24	33257.1
0.5	27270.31	32851.55	28721.84	40919.19	32719.96	25158.47	29929.89	24872.48	29874.11	25020.2
0.75	15341.7	17762	16196.85	22380.06	18325.81	12909.64	15790.56	12916.91	13521.8	11255.25
1	6894.58	7194.089	7093.317	9916.536	7741.316	5795.199	7578.904	6130.545	6367.819	5110.082
1.25	15578.13	17628.38	16959.32	23578.61	20792.5	15199.59	16296.06	15093.82	14160.08	11146.2
1.5	20048.65	19063.95	18150.04	27954.58	23109.53	16452.82	20276.9	16074.94	15363.63	12489.61
1.75	20398.96	22159.91	19310.78	29449.8	25652.36	16745.08	22454.6	17941.16	18728.17	14205.02
2	21338.68	22972.68	21903.03	32429.8	28627.06	19352.6	24336	18890.39	19629.22	16756.56
2.25	22641.23	24957.48	23625.69	35426.05	29517.47	21491.05	25520.49	22035.15	20686.52	16553.86
2.5	24730.28	26894.29	25482.68	37596.83	33411.48	21552.07	28252.05	23599.96	21407.56	18823.73
2.75	25831.23	27272.21	27152.71	38042.21	32871.46	22157.72	30632.85	24937.46	22557.98	20489.57
ola										
speed	sentence1	sentence2	sentence3	sentence4	sentence5	sentence6	sentence7	sentence8	sentence9	sentence10
0.25	65846.29	76425.41	64550.41	97575.31	74418.56	56461.18	69484.46	60619.3	56722.45	47041.05
0.5	23343.11	24636.17	22289.57	33970.28	25951.14	19066.99	23985.06	20881.02	19888.71	16218.57
0.75	12732.23	15104.52	12864.49	18159.59	15447.88	11283.36	13523.61	11626.82	11646.83	9742.718
1	6.341849	6.847515	7.628276	7.672218	7.49167	6.938226	7.306899	7.147512	7.651117	6.99104
1.25	8297.261	9643.755	9050.066	13144.3	9633.715	7632.144	9190.457	8258.293	7897.48	7106.229
1.5	10462.27	12559.74	11385.1	16033.52	12984.42	10269.59	12226.73	10300.51	10944.84	8805.62
1.75	13588.47	14649.65	13979.11	19920.8	15068.57	12482.04	15298.08	13179.33	12132.34	10447.42
2	15862.63	18233.3	16966.93	23204.8	18826.96	14423.73	17755.14	14058.06	14635.02	11777.55
2.25	17228.82	21408.53	19882.23	26442.42	21073.15	15269.83	18868.59	16349.33	16796.6	13176.87
2.5	19179.36	21587.26	20563.7	29084.18	23614.27	16172.74	19978.43	18566.15	18163.7	14775.93
2.75	21134.35	21458.75	21964.08	29322.67	25147.84	17070.56	21705.51	19157.64	18219.51	15311.76
wsola										
speed	sentence1	sentence2	sentence3	sentence4	sentence5	sentence6	sentence7	sentence8	sentence9	sentence10
0.25	25556.08	29455.58	29007.31	43997.53	34243.39	24325.47	32193.81	27487.51	26221.64	21300.92
0.5	12562.19	14195.54	14438.99	21630.23	16892.66	11763.55	15051.91	12398.38	12642.93	9924.31
0.75	9288.731	9833.873	9516.774	14569.07	11223.12	8079.981	8037.192	7074.29	8804.695	6971.243
1	7148.826	7125.882	7356.148	7933.054	7667.055	6527.696	7561.281	6099.988	6629.823	4252.961
1.25	10350.2	9362.391	9655.374	13387.24	9523.717	7590.66	9261.01	7594.629	7328.467	5544.797
1.5	11294.01	11463.2	11468.44	16141.85	12137.7	9800.362	11087.4	10459.69	8653.297	6408.128
1.75	13664.09	14068.4	13991.19	21648.79	14818.64	12247.51	14305.84	11837.58	11269.54	9203.405
2	15484.72	17870.61	16469.32	23059.29	18710.23	14507.59	16804.73	13206.48	12877.71	11562.57
2.25	21580.6	25255.06	23279.74	35430.36	27788.27	21826.26	25252.63	20079.86	19484.09	16385.86
2.5	21483.77	23228.52	24601.63	34600.59	30455.93	24124.96	23328.67	20987.21	22424.21	17797.32
2.75	23765.4	28511.57	25075	38664.2	33944.13	24777.35	31807.3	26007.64	23197.18	18619.8

