



دانشگاه تهران  
دانشکده علوم و فنون نوین

گزارش تمرین اول

نام و نام خانوادگی	فاطمه چیت ساز
شماره دانشجویی	830402092
تاریخ ارسال گزارش	18 اسفند 1402

## Contents

1	..... LEVERAGING SPEECH PTM, TEXT LLM, AND EMOTIONAL TTS	برسی مقاله
4	.....	آشنایی با یک سیستم بازشناسی گفتار
6	.....	آشنایی با numpy
6	.....	پیاده سازی: سنتز گفتار فارسی

مقدمه:

تشخیص احساسات گفتار (SER) اهمیت زیادی در استخراج احساسات گوینده از سیگنال‌های گفتاری دارد. اما چالش اصلی در این حوزه کمبود داده است. برای غلبه بر این چالش، مدل‌های پیش‌تنظیم‌شده خود-نظارت‌یافته (PTM) گفتار راه حلی موثر است.

مقاله نشان می‌دهد که مدل data2vec با استفاده کمتر، دقت بالاتری در تشخیص احساسات گفتاری دارد. همچنین استفاده از داده‌های بیشتر و متنوع‌تر می‌تواند عملکرد مدل را ارتقاء دهد. محدودیت‌های روش‌های اولیه افزایش داده:

روش‌های اولیه مانند تغییر سرعت، SpecAugment و mixup، فقط روی ویژگی‌های صوتی تمرکز دارند. روش‌های دیگر مانند ترکیب گفتار خنثی با احساسی، اطلاعات عاطفی جدیدی اضافه می‌کنند، اما ممکن است مصنوعی باشند. معرفی روش‌های جدیدتر افزایش داده:

این مقاله روش‌های جدیدتری را برای افزایش داده در SER معرفی می‌کند که محدودیت‌های روش‌های قبلی را برطرف می‌کنند. این روش‌ها شامل تولید متن و گفتار احساسی با استفاده از مدل‌های زبان بزرگ (LLM) و تکنیک‌های سنتز گفتار است.

روش‌های مختلفی برای تولید داده‌های گفتاری مصنوعی وجود دارد، از جمله:

مدل‌های مبتنی بر GAN: این مدل‌ها دو شبکه عصبی را به صورت رقابتی با یکدیگر آموزش می‌دهند. یک شبکه تلاش می‌کند تا داده‌های مصنوعی واقعی‌گرایانه تولید کند، در حالی که شبکه دیگر سعی می‌کند تا داده‌های مصنوعی را از داده‌های واقعی تشخیص دهد.

مدل‌های مبتنی بر انتشار: این مدل‌ها از نویز تصادفی برای تولید داده‌های مصنوعی استفاده می‌کنند.

مدل‌های مبتنی بر WaveNet: این مدل‌ها از شبکه‌های عصبی عمیق برای تولید موج‌های صوتی استفاده می‌کنند.

محدودیت‌های روش‌های موجود:

بسیاری از روش‌های موجود برای تولید داده‌های گفتاری مصنوعی، کیفیت پایینی دارند.

این داده‌ها ممکن است از نظر آکوستیک یا معنایی با داده‌های گفتاری واقعی همخوانی نداشته باشند.

برخی از روش‌ها به حجم زیادی از داده‌های واقعی برای آموزش نیاز دارند.

روش پیشنهادی:

در مرحله اول، از مدل GPT-4 که یکی از قدرتمندترین مدل‌های زبانی بزرگ است، برای تولید داده‌های متنی احساسی با کیفیت بالا استفاده می‌شود. GPT-4 قادر است متن را با لحن و احساسات مختلف تولید کند و با توجه به prompt مشخص شده، متن مورد نظر را تولید می‌کند. الگوریتم تولید متن:

الگوریتم به طور دقیق نحوه طراحی prompt برای مدل زبان بزرگ GPT-4 به منظور تولید متن با بار احساسی را شرح می‌دهد. این الگوریتم دارای ورودی و خروجی‌های خاصی است و از چند مرحله تشکیل شده است:

ورودی:

- لیست سبک‌های روایت N - (Narrative Styles)
- لیست سناریوها S - (Scenarios)
- لیست احساسات E - (Emotions)
- لیست حداکثر تعداد توکن‌ها M - (Max Tokens)

خروجی:

- لیست متن‌های تولید شده توسط U - GPT-4

مراحل:

1. تنظیم نقش سیستم:

"شما یک دستیار مفید با احساسات و سبک‌های صحبت انسان هستید."

