



北京交通大学

移动端模型优化 与部署研究

实习实训项目报告

答辩学生：王德阳

HiFi-GAN理论理解和验证

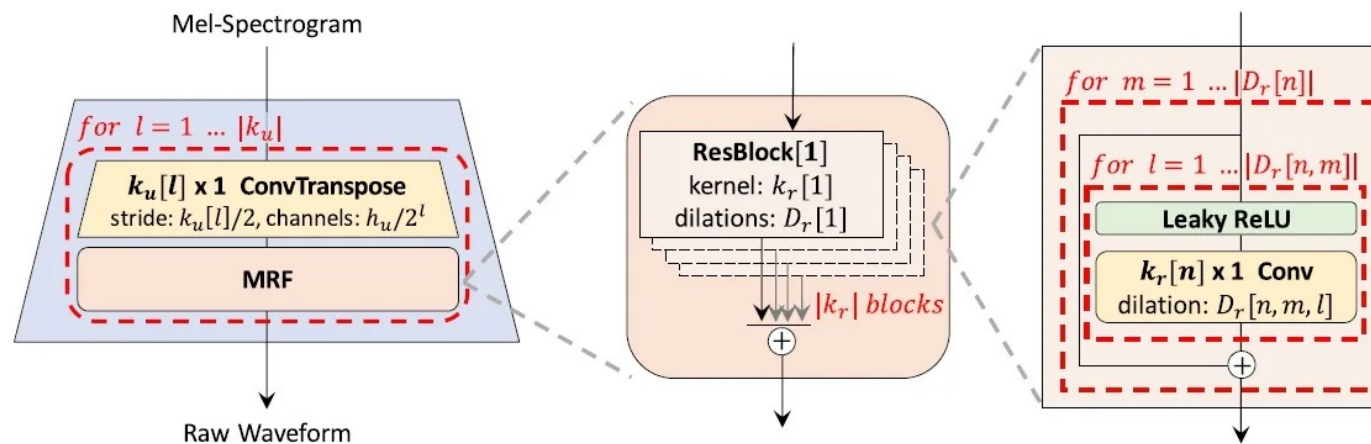


Figure 1: The generator upsamples mel-spectrograms up to $|k_u|$ times to match the temporal resolution of raw waveforms. A MRF module adds features from $|k_r|$ residual blocks of different kernel size and dilation rates. Lastly, the n -th residual block with kernel size $k_r[n]$ and dilation rates $D_r[n]$ in MRF module is depicted.

```
class Generator(torch.nn.Module):
    def __init__(self, h):
        super(Generator, self).__init__()
        self.h = h
        self.num_kernels = len(h.resblock_kernel_sizes)
        self.num_upsamples = len(h.upsample_rates)
        self.conv_pre = weight_norm(Conv1d(80, h.upsample_initial_channel, 7, 1, padding=3))
        resblock = ResBlock1 if h.resblock == '1' else ResBlock2

        self.ups = nn.ModuleList()
        for i, (u, k) in enumerate(zip(h.upsample_rates, h.upsample_kernel_sizes)):
            self.ups.append(weight_norm(
                ConvTranspose1d(h.upsample_initial_channel//(2**(i+1)),
                               h.upsample_initial_channel//(2**(i+1)),
                               k, u, padding=(k-u)//2)))

        self.resblocks = nn.ModuleList()
        for i in range(len(self.ups)):
            ch = h.upsample_initial_channel//(2**(i+1))
            for j, (k, d) in enumerate(zip(h.resblock_kernel_sizes, h.resblock_dilation_sizes)):
                self.resblocks.append(resblock(h, ch, k, d))

        self.conv_post = weight_norm(Conv1d(ch, 1, 7, 1, padding=3))
        self.ups.apply(init_weights)
        self.conv_post.apply(init_weights)

    def forward(self, x):
        x = self.conv_pre(x)
        for i in range(self.num_upsamples):
            x = F.leaky_relu(x, LRELU_SLOPE)
            x = self.ups[i](x)
            xs = None
            for j in range(self.num_kernels):
                if xs is None:
                    xs = self.resblocks[i*self.num_kernels+j](x)
                else:
                    xs += self.resblocks[i*self.num_kernels+j](x)
            x = xs / self.num_kernels
            x = F.leaky_relu(x)
            x = self.conv_post(x)
            x = torch.tanh(x)

        return x

    def remove_weight_norm(self):
        print('Removing weight norm...')
        for l in self.ups:
            remove_weight_norm(l)
        for l in self.resblocks:
            l.remove_weight_norm()
        remove_weight_norm(self.conv_pre)
        remove_weight_norm(self.conv_post)
```