音素拆分结论

- 正常：比如hey->[['HH', 'EY1']]
- 部分单次拆出两个部分，比如a->[['AH0'], ['EY1']], hello->[['HH', 'AH0', 'L', 'OW1'], ['HH', 'EH0', 'L', 'OW1']]

[['K', 'AO1', 'L']], [['M', 'AY1']], [['W', 'AY1', 'F']]

[['F', 'AA1', 'L', 'OW0']]

[['N', 'AW1']] [['AY1']]

a	AA	ɑ	b**a**lm, b**o**t
@	AE	æ	b**a**t
A	AH	ʌ	b**u**tt
c	AO	ɔ	st**o**ry
o	OW	oʊ	b**oa**t

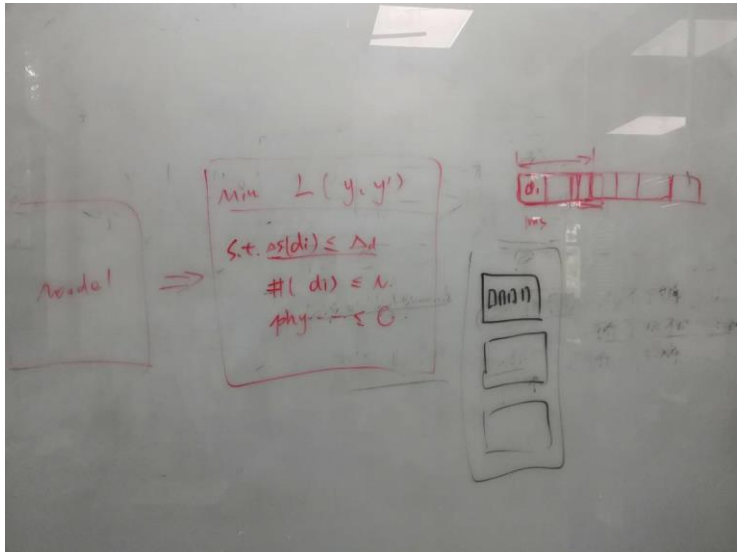
f	F	f	**f**ight
g	G	g	**g**uy
h	HH *or* H[3]	h	**h**igh
J	JH	dʒ	**j**ive
k	K	k	**k**ite

W	AW	aʊ	b**ou**t
x	AX	ə	comm**a**
N/A	AXR[3]	ɚ	lett**er**
Y	AY	aɪ	b**i**te

5月27日

会议纪要 —— 模型

- 以最小时间单元（参考ASR状态间隔或设置参数间隔）切分输入音频
- 优化条件：调节单元个数、单元调节速度以及可听性（心理声学）
- 与发音学模型相互印证



会议纪要 —— To do list

- 发音学模型建立（chaohao）
- ASR原理学习，尤其是音素分解与转换部分（chaohao）
- ASR设置参数提取（qinhong）
- 最小时间单元切分倍速随机测试（chaohao）
- 优化模型撰写（qinhong）
- Q1：探究快速或者慢速谁的效果更好（沁宏）
- Q2：元音、辅音谁更重要（chaohao）

会议纪要 —— 重要批注

- 1.1倍与1.25倍识别错误是因为正好影响了音素分解的间隔
- 让ASR找不准音素

基于Kaldi的优化模型 —— Lea Schönherr

- Adversarial Machine Learning

$$x' = x + \delta, \quad \text{such that } F(x) \neq F(x'),$$

基于Kaldi的优化模型 —— Lea Schönherr

- Measure loss
- cross-entropy

$$L(y_i, y') = -\sum y_i \log(y')$$

其中某个样本的正确答案即p是$[1,0,0]$，某模型经过Softmax激活后的答案即预测值q是$[0.5,0.4,0.1]$，那么这个预测值和正确答案之间的交叉熵为：

$$H(p=[1,0,0], q=[0.5,0.4,0.1]) = -(1*log0.5 + 0*log0.4 + 0*log0.1) \approx 0.3$$

