电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE

论文题目 **面向噪声环境的嵌入式语音识别系统设计与实现**

专业学位类别	工程硕士
学　　号	**201922080801**
作者姓名	胥江
指导教师	吴跃 教授
学　　院	计算机科学与工程学院

分类号 TP399 密级 公开

UDC [注1] 004.45

学 位 论 文

面向噪声环境的嵌入式语音识别系统设计与实现

（题名和副题名）

胥江

（作者姓名）

指导教师 吴跃 教 授

电子科技大学 成 都

（姓名、职称、单位名称）

申请学位级别 硕士 专业学位类别 工程硕士

专业学位领域 计算机技术

提交论文日期 2022 年 3 月 23 日 论文答辩日期 2022 年 5 月 20 日

学位授予单位和日期 电子科技大学 2022 年 6 月

答辩委员会主席 陈端兵

评阅人

注 1：注明《国际十进分类法 UDC》的类号。

Design and Implementation of Embedded Speech Recognition System for Noise Environment

A Master Thesis Submitted to
University of Electronic Science and Technology of China

Discipline	**Master of Engineering**
Student ID	**201922080801**
Author	**Xu Jiang**
Supervisor	**Wu Yue**
School	**School of Computer Science and Engineering**

摘 要

作为机器听觉领域研究最为广泛的技术，自动语音识别在智能家居、智能汽车、智能手机虚拟助手等场景下都有非常重要的应用价值。在理想环境下，自动语音识别的精度已经接近饱和。但是在噪声场景尤其是高噪声场景下，受稳态和非稳态噪声的影响，语音识别的精度出现大幅度下降乃至完全失去识别效果。同时，在一些特定的对能源使用要求较为严苛的军事应用场景，如航空航天卫星、全天候陆地侦察车等，对低功耗的嵌入式语音识别方案有较大需求。因此本文针对特定的噪声环境研究了嵌入式语音识别方案，并设计和实现了一个小型的嵌入式语音识别系统。本文的主要工作内容和创新点如下：

1. 考虑到稳态噪声和非稳态噪声对语音识别的影响，设计实现了满足实时性的语音降噪前端，作为语音识别的预处理前端。考虑到嵌入式平台的硬件资源限制，基于 DNN-HMM 框架设计了轻量化的语音识别模型 TDNN-13-SVD-M-Denoise-Trigram，结合语音降噪前端进行实验，在信噪比为[-10,2]的特定强噪声环境下测试 AISHELL 测试集，语音识别字错率为 6.81%。其中，语音降噪前端带来了 0.3%-0.5%的错误率下降。
2. 考虑到语音识别推理的性能，基于嵌入式异构计算环境（国产 FT-2000A CPU 和国产 690T FPGA）设计和实现了 TDNN 网络加速引擎，在 TDNN-13-SVD-M-Denoise 模型上测试，推理性能是 OpenBLAS 库加速的 3.07 倍。
3. 考虑到实际应用的需求，优化和实现了 C++嵌入式语音识别 API，该 API 既提供了语音识别功能，也提供了非固定关键词检索功能，声学模型的 DNN 网络可以选择 OpenBLAS 和 TDNN 加速引擎两种方案进行加速，由 CMake 进行配置。最后采用 6-8s 的语音数据测试 API 的推理接口，基于 OpenBLAS 加速的语音识别时间在 1.2s 左右，基于 TDNN 引擎加速的语音识别时间在 750ms 左右。同时，API 的语音识别接口的字错率为 7.13%。
4. 基于 API 设计和实现了一个离线、近场的嵌入式语音识别系统，该系统可对录音中的静音音频和纯背景噪声进行过滤，并且可根据语音内的停顿分割语句。系统的运行功耗仅为 5-6W，配备电池后可支持长时间的使用。

综上，本文设计的嵌入式语音识别系统具备抗噪性、实时性和低功耗的特点。

关键词：嵌入式语音识别，语音降噪，轻量化网络，现场可编程门阵列

ABSTRACT

As the most widely studied technology in the field of machine hearing, automatic speech recognition has very important application in smart home, smart car, virtual assistant for smart phone and other scenarios. Under ideal circumstances, the accuracy of automatic speech recognition is close to saturation. However, in noise scenes, especially in high noise scenes, the accuracy of speech recognition is greatly reduced or even completely lost due to the influence of steady and unsteady noises. At the same time, in some specific military application scenarios with strict energy usage requirements, such as aerospace satellites, all-weather land reconnaissance vehicles, etc., it is imperative to design an embedded speech recognition mechanism with low energy comsumption. Therefore, this thesis focuses on the noise-oriented embedded speech recognition scheme for a specific noise environment, and designs and implements a small embedded speech recognition system. The main innovations and workload of this thesis can be summarized as follows:

1. Considering the influence of steady-state noise and non-steady-state noise on speech recognition, a real-time speech noise reduction front-end is designed and implemented as a pre-processing front-end for speech recognition. Considering the hardware resource limitation of the embedded platform, a lightweight speech recognition model TDNN-13-SVD-M-Denoise-Trigram is designed based on DNN-HMM framework. Experiments are carried out in combination with the speech denoising front-end. The AISHELL test set is tested in a specific strong noise environment with a SIGNAL-NOISE RATIO(SNR) of [-10, 2]. The (CER Character Error Rate) of speech recognition is 6.81%. Among them, the speech noise reduction front-end brings 0.3% to 0.5% reduction of CER.
2. Considering the performance of speech recognition inference, the TDNN network acceleration engine is designed and implemented based on the embedded heterogeneous computing environment (domestic FT-2000A CPU and domestic 690T FPGA), and its inference performance is 3.07 times faster than the OpenBLAS library when tested on the TDNN-13-SVD-M-Denoise model.
3. Considering the requirements of practical applications, the embedded speech

recognition API based on C++ is optimized and implemented, which provides both speech recognition function and non-fixed keyword retrieval function. The DNN network of acoustic model can be accelerated by OpenBLAS and TDNN acceleration engine, which is configured by CMake Tool. Finally, 6-8s speech data is used to test the inference interface of the API. The speech recognition time accelerated by OpenBLAS is about 1.2s, the speech recognition time accelerated by TDNN engine is about 750ms. At the same time, the CER of the API's speech recognition interface is 7.13%.

4. An off-line, near-field embedded speech recognition system is designed and implemented based on API, which can filter silent audio and pure background noise in the recording, and can segment sentences according to the pauses in the speech. The operating power consumption of the system is only 5-6W, and it can support long-term use after being equipped with a battery.

To sum up, the embedded speech recognition system designed in this thesis has the characteristics of anti-noise, real-time and low power consumption.

Keywords: Embedded Speech Recognition, Noise Reduction of Speech, Lightweight Network, Field Programmable Gate Array(FPGA)

目 录

第一章 绪论

1.1 研究工作的背景与意义

语音识别是人工智能领域的一个重要分支，在人机自然交互技术中起到关键作用，配合键盘、鼠标和触摸屏等其它人机交互方式协同工作。近年来，结合语音合成、自然语言理解等其他技术，语音识别技术被普及到各种应用场景，其中包括智能汽车、智能家居、智慧生活与办公、智能手机虚拟助手等等，采用语音进行交互更为便捷，学习成本更低，给人们的生活和工作带来极大的便利。

语音识别从最初的基于模板匹配的方法，发展为以隐马尔可夫模型（hidden Markov model，HMM）为基础的 GMM-HMM（Gaussian mixture model-hidden Markov model）模型框架，在此基础上又引入深度学习[1]，使用 DNN 替换了声学模中的 GMM，发展成为 DNN-HMM 框架，最后发展为完全以深度学习建模的端到端框架。目前，在大型商用数据集的训练下，基于深度学习的语音识别算法在安静环境、标准口音、大词汇量的场景下的识别正确率已经超过了 98%，精度接近饱和，但是在大多数应用场景中，语音识别仍然存在挑战。首先以深度学习为基础的语音识别算法严重依赖于数据驱动，算法在数据体量更大的训练集上训练表现更优，然而现有的开源数据集相比大型的商用数据集而言体量差别较大，成为了限制模型精度的一个原因。其次，安静的理想环境条件极难得到满足，在噪声场景尤其是高噪声场景下，语音信息的分布易受到噪声频谱的干扰，识别的正确率显著下降。提升语音识别的抗噪能力，同时应对稳态噪声和非稳态噪声的干扰，可以使算法的泛化性能大大提升，从而应用到更复杂的场景之中。因此，面向噪声环境的语音识别技术的深入研究具有重大的意义。

语音识别技术应用到社会的各个领域，除了算法的支撑，还需要硬件技术不断更迭。在嵌入式领域，语音识别落地的硬件平台也在不断地改变，每一次硬件平台的迭代，都体现了设计方式的重大改变。对于一些远端离线应用场景，例如航空航天卫星、全天候野外侦察车等，对能源的使用要求极为严苛，因此对于低功耗的语音识别方案有着较大的需求。对于现有的云端语音识别方案，语音识别算法模型部署在大量的服务器集群上，核心运算在图形处理器上（Graphics Processing Unit，GPU）上加速，其功耗无疑是巨大的。相比于 GPU，专用集成电路（Application Specific Integrated Circuit，ASIC）以及现场可编程门阵列属于运行功耗较低的硬件平台，相比 ASIC，FPGA 的额外的优势是可重构，开发更为灵活，因此可选用 FPGA 作为语音识别算法核心计算的加速硬件。现有的 FPGA 大

多集成了 DSP 芯片，该芯片对于乘累加运算做了特殊优化，使得 FPGA 计算能力大大提升，但是 FPGA 的存储访问尤其是外存访问代价较高，相比于计算更容易成为系统延时性能的瓶颈，因此在设计 FPGA 加速引擎时，需要对 I/O 访存和计算同时进行建模，避免在计算资源利用率不高的情况下，存储访问“爆炸”。FPGA 适合密集性的流式计算，却不适合控制流程较多的操作，因此可以将语音识别系统中非计算密集型的部分放在嵌入式 CPU 上运行，将消耗计算资源的部分在 FPGA 上集中加速。所以，研究基于 CPU 和 FPGA 的异构嵌入式平台下的语音识别解决方案，对于离线低功耗的场景有较大意义。

在异构嵌入式平台上，有限的计算资源、内存限制以及异构平台间的传输带宽限制都会成为语音识别落地的挑战，为避免语音识别系统的高响应延迟，需要严格考虑网络的体量，关注系统运行过程中的性能和内存占用，不能为了追求高精度的网络结构而不断地加深网络。其次，需要适当地考虑模型轻量化策略，包括模型的权重压缩和更低数据精度的运算，可以适当地牺牲语音识别正确率来获得更高的推理性能，在语音识别的正确率和推理速度上寻找一个合适的折中点。

综上所述，关于面向噪声的嵌入式语音识别的系统设计主要有以下几大难点：

1）基于深度学习的算法严重依赖数据驱动，需要足够的开源数据集支撑训练。

2）语音识别的精度受环境噪声的影响显著下降，真实场景下的噪声往往是稳态噪声和非稳态噪声的叠加，难以完全消除，且要考虑噪声处理给语音本身带来的失真影响。

3）在嵌入式异构环境下，需要控制网络模型的大小和计算量，不能一味的追求网络精度而忽略模型的内存占用、计算代价和数据传输代价，需要找到精度和推理速度的折中点。

4）基于 FPGA 设计语音识别核心计算的加速引擎时，需要充分考虑 FPGA 的计算特性和 I/O 代价，这往往是设计加速引擎的难点。

1.2 国内外研究历史与现状

1.2.1 语音识别技术研究历史与现状

自动语音识别（Automatic Speech Recognition）简称语音识别，狭义上理解是一种将人的声音转换为文本的技术。从上世纪 50 年代开始至今，已有近 70 年的发展历史，国外语音识别的研究历程可大致分为 3 个主要阶段：探索阶段、HMM-GMM 阶段、深度学习阶段。

1）探索阶段

20 世纪 50 年代到 80 年代，是小型语音识别系统的发展阶段，也是语音识别技术的重要探索阶段。这一阶段的语音识别技术仅支持小词汇量识别和孤立词识别，语音识别的算法主要采用了基于模板匹配的思想，即使用训练模板匹配待识别的语音特征。世界上第一个语音识别系统 Audry 诞生于 1952 年，主要用于识别 10 个英文数字[2]，。其后几年，普林斯顿的 RCA（Radio Corporation of America）实验室以及麻省理工的林肯实验室先后研发了针对元音和辅音音素的识别器[3]，显著提高了音素的识别率。20 世纪 60 年代，更多的实验室加入了语音识别的研究行列，产生了三项标志性的成就。首先是由前苏联的 Vintsyuk 提出的利用动态规划解决语音输入输出不定长的技术，该技术后来被称为动态时间规整（Dynamic Time Warping，DTW）[4]，后来 Sakoe 等人在 Vintsyuk 的研究基础上成功将非定长语音在时间轴上对齐[5]。其次是由 RCA 实验室提出的归一化打分机制，大大减少了语音长度对于识别得分的影响。最后一项则是由卡内基梅隆大学提出的音素动态跟踪方法，标志着孤立词语音识别向连续语音识别的发展迈进。20 世纪 70 到 80 年代，俄国的 Velichko 和 Zagoruyko 将模式识别（Pattern Recognition，PR）技术引入语音识别[6]、美国的 Itakura 将线性预测编码技术引入语音识别，Linda 团队提出了基于矢量量化的码本生成方法[7]，都推动了语音识别的发展，同时，Bell 实验室等研究机构开始关注大词表连续语音识别技术的研究，并且重点关注研究说话人无关的语音识别技术。

2）HMM-GMM 阶段

20 世纪 80 年代语音识别技术的核心思想由模板匹配转变为统计模型，其中标志性的成果为隐马尔可夫模型 HMM 的提出和普及[8][9]，同时，伴随着统计语言模型技术的出现，两项技术将语音识别的研究推向了一个高峰，李开复等人研发了 SPHINX 系统[10]，该系统以 HMM 和高斯混合模型 GMM 建模，是世界上第一个高性能的非特定人、大词汇量的连续语音识别系统。20 世纪 90 年代以后，一直到深度学习热潮出现以前，GMM-HMM 框架在语音识别领域一直占据着统治地位，许多研究机构先后推出了自己的语音识别系统，其中又以剑桥大学推出的开源语音识别工具包 HTK（hidden Markov model toolkit）的使用最为广泛[11]。

3）深度学习阶段

21 世纪开始至今，得益于深度学习技术的发展，DNN 替代 GMM 完成语音信号的观察概率的建模，大词表连续语音识别技术由传统的 GMM-HMM 框架演变为 DNN-HMM 框架。2006 年，Hinton 第一次提出了"深度置信网络"的概念[12][13]，模型通过预训练得到近似解，然后通过微调的方式得到更优解。2009 年 Mohamed

等提出了 HMM 和深度置信网络结合的声学模型[14]，在小词汇量连续语音识别中取得较好效果，2012 年 Hinton 等将深度神经网络和 HMM 相结合，在大词汇量连续语音识别中取得较好的效果[15]，在这之后，深度学习在语音识别中的研究如火如荼，卷积神经网络（convolutional neural network，CNN）、时延神经网络（time delay neural network，TDNN）、循环神经网络（recurrent neural network，RNN）及其变体等各种类型的神经网络层被用于构建 DNN，均取得不错的效果。近几年，端到端的语音识别研究也取得显著成效，其与 DNN-HMM 框架的不同点在于直接将输入音频序列映射到单词或其它字符序列。端到端的模型主要包括了基于连接性时序分类（Connectionist Temporal Classification，CTC）的模型和基于注意力机制的模型，2015 年 Li 等人提出了基于 CTC 的端到端模型[16]，网络主要由 LSTM 层进行构建。其主要解决了数据对齐和直接输出目标转录的问题。Chorowski 等人将注意力机制融入到端到端语音识别[17]，解决了 CTC 遗留的输出序列与输入音频在时间上不对齐的问题。2017 年，Google 公司联合多伦多大学提出了 Transformer 结构[18]，充分利用了注意力机制。又有其它研究人员将 Transformer 应用到语音识别领域[19]，取得了标志性的成果。总之，基于深度学习的语音识别算法在许多应用场景下的语音识别性能已经超过了传统的 GMM-HMM 框架，仍然当下的研究热点，在工业界 DNN-HMM 框架已经比较成熟，有较多优秀的开源项目，例如 ESPnet[20]、Kaldi[21]、Pytorch-Kaldi 等[22]，端到端语音识别的成熟、落地和广泛应用仍有一段路要走。

