WeNet: Production First and Production Ready End-to-End Speech Recognition Toolkit

WeNet：生产优先和生产就绪的端到端语音识别工具包

*Binbin Zhang1, Di Wu1, Chao Yang1, Xiaoyu Chen1, Zhendong Peng1, Xiangming Wang1, Zhuoyuan Yao2, Xiong Wang2, Fan Yu2, Lei Xie2, Xin Lei1*

*张彬彬1，吴迪1，杨超1，陈晓宇1，彭振东1，王向明1，姚卓元2，王雄2，范宇2，谢磊2，辛磊1*

1Mobvoi Inc，Beijing, China

2Audio, Speech and Language Processing Group (ASLP@NPU), School of Computer Science, Northwestern Polytechnical University, Xi’an, China

binbinzhang@mobvoi.com

# 摘要

In this paper, we present a new open source, production first and production ready end-to-end (E2E) speech recognition toolkit named WeNet. The main motivation of WeNet is to close the gap between the research and the production of E2E speech recognition models. WeNet provides an efficient way to ship ASR applications in several real-world scenarios, which is the main difference and advantage to other open source E2E speech recognition toolkits. This paper introduces WeNet from three aspects, including model architecture, framework design and performance metrics. Our experiments on AISHELL-1 using WeNet, not only give a promising character error rate (CER) on a unified streaming and non-streaming two pass (U2) E2E model but also show reasonable RTF and latency, both of these aspects are favored for production adoption. The toolkit is publicly available at https://github.com/mobvoi/wenet.

**Index Terms**: WeNet, Production Ready, U2

在本文中，我们提出了一个新的开源，生产第一和生产准备端到端（E2E）语音识别工具包命名为WeNet。WeNet的主要目的是缩小E2E语音识别模型的研究和生产之间的差距。WeNet提供了一种在多个真实场景中发布ASR应用程序的有效方法，这是与其他开源E2E语音识别工具包的主要区别和优势。本文从模型体系结构、框架设计和性能度量三个方面介绍了WeNet。我们在AISHELL-1上使用WeNet进行的实验，不仅给出了一个统一的流式和非流式两遍（U2）E2E模型下有希望的字符错误率（CER），而且还显示了合理的RTF和延迟，这两个方面都有利于产品的采用。该工具包在https://github.com/mobvoi/wenet。

**关键词**：WeNet，生产就绪，U2

2021.01 arxiv

# 1.     前言

E2E模型，包括CTC[1，2]、递归神经网络变换器（RNN-T）[3，4]、和基于注意的编解码器（AED）[5，6，7]，近年来在语音识别领域得到了越来越多的关注。与混合ASR框架相比，E2E模型最大的优点是训练过程极其简化。最近的工作[8[[1]](#footnote-1)，9[[2]](#footnote-2)，10]也表明E2E系统在字错误率（WER）标准上已经超过了传统的混合ASR系统。考虑到前面提到的E2E模型的优点，将新兴的ASR框架部署到实际产品中是必要的。然而，部署一个E2E系统非常繁琐，有很多问题需要解决。

首先，**流式问题**。对于许多需要ASR系统以低延迟快速响应的场景，流式推理是必不可少的。然而，一些E2E模型很难进行流式处理，例如AED；让此类模型以流式方式工作[11[[3]](#footnote-3)，12[[4]](#footnote-4)，13[[5]](#footnote-5)]，需要付出很大的努力，或者导致较大的精度损失。

第二，**统一的流式和非流式模型**。统一的流式和非流式模型可以减少开发工作，降低训练和部署成本，尤其是对于小型公司，这是生产上的首选[14[[6]](#footnote-6)，15[[7]](#footnote-7)]。

第三，**生产问题**，这也是我们在WeNet设计中最关心的问题。尽管我们已经有了一个统一的模型，该模型在流式和非流式应用程序上都具有不错的性能，但要将E2E模型推广到实际的生产应用程序中还需要付出很大的努力。首先，我们必须将研究模型转换为生产模型，这对于基于动态图的深度学习工具包（如PyTorch[16]）来说非常困难。其次，需要从模型体系结构、应用程序和运行平台三个方面对推理工作流进行细致的设计。对于模型结构，大多数E2E模型首先进行编码器前向计算，然后进行自回归波束搜索。此工作流程比起一个简单网络做前向推理要复杂的多，如果将流式处理考虑进来，问题会变得更复杂。对于云端和on-device应用，还需要认真考虑计算和内存成本。特别是对于设备端模型，推理优化和模型量化起着非常重要的作用。对于运行时平台，虽然有各种平台可以用来进行神经网络推理，比如ONNX（Open neural network Exchange），Pytorch中的LibTorch，TensorRT[17]，OpenVINO，MNN[18]，NCNN，但仍然需要专业的语音知识和超高的深度学习优化知识来为应用程序选择最佳的平台。