2. گروه بندی ترکیبات مجاز:

الگوریتم ترکیباتی معتبر (T) را تشکیل می‌دهد که ترکیباتی از سبک روایت، سناریو، احساس و حداکثر تعداد توکن است.

3. ایجاد prompt برای هر ترکیب:

یک حلقه برای بررسی تک تک ترکیبات معتبر اجرا می‌شود.

- تعیین طول متن بر اساس حداکثر تعداد توکن.

- ایجاد prompt برای سبک‌های روایت گفتگو و شرح.

4. تولید متن با GPT-4:

برای هر prompt، تعدادی متن با استفاده از GPT-4 تولید شده و به لیست خروجی اضافه می‌شود.

5. پاکسازی داده‌ها:

عملیات پاکسازی روی متن‌های تولید شده انجام می‌شود.

6. خروجی نهایی:

لیست نهایی متن‌های تولید شده برگردانده می‌شود.

در مرحله دوم، از سرویس Azure TTS برای تبدیل متن به گفتار با کیفیت بالا استفاده می‌شود. این سرویس قادر است متن را به گفتار با کیفیت بالا تبدیل کند و لحن و صدای مناسب برای هر احساس را انتخاب کند.

در مرحله سوم، داده‌های گفتاری مصنوعی با داده‌های واقعی ادغام می‌شوند. این ادغام ممکن است از روش‌های مختلفی مانند random mixing و adversarial training ، transfer learning و curriculum learning استفاده کند. سپس با استفاده از مجموعه داده IEMOCAP، که یک مجموعه داده استاندارد برای تشخیص احساسات گفتار است، عملکرد روش پیشنهادی با EmoDiff، یک مدل تخصصی برای TTS احساسی، مقایسه می‌شود.

نتایج :

Representations	Training Data	WA ↑	UA ↑
last layer	IEMOCAP	64.52	68.22
	IEMOCAP + EmoDiff	68.02	69.67
	<b>IEMOCAP + ours</b>	<b>68.57</b>	<b>70.86</b>
multiple layers	IEMOCAP	68.02	69.95
	IEMOCAP + EmoDiff	68.39	71.06
	<b>IEMOCAP + ours</b>	<b>68.85</b>	<b>71.89</b>

نتایج نشان می‌دهد افزایش داده همیشه مفید نیست، زیرا اضافه کردن بیش از حد داده‌های مصنوعی برای تشخیص احساسات گفتار، همیشه سودمند نیست. بهترین عملکرد زمانی حاصل می‌شود که مقدار داده‌های مصنوعی اضافه شده، نصف داده‌های اصلی باشد. دلایل افت عملکرد این است که اضافه کردن بیش از حد داده‌های مصنوعی، باعث می‌شود تا توزیع داده‌های مصنوعی بر توزیع داده‌های واقعی غالب شود و عملکرد مدل را به آن انتقال دهد.

آزمایش‌ها بر روی مجموعه داده IEMOCAP نشان می‌دهد که روش مقاله کارایی خود را اثبات کرده است. در آزمایش‌ها، فقط یک زیر مجموعه کوچک از داده‌های مصنوعی مورد استفاده قرار گرفته است. در آینده، نویسندگان مقاله قصد دارند که بررسی کنند که چگونه می‌توانند از داده‌های مصنوعی بزرگ‌مقیاس و با کیفیت بالا استفاده کنند تا موقعیت‌های متقاطع، دامنه‌های متفاوت و زبان‌های متفاوت را تقویت کنند.

