Voice Convsion

許博竣

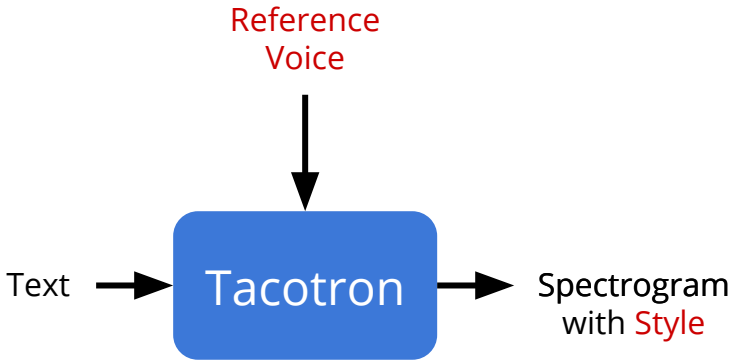
Outline

- Style Control of Voice
- GST-Tacotron
- Voice Cloning
- Voice Conversion
- Model Architecture
- Autoencoder
- Stage 1: Autoencoder With A Speaker Classifier
- GAN
- Stage 2: GAN
- Coding

Style Control of Voice

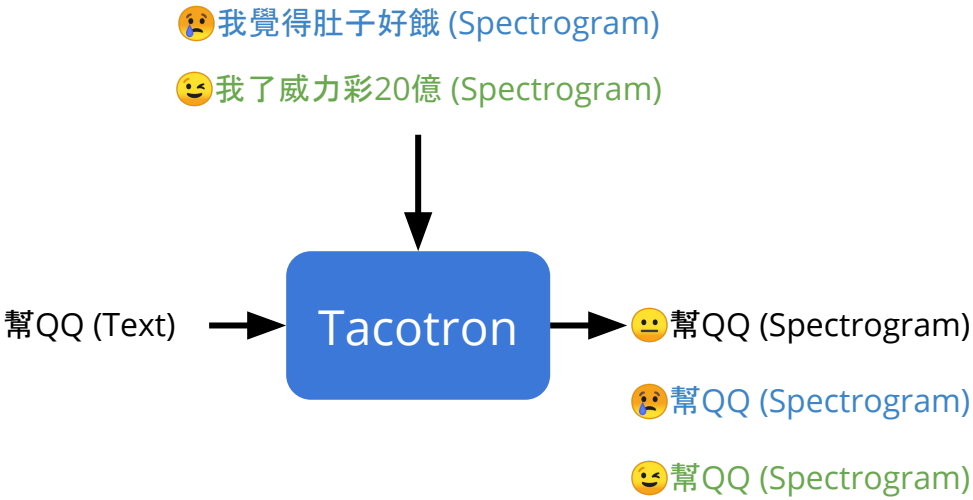
- GST-Tacotron
- Voice Cloning
- Voice Conversion

GST-Tacotron



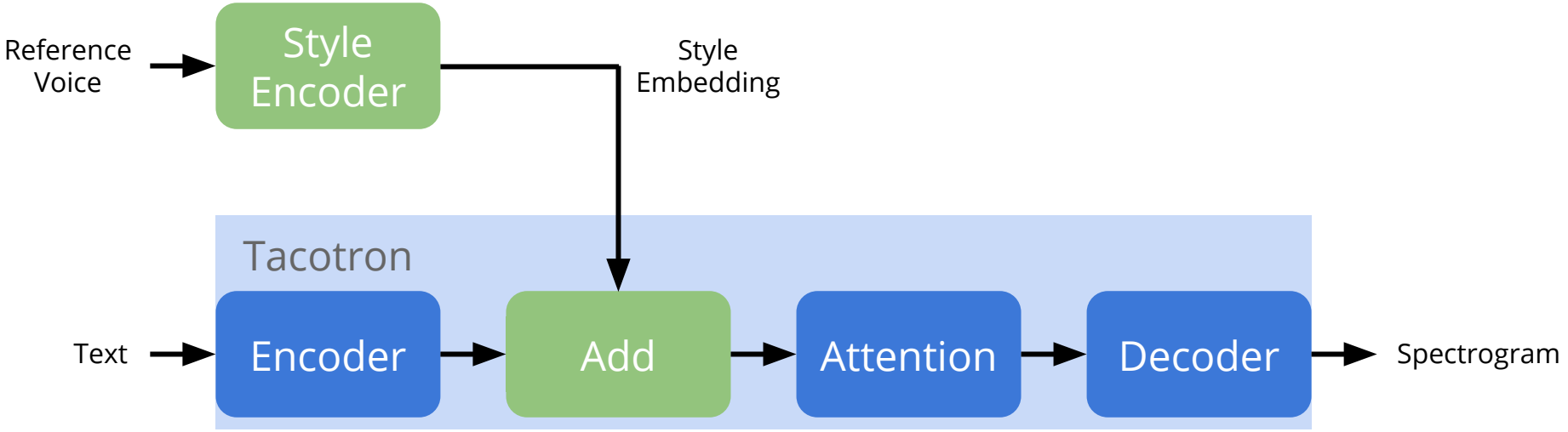
Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis

GST-Tacotron



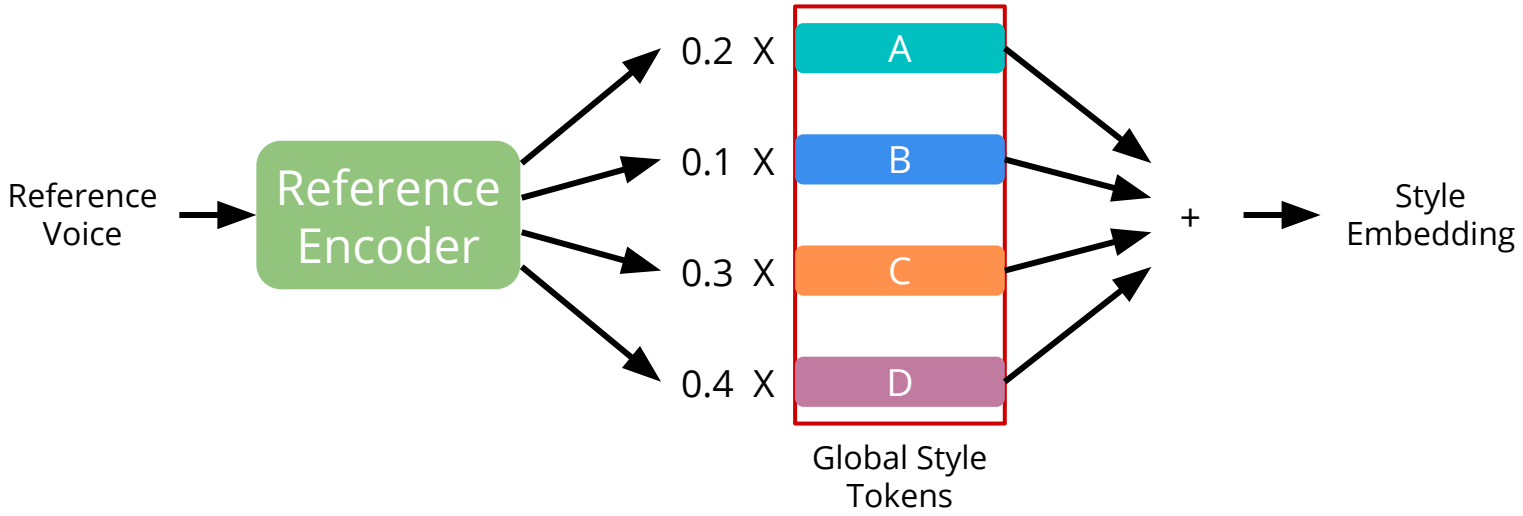
Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis

