FireRedTTS: A Foundation Text-To-Speech Framework for Industry-Level Generative Speech Applications

https://github.com/FireRedTeam/FireRedTTS

2024-09-25

Table of Contents

demo: https://fireredteam.github.io/demos/firered_tts

FIRE RED - TTS

FireRedTTS: A Foundation Text-To-Speech Framework for Industry-Level Generative Speech Applications

FIRE RED - TTS

[Paper] [Code] [Huggingface Demo]

FireRed Team

Abstract. This work proposes FireRedTTS, a foundation text-to-speech framework, to meet the growing demands for personalized and diverse generative speech applications. The framework comprises three parts: data processing, foundation system, and downstream applications. First, we comprehensively present our data processing pipeline, which transforms massive raw audio into a large-scale high-quality TTS dataset with rich annotations and a wide coverage of content, speaking style, and timbre. Then, we propose a language-model-based foundation TTS system. The speech signal is compressed into discrete semantic tokens via a semantic-aware speech tokenizer, and can be generated by a language model from the prompt text and audio. Then, a two-stage waveform generator is proposed to decode them to the high-fidelity waveform. We present two applications of this system: voice cloning for dubbing and human-like speech generation for chatbots. The experimental results demonstrate the solid in-context learning capability of FireRedTTS, which can stably synthesize high-quality speech consistent with the prompt text and audio. For dubbing, FireRedTTS can clone target voices in a zero-shot way for the UGC scenario and adapt to studio-level expressive voice characters in the PUGC scenario via few-shot fine-tuning with 1-hour recording. Moreover, FireRedTTS achieves controllable human-like speech generation in a casual style with paralinguistic behaviors and emotions via instruction tuning, to better serve spoken chatbots.

