语音合成 – 文本转语音TTS

流程框图

文本 提取 拼音和韵律
- 文本规则化
- 文本转拼音
- 拼音分解成音素

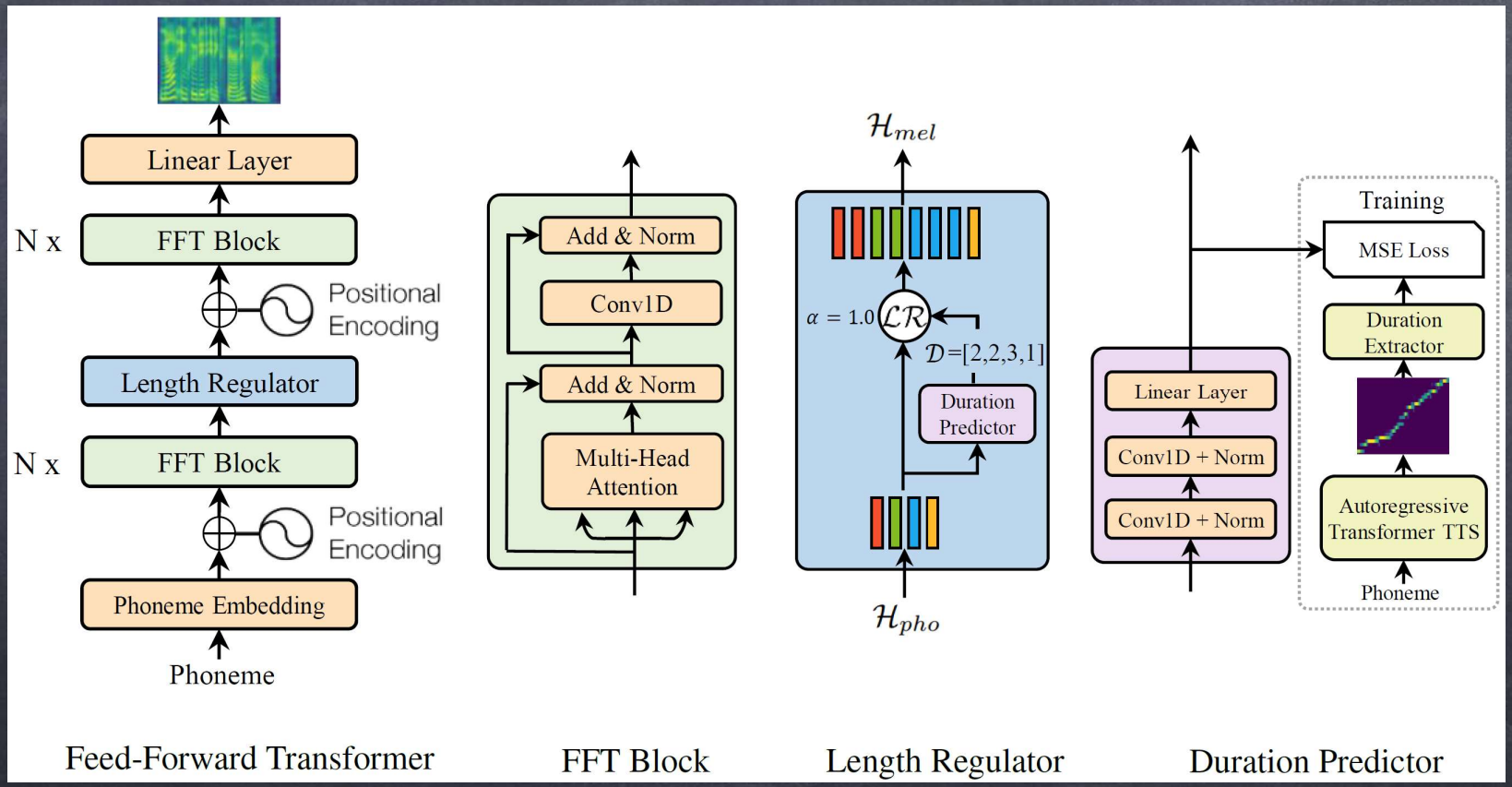
语言特征 生成 声学特征
- 语速、停顿 控制
- 韵律预测
- 文本与音素对齐