## آشنایی با یک سیستم بازشناسی گفتار

قسمت الف:

در این سوال هدف این است چندین جمله را با استفاده از سایت speechtext بخوانیم و نتایج را با جملات اصلی مقایسه کنیم

جملات اصلی من :

دوست داشتنی‌ترین چیز در زندگی، لبخند زدن و خوشحالی است.

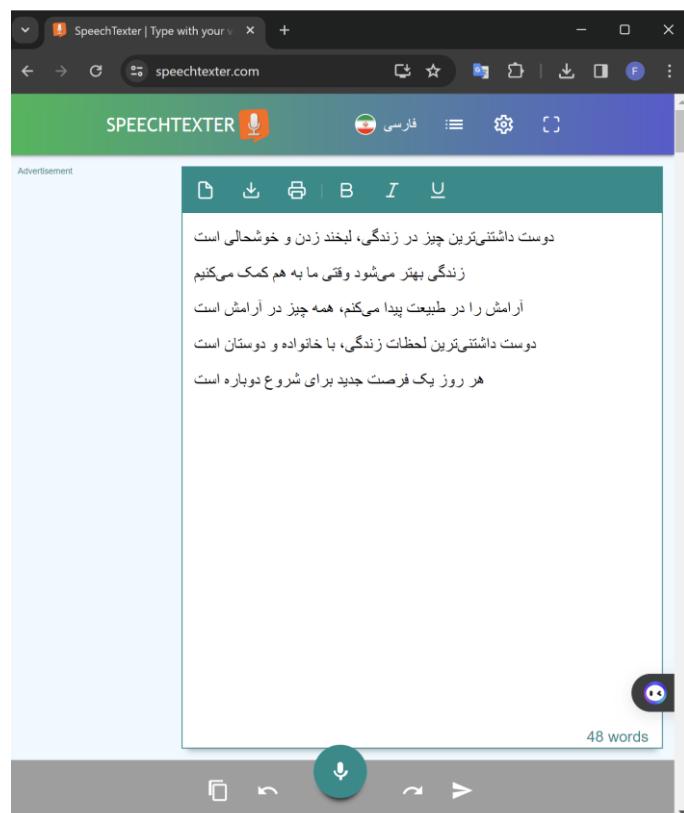
زندگی بهتر می‌شود وقتی ما به هم کمک می‌کنیم.

آرامش را در طبیعت پیدا می‌کنم، همه چیز در آرامش است.

دوست داشتنی‌ترین لحظات زندگی، با خانواده و دوستان است.

هر روز یک فرصت جدید برای شروع دوباره است.

جملات در سایت speech text:



نتیجه:

همانطور که دیده می‌شود جملات نوشته شده توسط سایت با جملات اصلی یکسان است

چرایی رخداد WER:

WER (Word Error Rate) یا نرخ خطای کلمه، معیاری برای سنجش دقت سیستم‌های بازشناسی گفتار است. این معیار، تعداد کلمات اشتباه تشخیص داده شده را به تعداد کل کلمات در متن مرجع تقسیم می‌کند.

دلایل مختلفی برای رخداد WER در سیستم‌های بازشناسی گفتار وجود دارد:

1. نویز:

نویز محیطی می‌تواند سیگنال گفتار را مخدوش کند و تشخیص کلمات را دشوار کند. نویز ضبط نیز می‌تواند از میکروفون یا دستگاه ضبط صدا به سیگنال گفتار اضافه شود.

2. تنوع گفتاری:

افراد مختلف با لهجه‌ها، سرعت گفتار و لحن‌های متفاوت صحبت می‌کنند که می‌تواند سیستم را در تشخیص کلمات با چالش مواجه کند.

3. عدم تطابق بین مدل و داده‌های واقعی:

اگر مدل بازشناسی گفتار با داده‌های واقعی که به آن ارائه می‌شود، همخوانی نداشته باشد، احتمال خطا افزایش می‌یابد.