如果另外一个模型的预测值q是$[0.8,0.1,0.1]$，那么这个预测值和正确答案之间的交叉熵为：

$$H(p=[1,0,0], q=[0.8,0.1,0.1]) = -(1*log0.8 + 0*log0.1 + 0*log0.1) \approx 0.1$$

基于Kaldi的优化模型 —— Lea Schönherr

- Calculate gradient
- The loss is back-propagated to the input *xi* of the neural network
- The derivative of F(xi) depends on the **topology of the neural network** and is also calculated via the chain rule, going backward through the different layers.

$$\nabla x_i = \frac{\partial L(y_i, y')}{\partial x_i} = \frac{\partial L(y_i, y')}{\partial F(x_i)} \cdot \frac{\partial F(x_i)}{\partial x_i}.$$

基于Kaldi的优化模型 —— Lea Schönherr

- Update

$$x_{i+1} = x_i - \nabla x_i \cdot \alpha.$$

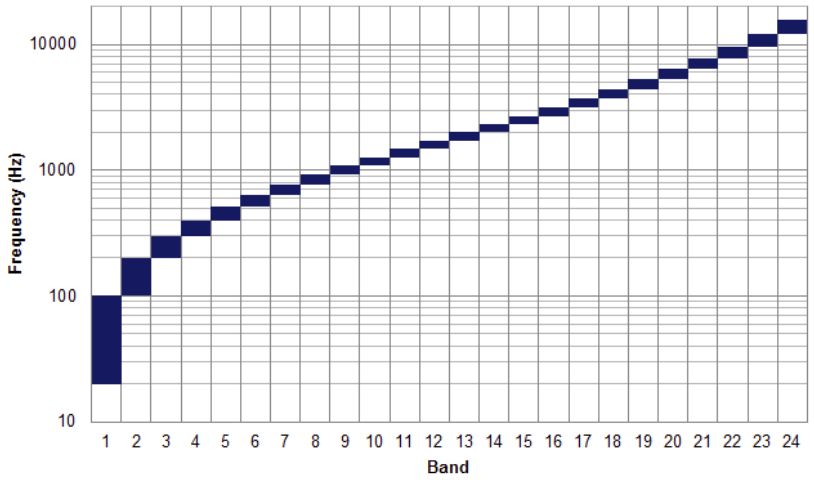
基于Kaldi的优化模型 —— Lea Schönherr

- Psychoacoustic Modeling
- MP3 compression
- define how **dependencies** between certain frequencies can mask, i. e., make inaudible, other parts of an audio signal
- **add inaudible** noise.

基于Kaldi的优化模型 —— MP3 Compression

- granule windows
- fast Fourier transform
- bark scale
- estimate the relevance of each band and compute its energy