声学特征 生成 语音信号
- 选择多个音素音节优选路径
- 声波参数生成, 参数合成

FastSpeech

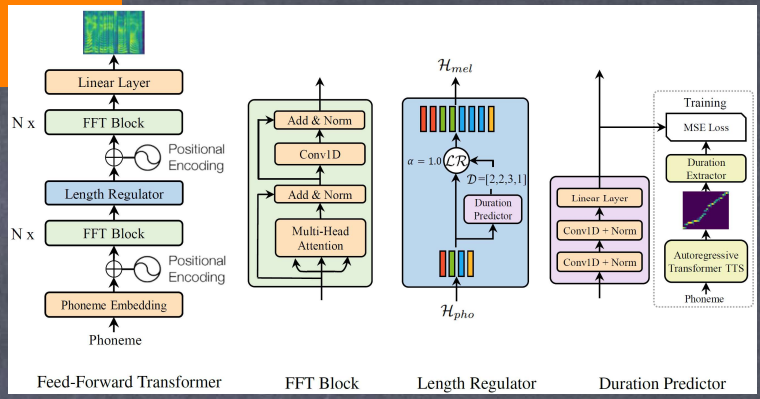
网络架构

- FastSpeech 非自回归 编码器+解码器架构



FastSpeech

网络架构



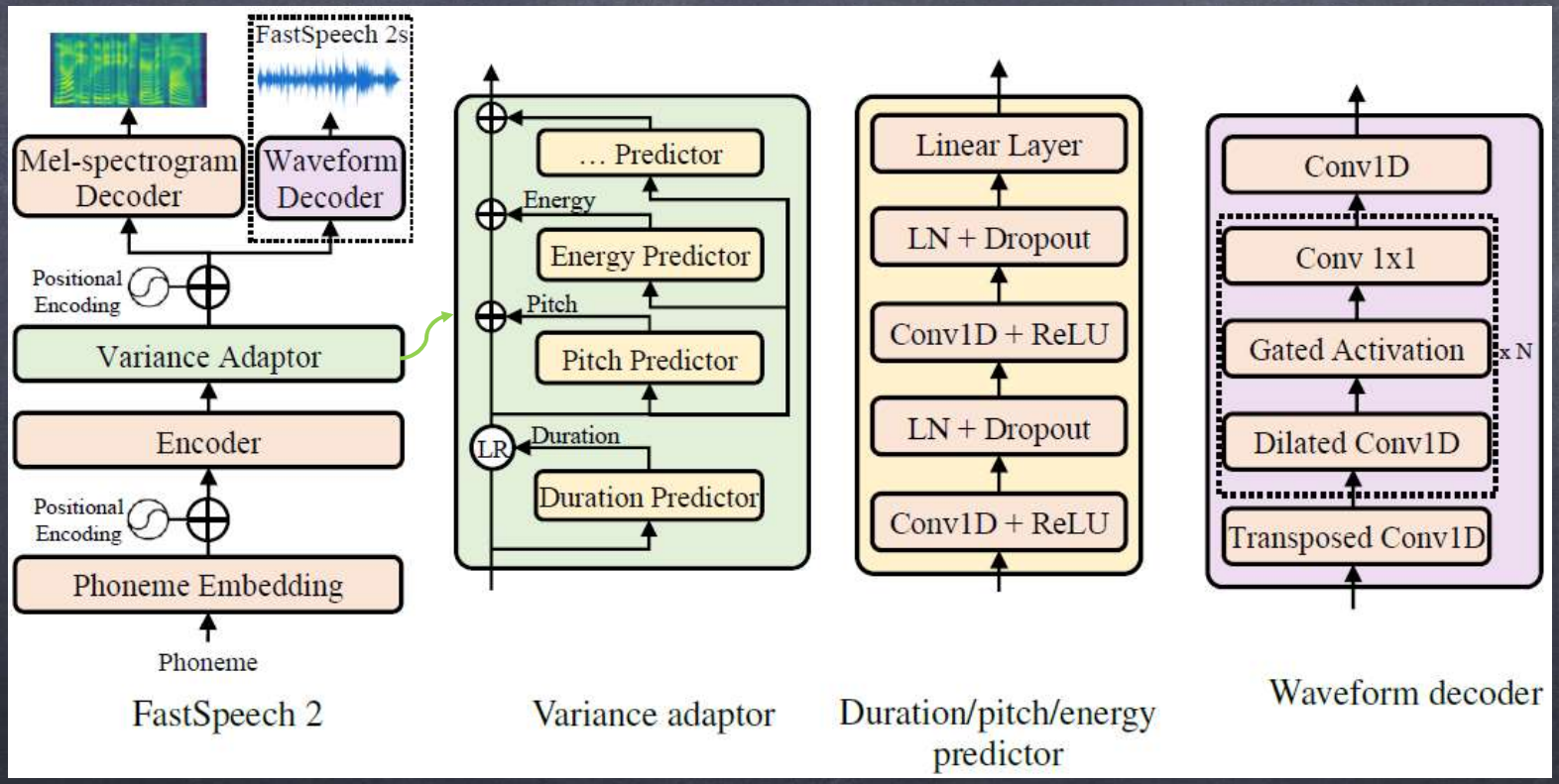
- FastSpeech 非自回归 编码器+解码器架构

postnet = TFTacotronPostnet	
mel_dense = tf.keras.layers.Dense	
Decoder = TFFastSpeechDecoder	
decoder_pos	range, expand_dims
length_regulator	reduce_sum, tf.repeat , pad, expand_dims
duration_predictor	Conv1D, LayerNorm, relu6, Sequential, Dense, relu6, squeeze
Encoder = TFFastSpeechEncoder	attention (SelfAtten, dense, LN), intermediate(Conv1D,mish), dropout + LayerNorm, expand_dims + cast_mask, add
Phoneme Embedding	position_embedding

