Speak, Read and Prompt

High-Fidelity Text-to-Speech with Minimal Supervision

2023-06-01

Outline

Introduction

Recap

- **w2v-BERT**: Combining Contrastive Learning and MLM for Self-Supervised Speech Pre-Training
- **SoundStream**: An End-to-End Neural Audio Codec

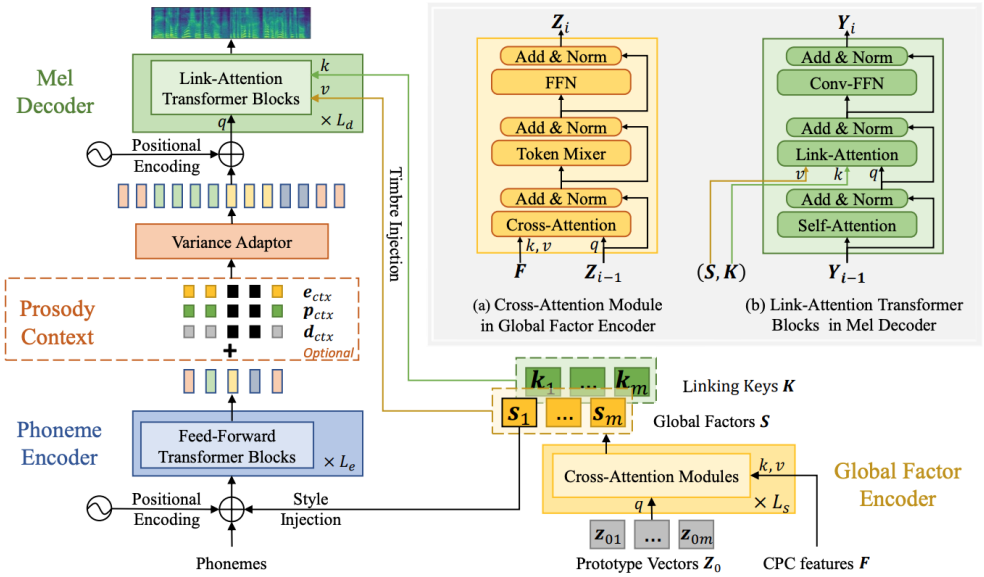
Application

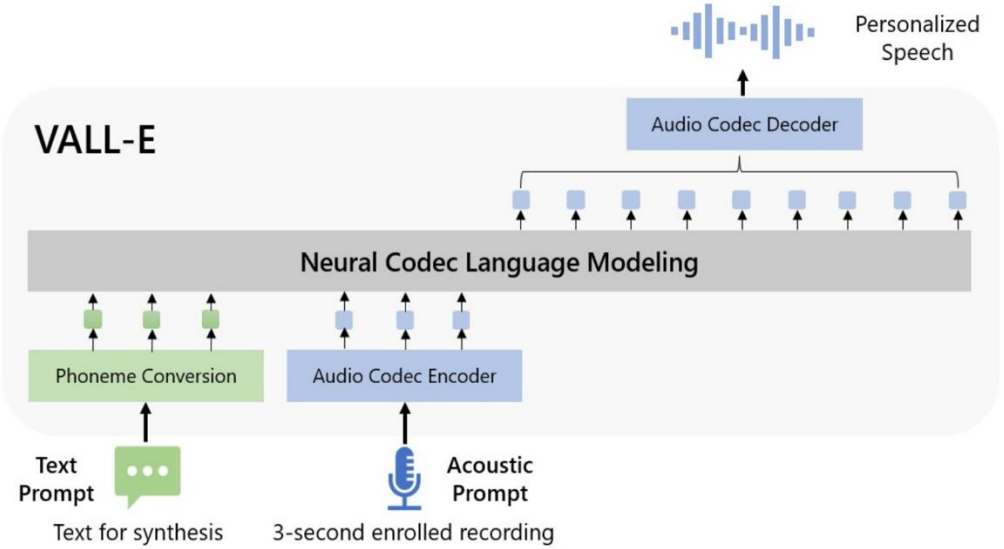
- ★ **SPEAR-TTS**: Speak, Read and Prompt: High-Fidelity Text-to-Speech with Minimal Supervision

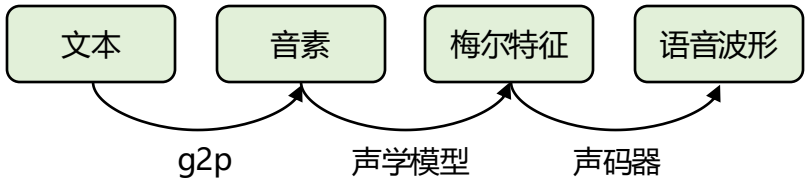
Conclusion

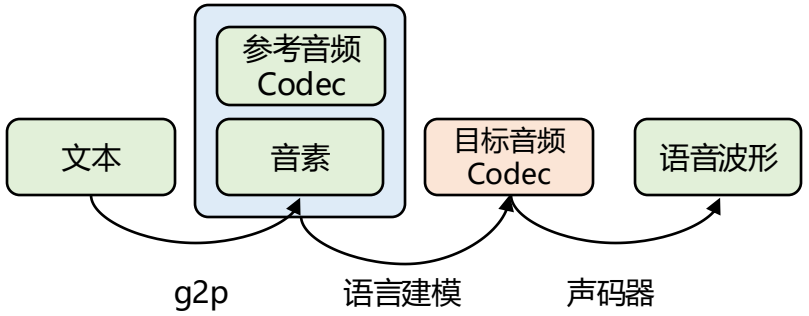
Introduction

目前典型的 TTS 模型



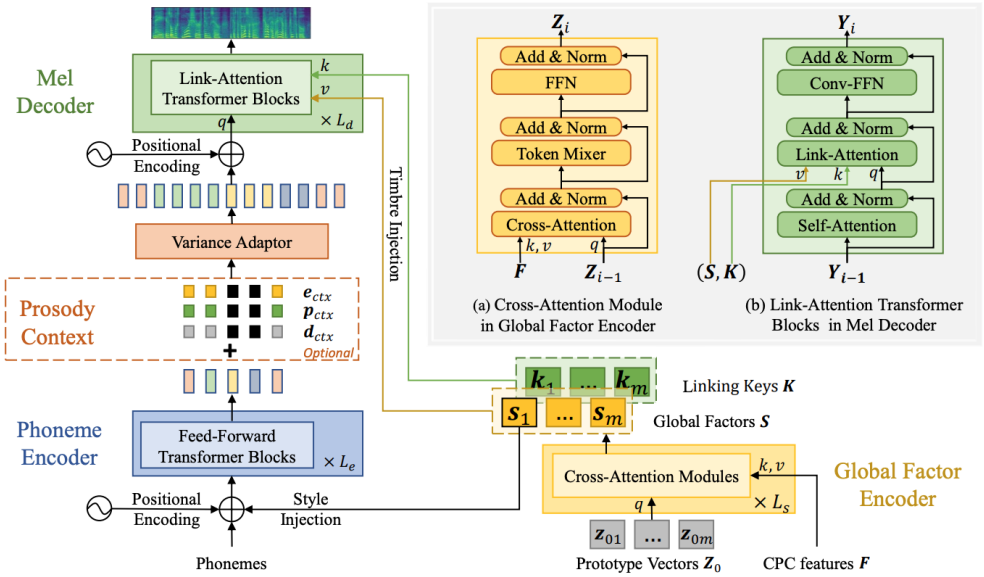


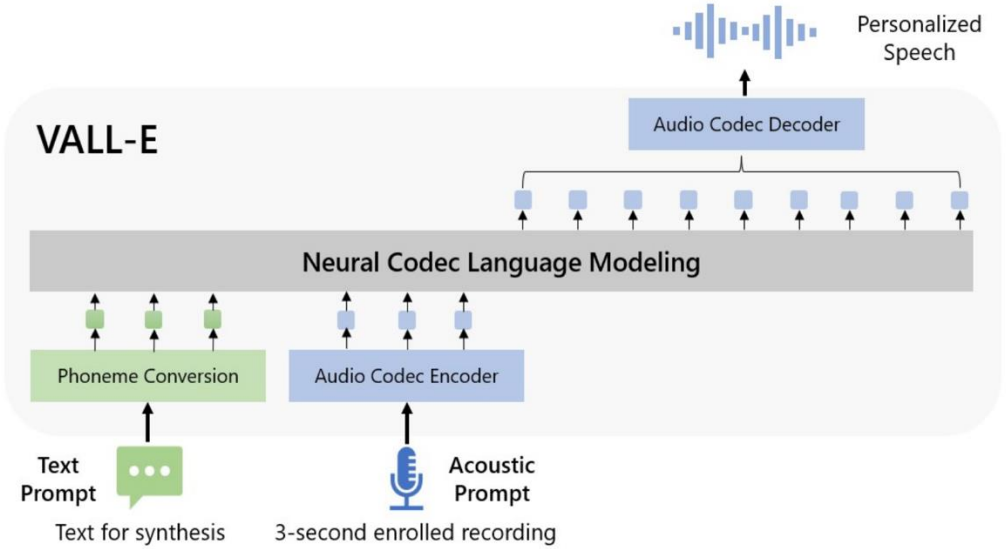




Introduction

目前典型的 TTS 模型

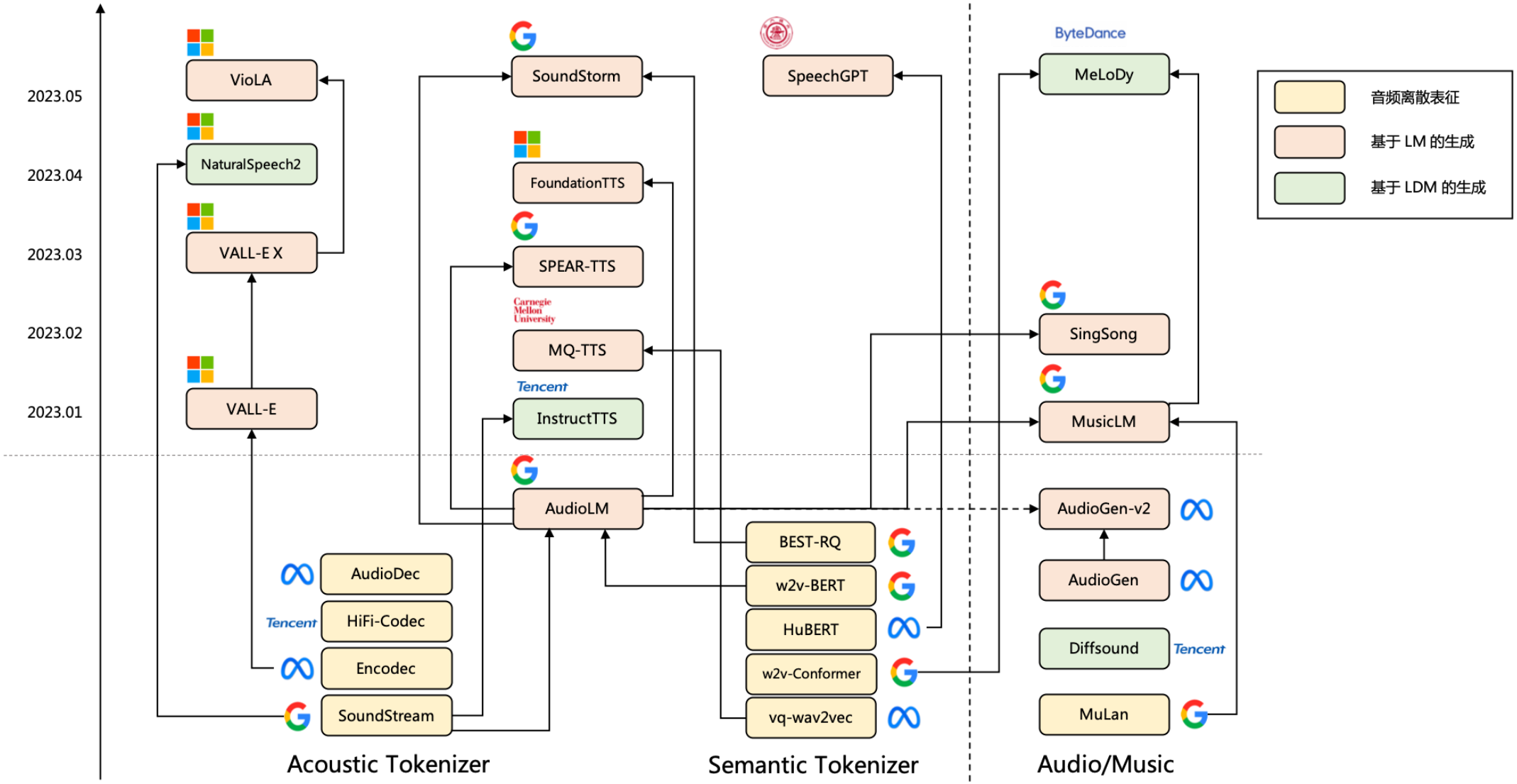




- 模型结构：较复杂，需要人工设计
- 中间特征：梅尔特征（连续空间）
- 建模方式：非自回归（时长预测）
- 参考音频输入：梅尔特征 / CPC
- 训练数据规模：<600 小时 (Libri-TTS)