$$\text{Bark} = 13\arctan(0.00076f) + 3.5\arctan((f/7500)^2)$$



基于Kaldi的优化模型 —— Overview

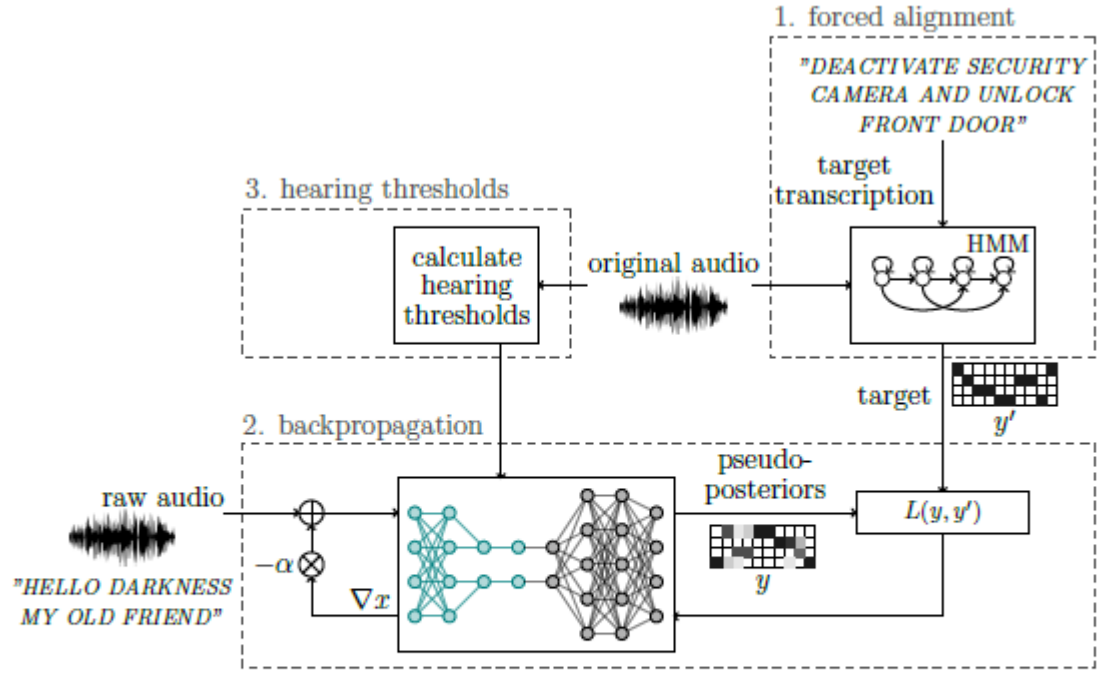


Fig. 3: The creation of adversarial examples can be divided into three components: (1) *forced alignment* to find an optimal target for the (2) backpropagation and the integration of (3) the hearing thresholds.

基于Kaldi的优化模型 —— Forced Alignment

- due to the decoding step—which includes a **graph search** — for a given transcription, many valid pseudo-posterior combinations exist
- For example, when the same text is spoken at different **speeds**, the sequence of the HMM states is correspondingly faster or slower
- This algorithm is provided by the Kaldi toolkit

基于Kaldi的优化模型 —— Integrating Preprocessing

- This design choice does not affect the accuracy of the ASR system, but it allows for manipulating the raw audio data by applying backpropagation to the preprocessing steps, directly giving us the optimally adversarial audio signal as result.

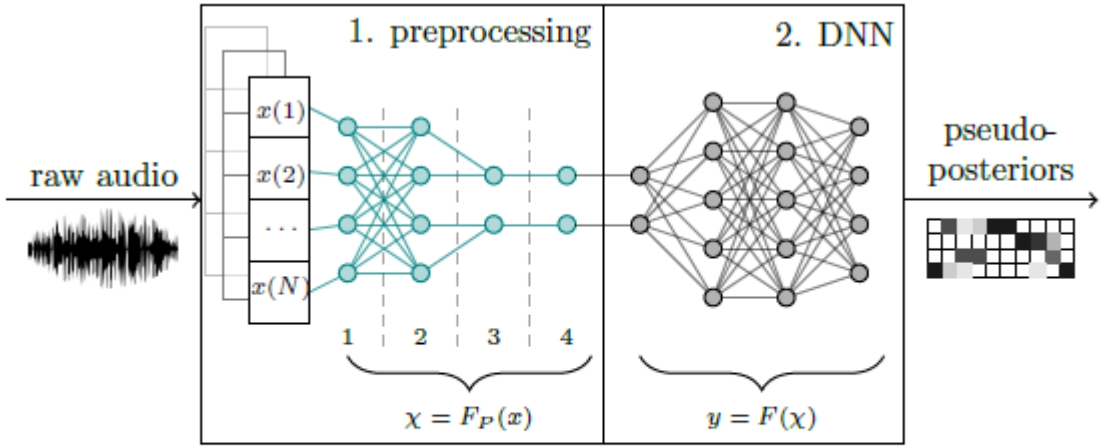


Fig. 4: For the creation of adversarial samples, we use an ASR system where the preprocessing is integrated into the DNN. Layers 1–4 represent the separate preprocessing steps.

基于Kaldi的优化模型 —— Backpropagation

- All preprocessing steps are included in X = Fp(x) and return the input features X for the DNN

$$\nabla x = \frac{\partial L(y, y')}{\partial F(\chi)} \cdot \frac{\partial F(\chi)}{\partial F_P(x)} \cdot \frac{\partial F_P(x)}{\partial x},$$

- 1) Framing and Window Function

$$x_w(t, n) = x(t, n) \cdot w(n), \quad n = 0, \ldots, N-1,$$

with $t = 0, \ldots, T-1$. Thus, the derivative is just

$$\frac{\partial x_w(t, n)}{\partial x(t, n)} = w(n).$$

- 2) Discrete Fourier Transform

$$X(t, k) = \sum_{n=0}^{N-1} x_w(t, n) e^{-i2\pi \frac{kn}{N}}, \quad k = 0, \ldots, N-1. \qquad \frac{\partial X(t, k)}{\partial x_w(t, n)} = e^{-i2\pi \frac{kn}{N}}, \quad k, n = 0, \ldots, N-1.$$