FastSpeech2

网络架构

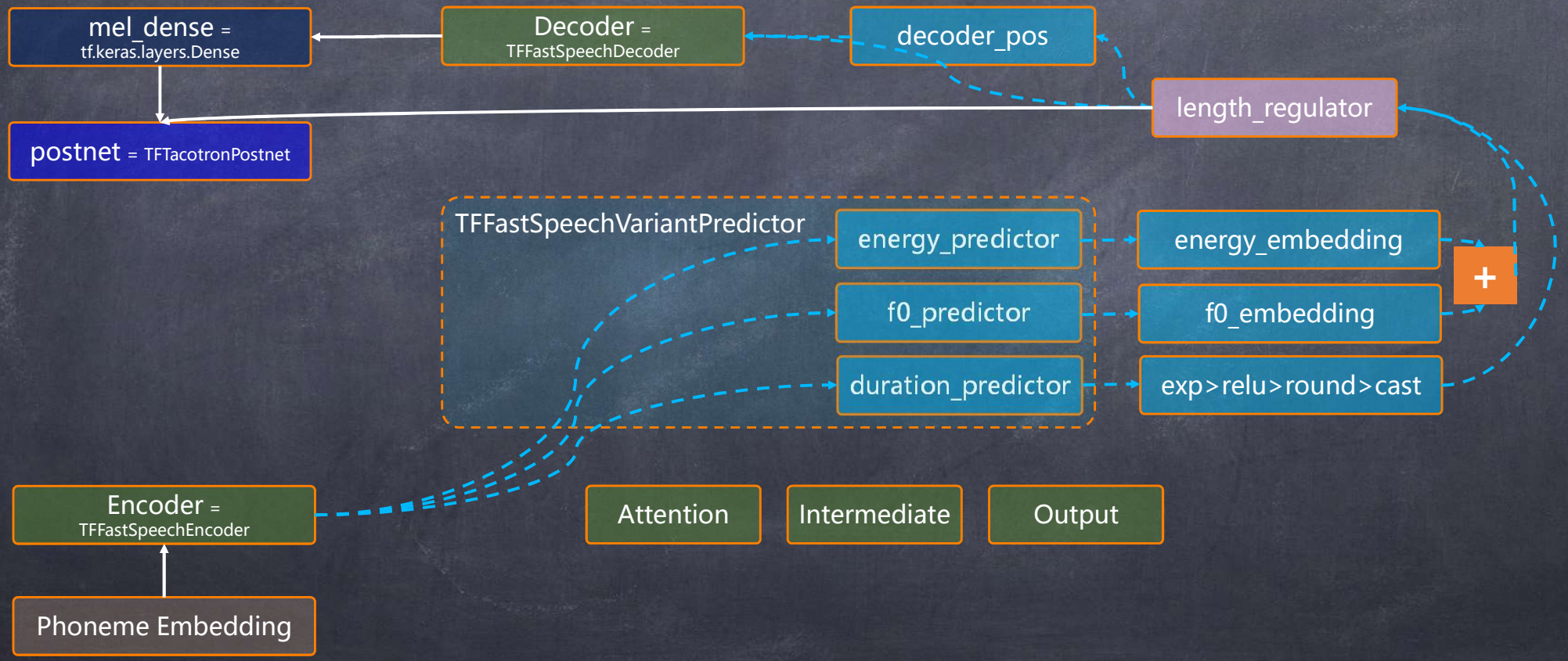
- FastSpeech2 非自回归 编码器+解码器架构



FastSpeech2

网络架构

- FastSpeech 非自回归 编码器+解码器架构



FastSpeech2

网络结构与数据处理

```python
class SavableTFFastSpeech2(TFFastSpeech2):
    def __init__(self, config, **kwargs):
        super().__init__(config, **kwargs)

    def call(self, inputs, training=False):
        input_ids, speaker_ids, speed_ratios, f0_ratios, energy_ratios = inputs
        return super()._inference(
            input_ids, speaker_ids, speed_ratios, f0_ratios, energy_ratios
        )

    def _build(self):
        input_ids = tf.convert_to_tensor([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]], tf.int32)
        speaker_ids = tf.convert_to_tensor([0], tf.int32)
        speed_ratios = tf.convert_to_tensor([1.0], tf.float32)
        f0_ratios = tf.convert_to_tensor([1.0], tf.float32)
        energy_ratios = tf.convert_to_tensor([1.0], tf.float32)
        self([input_ids, speaker_ids, speed_ratios, f0_ratios, energy_ratios])
```

```python
from tensorflow_tts.inference import AutoProcessor, AutoConfig, TFAutoModel

processor = AutoProcessor.from_pretrained(pretrained_path = "processor.json")
config = AutoConfig.from_pretrained("config.yml")
fastspeech2 = TFAutoModel.from_pretrained(pretrained_path = "model.h5", config=config)

text = "这是一个开源的端到端中文语音合成系统"
input_ids = processor.text_to_sequence(text, inference=True)

mel_before, mel_after, duration_outputs, _, _ = fastspeech2.inference(
    input_ids=tf.expand_dims(tf.convert_to_tensor(input_ids, dtype=tf.int32), 0),
    speaker_ids=tf.convert_to_tensor([0], dtype=tf.int32),
    speed_ratios=tf.convert_to_tensor([1.0], dtype=tf.float32),
    f0_ratios =tf.convert_to_tensor([1.0], dtype=tf.float32),
    energy_ratios =tf.convert_to_tensor([1.0], dtype=tf.float32),
)
```

```
{
    "symbol_to_id": {…
    },
    "id_to_symbol": {…
    },
    "speakers_map": {
        "baker": 0
    },
    "processor_name": "BakerProcessor",
    "pinyin_dict": {…
    }
}
```

```yaml
# FEATURE EXTRACTION SETTING
hop_size: 256            # Hop size.
format: "npy"
# NETWORK ARCHITECTURE SETTING
model_type: "fastspeech2"
fastspeech2_params:
    dataset: baker
    n_speakers: 1
    encoder_hidden_size: 256
    encoder_num_hidden_layers: 3
    encoder_num_attention_heads: 2
    encoder_attention_head_size: 16  # in v1, = 384//2
    encoder_intermediate_size: 1024
    encoder_intermediate_kernel_size: 3
    encoder_hidden_act: "mish"
    decoder_hidden_size: 256
    decoder_num_hidden_layers: 3
    decoder_num_attention_heads: 2
    decoder_attention_head_size: 16  # in v1, = 384//2
    decoder_intermediate_size: 1024
    decoder_intermediate_kernel_size: 3
    decoder_hidden_act: "mish"
    variant_prediction_num_conv_layers: 2
    variant_predictor_filter: 256
    variant_predictor_kernel_size: 3
    variant_predictor_dropout_rate: 0.5
    num_mels: 80
    hidden_dropout_prob: 0.2
    attention_probs_dropout_prob: 0.1
    max_position_embeddings: 2048
    initializer_range: 0.02
    output_attentions: False
    output_hidden_states: False
```