国内的语音识别研究也不断在发展和进步，最早源自中国科学院声学研究所对汉语语音信号的系统研究，在 20 世纪 80 年代以后，我国先后推出了“863”计划和“973”计划，吸引了国内众多高校和研究所投身语音识别研究，这期间最突出的成就是清华大学提出的基于段长分布的 HMM 语音识别模型[23]。近几年，中文语音识别技术已经达到了国际水准，在开源数据集方面，清华大学在 2015 年推出了 THCHS-30 中文语音数据库[24]，希尔贝壳公司也先后开源了 Aishell 系列数据集[25][26]，对中文语音识别的研究起到较大的推进作用。在语音识别研究方面，百度在 2014 年推出 Deepspeech[1]，又在 2016 年推出 Deepspeech 2[27]。科大讯飞公司于 2018 年推出深度全序列卷积神经网络（deep full-sequence convolution neural networks，DFCNN）[28]。其后，阿里巴巴也推出了低帧率深度前馈记忆网络[29]，该模型在降低错误率的同时，获得了更高的解码速度。

1.2.2 面向噪声环境的语音识别技术研究历史与现状

当下，语音识别在理想条件下已经取得非常好的效果，但是在大多数实际场

景中，理想条件并不存在，多噪声源干扰、低信噪比噪声干扰、回声干扰等均会影响语音识别的精度。在复杂环境下的语音识别研究是当前的难点问题，其大致可分为 3 个方向：

1）在语音识别的前端进行语音降噪，目的在于尽可能消除语音信号中的噪声，通常以麦克风阵列的形式，配合声源定位[30]、语音增强[31]、回声消除[32]等技术，提高语音质量，例如基于传统信号处理的维纳滤波算法[33]，以及基于深度学习生成式对抗网络的 SEGAN 等[34]，语音降噪由于作为语音识别的预处理机制，所以算法模型体量要足够小，其推理性能也需要满足实时性。

2）语音识别模型自身的抗噪性，某些网络结构对噪声有较强的适应能力，例如基于深度学习的模型，将含噪声语音数据集作为训练语料训练这类模型，可以使得语音识别具备一定抗噪能力。上海交通大学在 2016 年提出的非常深卷积神经网络（very deep convolutional neural networks，VDCNN）和 2018 年提出的非常深卷积残差网络（very deep convolutional residual networks，VDCRN）提升了语音识别在噪声环境下的鲁棒性[35][36]。2017 百度基于 LSTM-CTC（long short-term memory-Connectionist Temporal Classification）提出 Code Fusion[37]，在多种噪声环境下中英文语音识别的表现优异。

3）利用多模态信息做数据融合，除了利用听觉信息，还结合视觉信息，比如说话时的唇形特征等，同时利用注意力机制进行信息融合，以更丰富的特征信息来提升语音识别的鲁棒性。这个方向的代表性成果有 Zhou 等人提出的以注意力机制融合多模态信息的方法[38]，以及 Makino 等人提出的提取视觉特征的方法 vision-to-phoneme（V2P）[39]，他们利用融合信息做语音识别均在噪声环境下取得了良好性能。

1.2.3 语音识别硬件平台研究历史与现状

最早语音识别算法部署在通用处理器上，难以满足对于大词汇量、实时性以及识别精度的需求，优化算法推理的时间主要有两种途径，一种是优化算法复杂度，但是这样往往导致识别精度的下降，第二种是选择更高性能的硬件平台，目前语音识别算法采用的硬件平台主要有 MCU、GPU、DSP、ASIC 以及 FPGA。

MCU 的计算能力较弱，语音识别算法通常有较多的浮点运算，难以满足精度和速度的要求，DSP 处理器目前已有较多的语音识别算法部署的案例[40][41][42]，其内部针对乘累加运算设计了专用优化电路，可以对特定的操作进行加速，但是 DSP 无法应对更复杂的算法逻辑。GPU 的处理性能很强，很容易满足语音识别算法的实时性要求，但是其功耗过高，只适合用来构建大型的云端语音识别服务器，

并不适合离线场景或者移动端设备。将语音识别算法基于 ASIC 芯片实现是目前较好的一种落地方式，其针对语音识别算法的特点进行专用硬件设计，能获得很好的加速效果，且功耗很低，唯一的缺点是一旦 ASIC 芯片的硬件设计确定了就无法改变，不具备可重构的能力。FPGA 很好的弥补了 ASIC 这一缺陷，同时具有灵活性和可重配置的特点，新一代的 FPGA 已经将 DSP 集成为软核，计算性能获得大幅度提升，适合语音识别算法的落地。

基于 FPGA 的语音识别方案的设计不仅仅充分考虑到并行性的挖掘，更要保证高吞吐量和实时性的要求，然而 FPGA 的空间资源十分有限，这大大限制了设计方案的多样性。在设计过程中，除了要考虑计算资源的利用，还要 I/O 访存的优化问题，计算速度可以通过利用的更多的 DSP 来提升，但是存储资源往往在 DSP 利用率不高的情况下就已经消耗殆尽。因此，基于 FPGA 的设计往往是一个不断优化的过程，如何在时间性能和空间性能上做折中和协调，找到良好的平衡点，是设计的关键。目前基于 FPGA 平台的语音识别系统的研究已经有较多成果，2009 年 Amudha 团队研发了基于 HMM 模型的软硬协同的语音识别系统[43]，2016 年 Van-lan Dao 等将 MFCC 特征提取的过程落地到 FPGA 上，同年 Lee M 等将基于 RNN 的语音识别模型部署到 FPGA 上[44]。

1.3 本文的主要贡献与创新

本文主要面向噪声环境研究了适用于嵌入式设备的语音识别方案，同时基于 CPU 和 FPGA 嵌入式异构环境设计、实现和封装了可直接调用的 C++嵌入式语音识别应用程序编程接口（Application Programming Interface，API），并基于 API 搭建了一个小型的离线、近场的嵌入式语音识别系统，在目标噪声环境下测试满足实时性要求，本文在算法和工程方面主要贡献如下：

1）考虑到稳态噪声和非稳态噪声对语音识别的影响，设计研发了基于 GRU 和传统信号处理的语音降噪前端，该算法不仅提供语音降噪的功能，还提供面向噪声环境的活动语音检测功能，基于该功能提出了一种语音断句的方法。使用该语音降噪算法作为语音识别的前端，并使用该语音降噪前端处理后的数据微调干净语音训练得到的预训练模型，语音识别的字错率下降了 0.3-0.5%。

2）基于 Kaldi DNN-HMM 框架分别设计和训练了 TDNN-9、TDNN-11、TDNN-13 作为语音识别声学模型的 DNN，考虑到嵌入式平台部署，采用 SVD 技术以不同的分解力度分解 TDNN 模型权重，在牺牲了微小精度的前提下，分解后模型尺寸是基线模型的 47%-65%，经过大量的精度对比实验，最终选用 TDNN-13-SVD-M-Denoise 作为声学模型的 DNN，结合语言模型 Trigram 和语音降噪前端

进行测试，在 SNR 为[-10,2]的特定噪声环境下语音识别字错率为 6.81%

3）根据 TDNN 的运算特性（底层看作是多次顺序执行的矩阵乘法），采用 HLS 技术路线设计实现了基于 FPGA 的 GEMM IP 核，开发了基于异构环境的 TDNN 加速引擎，对比了 OpenBLAS 加速库和 TDNN 加速引擎对 TDNN-13-SVD-M-Denoise 模型的加速性能，经实验测试，TDNN 引擎加速模型的推理速度是 OpenBLAS 在 FT-2000A CPU 上推理速度的 3.07 倍。

4）考虑到实际应用的需求，封装实现了 C++嵌入式语音识别 API，该 API 既提供了语音识别功能，也提供了非固定语音关键词检索功能，声学模型 DNN 的部分既可以选择 OpenBLAS 在 CPU 上加速，也可以选择 TDNN 引擎在 FPGA 上加速，可通过 CMake 灵活配置，最后在异构环境下基于 TDNN-13-SVD-M-Denoise-Trigram 模型测试 API 推理性能，测试采用 6-8s 的语音，基于 OpenBLAS 加速的解码时间为 1.2s 左右，基于 TDNN 引擎加速的解码时间为 750ms 左右，语音识别 API 在嵌入式平台满足实时性要求。

1.4 本论文的结构安排

结合本文研究的实际调研、设计和实现过程，本文的章节结构安排如下：

第一章绪论。说明了本文研究工作的背景和意义，梳理了语音识别、噪声环境语音识别以及语音识别硬件平台的发展历史和研究现状，同时概括了本文的贡献与成果。

第二章理论基础及相关技术概述。首先，介绍了本文搭建网络结构用到的深度学习基本算法的原理，其次，简单介绍了语音识别以及 DNN-HMM 框架的原理，然后介绍了通用矩阵乘法 GEMM，这是设计 FPGA 加速引擎的基础，然后又介绍了本文用于压缩模型的奇异值分解（singular value decomposition，SVD）技术的原理，最后详细介绍了基于 HLS（High Level Synthesis）技术路线设计 FPGA IP 核的基本步骤和优化指令。

第三章面向噪声的轻量化语音识别网络设计，首先介绍了语音降噪前端的设计和实验分析，然后分点介绍了轻量化语音识别模型的设计过程，包括算子选型、网络结构的设计、模型的轻量化等等，最后详细介绍了模型的训练过程和对比实验，选出综合性能最佳的语音识别模型。

第四章基于异构环境的 TDNN 加速引擎设计。首先介绍了引擎的架构，然后基于 FPGA 设计了 GEMM IP 核，详细介绍设计的存储方案、计算方案和分块策略，然后简要介绍了 IP 核的实现架构和接口，最后以第三章设计的轻量化的 TDNN 模型测试引擎的推理性能。

第五章面向噪声的嵌入式语音识别系统实现及测试。首先介绍了基于 Kaldi、OpenBLAS 以及 TDNN 引擎开发的嵌入式语音识别 API 及其优化过程，然后阐述了嵌入式语音识别系统的设计实现过程，包括录音模块、断句模块的设计实现等，最后针对 API 接口和嵌入式语音识别系统做了大量测试和实验分析。

第六章总结与展望。对本文的研究工作的成果和不足进行了总结，并且对后续研究工作进行分析和展望。

第二章 理论基础及相关技术概述

语音识别是人工智能的一个重要分支，经过近 70 年的探索和发展，开始广泛应用到各个领域，苹果手机的 Siri 系统、百度地图的语音导航系统、微信 APP 的语音转文字接口等，都为生活带来了极大便利。语音识别系统的普及，深度学习技术起到了关键的推进作用。首先，本文将基于深度学习对语音识别 DNN-HMM 框架进行研究，同时研究了语音识别算法对于特定噪声环境的适应性。其次，深度学习算法相比传统机器学习算法而言需要更多的算力支持，在计算资源、存储资源受限的嵌入式平台部署深度学习算法，不仅需要设计轻量化的模型，还需要充分利用硬件资源对算法加速，保证模型推理的实时性。本章节将分别对本文研究涉及的理论基础和技术原理进行简单的介绍。

2.1 深度学习算法原理

在深度学习蓬勃发展的十几年内，许多经典的网络结构被用于处理各种各样复杂的任务，卷积神经网络因其强大的特征提取能力被广泛用于计算机视觉（Computer Vision，CV）领域，循环神经网络因其对时间序列的敏感性而被广泛用于处理和视频、音频、文字相关的任务。时延神经网络和卷积神经网络具有类似原理，同时也是卷积神经网络的前身，适合用于处理语音信号。本文的语音降噪算法和语音识别算法主要用到了循环神经网络和时延神经网络，下面对这两个网络结构的原理做基本介绍。

2.1.1 循环神经网络

循环神经网络在自然语言处理领域取得巨大成功，它不是特指某一种网络，而是指一类以序列数据为输入，在序列的演进方向进行递归且所有节点按链式连接的递归神经网络。下图是典型的传统 RNN 网络结构图，其特点是：网络隐藏层的神经元之间相互连接，并且隐藏层的计算依赖于输入层的结果以及上一时刻隐藏层的结果，在实践过程中，为了简化 RNN，假设当前状态只与前面几个状态有关系，由此带来的弊端是：如果序列很长，较早时刻的信息无法传递到后面的时刻，其次，在反向转播过程中，RNN 梯度会随着时间的推进逐渐收缩，最后梯度消失，可能会丢失在长序列中学习到的信息，形成短期记忆的问题。

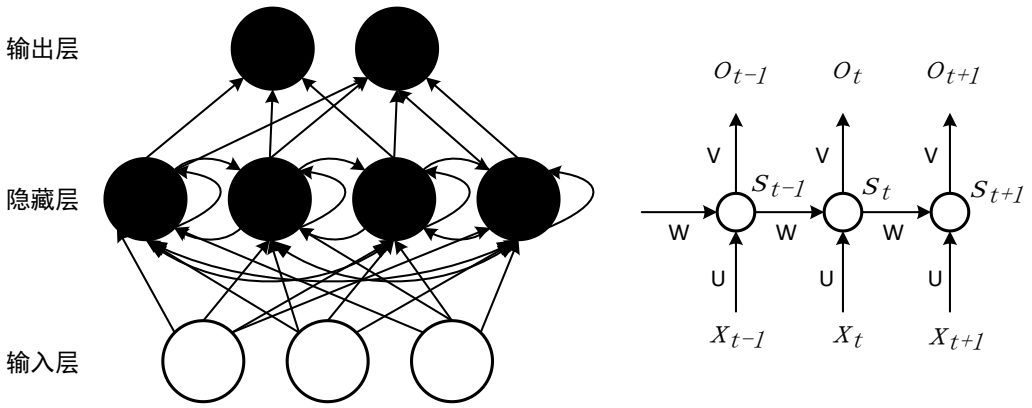

图 2-1 循环神经网络的结构

RNN 受短期记忆的影响，容易遗漏前面的重要信息，于是一些基于 RNN 改进的网络结构应运而生，其中比较经典的是长短时记忆网络和门控循环单元[45]，如下图所示：

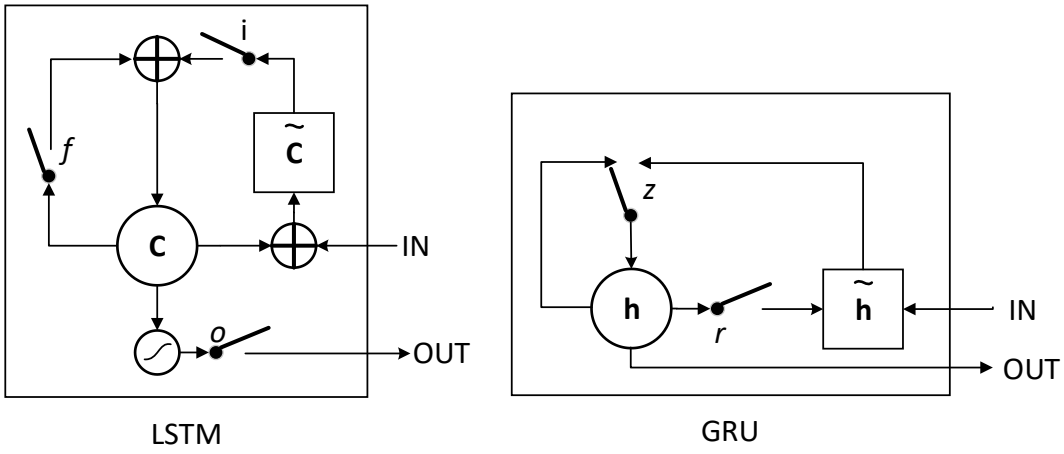

图 2-2 LSTM 和 GRU 结构

结构内部采用门（Gate）机制，这些门可以决定序列中哪些数据被留存，哪些数据被丢弃。其中，LSTM 由单元状态构成，单元状态指多种不同的门结构（输入门、输出门、遗忘门），它们可以在序列的处理过程中利用较早时刻的信息。相比 LSTM，GRU 结构更加简单，仅仅使用更新门和重置门两种门结构，更新门决定添加和丢弃哪些信息，而重置门决定需要忘记过去多少信息，其具有更少的张量操作，在训练过程中能节省更多的时间。本文设计的语音降噪算法就使用了 GRU 作为基本结构，模型尺寸和运算量都比较小，即便在嵌入式平台上部署也不会消耗太多的计算资源和内存资源。

2.1.2 时延神经网络

TDNN 在 1989 年由 Waibei A 等人提出[46]，最早用来解决音素识别的问题。对于全连接神经网络而言，输入层和隐藏层的神经元一一连接，神经元只能学习到当前帧的特征信息，然而对于序列如果能利用上下文的信息，模型的建模能力会更强。因此 TDNN 考虑连续多帧特征的学习，这样能表达语音特征在时间上的关系，从而从短时特征中学习长时动态上下文，下图是一个 5 层 TDNN 的模型：