4. خطاهای زبانی:

سیستم‌های بازشناسی گفتار ممکن است در تشخیص کلمات مشابه یا کلماتی که در زمینه‌های مختلف معانی متفاوتی دارند، دچار خطا شوند.

5. محدودیت‌های تکنولوژی:

تکنولوژی بازشناسی گفتار هنوز در حال پیشرفت است و قادر به تشخیص دقیق تمام کلمات در تمام شرایط نیست.

راهکارهای مختلفی برای کاهش WER در سیستم‌های بازشناسی گفتار وجود دارد:

- استفاده از الگوریتم‌های پیشرفته‌تر
- بهبود کیفیت داده‌های آموزشی
- استفاده از تکنیک‌های پیش پردازش و پس پردازش
- تطبیق مدل با داده‌های واقعی
- استفاده از زیانشناسی و دانش زبانی

قسمت ب:

$$WER = \frac{D + I + S}{X}$$

با توجه به اینکه تمامی کلمات درست بیان شده wer برای همه کلمات صفر است

### آشنایی با numpy

پاسخ این قسمت در پوشه Q3 آورده شده است

### پیاده سازی: سنتز گفتار فارسی

قسمت اول: فایل این قسمت در پوشه Q4 و سپس پوشه a وجود دارد

توضیحات :

در این قسمت قرار است که ما یک سیستم سنتز گفتار کوچک را پیاده سازی کنیم که بتواند اعداد را از صفر تا یک میلیون را بخواند و این اعداد میتوانند هم به حروف و هم به عدد باشند

مرحله یک : ابتدا باید فایل ها ضبط شوند فایل های صوتی شامل اعداد صفر تا بیست و سپس سی و چهل و ... و صد و دویست و ... و هزار و میلیون و کلمه و است که در مجموع چهل عضو دارد

```
persian_number_map = {
    'صفر': 0,
    'یک': 1,
    'دو': 2,
    'سه': 3,
    'چهار': 4,
    'پنج': 5,
    'شش': 6,
    'هفت': 7,
    'هشت': 8,
    'نه': 9,
    'ده': 10,
    'یازده': 11,
    'دوازده': 12,
    'سیزده': 13,
    'چهارده': 14,
    'پانزده': 15,
    'شانزده': 16,
    'هفده': 17,
    'هجده': 18,
    'نوزده': 19,
    'بیست': 20,
    'سی': 30,
    'چهل': 40,
    'پنجاه': 50,
    'شصت': 60,
    'هفتاد': 70,
    'شتاد': 80,
    'نود': 90,
    'صد': 100,
    'دویست': 200,
    'سیصد': 300,
    'چهارصد': 400,
    'پانصد': 500,
    'ششصد': 600,
    'هفتصد': 700,
    'هشتصد': 800,
}
```

```
print(len(persian_number_map))
```

✓ 0.0s

40

من نام فایل ها را مطابق با همان اعداد آنها برای سادگی قرار دادم



در مرحله بعدی به آزادی وردی عدد ما باید فرایندی را تولید کنیم که لیست فایل هایی که باید باهم تجميع شوند را به ما بدهد:

```
def parse_number(number):
    # Split the number into hundreds, tens, and ones
    parts = []
    if number >= 1000000:
        million = number // 1000000
        parts.append(f"{million*1000000}" + ".wav")
        number %= 1000
        if number > 0:
            parts.append("and.wav")

    if number >= 1000:
        thousand = number // 1000
        part = parse_number(thousand)
        for p in part:
            parts.append(p)
        parts.append(f"{1000}" + ".wav")
        number %= 1000
        if number > 0:
            parts.append("and.wav")

    if number >= 100:
        hundreds = number // 100
        parts.append(f"{hundreds*100}" + ".wav")
        number %= 100
        if number > 0:
            parts.append("and.wav")

    if 20 <= number < 100:
        tens = (number // 10) * 10
        parts.append(f"{tens}" + ".wav")
        number %= 10
        if number > 0:
            parts.append("and.wav")
```

تابع `parse_number` به عنوان ورودی یک عدد صحیح می گیرد و آن را به صورت مجموعه ای از رشته ها (فایل های صوتی) با توجه به ساختار عدد از صدها، ده ها و یک ها تجزیه می کند. این تابع برای اعداد تا میلیون (بزرگترین واحد در دیکشنری) محاسبه می شود.