FastSpeech2 网络结构代码

```python
class FastSpeechConfig(BaseConfig):
    """Initialize FastSpeech Config."""

    def __init__(
        self,
        dataset="ljspeech",
        vocab_size=len(lj_symbols),
        n_speakers=1,
        encoder_hidden_size=384,
        encoder_num_hidden_layers=4,
        encoder_num_attention_heads=2,
        encoder_attention_head_size=192,
        encoder_intermediate_size=1024,
        encoder_intermediate_kernel_size=3,
        encoder_hidden_act="mish",

        decoder_hidden_size=384,
        decoder_num_hidden_layers=4,
        decoder_num_attention_heads=2,
        decoder_attention_head_size=192,
        decoder_intermediate_size=1024,
        decoder_intermediate_kernel_size=3,
        decoder_hidden_act="mish",

        output_attentions=True,
        output_hidden_states=True,
        hidden_dropout_prob=0.1,
        attention_probs_dropout_prob=0.1,
        initializer_range=0.02,
        layer_norm_eps=1e-5,
        max_position_embeddings=2048,
        num_duration_conv_layers=2,
        duration_predictor_filters=256,
        duration_predictor_kernel_sizes=3,
        num_mels=80,
        duration_predictor_dropout_probs=0.1,
        n_conv_postnet=5,
        postnet_conv_filters=512,
        postnet_conv_kernel_sizes=5,
        postnet_dropout_rate=0.1,
        **kwargs
    ):
```

```python
# encoder params
self.encoder_self_attention_params = SelfAttentionParams(
    n_speakers=n_speakers,  # 1
    hidden_size=encoder_hidden_size,  # 384
    num_hidden_layers=encoder_num_hidden_layers,      # 4
    num_attention_heads=encoder_num_attention_heads,  # 2
    attention_head_size=encoder_attention_head_size,  # 192
    hidden_act=encoder_hidden_act,                    # "mish"
    intermediate_size=encoder_intermediate_size,      # 1024
    intermediate_kernel_size=encoder_intermediate_kernel_size, # 3
    output_attentions=output_attentions,              # True
    output_hidden_states=output_hidden_states,        # True
    initializer_range=initializer_range,              # 0.02
    hidden_dropout_prob=hidden_dropout_prob,          # 0.1
    attention_probs_dropout_prob=attention_probs_dropout_prob, # 0.1
    layer_norm_eps=layer_norm_eps,                    # 1e-5
    max_position_embeddings=max_position_embeddings, # 2048
)
```

256
3
2
16
“mish”
1024
3

Encoder 和 Decoder 结构完全一样

```python
# decoder params
self.decoder_self_attention_params = SelfAttentionParams(
    n_speakers=n_speakers,                    # 1
    hidden_size=decoder_hidden_size,          # 384
    num_hidden_layers=decoder_num_hidden_layers,      # 4
    num_attention_heads=decoder_num_attention_heads,  # 2
    attention_head_size=decoder_attention_head_size,  # 192
    hidden_act=decoder_hidden_act,                    # "mish"
    intermediate_size=decoder_intermediate_size,      # 1024
    intermediate_kernel_size=decoder_intermediate_kernel_size, # 3
    output_attentions=output_attentions,              # True
    output_hidden_states=output_hidden_states,        # True
    initializer_range=initializer_range,              # 0.02
    hidden_dropout_prob=hidden_dropout_prob,          # 0.1
    attention_probs_dropout_prob=attention_probs_dropout_prob, # 0.1
    layer_norm_eps=layer_norm_eps,                    # 1e-5
    max_position_embeddings=max_position_embeddings,  # 2048
)
```