- 模型结构：纯粹的 Transformer LM
- 中间特征：**音频 Codec（离散 Token）**
- 建模方式：自回归（语言模型）
- 参考音频输入：音频 Codec（离散 Token）
- 训练数据规模：**60k+ 小时 (Libri-Light) !!!**

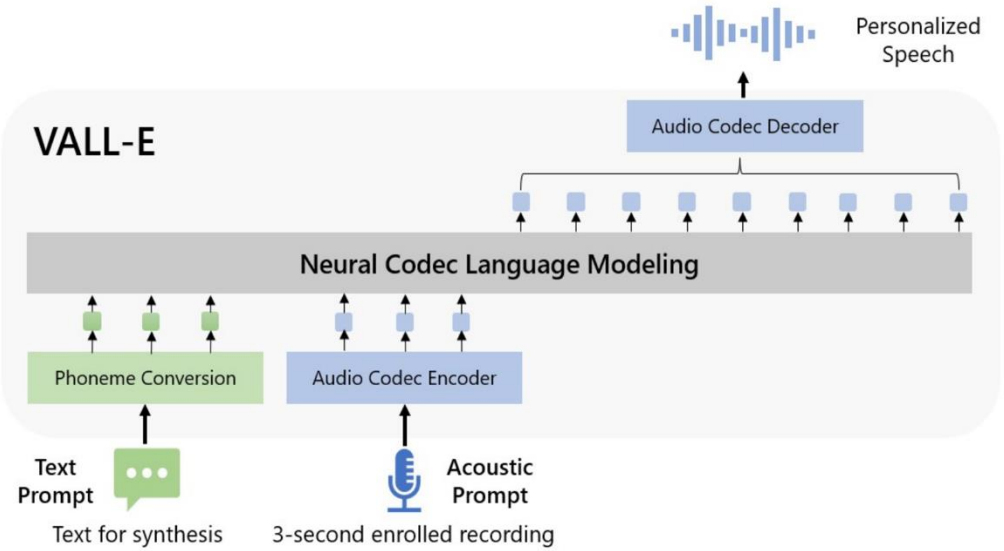
基于离散化语音表征的建模成为一种新范式



基于离散化语音表征的音频生成模型 · 一览表

Introduction: SPEAR-TTS

从 VALL-E 到 SPEAR-TTS



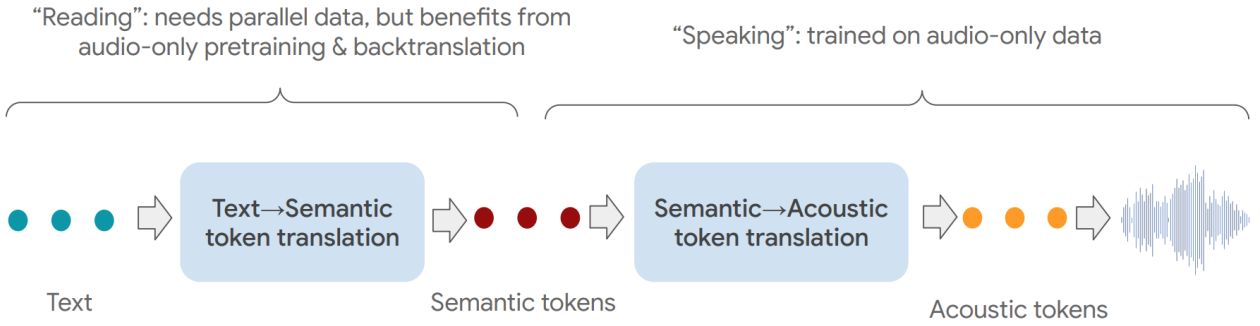


Figure 1: **SPEAR-TTS.** The first stage $\mathcal{S}_1$ ("reading") maps tokenized text to semantic tokens. The second stage $\mathcal{S}_2$ ("speaking") maps semantic tokens to acoustic tokens. Acoustic tokens are decoded to audio waveforms.

- VALL-E 使用大量的文本-语音数据对，硬 train 一发
- 问题：如何用更少的文本-语音数据对，达到高质量的 TTS?

VALL-E 两阶段建模

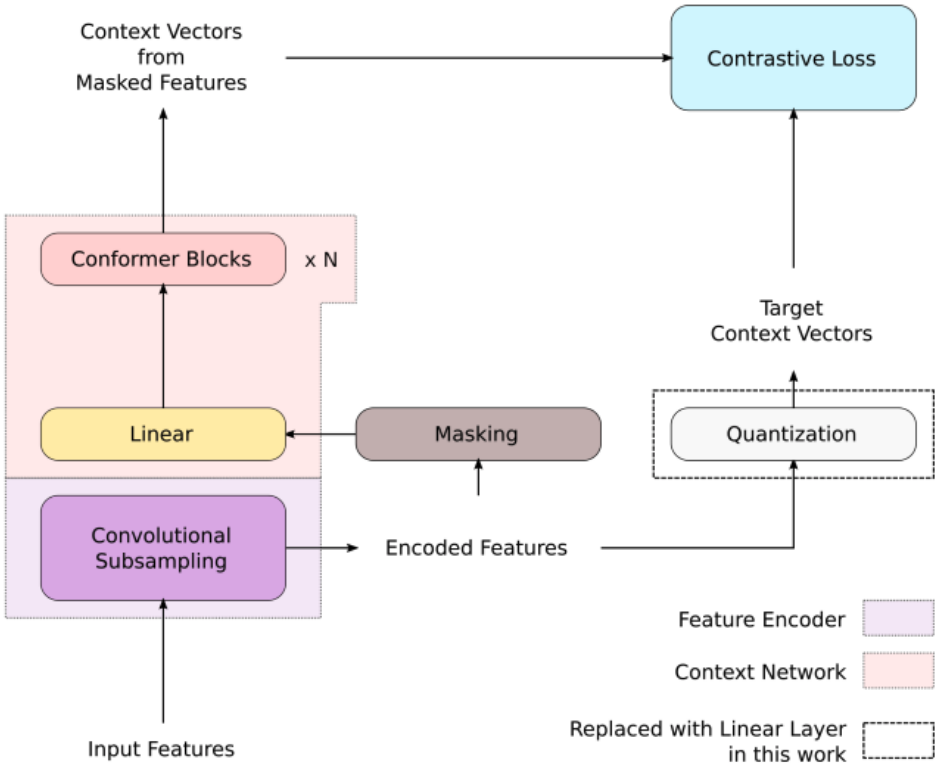
Text → **Acoustic Tokens** → Wave

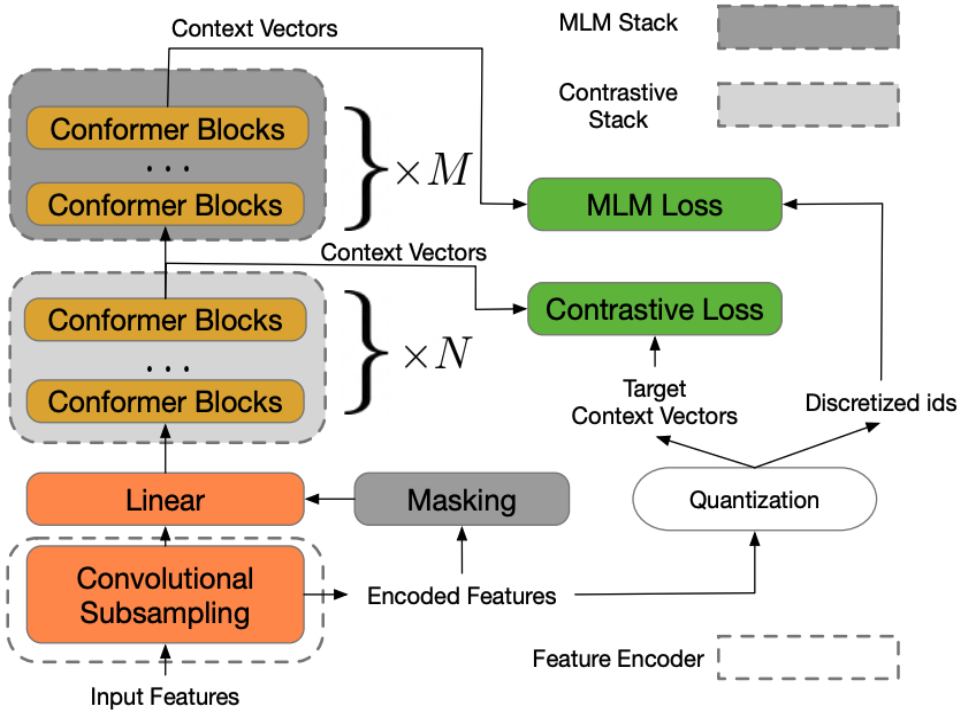
SPEAR-TTS 三阶段建模

Text → **Semantic Tokens** → **Acoustic Tokens** → Wave

Semantic Tokenizer: w2v-BERT

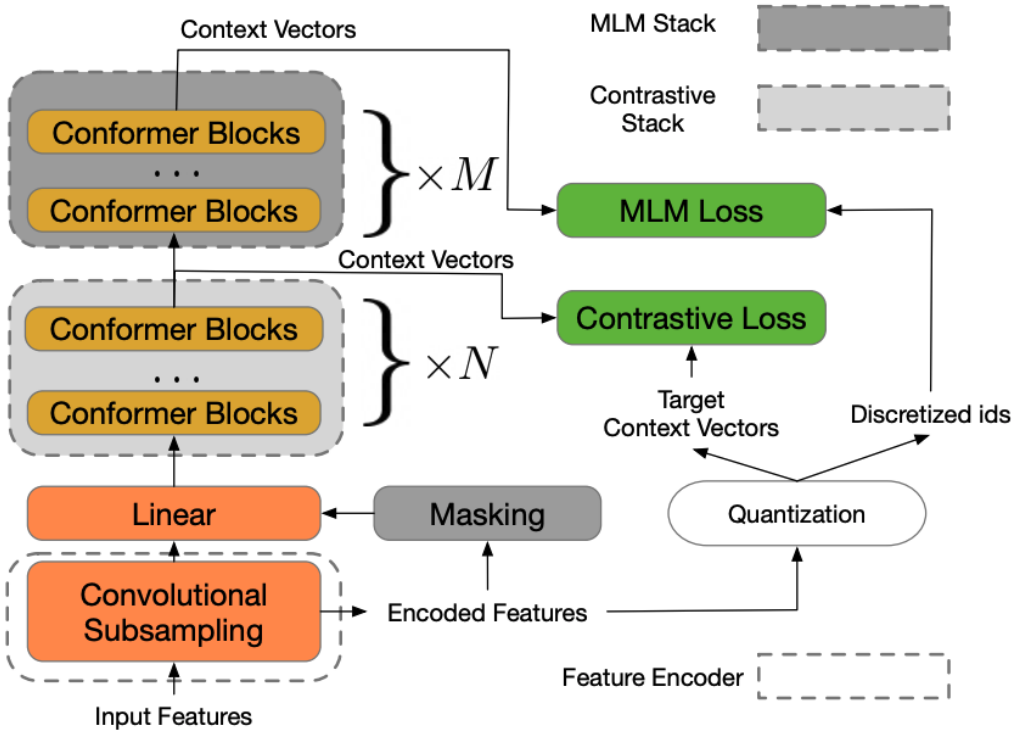
- 出发点：从语音中抽取出内容和语义信息
- 应用：语音识别/理解任务，偏重文本层面，不关注波形的细节信息