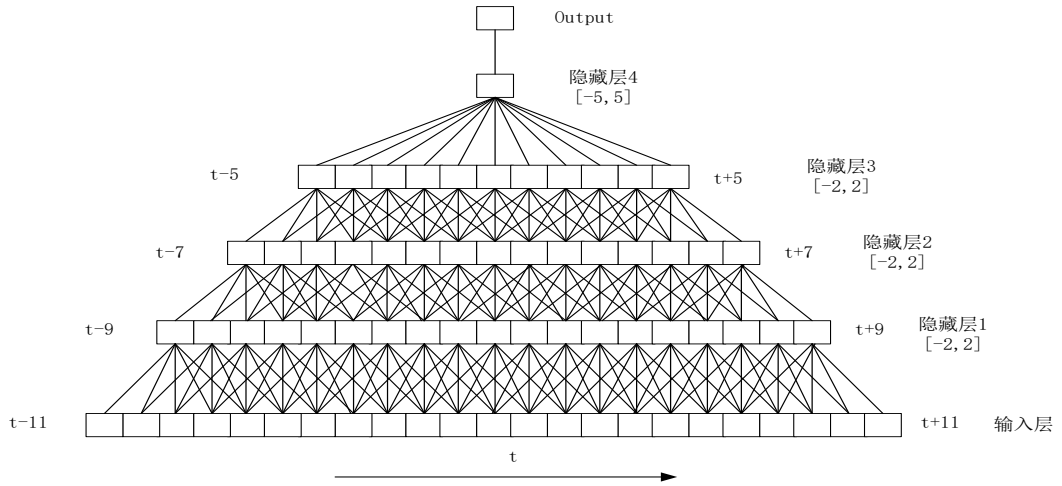

图 2-3 TDNN 网络结构

TDNN 也属于前馈神经网络，其采用了和循环神经网络不同的方式来学习序列信息，即将时序输入截成长度规整的小段，相邻若干小段拼接在一起形成新的输入，这种新的输入长度一致，且包含了相邻样本之间的依赖关系，具有短时记忆功能。例如，隐藏层 4 中的[5,-5]代表以当前帧为基准，前后各五帧共十一帧作为网络的输入隐藏层都可以以较大的视野来建模语音信号，对于更深的网络层，其学习和观察的上下文的视野更大，能够学习到更多输入层特征的信息，因此 TDNN 虽然不是时序神经网络，却可以对时序信号进行很好的建模。

2.2 语音识别基本原理

本文主要基于 DNN-HMM 的研究语音识别，得益于深度学习技术的发展，DNN-HMM 由传统的 GMM-HMM 框架演变得到，为语音识别的性能带来大幅度提升。本节先对语音识别的基础做简要介绍，然后着重阐述 DNN-HMM 的基本原理以及相比 GMM-HMM 的优势，最后介绍 DNN-HMM 的训练方法。

2.2.1 传统语音识别基础概述

如公式（2-1）所示，传统的语音识别主要基于统计学的思想，针对一个训练语句 O，利用贝叶斯公式来计算最佳的词序列 W^*，即从所有可能产生特征向量 O 的文本序列中找到概率最大的 W^*：

$$W^* = argmax_W P(W|O) = argmax_W \frac{P(O|W)P(W)}{P(O)}$$
$$\propto argmax_W P(O|W)P(W) \quad (2\text{-}1)$$

其中，$O=[o_1,o_2,\ldots,o_N]$,表示一段语音经过特征提取得到的特征向量，o_i 表示某一帧的特征向量，$W=[w_1,w_2,\ldots,w_M]$表示该段语音对应的文本序列，w_i 表示文本序列的一个基本的组成单元，如音素、字符等。P(O)是已知固定的，要找到最佳词序列，则需要使 P(O|W)和 P(W)的乘积最大，P(W)和 P(O|W)的建模则分别代表语言模型和声学模型。

相比于端到端的语音识别，传统语音识别的特点是包含多个模块，各个子模块分开训练，最后协同完成语音识别任务。一个完整语音识别系统的构成如下：

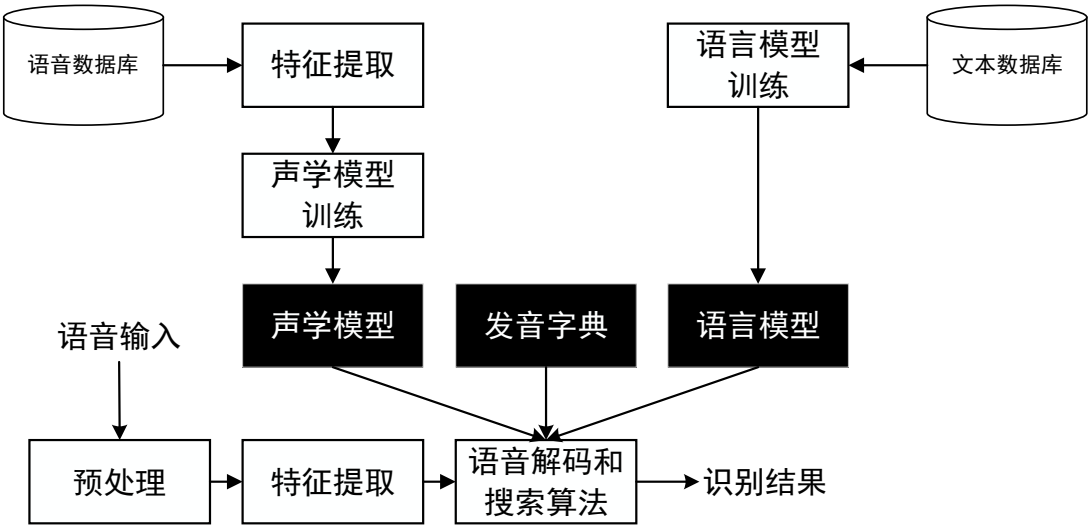

图 2-4 语音识别系统的构成

一般来说，传统语音识别的主要流程分为 3 步：特征提取、语言模型和声学模型的训练和准备、利用语言模型、声学模型、发音字典以及解码和搜索算法进行识别。

1）特征提取

在语音识别之前，需要对语音信号进行特征提取，作为训练模型的输入。目前识别中最常用的两种声学特征是梅尔频率倒谱系数（即 MFCC 特征）和梅尔滤波器组系数（即 FBank 特征）。以 MFCC 为例，特征提取的过程如下：

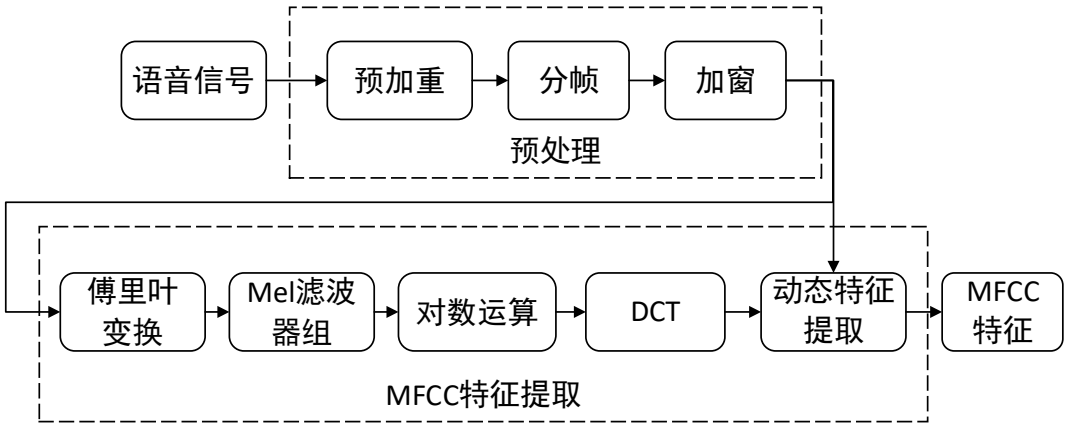

图 2-5 MFCC 特征提取流程

MFCC 提取过程包括信号预处理、FFT、Mel 滤波器组、对数运算、离散余弦变换、动态特征提取等步骤。其提取过程模仿了人耳的听觉过程，计算复杂性低，在噪声环境也具有一定的鲁棒性。

2）声学模型

声学模型的目的是在语音特征和音素之间建立联系，声学模型的构建主要面临两个问题：第一是特征向量序列的可变长性，即不同长度的语音经过特征提取之后得到不同长度的特征序列，学术上主要通过动态时间弯曲技术（DTW）和隐马尔可夫模型（HMM）解决了这个问题，第二是音频信号的丰富多样性，包括说话人口音、环境噪声干扰等因素，因此声学模型的建模能力要足够强，以适应多种因素干扰带来的影响。

3）语言模型

语言模型用于计算一个句子的概率，即计算公式（2-1）中 W 序列的概率，$W=[w_1,w_2,\ldots,w_M]$，见式（2-2）：

$$P(W) = P(w_1, w_2, \ldots, w_M)$$
$$= P(w_1)P(w_2|w_1) \ldots P(w_M|w_1, w_2, \ldots, w_{M-1}) \quad (2\text{-}2)$$

这样存在两个较大的问题：第一是 $P(W_M|W_1,W_2,\ldots, W_{M-1})$的可能性较多，无法估计，第二是在语料库容量较小时，很可能并不包含某些词对的组合。为了解决了这些问题，提出了马尔可夫假设，即某一个词出现的可能性只与前面的有限个词相关，基于该假设又提出了语言模型 N-Gram，N 表示一个词只与前 N-1 个词有关系，当N为1时被称为一元语言模型（unigram），还有二元语言模型（bigram）和三元语言模型（trigram），实践中应用较多的是 bigram 和 trigram，高于四元的语言模型需要庞大的语料支撑，且仍然避免不了上述第二个问题，因此应用较少。

4）发音字典

发音字典，中文中就是拼音与汉字的对应，英文中就是音标与单词的对应，其目的是根据声学模型识别出来的音素，来找到对应的汉字（词）或者单词，用来在声学模型和语言模型建立桥梁，将两者联系起来。

5）语音解码和搜索算法

解码算法的本质是在图网络中搜索一条最优路径，一般分为动态解码和静态解码，基于 Viterbi 算法的动态解码是最基础的解码方法，其将发音字典编译成状态网络，形成线性词典，线性词典的占用内存较大，后来使用树形词典构建网络，减少了内存占用，动态解码计算量较大，解码速度慢。后来提出了基于 WFST 的 Viterbi 静态解码方法，它把动态知识源提前编译好，将搜索空间全部展开，形成静态网络，在解码的时候直接调用，因此解码速度非常快，代价则是静态图会占用较大的内存空间。所以在嵌入式设备上构建语音识别系统，需要综合权衡资源占用和推理性能两个因素，合理地选择最适合的解码方法。

2.2.2 GMM-HMM 向 DNN-HMM 的演进

1）隐马尔可夫模型（HMM）

隐马尔可夫模型是基于马尔可夫模型提出的，马尔可夫模型中的状态都是可观测的，然而对应到现实生活中，并不是所有的事件都是可观测的，这在某种程度上限制了模型的适应性。于是在隐马尔可夫模型中引入隐状态的概念，隐状态序列不可观测，只知道状态转移的概率，该模型是一个双重随机过程，包括模型的状态转换和特定状态下可观测事件的随机。假设系统有 N 个状态 S_1，S_2，…,S_N，状态的转换与时间无关，系统的隐状态序列为 q，可观测序列为 o，如图：

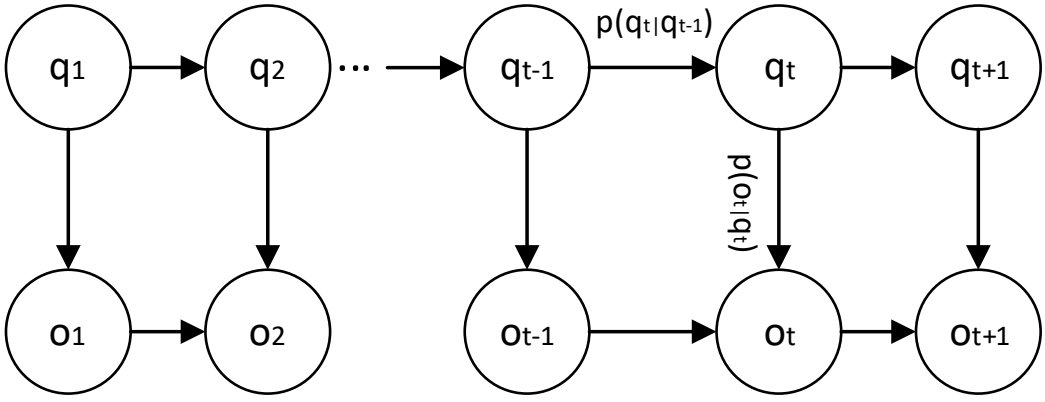

图 2-6 HMM 结构示意图

一般将 HMM 模型表示为$\lambda = [\pi, A, B]$，包括三要素：

• 初始状态概率 Π：表示模型在初始时刻各个状态出现的概率，通常记为

$\Pi=(\Pi_1,\Pi_2,\ldots,\Pi_N)$，表示模型的初始状态为 S_i 的概率。

• 状态转移概率 A：模型在各个状态间转换的概率，通常记为矩阵 $A=[a_{ij}]$,其中 a_{ij}，在任意时刻 t，若状态为 S_i，则在下一时刻状态为 S_j 的概率。

• 输出观测概率 B：模型根据当前状态获得各个观测值的概率通常记为矩阵 $B=[b_{ij}]$。其中，b_{ij} 表示在任意时刻 t，若状态为 Si，则观测值被获取的概率。

一般 HMM 用来解决三个基本问题：

• 评估：给定 HMM 模型，求得到某个观察序列的概率

• 学习：给定一个观察序列，得到一个 HMM 模型。

• 解码：给定 HMM 模型，以及观察序列，求得隐状态序列。

2）GMM-HMM 声学模型

在语音识别中，语音特征向量就是上面提到的可观测序列，HMM 模型对时序信息进行建模时，在给定 HMM 的一个状态后，GMM 对属于该状态的语音特征向量的概率分布进行建模，即使用混合高斯模型模拟特征，然后将均值和方差输入到 HMM 的模型中。基于 GMM-HMM 的语音识别流程为：

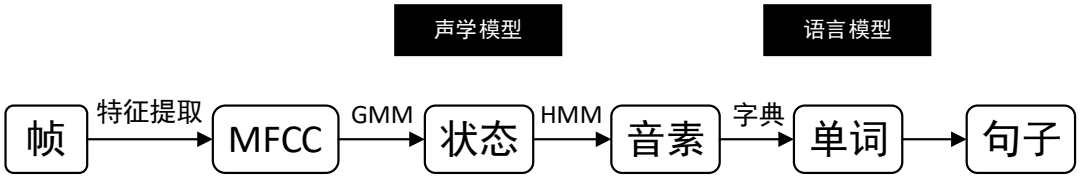

图 2-7 基于 GMM-HMM 的语音识别流程

在 GMM-HMM 中以音素为单位进行建模，对连续语音信号提取 MFCC 特征，将特征对应到状态这个最小单位，通过状态获得音素，音素再组合成单词，单词串起来变成句子。最早使用 GMM-HMM 对单音素进行建模，效果较差，后来考虑到音素的上下文环境，将音素的前一个以及后一个音素都考虑进来，形成基于 GMM-HMM 的三音素模型，三音素的模型基于单因素模型训练的结果，训练过程均采用无监督方法。

3）DNN-HMM 声学模型

传统的 GMM-HMM 建模能力有限，无法准确捕获语音特征内部的高阶相关性，识别率较低。而在 DNN-HMM 框架中，使用 DNN 代替 GMM，可以得出观测值在每个节点(状态)上的后验概率，其不需要假设声学特征的分布，直接对有相关性的语音特征建模，利用大量的数据特征信息进行有监督的学习。除此之外，DNN 可以输入连续的多帧语音特征向量，能更好的对语音信号的上下文进行建模。DNN 输出向量的维度对应了 HMM 中状态的个数，如下图所示：

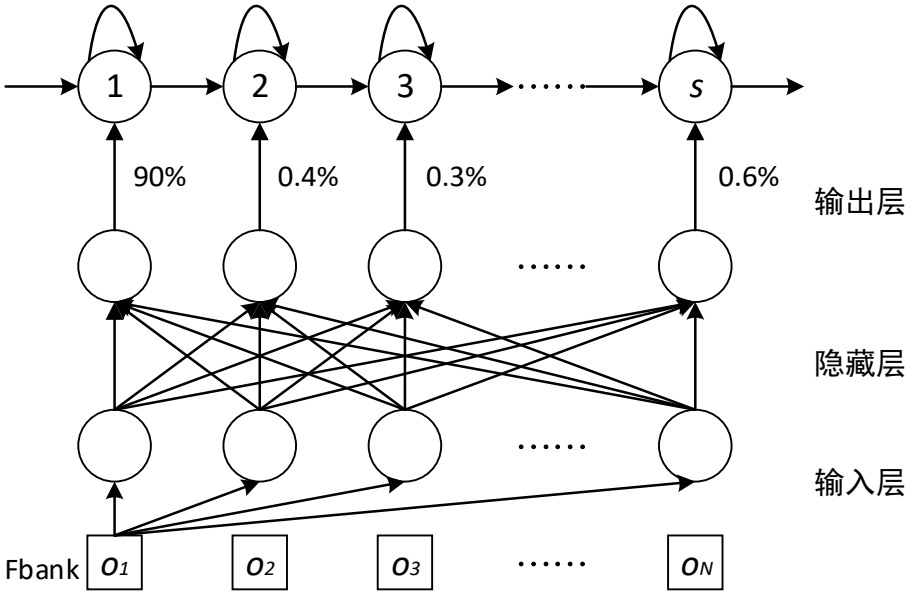

图 2-8 DNN-HMM 的构建

2.2.3 DNN-HMM 训练方法

本文基于 DNN-HMM 框架构建声学模型，整个训练过程基于 Kaldi 开源框架完成，一般的声学模型训练最大化式 2-1 中 P(O|W)的概率，称之为最大似然（Maximum Likelihood, ML）的声学模型训练，其训练流程如下：