256
3
2
16
“mish”
1024
3

```python
self.duration_predictor_dropout_probs = duration_predictor_dropout_probs # 0.1
self.num_duration_conv_layers = num_duration_conv_layers                 # 2
self.duration_predictor_filters = duration_predictor_filters             # 256
self.duration_predictor_kernel_sizes = duration_predictor_kernel_sizes   # 3
self.num_mels = num_mels                                                 # 80
```

```python
# postnet
self.n_conv_postnet = n_conv_postnet            # 5
self.postnet_conv_filters = postnet_conv_filters # 512
self.postnet_conv_kernel_sizes = postnet_conv_kernel_sizes # 5
self.postnet_dropout_rate = postnet_dropout_rate # 0.1
```

envs/tts/lib/python3.8/site-packages/tensorflow_tts/processor/baker.py

标贝数据集数据预处理，特征提取

```python
SelfAttentionParams = collections.namedtuple(
    "SelfAttentionParams",
    [
        "n_speakers",
        "hidden_size",
        "num_hidden_layers",
        "num_attention_heads",
        "attention_head_size",
        "intermediate_size",
        "intermediate_kernel_size",
        "hidden_act",
        "output_attentions",
        "output_hidden_states",
        "initializer_range",
        "hidden_dropout_prob",
        "attention_probs_dropout_prob",
        "layer_norm_eps",
        "max_position_embeddings",
    ],
)
```

```python
from tensorflow_tts.configs import FastSpeechConfig

class FastSpeech2Config(FastSpeechConfig):
    """Initialize FastSpeech2 Config."""

    def __init__(
        self,
        variant_prediction_num_conv_layers=2,
        variant_kernel_size=9,
        variant_dropout_rate=0.5,
        variant_predictor_filter=256,
        variant_predictor_kernel_size=3,
        variant_predictor_dropout_rate=0.5,
        **kwargs
    ):
        super().__init__(**kwargs)
        self.variant_prediction_num_conv_layers = variant_prediction_num_conv_layers # 2
        self.variant_predictor_kernel_size = variant_predictor_kernel_size           # 3
        self.variant_predictor_dropout_rate = variant_predictor_dropout_rate         # 0.5
        self.variant_predictor_filter = variant_predictor_filter                     # 256
```

```
num_mels: 80
hidden_dropout_prob: 0.2
attention_probs_dropout_prob: 0.1
max_position_embeddings: 2048
initializer_range: 0.02
output_attentions: False
output_hidden_states: False
```

