DLHLP 2023 Fall HW1 TTS

2023.08.14

TA: 黃維坪 (r11942102@ntu.edu.tw)

Overview

1. TTS 解說
2. 範例程式碼
3. 報告方向

TTS 解説

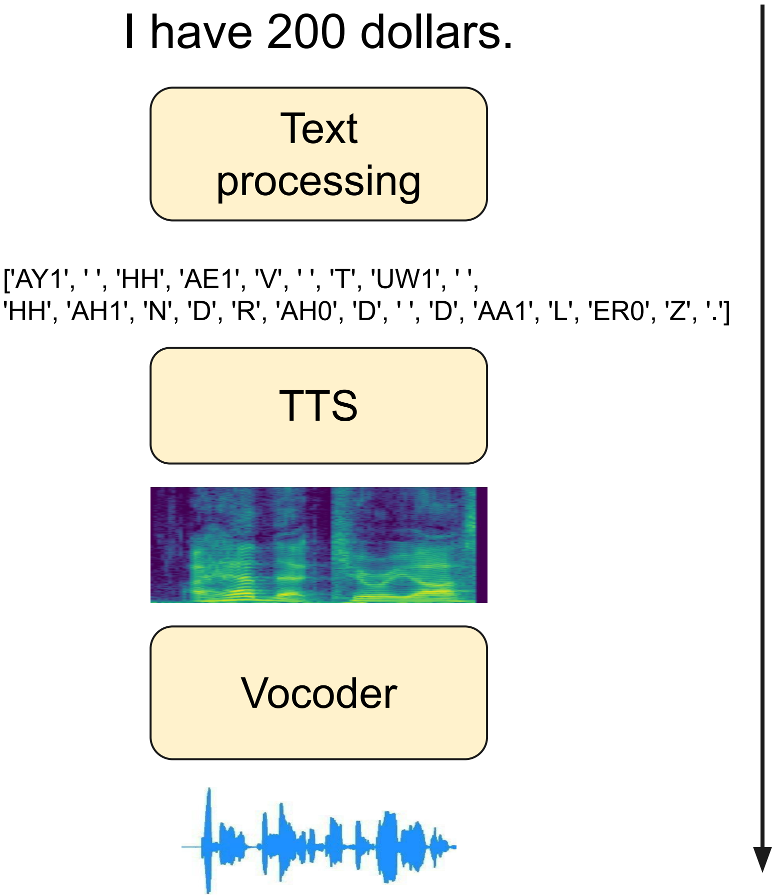
TTS 解說

常見的 TTS 架構包含三個部份:

1. Text (Pre)processing
2. Text to Mel: TTS
3. Mel to wav: Vocoder

* Mel = Mel spectrogram

近年 E2E / 兩階段生成技術逐漸增多



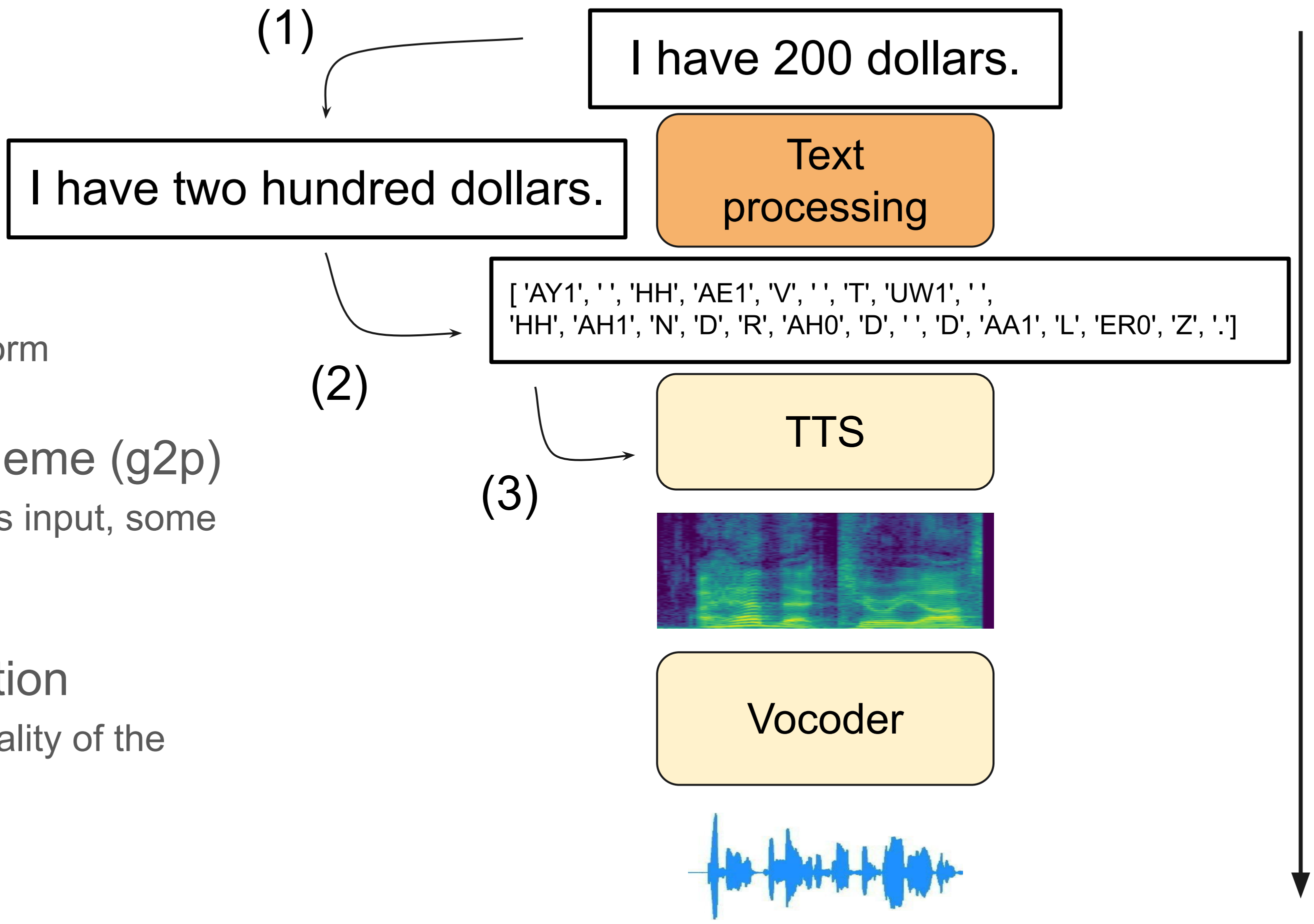
TTS 解說

<u>“Text Processing”</u>

(1) Text normalization
written form to spoken form

(2) Grapheme to Phoneme (g2p)
If TTS takes phoneme as input, some also use char.