基于Kaldi的优化模型 —— Backpropagation

- All preprocessing steps are included in X = Fp(x) and return the input features X for the DNN

$$\nabla x = \frac{\partial L(y, y')}{\partial F(\chi)} \cdot \frac{\partial F(\chi)}{\partial F_P(x)} \cdot \frac{\partial F_P(x)}{\partial x},$$

- 3）Magnitude

$$\begin{aligned} |X(t,k)|^2 &= a(t,k)^2 + b(t,k)^2, \\ \text{with} \quad a(t,k) &= \mathrm{Re}(X(t,k)), \\ b(t,k) &= \mathrm{Im}(X(t,k)), \end{aligned}$$

$$\nabla X(t,k) = \begin{pmatrix} \frac{\partial |X(t,k)|^2}{\partial \mathrm{Re}(X(t,k))} \\ \frac{\partial |X(t,k)|^2}{\partial \mathrm{Im}(X(t,k))} \end{pmatrix} = \begin{pmatrix} 2 \cdot \mathrm{Re}(X(t,k)) \\ 2 \cdot \mathrm{Im}(X(t,k)) \end{pmatrix}.$$

基于Kaldi的优化模型 —— Backpropagation

- All preprocessing steps are included in X = Fp(x) and return the input features X for the DNN

$$\nabla x = \frac{\partial L(y, y')}{\partial F(\chi)} \cdot \frac{\partial F(\chi)}{\partial F_P(x)} \cdot \frac{\partial F_P(x)}{\partial x},$$

- 3）Magnitude

$$\begin{aligned} |X(t,k)|^2 &= a(t,k)^2 + b(t,k)^2, \\ \text{with} \quad a(t,k) &= \mathrm{Re}(X(t,k)), \\ b(t,k) &= \mathrm{Im}(X(t,k)), \end{aligned}$$

$$\nabla X(t,k) = \begin{pmatrix} \frac{\partial |X(t,k)|^2}{\partial \mathrm{Re}(X(t,k))} \\ \frac{\partial |X(t,k)|^2}{\partial \mathrm{Im}(X(t,k))} \end{pmatrix} = \begin{pmatrix} 2 \cdot \mathrm{Re}(X(t,k)) \\ 2 \cdot \mathrm{Im}(X(t,k)) \end{pmatrix}.$$

基于Kaldi的优化模型 —— Backpropagation

- All preprocessing steps are included in X = Fp(x) and return the input features X for the DNN

$$\nabla x = \frac{\partial L(y, y')}{\partial F(\chi)} \cdot \frac{\partial F(\chi)}{\partial F_P(x)} \cdot \frac{\partial F_P(x)}{\partial x},$$

- 4）Logarithm

$$\chi = \log(|X(t,k)|^2)$$

$$\frac{\partial \chi}{\partial |X(t,k)|^2} = \frac{1}{|X(t,k)|^2}.$$

基于Kaldi的优化模型 —— Hearing Threshold

- we use the original audio signal to calculate the hearing thresholds H
- We limit the differences D between the original signal spectrum S and the modified signal spectrum M to the threshold of human perception for all times t and frequencies k

$$D(t,f) \leq H(t,k), \quad \forall t,k,$$

$$\text{with} \quad D(t,k) = 20 \cdot \log_{10} \frac{|S(t,k) - M(t,k)|}{\max_{t,k}(|\mathbf{S}|)}.$$

基于Kaldi的优化模型 —— Hearing Threshold

- We calculate the amount of distortion that is still acceptable via …

$$\mathbf{\Phi} = \mathrm{H} - \mathrm{D}.$$

- First, because the thresholds are tight, an additional variable λ is added, to allow the algorithm to differ from the hearing thresholds by small amounts

$$\mathbf{\Phi}^* = \mathbf{\Phi} + \lambda.$$

- a **negative** value for $\Phi^*(t,k)$ indicates that we crossed the threshold
- we want to avoid more noise for these time-frequency-bins, we set all $\Phi^*(t,k) < 0$ to zero.