HiFi-GAN理论理解和验证

HIFIGAN	Inference from wav file	输入的数据为由wav音频通过wav, sr = load_wav(os.path.join(a.input_wavs_dir, filename))提取出来的一维的列表, 类似[-9 0 0 ... -29 -32 -36], 通过 wav = torch.FloatTensor(wav).to(device)转换后, 尺寸变为torch.Size([41885]), x = get_mel(wav.unsqueeze(0))使其变为torch.Size([1, 80, 163]), 便于后续算子的输入。(第三维的长度作为数据的长度并不固定, 但前两个维度的值是确定的)
	Inference for end-to-end speech synthesis	通过x = np.load(os.path.join(a.input_mels_dir, filename))输入, 得到的其尺寸为(1, 80, 186), 通过wav = torch.FloatTensor(wav).to(device)转换后, 尺寸变为torch.Size([1, 80, 186]) (第三维的长度作为数据的长度并不固定, 但前两个维度的值是确定的)

y_g_hat = get_y_g_hat(y_g_hat尺寸为[1, 41728]) (第三维的长度作为数据的长度并不固定, 但前两个维度的值是确定的), 再将其尺寸变为torch.Size([1, 41728]) 成列表, 最终

Pre-processing Convolution		(batch_size, 80, input length)	(batch_size, 512, output length)	(512, 80, 7)	
E9 有提供int8版本的算子					
A	B	C	D	E	F
2 onnx算子	说明	对应的nn库算子	简略分析	结论	
Identity	这个算子返回的是输入本身, 不做任何修改	Nop	nn_node_ops nn_ops_for_Nop 这个节点是一个无操作的占位符节点, 没有特定的数据类型限制。	不受数据类型限制	
Unsqueeze	这个算子增加输入张量的维度, 但不增加新的元素	Reshape, ExpandDims	"QuantizedReshape": 输入数据是量化的8位整数 (8-bit quantized) 数据。 输出数据类型也是量化的8位整数。 ExpandDims_int32用于处理int32类型的张量数据, 它将输入张量复制到输出张量中, 没有进行任何实际操作。该节点支持1到3个输入张量和1个输出张量。 ExpandDims_f用于处理float类型的张量数据, 同样也是将输入张量复制到输出张量中, 没有进行任何实际操作。该节点也支持1到3个输入张量和1个输出张量。	Reshape有提供int8版本的算子 ExpandDims没有提供int8版本的算子	
Conv	这是卷积运算的算子, 主要在卷积神经网络 (Convolutional Neural Networks, CNNs) 中使用	QuantizedConv2d_8x8to32、 QuantizedConv2d_16x16to32	nn_ops_for_QuantizedConv2d_8x8to32: 这个算子执行的是8位整数 (int8) 量化的卷积操作。它接受8位整数格式的输入张量, 执行卷积计算, 并将结果输出为32位整数格式的张量。		
Relu	这是ReLU (Rectified Linear Unit) 激活函数的算子	QuantizedRelu_8	nn_ops_for_QuantizedRelu_8和nn_ops_for_QuantizedRelu_8_ref: 输入和输出数据类型为uint8 (8位无符号整数)。这是用于量化修正线性单元的操作。 nn_ops_for_QuantizedReluX_8和nn_ops_for_QuantizedReluX_8_ref: 输入数据类型为uint8 (8位无符号整数), 输出数据类型为uint8 (8位无符号整数)。这是用于量化修正线性单元 (带上限值) 的节点操作。	有提供int8版本的算子	
Transpose	这个算子将输入张量的维度进行转置	Transpose_x	OP_Transpose_8: 输入和输出数据类型为uint8 (8位无符号整数)	有提供int8版本的算子	
Constant	这个算子生成一个常数张量	Const	nn_ops_for_Const 常量节点的输出数据类型由数据本身的大小决定, 当数据大小为1字节时, 输出数据类型为uint8 (8位无符号整数)。	有提供int8版本的算子	