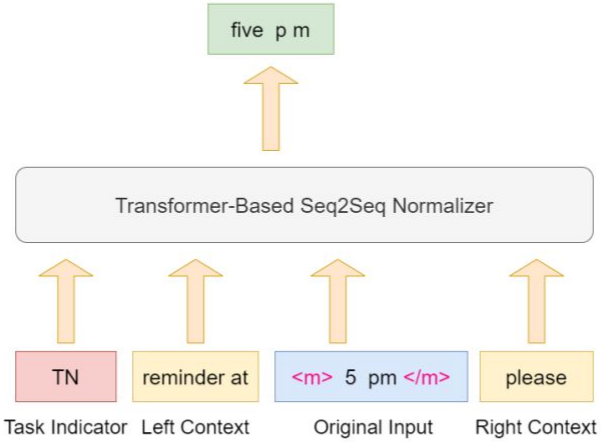
(3) Text feature extraction
optional, improve the quality of the generated speech

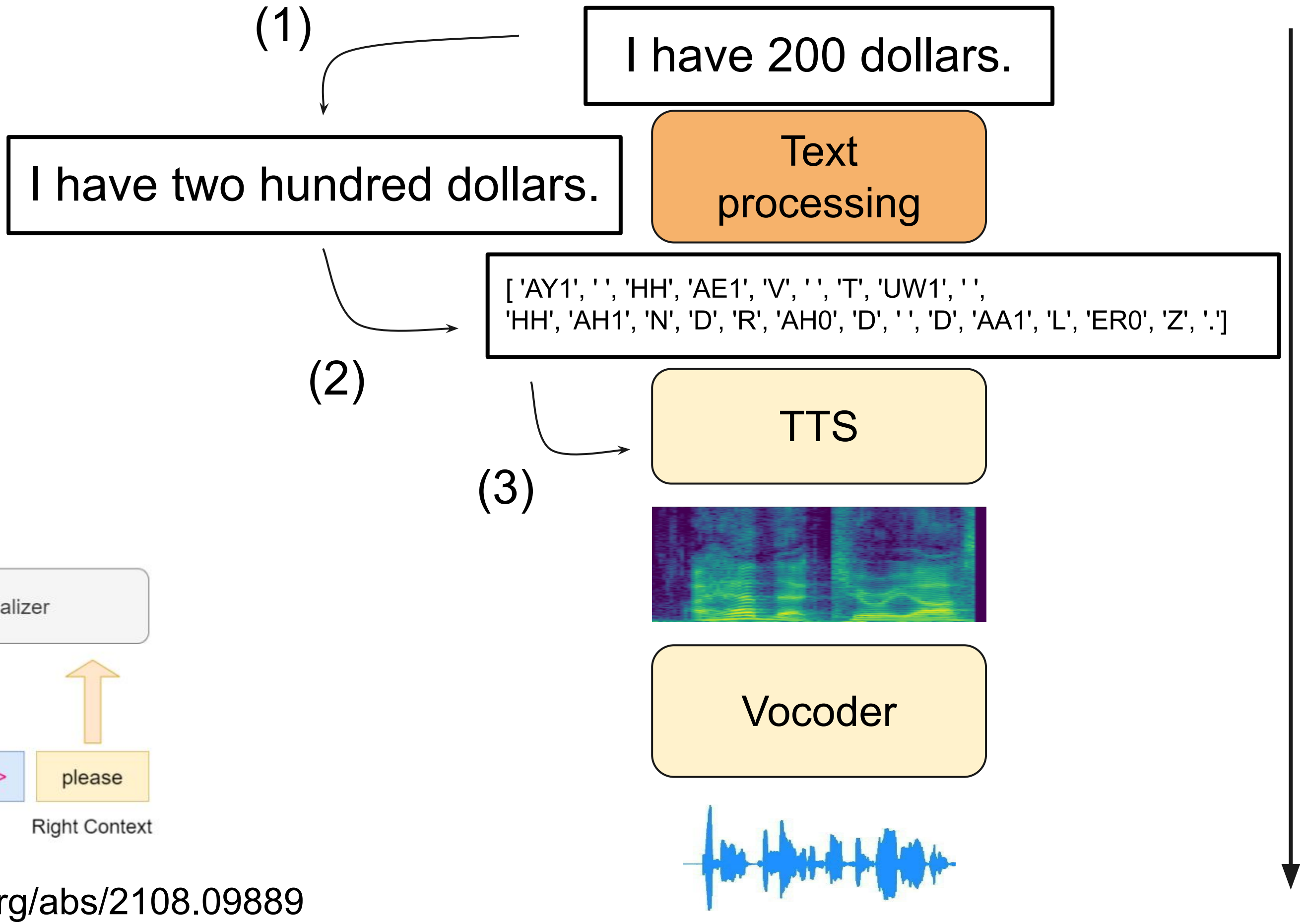


TTS 解說

“Text Processing”

(1) Text normalization



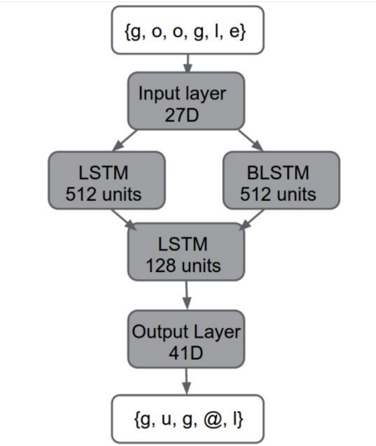


[Lai et.al., arXiv’21] https://arxiv.org/abs/2108.09889

TTS 解說

“Text Processing”

(2) Grapheme to Phoneme (g2p)



I have 200 dollars.

(1)

Text processing

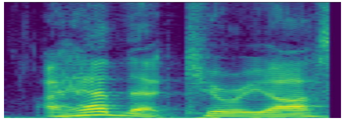
I have two hundred dollars.

(2)

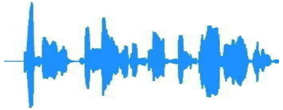
['AY1', ' ', 'HH', 'AE1', 'V', ' ', 'T', 'UW1', ' ', 'HH', 'AH1', 'N', 'D', 'R', 'AH0', 'D', ' ', 'D', 'AA1', 'L', 'ER0', 'Z', '.']

(3)

TTS



Vocoder

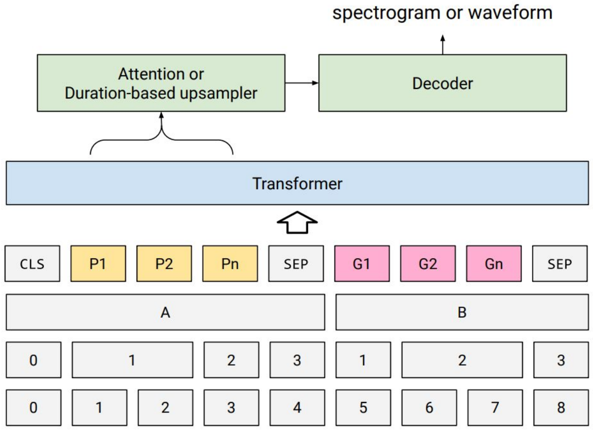


[Rao et.al., ICASSP’15] https://ieeexplore.ieee.org/document/7178767
[Park, Lee, INTERSPEECH’20] https://arxiv.org/abs/2004.03136