برای هر واحد از عدد (صدها، هزارها، میلیونها) محاسبه می شود و اگر واحد بعدی نیز مقدار داشته باشد، کلمه "and" به لیست اضافه می شود. در نهایت، لیستی از رشته ها که نشان دهنده صوت های مربوط به هر بخش از عدد است، به عنوان خروجی تابع برگردانده می شود.

برای اعدادی که بیشتر مساوی هزار یا ملیون هستند تابع دوباره خودش را فراخوانی میکند تا باقی مانده آنها به هزار را `parse` کند

مثال :

```
202010
parts: ['200.wav', 'and.wav', '2.wav', '1000.wav', 'and.wav', '10.wav']
```

در مرحله دوم باید لیست فایل هایی که داریم را باهم تجميع کنیم که برای این کار :

```
def synthesize_number(number):
    audio_files_to_concatenate = parse_number(number)
    print("parts: ", audio_files_to_concatenate)
    combined = AudioSegment.empty()
    for file_name in audio_files_to_concatenate:
        combined += AudioSegment.from_wav("record_files/" + file_name)
    return combined
```

توضیح تابع:

synthesize\_number یک عدد صحیح را به عنوان ورودی می‌گیرد و سپس از تابع parse\_number برای تجزیه عدد به صورت فایل‌های صوتی استفاده می‌کند. سپس با استفاده از کتابخانه‌ی AudioSegment این فایل‌های صوتی را با یکدیگر ترکیب کرده و یک فایل صوتی کلی را ایجاد می‌کند.

فایل صوتی جدید با نام عددی خود ذخیره می‌شود

مثال :

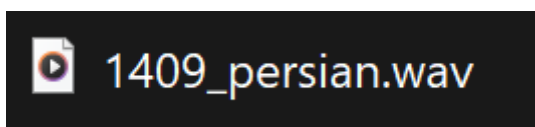
```
# Example usage:
number = 1409
audio_output = synthesize_number(number)
audio_output.export(str(number)+"_persian.wav", format="wav")

✓ 0.0s

parts: ['1.wav', '1000.wav', 'and.wav', '400.wav', 'and.wav', '9.wav']

<_io.BufferedRandom name='1409_persian.wav'>
```

عدد هزار و چهارصد و به لیست فایل‌ها شکسته شده و سپس فایل‌ها باهم جمع شدند و در پوشه‌ای که فایل پایتون قرار دارد با نام 1409\_persian ذخیره می‌شود



مرحله بعدی ما باید مقادیر عددی به صورت حروف را نیز بتوانیم سنتز کنیم برای اینکه بتوانیم از سیستم قبلی خود استفاده کنیم من مقادیر حرفی را به مقادیر عددی تبدیل کرده و آن را به سیستم قبلی می‌دهم برای تبدیل اعداد حرفی به عددی نیز از یک مپ استفاده می‌کنیم که به صورت زیر تعریف می‌شود

```
def text_to_number(persian_text):
    # Convert Persian text to a number
    parts = persian_text.split()
    number = 0
    temp_number = 0
    for part in parts:
        if part in persian_number_map:
            value = persian_number_map[part]
            if isinstance(value, int):
                if value >= 1000:
                    number += temp_number * value
                    temp_number = 0
                else:
                    temp_number += value
            elif value == 'and':
                continue
        number += temp_number
    return number
```

text\_to\_number یک متن فارسی را به عدد تبدیل می‌کند و از دیکشنری persian\_number\_map که قبلاً تعریف شده است استفاده می‌کند