GST-Tacotron



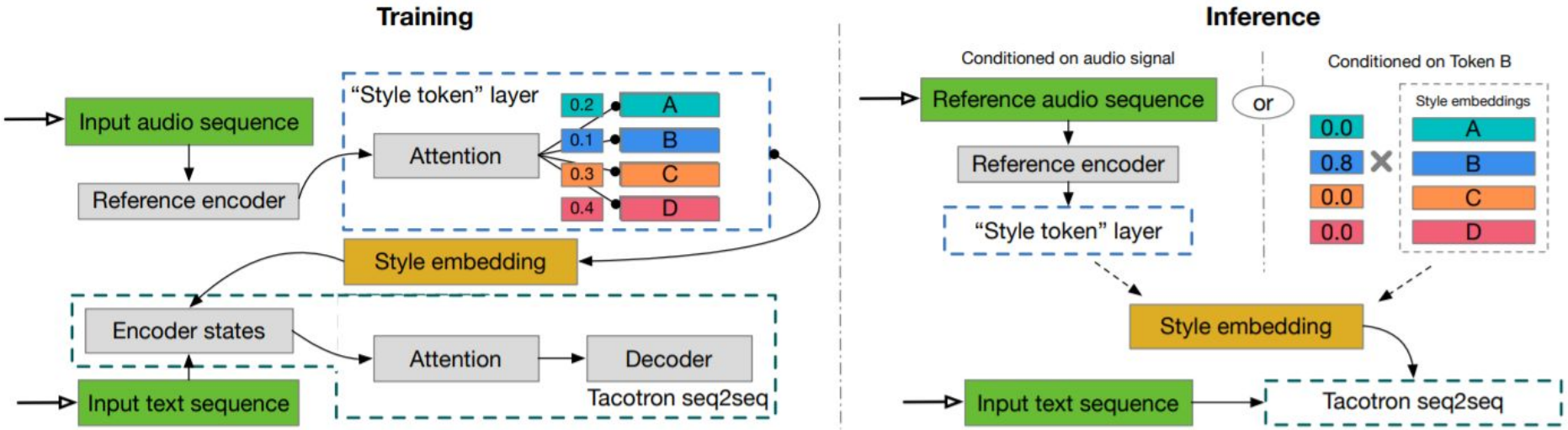
Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis

GST-Tacotron



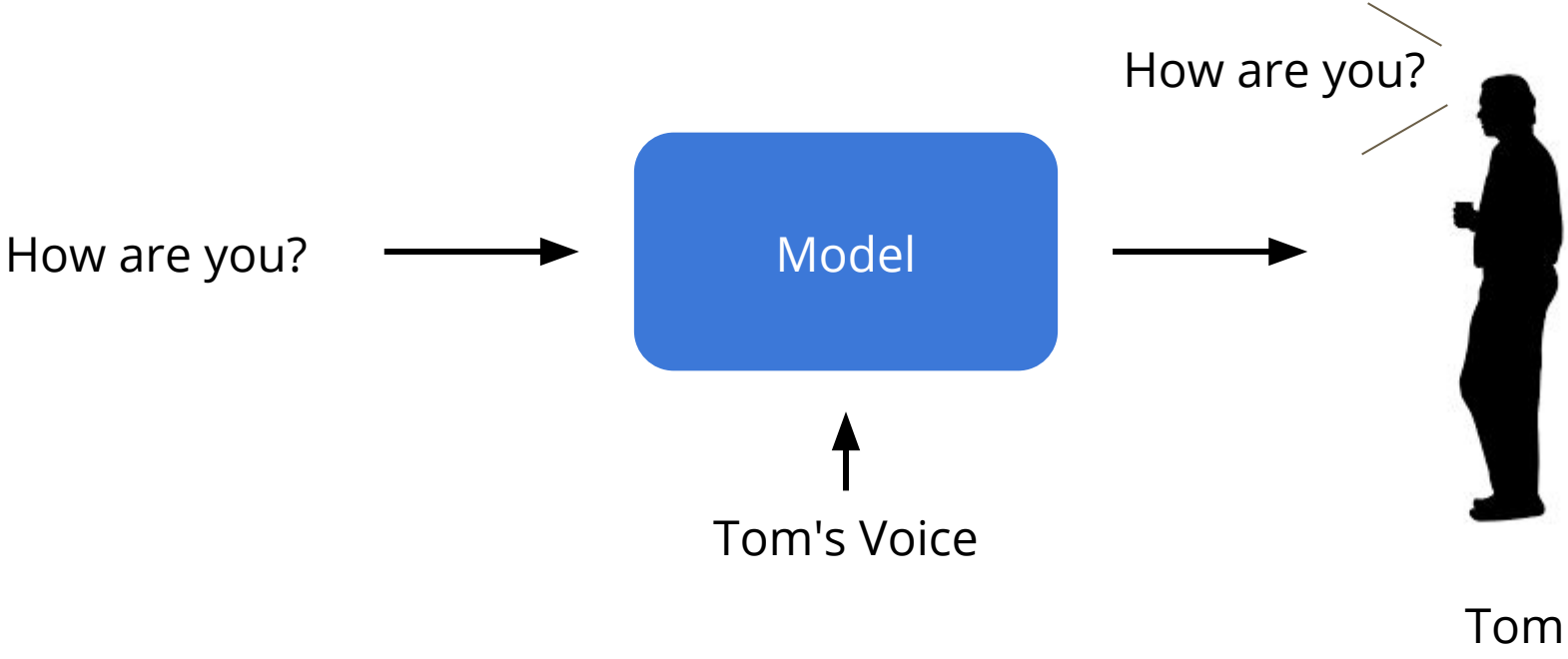
Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis