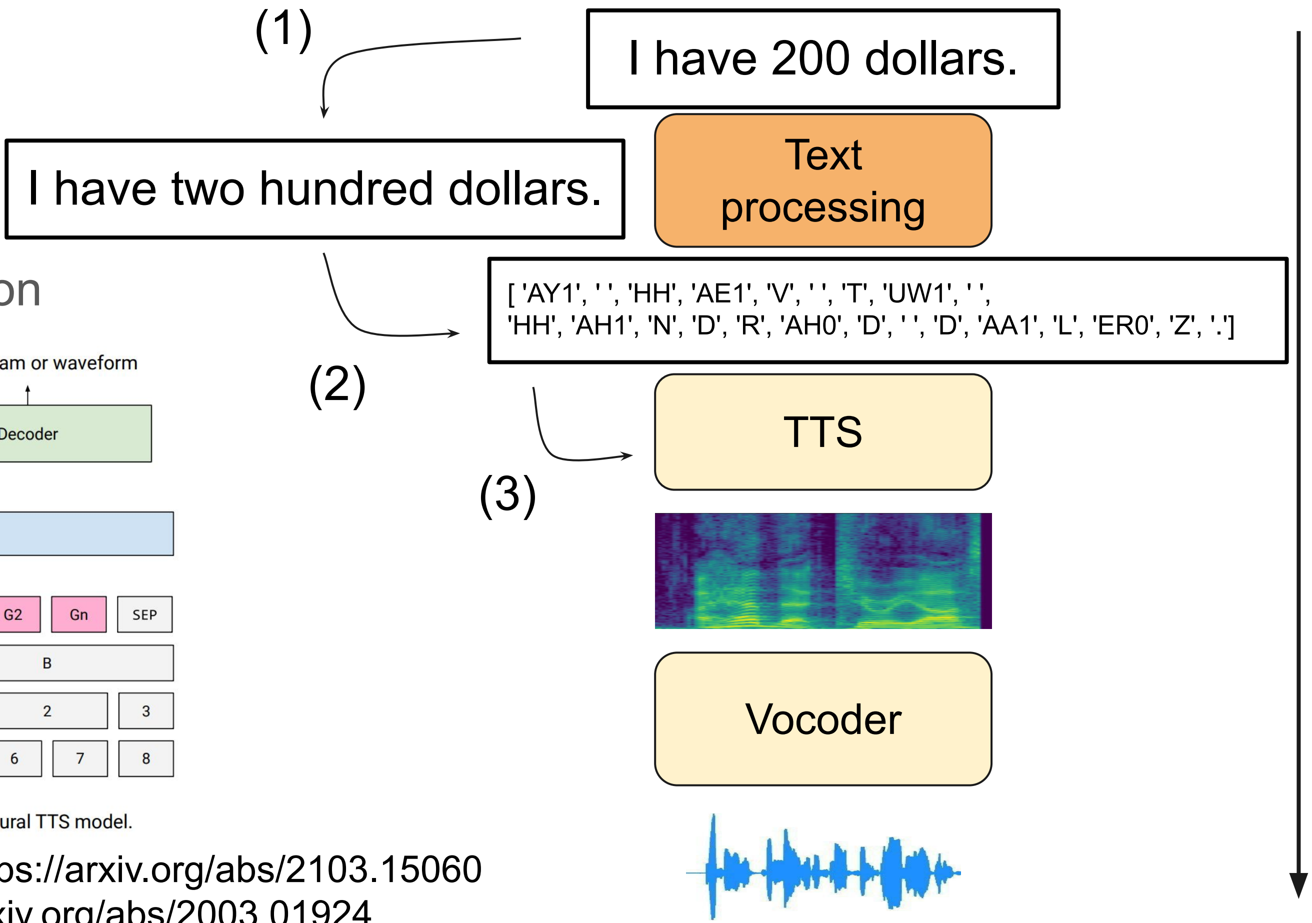
TTS 解說

“Text Processing”

(3) Text feature extraction



(b) Using PnG BERT as the encoder for a neural TTS model.

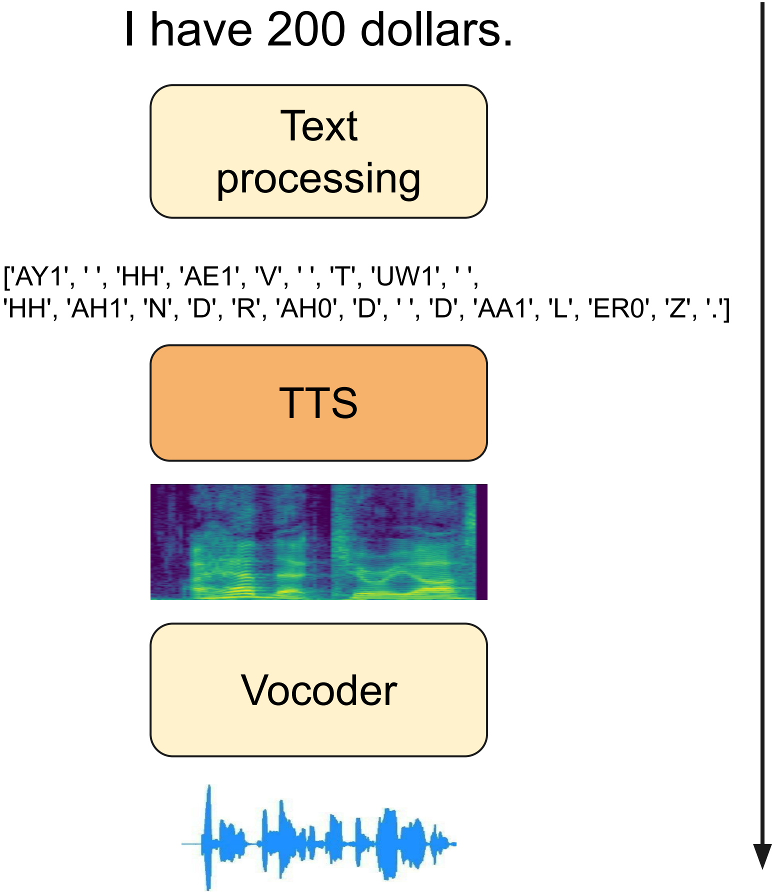


[Jia et.al., INTERSPEECH’21] https://arxiv.org/abs/2103.15060
[Sun et.al., ICASSP’20] https://arxiv.org/abs/2003.01924

TTS 解說

<u>“Text to Mel: TTS”</u>

- Input text (phoneme, char., other feature)
- Output mel
- Text Encoder
- Mel Decoder



TTS 兩大支柱

- Tacotron2
- FastSpeech2

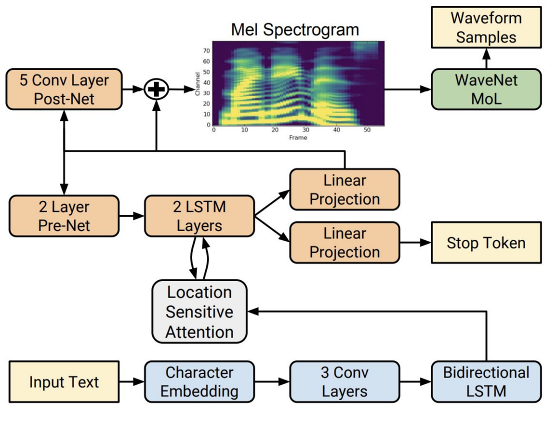


Fig. 1. Block diagram of the Tacotron 2 system architecture.