الگوریتم این تابع به این صورت است:

- متن ورودی را به عناصر تکی تقسیم می‌کند (با استفاده از فاصله به عنوان جداکننده).
- برای هر عنصر متنی، مقدار متناظر آن را در `persian_number_map` جستجو می‌کند.
- اگر مقدار متناظر یک عدد صحیح بود، آن را به عدد موقت (`temp_number`) اضافه می‌کند. اگر مقدار متناظر با 'and' بود، ادامه می‌دهد.
- اگر عدد متناظر بزرگتر از یک هزار بود، آن را با مقدار عدد موقت ضرب کرده و به عدد اصلی (`number`) اضافه می‌کند و عدد موقت را صفر می‌کند.
- در نهایت، عدد موقت را به عدد اصلی اضافه کرده و عدد نهایی را برمی‌گرداند.

مثال :

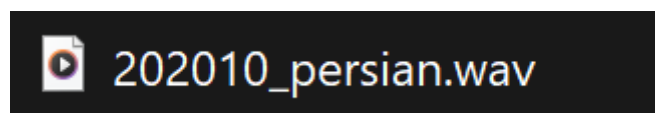
```
input_text = "دویست و دو هزار و ده"
number = text_to_number(input_text)
print(number)
audio_output = synthesize_number(number)
audio_output.export(f"{number}_persian.wav", format="wav")
✓ 0.0s

202010
parts: ['200.wav', 'and.wav', '2.wav', '1000.wav', 'and.wav', '10.wav']

<_io.BufferedRandom name='202010_persian.wav'>
```

در اینجا عدد دویست و دو هزار و ده ابتدا به عدد تبدیل شده و سپس از طریق متد قبلی به لیست فایل‌ها تبدیل شده و سپس فایل‌ها تجمیع شدند و فایل `audio` را میسازند

خروجی :



قسمت دوم : فایل این قسمت در پوشه Q4 و سپس پوشه b وجود دارد

در این قسمت ما می‌خواهیم واج‌ها را به هم بچسبانیم و سنتز گفتار را از این طریق انجام دهیم برای این کار طبق جدول آورده شده بیست و نه فایل مربوط به واج‌ها را ضبط می‌کنیم و من برای سادگی نام آنها را مطابق `index` آنها قرار دادم

#	IPA	Char	Code	Farsi Letter	Phonetic Description
1	I	i	105	ای	high front unrounded
2	e	e	101	اِ	mid front unrounded
3	a	a	97	آ	low front unrounded
4	u	u	117	او	high back rounded
5	o	o	111	اُ	mid back rounded
6	/	/	47	اَ	low back rounded
7	p	p	112	پ	unvoiced bilabial plosive
8	b	b	98	ب	voiced bilabial plosive
9	t	t	116	ت، ط	unvoiced dental plosive
10	d	d	100	د	voiced dental plosive
11	k	k	107	ک	unvoiced velar plosive
12	g	g	103	گ	voiced velar plosive
13	q	q	113	ق، غ	voiced uvular plosive
14	]	]	93	ا، و، ع	glottal stop
15	\$	\$	36	چ	unvoiced alveopalatal affricate closure
16	,	,	44	ج	voiced alveopalatal affricate
17	f	f	102	ف	unvoiced labiodental fricative
18	v	v	118	و	voiced labiodental fricative
19	s	s	115	س، ث، ص	unvoiced alveolar fricative
20	z	z	122	ز، ذ، ظ، ض	voiced alveolar fricative
21	.	.	46	ش	unvoiced alveopalatal fricative
22	[	[	91	ژ	voiced alveopalatal fricative
23	x	x	120	خ	unvoiced uvular fricative
24	h	h	104	ه، ح	unvoiced glottal fricative
25	l	l	108	ل	lateral alveolar
26	r	r	114	ر	trill alveolar
27	m	m	109	م	nasal bilabial
28	n	n	110	ن	nasal alveolar
29	y	y	121	ی	approximant palatal