Semantic Tokenizer: w2v-BERT

- 出发点：从语音中抽取出内容和语义信息
- 应用：语音识别/理解任务，偏重文本层面，不关注波形的细节信息

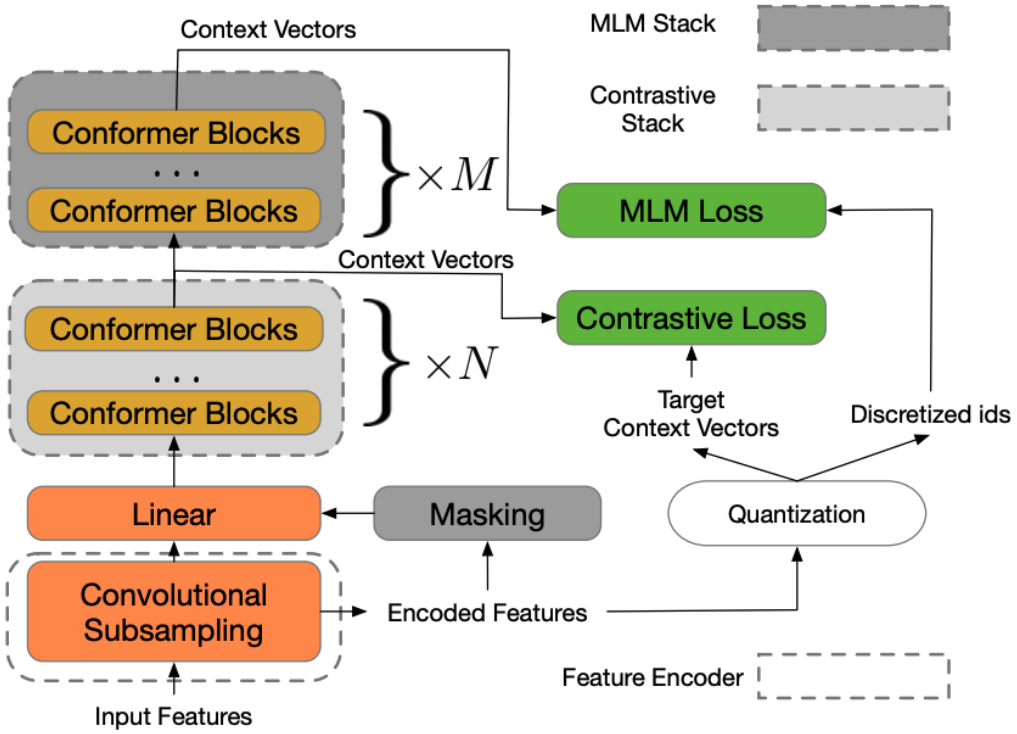


- 思想：MLM/对比学习，两个预训练目标（多任务），提高预训练效果
- 模型参数：M=12，N=12，总参数量 0.6B (XL)

Method	Unlabeled Data (hrs)	AM Size (B)	LM Size (B)	No LM dev	No LM dev-other	No LM test	No LM test-other
Trained from Scratch							
Conformer L [21]*	N/A	0.1	0.1	1.9	4.4	2.1	4.3
Self-training Only							
Conformer L with NST [21]	60k	0.1	0.1	1.6	3.3	1.7	3.5
Pre-training Only							
wav2vec 2.0 [22]	60k	0.3	$> 0.4^{\dagger}$	2.1	4.5	2.2	4.5
HuBERT Large [25]	60k	0.3	–	–	–	–	–
HuBERT X-Large [25]	60k	1.0	–	–	–	–	–
w2v-Conformer XL [21]	60k	0.6	0.1	1.7	3.5	1.7	3.5
w2v-Conformer XXL [21]	60k	1.0	0.1	1.6	3.2	1.6	3.3
w2v-BERT XL (Ours)	60k	0.6	0.1	**1.5**	2.9	**1.5**	2.9
w2v-BERT XXL (Ours)	60k	1.0	0.1	**1.5**	**2.7**	**1.5**	**2.8**

Semantic Tokenizer: w2v-BERT

- 出发点：从语音中抽取出内容和语义信息
- 应用：语音识别/理解任务，偏重文本层面，不关注波形的细节信息



Semantic Token 提取

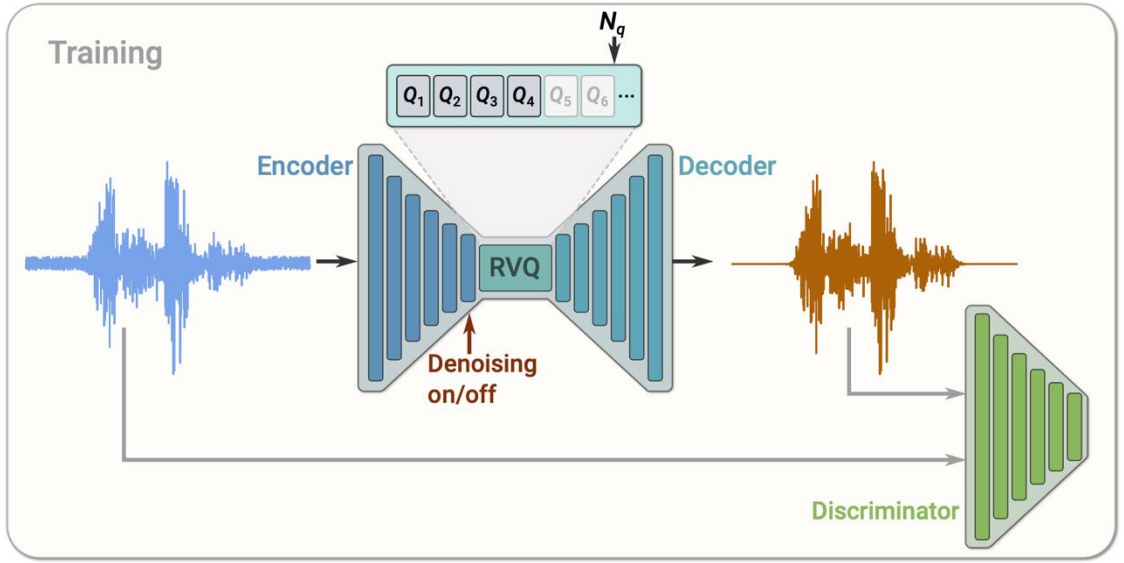
- 抽取位置：MLM 模块的第 7 层输出
- 先对输出 embedding 进行正则化（均值为 0，方差为 1）
- 再预设聚类个数 K（1024）进行 k-means 聚类
- 聚类后类别 id 作为离散的语义符号（**Semantic Token**）

Acoustic Tokenizer: SoundStream

设计初衷：低码率的端到端神经网络编解码器

- 目标：对语音进行压缩，压缩后的表征能够尽可能还原波形

- **码率/比特率 (bit rate)**：每秒钟数据需要传输的比特数。
- 比如：采样率为 24kHz 的 PCM，每个采样点用 16 bit 有符号整数存储，码率为 24kHz × 16 bit = 384 kbps。



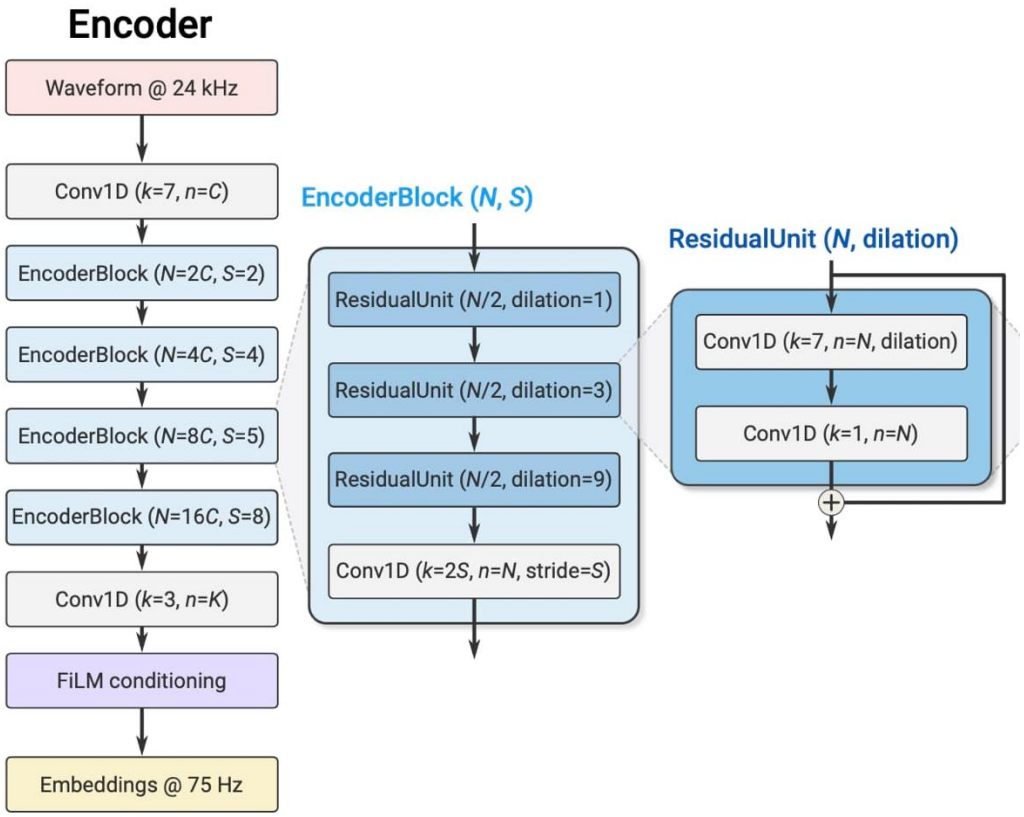

```
Input File     : 'SSB06320260.wav'
Channels       : 1
Sample Rate    : 24000
Precision      : 16-bit
Duration       : 00:00:05.67 = 135976 samples
File Size      : 272k
Bit Rate       : 384k
Sample Encoding: 16-bit Signed Integer PCM
```

SoundStream 核心步骤

- 编码器 Encoder 压缩得到 embedding
- VQ 计算得到距离最近的码本向量的整数 id 序列
- 将整数 id 序列传送到解码器 Decoder
- 解码器端根据码本向量「还原」出 embedding
- Decoder 恢复出语音波形