Data Process Pipeline

Source Separation

模型：Meta: Demucs

任务：分背景音乐

Denoise

模型：DeepFilterNet

任务：降噪

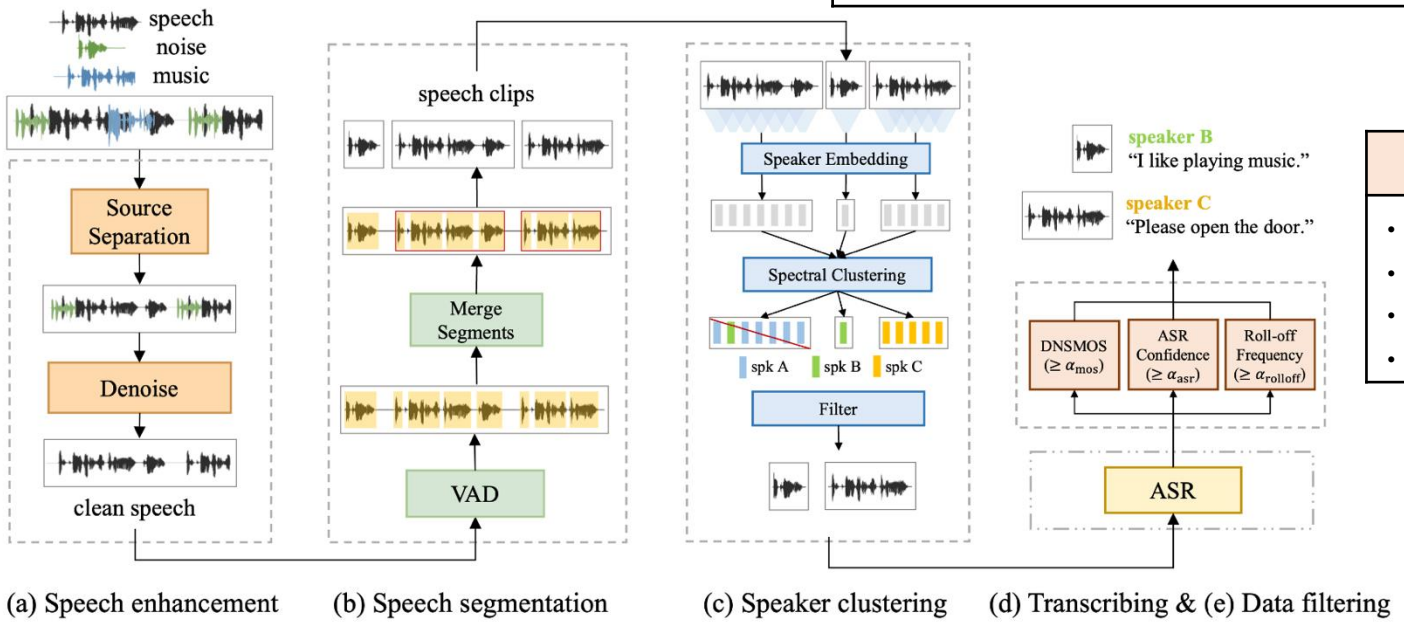
Merge Segments

策略1：合并静音少于 1s 的相邻片段

策略2：合并后的 segment 首尾保留 0.3s 静音

Speaker Embedding/Speaker Clustering/Filter

- embedding 模型：wespeaker
- 聚类策略：chunk 级别计算 embedding，谱聚类
- 过滤策略一：包含不同 spk-id 的片段直接去除
- 过滤策略二：与聚类中心太远的片段直接去除



Data Filtering Methods

- DNSMOS OVRL score ≥ 3.3
- ASR confidence score ≥ 0.8
- Roll-off 频率 ≥ 7kHz
- Roll-off 频率：99.5% 的能量在这个频率以内