- 1.wav
- 2.wav
- 3.wav
- 4.wav
- 5.wav
- 6.wav
- 7.wav
- 8.wav
- 9.wav
- 10.wav
- 11.wav
- 12.wav
- 13.wav
- 14.wav
- 15.wav

سپس از ما خواسته شده است که چندین جمله مختلف را با چسباندن این واج ها به یکدیگر بسازیم

مثال اول : هزار و چهارصد و نه

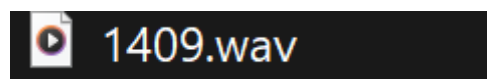
```

1402
parts=['24.wav','2.wav','20.wav','6.wav','26.wav','5.wav','15.wav','6.wav','24.wav','6.wav','26.wav','19.wav','3.wav','10.wav','5.wav','28.wav','5.wav','24.wav']
combined = AudioSegment.empty()
for part in parts:
    combined += AudioSegment.from_wav(part)

combined.export("1409.wav", format="wav")
Python
<_io.BufferedReader name='1409.wav'>

```

در اینجا ما با قرار دادن لیست واج های کلمه هزار چهارصد و دو و تجميع آنها با AudioSegment صدای مربوط به کلمه هزار و چهارصد و نه را میسازیم



مثال اول : دویست و دو هزار و ده

```


دویست و دو هزار و ده
parts=['10.wav','1.wav','18.wav','1.wav','19.wav','9.wav','5.wav','10.wav','5.wav','24.wav','24.wav','2.wav','20.wav','6.wav','26.wav','5.wav','10.wav','3.wav','24.wav']
combined = AudioSegment.empty()
for part in parts:
    combined += AudioSegment.from_wav(part)


combined.export("202010.wav", format="wav")
Python
<_io.BufferedReader name='202010.wav'>


```

در اینجا ما با قرار دادن لیست واج های کلمه هزار چهارصد و دو و تجميع آنها با AudioSegment صدای مربوط به کلمه دویست و دو هزار و ده را میسازیم

این کار را برای فاطمه چیت ساز و دانشگاه تهران و به کجا چنین شتابان نیز انجام می‌دهیم

 به کجا چنین شتابان.wav

 دانشگاه تهران.wav

 فاطمه چیت ساز.wav

مشکل کیفیت خروجی::

برای افزایش قابل فهم بودن صداهای تولید شده، می‌توان از روش‌های مختلفی استفاده کرد. برای مثال، می‌توان از روش‌های پردازش سیگنال صوتی مانند پس‌فیلترینگ، تقویت سیگنال، یا حتی استفاده از تکنیک‌های پردازش زبان مانند افزودن اینتوناسیون مناسب استفاده کرد. این کارها می‌توانند باعث بهبود قابل فهم بودن و صدای طبیعی‌تر شدن صداهای تولید شده توسط سیستم شوند.

روش‌های مختلفی برای افزایش وضوح صدا در این حالت وجود دارد:

- استفاده از مدل‌های پیش‌آموزش‌دیده:
  - مدل‌های پیش‌آموزش‌دیده سنتزگفتار مانند Tacotron 2 یا WaveNet می‌توانند کیفیت صدای تولید شده را به طور قابل توجهی ارتقا دهند.
  - اعمال تکنیک‌های پیش‌پردازش و پس‌پردازش:
  - تکنیک‌هایی مانند حذف نویز، تنظیم وضوح و اعمال افکت‌های مختلف می‌توانند به وضوح و طبیعی‌تر شدن صدا کمک کنند.
  - استفاده از روش‌های بهینه‌سازی:
  - با تنظیم دقیق پارامترهای مدل و استفاده از روش‌های بهینه‌سازی مانند VQ-VAE می‌توان کیفیت صدای تولید شده را بهبود بخشید.
- در مورد کیفیت فایل‌های صوتی حاصل، بستگی به عوامل مختلفی دارد از جمله کیفیت ضبط، محیط ضبط، و توانایی پردازش سیگنال.