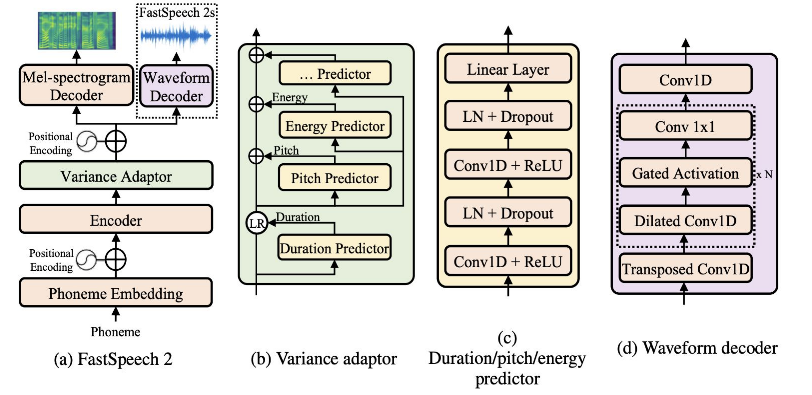
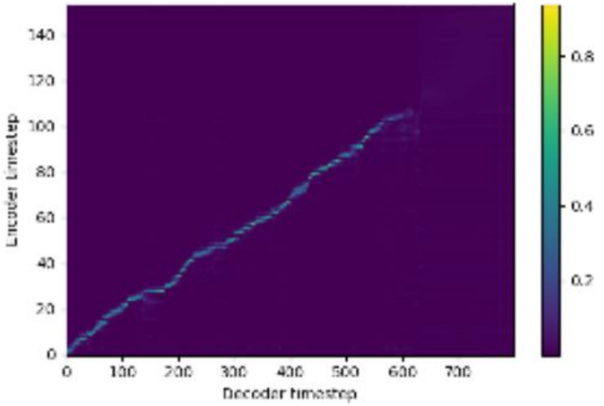


Figure 1: The overall architecture for FastSpeech 2 and 2s. LR in subfigure (b) denotes the length regulator proposed in FastSpeech. LN in subfigure (c) denotes layer normalization.

[Shen et.al., ICASSP'18] https://arxiv.org/abs/1712.05884
[Ren et.al., ICLR'21] https://arxiv.org/abs/2006.04558

TTS 兩大支柱 (1)

- Tacotron2
  - Sample Code
  - Autoregressive
  - High quality
  - Slow



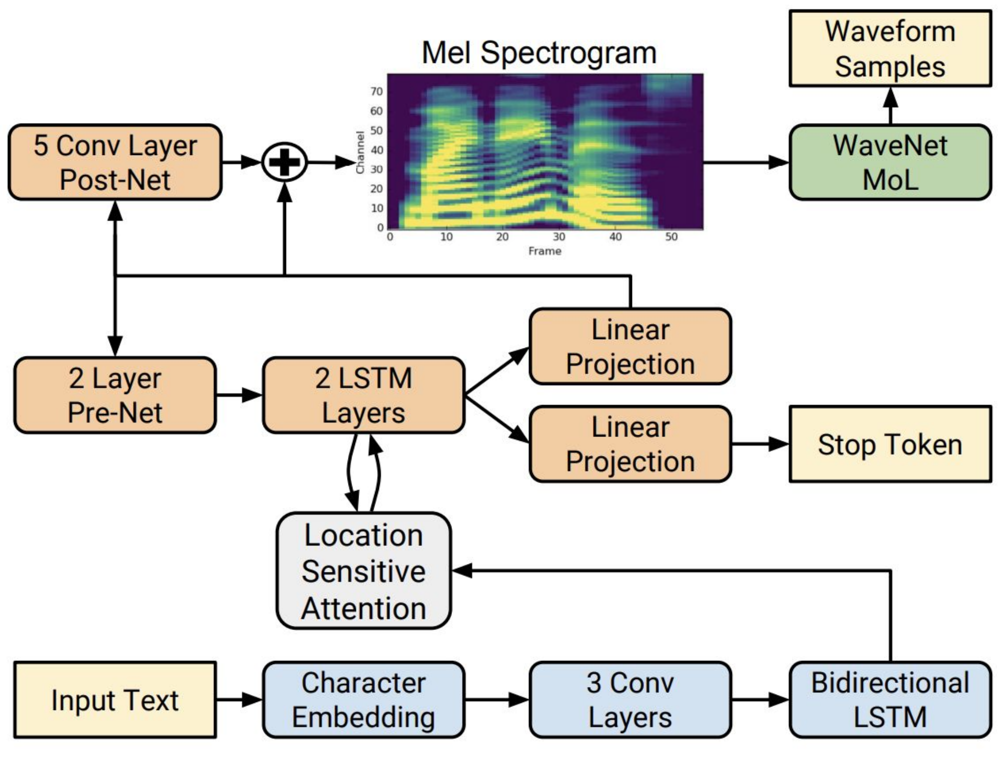


Fig. 1. Block diagram of the Tacotron 2 system architecture.

TTS 兩大支柱 (2)

- FastSpeech2
  - Sample Code
  - Non Autoregressive
  - Fast
  - Need duration data to train (duration modeling)