GST-Tacotron

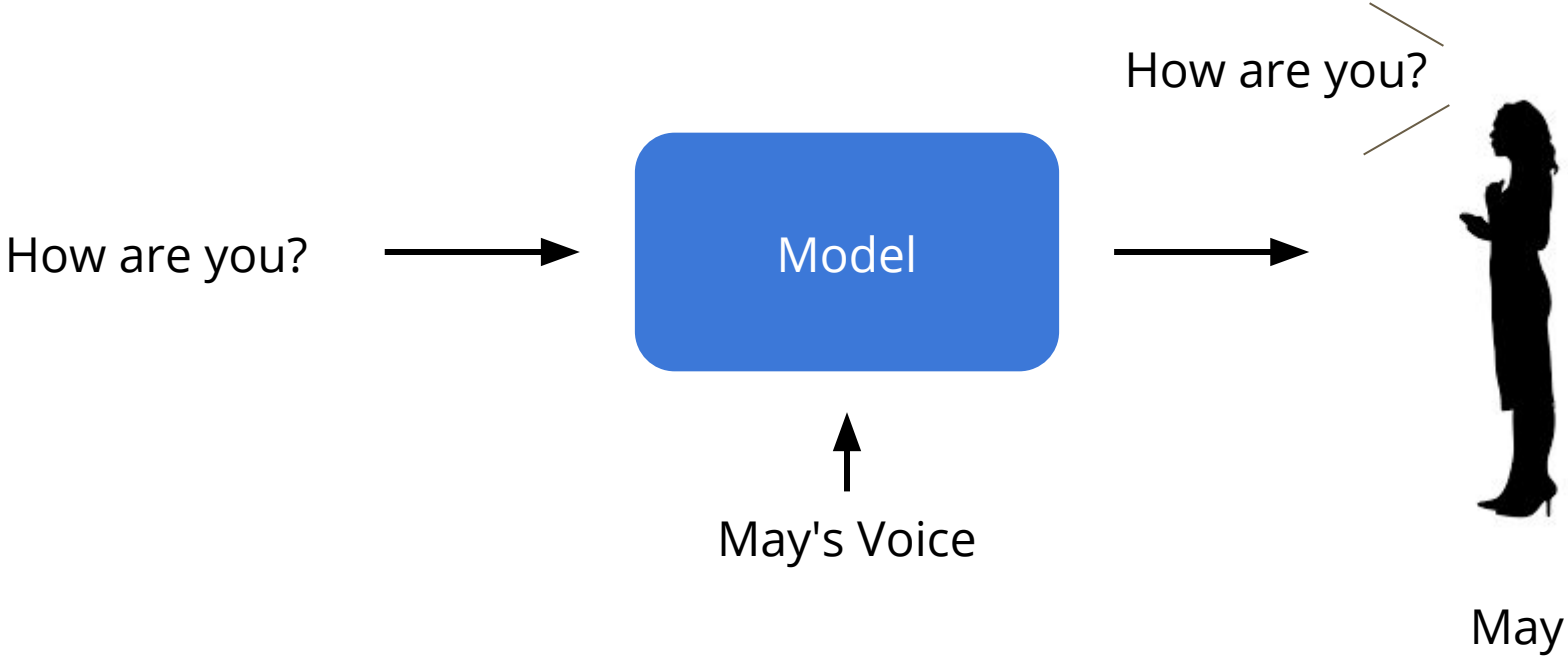


Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis

Voice Cloning

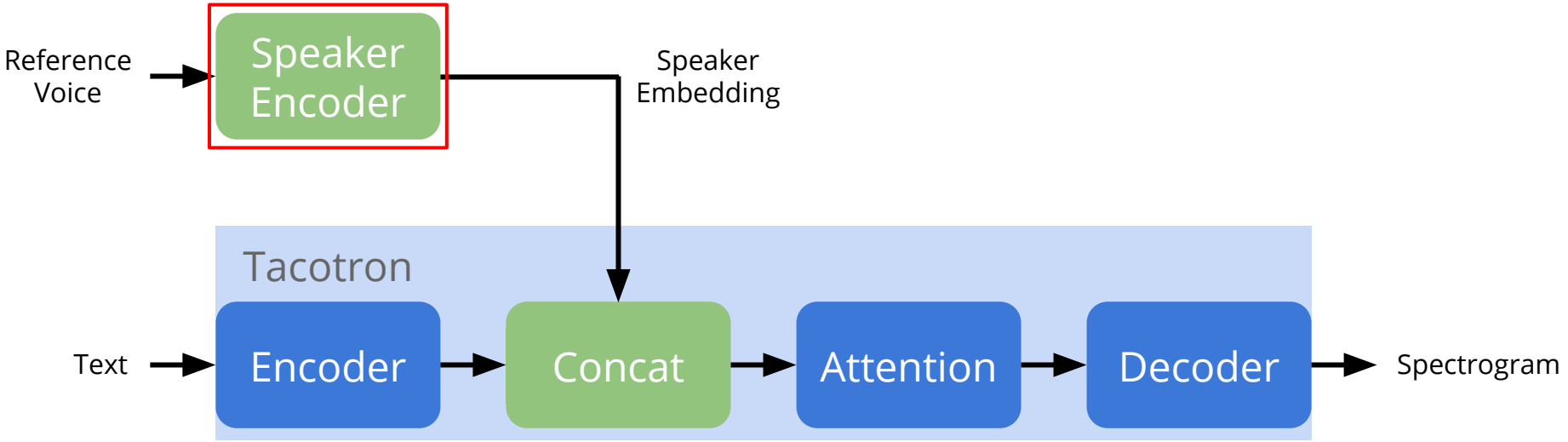


Voice Cloning



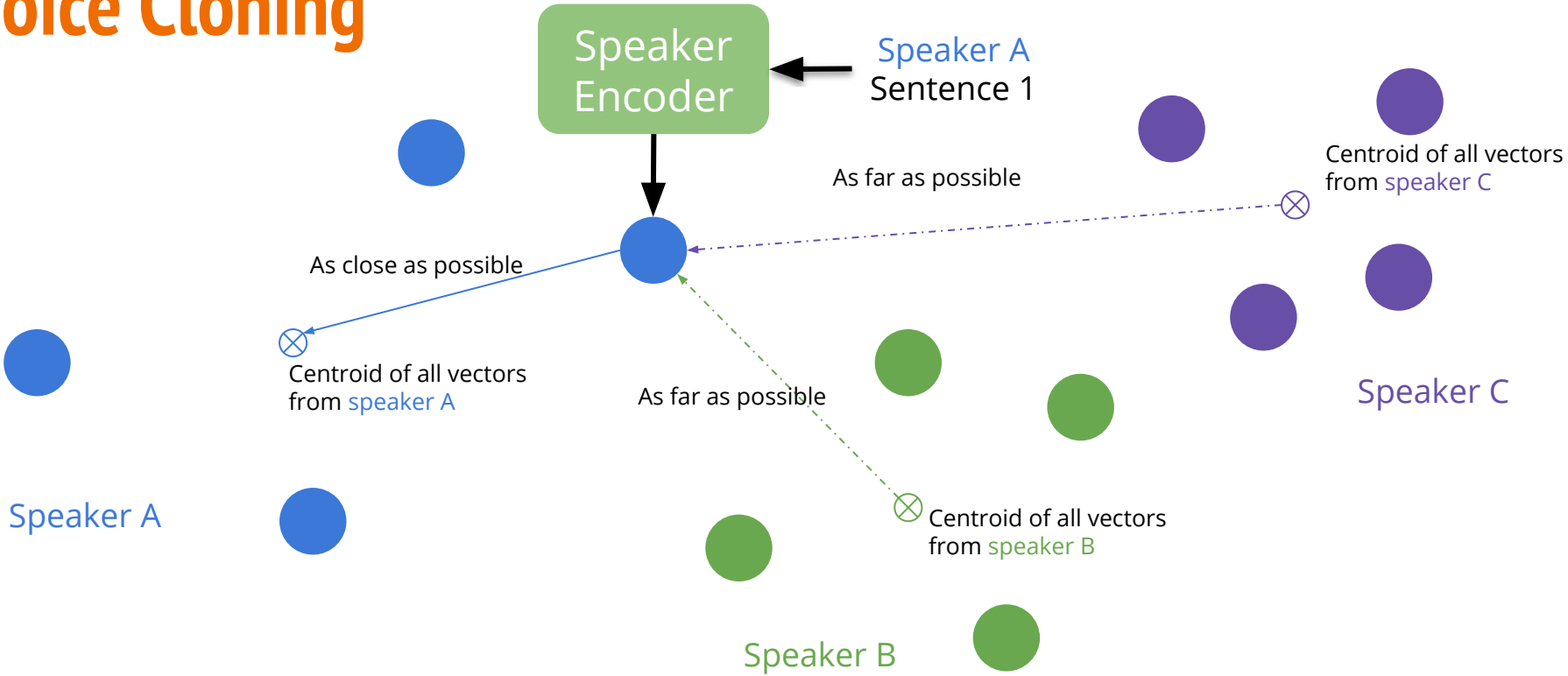
Voice Cloning

Based on GST-Tacotron, using Generalized End-to-End(GE2E) loss to train the speaker encoder.