$$\hat{\Phi}(t,k) = \frac{\Phi^*(t,k) - \min_{t,k}(\mathbf{\Phi}^*)}{\max_{t,k}(\mathbf{\Phi}^*) - \min_{t,k}(\mathbf{\Phi}^*)}, \quad \forall t,k.$$

基于Kaldi的优化模型 —— Hearing Threshold

- Using the resulting scaling factors $\hat{\Phi}(t,k)$ typically leads to good results, but especially in the cases where only very small changes are acceptable, this scaling factor alone is not enough to satisfy the hearing thresholds. Therefore, we use another, fixed scaling factor, which only depends on the hearing thresholds H. For this purpose, H is also scaled to values between zero and one, denoted by ^H

$$\hat{\Phi}(t,k) = \frac{\Phi^*(t,k) - \min_{t,k}(\Phi^*)}{\max_{t,k}(\Phi^*) - \min_{t,k}(\Phi^*)}, \quad \forall t,k.$$

$$\nabla X^*(t,k) = \nabla X(t,k) \cdot \hat{\Phi}(t,k) \cdot \hat{H}(t,k), \quad \forall t,k.$$

基于Kaldi的优化模型 —— Hearing Threshold

- raw_thresholds(x)
- [TH, Map, LTq] = Table_absolute_threshold(1, fs, 128); % Threshold in quiet
- [Flags, Tonal_list, Non_tonal_list] = Find_tonal_components(X, TH, Map, CB);
- Individual_masking_thresholds(X, Tonal_list, Non_tonal_list, TH, Map);
- LTg = Global_masking_threshold(LTq, LTt, LTn);

```
% Frequency | Crit Band Rate | Absolute threshold
TH = [
      86.13    0.850   25.87   ;    172.27    1.694   14.85   ;
     258.40    2.525   10.72   ;    344.53    3.337    8.50   ;
     430.66    4.124    7.10   ;    516.80    4.882    6.11   ;
     602.93    5.608    5.37   ;    689.06    6.301    4.79   ;
     775.20    6.959    4.32   ;    861.33    7.581    3.92   ;
     947.46    8.169    3.57   ;   1033.59    8.723    3.25   ;
    1119.73    9.244    2.95   ;   1205.86    9.734    2.67   ;
    1291.99   10.195    2.39   ;   1378.13   10.629    2.11   ;
```

基于DeepSpeech的AE优化模型 —— distortion

- We measure distortion in Decibels (dB): a logarithmic scale that measures the relative loudness of an audio sample:

$$dB(x) = \max_i 20 \cdot \log_{10}(x_i).$$

$$dB_x(\delta) = dB(\delta) - dB(x).$$

基于DeepSpeech的AE优化模型 —— optimization problem

- given a natural example x and any target phrase t,

$$\begin{aligned}&\text{minimize } |\delta|_2^2 + c \cdot \ell(x+\delta, t)\\&\text{such that } dB_x(\delta) \le \tau\end{aligned}$$

$$\begin{aligned}&\text{minimize } dB_x(\delta)\\&\text{such that } C(x+\delta) = t\\&\qquad x+\delta \in [-M, M]\end{aligned}$$

$$\text{minimize } dB_x(\delta) + c \cdot \ell(x+\delta, t)$$

$$\begin{aligned}&\text{minimize } |\delta|_2^2 + \sum_i c_i \cdot L_i(x+\delta, \pi_i)\\&\text{such that } dB_x(\delta) < \tau\end{aligned}$$

where the loss function $\ell(\cdot)$ is constructed so that $\ell(x', t) \le 0 \iff C(x') = t$. The parameter c trades off the relative importance of being adversarial and remaining close to the original example.

基于DeepSpeech的AE优化模型 —— beam-search decoder

- Greedy decoding

$$C_{\text{greedy}}(x) = \text{reduce}(\arg\max_{\pi} \Pr(\pi|f(x)))$$

$$C(x) = \arg\max_{p} \Pr(p|f(x)).$$

- Beam Search decoding

基于DeepSpeech的AE优化模型 —— loss function

- While CTC-loss is highly useful for training the neural network, we show that a carefully designed loss function allows generating better lower-distortion adversarial examples.

$$\text{CTC-Loss}(f(x), p) = -\log \Pr(p|f(x)).$$

$$L(x, \pi) = \sum_i \ell(f(x)^i, \pi_i).$$