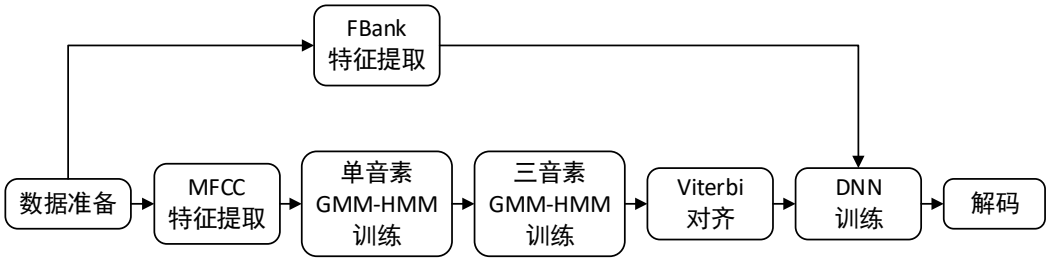

图 2-9 基于 ML 的声学模型训练流程

首先利用 MFCC 特征训练传统的 GMM-HMM，包括单音素的 GMM-HMM 训练和三音素的 GMM-HMM 训练，然后利用 Viterbi 算法强制对齐得到每一帧语音，得到 DNN 训练需要的标签，最后利用标签和 Fbank 特征训练 DNN。

在 ML 训练之后，又提出了直接最大化 P(W|O)的准则，称为最大互信息准则，这种基于 MMI 准则的声学模型训练方法叫做区分性（Discriminative Training，DT）训练，其训练依赖于 Lattice，Lattice 又依赖已经训练好的声学模型，所以在 DNN 的声学模型中，要先基于 ML 准则先训练好 DNN 模型，然后做 MMI 的训练，比较麻烦。Kaldi 的 chain model 训练基于 LF-MMI（Lattice Free-Maximux Mutual

Information），其引入了 N-Gram 语言模型的思想，无需再对训练语料进行解码，也无需生成 Lattice，具有更高的识别率的同时还加速了模型模型的训练和解码过程。本文的 DNN-HMM 的训练基于 chain model，训练流程如下：

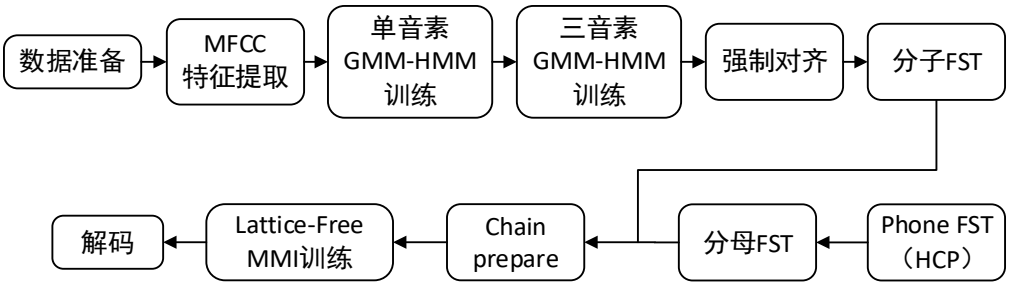

图 2-10 基于 chain model 的 LF-MMI 训练流程

2.2.4 语音识别性能评价指标

在 DNN-HMM 框架中，首先完成训练的是语言模型，一个合适的语言模型对语音识别的性能提升很大，通常可以将语言模型放到具体的语音识别任务中测试语言模型对于端到端语音识别性能的提升，以此判别语言模型的好坏，这样测试的缺陷在于，当测试数据集较大时，评估的总时间很长。因此需要有能单独评价语言模型性能的指标，对于 N-Gram 语言模型，通常采用困惑度（Perplexity）来评价语言模型的好坏。困惑度计算见式（2-3）：

$$\mathrm{PP(W)} = \mathrm{P}(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}} \tag{2-3}$$

由式（2-3）可知，概率和困惑度呈反比关系，因此困惑度越低就说明语言模型的性能更优。

除了语言模型本身的性能评价，还需要有能评估整个语音识别性能的评价指标，在英语、阿拉伯语的语音识别任务中，句子的最小单位是单词，所以通常采用词错率（Word Error Rate，WER）来评价预测文本和标签文本之间的差异，WER 越小说明语音识别的效果越好，和英文不同的是，中文句子的最小单位是汉字，所以通常采用字错率（Character Error Rate，CER）来评价语音识别的性能。WER 和 CER 的计算方式一致，以 CER 为例，计算方法见式（2-4）：

$$\mathrm{CER} = \frac{S+D+I}{N} = \frac{S+D+I}{S+D+C} \tag{2-4}$$

假设预测文本为 Text，标签文本为 Label，S 表示将 Text 转化为 Label 时替换的汉字的个数，D 表示将 Text 转化为 Label 时删除的汉字的个数，I 表示将 Text 转化为 Label 时添加的汉字的个数，N 表示 Label 中的总的汉字个数，C 表示 Text 相比 Label 而言识别正确的汉字的个数，因此 N=S+D+C。在 Label 已知的情况下，

N 可以直接统计得到，而 S+D+I 可以通过编辑距离（也称为莱文斯基距离）计算得到，计算方法见式（2-5），该公式可以通过动态规划的方法求解。

$$lev_{a,b}(x,y)=\begin{cases}\max(x,y) & if\ \min(x,y)=0\\ min\begin{cases}lev_{a,b}(x-1,y)+1\\ lev_{a,b}(x,y-1)+1\\ lev_{a,b}(x-1,y-1)+w_{ax\neq by}\end{cases} & otherwise\end{cases} \tag{2-5}$$

2.3 通用矩阵乘法 GEMM

通用矩阵乘法（General Matrix Multiplication，GEMM）在深度学习中十分重要，许多基础的神经网络层都可以转换为 GEMM 运算，其本质是两个输入矩阵 A 和 B 相乘得到一个输出矩阵 C，示意图如下：

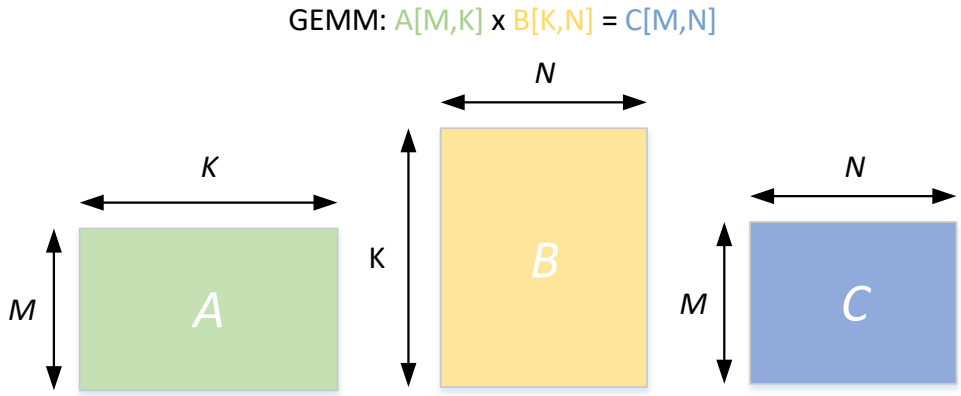

图 2-11 GEMM 示意图

并且目前几乎所有的主流神经网络训练和推理框架都针对硬件特性优化和实现了 GEMM 核心算子，GEMM 的朴素实现方法为：

```
for (int m = 0; m < M; m++) {
    for (int n = 0; n < N; n++) {
        for (int k = 0; k < K; k++) {
            C[m][n]+= A[m][k] * B[k][n];
        }
    }
}
```

图 2-12 GEMM 朴素实现方法

是一个典型的三层循环（循环边界分别为 M，N，K），该算法的总计算量（包括乘法和加法）为 2MNK FLOPs，内存访问总次数（包括读和写）为 4MNK，GEMM 的所有优化策略均是以该朴素实现为基础。在 CPU 端优化该算法时，可以通过提高访问内存时 Cache 的命中率、利用单指令多数据（SIMD）处理批量数据等方式来提升性能。

对于本文构建网络用到的 TDNN，在底层也可以看作是多次调用了 GEMM 运算，在第四章将详细阐述在 FPGA 上优化 TDNN 推理的 GEMM 加速引擎的设计。

2.4 奇异值分解技术

深度学习算法一般需要大量的运算支持，且其 I/O 的操作量也较大，在资源有限的嵌入式平台上部署面临许多挑战。于是许多模型轻量化的技术被提出，包括量化技术、剪枝技术、低秩分解技术等。奇异值分解（SVD）是低秩分解技术中的一种，其目的是取出一个矩阵中最重要的特征，它和特征值分解息息相关，特征值分解处理的矩阵只能是方阵，而奇异值分解可以处理任意大小的矩阵。其计算原理见式（2-6）：

$$A_{MXN} = U_{MXM}\Sigma_{MXN}V^T{}_{NXN} \tag{2-6}$$

假设 A 矩阵的维度为 MxN，那么 U 矩阵的维度为 MxM（U 中的向量全部是正交的，称为左奇异向量），Σ 是一个 MxN 的矩阵（除了主对角线上的元素，其它元素都为 0，主对角线上的元素称为奇异值，按大小排好序），V^T 矩阵的维度为 NxN（V^T 中的向量也全部是正交向量，称为右奇异向量）。Σ 矩阵中对角线上的奇异值按照降序的方式排列，前面的很少一部分奇异值的能量就占了所有奇异值能量的 90%以上，所以，可以去掉后面较小的奇异值以及它们所对应的 U、V 方阵中的奇异向量，仅用前 r 个奇异值来代表所有的奇异值。式（2-6）转变为式（2-7）：

$$A_{MXN} = U_{MXr}\Sigma_{rXr}V^T{}_{rXN} \tag{2-7}$$

式子中 r 的大小远远小于 M 和 N，这意味着 A 矩阵由三个更小的矩阵的乘积来表示，大大节省了内存占用。

在 Kaldi 中，SVD 技术可以用来对 TDNN 进行分解，其将每个权重矩阵因子分解为两个更小的因子的乘积，并且其中一个子矩阵是半正定的，例如：一个 512x1280 的矩阵经过 SVD 之后被分解为 512x88 和 88x1280 两个矩阵，参数量减少了近 76%。TDNN 经过分解后，主要节省了内存的占用，同时能获得一定的速度增益，唯一的缺陷是，分解后得到矩阵解释性不强，语音识别的效果大大降低，

因此在 SVD 之后，往往还需要对模型重新训练，提升新模型对数据的适应能力。

2.5 HLS 工具及相关优化指令概述

目前国外的 FPGA 解决方案提供商有两大巨头，Xilinx 和 Altera，它们分别提供了 Vivado 系列和 Quartus 系列开发平台，本文的研发工作基于前者，Vivado HLS 工具是 Vivado 系列工具中的一种，它为软件工程师使用软件开发语言在 FPGA 上开发程序提供了一种可行且高效的途径，即按照 HLS 的规则使用 C/C++代码重构和实现算法，在算法需要优化的地方添加相应的 HLS 优化指令，然后通过 HLS 内部的高级综合编译工具将 C/C++代码转换为硬件描述语言。HLS 内部有完整的仿真工具来验证 C/C++层面算法的正确性以及硬件描述语言层面的正确性，且提供了封装完整 IP 核的接口，因此研究人员可以在较短的时间内实现和验证基于 FPGA 的算法优化方案。当然，该方案也存在弊端，高级综合工具编译器的行为不是完全可控的，高性能的设计方案也依赖于编译器本身的优化能力，对于资深的硬件开发工程师而言，往往直接采用硬件描述语言实现方案可以获得更佳的性能。下面详细介绍 HLS 的开发流程以及算法性能优化采用的 HLS 优化指令。

2.5.1 HLS 工具的开发流程

利用 HLS 工具设计开发时，目录中一般包含两类文件：一种是设计文件，由多个头文件和 cpp 文件构成，还有一种是仿真测试文件 Test Bench，其主要用来验证算法优化设计之后在 C/C++层面功能的正确性。

HLS 开发的流程主要包括了 4 个阶段，如图 2-13：

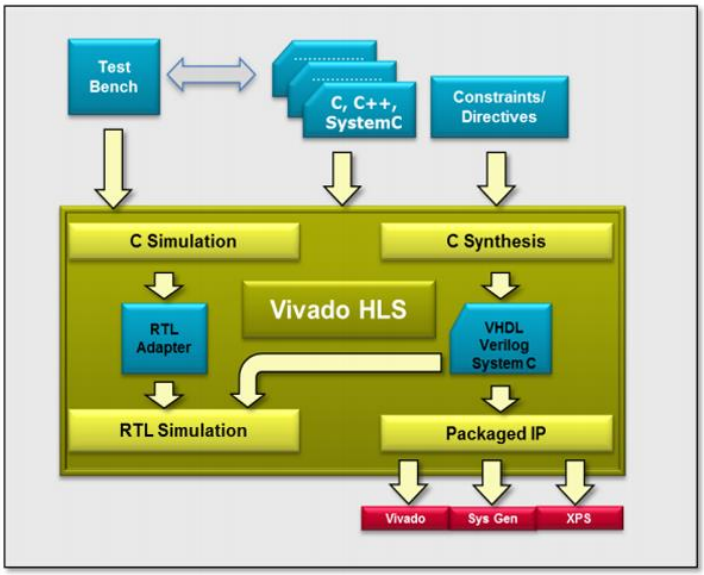

图 2-13 HLS 开发流程图

1）第一阶段是 C Simulation，用于验证优化之后的算法的顶层函数接口的功能是否正确，为此，需要提前准备好算法的输入数据和正确的输出结果，在 Test Bench 的 main 函数中编写软件测试代码，将函数接口执行的结果与正确的输出结果一一比对，其误差在一定范围内则说明功能逻辑正确。在本文中，输入的数据指 GEMM 的输入矩阵 A 和 B，输出结果指 GEMM 的运算结果 C。

2）第二阶段是 C Synthesis，C Simulation 仿真正确后，就可以对优化后的算法进行综合，编译器在这一阶段解析优化指令，理解设计人员的优化意图，最后的输出包括转化后的硬件描述语言（Verilog 或 VHDL）以及对应的综合报告，综合报告中有当前方案的时钟周期、资源消耗以及性能评估等信息，设计人员查看分析报告后可以进一步迭代优化或对方案进行改进，最终得到比较理想的设计方案。

3）第三阶段是 C/RTL Cosimulation，首先需要通过测试台的自我检查，Test Bench 返回 0 标志测试通过，返回非零则表示测试失败，然后要求优化指令和接口综合选项使用合法，最后利用仿真模拟器对 RTL 级语言仿真，这种方式避免了繁琐的上板调试，提升了调式的速度，但是并不能检查出所有的错误，因此最终还是需要上板测试确保设计方案无误。

4）最后一个阶段是 Packaged IP，IP 核的设计理念极为重要，设计方案需要在最后封装成一个完整的 IP 核，在 Xilinx Vivado 中可以导入 IP 核进行实例化，并通过连线接入到系统中，作为搭建硬件平台的模块。

2.5.2 HLS 设计相关的优化指令

1）Inline Directive

类似于 C 语言中的 Inline，可以将被调用函数在调用函数中进行展开， Inline 去除了函数的层次化，可以改善硬件资源的消耗，对于频繁调用并且简单的函数，HLS 会自动 Inline，设计人员也可以显示设定一个函数不被 Inline，一个函数一旦被 Inline 之后，在最后生成的综合报告中不会出现其相关信息。

2）Pipeline Directive

Pipeline 指令主要用于对 For 循环和函数接口进行流水线化，将互不冲突的操作在时间线上并行化，其作用是缩短循环或者函数的指令触发间隔，默认的触发间隔为 N，经过设计优化之后，最优的触发间隔为 1。

以图 2-14 为例，在 Pipeline 优化前，读、计算和写三个操作串行执行，下一轮的读操作被延迟到上一轮写操作执行之后，在前后没有数据依赖的前提下，下一轮的读操作和计算操作可以和上一轮的写操作和计算操作并行，这样设计，硬