VAD

模型：基于 TDNN 的 VAD

目标：得到 2-20s 的音频

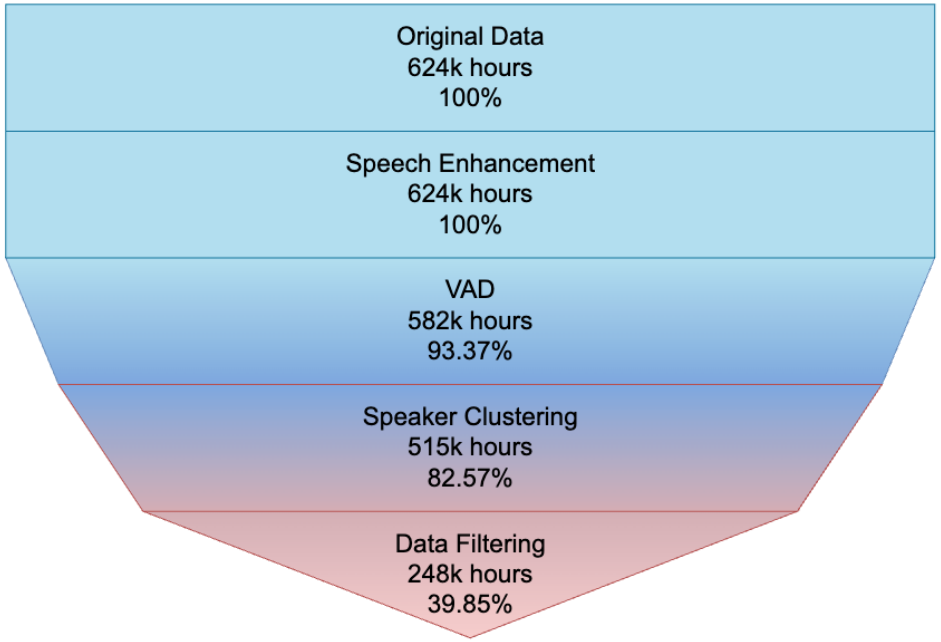
ASR

ASR 模型：RNN-T + LAS 的 2-pass 模型

Data Process Pipeline

训练数据量

- 共清洗 248k 小时
- 论文使用了其中 15 万小时的子集
  - 中文 11 万小时 + 英文 4 万小时



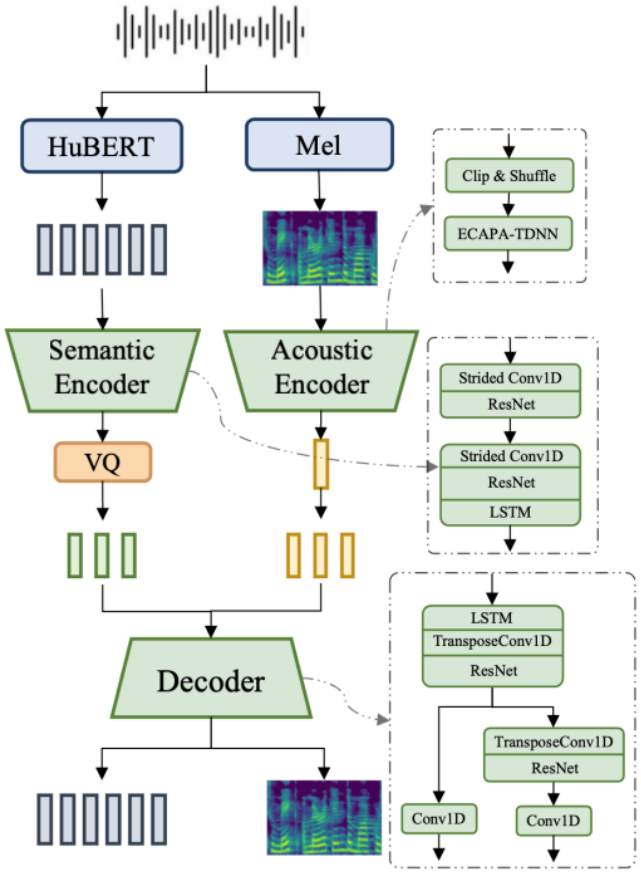
对比 CosyVoice 训练数据量

Language	Duration (hr)
ZH	130,000
EN	30,000
Yue	5,000
JP	4,600
KO	2,200

Table 2: Hours of CosyVoice training data across languages in the large-scale experiments.

Model: Semantic-Aware Speech Tokenizer

SAST：Semantic-Aware Speech Tokenizer

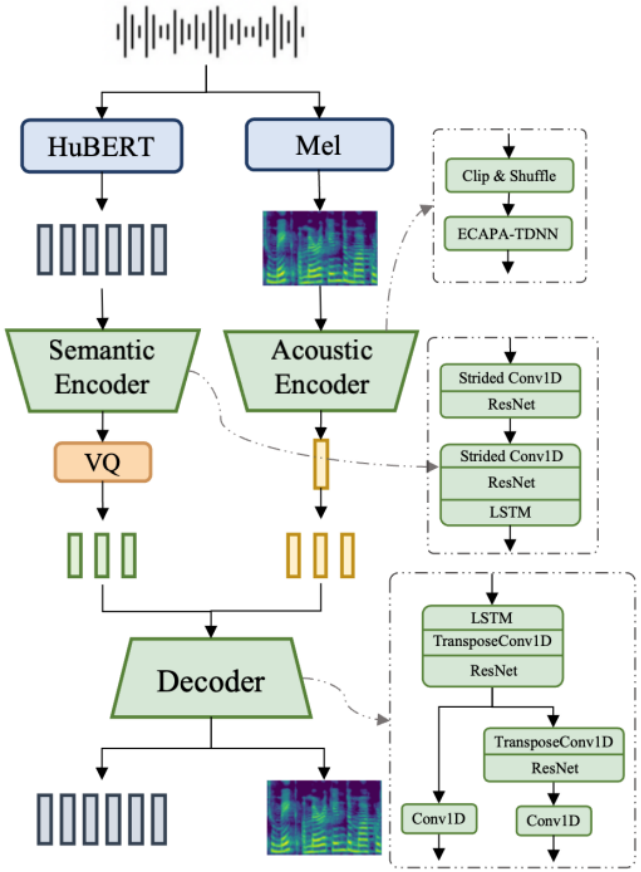


建模参数：16kHz 音频，帧率 50Hz

- **Semantic Encoder**
  - 卷积下采样至 25Hz 帧率（两层 Strided Conv?）
  - 增加 LSTM 层利用历史信息
- **Acoustic Encoder：Timbre/Style Encoder（ECAPA-TDNN）**
  - 解耦方案：梅尔谱输入 Encoder 之前 ***Clip & Shuffle***
  - 每条 utterance 随机选择 25%-75% 的时长范围
  - 每 1 s 作为一个小片段，片段shuffle之后拼接
- **Decoder：双路输出**
  - 先经过 LSTM 和 Transpose Conv 上采样
  - Hubert embedding 重建，作为一个输出分支
  - 梅尔特征重建，作为一个输出分支（多一次上采样，50Hz帧率）