Reshape	这个算子改变输入张量的形状, 但不改变其元素	Reshape、QuantizedReshape	"QuantizedReshape": 输入数据是量化的8位整数 (8-bit quantized) 数据。 输出数据类型也是量化的8位整数。	有提供int8版本的算子	
Pow	这个算子执行幂运算	Mul_f、QuantizedMul_8x8to32	QuantizedMul_8x8to8: 操作: 量化的8位整数乘法, 结果为8位整数。	有提供int8版本的算子	
Cast	这个算子用于改变输入张量的数据类型	Convert_to_aix_d32、Convert_from_d32、 Convert_to_d32	Convert_8_16 / Convert_8_u16: 将输入的uint8类型数据转换为int16或uint16类型数据。 Convert_16_8 / Convert_u16_8: 将输入的int16或uint16类型数据转换为uint8类型数据。	不受数据类型限制	
Div	这个算子执行张量的元素级别的除法	Div_f	QuantizedDiv_8操作, 用于对两个量化的输入数据进行除法运算。在代码中, 数据类型主要使用了uint8_t, 也同样会使用float。	有提供int8版本的算子	
Sub	这个算子执行张量的元素级别的减法	Sub_f、Sub_int32	QuantizedSub_8p8to8用于处理8位量化数据的减法。	有提供int8版本的算子	
MatMul	这个算子执行矩阵乘法	QuantizedMatMul_8x8to32	nn_ops_for_QuantizedMatMul_8x8to32 nn_ops_for_QuantizedBatchMatMul_8x8to32 nn_ops_for_QuantizedMatMul_8x8to32_ref 这三个结构体定义是针对矩阵乘法运算的操作, 其中输入数据和输出数据的数据类型为8位整数, 输出为32位。		
Add	这个算子执行张量的元素级别的加法	Add_f、Add_int32	QuantizedAdd_8p8to8用于处理8位量化数据的加法。	有提供int8版本的算子	
ReduceMean	这个算子计算输入张量的所有元素的平均值	QuantizedMean_8	用于执行量化的均值 (mean) 计算的节点操作 (nn_ops_for_QuantizedMean_8)。这个节点操作用于处理8位量化数据。	有提供int8版本的算子	
Sqrt	这个算子计算输入张量每个元素的平方根	QuantizedSqrt_8	用于执行量化的平方根 (sqrt) 计算的节点操作 (nn_ops_for_QuantizedSqrt_8)。这个节点操作用于处理8位量化数据。	有提供int8版本的算子	



HiFi-GAN模型导出

```
model = Generator(h).to(device)

state_dict_g = load_checkpoint(a.checkpoint_file, device)
model.load_state_dict(state_dict_g['generator'])
model.eval()

# 导出为ONNX格式
output_path = 'hifigan_model.onnx'
seq_len=1000 # 可变长度
input_tensor = torch.randn(1, 80, seq_len)

# 动态轮廓设置
dynamic_axes = {'input': {2: 'seq_len'}, 'output': {2: 'seq_len'}}
torch.onnx.export(model, x, output_path, opset_version=11, verbose=False, dynamic_axes=dynamic_axes, input_names=['inputs'], output_names=['output'], do_constant_folding=True)
```

```
for i, filename in enumerate(filelist):
    wav, sr = load_wav(os.path.join(a.input_wavs_dir, filename))
    wav = wav / MAX_WAV_VALUE
    wav = torch.FloatTensor(wav).to(device)
    x = get_mel(wav.unsqueeze(0))
    x = x.to(torch.float32)
```