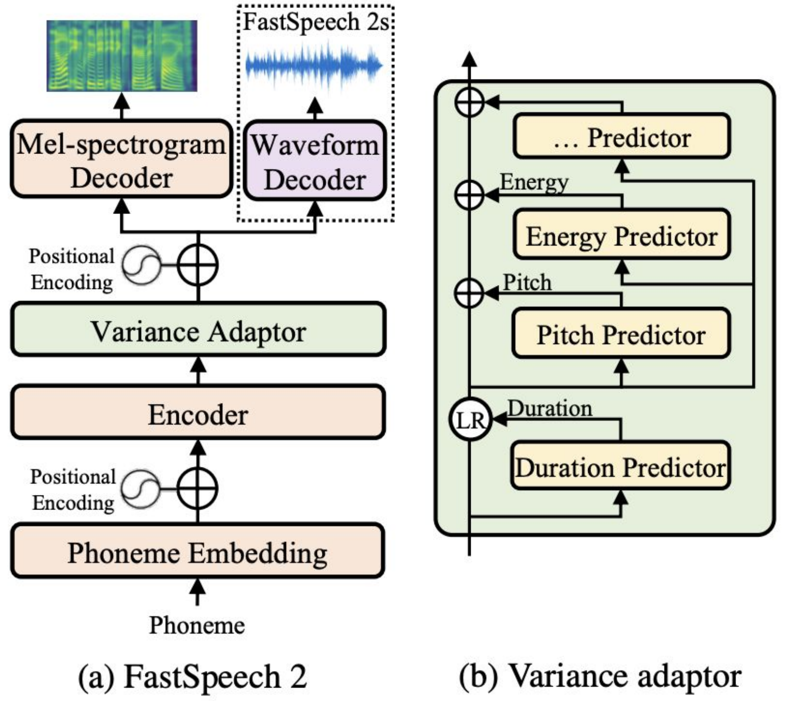


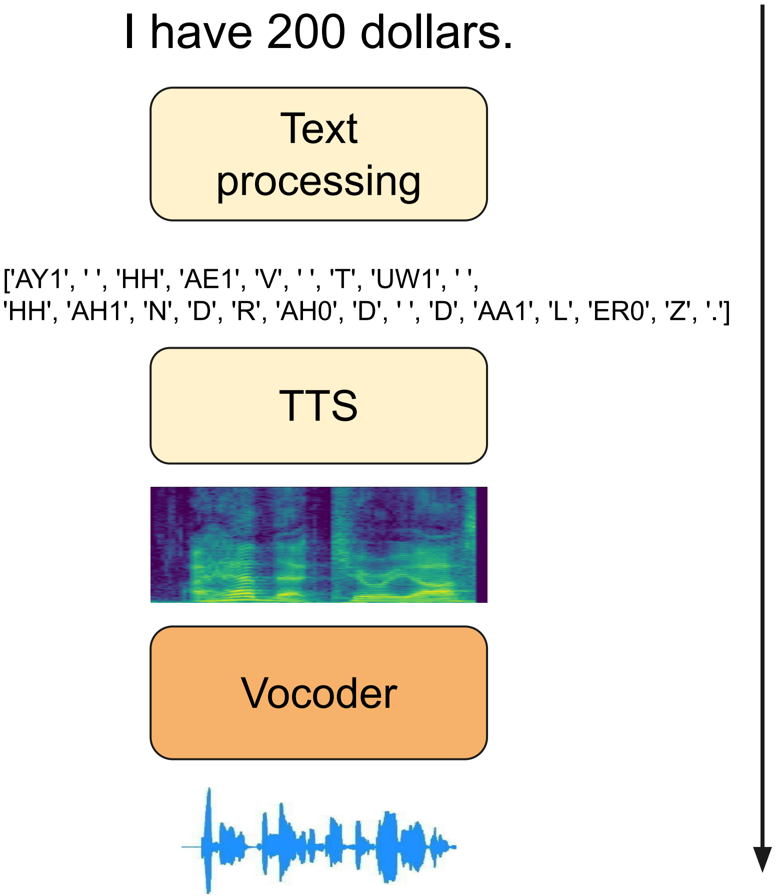
Figure 1: The overall architecture for FastSpeech 2
regulator proposed in FastSpeech. LN in subfigure (

TTS 解說

“Vocoder”

- Input: mel
- Output: waveform
- Griffin-Lim
- Neural Vocoder
  - WaveNet
  - WaveRNN
  - MelGAN
  - **HifiGAN**

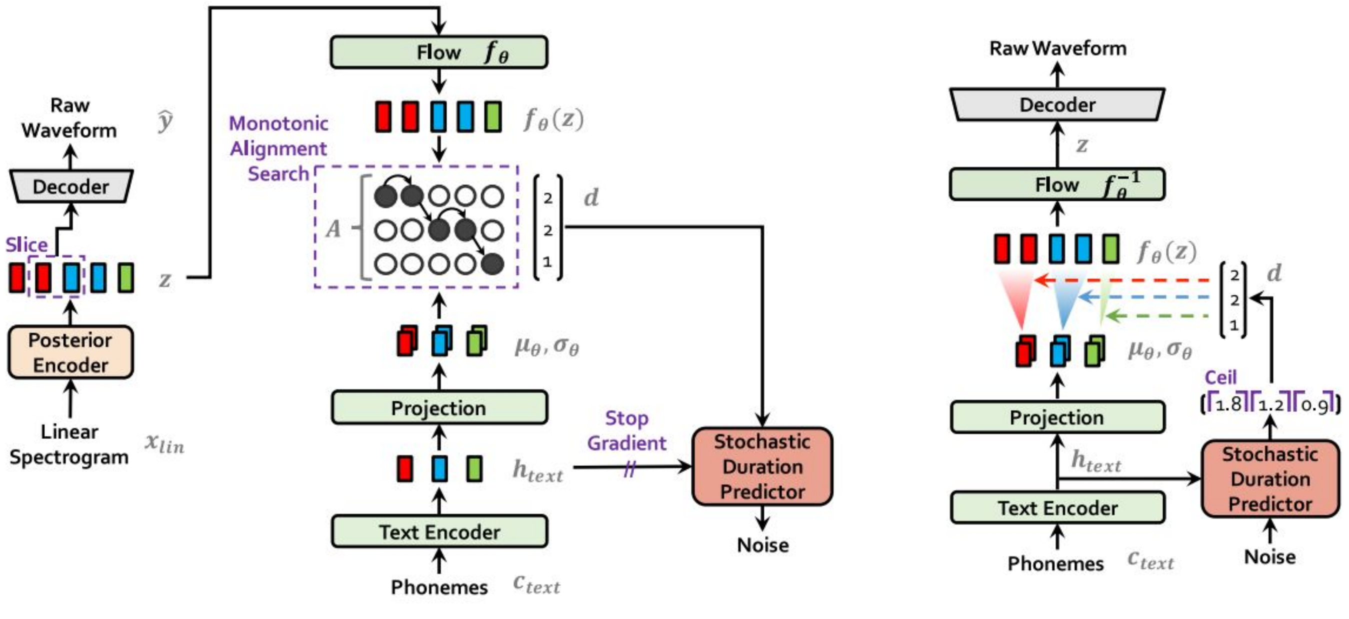
[Kong et.al., NeurIPS’20] https://arxiv.org/abs/2010.05646