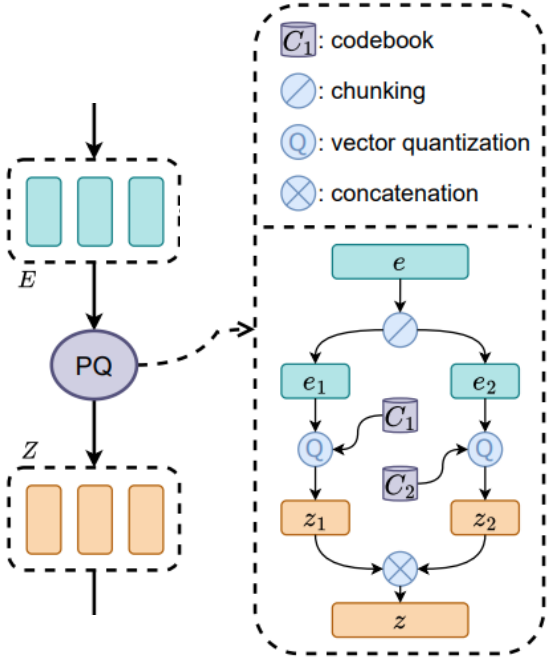
Model: Semantic-Aware Speech Tokenizer

SAST：Semantic-Aware Speech Tokenizer



VQ 模块（Product Quantization）

- 将 embedding 在维度上均分成多部分
- 每部分用单独的 codebook 学习
- 总 codebook size 可以认为是 codebook size 的乘积组合
  - 总 codebook size = 16384 = 128*128
  - 猜测：两个 codebook=128 的 VQ 构成的 Group-VQ



训练 Loss 构成

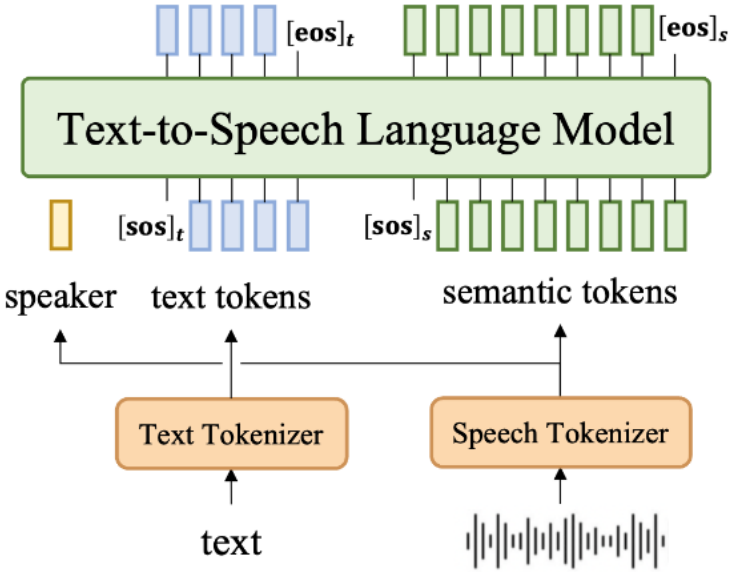
$$\mathcal{L}_c = \lambda_{vq} * \mathcal{L}_{vq} + \lambda_s * \mathcal{L}_s + \lambda_a * \mathcal{L}_a$$

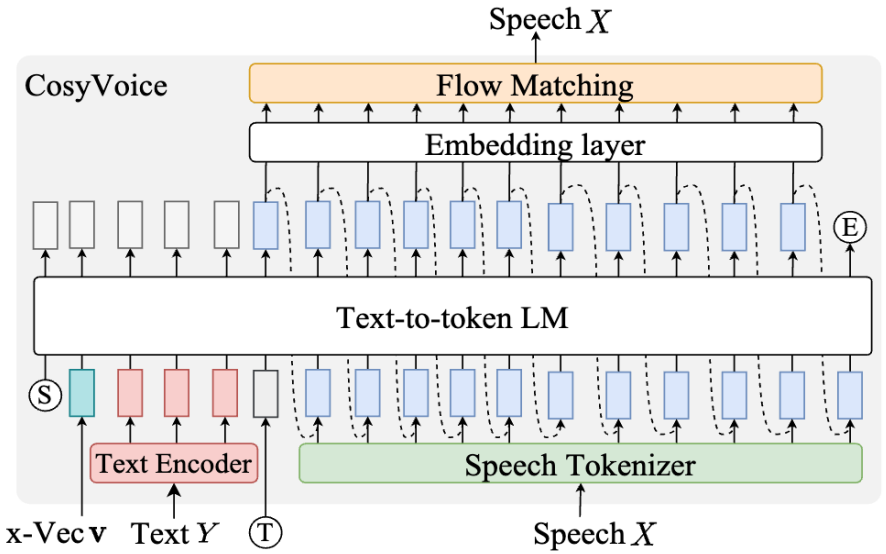
$$\lambda_{vq} = 1, \lambda_s = 1000, \lambda_a = 1$$