使用生成的 hifigan_model.onnx 文件进行推理

```
# 加载ONNX模型
model_path = 'C:\\Users\\YodelYang\\Desktop\\Python\\hifi-gan-master\\hifigan_model.onnx'
session = onnxruntime.InferenceSession(model_path)

# 运行模型
start_time = time.time()
outputs = session.run(None, {"inputs": x.numpy()})
```

```
# 获取输出结果
output_data = outputs[0]
output_data_tensor = torch.from_numpy(output_data)
output_data_tensor = output_data_tensor.cpu()
output_data_tensor = output_data_tensor * MAX_WAV_VALUE

output_data_array = output_data_tensor.numpy().astype('int16')

parser_new = argparse.ArgumentParser()
parser_new.add_argument('--input_wavs_dir', default='generated_files_onnx')
parser_new.add_argument('--output_dir', default='generated_files_onnx')
parser_new.add_argument('--checkpoint_file', required=True)
b = parser_new.parse_args()

output_file = os.path.join(b.output_dir, os.path.splitext(filename)[0] + '_generated_onnx.wav')
write(output_file, h.sampling_rate, output_data_array)
```



使用生成的 hifigan_model.onnx 文件进行推理

```
# 运行模型
start_time = time.time()
outputs = session.run(None, {"inputs": x.numpy()})
end_time = time.time()
elapsed2_time = end_time - start_time
print('Time consumption of existing methods: ', elapsed1_time, 'seconds')
print('Time consumption of onnx method:', elapsed2_time, 'seconds')
```

```
total_accuracy +=percentage
times+=1
print('Ratio of close elements:', percentage)
if times==100:
    break
```

```
Time consumption of existing methods: 2.716029644012451 seconds
Time consumption of onnx method: 1.1162590980529785 seconds
Ratio of close elements: 99.88922574626866
The total accuracy is 99.9
```