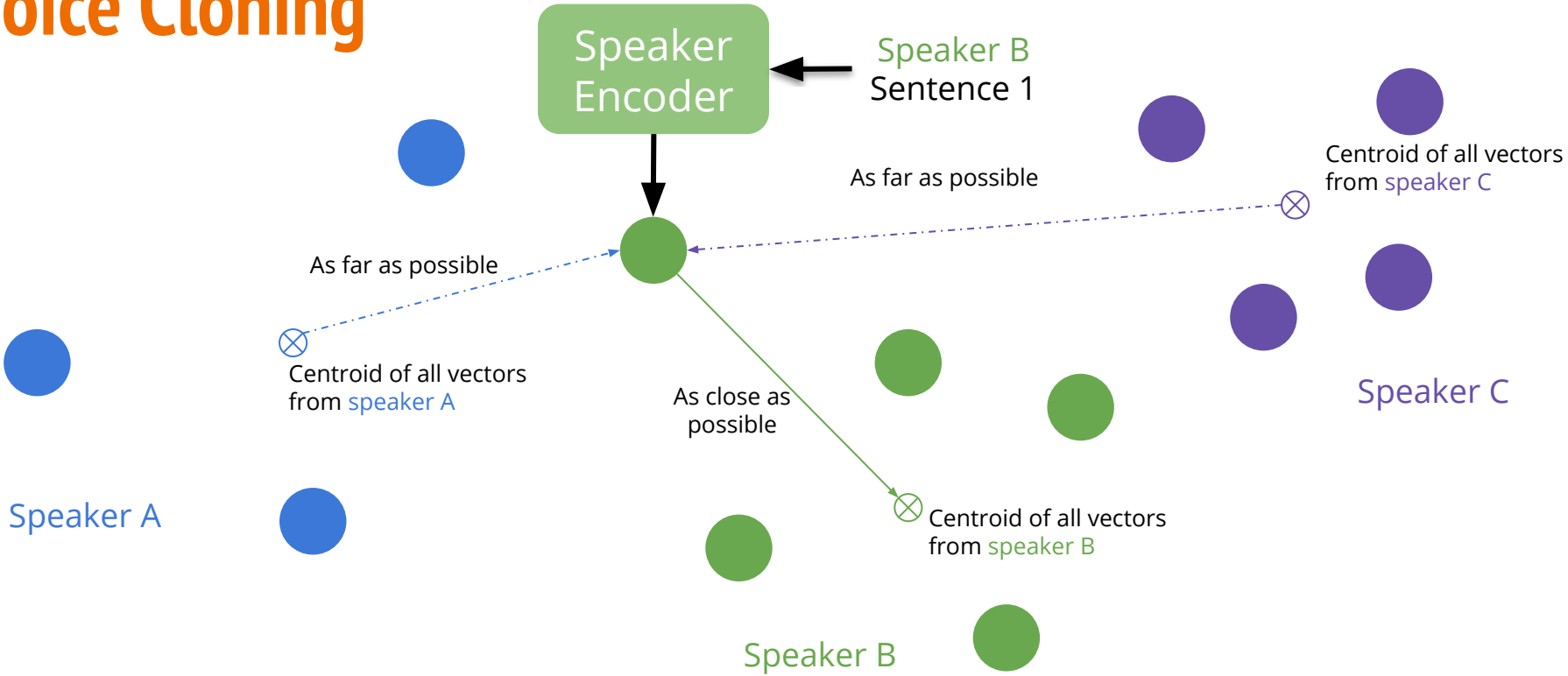
Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis

Voice Cloning



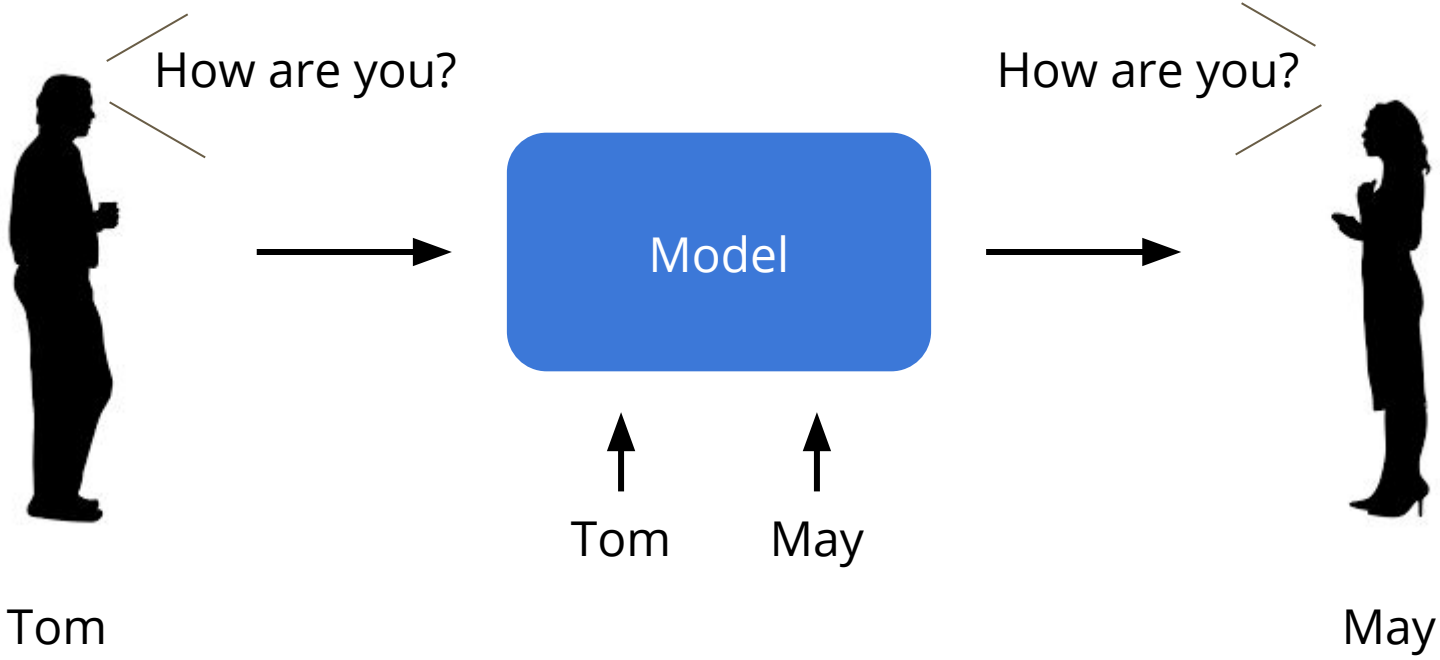
Generalized End-to-End Loss for Speaker Verification

Voice Cloning



Generalized End-to-End Loss for Speaker Verification

Voice Conversion



Voice Conversion

Voice Conversion - Past

Tom

How are you?

She sells seashells by the seashore.

He threw three free throws.

.
.
.

May

How are you?

She sells seashells by the seashore.

He threw three free throws.

.
.
.

Voice Conversion - Past

Tom	May	John
How are you?	How are you?	How are you?
She sells seashells by the seashore.	She sells seashells by the seashore.	She sells seashells by the seashore.
He threw three free throws.	He threw three free throws.	He threw three free throws.
.	.	.
.	.	.
.	.	.