batch size: 6400s，训练 300k steps

Model: Text-To-Speech Language Model

Text-To-Speech Language Model





(b) An overview of the proposed CosyVoice LM

Text tokenizer: Whisper 的 tokenizer

训练阶段

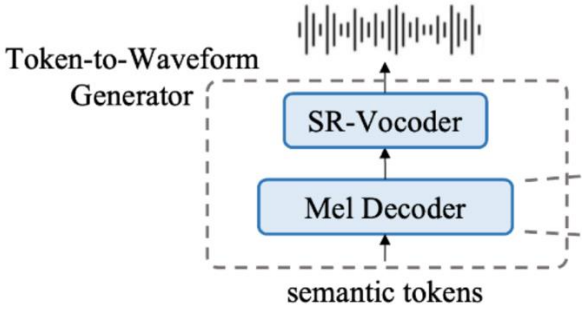
- 从训练音频中，抽取 semantic token 和 speaker embedding
- 训练自回归模型：30 层，参数量 400M 的 Transformer

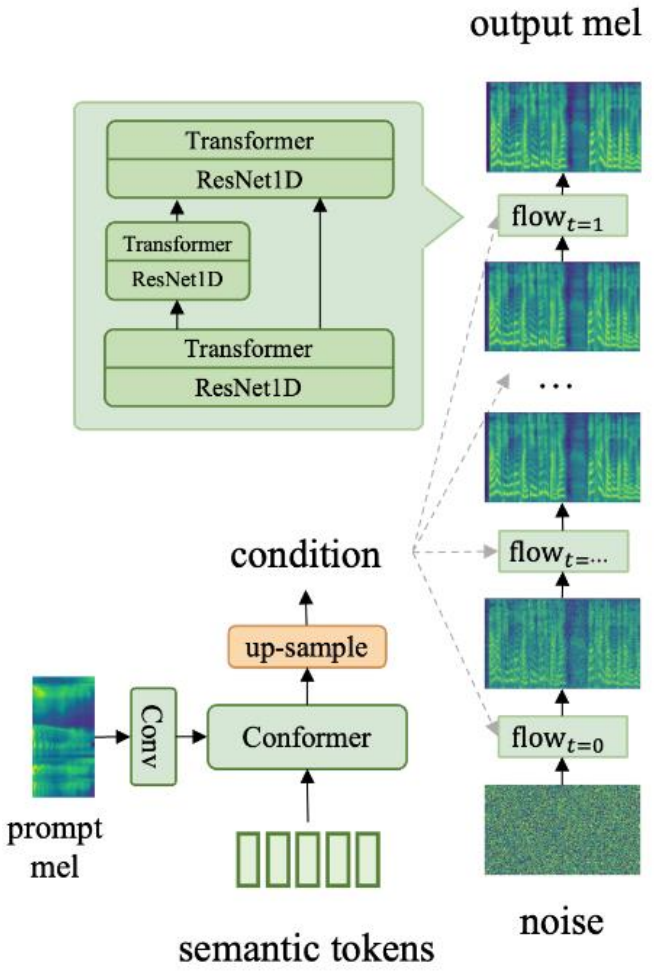
推理阶段

- 提供 prompt 的 speaker embedding + text/semantic token prompt

Model: Text-To-Speech Language Model

Token-to-Waveform





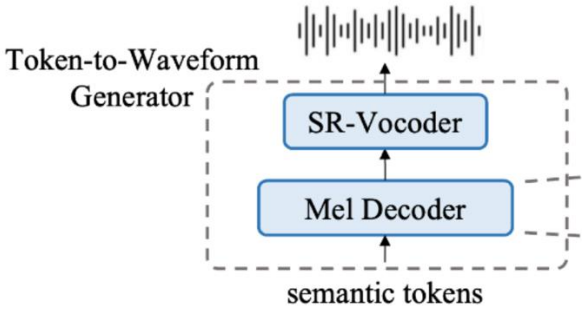
Non-Streaming: semantic token → mel (16kHz)