件资源的利用率在时间线上是饱和的，虽然并不能优化每一轮操作的计算延迟，但是每一个时钟周期都可以输出一个结果，提升了吞吐率。

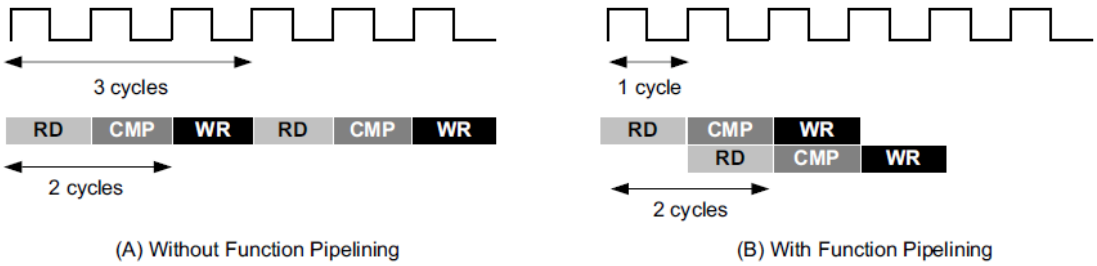

图 2-14 流水线切分原理

对 For 循环 Pipeline 和对函数进行 Pipeline 的不同点在于，For 循环需要显式指定循环边界，如果循环边界本身是一个常量，那么无需指定，如果不是，则必须通过 loop_tripcount 指令指定边界，这为综合时硬件资源的估计提供参考。

Pipeline 优化性能的代价是：切分的多级流水线之间需要插入寄存器暂存中间结果，这将增加空间资源的开销和时间的开销，如果流水线的切分不合理，很可能浪费了空间资源还使性能变得更差，所以在添加 Pipeline 指令之前，需要重构算法的 C 代码，综合时指导编译器进行合理的流水线的划分，对于多个 For 循环的嵌套情况，在设计方案时需要进行拆分，然后对每一层 For 循环做相应的优化。

3）Unroll Directive

For 循环在默认情况下是折叠的，而 Unroll 指令可以展开循环以创建多个独立的操作，由一个循环展开的因子 N 来控制展开的程度，当 N 达到循环边界的大小时，称为完全展开，否则称为部分展开，在 RTL 层面上会创建多份硬件资源，循环体内多轮迭代过程在多份硬件资源上同时执行，这将显著缩短循环执行的时间，多轮迭代过程也需要能够同时获取到执行需要的数据，且能够同时写入执行结果，因此 RTL 层面需要提供足够的读写端口数量。

4）Array Partition Directive

Array Partition 指令通常配合 Pipeline 和 Unroll 指令一起使用，它将数组划分为更小的数组或者单个元素。这种划分使 RTL 具有多个小内存或者多个寄存器，而不是一个大内存，主要用于有效增加存储的读写端口数量，这样并行执行的多个单元可以同时读写数据，读写数据的过程不会成为性能瓶颈。

Array Partition 提供了三种数组划分的方式，如图 2-15 所示：block 方式将数组分为多个连续的更小的数组，每一个数组中存放的是原始数组中的连续元素，cyclic 方式将原始数组中的元素交错来创建更小的数组，数组中的连续多个元素被分到多个不同的数组中，complete 方式对数组在某一维度或者多个维度上完全

进行切分，如果将数组在所有维度都进行切分，那么在 RTL 层面每一个元素都采用一个寄存器实现。

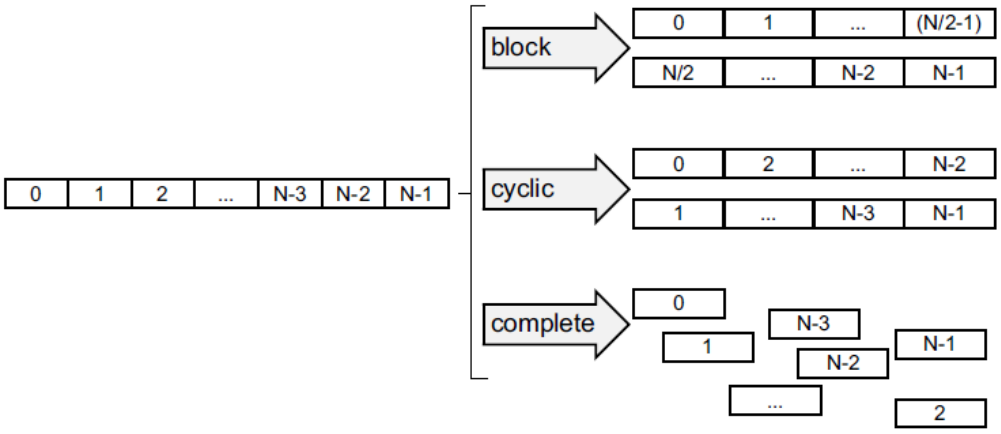

图 2-15 Array Partition 三种工作方式示意图

Array Partition 对于 LUT 和 BRAM 等存储资源消耗巨大，因此设计的方案应该合理的控制读写端口的数量，避免对于特别大的数组做过细的数组切分，浪费存储资源。

5）Dependence Directive

Dependence 指令用于显式指定数据依赖并解决错误的依赖关系，在某些复杂的场景下，编译器的自动依赖工具的数据相关性分析可能无法过滤掉错误的依赖，导致硬件实现错误，因此允许显式指定，数据依赖/相关主要有三种：RAW、WAR、WAW 和 RAR。

• RAW（Read-After-Write-true-dependence）：是指读指令提前读取了写指令将要写入的值，导致读取了错值。例如 x = a+b；c=x+1。

• WAR（Write-After-Read-anti-dependence）：是指读指令得到一个被写指令覆盖的值。例如 b = x+a；x=3。

• WAW（Write-After-Write-output dependence）：是指两个写指令写入同一个值的先后顺序发生反转。例如 t=4；t=3 最后实际执行时变为 t=3；t=4。

Dependence 指令不仅可以设定依赖关系，还可以去除依赖关系，当设计人员确定不存在数据依赖时，可以显式设定，循环体的依赖关系一般分为两种：

• Intra（迭代内相关）：是指同一个循环迭代中的依赖，当依赖指定为 false 时，HLS 可以在循环内自由的移动操作,从而提高它们的移动性并潜在地优化性能和面积，当指定为 true 时，必须按照指定的顺序执行操作。

• Inter（迭代间相关）：是指不同循环迭代之间的依赖关系，如果指定为 false，则允许循环被流水线化、循环展开或部分展示并行执行操作。

6）Resource and Interface Directive

Resource 指令用于显式指定变量在 RTL 中实现的存储资源和方式，例如数组在默认情况被实现为单端口的RAM，但是如果要同时读写，则需要指定为双端口的实现。Interface 用于指定函数接口的类型和协议，从而规定数据的输入输出的方式。本文中使用了 m_axi 接口用于输入输出数据的传输，s_axilite 接口用于控制信号的传输。

2.6 本章小结

本章首先介绍了本文构建语音降噪和语音识别模型用到的深度学习算法的基本原理，其后介绍了传统语音识别的原理和一个完整语音识别系统的基本构成，重点对比了 GMM-HMM 和 DNN-HMM 声学模型框架，然后介绍了 GEMM 的原理和奇异值分解技术，为后文设计加速引擎做铺垫，最后对 Xilinx 的 HLS 高级综合工具开发流程做了简单介绍，并详细阐述了本次课题中用到的相关优化指令的作用和局限性。

第三章 面向噪声的轻量化语音识别模型设计

语音识别在安静环境下的表现效果良好，但是在语音识别系统的实际应用过程中，背景噪声的干扰是不可避免的，背景噪声的频谱很容易与原始语音频谱重合，将原始语音掩盖在噪音频谱范围内，导致识别系统不能准确分离出原声音，对于噪声环境的鲁棒性较弱。本文考虑到嵌入式平台的部署，设计了运算量极小的语音降噪算法作为语音识别的前端，同时也设计了轻量化的语音识别模型，最后综合考虑了模型精度和模型尺寸，利用 SVD 技术对模型权重进行压缩，满足嵌入式部署的要求。

3.1 轻量化语音降噪前端

现实生活中的噪声源类型较多，通常分为稳态噪声和非稳态噪声，在测量时间内，被测声源的声级起伏小于 3dB 的噪声称为稳态噪声，反之称为非稳态噪声，本文所使用的背景噪声来自于以柴油为发动机燃料的大型车辆，包含了多种类型的噪声源：主要有柴油发动机噪声、电子设备噪声等，车辆在驻停状态下噪声主要来自于电子设备，在行驶状态下噪声主要来自于发动机，经过对噪声源的频率图进行分析，设备噪声和发动机噪声都属于非稳态的噪声，并且噪声的峰值声级强度在80dB左右。本文在不踩油门和踩油门的情况下分别录制了环境噪声、干净语音以及含环境噪声的语音，并且以信噪比（SIGNAL-NOISE RATIO，SNR）来衡量背景噪声的恶劣程度，其计算方法见式（3-1）：

$$SNR = 10\lg\left(\frac{P_{signal}}{P_{noise}}\right)(\mathrm{dB}) \qquad (3\text{-}1)$$

经实验分析，不踩油门情况下的信噪比范围为[-4,10]，踩油门情况下的信噪比范围为[-10,2]。

3.1.1 语音降噪算法

语音降噪可以分为有监督和无监督的方法。其中无监督的算法包括谱减法和基于统计模型的方法，也被称为传统方法。总体来说，传统的方法对平稳噪声的抑制效果比较显著，但是对于非平稳噪声，其往往不能得到很好的降噪效果。有监督的算法主要包括基于机器学习模型的方法和基于深度神经网络类的算法，有监督类算法对于非平稳噪声往往能得到更好的降噪效果。综合考虑算法对平稳噪声和非平稳噪声的处理能力、落地实时性和可行性，本文设计了结合传统信号处

理和循环神经网络的算法。算法处理流程如下：

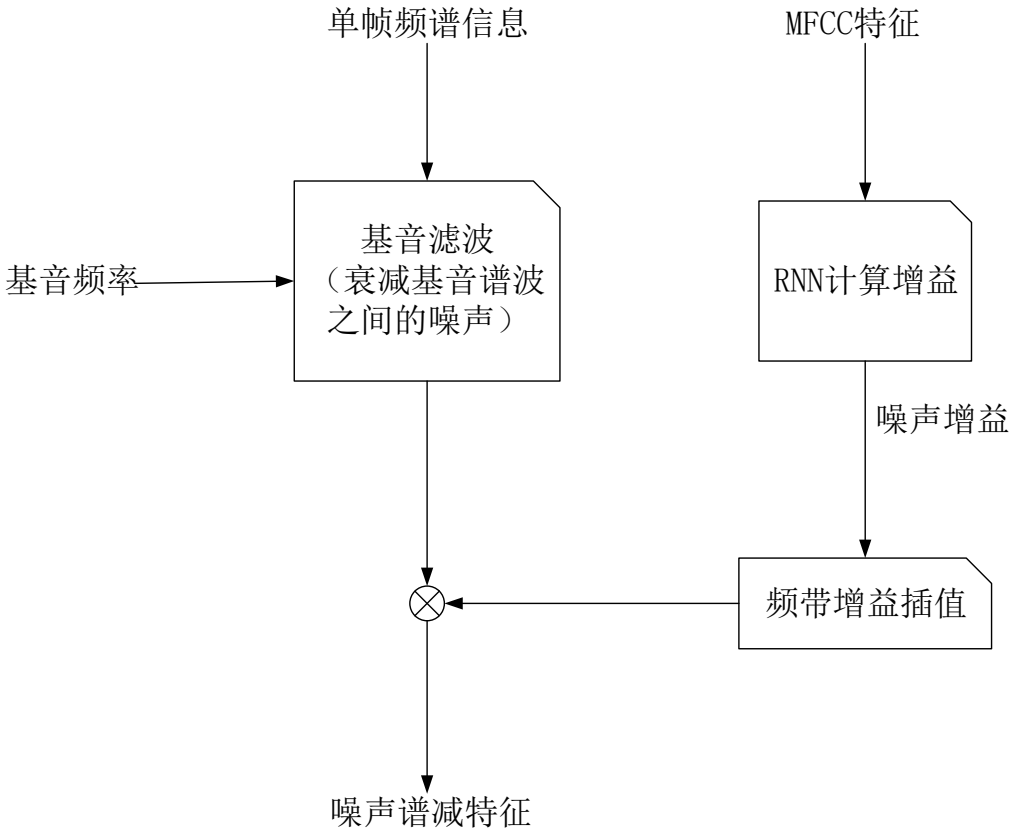

图 3-1 语音降噪处理流程

1）基音滤波：对经过 FFT 的频谱信息和基音分析得到的基音频率进行基音滤波，以衰减基音谱波之间的噪声。

2）RNN 计算增益：将提取的 MFCC 特征输入训练好的 RNN 网络计算噪声增益。

3）频带增益插值：在低分辨率频谱包络上执行 RNN 计算得到的噪声增益。

4）将通过基音滤波后的数据和频带增益插值后的数据合成降噪后的语音特征。

其中利用 RNN 计算噪声增益的网络结构如图 3-2 所示，考虑到算法的简洁性和实时性，网络结构主要以三个全连接层和三个 GRU 层进行构建，GRU 可以解决传统 RNN 的短期记忆问题，同时与 LSTM 相比需要更少的算力和内存。网络设计首先考虑到噪声增益的估计，输出 18 维的增益，用于频带增益插值。其次附加考虑了语音端点检测（Voice Activity Detection,VAD）的功能，利用神经网络评估一段特征中存在语音的可能性，输出一个一维的概率，VAD 对背景噪声环境的适应性较强，不会把目标背景噪声误判为人声，VAD 输出的概率可以作为静音段和纯背景噪声段的过滤依据。

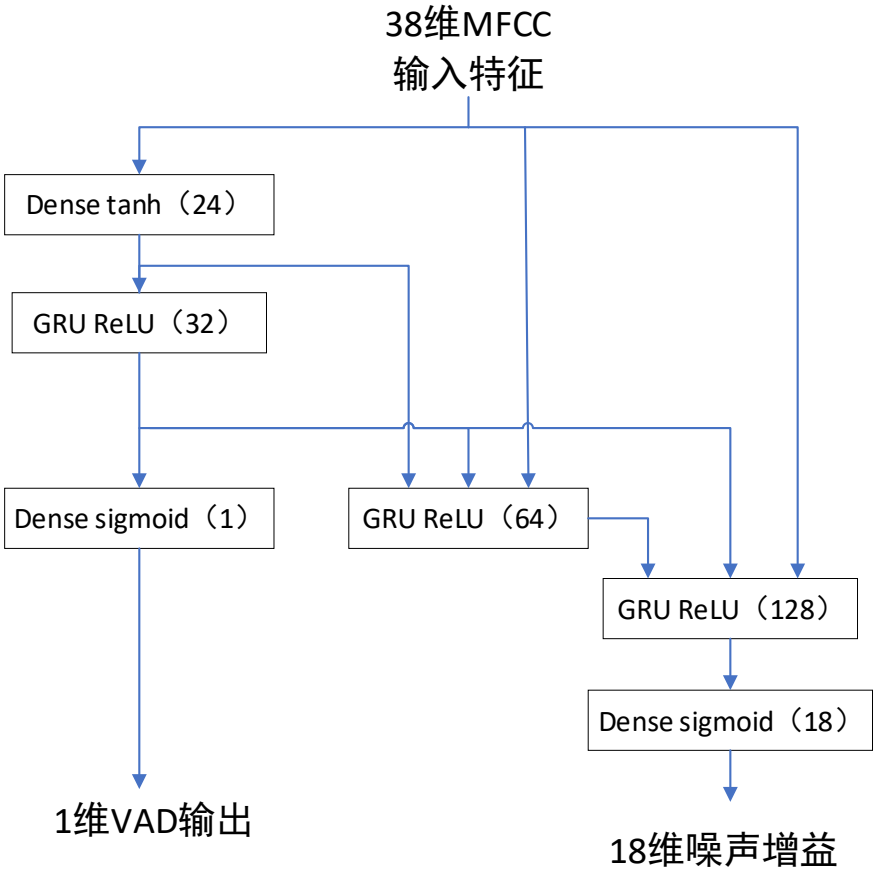

图 3-2 噪声估计网络结构

3.1.2 训练和测试评估

本文从开源中文语音数据集 MAGICDATA、aidatatang、AISHELL 中提取 8GB 的 MFCC 特征，分别以[-15,40]、[-10,2]的 SNR 范围对干净语音进行随机加噪，然后在 tensorflow 和 keras 训练框架下训练，训练时的优化器采用 adam 优化器，噪声估计网络的训练配置如下：

表 3-1 噪声估计网络训练配置

	配置
硬件	Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz 32 核
软件	Ubuntu 18.04.5 LTS tensorflow-gpu=1.14，keras=2.2.4，librosa=0.8.1
主要参数	epochs=120，batch_size=128 L2 regularization=10^{-6}