Acoustic Tokenizer: SoundStream

模块一：编码器（Encoder）



Encoder

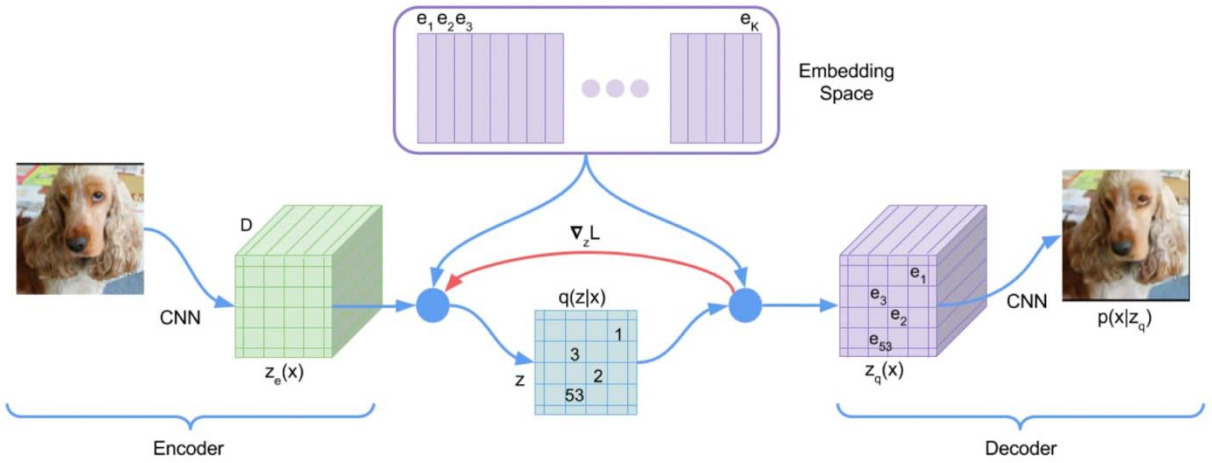
- 作用：对语音信号的信息进行压缩，降低传输带宽要求
- 网络结构设计：
  - 降采样倍数逐渐增加：320 倍降采样
  - 空洞卷积扩大深层一维卷积的感受野
- 输入：24kHz 的一维采样点数值序列
- 输出：75Hz 的 K 维向量序列

问题：帧率降低后码率是否降低？

- 输出 embedding：K = 1024，用 float32 存储
- 码率：75 Hz × 1024 × 32bit = 2.4 Mbps
- 结论：如果直接用Encoder 输出的连续空间隐层表征，码率过高
- 假设网络带宽要求音频比特率为 6kbps
- 每帧的 bit 数： 6kbps / 75Hz = 80 bit
- 如果传输浮点数：80bit / 32bit = 2.5
- **如果传输整数：能表示 2^80 范围内的整数**

Acoustic Tokenizer: SoundStream

模块二：VQ（Vector Quantization）



使用一组有限的向量集合，来表示该维度的所有向量

- codebook (码本)：有限的向量集合
- centroid：码本中的向量
- codebook size (码本大小)：codebook 集合中向量的个数
- Tokenizer：对于一个向量，码本中与之欧式距离最近的向量 id（整数）

回顾：VQ 的几个问题

1. argmin 不可导：Straight-Through Estimator

$$z_q(x) = e_k, k = arg\min_j ||z_e(x) - e_j||_2, j = 1, 2, \ldots, N$$

2. codebook 初始化

- 第一个 batch 数据 Encoder 的 embedding 进行 k-means 聚类，使用聚类中心作为各码本向量的初始值

3. 损失函数

$$L_{total} = L_{rec} + ||sg[z_e(x)] - e||_2^2 + ||z_e(x) - sg[e]||_2^2$$

第一项用来训练encoder和decoder. 从上面图中的红线可以看出, bp的时候 $z_q(x)$ 的梯度直接copy给 $z_e(x)$, 而不给codebook里的embedding, 所以这一项只训练encoder和decoder.

第二项叫codebook loss, 只训练codebook, 让codebook中的embedding向各自最近的$z_e(x)$靠近.

第三项叫commitment loss, 只训练encoder, 目的是encourage the output of encoder to stay close to the chosen codebook vector to prevent it from fluctuating too frequently from one code vector to another, 即防止encoder的输出频繁在各个codebook embedding之间跳.

Acoustic Tokenizer: SoundStream

模块二：VQ（Vector Quantization）

VQ （Vector Quantization）

- Encoder 每帧（帧率 75Hz）输出一个 embedding 向量
- 每帧的 bit 数： 6kbps / 75Hz = 80 bit
- 80bit 能够表示的整数 id 个数： 2^80，建模 2^80个码本向量？！
- 问题：码本过大，模型无法学习

RVQ （Residual Vector Quantization）

- Nq=8 个 VQ 量化层，每层用 10 bit 表示 Token 数值范围，
- 对应的每层码本向量为 2^10 = 1024 个码本向量

Algorithm 1: Residual Vector Quantization

Input: $y = \text{enc}(x)$ the output of the encoder, vector quantizers Q_i for $i = 1..N_q$

Output: the quantized $\hat{y}$

$\hat{y} \leftarrow 0.0$

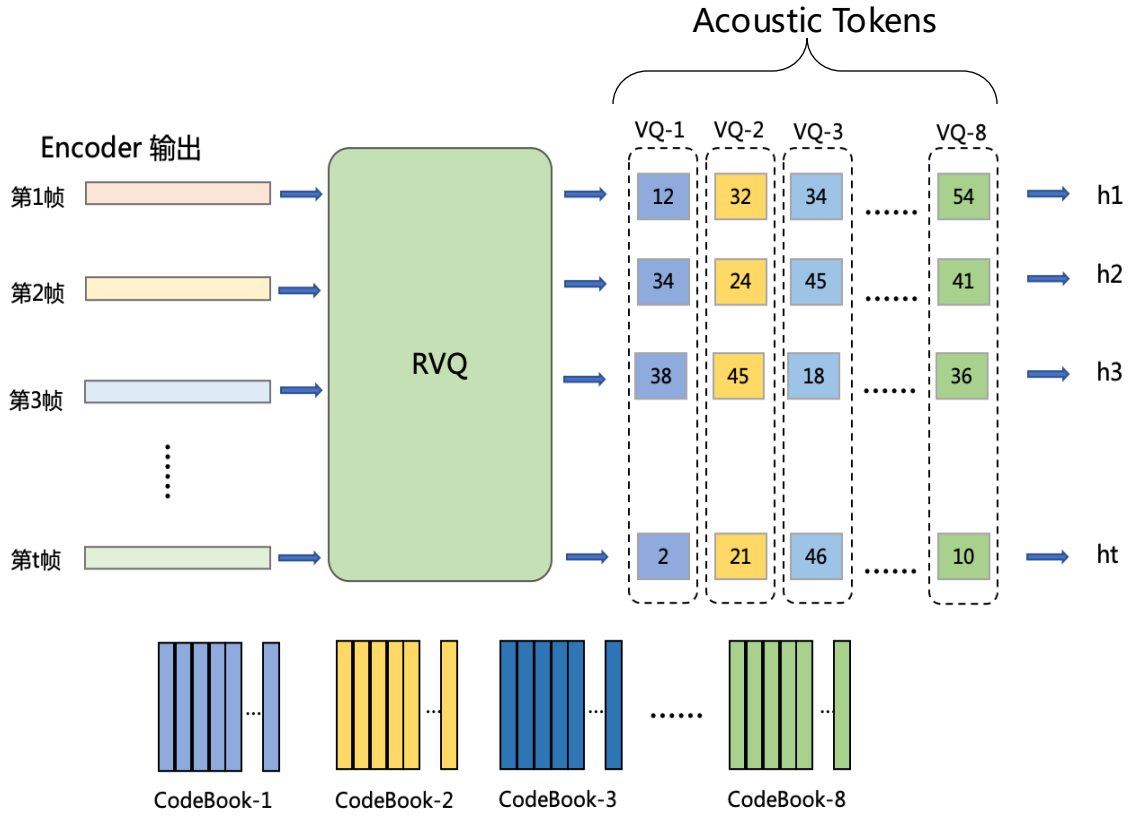
residual $\leftarrow y$

for $i = 1$ *to* N_q **do**

  $\hat{y}$ += Q_i(residual)

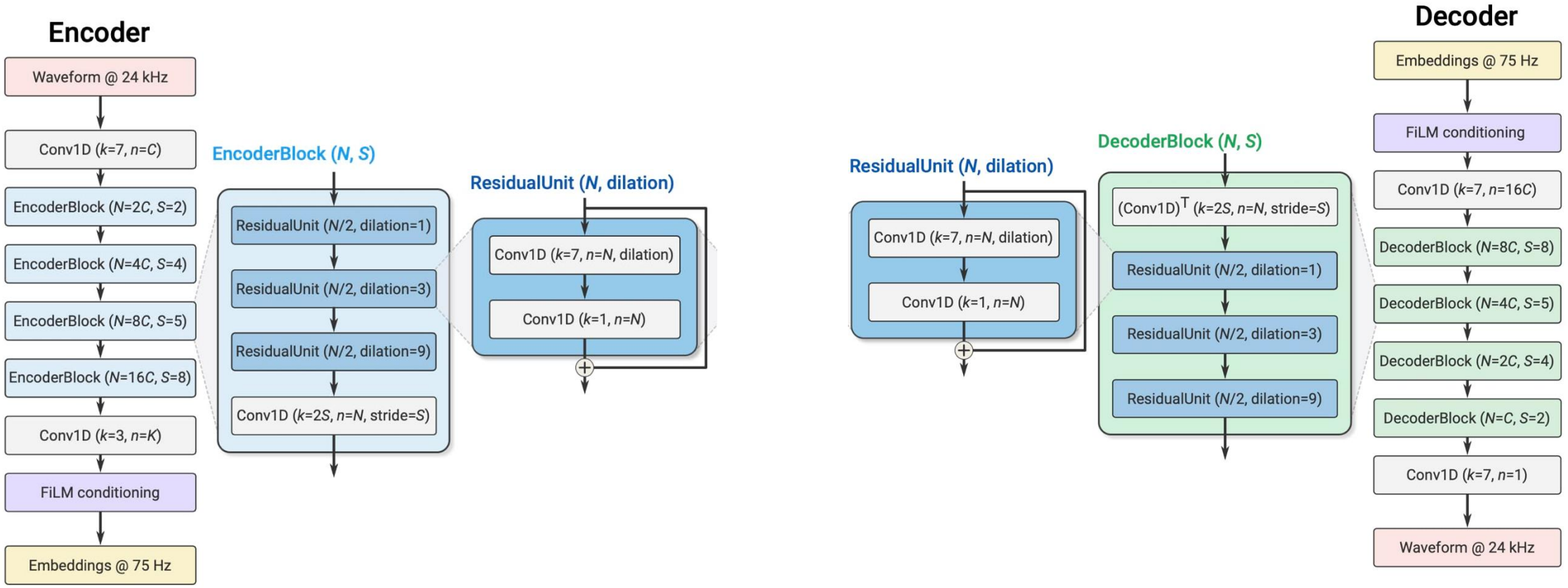
  residual −= Q_i(residual)

return $\hat{y}$



Acoustic Tokenizer: SoundStream

模块三：解码器（Decoder）



Acoustic Tokenizer: SoundStream

模块四：判别器（Discriminator）

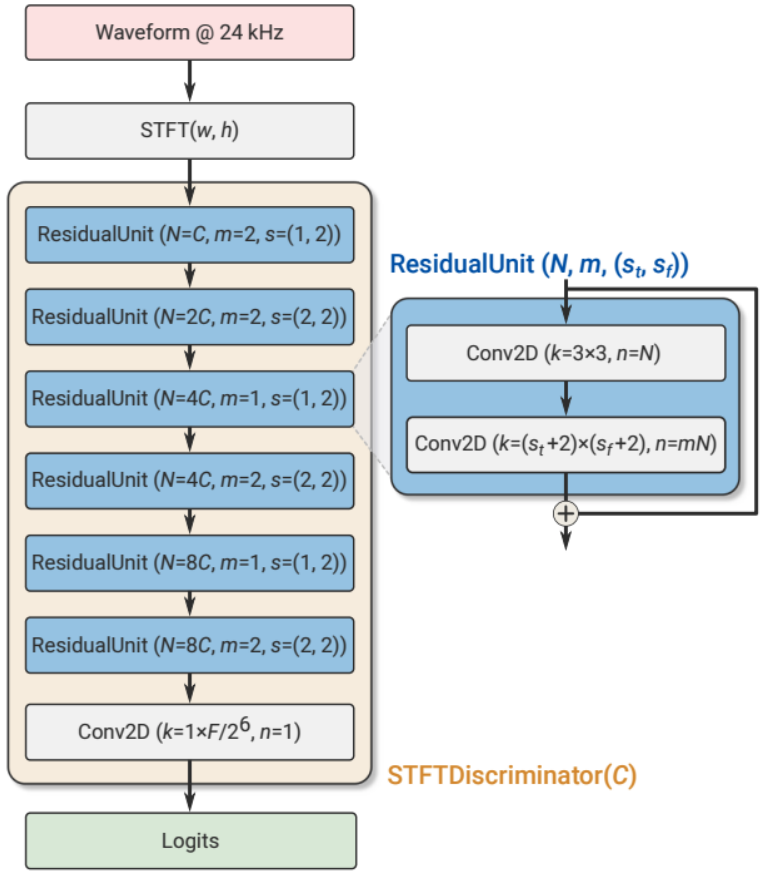
如何尽可能恢复出语音的原始波形?