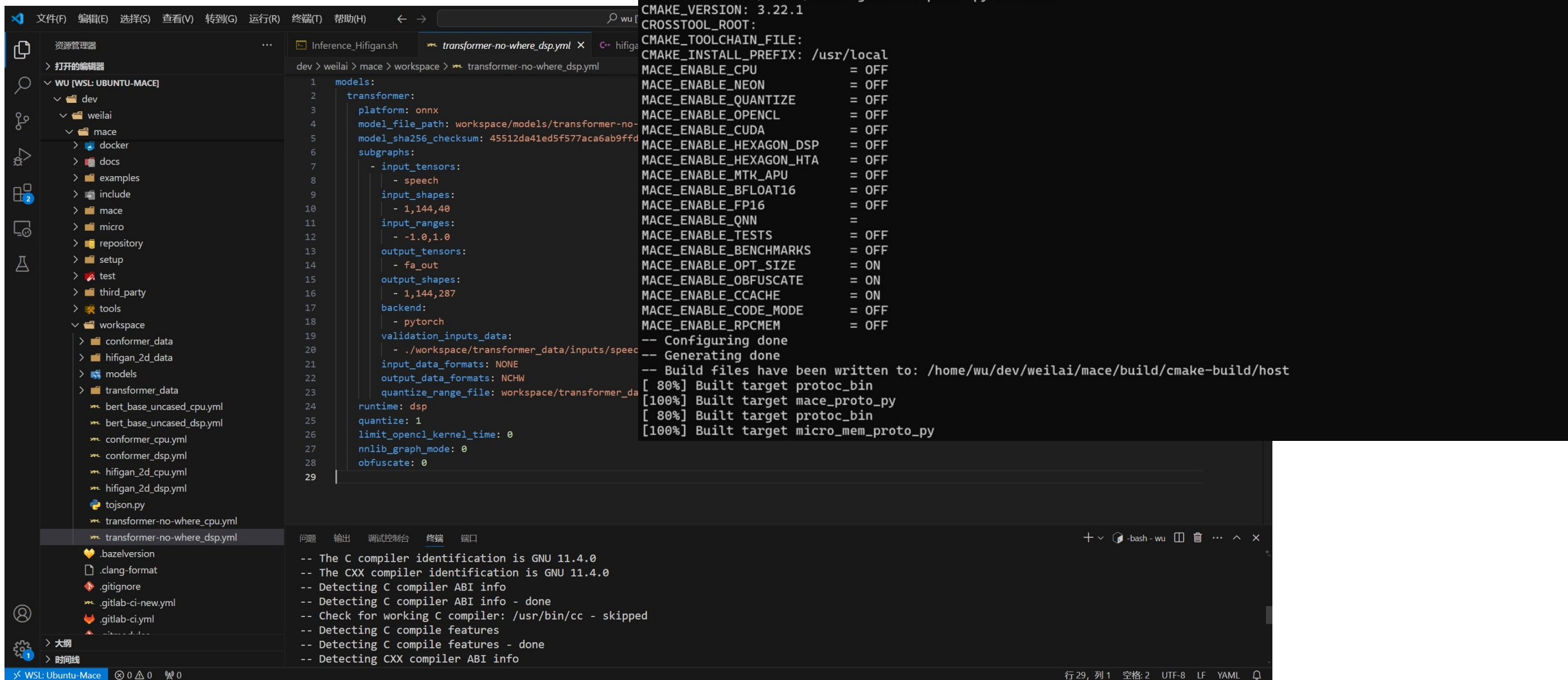

Linux和Android上C++版本的OnnxRuntime测试

Sheet1			
Sheet2			
Sheet3			
Sheet4			
Sheet5			
Sheet6			
Sheet7			
Sheet8			
Sheet9			
Sheet10			
Sheet11			
Sheet12			
Sheet13			
Sheet14			
Sheet15			
Sheet16			
Sheet17			
Sheet18			
Sheet19			
Sheet20			
Sheet21			
Sheet22			
Sheet23			
Sheet24			
Sheet25			
Sheet26			
Sheet27			
Sheet28			
Sheet29			
Sheet30			
Sheet31			
Sheet32			
Sheet33			
Sheet34			
Sheet35			
Sheet36			
Sheet37			
Sheet38			
Sheet39			
Sheet40			
Sheet41			
Sheet42			
Sheet43			
Sheet44			
Sheet45			
Sheet46			
Sheet47			
Sheet48			
Sheet49			
Sheet50			
Sheet51			
Sheet52			
Sheet53			
Sheet54			
Sheet55			
Sheet56			
Sheet57			
Sheet58			
Sheet59			
Sheet60			
Sheet61			
Sheet62			
Sheet63			
Sheet64			
Sheet65			
Sheet66			
Sheet67			
Sheet68			
Sheet69			
Sheet70			
Sheet71			
Sheet72			
Sheet73			
Sheet74			
Sheet75			
Sheet76			
Sheet77			
Sheet78			
Sheet79			
Sheet80			
Sheet81			
Sheet82			
Sheet83			
Sheet84			
Sheet85			
Sheet86			
Sheet87			
Sheet88			
Sheet89			
Sheet90			
Sheet91			
Sheet92			
Sheet93			
Sheet94			
Sheet95			
Sheet96			
Sheet97			
Sheet98			
Sheet99			
Sheet100			

```
(mace) ~/dev/weilai/hexagon_apps/onnxruntime/model/hifigan/cxx$ sudo bash Inference_Hifigan.sh -l 00 -p linux
*****Command Params*****
PLATFORM: linux
optim_level: 00
*****OnnxRuntime Python*****
python cost time: 2213.0 ms
output tensor shape: (1, 1, 66560, 1)
Success!
Compile Platform: linux
***** OnnxRuntime C++ Linux*****
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- BUILD_TYPE: Release
-- OPTIMIAZATION_LEVEL: 00
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wu/dev/weilai/hexagon_apps/onnxruntime/model/hifigan/cxx/build
[ 50%] Building CXX object CMakeFiles/Hifigan_onnx_linux.dir/hifigan_onnx.cpp.o
[100%] Linking CXX executable ../Hifigan_onnx_linux
[100%] Built target Hifigan_onnx_linux
c++ cost time: 2314 ms
output tensor shape: 1 1 66560 1
Success!
```