在这项工作中，我们提出了WeNet来解决上述问题。WeNet中的“We”是以“WeChat”为灵感的，意思是连接和分享，“Net”来自Espnet[19]，因为我们参考了Espnet中很多优秀的设计。WeNet的主要初衷是缩小E2E语音识别模型的研究与生产之间的差距，减少E2E模型生产的工作量，探索更好的E2E模型进行生产。因此，WeNet本质上是为生产而设计的，这使得我们的WeNet明显不同于其他工具包。

WeNet以生产优先和生产就绪（production first and production ready）原则进行实现。首先，WeNet采用统一两遍（U2）[20[[8]](#footnote-8)]框架来解决流式和非流式问题。其次，从模型训练到部署，WeNet只依赖于PyTorch及其生态系统。此外，WeNet还为云服务器和设备上（Android）部署提供了一个现成的pipeline。WeNet的主要优势是：

1.   **生产优先，生产就绪**：WeNet的Python代码满足TorchScript的要求，因此WeNet训练的模型可以直接通过Torch JIT导出，并使用LibTorch进行推理。研究模式和生产模式之间没有差距。模型推理既不需要模型转换，也不需要额外的代码。

2.   **流式和非流式ASR的统一解决方案**：WeNet采用U2框架，实现了准确、快速、统一的E2E模型，有利于行业采用。

3.   **可移植运行时**：将提供几个运行时来展示如何在不同的平台上托管经过WeNet训练的模型，包括服务器（x86）和嵌入式（Android平台中的ARM）。

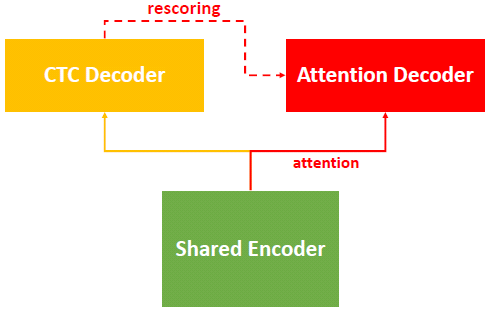
4.   **轻量级**：WeNet是专门为E2E语音识别设计的，代码简洁明了。这一切都是基于PyTorch及其相应的生态系统。它不依赖于Kaldi，简化了安装和使用。

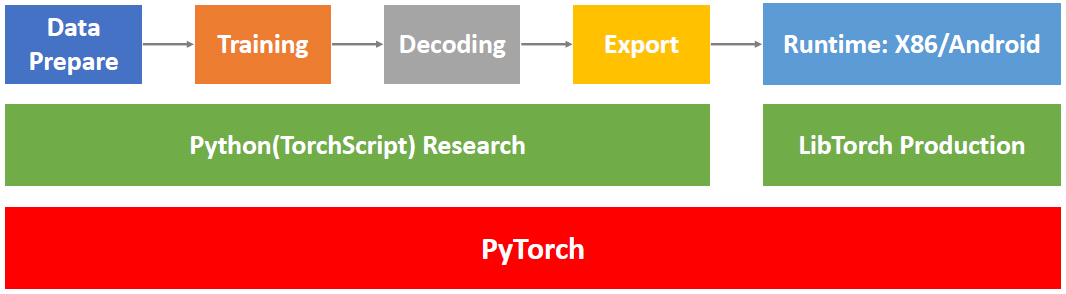
实验表明，WeNet是一个简单、准确的语音识别工具，从研究到生产都提供了端到端的解决方案。

# 2.     相关工作

Espnet是目前最流行的端到端语音研究的开源平台，它主要致力于端到端的自动语音识别（ASR），采用了广泛使用的动态神经网络工具箱Chainer和PyTorch作为主要的深度学习引擎。它提供了E2E实现，包括CTC、AED、RNN-T和RNN语言模型重打分。虽然Espnet在科研中应用广泛，但Espnet训练出来的模型在生产中很难直接使用，Espnet设计中也没有生产方面的考虑和支持。