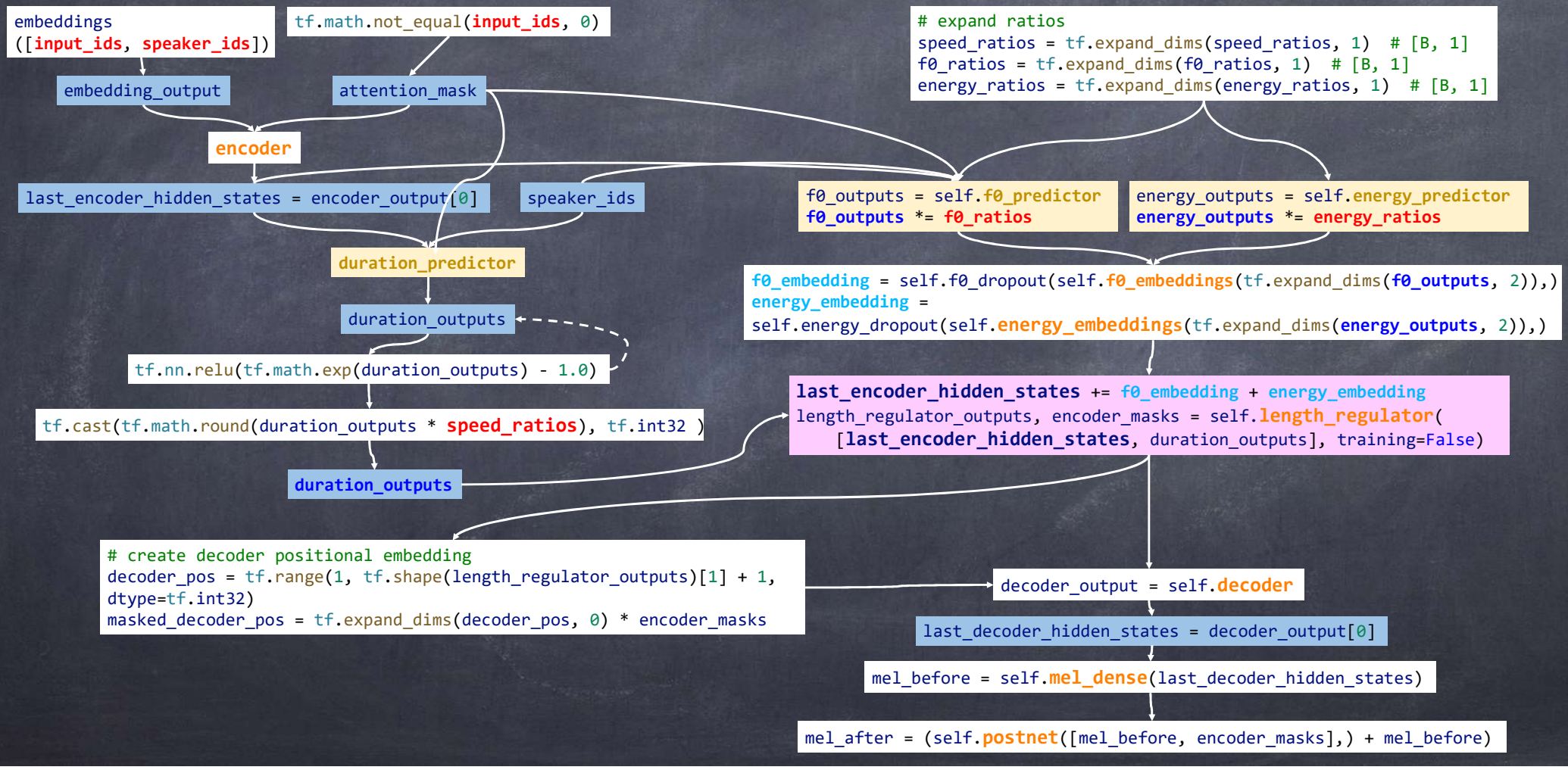
FastSpeech2 网络结构代码

```python
def call(self,  input_ids, speaker_ids, duration_gts,  f0_gts,  energy_gts,  training=False,  **kwargs, ):
    """Call logic."""
    attention_mask = tf.math.not_equal(input_ids, 0)
    embedding_output = self.embeddings([input_ids, speaker_ids], training=training)
    encoder_output = self.encoder([embedding_output, attention_mask], training=training)
    last_encoder_hidden_states = encoder_output[0]

    # energy predictor, here use last_encoder_hidden_states, u can use more hidden_states layers
    # rather than just use last_hidden_states of encoder for energy_predictor.
    duration_outputs = self.duration_predictor([last_encoder_hidden_states, speaker_ids, attention_mask])
    # [batch_size, length]

    f0_outputs = self.f0_predictor([last_encoder_hidden_states, speaker_ids, attention_mask], training=training)
    energy_outputs = self.energy_predictor([last_encoder_hidden_states, speaker_ids, attention_mask], training=training)

    f0_embedding = self.f0_embeddings(tf.expand_dims(f0_gts, 2))  # [barch_size, mel_length, feature]
    energy_embedding = self.energy_embeddings(tf.expand_dims(energy_gts, 2))  # [barch_size, mel_length, feature]

    # apply dropout both training/inference
    f0_embedding = self.f0_dropout(f0_embedding, training=True)
    energy_embedding = self.energy_dropout(energy_embedding, training=True)

    # sum features
    last_encoder_hidden_states += f0_embedding + energy_embedding

    length_regulator_outputs, encoder_masks = self.length_regulator(
        [last_encoder_hidden_states, duration_gts], training=training
    )

    # create decoder positional embedding
    decoder_pos = tf.range(1, tf.shape(length_regulator_outputs)[1] + 1, dtype=tf.int32)
    masked_decoder_pos = tf.expand_dims(decoder_pos, 0) * encoder_masks

    decoder_output = self.decoder([length_regulator_outputs, speaker_ids, encoder_masks, masked_decoder_pos],
        training=training, )
    last_decoder_hidden_states = decoder_output[0]

    # here u can use sum or concat more than 1 hidden states layers from decoder.
    mels_before = self.mel_dense(last_decoder_hidden_states)
    mels_after = (self.postnet([mels_before, encoder_masks], training=training) + mels_before)

    outputs = (mels_before, mels_after, duration_outputs, f0_outputs,  energy_outputs, )
```

```python
def _inference(self, input_ids, speaker_ids, speed_ratios, f0_ratios, energy_ratios, **kwargs,):
    """Call logic."""
    attention_mask = tf.math.not_equal(input_ids, 0)
    embedding_output = self.embeddings([input_ids, speaker_ids], training=False)
    encoder_output = self.encoder([embedding_output, attention_mask], training=False)
    last_encoder_hidden_states = encoder_output[0]

    # expand ratios
    speed_ratios = tf.expand_dims(speed_ratios, 1)  # [B, 1]
    f0_ratios = tf.expand_dims(f0_ratios, 1)  # [B, 1]
    energy_ratios = tf.expand_dims(energy_ratios, 1)  # [B, 1]

    # energy predictor, here use last_encoder_hidden_states, u can use more hidden_states layers
    # rather than just use last_hidden_states of encoder for energy_predictor.
    duration_outputs = self.duration_predictor([last_encoder_hidden_states, speaker_ids, attention_mask])
    # [batch_size, length]
    duration_outputs = tf.nn.relu(tf.math.exp(duration_outputs) - 1.0)
    duration_outputs = tf.cast(tf.math.round(duration_outputs * speed_ratios), tf.int32)

    f0_outputs = self.f0_predictor([last_encoder_hidden_states, speaker_ids, attention_mask], training=False)
    f0_outputs *= f0_ratios

    energy_outputs = self.energy_predictor([last_encoder_hidden_states, speaker_ids, attention_mask], training=False)
    energy_outputs *= energy_ratios

    f0_embedding = self.f0_dropout(self.f0_embeddings(tf.expand_dims(f0_outputs, 2)), training=True)
    energy_embedding = self.energy_dropout(self.energy_embeddings(tf.expand_dims(energy_outputs, 2)), training=True)

    # sum features
    last_encoder_hidden_states += f0_embedding + energy_embedding

    length_regulator_outputs, encoder_masks = self.length_regulator(
        [last_encoder_hidden_states, duration_outputs], training=False
    )

    # create decoder positional embedding
    decoder_pos = tf.range(1, tf.shape(length_regulator_outputs)[1] + 1, dtype=tf.int32)
    masked_decoder_pos = tf.expand_dims(decoder_pos, 0) * encoder_masks

    decoder_output = self.decoder([length_regulator_outputs, speaker_ids, encoder_masks, masked_decoder_pos],
        training=False, )
    last_decoder_hidden_states = decoder_output[0]

    # here u can use sum or concat more than 1 hidden states layers from decoder.
    mel_before = self.mel_dense(last_decoder_hidden_states)
    mel_after = (self.postnet([mel_before, encoder_masks], training=False) + mel_before)

    outputs = (mel_before, mel_after, duration_outputs, f0_outputs, energy_outputs)
    return outputs
```