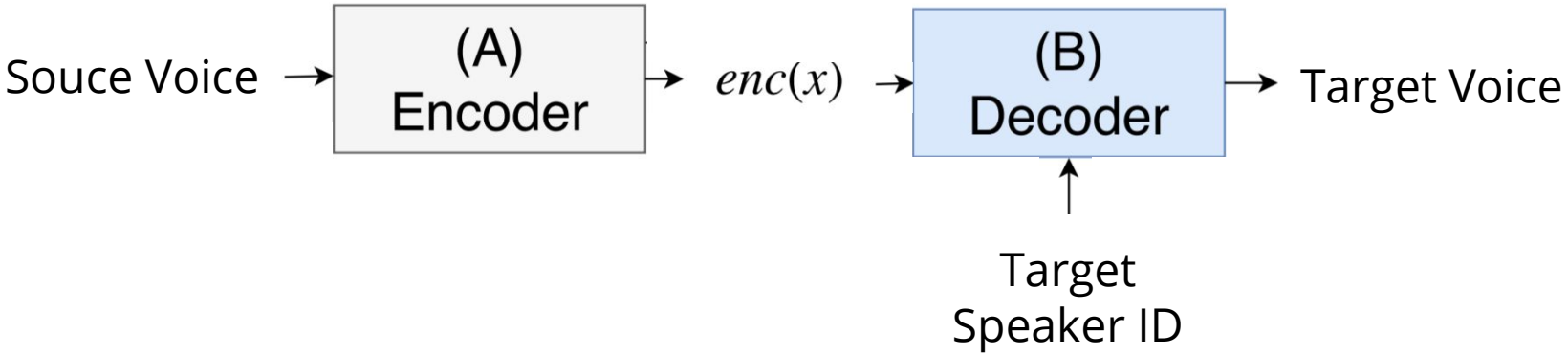
Voice Conversion - Now

Tom	May
How are you?	我覺得可以。
She sells seashells by the seashore.	我覺得不行。
He threw three free throws.	阿岳真的很嚴格。
.	.
.	.
.	.

Voice Conversion - Now

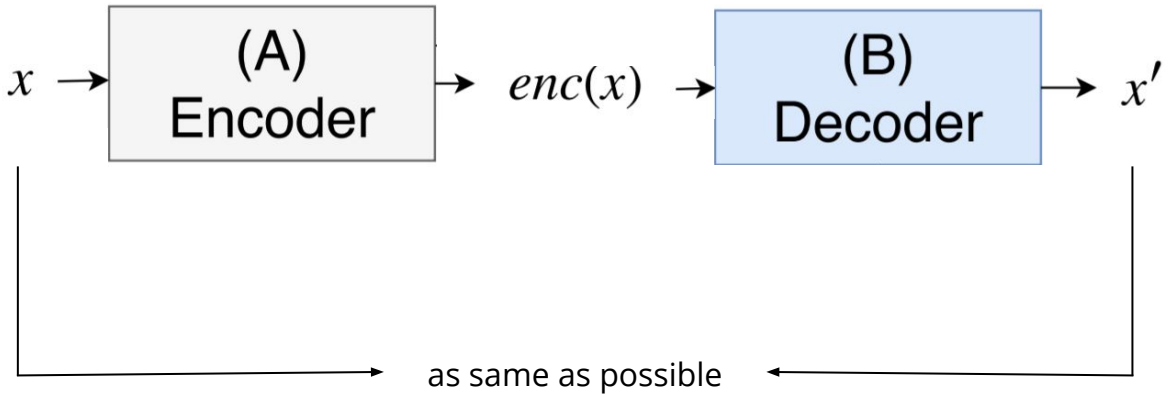
Tom	May	John
How are you?	我覺得可以。	Just do it.
She sells seashells by the seashore.	我覺得不行。	The Ultimate Driving machine.
He threw three free throws.	阿岳真的很嚴格。	I'm lovin' it.
.	.	.
.	.	.
.	.	.

Model Architecture

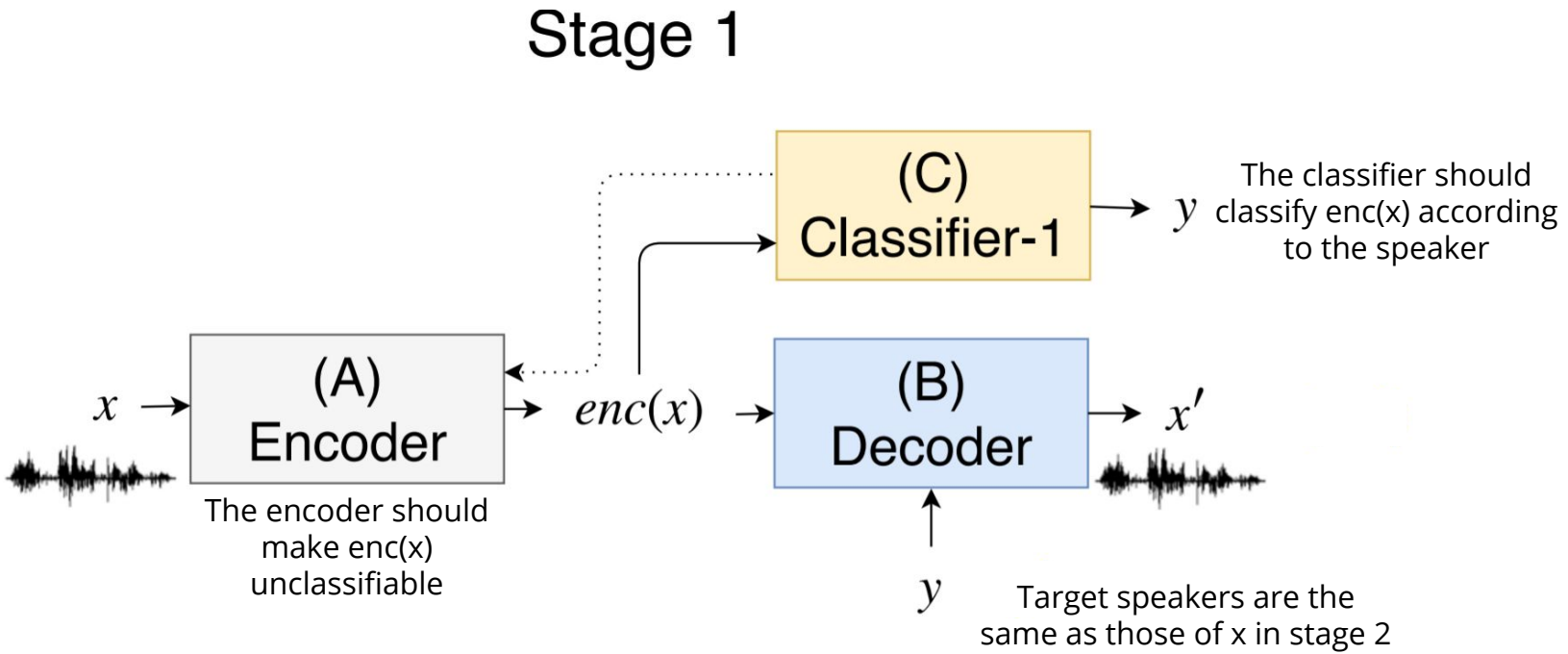


Multi-target Voice Conversion without Parallel Data by Adversarially Learning Disentangled Audio Representations