目前，还没有专注于生成级别的E2E语音识别工具包。





# 3.     WeNet

## *3.1.    模型体系结构*

由于我们的目标是解决流式问题，统一问题，生产问题，解决方案应该简单，易于搭建和训练，具有良好的性能以及易于在运行时生产。

如图1所示，一个统一的双通道解决了这些问题，这是一个联合的CTC/AED模型。它由三部分组成，一个共享编码器，一个CTC解码器和一个注意力解码器。共享编码器由多个Transformer[21]或Conformer[22]编码器层组成。CTC解码器由一个线性层组成，该层将共享编码器输出转换为CTC激活。注意译码器由多个Transformer译码器层组成。共享编码器只能看到有限的右侧上下文，CTC解码器在第一遍中以流模式运行，在第二遍中使用注意解码器以给出更准确的结果。

### 3.1.1.    训练

训练中采用了CTC损失和AED损失的组合损失，如式1所示，其中x是声学特征，是y相应的注释，LCTC(x, y)，LAED(x, y)分别是CTC和AED损失，λ是平衡CTC和AED损失重要性的超参数。

Lcombined(x, y) = λLCTC (x, y) + (1−λ) LAED (x, y)（1）

在训练中采用了动态块训练技术，统一了非流式模型和流式模型，实现了时延控制。首先，将输入按固定的块大小分成若干块，每个块的输入为[t+1，t+2，…，t+C]，每个块都关注它自己和之前的所有块，第一遍解码时CTC整体延迟仅取决于块大小。如果块大小是受限的，它将以流方式工作，反之它将以非流方式工作。第二，在训练过程中，块大小是从1到当前训练话语的最大长度之间变化，训练后的模型学习到了预测任意大小的块。根据经验，块大小越大，延迟越高，结果越好，因此我们可以通过在运行时调整块大小来轻松地平衡准确性和延迟。

### 3.1.2.    解码

对于研究阶段的Python解码，为了比较和评估CTC/AED联合模型的不同部分，WeNet支持以下四种解码模式：

1.   attention：在模型的AED部分应用标准自回归波束搜索。

2.   ctc贪心搜索：将ctc贪心搜索应用到模型的ctc部分，ctc贪心搜索比其他模式速度快。

3.   ctc前缀波束搜索：在模型的ctc部分应用ctc前缀波束搜索，可以给出n个最佳候选。

4.   attention\_rescoring：首先在模型的CTC部分应用CTC前缀波束搜索生成n个最佳候选，然后在AED解码器部分用相应的编码器输出重排序n个最佳候选。

对于运行时阶段的解码，只支持attention\_rescoring，因为它是我们的最终生产解决方案。

## *3.2.    系统设计*

WeNet的总体设计栈如图2所示。整个框架完全基于PyTorch，它是一个生态系统作为底层栈，您将在下一节看到，TorchScript用于开发模型，Torchaudio用于动态特征提取，DistributedDataParallel用于分布式训练，torch JIT（实时）用于模型导出，量化模型采用PyTorch量化，生产运行时采用LibTorch量化。

第二个堆栈由两部分组成。Python（TorchScript）Research是为了开发一个研究模型，TorchScript是用来确保模型能够正确导出为生产模型的。LibTorch Production用于托管生产模型，该模型设计用于支持各种硬件和平台，如CPU、GPU（CUDA）、Linux、Android和IOS。

第三部分是对WeNet生产管道的典型研究，下面将对这些模块进行详细设计。

### 3.2.1.    数据准备

WeNet中的数据准备非常简单，一个Kaldi风格的标签文件和wave列表文件，一个将模型单元映射到相应整数id的模型单元字典文件，就是您所需要的。在数据准备阶段不需要任何特征提取，因为我们在训练中使用动态特征提取。

### 3.2.2.    训练

WeNet的培训阶段有以下主要特点：

动态特征提取：这是基于Torchaudio的，它可以生成与Kaldi相同的FBANK特征。由于特征是从原始PCM数据中动态提取的，因此可以同时在时间、频率和最终特征三个层次上对原始PCM进行数据扩充，丰富了数据的多样性和数据扩充。