FastSpeech2 网络结构代码

推理流程



FastSpeech2

length_regulator 详解-原理

length_regulator

- 长度不匹配：phoneme & spectrogram sequence
- 控制 voice speed 和 韵律部分 part of prosody
- phoneme sequence 长度通常小于对应的 mel-spectrogram sequence
- 每个 phoneme 对应 多个 mel-spectrograms
- mel-spectrograms 对应的 那个 phoneme 叫作 phoneme duration
- 基于 phoneme duration d, length_regulator 扩展 phoneme sequence 隐藏状态 d 次
- 隐藏状态的总长度 就 等于 mel-spectrograms 的长度

- α 是超参数, 决定了 *Hmel* 的长度
- phoneme sequence 隐藏状态, n 是序列长度 $\mathcal{H}_{pho} = [h1, h2, \dots, hn]$
- phoneme duration sequence: $\mathcal{D}$ $\mathcal{D} = [d1, d2, \dots, dn]$
- m 是 mel_spectrogram 序列长度 $\sum_{n=1}^{n} d_i = m$

$$\mathcal{H}_{mel} = \mathcal{LR}(\mathcal{H}_{pho}, \mathcal{D}, \alpha)$$

- α = 1, α = 1.3, α = 0.5

$\mathcal{H}_{mel} = [h1, h1, h2, h2, h3, h3, h3, h4]$

$\mathcal{H}_{pho} = [h1, h2, \dots, hn]$

$\mathcal{D} = [2, 2, 3, 1]$

$\mathcal{D} = [2.6, 2.6, 3.9, 1.3]$ $\mathcal{D} = [3, 3, 4, 1]$

$\mathcal{D} = [1, 1, 1.5, 0.5]$ $\mathcal{D} = [1, 1, 2, 1]$

FastSpeech2

length_regulator 详解- TF2.x 代码

```
last_encoder_hidden_states += f0_embedding + energy_embedding
length_regulator_outputs, encoder_masks = self.length_regulator(
    [last_encoder_hidden_states, duration_outputs], training=False)
```

```
"""Length regulator logic."""
sum_durations = tf.reduce_sum(durations, axis=-1)  # [batch_size]
max_durations = tf.reduce_max(sum_durations)

input_shape = tf.shape(encoder_hidden_states)
batch_size = input_shape[0]
hidden_size = input_shape[-1]

# initialize output hidden states and encoder masking.
# There is only 1 batch in inference, so we don't have to use
# `tf.While` op with 3-D output tensor.
repeats = durations[0]
real_length = tf.reduce_sum(repeats)
pad_size = max_durations - real_length
# masks : [max_durations]
masks = tf.sequence_mask([real_length], max_durations, dtype=tf.int32)
repeat_encoder_hidden_states = tf.repeat(
    encoder_hidden_states[0], repeats=repeats, axis=0
)
repeat_encoder_hidden_states = tf.expand_dims(
    tf.pad(repeat_encoder_hidden_states, [[0, pad_size], [0, 0]]), 0
)  # [1, max_durations, hidden_size]

outputs = repeat_encoder_hidden_states
encoder_masks = masks
```

计算 tensor 沿着某一维度的和，可以在求和后降维
计算 tensor 指定轴方向上的各个元素的最大值

类似 numpy ndarray.shape → (m, n, x, ...)

返回 mask 张量, 1: lengths 输出结果的长度 数值或多维数组, 2: maxlen – None 为 length中最大数值, 设置了值N ， 最大长度就是 N; 3: 值代表 T/F

对特定元素进行重复, 1: tensor, 2: 数值或多维数组, repeats为一个整数时，所有元素均重复N次，对于一位数组[a,b]，input的第一个元素重复a次，第二个元素重复b次。