Autoencoder

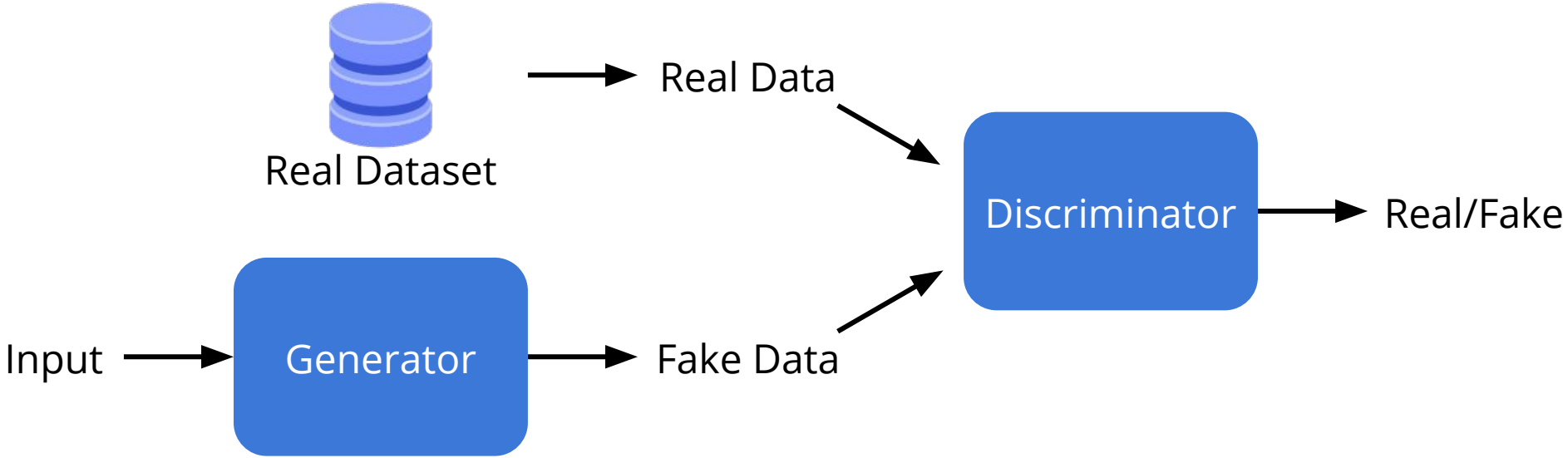


Stage 1: Autoencoder With A Speaker Classifier

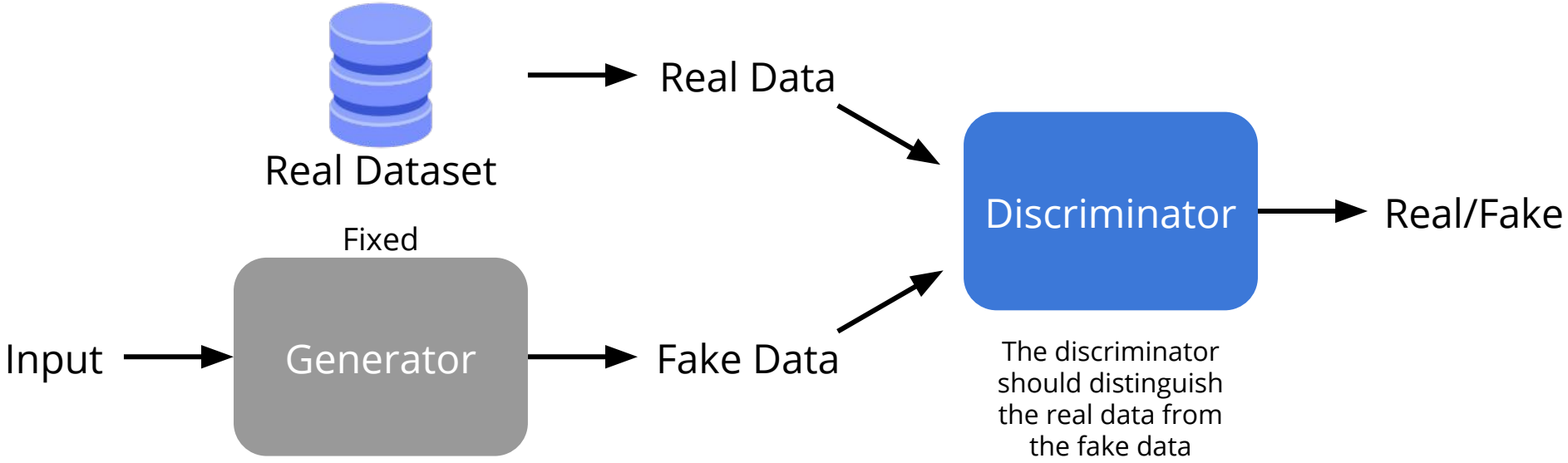


Multi-target Voice Conversion without Parallel Data by Adversarially Learning Disentangled Audio Representations