最后训练得到两个版本的模型权重 v1（对应 SNR[-15,40]）、v2（对应 SNR[-10,2]），对应上面的两种 SNR 范围，本文采用常用的语音降噪的客观评价指标来评价语音降噪的性能，主要包括短时客观可懂度（Short-Time Objective Intelligibility，STOI），语音质量感知评估（Perceptual Evaluation of Speech Quality，PESQ），信噪比增益（SNR Gain）。其中，STOI 和 PESQ 描述了降噪之后语音的质量，也代表语音失真的程度，越高说明语音质量越好。SNR Gain 描述了语音降噪前后的信噪比的增量，也代表噪声抑制的程度，越高代表噪声抑制效果越明显。同时，本文分别以[-10,2]、[-15,40]范围的信噪比对 AISHELL 测试集添加随机噪声，构建了两个版本的测试集 test1、test2，它们使用的干净语音完全一致。使用模型 v1、v2 分别在 test1、test2 上评估 PESQ、STOI、SNR Gain 三个指标，实验结果如下：

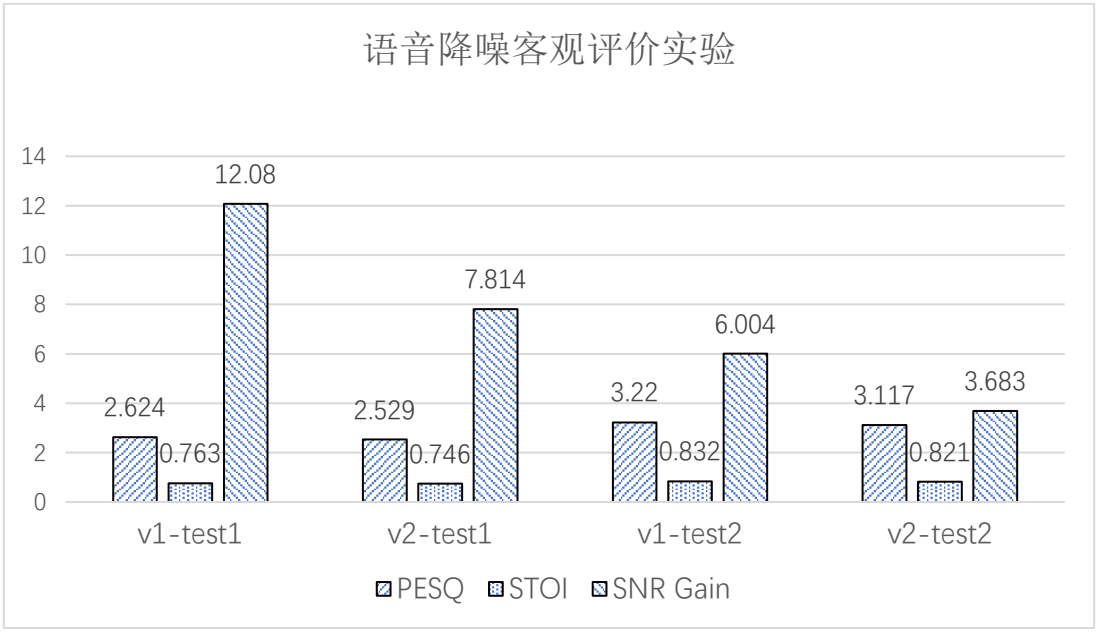

图 3-3 语音降噪客观指标评价实验

从图中可以看出,在相同的测试集上，v1 的效果始终优于 v2，说明以更大的 SNR 范围添加随机噪声训练出的模型的鲁棒性更好。同时，对于相同模型在 test1 和 test2 上的表现，发现了模型在 test1 数据集上噪声抑制程度更强，但在 test2 上的降噪之后的语音质量更好，说明模型在恶劣噪声环境的适应性较好，但是降噪之后的语音有一定程度的失真。对于语音识别而言，这种失真会导致语音数据信息分布的改变，所以需要让语音识别模型学习新的信息分布，才能具有更好的鲁棒性。

最后，在基于 FT-2000A/2 处理器的定制嵌入式平台上进行实验测试，对语音降噪算法的推理性能进行评估，测试环境如下：

表 3-2 语音降噪算法测试环境

	配置
硬件	FT-2000A/2 FTC661@1.0GHz
软件	Kylin V10 SP1 cmake=3.5，g++/gcc=9.3.0

测试结果如下：

```
ndsl@sancog:~/sdk/src/denoise$ numactl -C 0 ./denoise test.wav out.wav 0
test.wav: has voice
process 1 wavs,total_time 293.278 ms,average time 293.278 ms.
```

图 3-4 语音降噪前端性能测试

可以看出，对于 5-10s 的测试语音，处理时延在 300ms 以内，算法在嵌入式平台达到实时性的要求，并且在测试时利用 top 工具查看了 CPU 和内存的利用率，单核 CPU 的利用率在 14%左右，模型内存占用只有 500KB 左右，只占用了较少的嵌入式平台的计算资源和存储资源。

3.2 轻量化语音识别模型设计

深度学习在语音识别领域盛行以后，利用 DNN 替代 GMM 构建语音识别声学模型让识别效果有了显著提升，主要原因是 DNN 对语音特征内部的高阶相关性具有更强的建模能力。DNN 可以选取各种深度学习的算法进行构建，RNN 就是其中一种，RNN 可以有效地对时间依赖帧进行建模，但是其避免不了短期记忆的问题，且较深层次的 RNN 网络的参数量一般比较大。基于 RNN 改进的 LSTM 和 GRU 可以有效地解决短期记忆的问题，但是 LSTM 和 GRU 神经元之间存在时序依赖关系，并行性不高，难以在嵌入式平台尤其是 FPGA 上充分利用硬件资源进行加速。本文基于 DNN-HMM 框架构建语音识别声学模型，并采用 TDNN 设计轻量化的 DNN 模型结构，主要考虑到 TDNN 的以下几点优势：

1）TDNN 网络是多层的，每一层对特征都有较强的抽象能力，越深层的 TDNN 可以对越大时域范围的语音特征信息进行建模。

2）TDNN 以连续的多帧特征向量作为输入，可以较好的表达语音特征在时间上的关系，且具有时间不变性。

3）TDNN 学习时不要求对所学的标记进行精确的定义，且延时训练时共享权重，总参数量较少。

4）TDNN 本质上也是前馈神经网络，其计算原理类似于一维卷积，有较高的并行性，且其底层运算方式可以变换为矩阵向量运算，便于在 FPGA 芯片上进行加速落地。

3.2.1 基于 TDNN 的声学模型结构

本文基于 Kaldi 搭建 DNN 模型，分别构建了 9、11、13 层的 TDNN 网络，命名为 TDNN-9、TDNN-11、TDNN-13。以 TDNN-13 为例，网络结构如图 3-5 所示，

layer	name	layer-type	input-dim	output-dim	input
0	input	input	40	——	——
1	lda	lda	200	200	input[-2,-1,0,1,2]
2	tdnn1	affine-relu-batchnorm	200	1280	lda[0]
3	tdnn2l	linear	2560	256	tdnn1[-1,0]
4	tdnn2	affine-relu-batchnorm	512	1280	tdnn2l[0,1]
5	tdnn3l	linear	1280	256	tdnn2[0]
6	tdnn3	affine-relu-batchnorm	256	1280	tdnn3l[0]
7	tdnn4l	linear	2560	256	tdnn3[-1,0]
8	tdnn4	affine-relu-batchnorm	512	1280	tdnn4l[0,1]
9	tdnn5l	linear	1280	256	tdnn4[0]
10	tdnn5	affine-relu-batchnorm	512	1280	tdnn5l[0],tdnn3l[0]
11	tdnn6l	linear	2560	256	tdnn5[-3,0]
12	tdnn6	affine-relu-batchnorm	512	1280	tdnn6l[0,3]
13	tdnn7l	linear	2560	256	tdnn6[-3,0]
14	tdnn7	affine-relu-batchnorm	1280	1280	tdnn7l[0,3],tdnn6l,tdnn4l,tdnn2l
15	tdnn8l	linear	2560	256	tdnn7[-3,0]
16	tdnn8	affine-relu-batchnorm	512	1280	tdnn8l[0,3]
17	tdnn9l	linear	2560	256	tdnn8[-3,0]
18	tdnn9	affine-relu-batchnorm	1280	1280	tdnn9l[0,3],tdnn8l,tdnn6l,tdnn4l
19	tdnn10l	linear	2560	256	tdnn9[-3,0]
20	tdnn10	affine-relu-batchnorm	512	1280	tdnn10l[0,3]
21	tdnn11l	linear	2560	256	tdnn10[-3,0]
22	tdnn11	affine-relu-batchnorm	1280	1280	tdnn11l[0,3],tdnn10l,tdnn8l,tdnn6l
23	tdnn12l	linear	2560	256	tdnn11[-3,0]
24	tdnn12	affine-relu-batchnorm	512	1280	tdnn12l[0,3]
25	tdnn13l	linear	2560	256	tdnn12[-3,0]
26	tdnn13	affine-relu-batchnorm	1280	1280	tdnn13l[0,3],tdnn12l,tdnn10l,tdnn8l
27	prefinal-l	linear	1280	256	tdnn13[0]
28	prefinal-chain	affine-relu-batchnorm	256	1280	prefinal-l[0]
29	output-l	linear	1280	256	prefinal-chain[0]
30	output	affine	256	4336	output-l[0]
31	prefinal-xent	affine-relu-batchnorm	256	1280	prefinal-l[0]
32	output-xent-l	linear	1280	256	prefinal-xent[0]
33	output-xent	affine-softmax-log	256	4336	output-l[0]

图 3-5 TDNN-13 网络模型结构

网络除了采用 TDNN 层以外，还包含了 ReLU 激活层、BatchNorm 归一化层、全连接层、lda 层等，网络的输入是 40 维的 MFCC 特征，将当前帧、当前帧前两个时刻的帧、当前帧后两个时刻的帧共 5 帧（200 维）拼接送入 lda 层，lda 内部是一个已经训练好的矩阵，用于去除这 200 维特征的相关性，输出维度同样是 200 维，去除相关性十分必要，因为 DNN 特征和相关性有很大关系，要求特征不同维

度之间的相关性较弱。所以 lda 层的参数在整个 DNN 训练过程中保持不变，初始化时从外部读入训练好的 lda.mat 矩阵。lda 层之后是多层连续交替的 affine-relu-batchnorm 层和 linear 层，这里的 affine 和 linear 本质上都是全连接层，但是 affine 在训练过程中利用 NGD（Natural Gradient Descent）方法更新权重，模型的有些中间层利用了前一层多个时刻的输出结果，有些层利用了前面多层的输出结果，本质上都是利用过去、当前和未来的语音信息，对特征向量在时域范围内的高阶相关性进行建模。网络模型的输出是 4336 维的后验概率，用于后续 HMM 的建模。最后构建的 TDNN 基线模型的尺寸如表 3-3 所示：

表 3-3 TDNN 基线模型尺寸

基线模型	模型尺寸
TDNN-9	61MB
TDNN-11	74MB
TDNN-13	88MB

3.2.2 N-Gram 语言模型

N-Gram 语言模型基于马尔可夫假设，N 表示一个词出现的概率与他前面出现的 N 个词有关系，一般而言，在训练语料足够大的情况下，N 越大 N-Gram 模型的效果相对较好，但是语言模型的尺寸相对更大，推理时间也会相对更长，本文基于 kaldi_lm 分别构建了 Bigram、Trigram 和 4-Gram 语言模型，构建过程如下：

1）准备好分好词的文本数据和以及一个词典 lexicon.txt（本文采用开源的中文大词典 DaCiDian）.

2）词频统计：将文本数据中没有出现在词典中的词替换为<UNK>，然后统计出现的词的频数，将所有出现的词按照词频由高到低排序，将所有词和词典中的非静音词合并，再统计词频，按降序排列，生成 unigram.counts，最后将 unigram.counts 中的词改成简短的形式，得到 word_map。

3）利用 word_map 分别训练 Bigram、Trigram 和 4-Gram 语言模型。

如表 3-4 所示，利用 kaldi_lm 构建得到的 Bigram、Trigram 和 4-Gram 语言模型的尺寸依次递增。

表 3-4 语言模型尺寸

语言模型	模型尺寸
Bigram	15MB
Trigram	32MB
4-Gram	87MB

3.2.3 模型轻量化

考虑到模型在嵌入式平台部署的内存资源占用和推理性能这两个因素，本文利用奇异值分解技术对声学模型 TDNN 的模型权重进行分解，以减少参数数量，Kaldi 内部提供了 apply-svd 工具对 DNN 模型做分解，通过指定参数 energy-threshold 和 shrinkage-threshold 来控制网络某些层或者整个网络分解的力度，energy_threshold 用来限制从奇异值矩阵中取得的奇异值的数量，shrinkage-threshold 用于限制权重矩阵的收缩率，当某一层的收缩率高于阈值时，SVD 的过程就会被终止，这两个值越小意味着最后分解得到的模型的尺寸就越小，本文分别对 TDNN-9、TDNN-11、TDNN-13 这三个基线模型（final.mdl）的所有层按照三种不同的分解力度做 SVD 分解，分解后的模型及模型尺寸如表 3-5 所示：

表 3-5 SVD 分解实验

分解后模型及大小	<energy-threshold,shrinkage-threshold>	基线模型及大小
TDNN-9-SVD-S（29MB）	<0.49,0.49>	TDNN-9（61MB）
TDNN-9-SVD-M（35MB）	<0.64,0.49>	
TDNN-9-SVD-L（40MB）	<0.81,0.64>	
TDNN-11-SVD-S（35MB）	<0.49,0.49>	TDNN-11（74MB）
TDNN-11-SVD-M（42MB）	<0.64,0.49>	
TDNN-11-SVD-L（49MB）	<0.81,0.64>	
TDNN-13-SVD-S（42MB）	<0.49,0.49>	TDNN-13（88MB）
TDNN-13-SVD-M（51MB）	<0.64,0.49>	
TDNN-13-SVD-L（57MB）	<0.81,0.64>	

经过不同力度的分解，分解后的模型大小为基线模型大小的 47%~65%，在后文中将对以上模型进行精度测试，以 TDNN-9-SVD-S 为例，分解后各个 affine 层维度的变化所图 3-6 所示，SVD 分解只发生在 affine 层，原始权重矩阵被表示为两个更小的特征矩阵。

layer	input-dim*output-dim	input-dim*bottleneck-dim, bottleneck-dim*output-dim
tdnn1.affine	200*1280	(200*31), (31*1280)
tdnn2.affine	512*1280	(512,50), (50,1280)
tdnn3.affine	256*1280	(256,40), (40,1280)
tdnn4.affine	512*1280	(512,54), (54,1280)
tdnn5.affine	512*1280	(512,60), (60,1280)
tdnn6.affine	512*1280	(512,61), (61,1280)
tdnn7.affine	1280*1280	(1280,84), (84,1280)
tdnn8.affine	512*1280	(512,56), (56,1280)
tdnn9.affine	1280*1280	(1280,70), (70,1280)
prefinal-chain.affine	256*1280	(256,42), (42,1280)
output.affine	256*4336	(256,79), (79,4336)
prefinal-xent.affine	256*1280	(256,51), (51,1280)
output-xent.affine	256*4336	(256,53), (53,4336)

图 3-6 TDNN-9-SVD-S 各个 affine 层的维度变化

3.2.4 训练过程

本文的数据集由多个开源中文语音数据集构成，具体说明如表 3-6 所示：

表 3-6 数据集构成

数据集	说明
NDSL-Clean-Train	800h 干净训练语料，从 MAGIC_DATA、aidatatang、AISHELL 等开源中文语料的训练集中随机抽取得到
NDSL-Noisy-Train	800h 含噪声语料，对 NDSL-Clean-Train 按照[-10,10]的 SNR 范围添加目标背景噪声
NDSL-Denoise-Train	800h 降噪处理后的语料，对 NDSL-Noisy-Train 采用语音降噪前端处理后生成的语料
AISHELL-Clean-Dev-Test	AISHELL 官方数据集，包含 7588 条测试集语音和 14326 条验证集语音
AISHELL-Noisy-Dev-Test	分别对 AISHELL-Clean-Dev-Test 的测试集和验证集按照[-10,2]的 SNR 范围添加目标背景噪声，测试集和验证集大小同上
AISHELL-Denoise-Dev-Test	分别对 AISHELL-Noisy-Dev-Test 的测试集和验证集采用语音降噪前端处理，测试集和验证集大小同上

语音识别采用基于chain模型的LF-MMI训练方法，为了训练出噪声鲁棒性较强的模型，采用多阶段训练的方式，训练环境的配置如下：

表3-7 语音识别训练环境配置

	配置
硬件	Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz 32 核
软件	Ubuntu 18.04.5 LTS Kaldi, kaldi_lm, jieba, openfst, DaCiDian 等
主要参数	minibatch_size=256 （预训练）epochs=8，learning_rate=(0.001→0.0004) （噪声微调）epochs=3，learning_rate =(0.0004→0.0001) （降噪微调）epochs=3，learning_rate =(0.0004→0.0001) （SVD 后微调）epochs=1，learning_rate =(0.0002→0.00005)