TTS SOTA

[Kim et.al., ICML'21] https://arxiv.org/abs/2106.06103

- VITS



(a) Training procedure

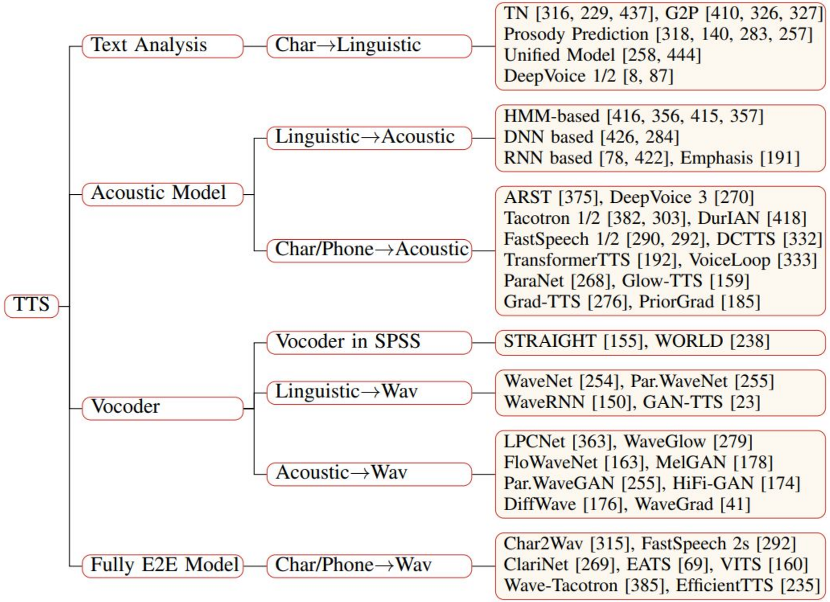
(b) Inference procedure

- VITS 後續發展: https://zhuanlan.zhihu.com/p/474601997

TTS Survey Paper

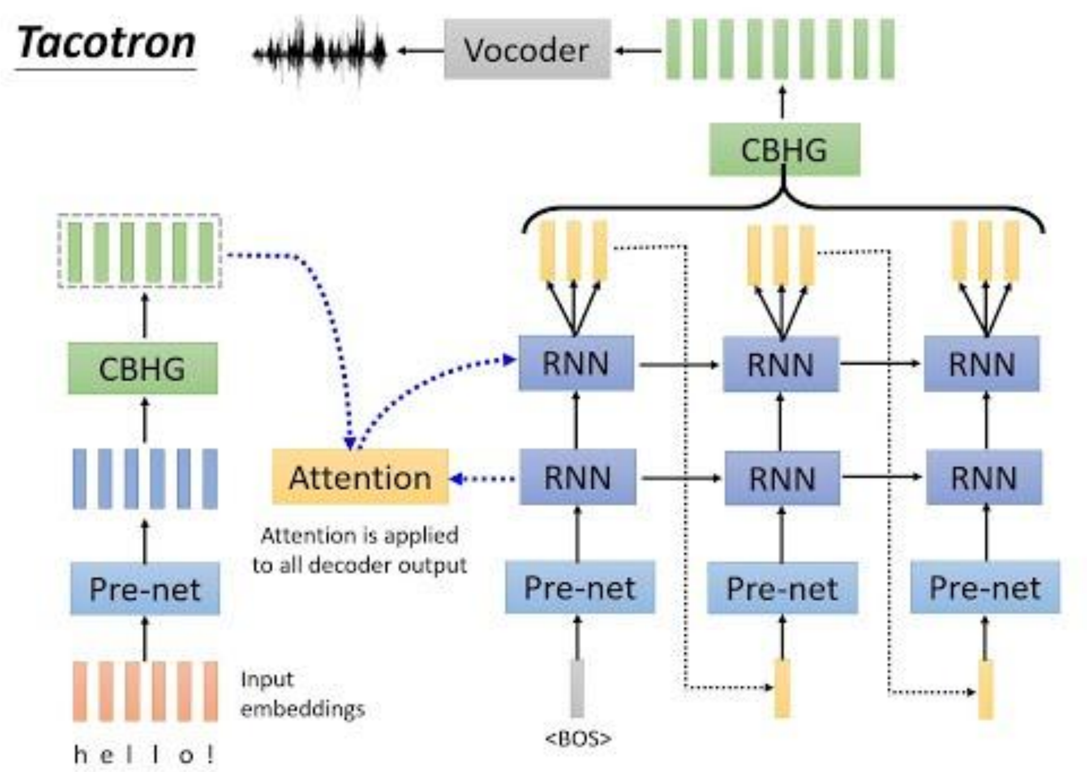
A Survey on Neural Speech Synthesis

[Tan et.al., arXiv'21]https://arxiv.org/abs/2106.15561



(a) A taxonomy of neural TTS.

[DLHLP 2020] Speech Synthesis



範例程式碼 – TTS systems

Example Code – TTS Systems

sample code: https://github.com/hhhaaahhhaa/TTS-systems

- single speaker (LJSpeech)
- mult-speaker (LibriTTS & AISHELL-3, embedding table)

based on:

https://github.com/BogiHsu/Tacotron2-PyTorch (star here)

https://github.com/ming024/FastSpeech2 (star here)

Installation

Create a new environment first…

- git clone https://github.com/hhhaaahhhaa/TTS-systems.git
- cd TTS-systems
- pip install -r requirements.txt

Datasets

LJSpeech: https://keithito.com/LJ-Speech-Dataset/

LibriTTS: http://www.openslr.org/60/ (只需要 train-clean-100/dev-clean/test/clean)

AISHELL-3: https://www.aishelltech.com/aishell_3, https://www.openslr.org/93/

Alignments: https://drive.google.com/drive/folders/1OyEh823slo4Taw9A-zIC9ruS45hz8Y81?usp=sharing (unzip to preprocessed_data/[DATASET_NAME]/TextGrid)

Preprocess

Run the following commands. (Remember to prepare textgrid first.)

python preprocess.py [raw_dir] [preprocessed_dir]

    --dataset [DATASET_TAG] --parse_raw --preprocess

python clean.py [preprocessed_dir] [clean_results_path] (已跑好)

python preprocess.py [raw_dir] [preprocessed_dir]

    --dataset [DATASET_TAG] --create_dataset [clean_results_path] (已跑好)

Train

python fastspeech2_train.py (or tacotron2_train.py)

```
--stage train
--data_config [data_config_dir] --model_config [model_config_path]
--train_config [train_config_path] --algorithm_config [algorithm_config_path]
--exp_name [experiment name]
```

- Remember to change to your local path inside data_config
- Results are under output/[exp_name]
- For multispeaker dataset, use base-multispeaker.yaml as model_config

Inference

Modify the parameters in script.

python fastspeech2_inference.py

python tacotron2_inference.py

Vocoder

Change inside model_config or inference scripts.

Default use **HifiGAN**.

Evaluation

It is hard to fairly evaluate TTS systems!

Subjective (Higher credibility if well-conducted, high cost, not scalable):

- Mean Opinion Score (MOS)

Objective (Use ASR model, low or zero cost, scalable):

- Word Error Rate
- Character Error Rate

Helpful packages: SpeechRecognition, jiwer

We provide some useful functions in evaluation.py, feel free to use them.

Other

- Tacotron2 training 至少要一天以上，注意作業時間
- batch size (inside train_config) 可以開大一點
- How do we fetch data? Refer to the code in
  - datasets/FastSpeech2Dataset.py
  - datasets/Tacotron2Dataset.py

報告方向

作業

從助教提供的 Sample code 出發，理解 TTS 在做什麼

基本問題

1. 文字如何被處理成模型輸入?
2. 下載的原始音檔經過了甚麼前處理？
3. Input (text, feature…) “長什麼樣子” (e.g. alignment, spker embed…)
4. Output “長什麼樣子” (e.g. mel visualize, pitch contour)
5. 不同的TTS模型分別算了哪些 loss?
6. 訓練過程 (training curve, 大概要訓練多久)
7. TTS模型生成聲音的速度如何(生成 x 秒的音檔需要 y 秒)
8. Demo (好的例子/生壞的例子/缺點是甚麼)

Run Sample Code

1. Try different architectures. (Non-autoregressive vs Autoregressive)
2. Try different text inputs. (Character vs Phoneme, Tacotron2 input is set to character, try to modify the code under datasets/)
3. Play around with inference.

Other Languages - Japanese

JSUT: https://sites.google.com/site/shinnosuketakamichi/publication/jsut

JSUT alignments: https://github.com/sarulab-speech/jsut-label

1. Generate JSUT textgrid(python -m scripts.jsut_hts2textgrid).
2. Preprocess (Dataset tags in Parsers/__init__.py)
3. Clean & split dataset.
4. Collect phoneme set (python -m scripts.collect_phonemes) and register them to text/define.py.
5. Create data config yaml file and train(single speaker).
6. Write a script to inference.