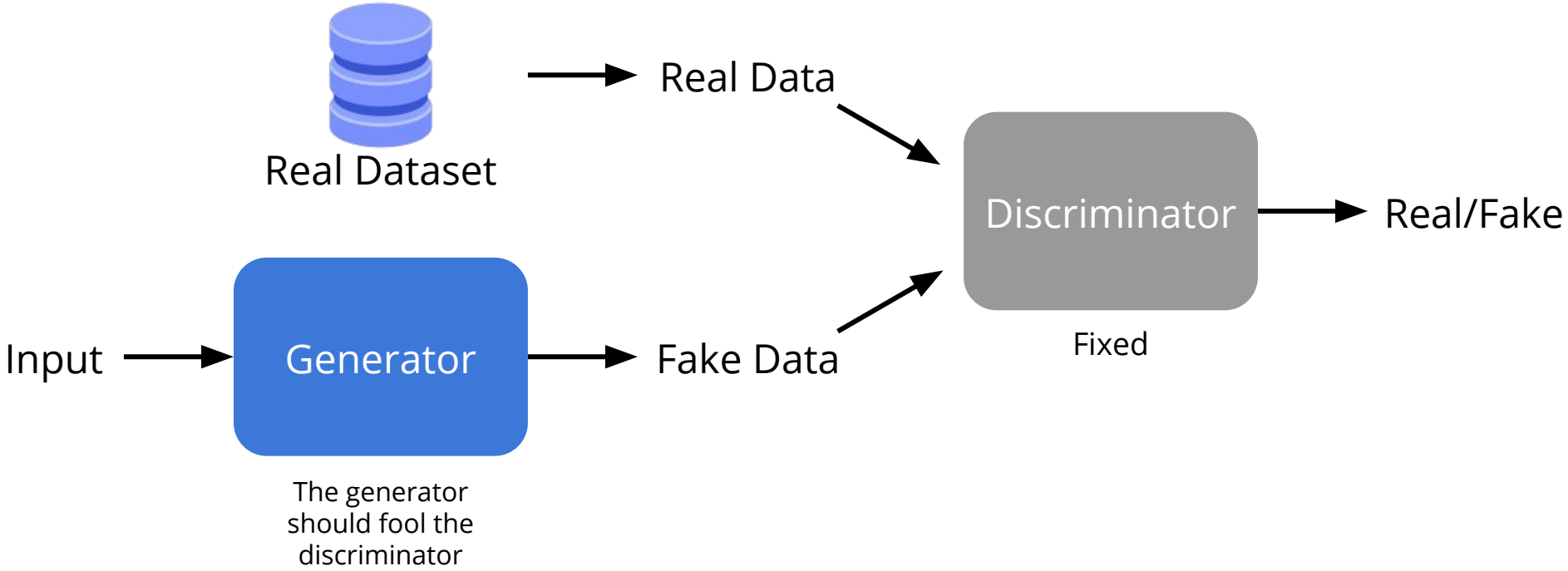
GAN


Real Dataset
Real Data
Discriminator
Real/Fake
Input
Generator
Fake Data

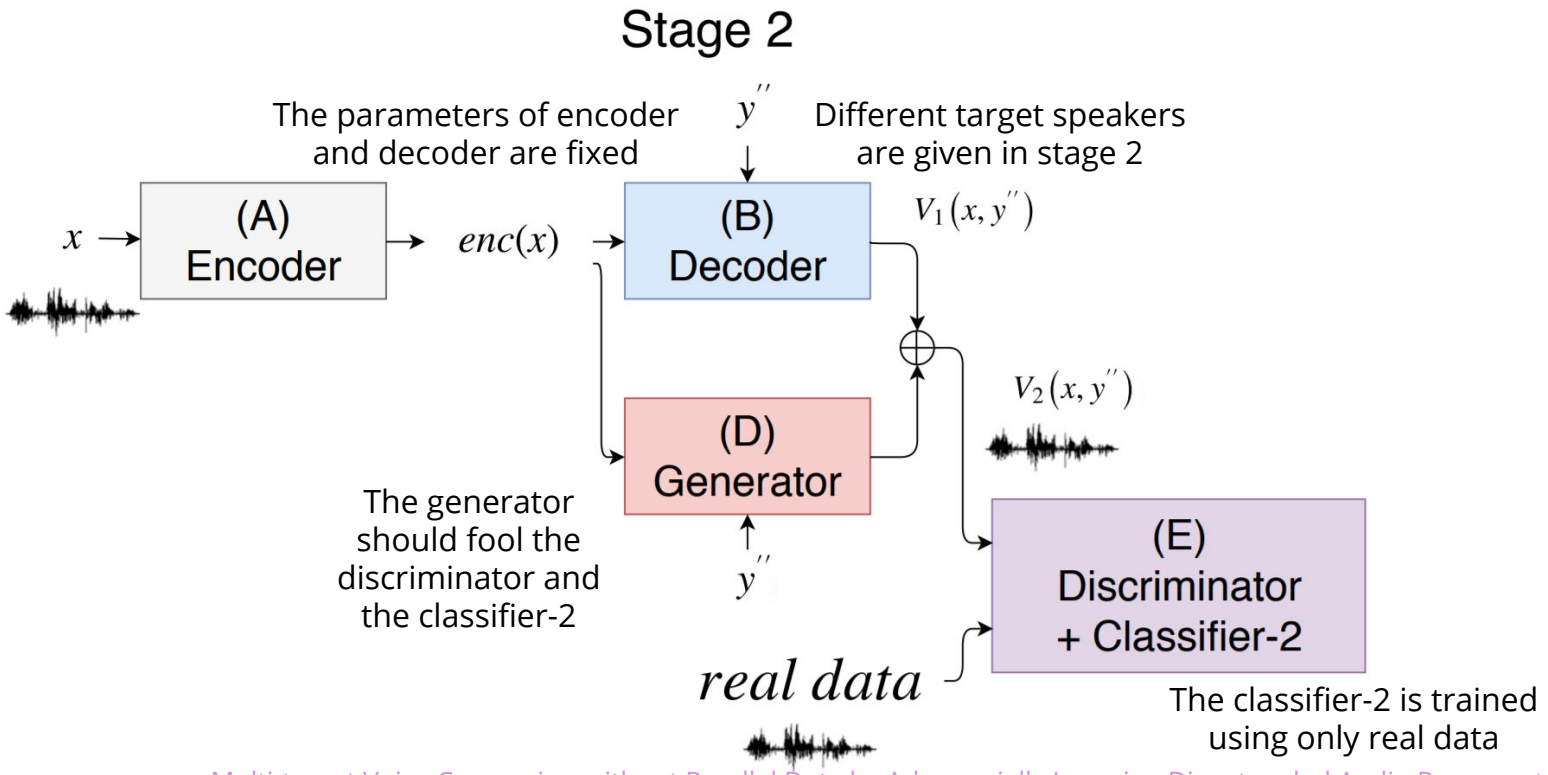
GAN - Train Discriminator



GAN - Train Generator



Stage 2: GAN



Multi-target Voice Conversion without Parallel Data by Adversarially Learning Disentangled Audio Representations

Coding

GitHub

solver.py

```
28  class Solver(object):
29      def __init__(self, hps, data_loader, log_dir='./log/'):
30          self.hps = hps
31          self.data_loader = data_loader
32          self.model_kept = []
33          self.max_keep = 100
34          self.build_model()
35          self.logger = Logger(log_dir) if log_dir != None else None
36
37      def build_model(self):
38          hps = self.hps
39          ns = self.hps.ns
40          emb_size = self.hps.emb_size
41          self.Encoder = cc(Encoder(ns=ns, dp=hps.enc_dp))
42          self.Decoder = cc(Decoder(ns=ns, c_a=hps.n_speakers, emb_size=emb_size))
43          self.Generator = cc(Decoder(ns=ns, c_a=hps.n_speakers, emb_size=emb_size))
44          self.SpeakerClassifier = cc(SpeakerClassifier(ns=ns, n_class=hps.n_speakers, dp=hps.dis_dp))
45          self.PatchDiscriminator = cc(nn.DataParallel(PatchDiscriminator(ns=ns, n_class=hps.n_speakers)))
46          betas = (0.5, 0.9)
47          params = list(self.Encoder.parameters()) + list(self.Decoder.parameters())
48          self.ae_opt = optim.Adam(params, lr=self.hps.lr, betas=betas)
49          self.clf_opt = optim.Adam(self.SpeakerClassifier.parameters(), lr=self.hps.lr, betas=betas)
50          self.gen_opt = optim.Adam(self.Generator.parameters(), lr=self.hps.lr, betas=betas)
51          self.patch_opt = optim.Adam(self.PatchDiscriminator.parameters(), lr=self.hps.lr, betas=betas)
```

solver.py

```
53      def save_model(self, model_path, iteration, enc_only=True):
54          if not enc_only:
55              all_model = {
56                  'encoder': self.Encoder.state_dict(),
57                  'decoder': self.Decoder.state_dict(),
58                  'generator': self.Generator.state_dict(),
59                  'classifier': self.SpeakerClassifier.state_dict(),
60                  'patch_discriminator': self.PatchDiscriminator.state_dict(),
61              }
62          else:
63              all_model = {
64                  'encoder': self.Encoder.state_dict(),
65                  'decoder': self.Decoder.state_dict(),
66                  'generator': self.Generator.state_dict(),
67              }
68          new_model_path = '{}-{}'.format(model_path, iteration)
69          with open(new_model_path, 'wb') as f_out:
70              torch.save(all_model, f_out)
71          self.model_kept.append(new_model_path)
72
73          if len(self.model_kept) >= self.max_keep:
74              os.remove(self.model_kept[0])
75              self.model_kept.pop(0)
```

solver.py

```
77      def load_model(self, model_path, enc_only=True):
78          print('load model from {}'.format(model_path))
79          with open(model_path, 'rb') as f_in:
80              all_model = torch.load(f_in)
81              self.Encoder.load_state_dict(all_model['encoder'])
82              self.Decoder.load_state_dict(all_model['decoder'])
83              self.Generator.load_state_dict(all_model['generator'])
84              if not enc_only:
85                  self.SpeakerClassifier.load_state_dict(all_model['classifier'])
86                  self.PatchDiscriminator.load_state_dict(all_model['patch_discriminator'])
```

solver.py

```
117     def encode_step(self, x):
118         enc = self.Encoder(x)
119         return enc
120
121     def decode_step(self, enc, c):
122         x_tilde = self.Decoder(enc, c)
123         return x_tilde
124
125     def patch_step(self, x, x_tilde, is_dis=True):
126         D_real, real_logits = self.PatchDiscriminator(x, classify=True)
127         D_fake, fake_logits = self.PatchDiscriminator(x_tilde, classify=True)
128         if is_dis:
129             w_dis = torch.mean(D_real - D_fake)
130             gp = calculate_gradients_penalty(self.PatchDiscriminator, x, x_tilde)
131             return w_dis, real_logits, gp
132         else:
133             return -torch.mean(D_fake), fake_logits
134
135     def gen_step(self, enc, c):
136         x_gen = self.Decoder(enc, c) + self.Generator(enc, c)
137         return x_gen
138
139     def clf_step(self, enc):
140         logits = self.SpeakerClassifier(enc)
141         return logits
```