在实验过程中发现，直接利用NDSL-Noisy-Train训练TDNN模型，模型较难收敛，最终模型的识别效果也不佳，所以本文先用干净语料对模型进行预训练，然后利用含噪声数据或降噪后的数据进行微调，在模型SVD分解后，识别性能显著下降，需要对分解后的模型重新微调。下面简要说明完整的训练步骤：

1）利用NDSL-Clean-Train训练GMM-HMM模型（包括语言模型训练、单音素模型训练、三音素模型训练等），然后训练TDNN共8个epoch，得到预训练模型TDNN-N。

2）利用NDSL-Noisy-Train训练GMM-HMM模型，然后以预训练模型为初始模型，对模型微调3个epoch，得到噪声数据微调模型TDNN-N-Noisy。

3）利用NDSL-Denoise-Train训练GMM-HMM模型，然后以预训练模型为初始模型，对模型微调3个epoch，得到降噪数据微调模型TDNN-N-Denoise。

4）利用SVD对TDNN-N-Noisy进行分解，得到TDNN-N-SVD-S-Noisy，再利用NDSL-Noisy-Train对TDNN-N-SVD-S-Noisy微调1个epoch。

5）利用SVD对TDNN-N-Denoise进行分解，得到TDNN-N-SVD-S-Denoise，再利用NDSL-Noisy-Train对TDNN-N-SVD-S-Noisy微调1个epoch。

以上训练步骤考虑到了采用/不采用语音降噪前端的差异，也考虑到了采用/不采用SVD技术的差异，下文将对基线模型和多个优化模型进行精度测试。

3.2.5 实验测试和评估

本节实验测试和评估的软硬件环境和训练时一致，首先对训练得到的 Bigram、Trigram、4-Gram 语言模型的性能进行评估，以困惑度作为 N-Gram 模型的性能评价指标，可以看出 N 越大语言模型的困惑度越低，性能越好，综合考虑模型尺寸和性能，选用 Trigram 作为最终的语言模型。

表 3-8 语言模型的性能评价

语言模型	模型尺寸	困惑度
Bigram	15MB	69.251
Trigram	32MB	35.1607
4-Gram	87MB	23.273

同时，本文基于 TDNN 层数、是否让模型在噪声环境下自适应训练、是否采用语音降噪前端以及是否对模型做 SVD 等多个维度做了对比实验。将训练得到的基线模型和优化模型分别在 AISHELL-Clean-Dev-Test（ACDT）、AISHELL-Noisy-Dev-Test（ANDT）、AISHELL-Denoise-Dev-Test（ADDT）测试集上测试，语言模型选用 Trigram，采用字错率 CER 作为中文语音识别精度的评价指标，同时在 ACDT 测试集上测试了各个模型的推理时延，实验结果如表 3-9 和表 3-10 所示：

表 3-9 语音识别模型的性能评价-1

模型名称	**模型尺寸**	**ACDT**	**ANDT**	**ADDT**	**推理时延**
TDNN-9	61MB	6.97%	——	——	446.932 ms
TDNN-11	74MB	6.41%	——	——	528.878 ms
TDNN-13	88MB	6.11%	——	——	606.957 ms
TDNN-9-Noisy	61MB	7.12%	7.94%	——	395.726 ms
TDNN-9-Denoise	61MB	7.03%	——	7.46%	396.711 ms
TDNN-11-Noisy	74MB	6.5%	7.39%	——	443.278 ms
TDNN-11-Denoise	74MB	6.49%	——	7.02%	442.18 ms
TDNN-13-Noisy	88MB	6.16%	7.13%	——	512.744 ms
TDNN-13-Denoise	88MB	6.19%	——	6.64%	519.328 ms
TDNN-9-SVD-S-Denoise	29MB	7.72%	——	8.36%	267.984 ms
TDNN-9-SVD-M-Denoise	35MB	7.49%	——	8.07%	296.454 ms.
TDNN-9-SVD-L-Denoise	40MB	7.36%	——	7.94%	332.494 ms

表 3-10 语音识别模型的性能评价-2

模型名称	模型尺寸	ACDT	ANDT	ADDT	推理时延
TDNN-11-SVD-S-Denoise	35MB	6.95%	——	7.62%	278.555 ms
TDNN-11-SVD-M-Denoise	42MB	6.72%	——	7.33%	327.544 ms
TDNN-11-SVD-L-Denoise	49MB	6.58%	——	7.15%	353.189 ms
TDNN-13-SVD-S-Denoise	42MB	6.67%	——	7.06%	321.174 ms
TDNN-13-SVD-M-Denoise	**51MB**	**6.36%**	**——**	**6.81%**	**347.116 ms**
TDNN-13-SVD-L-Denoise	57MB	6.21%	——	6.68%	382.184 ms

从实验结果来看，预训练模型经过含噪声语音微调过后，在干净语音测试集上的识别效果略微下降，模型在噪声环境下的鲁棒性较强，主要是因为 TDNN 模型本身的抗噪性，语音降噪前端对于语音识别的效果有一定的提升（使语音识别的字错率下降 0.3%-0.5%），经 SVD 分解后的模型尺寸越小，精度丢失越大，推理时延相对下降，综合考虑模型尺寸、语音识别字错率以及推理时延，最终声学模型的 DNN 网络选用综合性能最佳的 TDNN-13-SVD-M-Denoise。

3.3 本章小结

本章首先结合 GRU 和传统信号处理方法设计了轻量化的语音降噪前端，可以将语音识别字错率 CER 降低 0.3%-0.5%，经测试语音降噪前端在 FT-2000A 处理器上满足实时性要求。其后基于 Kaldi 框架设计了不同层数的 TDNN 模型作为基线模型，通过多阶段训练得到适应噪声环境和降噪前端的 TDNN 模型，采用 SVD 技术对 TDNN 模型做了不同力度的分解，减小了模型尺寸，最后综合考虑模型大小和识别性能，通过对比实验选择 TDNN-13-SVD-M-Denoise 作为轻量化的 TDNN 模型，选择 Trigram 作为 N-Gram 语言模型，最终得到语音识别模型 TDNN-13-SVD-M-Denoise-Trigram，该模型在 SNR 为[-10,2]的强噪声环境下字错率为 6.81%，有较强的鲁棒性，且在高性能测试平台上的推理时延在 350ms 以内。

第四章 基于异构环境的 TDNN 加速引擎设计

TDNN 是一个前馈神经网络，和卷积层、全连接层等类似，其底层的推理过程可以看作是大量的矩阵乘法运算的组合，因此矩阵乘法的性能决定了 TDNN 前向推理的延时。在 CPU 端加速矩阵乘法运算已经有许多开源的高性能方案，例如 OpenBLAS、ATLAS2 等提供了一整套线性代数运算加速的方案。在嵌入式平台上，受限于 CPU 的主频，运算和访存能力都较弱，对于一些算力需求较高、响应需求也较高的任务，例如在人机交互场景下的语音识别等，配合加速库优化性能也较难满足实时性，此时只能充分利用嵌入式平台的多核 CPU，或者为嵌入式平台扩展额外的计算资源。一般嵌入式平台的 CPU 核数为 1-4 个，利用多核 CPU 做任务级并行，可以提高任务的吞吐率，但是响应延迟较高的问题仍然存在。为了优化响应延迟，可以添加额外的硬件来加速任务中的性能瓶颈段，并且额外的硬件扩展不会过多的增加整个平台的功耗。一个经典的例子是，苹果公司的 Siri 系统在 iphone 4s 上需要插上电源才可以工作，但是在 iphone 6s 上就可以脱离电源独立工作，其原因是手机内部的主板集成了面向语音的超低功耗的高性能专用集成电路芯片，扩展了硬件的算力。综上，本文选用低功耗的 FPGA 作为嵌入式平台的硬件扩展，搭建了基于 FT-2000A/2 CPU 和国威 690T FPGA 的异构计算环境，采用 PCI-E 作为两者的通信协议，将 TDNN 底层的运算转换为矩阵乘法，并在 FPGA 上设计了 GEMM IP 核加速矩阵乘法运算，基于 C/C++采用 HLS 技术实现了 TDNN 加速引擎，最后对第三章设计的 TDNN-13-SVD-M-Denoise 模型的前向推理过程进行实验测试。

4.1 TDNN 加速引擎架构

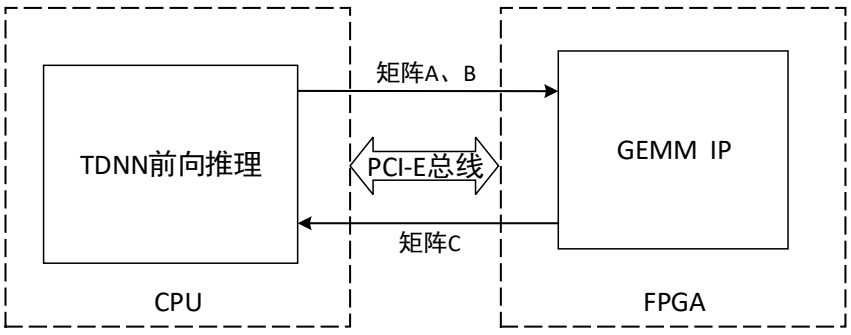

图 4-1 TDNN 加速引擎架构

TDNN 前向推理的过程可拆分为多个顺序执行的矩阵乘法，执行的矩阵乘法

的次数由输入语音的长度和 TDNN 模型的层数共同决定，执行的矩阵乘法的维度则由 TDNN 模型每一层的权重矩阵的维度决定，因此，在 CPU 端准备好待运算的矩阵 A，B，通过 PCI-E 总线发送到 FPGA 端，由 GEMM 引擎加速矩阵乘法得到结果矩阵 C，再通过 PCI-E 总线发回到 CPU 进行其它操作，通过该架构可以对 TDNN 网络模型中的 affine 层和 linear 层进行加速。

4.2 GEMM IP 核设计

在 CPU 端优化 GEMM 时，通常采用 SIMD、数据对齐等技术，程序大部分优化都可以由编译器来完成。而在 FPGA 上，基于可用的逻辑资源设计灵活的架构，即便是对最简单的矩阵乘法运算也面临较大的挑战。基于 FPGA 设计方案和基于 CPU 最大的不同点在于，CPU 端的内存管理由操作系统负责，设计人员只能根据操作系统内核提供的接口（系统调用）来分配和释放资源，再根据分配到的资源做优化设计，例如 STL 容器分配器的设计等。CPU 端设计人员对于内存的控制权较低，而 FPGA 的存储资源可以由设计人员直接管理，所以对于一个特定的算法，需要综合考虑计算代价和 I/O 代价。

为了便于讨论，对矩阵乘法定义如下：两个维度分别为 MxK 和 KxN 的矩阵 A 和 B 相乘得到维度为 MxN 的矩阵 C，即 C=AB。

4.2.1 硬件资源约束

假设 FPGA 内部有 D 种逻辑资源，包括查找表（loopup tables，LUTs）、DSP 等，把这些逻辑资源以及资源数量表示为一个向量 R，见式（4-1）：

$$R_{max} = [R_{1,max}, R_{2,max}, \dots R_{l,max}, \dots, R_{L,max}] \tag{4-1}$$

定义一个基本的运算单元为 PE（Processing Element），PE 可以在一定时钟周期内完成一次浮点乘累加操作，且构建 PE 会消耗 R 中多种类型的逻辑资源，将每个 PE 消耗的逻辑资源定义为 R_{PE}。则构建 PE 总数量 N_{PE} 受限于 FPGA 逻辑资源的总量，N_{PE} 也代表了 FPGA 所支持的最大并行度，资源约束见式（4-2）：

$$\forall_{d\in(0,D)} R_{d,PE} N_{PE} \leq R_{d,max} \tag{4-2}$$

其中 $R_{d,PE}$ 表示 PE 对第 d 种逻辑资源的消耗。

4.2.2 存储方案

新一代 FPGA 的逻辑单元规模已达到百万级，且集成了数千个 DSP 软核，运算能力较强。FPGA 是三级存储的结构，如图 4-2 所示：

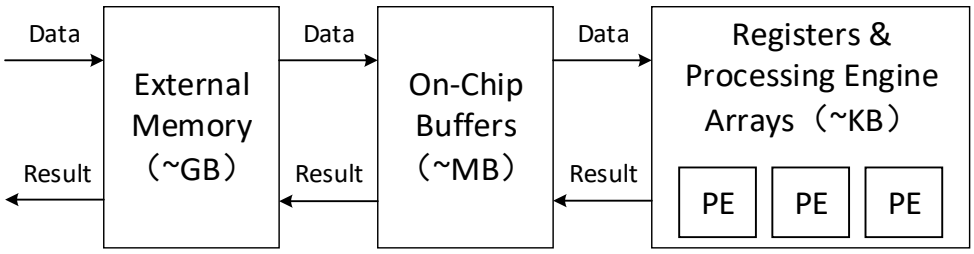

图 4-2 FPGA 三级存储结构

最外层是 External Memory，通常由多片 DRAM 构成，存储空间达到 GB 级，中间层是 On-Chip Buffers，由很多片 18Kb 和 36Kb 的 BRAM 组成，存储空间只有几 MB，最内层是寄存器，由 PE 直接访问，一般存储空间只有 KB 级。存储结构由内到外存储容量越来越大，访存代价也越来越高。本文采用的国威 FPGA 板卡的 External Memory 大小为 8GB，On-chip Buffers 的大小仅有 8.5MB。

一个良好的 FPGA 加速方案往往需要考虑 I/O 访存的代价，即充分利用极有限的 Registers 和 On-Chip Buffers，尽可能减少对 External Memory 的 I/O 操作次数。所以本文首先对 TDNN 网络传输的矩阵的存储空间大小进行分析，计算出在整个推理过程中传输的 A、B、C 矩阵的占用存储空间的最大值，计算方法见式（4-3）：

$$\text{max\_gemm\_storage\_size} = \forall_{l\in(0,L)} \max\left(4\left(S_{l,M}S_{l,K} + S_{l,K}S_{l,N} + S_{l,M}S_{l,N}\right)\right) \quad (4\text{-}3)$$

L 表示调用矩阵乘法的最大次数，$S_{l,M}$、$S_{l,K}$、$S_{l,N}$ 表示某一次调用时 A、B、C 矩阵的各个维度的大小，以 TDNN-13-SVD-M-Denoise 模型为例，测试结果如下：

```
ndsl@sancog:~/sdk/src/asr_kws_16k$ ./asr_kws_engine ./test/00000020.wav asr 0
./test/00000020.wav:他走进值班民警宿舍仔细查看室内涉事询问他们的工作和生活情况
L: 1936
max_gemm_storage_size: 4.35699 MB
```

图 4-3 输入输出矩阵的空间资源占用

由测试结果知，A、B、C 三个矩阵占用的最大存储空间为 4.36MB，小于国威 FPGA 板卡的 BRAM 空间，因此考虑将 A、B、C 矩阵完全存放在 BRAM 中，从而避免 DRAM 到 BRAM 的数据分块，因此从 DRAM 到 BRAM 每一个数据只需要加载一次，I/O 操作总次数是最优的。

4.2.3 计算方案

GEMM 本质上可以看作是多个向量乘加运算的组合，即 $O_{i,j} = MAC(V_{A,i}, V_{B,j})$,如下图：

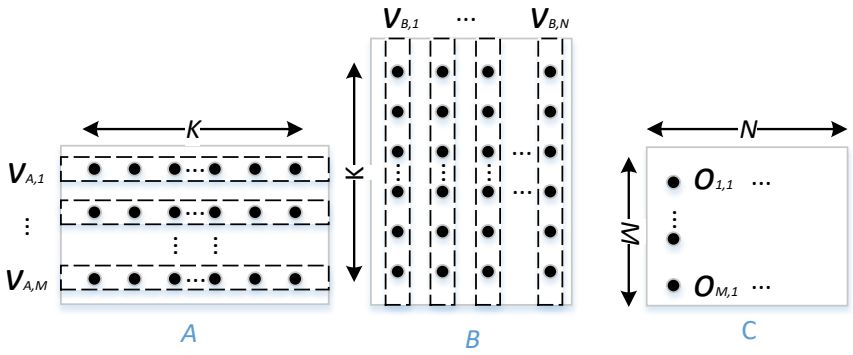

图 4-4 矩阵向量乘加原理

每一条向量的长度都为 K，两个向量做乘加运算时，由于累加过程，前后会产生数据依赖，不同向量的运算之间没有数据依赖关系（但是有重复的数据），可以完全并行执行。如果把每两个向量的乘加运算都用一个 PE（Processing Element）（内部包含一个乘累加器）用多个时钟周期完成，那么要同时做完所有的向量乘加运算共需要 M*N 个 PE，每个 PE 需要进行 2K 次读操作和一次写操作，如果每个 PE 均从 BRAM 获取所有的输入数据，那么其中重复的数据会被多次读取，增加了带宽消耗，其次，一片 BRAM 最多提供一个读接口和一个写接口，如果各个 PE 需要的数据并不连续，会增加寻址和读操作的时间，如果采用多片 BRAM 提供更多的读写接口，则会出现 BRAM 利用率不高的弊端。