采用 GAN 的思想：

1. 采用的 GAN 的训练方式提高生成语音的质量
2. 将整个编码器+RVQ+解码器视为生成器
3. 加入判别器区分原始语音和解码后生成的语音

频域判别器：STFT 判别器

- 输入 24kHz 原始波形进行 STFT 短时傅里叶变换
  - W=1024，H=256
- STFT 得到二维频域输出，T/H × F（帧数×频域特征维度）
- 时域降采样 8 倍，频域降采样 64 倍
- 输出 logits 表示当前输入是真实语音还是生成语音

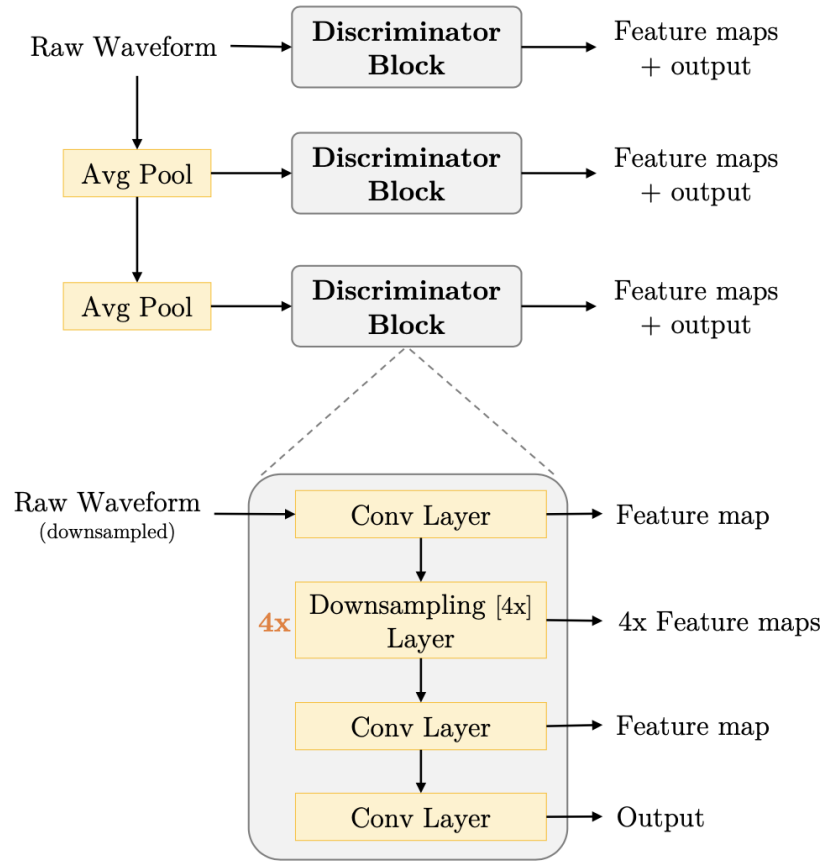


Acoustic Tokenizer: SoundStream

模块四：判别器（Discriminator）

时域判别器：多尺度判别器（来源：MelGAN/HiFi-GAN）

- 3 个 Discriminator Block 结构相同
- 三个不同尺度的判别器分别为 D1、D2、D3
  - 输入的音频相当于是不同的采样率
  - D1 处理的是原始音频帧率的采样点
  - D2 处理的是 2 倍降采样之后的音频采样点
  - D3 处理的 4 倍降采样后的采样点
- 每个判别器都会输出当前输入波形（降采样后）是真实语音还是生成语音



Acoustic Tokenizer: SoundStream

训练目标汇总

Generator：编码器 + RVQ + 解码器

$$L_{\mathcal{G}}^{vq} = L_{\mathcal{G}}^{rec} + ||sg[z_e(x)] - e||_2^2 + ||z_e(x) - sg[e]||_2^2$$

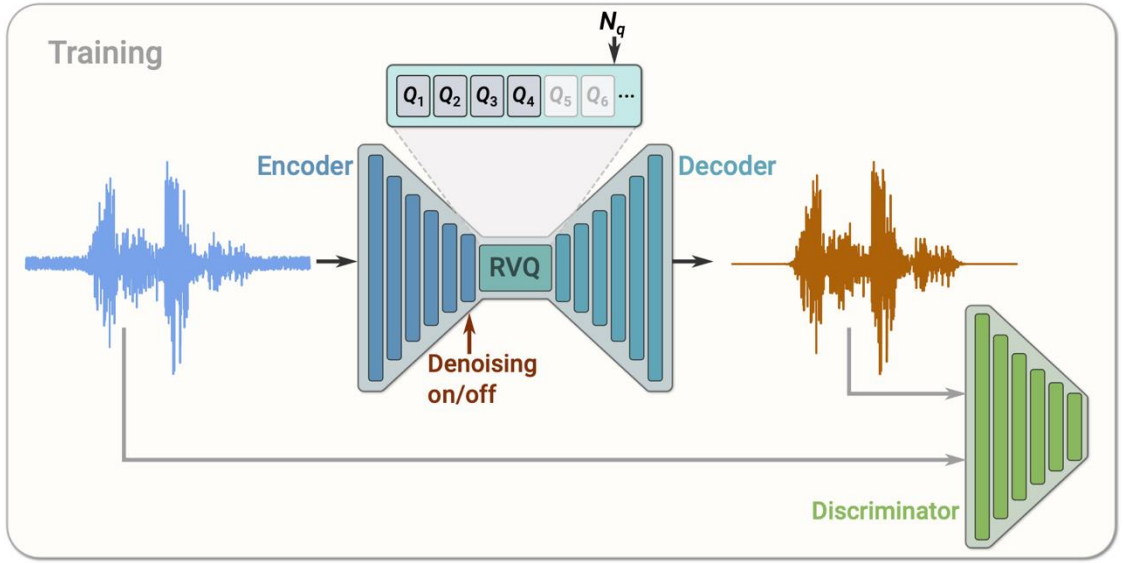
$$\mathcal{L}_{\mathcal{G}}^{\text{rec}} = \sum_{s \in 2^6, \ldots, 2^{11}} \sum_t \|\mathcal{S}_t^s(x) - \mathcal{S}_t^s(\mathcal{G}(x))\|_1 +$$

$$\alpha_s \sum_t \| \log \mathcal{S}_t^s(x) - \log \mathcal{S}_t^s(\mathcal{G}(x))\|_2$$

$$\mathcal{L}_{\mathcal{G}}^{\text{adv}} = E_x \left[\frac{1}{K} \sum_{k,t} \frac{1}{T_k} \max\left(0, 1 - \mathcal{D}_{k,t}(\mathcal{G}(x))\right) \right]$$

$$\mathcal{L}_{\mathcal{G}}^{\text{feat}} = E_x \left[\frac{1}{KL} \sum_{k,l} \frac{1}{T_{k,l}} \sum_t \left| \mathcal{D}_{k,t}^{(l)}(x) - \mathcal{D}_{k,t}^{(l)}(\mathcal{G}(x)) \right| \right]$$

$$\mathcal{L}_G = \lambda_{\text{adv}} \mathcal{L}_G^{\text{adv}} + \lambda_{\text{feat}} \cdot \mathcal{L}_G^{\text{feat}} + \lambda_{\text{rec}} \cdot \mathcal{L}_G^{\text{rec}}$$



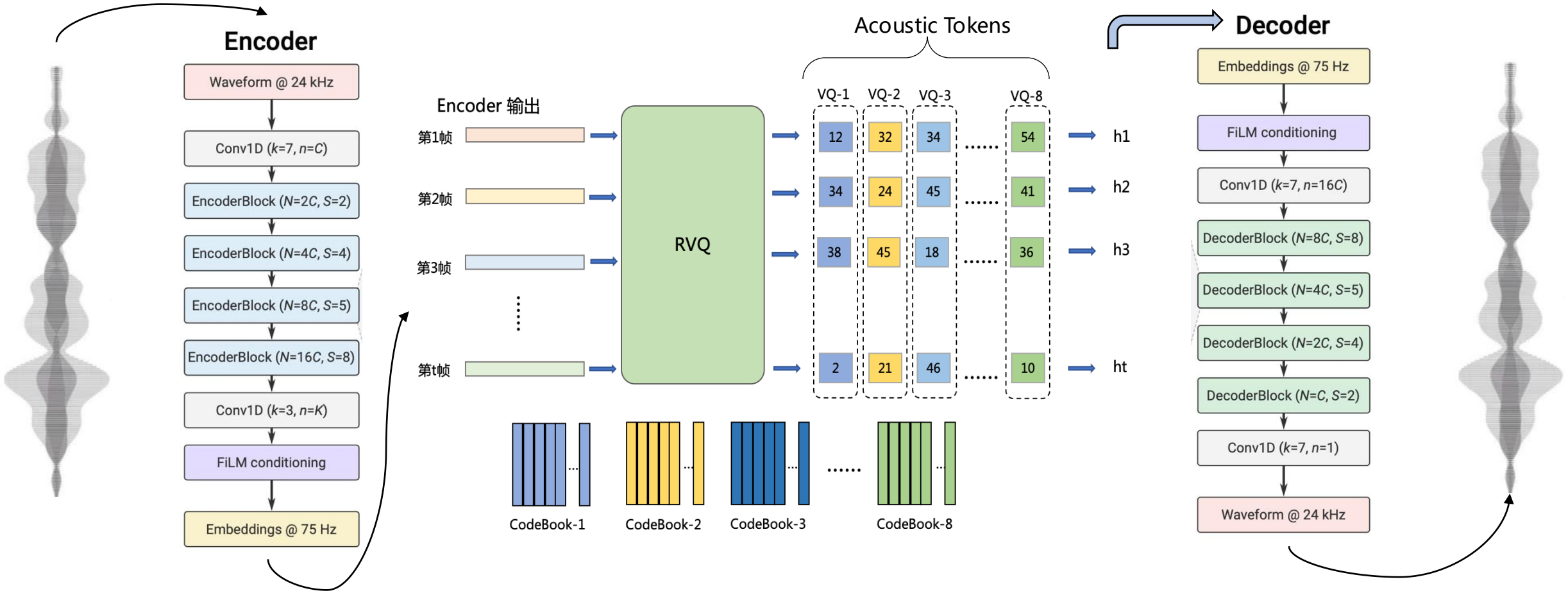
Discriminator：STFT 判别器：k=0 + 多尺度判别器：k=1,2,3

$$\mathcal{L}_{\mathcal{D}} = E_x \left[\frac{1}{K} \sum_k \frac{1}{T_k} \sum_t \max\left(0, 1 - \mathcal{D}_{k,t}(x)\right) \right] +$$

$$E_x \left[\frac{1}{K} \sum_k \frac{1}{T_k} \sum_t \max\left(0, 1 + \mathcal{D}_{k,t}(\mathcal{G}(x))\right) \right]$$