All training processes are written in train(self, model_path, flag='train', mode='train')

model.py

```
class Encoder(nn.Module):
    def __init__(self, c_in=513, c_h1=128, c_h2=512, c_h3=128, ns=0.2, dp=0.5):
        super(Encoder, self).__init__()
        self.ns = ns
        self.conv1s = nn.ModuleList(
                [nn.Conv1d(c_in, c_h1, kernel_size=k) for k in range(1, 8)]
            )
        self.conv2 = nn.Conv1d(len(self.conv1s)*c_h1 + c_in, c_h2, kernel_size=1)
        self.conv3 = nn.Conv1d(c_h2, c_h2, kernel_size=5)
        self.conv4 = nn.Conv1d(c_h2, c_h2, kernel_size=5, stride=2)
        self.conv5 = nn.Conv1d(c_h2, c_h2, kernel_size=5)
        self.conv6 = nn.Conv1d(c_h2, c_h2, kernel_size=5, stride=2)
        self.conv7 = nn.Conv1d(c_h2, c_h2, kernel_size=5)
        self.conv8 = nn.Conv1d(c_h2, c_h2, kernel_size=5, stride=2)
        self.dense1 = nn.Linear(c_h2, c_h2)
        self.dense2 = nn.Linear(c_h2, c_h2)
        self.dense3 = nn.Linear(c_h2, c_h2)
        self.dense4 = nn.Linear(c_h2, c_h2)
        self.RNN = nn.GRU(input_size=c_h2, hidden_size=c_h3, num_layers=1, bidirectional=True)
        self.linear = nn.Linear(c_h2 + 2*c_h3, c_h2)
        # normalization layer
        self.ins_norm1 = nn.InstanceNorm1d(c_h2)
        self.ins_norm2 = nn.InstanceNorm1d(c_h2)
        self.ins_norm3 = nn.InstanceNorm1d(c_h2)
        self.ins_norm4 = nn.InstanceNorm1d(c_h2)
        self.ins_norm5 = nn.InstanceNorm1d(c_h2)
        self.ins_norm6 = nn.InstanceNorm1d(c_h2)
        # dropout layer
        self.drop1 = nn.Dropout(p=dp)
        self.drop2 = nn.Dropout(p=dp)
        self.drop3 = nn.Dropout(p=dp)
        self.drop4 = nn.Dropout(p=dp)
        self.drop5 = nn.Dropout(p=dp)
        self.drop6 = nn.Dropout(p=dp)
```

model.py

```
328 class Decoder(nn.Module):
329     def __init__(self, c_in=512, c_out=513, c_h=512, c_a=8, emb_size=128, ns=0.2):
330         super(Decoder, self).__init__()
331         self.ns = ns
332         self.conv1 = nn.Conv1d(c_in, 2*c_h, kernel_size=3)
333         self.conv2 = nn.Conv1d(c_h, c_h, kernel_size=3)
334         self.conv3 = nn.Conv1d(c_h, 2*c_h, kernel_size=3)
335         self.conv4 = nn.Conv1d(c_h, c_h, kernel_size=3)
336         self.conv5 = nn.Conv1d(c_h, 2*c_h, kernel_size=3)
337         self.conv6 = nn.Conv1d(c_h, c_h, kernel_size=3)
338         self.dense1 = nn.Linear(c_h, c_h)
339         self.dense2 = nn.Linear(c_h, c_h)
340         self.dense3 = nn.Linear(c_h, c_h)
341         self.dense4 = nn.Linear(c_h, c_h)
342         self.RNN = nn.GRU(input_size=c_h, hidden_size=c_h//2, num_layers=1, bidirectional=True)
343         self.dense5 = nn.Linear(2*c_h + c_h, c_h)
344         self.linear = nn.Linear(c_h, c_out)
345         # normalization layer
346         self.ins_norm1 = nn.InstanceNorm1d(c_h)
347         self.ins_norm2 = nn.InstanceNorm1d(c_h)
348         self.ins_norm3 = nn.InstanceNorm1d(c_h)
349         self.ins_norm4 = nn.InstanceNorm1d(c_h)
350         self.ins_norm5 = nn.InstanceNorm1d(c_h)
351         # embedding layer
352         self.emb1 = nn.Embedding(c_a, c_h)
353         self.emb2 = nn.Embedding(c_a, c_h)
354         self.emb3 = nn.Embedding(c_a, c_h)
355         self.emb4 = nn.Embedding(c_a, c_h)
356         self.emb5 = nn.Embedding(c_a, c_h)
```

model.py

```
198 class SpeakerClassifier(nn.Module):
199     def __init__(self, c_in=512, c_h=512, n_class=8, dp=0.1, ns=0.01):
200         super(SpeakerClassifier, self).__init__()
201         self.dp, self.ns = dp, ns
202         self.conv1 = nn.Conv1d(c_in, c_h, kernel_size=5)
203         self.conv2 = nn.Conv1d(c_h, c_h, kernel_size=5)
204         self.conv3 = nn.Conv1d(c_h, c_h, kernel_size=5)
205         self.conv4 = nn.Conv1d(c_h, c_h, kernel_size=5)
206         self.conv5 = nn.Conv1d(c_h, c_h, kernel_size=5)
207         self.conv6 = nn.Conv1d(c_h, c_h, kernel_size=5)
208         self.conv7 = nn.Conv1d(c_h, c_h//2, kernel_size=3)
209         self.conv8 = nn.Conv1d(c_h//2, c_h//4, kernel_size=3)
210         self.conv9 = nn.Conv1d(c_h//4, n_class, kernel_size=16)
211         self.drop1 = nn.Dropout(p=dp)
212         self.drop2 = nn.Dropout(p=dp)
213         self.drop3 = nn.Dropout(p=dp)
214         self.drop4 = nn.Dropout(p=dp)
215         self.ins_norm1 = nn.InstanceNorm1d(c_h)
216         self.ins_norm2 = nn.InstanceNorm1d(c_h)
217         self.ins_norm3 = nn.InstanceNorm1d(c_h)
218         self.ins_norm4 = nn.InstanceNorm1d(c_h//4)
```

model.py

```
113  class PatchDiscriminator(nn.Module):
114      def __init__(self, n_class=33, ns=0.2, dp=0.1):
115          super(PatchDiscriminator, self).__init__()
116          self.ns = ns
117          self.conv1 = nn.Conv2d(1, 64, kernel_size=5, stride=2)
118          self.conv2 = nn.Conv2d(64, 128, kernel_size=5, stride=2)
119          self.conv3 = nn.Conv2d(128, 256, kernel_size=5, stride=2)
120          self.conv4 = nn.Conv2d(256, 512, kernel_size=5, stride=2)
121          self.conv5 = nn.Conv2d(512, 512, kernel_size=5, stride=2)
122          self.conv6 = nn.Conv2d(512, 32, kernel_size=1)
123          self.conv7 = nn.Conv2d(32, 1, kernel_size=(17, 4))
124          #self.conv_classify = nn.Conv2d(512, n_class, kernel_size=(17, 4))
125          self.conv_classify = nn.Conv2d(32, n_class, kernel_size=(17, 4))
126          self.drop1 = nn.Dropout2d(p=dp)
127          self.drop2 = nn.Dropout2d(p=dp)
128          self.drop3 = nn.Dropout2d(p=dp)
129          self.drop4 = nn.Dropout2d(p=dp)
130          self.drop5 = nn.Dropout2d(p=dp)
131          self.drop6 = nn.Dropout2d(p=dp)
132          self.ins_norm1 = nn.InstanceNorm2d(self.conv1.out_channels)
133          self.ins_norm2 = nn.InstanceNorm2d(self.conv2.out_channels)
134          self.ins_norm3 = nn.InstanceNorm2d(self.conv3.out_channels)
135          self.ins_norm4 = nn.InstanceNorm2d(self.conv4.out_channels)
136          self.ins_norm5 = nn.InstanceNorm2d(self.conv5.out_channels)
137          self.ins_norm6 = nn.InstanceNorm2d(self.conv6.out_channels)
```

Demo

End