联合训练：联合训练加快了训练的收敛速度，提高了训练的稳定性，并给出了更好的识别结果。

分布式训练：WeNet支持多个GPU训练，PyTorch中的分布式数据并行。

### 3.2.3.    解码

提供了一组Python工具来识别wave文件并计算其精度。这些工具帮助用户在将模型部署到生产环境之前验证和调试模型。支持第3.1.2节中的所有解码算法。

### 3.2.4.    导出

由于WeNet模型是在TorchScript中实现的，因此可以通过torch-JIT直接安全地导出到生产模型中。然后可以在运行时使用LibTorch库托管导出的模型。支持浮点模型和量化int8模型。在Android设备上使用量化模型可以使推理速度提高一倍甚至更多。

### 3.2.5.    运行时

目前，我们支持在两个主流平台上托管WeNet生产模型，即x86作为服务器运行时和Android作为设备运行时。提供了两个平台的C++ API库和可运行的演示。用户还可以通过使用C++库来实现他们的定制系统。我们仔细评估了ASR系统的三个关键指标，即准确度、实时因子（RTF）和延迟。该性能适用于许多ASR应用，如服务API和设备上的语音助理。结果见第4.2节。

# 4.     实验

我们在开源的汉语普通话语音语料库AISHELL-1[23]上进行了实验，其中包括150小时的训练集、10小时的开发集和5小时的测试集。测试集共包含7176个句子。我们使用Torchaudio实时计算的80维log-Mel滤波器组（FBANK），以25ms窗口和10ms偏移作为特征。SpecAugment[24]应用了2个最大频率掩码（*F*=10）的频率掩码和2个最大时间掩码（*T*=50）的时间掩码。编码器前端采用了两个卷积降采样层，核尺寸为3\*3，步长为2，所以最终降采样至1/4。我们使用12个transformer层作为编码器，6个transformer层作为解码器。训练中使用Adam优化器和一个具有25000个warm-up步骤的学习率计划。此外，我们通过对top-K最佳模型（对dev的损失较小）取平均得到最终模型。

## *4.1.    统一模型评价*

我们首先训练了一个非流式模型（M1）作为我们的基线，该模型是在full attention的情况下训练和推理的。然后采用动态组块策略训练统一模型（M2）。在解码时用不同的块大小——full/16/8/4——推断M2，full表示完全注意非流式情况，16/8/4表示流式情况。

表1：统一模型评估

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 解码方法 | M1 | full | M2  16 | 8 | 4 |
| attention | 5.76 | 6.13 | 6.43 | 6.59 | 6.8 |
| ctc贪婪搜索 | 6.21 | 6.75 | 7.85 | 8.41 | 9.44 |
| ctc前缀波束搜索 | 6.21 | 6.74 | 7.85 | 8.41 | 9.43 |
| attention\_rescoring | 5.47 | 5.79 | 6.50 | 6.89 | 7.49 |

首先，如表1所示，统一模型不仅在full attention情况下显示出与非流式模型相当的结果，而且在有限块大小为16/8/4的流式情况下也给出了很好的结果，这表明了动态块训练策略的有效性。

第二，通过比较这四种解码模式，我们可以看出，无论是在非流模式还是在统一模式下，attention\_rescoring模式都能提高CTC的效果。ctc贪婪搜索和ctc前缀波束搜索的性能几乎相同，并且随着块大小的减小，它们的性能会显著降低。attention模式随着窗口减小性能略微下降；attention\_rescoring模式可以减轻ctc前缀波束搜索模式的下降幅度。如U2论文所示，由于attention模式是一个自回归过程，而attention\_rescoring不是，因此attention\_rescoring比attention更快，具有更好的RTF。总的来说，attention\_rescoring不仅取得了较好的结果，而且具有较低的RTF。

因此，在生产上我们选择了基于动态块的统一模型和attention\_rescoring解码方式，runtime时只支持attention\_rescoring。

## *4.2.    运行时基准*

本节展示了上述统一模型M2上的量化、RTF和延迟基准。我们分别在x86服务器平台和arm android设备平台上进行了基准测试。

对于x86云服务器，CPU为4核Intel（R）Xeon（R）CPU E5-2620 v4@2.10GHz，内存总计16G。解码每个句子时只用一个cpu线程进行工作和TorchScript推断。因为云服务需要并行处理，单线程测试可以避免并行测试时的性能下降。