axis=1, 横向增加; 0, 列向增加; 不设置, 先拉平, 再增加

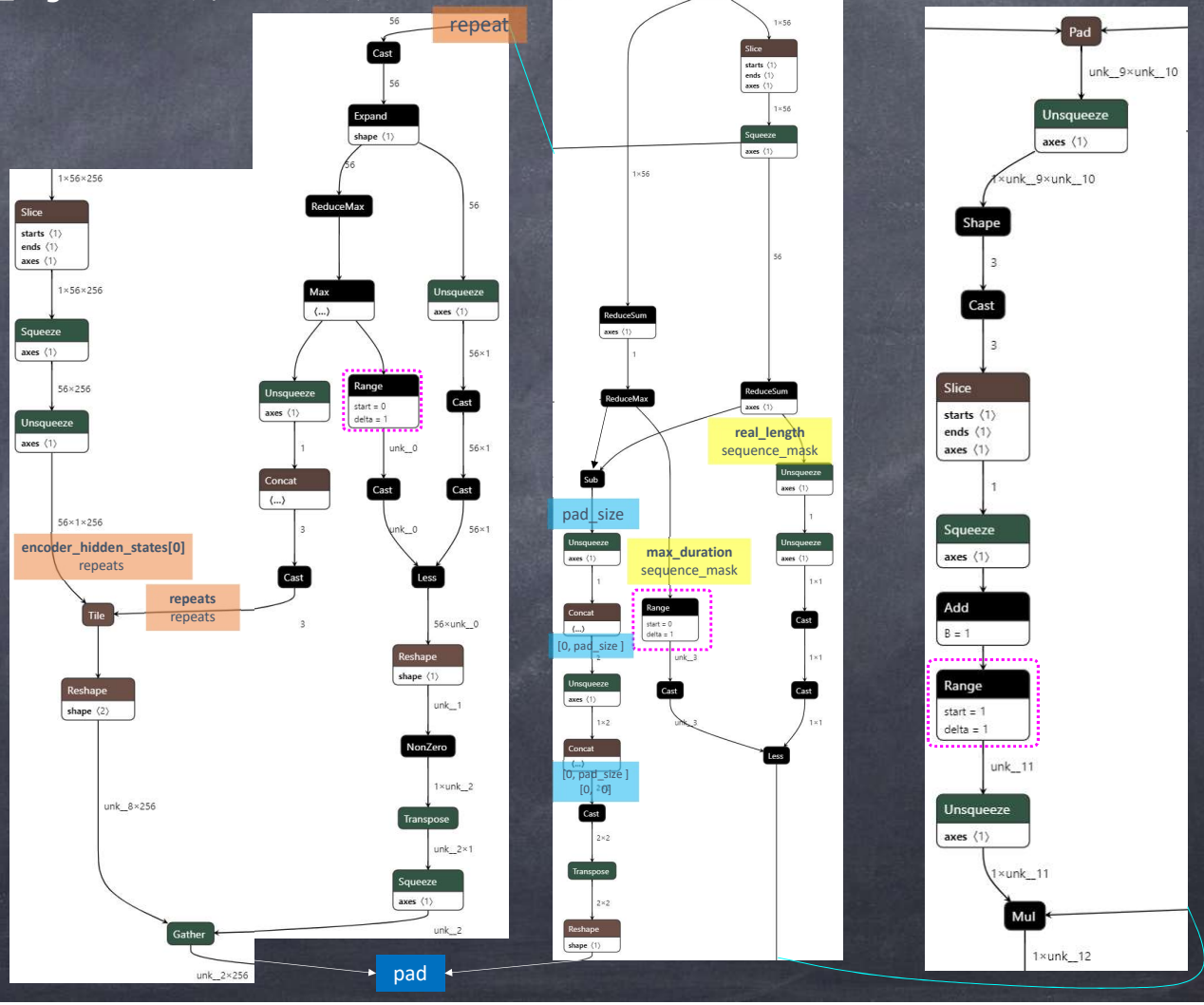
对张量在指定维度上增加一维，默认为 维度的值取1

对张量填充, 1: tensor, 2: (上下左右)每维填充的数量, 3: 默认填0

FastSpeech2

length_regulator 详解-代码对应关系

```
tf.reduce_sum(durations_gt, axis=-1)
real_length = tf.reduce_sum(repeats)
max_durations = tf.reduce_max(sum_durations)
repeats = durations_gt[0]
masks = tf.sequence_mask([real_length],
        max_durations, dtype=tf.int32)
repeat_encoder_hidden_states = tf.repeat(
  encoder_hidden_states[0], repeats=repeats, axis=0 )
pad_size = max_durations - real_length
repeat_encoder_hidden_states = tf.expand_dims(
  tf.pad(repeat_encoder_hidden_states, [[0, pad_size], [0, 0]]), 0 )
```



onnx

tile: output = tile(input, repeats) # repeats [a,b] 行扩a遍；列扩b遍

reshape 类似numpy reshape, 形状 -1 时, 表示不确定

pad: output = pad(input, pads, constant_value, axes)

pads -- [x1_begin, x2_begin,...,x1_end, x2_end,...]

FastSpeech2：数据预处理

envs/tts/lib/python3.8/site-packages/tensorflow_tts/processor/baker.py

标贝数据集数据预处理，特征提取

```
processor = AutoProcessor.from_pretrained(pretrained_path = "processor.json") # BakerProcessor
text = "这是一个开源的端到端中文语音合成系统"
input_ids = processor.text_to_sequence(text, inference=True)
```

```
def text_to_sequence(self, text, inference=False):
    if inference:
        pinyin = self.pinyin_parser(text, style=Style.TONE3, errors="ignore")
        new_pinyin = []
        for x in pinyin:
            x = "".join(x)
            if "#" not in x:
                new_pinyin.append(x)
        phonemes = self.get_phoneme_from_char_and_pinyin(text, new_pinyin)
        text = " ".join(phonemes)
        print(f"phoneme seq: {text}")

    sequence = []
    for symbol in text.split():
        idx = self.symbol_to_id[symbol]
        sequence.append(idx)

    # add eos tokens
    sequence += [self.eos_id]
    return sequence
```

文本转拼音

拼音转音素

音素 转 id