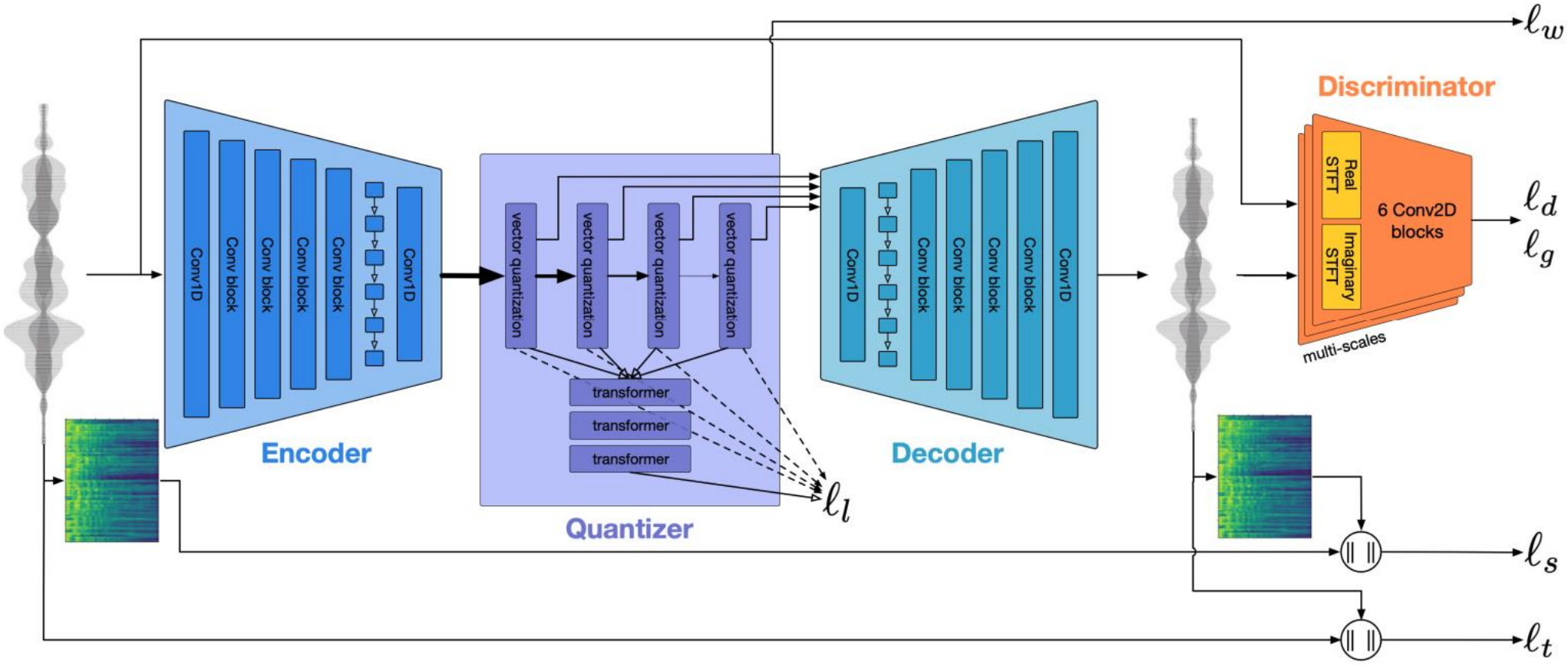
Acoustic Tokenizer: SoundStream

SoundStream 原理图



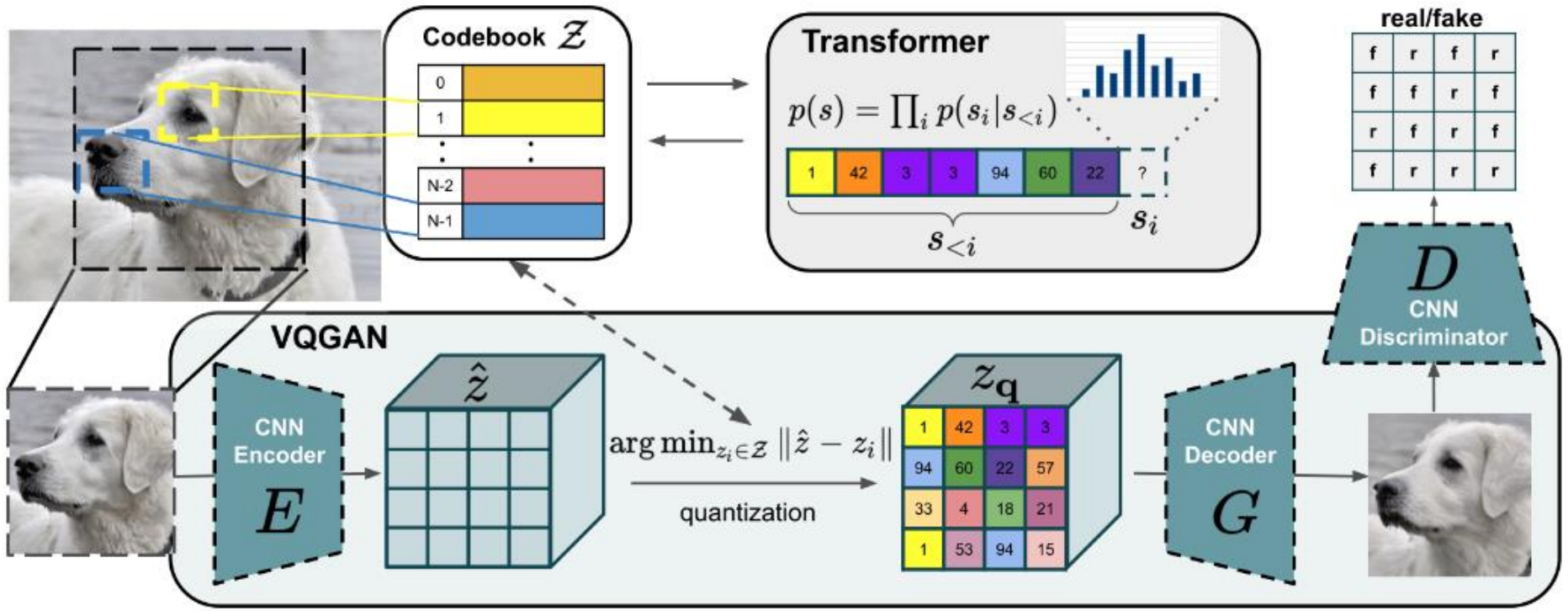
Acoustic Tokenizer: Encodec（VALL-E)

Encodec 原理图（VQ-GAN）



Acoustic Tokenizer: Encodec (VALL-E)

VQ-GAN



思考一：两种 Audio Token 的对比

	Semantic Token	Acoustic Token
输入特征	• 原始采样点：CPC/wav2vec2/Hubert • 梅尔特征：w2v-Conformer/w2v-BERT	原始采样点
训练方式	对比学习 / MLM	GAN
更倾向于的模态	文本	语音
编码的信息	语音的语义内容	波形细节信息

SPEAR-TTS

核心思路

出发点：VALL-E 数据量过大，小数据量下能否达到高质量 TTS?

1. S1 (Read)：文本 → Semantic Token，需要文本-语音数据对
2. S2 (Speaking)：Semantic Token → Acoustic Token，只需要语音数据
3. 音频解码：Acoustic Token → 语音波形，只需要语音数据

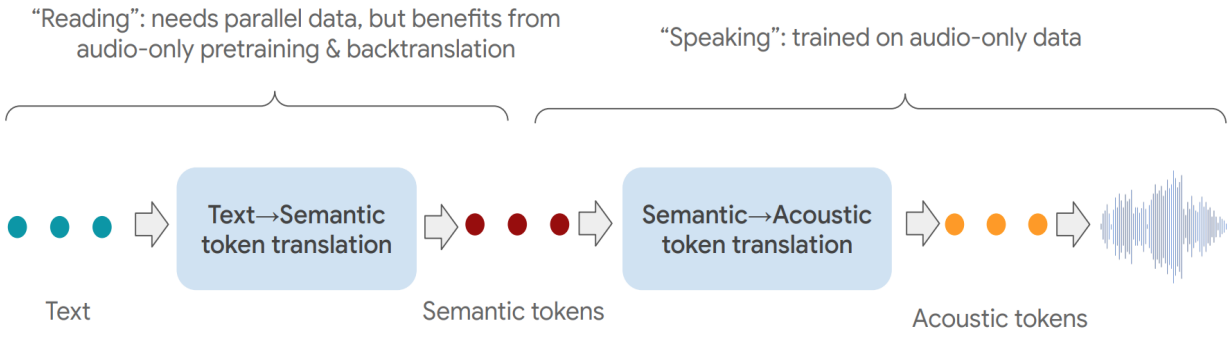


Figure 1: **SPEAR-TTS.** The first stage $\mathcal{S}_1$ ("reading") maps tokenized text to semantic tokens. The second stage $\mathcal{S}_2$ ("speaking") maps semantic tokens to acoustic tokens. Acoustic tokens are decoded to audio waveforms.

wav2vec-Bert 用于提取语义 token

- 语义内容的信息，更接近文本模态

SoundStream 提取细粒度声学 token

- 音频波形细节的信息，更接近语音模态

显著优势

1. 相比于 VALL-E 直接从文本到声学 token，SPEAR-TTS 以语义 token 作为中间特征，降低了建模难度
2. 语义 token → 声学 token 的建模过程不需要 pair 数据，只需要无标注语音，降低了对数据的要求
3. 语义信息基本去除了音色信息，因此 S1 阶段可以用单说话人的数据训练，S2 阶段基于 Prompt 控制合成语音的音色

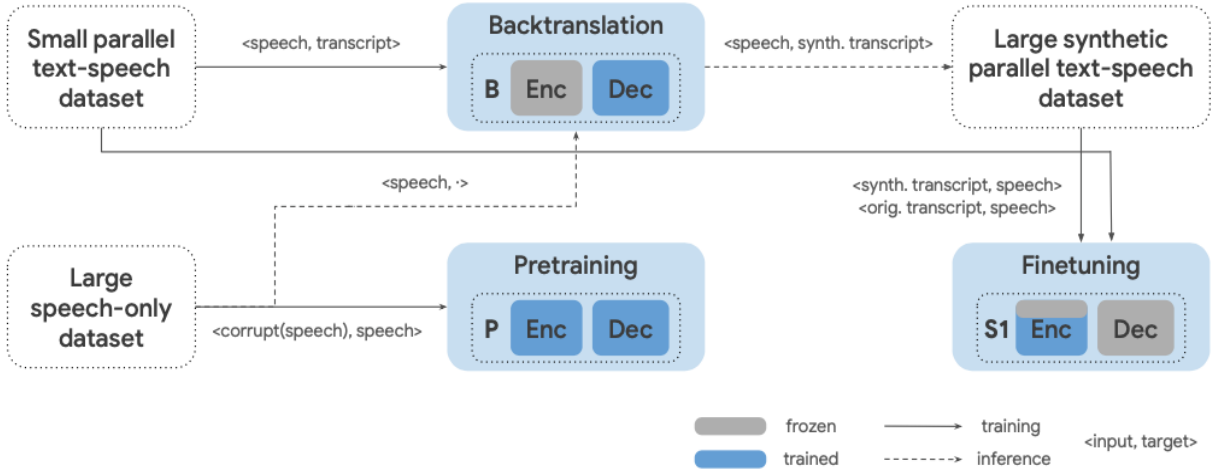
SPEAR-TTS

S1（第一阶段）：文本 → 语义 Token

- 建模方式：Transformer（Encoder-Decoder）
- Transformer 对 pair 数据量要求高，**如果避免小数据量时过拟合?**

优化方案一：BART/T5 预训练

- 第一步：预训练
  - 输入：破坏后（corrupted）的语义 token 序列
  - 输出：真实的语义 token 序列
  - 破坏：随机插入/删除/mask 一个或多个 token
- 第二步：使用少量文本-语音 pair 数据微调
  - 输入：**文本**；输出：**语义序列**
  - 固定 Encoder 深层参数和 Decoder 不涉及 Cross-Attention 的参数
  - 微调 Encoder 浅层参数和 Cross-Attention 的相关参数