Other Languages - Korean

KSS:
https://drive.google.com/file/d/1v9rBWURlteQImf81MpDXTY3HNGSNI06Q/view?usp=drive_link

Prepare TextGrid

1. Install MFA2.0
   https://montreal-forced-aligner.readthedocs.io/en/latest/installation.html.
2. Download the prepared acoustic model to MFA/kss/acoustic_model.zip.
   https://drive.google.com/drive/folders/1puGSyTi4__8I2cGysxa4GwKcg6WyVIkE?usp=drive_link
3. Run preprocess.py with --parse_raw --prepare_mfa.
4. Generate dictionary (python -m scripts.kss).
5. ~~Train an MFA model~~ (prepared by TA)
6. Run preprocess.py with --mfa.

Evaluate TTS Models

1. Write scripts to inference the test set.
2. Write scripts to calculate CER by automatic speech recognition.

ESPnet-TTS

- Architecture
- Speaker
- Language
- Vocoders

[Colab Demo (inference)]

TTS: Text-to-speech

- Architecture
  - Tacotron2
  - Transformer-TTS
  - FastSpeech
  - FastSpeech2
  - Conformer FastSpeech & FastSpeech2
  - VITS
- Multi-speaker & multi-language extention
  - Pretrined speaker embedding (e.g., X-vector)
  - Speaker ID embedding
  - Language ID embedding
  - Global style token (GST) embedding
  - Mix of the above embeddings
- End-to-end training
  - End-to-end text-to-wav model (e.g., VITS)
  - Joint training of text2mel and vocoder
- Various language support
  - En / Jp / Zn / De / Ru / And more...
- Integration with neural vocoders
  - Parallel WaveGAN
  - MelGAN
  - Multi-band MelGAN
  - HiFiGAN
  - StyleMelGAN
  - Mix of the above models

其它

MiniConda3

下載: https://docs.conda.io/en/latest/miniconda.html
教學: https://simplelearn.tw/python-conda-virtual-environment/

以下針對TWCC 國網使用者

- 將 /home/[user]/.local/.pip/pip.conf 裡面 user 值改成 false
- 不然 package 會預設裝在 user 路徑下(/home/[user]/.local/.lib/)而非環境裡面

TWCC相關

檔案傳輸

https://man.twcc.ai/@twccdocs/doc-hfs-main-zh/%2F%40twccdocs%2Fguide-hfs-connect-to-data-transfer-node-zh

Tensorboard

https://man.twcc.ai/@twccdocs/howto-ccs-launch-tensorboard-zh

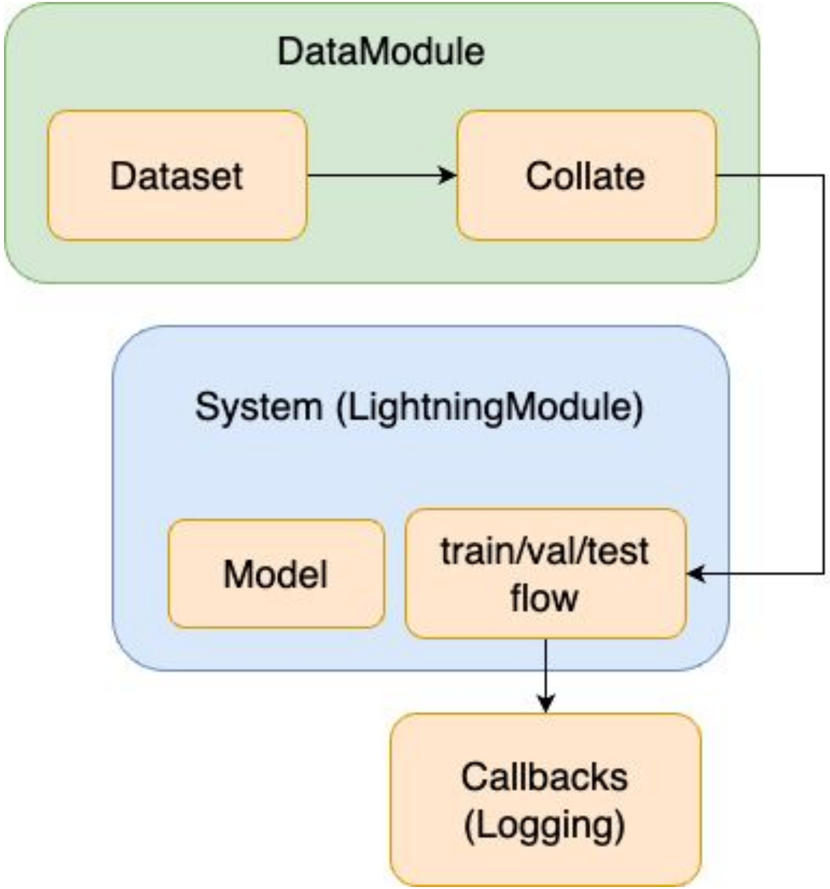
Google雲端下載指令

https://chemicloud.com/blog/download-google-drive-files-using-wget/

Pytorch Lightning

Modularize standard deep learning pipeline.

- Dataset
- DataModule
- Collate
- Models (torch.nn.Module)
- Systems (LightningModule)
- Callbacks



Documentation

Introduction

https://pytorch-lightning.readthedocs.io/en/stable/starter/introduction.html

LightningModule

https://pytorch-lightning.readthedocs.io/en/stable/common/lightning_module.html

LightningDataModule

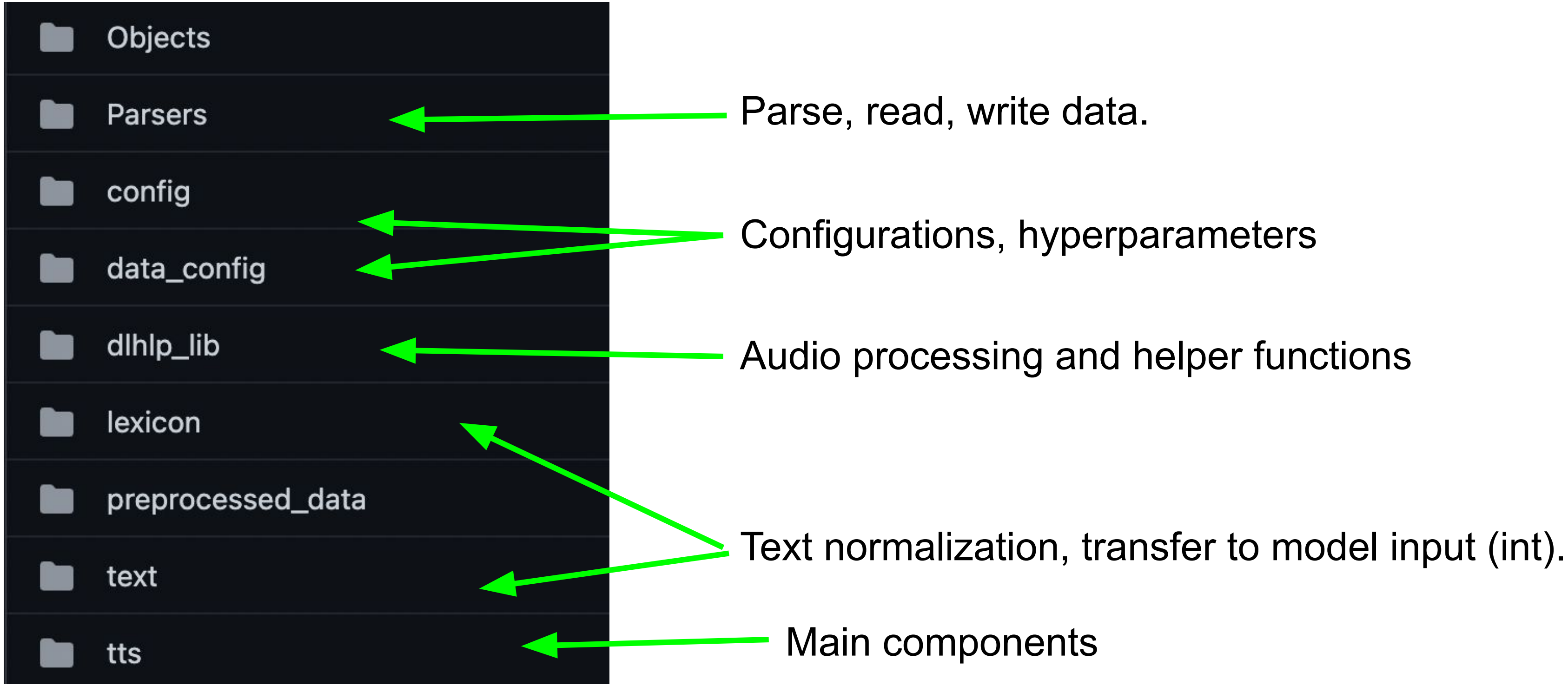
https://pytorch-lightning.readthedocs.io/en/latest/data/datamodule.html