对于Android设备，CPU为4核高通Snapdragon 865，内存为8.00GB。在设备上推理也使用单线程。

### 4.2.1.    量化

这里我们只是比较量化前后的CER差异。量化的RTF在下面的部分中示出。如表2所示，当应用量化时，CER是可比较的。float模型的CER与我们在表1中列出的略有不同，因为表1是由Python研究工具测试的，而这里的结果是由运行时工具测试的。

表2：量化前后的CER

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 量化/解码块 | 满的 | 16 | 8 | 4 |
| 否（浮动32） | 5.87 | 6.49 | 6.88 | 7.46 |
| 是（国际8） | 5.89 | 6.54 | 6.89 | 7.51 |

### 4.2.2.    RTF

如表3所示，我们可以看到RTF随着块大小的减小而增加，因为较小的块需要更多的迭代来进行正向计算。此外，量化在设备（Android）上产生大约2倍的加速，在服务器（x86）上有些许的提升。

### 4.2.3.    延迟

对于延迟基准测试，我们创建了一个WebSocket服务器/客户机来模拟真实的流应用程序。这个基准只在x86服务器平台上执行。我们评估的平均延迟如下所述：

表3:RTF基准

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模型/解码块 | full | 16 | 8 | 4 |
| 服务器（x86）float32 | 0.079 | 0.095 | 0.128 | 0.186 |
| 服务器（x86）int8 | 0.072 | 0.081 | 0.098 | 0.134 |
| 设备上（Android）float32 | 0.164 | 0.251 | 0.350 | 0.505 |
| 设备上（Android）int8 | 0.082 | 0.114 | 0.130 | 0.201 |

1.   模型延迟（L1）：模型结构引入的等待时间。对于基于块的解码，平均等待时间是块的一半。我们模型的总模型延迟是（chunk/2\*4+6）\*10（ms），其中4是子采样率，6是编码器中前两个CNN层引入的lookahead，10是帧移。

2.   重新扫描成本（L2）：第二遍注意力重新扫描的时间成本。

3.   最终延迟（L3）：用户（客户端）感知到的延迟，即用户停止说话和获得识别结果之间的时间差。当我们的ASR服务器接收到语音停止信号时，它首先处理剩下的语音进行CTC搜索，然后进行第二遍attention重打分，因此重打分成本是最终延迟的一部分。在实际生产中也应该考虑网络延迟，但是因为我们在同一台机器上测试了服务器/客户机，所以网络延迟可以忽略不计。

表4：延迟基准

|  |  |  |  |
| --- | --- | --- | --- |
| 解码块 | L1（毫秒） | L2（毫秒） | L3（毫秒） |
| 16 | 380 | 115 | 142 |
| 8 | 220 | 115 | 135 |
| 4 | 140 | 114 | 130 |

正如我们在表4中看到的，首先，对于不同的块大小，重打分成本几乎相同，这是合理的，因为重打分计算对块大小是不变的。第二，最终延迟中重打分占绝对比重，这意味着我们可以通过降低重打分成本来进一步降低最终延迟。第三，当解码块从4到8到16变化时，最终延迟稍微增加。

# 5.     结论

我们提出了一个新的开源E2E语音识别工具包，它是production first和production ready的。我们提供了用于流式/非流式的统一解决方案，附带基准测试的准确性，RTF和延迟。整个工具包设计精良，轻量级，表现出很好的性能。

# 6.     工具书类

[1] A. Graves, S. Fern´andez, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks,” in Proceedings of the 23rd international conference on Machine learning, 2006, pp. 369–376.

[2] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen et al., “Deep speech 2: End-to-end speech recognition in english and mandarin,” in International conference on machine learning, 2016, pp. 173–182.

[3] A. Graves, “Sequence transduction with recurrent neural networks,” arXiv preprint arXiv:1211.3711, 2012.

[4] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, 2013, pp. 6645–6649.

[5] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio, “End-to-end continuous speech recognition using attention-based recurrent nn: First results,” arXiv preprint arXiv:1412.1602, 2014.

[6] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, “Listen, attend and spell,” arXiv preprint arXiv:1508.01211, 2015.