MACE框架下Transformer 成功运行



```
(mace) ~/dev/weilai/mace$ python tools/python/run_model.py --config ./workspace/yml/conformer_dsp.yml --benchmark --vlog_level=1 --runtime=dsp --target_abi=arm64-v8a
CMD> bash tools/cmake/cmake-generate-proto-py-host.sh
CMAKE_VERSION: 3.22.1
CROSS TOOL ROOT:
CMAKE_TOOLCHAIN_FILE:
CMAKE_INSTALL_PREFIX: /usr/local
MACE_ENABLE_CPU = OFF
MACE_ENABLE_NEON = OFF
MACE_ENABLE_QUANTIZE = OFF
MACE_ENABLE_OPENCL = OFF
MACE_ENABLE_CUDA = OFF
MACE_ENABLE_HEXAGON_DSP = OFF
MACE_ENABLE_HEXAGON_HTA = OFF
MACE_ENABLE_MTK_APU = OFF
MACE_ENABLE_BFLOAT16 = OFF
MACE_ENABLE_FP16 = OFF
MACE_ENABLE_QNN = OFF
MACE_ENABLE_TESTS = OFF
MACE_ENABLE_BENCHMARKS = OFF
MACE_ENABLE_OPT_SIZE = ON
MACE_ENABLE_OBFUSCATE = ON
MACE_ENABLE_CCACHE = ON
MACE_ENABLE_CODE_MODE = OFF
MACE_ENABLE_RPCMEM = OFF
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wu/dev/weilai/mace/build/cmake-build/host
[ 80%] Built target protoc_bin
[100%] Built target mace_proto_py
[ 80%] Built target protoc_bin
[100%] Built target micro_mem_proto_py
```

```
1 models:
2   transformer:
3     platform: onnx
4     model_file_path: workspace/models/transformer-no-
5     model_sha256_checksum: 45512da41ed5f577aca6ab9ff
6     subgraphs:
7       - input_tensors:
8         - speech
9         input_shapes:
10          - 1,144,40
11         input_ranges:
12          - -1.0,1.0
13         output_tensors:
14          - fa_out
15         output_shapes:
16          - 1,144,287
17         backend:
18          - pytorch
19         validation_inputs_data:
20          - ./workspace/transformer_data/inputs/speech
21         input_data_formats: NONE
22         output_data_formats: NCHW
23         quantize_range_file: workspace/transformer_da
24     runtime: dsp
25     quantize: 1
26     limit_opencl_kernel_time: 0
27     nnlib_graph_mode: 0
28     obfuscate: 0
29
```

```
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
```