Some notes on data flow...

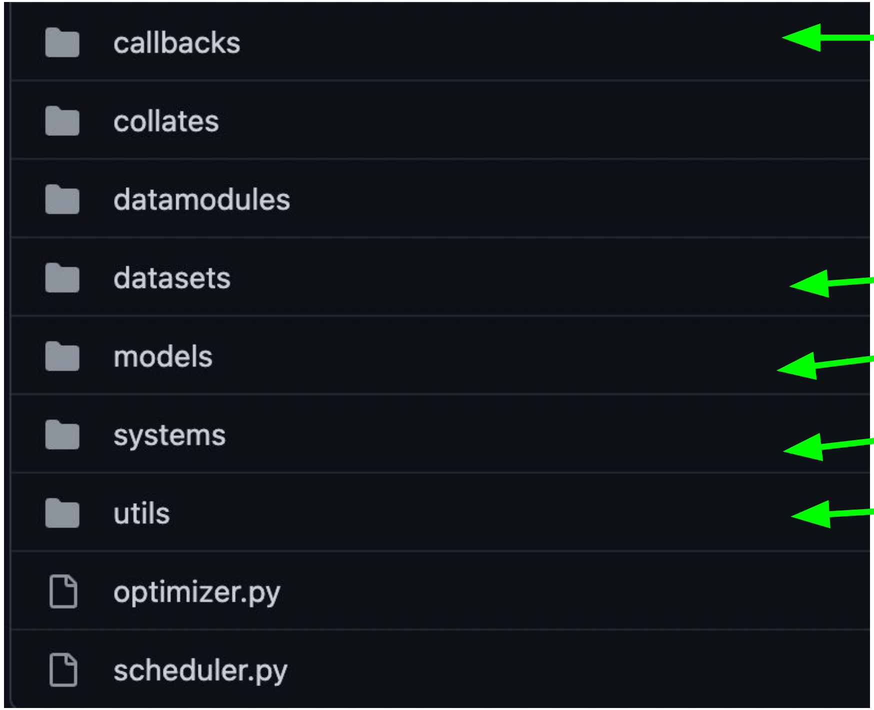
1. **Dataset** returns data.
2. Pass data through **collate** function to form a batch. (padding, batching, etc)
3. Pass batch through **system**'s train/val/test steps.
4. Output of train/val/test steps is passed to callbacks(**Saver**).

- Data passing is already handled by pytorch-lightning package internally.
- Pytorch lightning will also handle how to tranfer data between CPU/GPU.
- Just focus on input and output of every component.

Code Structure



Code Structure (tts/)



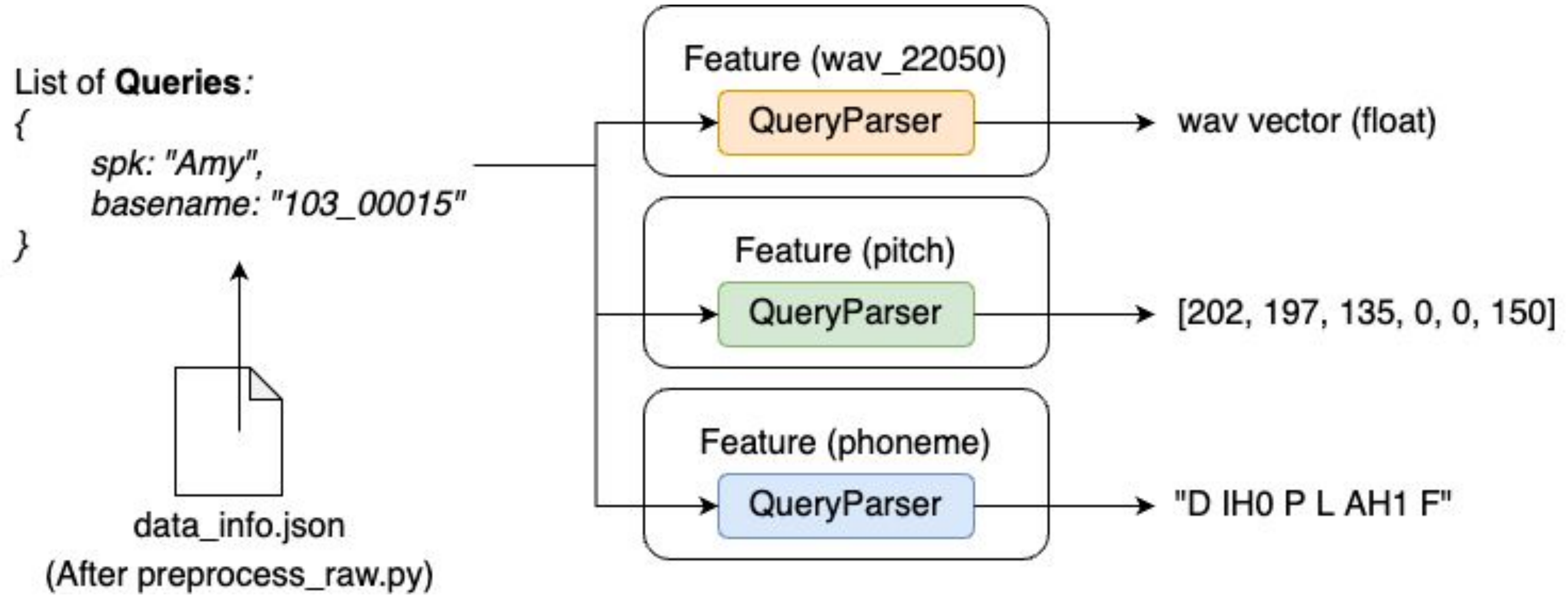
Logging

Torch datasets

Model architectures

LightningModule (train/val/test step)

Helper functions.



- Filename operations are automatically handled by data parser.

  wav = wav_feature.read_from_query(query)   # read data

  wav_feature.save(wav, query)   # save data