Flow Matching Decoder

- 概述：基于 semantic token 预测生成 16kHz 的 mel 特征
- ***获取 condition 输入***
  - Semantic token 经过 Conformer，最近邻插值，上采样到 mel 帧率
  - Conformer 每一层的 self-attention 之后再加一层 cross-attention
    - K/V 是 prompt 经过卷积 Encode 的 embedding 序列
    - prompt 选择的是同一 speaker 的其他音频

Model: Text-To-Speech Language Model

Token-to-Waveform



Non-Streaming: semantic token → mel (16kHz)

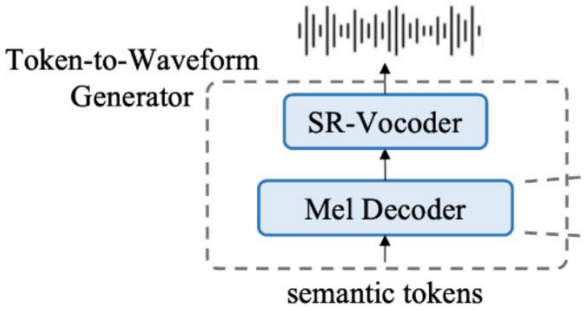
Flow Matching Decoder

- 概述：基于 semantic token 预测生成 16kHz 的 mel 特征
- ***基于 Transformer-UNet 的 Flow Matching***
  - 模型结构为 Transformer-UNet
  - 输入包括：(不直接引入 speaker embedding)
    - x_0 为随机高斯分布采样（x_{t-1}）
    - mu 为 semantic 得到的 embeddings (condition)
    - t 为时间步 timestep 数值（cosine scheduler, 0-1 之间的数值）
  - 输出： Transformer-Unet 预测得到的 x_t

```python
class ConditionalCFM(nn.Module):
    def __init__(self,
                 estimator: nn.Module,
                 t_scheduler: str = "cosine",
                 inference_cfg_rate: float = 0.7,
                 ):
        super().__init__()
        self.estimator = estimator
        self.t_scheduler = t_scheduler
        self.inference_cfg_rate = inference_cfg_rate

    def solve_euler(self, x, t_span, mu, mask):
        t, _, dt = t_span[0], t_span[-1], t_span[1] - t_span[0]

        # I am storing this because I can later plot it by putting a debugger here and saving it to a file
        # Or in future might add like a return_all_steps flag
        sol = []

        for step in range(1, len(t_span)):
            dphi_dt = self.estimator(x, mask, mu, t)
            # Classifier-Free Guidance inference introduced in VoiceBox
            if self.inference_cfg_rate > 0:
                cfg_dphi_dt = self.estimator(x, mask, torch.zeros_like(mu), t)
                dphi_dt = ((1.0 + self.inference_cfg_rate) * dphi_dt -
                           self.inference_cfg_rate * cfg_dphi_dt)
            x = x + dt * dphi_dt
            t = t + dt
            sol.append(x)
            if step < len(t_span) - 1:
                dt = t_span[step + 1] - t

        return sol[-1]

    def inference(self, mu, mask, n_timesteps, temperature: float=1.0):
        z = torch.randn_like(mu) * temperature
        t_span = torch.linspace(0, 1, n_timesteps + 1, device=mu.device)
        if self.t_scheduler == 'cosine':
            t_span = 1 - torch.cos(t_span * 0.5 * torch.pi)
        return self.solve_euler(z, t_span=t_span, mu=mu, mask=mask)
```

Model: Text-To-Speech Language Model

Token-to-Waveform



Non-Streaming: semantic token → mel (16kHz)

Flow Matching Decoder

- 概述：基于 semantic token 预测生成 16kHz 的 mel 特征
- ***CFG：Classifier-Free Guidance***
  - 训练阶段每个 batch 随机去除 20% 样本的 condition

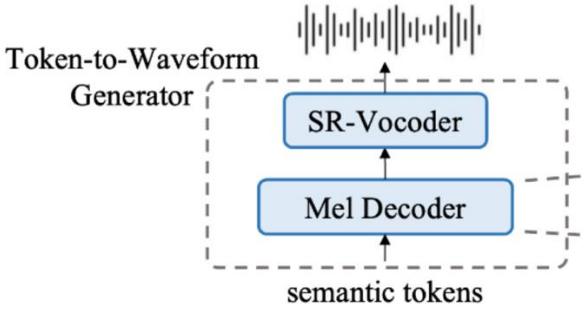
$$v_t^{\text{cfg}} = (1+\alpha)\text{NN}_\theta(x_t, t, \Psi) - \alpha\text{NN}_\theta(x_t, t)$$