[7] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-based models for speech recognition,” in Advances in neural information processing systems, 2015, pp. 577–585.

[8] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, “A comparison of sequence-to-sequence models for speech recognition.” in Interspeech, 2017, pp. 939–943.

[9] T. N. Sainath, R. Pang, D. Rybach, Y. He, R. Prabhavalkar, W. Li, M. Visontai, Q. Liang, T. Strohman, Y. Wu et al., “Two-pass end-to-end speech recognition,” arXiv preprint arXiv:1908.10992, 2019.

[10] S. Kim, T. Hori, and S. Watanabe, “Joint ctc-attention based end-to-end speech recognition using multi-task learning,” in 2017 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2017, pp. 4835–4839.

[11] C. Raffel, M.-T. Luong, P. J. Liu, R. J.Weiss, and D. Eck, “Online and linear-time attention by enforcing monotonic alignments,” arXiv preprint arXiv:1704.00784, 2017.

[12] C.-C. Chiu and C. Raffel, “Monotonic chunkwise attention,” arXiv preprint arXiv:1712.05382, 2017.

[13] H. Inaguma, M. Mimura, and T. Kawahara, “Enhancing monotonic multihead attention for streaming asr,” arXiv preprint arXiv:2005.09394, 2020.

[14] J. Yu, W. Han, A. Gulati, C.-C. Chiu, B. Li, T. N. Sainath, Y. Wu, and R. Pang, “Universal asr: Unify and improve streaming asr with full-context modeling,” arXiv preprint arXiv:2010.06030, 2020.

[15] A. Tripathi, J. Kim, Q. Zhang, H. Lu, and H. Sak, “Transformer transducer: One model unifying streaming and non-streaming speech recognition,” arXiv preprint arXiv:2010.03192, 2020.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., “Pytorch: An imperative style, high-performance deep learning library,” in Advances in neural information processing systems, 2019, pp. 8026–8037.

[17] H. Vanholder, “Efficient inference with tensorrt,” 2016.

[18] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu, C. Lv, and Z. Wu, “Mnn: A universal and efficient inference engine,” in MLSys, 2020.

[19] S.Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. E. Y. Soplin, J. Heymann, M. Wiesner, N. Chen et al., “Espnet: End-to-end speech processing toolkit,” arXiv preprint arXiv:1804.00015, 2018.

[20] B. Zhang, D.Wu, Z. Yao, X.Wang, F. Yu, C. Yang, L. Guo, Y. Hu, L. Xie, and X. Lei, “Unified streaming and non-streaming twopass end-to-end model for speech recognition,” arXiv preprint arXiv:2012.05481, 2020.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in Advances in neural information processing systems, 2017, pp. 5998–6008.

[22] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu,W. Han, S. Wang, Z. Zhang, Y. Wu et al., “Conformer: Convolutionaugmented transformer for speech recognition,” arXiv preprint arXiv:2005.08100, 2020.

[23] H. Bu, J. Du, X. Na, B. Wu, and H. Zheng, “Aishell-1: An opensource mandarin speech corpus and a speech recognition baseline,” in 2017 20th Conference of the Oriental Chapter of the International Coordinating Committee on Speech Databases and Speech I/O Systems and Assessment (O-COCOSDA). IEEE, 2017, pp. 1–5.

[24] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, “Specaugment: A simple data augmentation method for automatic speech recognition,” arXiv preprint arXiv:1904.08779, 2019.

1. *A comparison of sequence-to-sequence models for speech recognition*. 2017. [↑](#footnote-ref-1)
2. *Two-pass end-to-end speech recognition*. 2019. [↑](#footnote-ref-2)
3. *Online and linear-time attention by enforcing monotonic alignments.* 2017. [↑](#footnote-ref-3)
4. *Monotonic chunkwise attention.* 2017. [↑](#footnote-ref-4)
5. *Enhancing monotonic multihead attention for streaming asr.* 2017. [↑](#footnote-ref-5)
6. *Universal asr: Unify and improve streaming asr with full-context modeling.* 2020. [↑](#footnote-ref-6)
7. *Transformer transducer: One model unifying streaming and non-streaming speech recognition.* 2020. [↑](#footnote-ref-7)
8. *Unified streaming and non-streaming twopass end-to-end model for speech recognition.* 2020. [↑](#footnote-ref-8)