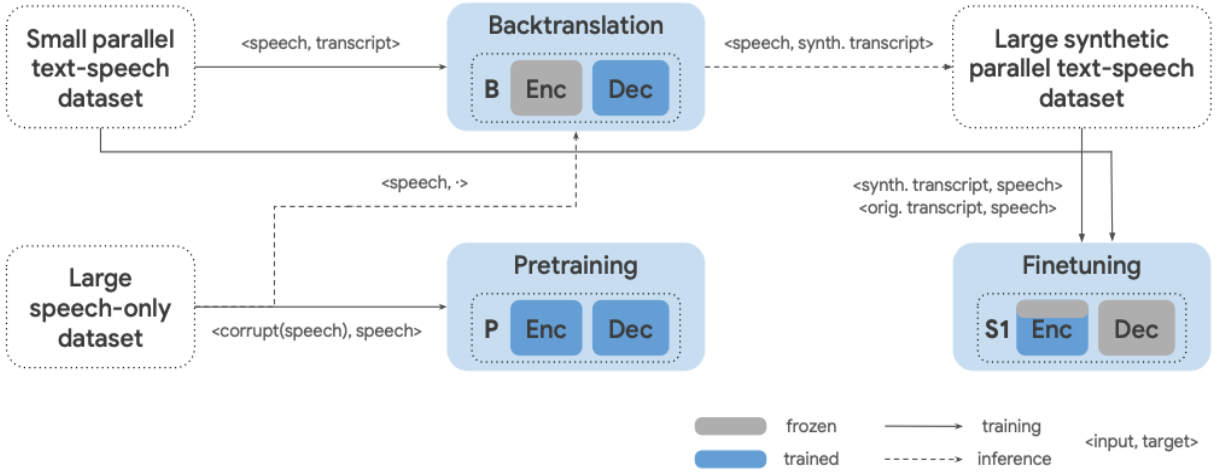
SPEAR-TTS

S1（第一阶段）：文本 → 语义 Token

- 建模方式：Transformer（Encoder-Decoder）
- Transformer 对 pair 数据量要求高，**如果避免小数据量时过拟合?**

优化方案二：Back-Translation

- 思想：使用现有 pair 数据训练 ASR 模型，对无标注的语音进行识别
- 第一步：方案一中预训练的 T5 模型，使用 pair 数据，输入是语义序列，输出改为文本
- 第二步：固定 Encoder 参数，只微调 Decoder 参数（相当得到一个 ASR 模型）



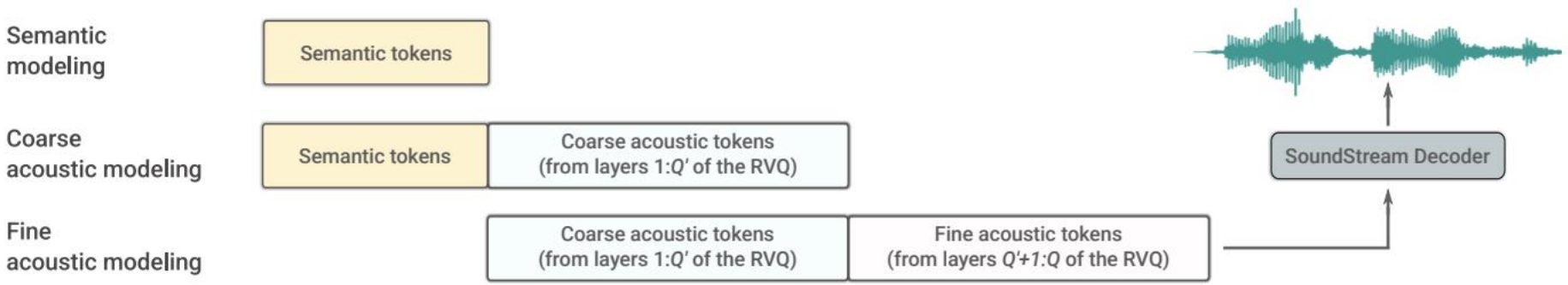
SPEAR-TTS

S2（第二阶段）：语义 Token → 声学 Token

- 语义 token并不包含说话人层面信息，直接预测声学 Token，合成语音的音色不可控（随机）
- 如何指定合成语音的音色？ → **AudioLM**

AudioLM 的思想：

- SoundStream 的 RVQ 属于分阶段量化，声学 token 携带的信息也具有一定的层次结构
- 前若干个 (Q') 量化器（粗粒度量化器，coarse quantizer）用来恢复说话人特性、录制环境等全局的粗粒度信息
- 剩下的量化器（细粒度量化器，fine quantizer）更侧重于波形的细节信息

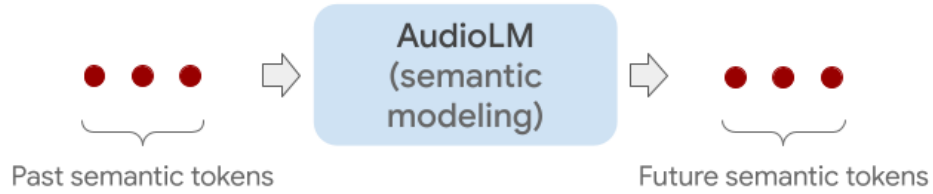


SPEAR-TTS

S2（第二阶段）：语义 Token → 声学 Token

问题设定

- 给定一段语音/音频作为 prompt，生成后续的音频
- t 时刻语义 token 为 z_t
- t 时刻声学 token (Q个)为 y_t^i (i=1,2,...,Q)

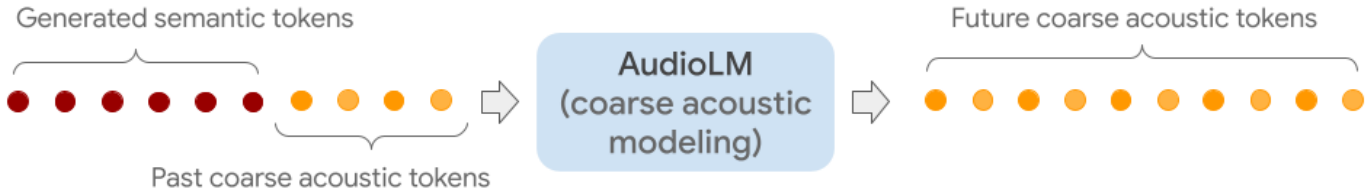


AudioLM 三步建模法

第一步：语义建模

$$p(z_t|z_{<t})$$

第二步：粗粒度声建模



阶段二主要建模$p(y_t^q|z, y_{<t}^{\leq Q'}, y_t^{<q}), q \leq Q'$，其中：

- z表示第一阶段得到的离散语义 token 序列；
- $y_{<t}^{\leq Q'}$表示之前时刻，前Q'个粗粒度量化器的声学 token 序列，并且将矩阵展开（flatten）
- $y_t^{<q}$表示当前时刻下，前$q-1(q \leq Q')$个量化器的 token 序列，同样进行展开。

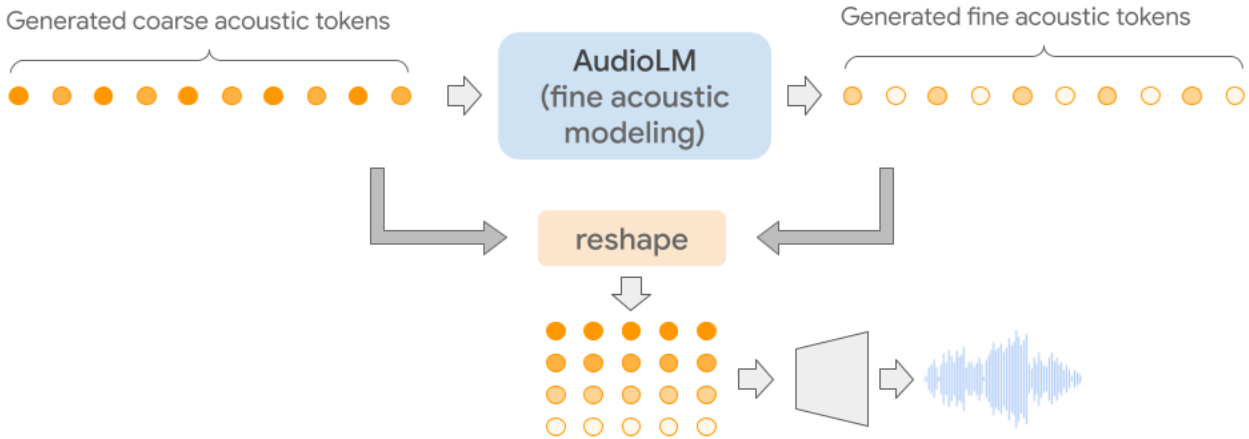
SPEAR-TTS

https://google-research.github.io/seanet/audiolm/examples/

S2（第二阶段）：语义 Token → 声学 Token

前提假设

- t 时刻语义 token 为 z_t
- t 时刻声学 token (Q个)为 y_t^i (i=1,2,...,Q)



AudioLM 三步建模法

第三步：细粒度声学建模

阶段三在细粒度量化器输出的声学 token 上进行建模，使用Q'个粗粒度量化器的 token 作为条件输入，对$p(y_t^q|y^{\leq Q'}, y_{<t}^{>Q'}, y_t^{<q}), Q' < q \leq Q$进行建模，其中：

- $y^{\leq Q'}$表示前Q'个量化器输出的 token 序列，将矩阵展开；
- $y_{<t}^{>Q'}$表示之前时刻，第$Q'+1$到第Q个量化器（一共Q个量化器）输出的 token 序列，并且展开；
- $y_t^{<q}$表示当前时刻下，前$q-1(Q' < q \leq Q)$个量化器的 token 序列，并且展开。

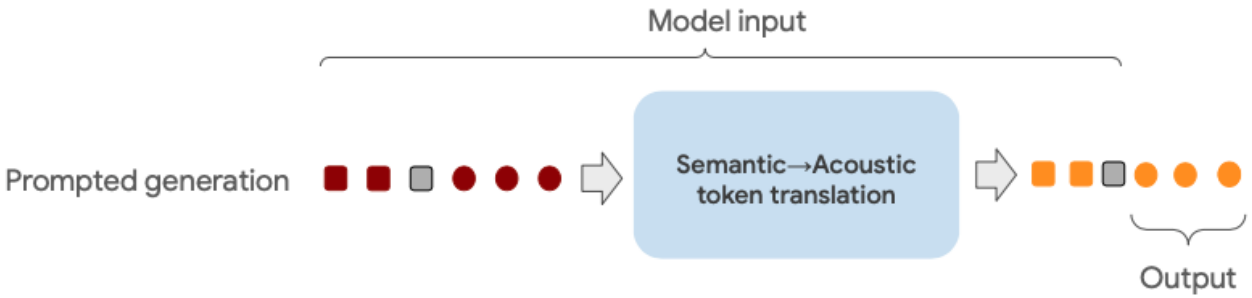
SPEAR-TTS

S2（第二阶段）：语义 Token → 声学 Token

训练过程：

同一条音频选取两个不重叠的语音片段，分别作为 prompt 和目标语音

- 输入：
  - prompt 的语义 token （w2v-BERT）
  - 目标语音的语义 token （w2v-BERT）
  - Prompt 的声学 token （SoundStream）
- 预测目标：
  - 目标语音的声学 token



推理过程

- 目标语音的语义 token 是使用第一阶段的文本 → 语义 token 模型预测得到的
- 基于预测的声学 token 序列，使用 SoundStream 的解码器生成语音

SPEAR-TTS

实验配置

SoundStream

- 训练数据：Libri-Light（60k 小时）
- 音频采样率：16 kHz
- 降采样倍数：320 倍
- RVQ 层数：3 层
- 每秒 token 个数：50 × 3 = 150
- 每层 codebook 大小：1024（10bit）
- 比特率：1500 bps

w2v-BERT

- 训练数据：Libri-Light（60k 小时）
- 音频采样率：16 kHz
- 降采样倍数：640 倍
- 每秒 token 个数：25 个
- embedding 抽取位置：MLM 模块第 7 层
- k-means 聚类个数：512 (9 bit)
- 比特率：25×9=225 bps

第一阶段：Text → Semantic Token

- T5 预训练+Back-Translation+微调
- 训练数据：Libri-Light（60k 小时）
- 模型结构：T5-large
- Back-Translation：Libri-TTS (550 小时，无标注)
- 微调数据：LJSpeech 单说话人（24 小时）

	Embed. dim.	FFN dim.	Head dim.	# heads
T5-small	256	512	64	6
T5-base	768	2048	64	12
T5-large	1024	2816	64	16

Table 9: **Architecture details.** We report details for T5-small, T5-base, and T5-large layers. The number of layers used is defined by a grid search (see Section 7).