Matmul和BatchMatmul算子 优化与性能提升

```
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)  < -> wu [WSL: Ubuntu-Mace]

资源管理器
> 打开的编辑器
WU [WSL: UBUNTU-MACE]
  > dev
    > weilai
      > mace
        > tools
          > dana
            > hexagon_compiler
            > image
            > python
              > __pycache__
              > config
              > micro
              > py_proto
              > quantize
              > template
              > transform
                > __pycache__
                > _init_.py
                > apu_converter.py
                > base_converter.py
                > caffe_converter.py
                > hexagon_converter.py 2
                > keras_converter.py
                > megengine_converter.py
                > onnx_converter.py
                > pytorch_converter.py
                > shape_inference.py
                > tensorflow_converter.py
                > transformer.py
              > utils
              > visualize
                > _init_.py
                > apu_utils.py
                > CMakeLists.txt
          > 大纲
          > 时间线
        > WSL: Ubuntu-Mace 0 0 2 0

Inference_Hifigan.sh hexagon_converter.py 2 x hifigan_onnx.cpp CMakeLists.txt onnxruntime-linux-as
dev > weilai > mace > tools > python > transform > hexagon_converter.py > HexagonConverter > convert_matmul
156 class HexagonConverter(base_converter.ConverterInterface):
1015
1016 def convert_instancenorm(self, op):
1017     affine = ConverterUtil.get_arg(op, MaceKeyword.mace_affine_str).i
1018     if not affine:
1019         del op.input[1:]
1020         self.add_min_max_const_node(op, op.input[0])
1021         op.type = HexagonOp.QuantizedInstanceNorm_8.name
1022     else:
1023         mace_check(False,
1024                     "Hexagon does not support instancenorm with affine")
1025
1026
1027 def convert_matmul(self, op):
1028     requantize_op = copy.deepcopy(op)
1029
1030     if len(op.output_shape[0].dims) == 4:
1031         op.output[0] = self.new_tensor(op.output[0], '_batchmatmul:0', op.output_shape[0].dims)
1032         op.type = HexagonOp.QuantizedBatchMatMul_8x8to32.name
1033     else:
1034         op.output[0] = self.new_tensor(op.output[0], '_matmul:0', op.output_shape[0].dims)
1035         op.type = HexagonOp.QuantizedMatMul_8x8to32.name
1036
1037     self.add_min_max_const_node(op, op.input[0])
1038     self.add_min_max_const_node(op, op.input[1])
1039     op.name = op.name + '_matmul'
1040     self.post_convert(op)
1041
1042     del requantize_op.input[1:]
1043     requantize_op.input[0] = op.output[0]
1044     self.add_min_max_const_node(requantize_op, requantize_op.input[0], True, True, False)
1045     self.add_min_max_const_node(requantize_op, requantize_op.output[0], True, True, False)
1046
1047
1048 -- The C compiler identification is GNU 11.4.0
1049 -- The CXX compiler identification is GNU 11.4.0
1050 -- Detecting C compiler ABI info
1051 -- Detecting C compiler ABI info - done
1052 -- Check for working C compiler: /usr/bin/cc - skipped
1053 -- Detecting C compile features
1054 -- Detecting C compile features - done
1055 -- Detecting CXX compiler ABI info
```

66 分开Batchmatmul和Matmul之后:

67 Sort by Duration

68	Node	Type	Times	Duration(ms)
69				
70				
71		QuantizedDiv_8	13	43.847
72		QuantizedMatMul_8x8to32	26	14.342
73		Convert_to_d32	107	7.259
74		Transpose_8	21	6.357
75		QuantizedBatchMatMul_8x8to32	8	5.677
76		Requantize_32to8	34	5.174
77		QuantizedReshape	18	3.406
78		QuantizedAdd_8p8to8_d32	52	3.230
79		QuantizedMul_8x8to8	9	2.549
80		QuantizedMean_8	18	1.632
81		Supernode_8x8p32to8_d32	1	0.988
82		QuantizedMul_8x8to8_d32	11	0.719
83		QuantizedSub_8p8to8_d32	10	0.607
84		QuantizedRelu_8	4	0.577
85		QuantizedSoftmax_8_d32	5	0.484
86		Convert_from_d32	69	0.347
87		InputSupernode_8x8p32to8_outd32	1	0.113
88		QuantizeINPUT_f_to_8	1	0.079
89		DequantizeOUTPUT_8tof	1	0.052
90		QuantizedSqrt_8	9	0.049
91		QuantizedAdd_8p8to8	1	0.014
92				
93				



项目贡献

1. 深入理解了HiFi-GAN模型的理论基础和算法结构。
2. 成功导出了HiFi-GAN模型为ONNX格式，并验证了其导出模型的正确性和性能。
3. 优化和部署了ONNX模型在移动设备上的表现，通过性能测试和算子优化提升了模型的推理速度和效率。
4. 探索了MACE框架在移动端的应用，实现了模型的量化和性能优化。





北京交通大学

请评委老师批评指正

答辩学生：王德阳