条件式生成　　无条件式生成

```python
class ConditionalCFM(nn.Module):
    def __init__(self,
                 estimator: nn.Module,
                 t_scheduler: str = "cosine",
                 inference_cfg_rate: float = 0.7,
                 ):
        super().__init__()
        self.estimator = estimator
        self.t_scheduler = t_scheduler
        self.inference_cfg_rate = inference_cfg_rate

    def solve_euler(self, x, t_span, mu, mask):
        t, _, dt = t_span[0], t_span[-1], t_span[1] - t_span[0]

        # I am storing this because I can later plot it by putting a debugger here and saving it to a file
        # Or in future might add like a return_all_steps flag
        sol = []

        for step in range(1, len(t_span)):
            dphi_dt = self.estimator(x, mask, mu, t)
            # Classifier-Free Guidance inference introduced in VoiceBox
            if self.inference_cfg_rate > 0:
                cfg_dphi_dt = self.estimator(x, mask, torch.zeros_like(mu), t)
                dphi_dt = ((1.0 + self.inference_cfg_rate) * dphi_dt -
                           self.inference_cfg_rate * cfg_dphi_dt)
            x = x + dt * dphi_dt
            t = t + dt
            sol.append(x)
            if step < len(t_span) - 1:
                dt = t_span[step + 1] - t

        return sol[-1]

    def inference(self, mu, mask, n_timesteps, temperature: float=1.0):
        z = torch.randn_like(mu) * temperature
        t_span = torch.linspace(0, 1, n_timesteps + 1, device=mu.device)
        if self.t_scheduler == 'cosine':
            t_span = 1 - torch.cos(t_span * 0.5 * torch.pi)
        return self.solve_euler(z, t_span=t_span, mu=mu, mask=mask)
```

Model: Text-To-Speech Language Model

Token-to-Waveform



SR-Vocoder: 16kHz mel → 48kHz Waveform

BigVGAN 模型结构，100 帧率上采样 480 倍，生成 48kHz 波形

CoMOS: 风格和韵律的一致性主观分数

System	CoMOS(↑)
GroundTruth	4.53
CosyVoice	4.15
FireRedTTS	4.32

Table 2: The CoMOS evaluation results.

System	Overall(%)			Sub(%)			#Ins.&Del.(%)		
	ZH	EN	MIX	ZH	EN	MIX	ZH	EN	MIX
CosyVoice	5.68	12.17	29.50	3.76	6.67	25.00	1.92	5.50	4.50
Ours	2.09	12.00	8.50	1.00	0.50	4.50	1.09	11.50	4.00

Table 3: The stability evaluation results. “MIX" denotes the code-switch utterances. “Sub" denotes the ratio of substitution errors. “#Ins.&Del" denotes the ratio of insertion and deletion errors.

Comparison with Similar Models

名称	论文	时间	spk-emb 拼接策略	Tokenizer	Decoder	Vocoder	Vocoder 输入
Tortoise-TTS	2305.07243	2023.05	不拼接	Mel-VQ	DDPM/DDIM	Univnet	mel
GPT-soVITS	-	2024.01	不拼接	Hubert	VITS	HiFiGAN	embedding
Base-TTS	2402.08093	2024.02	拼接单个 spk-emb	解耦方案	HiFiGAN 结构	BigVGAN	embedding
Chat-TTS	-	2024.05	不拼接	Mel-VQ	Decoder	Vocos	mel
Fish-speech	-	2024.05	不拼接	Mel VQ	VITS	HiFiGAN	embedding
Seed-TTS	2406.02430	2024.06	不拼接	未说明细节	DiT	未说明细节	embedding
XTTS-v2	2406.04904	2024.06	拼接：perceiver	Mel-VQ	HiFiGAN		embedding
CosyVoice	2407.05407	2024.07	拼接单个 spk-emb	S3-Tokenizer	Flow-Matching	HiFTNet	mel
FireRedTTS	2409.03283	2024.09	拼接单个 spk-emb	SAST	Flow-Matching	BigVGAN	mel