SPEAR-TTS

实验配置

第二阶段：Semantic Token → Acoustic Token

- 训练数据：Libri-Light (60k 小时)
- 模型结构：12 层 Decoder-Only Transformer
  - head=12, d = 64，FFN 隐层 2048

实验基线

- FastSpeech2 + HiFi-GAN
- VALL-E

评测数据

- LibriSpeech test-clean 2700 条

评测指标

- CER：判断合成语音的正确性
- 音色一致性：合成语音与 prompt 属于同一个说话人的概率
- MOS：1-5

实验结果

Table 4: **Comparing voice preservation with baselines (cosine similarity).** Results for YourTTS and VALL-E are taken from (Wang et al., 2023, Table 2).

Model	Parallel training data	Cosine similarity
YourTTS	∼ 600 h	0.34
VALL-E	60,000 h	0.58
SPEAR-TTS	15 min	0.56

Table 6: **Mean Opinion Score (MOS) evaluation for prompted generation.** Prompts for both systems and samples for VALL-E are taken from the demo page of VALL-E. ± indicates 95% CI obtained by bootstrap.

System	VALL-E	SPEAR-TTS (15 min)
MOS	$3.35_{\pm 0.12}$	$\mathbf{4.75}_{\pm 0.06}$

SPEAR-TTS

https://google-research.github.io/seanet/speartts/examples/

实验结果

Table 1: **Intelligibility** of SPEAR-TTS and our baselines, depending on the training scenario and the amount of parallel data available from LJSpeech. We measure CER (%, lower is better) on LibriSpeech test-clean. ± indicates 95% CI obtained by bootstrap."×" indicates models that produce unintelligible speech.

		SPEAR-TTS			
		Training		Backtranslation	
Parallel training data	FastSpeech2-LR	from scratch (a)	Pretraining (b)	from scratch (c)	pretraining (d)
24 h	$1.99_{\pm 0.20}$	$3.67_{\pm 0.21}$	$2.38_{\pm 0.13}$	$2.26_{\pm 0.14}$	$2.06_{\pm 0.12}$
12 h	-	$4.31_{\pm 0.28}$	$2.54_{\pm 0.14}$	$2.27_{\pm 0.14}$	$2.03_{\pm 0.12}$
3 h	$2.52_{\pm 0.25}$	$20.1_{\pm 0.74}$	$3.07_{\pm 0.15}$	$2.21_{\pm 0.12}$	$2.01_{\pm 0.12}$
2 h	-	$24.7_{\pm 0.71}$	$3.73_{\pm 0.17}$	$2.22_{\pm 0.13}$	$2.09_{\pm 0.12}$
1 h	$2.74_{\pm 0.27}$	×	$5.51_{\pm 0.21}$	$2.23_{\pm 0.13}$	$2.16_{\pm 0.13}$
30 min	$3.18_{\pm 0.28}$	×	$21.3_{\pm 0.43}$	$2.52_{\pm 0.15}$	$2.20_{\pm 0.12}$
15 min	$4.90_{\pm 0.34}$	×	×	$2.88_{\pm 0.19}$	$2.21_{\pm 0.12}$

Table 5: **Mean Opinion Score (MOS) evaluation.** All compared systems are trained on subsets of LJSpeech (Ito and Johnson, 2017). ± indicates 95% CI obtained by bootstrap.

	FastSpeech2-LR			SPEAR-TTS	Ground-truth
Parallel training data	15 min	1 h	24 h	15 min	-
MOS	$1.72_{\pm 0.04}$	$2.08_{\pm 0.04}$	$2.11_{\pm 0.04}$	$\mathbf{4.96}_{\pm 0.02}$	$4.92_{\pm 0.04}$

思考

Semantic Token / Acoustic Token 对比
- 网络结构的设计
- 目标函数的不同
- 是否可以将两者结合

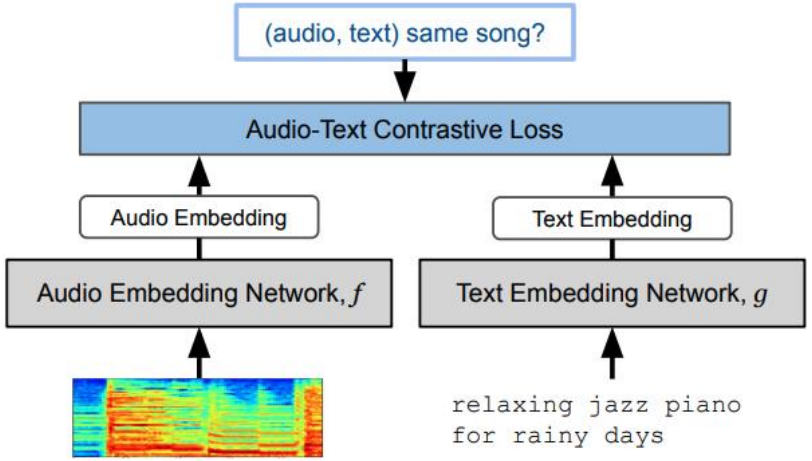
- 不使用自回归，MQ-TTS

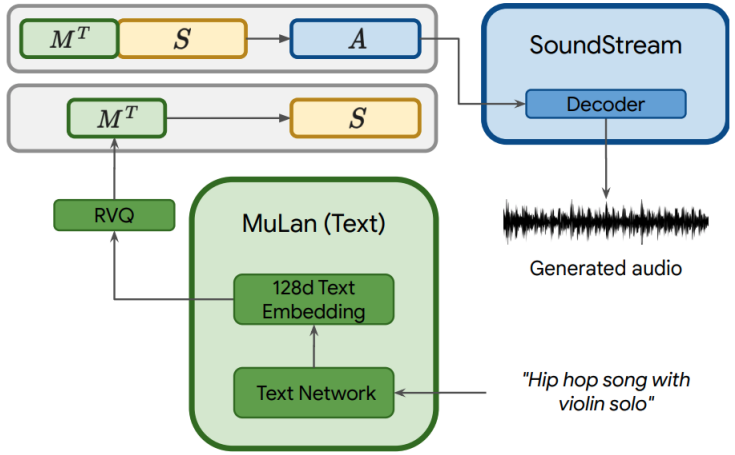
算力的提升带来技术思想的倒退?
- 单纯的 TTS 合成是否有必要走 ChatGPT 的路子
- 大数据量对于 TTS 任务的收益是什么?

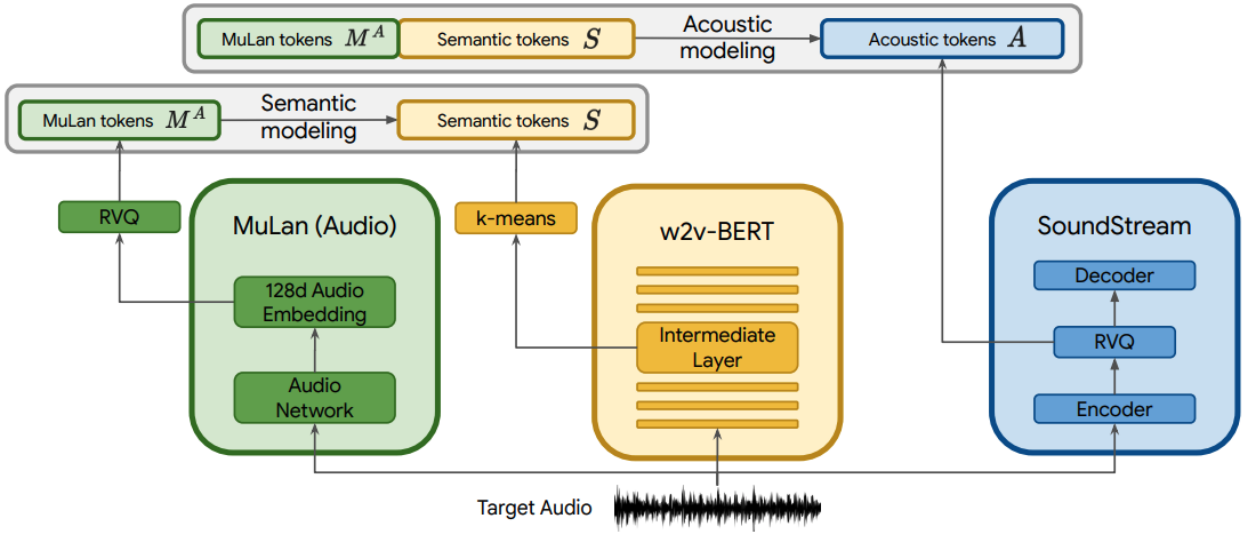
文本/语音/图像的多模态统一模型
- SpeechGPT
- VQ-GAN/MaskGIT/MUSE
- 训练不匹配问题

模型	语种	数据集	数据量（小时）
VALL-E	英文	Libri-Light	60k
VALL-E X	英文+中文	Libri-Light + WenetSpeech	60k+10k
NaturalSpeech2	英文	MLS-en	44k
MQ-TTS	英文	GigaSpeech	7k+
SoundStorm	英文	自有	100k

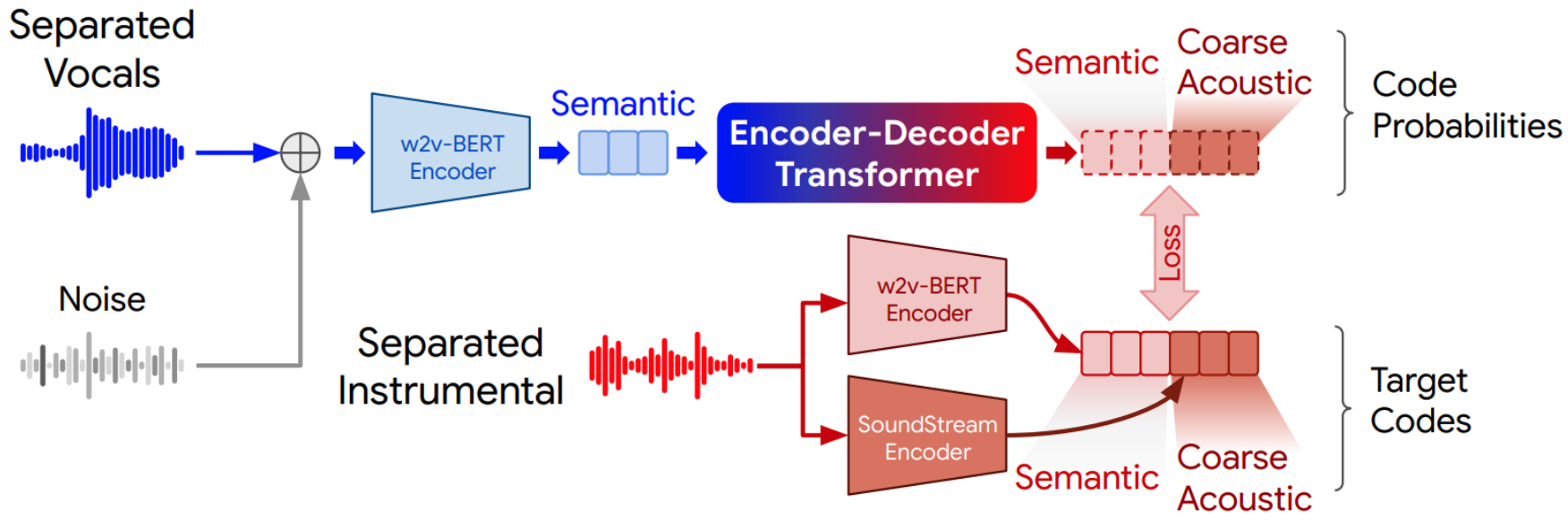
音乐生成：MusicLM







伴奏生成：MusicLM



Thanks!