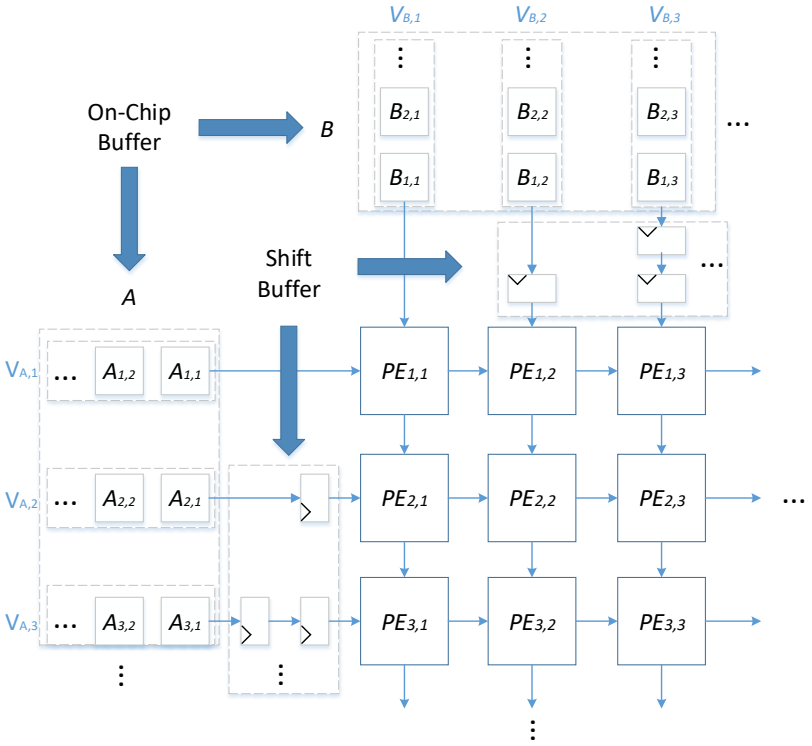

图 4-5 GEMM 脉动阵列实现结构

考虑到数据重用和 I/O 代价，本文采用脉动阵列实现 GEMM，原理如图 4-5 所示，图中蓝色细箭头表明了脉动阵列的数据流向，A 矩阵和 B 矩阵分别存放在 On-Chip Buffer 中，与图 4-4 相同的地方在于，每两个向量的乘加运算仍然是交给一个 PE 分多个时钟周期来做，例如 $PE_{1,1}$ 负责处理 $V_{A,1}$ 和 $V_{B,1}$，$PE_{2,3}$ 负责处理 $V_{A,2}$ 和 $V_{B,3}$。不同点在于，数据通过传输在 PE 之间共享，只有最外层（第一行、第一列）的 PE 从 BRAM 中获取数据，其它的 PE 均从外层的 PE 获得共享数据，这大大减小了存储的带宽。值得一提的是，根据脉动阵列的实现原理，A 矩阵同一列的数据 $A_{1,1}$、$A_{2,1}$、$A_{3,1}$ 和 B 矩阵同一行的数据 $B_{1,1}$、$B_{1,2}$、$B_{1,3}$ 并不是在同一时刻进入脉动阵列，本文采用 Shift Buffer 的形式将数据在时间轴上错开，Shift Buffer 内部由多个移位寄存器构成，个数取决于 PE 阵列的维度。不妨设 PE 阵列的维度为 P_m x P_n（$N_{PE}=P_m * P_n$），Pm 和 M 方向上的维度一致，Pn 和 N 方向上的维度一致，假设单个数据的位宽为 W，则每一时刻，分别从 A 矩阵和 B 矩阵（两块 BRAM）中读取 Pm*W 和 Pn*W bit 的数据。Shift Buffer 总共需要的移位寄存器个数的计算见式（4-4）：

$$\begin{aligned}\mathrm{Total}_{shift\ register} &= \sum 1+2+\cdots+P_m-1+\sum 1+2+\cdots+P_n-1 \\ &= \frac{P_m(P_m-1)+P_n(P_n-1)}{2}\end{aligned} \tag{4-4}$$

下面基于 2x2 的脉动阵列，对脉动阵列的计算过程进行分析，这关系到矩阵运算的结果在何时写回 BRAM：

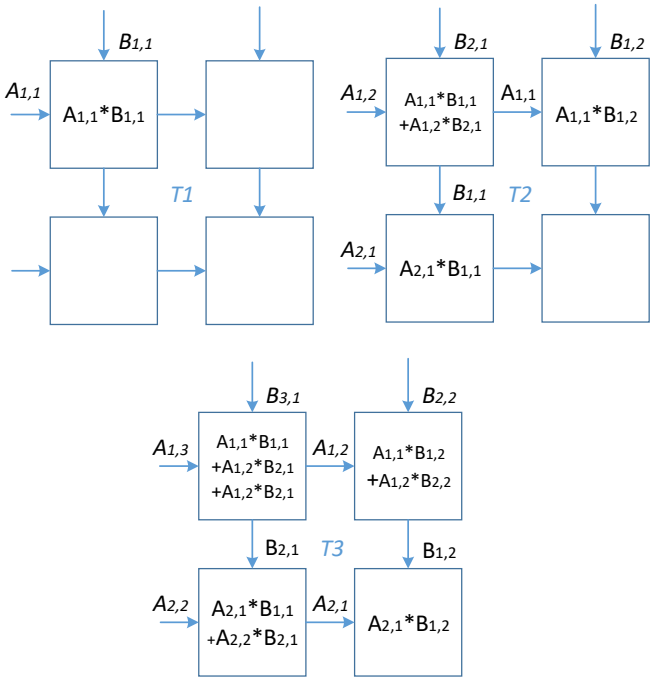

图 4-6 GEMM 脉动阵列计算过程

$PE_{1,1}$ 在 T1 时刻处理完一次乘法，在 T3 时刻处理完三次乘法，由于向量的长度为 K，所以 $PE_{1,1}$ 在第 K 个时刻才会产生向量的最终运算结果，以此类推，$PE_{1,2}$ 和 $PE_{2,1}$ 在第 K+1 个时刻产生最终结果，$PE_{2,2}$ 在第 K+2 个时刻产生最终结果。更一般的，$PE_{i,j}$ 产生最终结果的时刻见式（4-5）：

$$\forall_{i\in(1,P_m),j\in(1,P_n)}\text{Output Time of } PE_{i,j} = K + i + j - 2 \quad (4\text{-}5)$$

因此在脉动阵列中，左上角的 $PE_{1,1}$ 在第 K 个时刻最先得到结果，$PE_{Pm,Pn}$ 在第 K+Pm+Pn-2 个时刻最后得到结果。此时需要将脉动阵列中所有计算结果写回到 BRAM，同样通过脉动阵列传输结果，如下图：

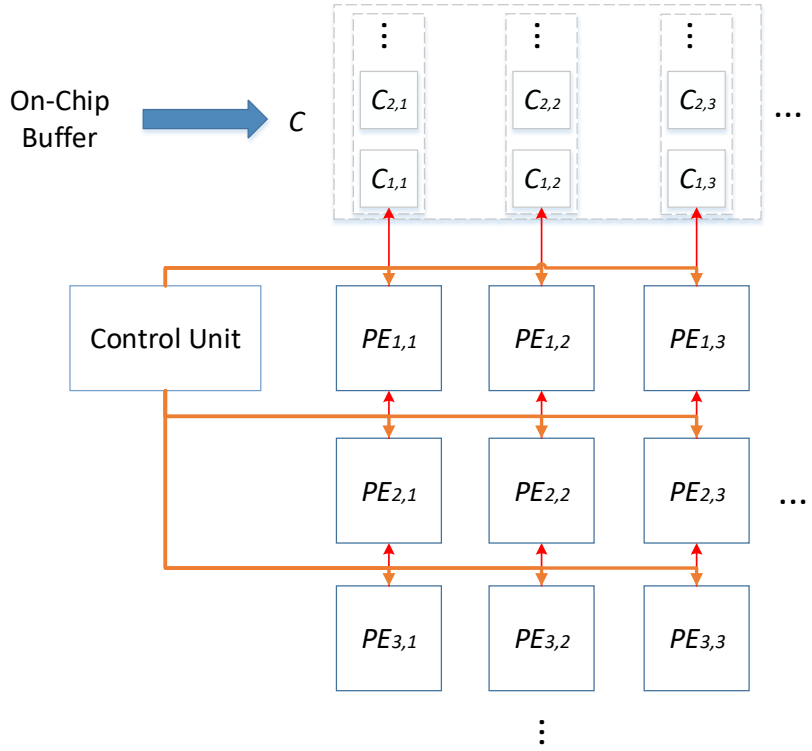

图 4-7 GEMM 脉动阵列写结果

C 矩阵同样存放在 On-Chip Buffer 上，当 $PE_{Pm,Pn}$ 得到最终结果后，通过控制单元的控制信号，将结果由上至下按行写回到 BRAM 中。经过 P_m 个时刻，所有的结果写回到 BRAM 中。因此，从 A、B 矩阵准备好到 C 矩阵生成，GEMM 运算共需要 K+2*Pm+Pn-2 个时刻。

脉动阵列内部的 PE 采用流水结构设计，每个 PE 内部有一个乘累加器，考虑到做 32bit 的浮点乘法，无法在一个时钟周期内部完成一次乘累加，所以需要延时等待输出结果，在每个 PE 内部维护两个寄存器分别用于暂存当前时刻 A 矩阵

的数据和 B 矩阵的数据，便于下一时刻将数据传输到右边和下边的 PE。脉动阵列内 PE 的结构如下：

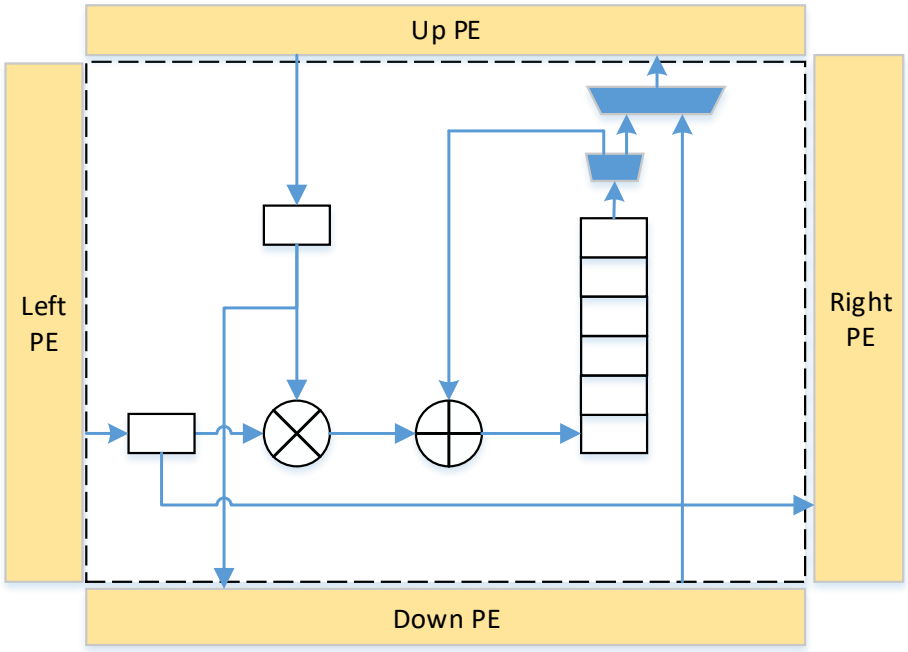

图 4-8 脉动阵列 PE 结构设计

4.2.4 分块策略和参数搜索

脉动阵列一旦硬化之后，P_m 和 P_n 就确定好了，脉动阵列中 PE 的个数受限于硬件资源，所以 P_m 和 P_n 设定的过小或者过大都不合理，在 TDNN 推理过程中，矩阵的维度是不断变化的，即 M、N、K 不是固定的，因此脉动阵列在每一轮计算过程中，输入的不是 A 和 B 的完整矩阵，而是 A 和 B 的一个分块。本文在 M 和 N 维度上对矩阵进行切分，假定 A 矩阵的分块 T_A 的维度为 P_m x K，B 矩阵的分块 T_B 为 K x P_n，矩阵分块的示意图如下：

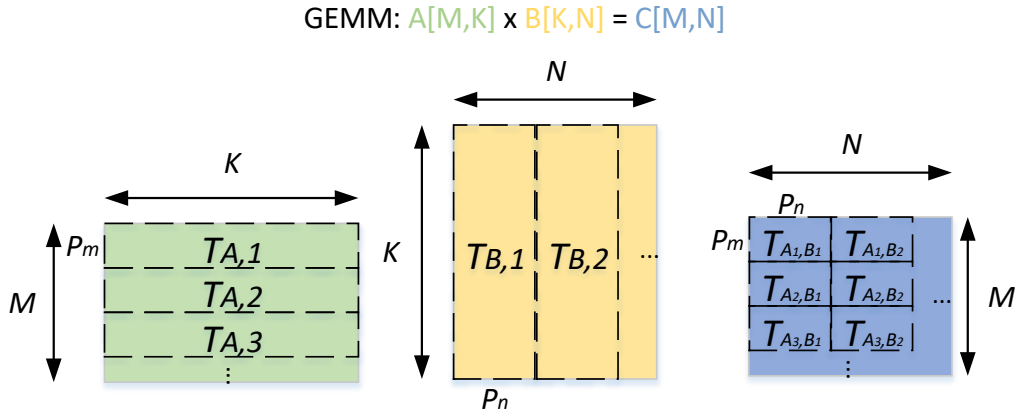

图 4-9 矩阵分块示意图

因此 A、B 矩阵的分块数量以及 C 矩阵的结果分块数量计算见式（4-6）：

$$\mathrm{T}_{A,max} = ceil\left(\frac{M}{P_m}\right)$$

$$\mathrm{T}_{B,max} = ceil\left(\frac{N}{P_n}\right)$$

$$\mathrm{T}_{C,max} = \mathrm{T}_{A,max} * \mathrm{T}_{B,max} \quad (4\text{-}6)$$

$\mathrm{T}_{C,max}$ 也表示调用脉动阵列计算的次数，$\mathrm{P_m}$ 和 $\mathrm{P_n}$ 并不能整除 M 和 N，因此 A 矩阵和 B 矩阵的最后一个分块的有效数据并不能填满整个分块，本文的策略是对 A 矩阵和 B 矩阵的最后一个分块补 0，这将增加额外的 BRAM 资源的使用，且对于不同层而言，受 $\mathrm{P_m}$ 和 Pn 参数的影响，填补 0 的个数也不尽相同，某一层矩阵填补 0 的总量的计算方法见式（4-7）：

$$\mathrm{Total\_Num}_l = \left(\mathrm{P_m} * ceil\left(\frac{M_l}{P_m}\right) + \mathrm{P_n} * \mathrm{ceil}\left(\frac{N_l}{P_n}\right) - M_l - N_l\right) * \mathrm{K}_l \quad (4\text{-}7)$$

不考虑其它时间，对脉动阵列总的运行时刻数进行估计，见式（4-8）：

$$\mathrm{Total\_Time\_Counts}_l = (\mathrm{K}_l + 2\mathrm{P_m} + \mathrm{P_n} - 2) * ceil\left(\frac{M_l}{P_m}\right) * ceil\left(\frac{N_l}{P_n}\right) \quad (4\text{-}8)$$

因此，合适的 $\mathrm{P_m}$ 和 $\mathrm{P_n}$ 值的选择依赖于以下优化目标和约束，见式（4-9）：

$$\mathrm{minimize} \sum_{l\in(0,L)} \mathrm{Total\_Num}_l \ \text{ and } \sum_{l\in(0,L)} \mathrm{Total\_Time\_Counts}_l$$

$$\text{subject to}$$

$$0 \le \mathrm{P_m} \le \max_{l\in(0,L)} M_l$$

$$0 \le \mathrm{P_n} \le \max_{l\in(0,L)} N_l$$

$$\forall_{\mathrm{d}\in(0,\mathrm{D})} R_{d,PE} P_m P_n \le R_{d,max} \quad (4\text{-}9)$$

即最小化所有矩阵填补 0 的总容量以及最小化脉动矩阵运行的总时刻数，约束条件包括硬件资源约束，基于该优化目标进行搜索，最后将 Pm 定为 14，将 Pn 定为 64。

4.2.5 GEMM IP 核实现

基于上述方案和策略，实现的 GEMM IP 核架构如图 4-10 所示，矩阵的浮点数据通过 PCI-E 总线达到 FPGA 端之后，再通过 AXI 总线传输到 DDR3 中，A、B、C 矩阵可以全部存放在 On-Chip Buffer 中，所以使用 Data Router 将矩阵数据进行重排，然后将数据全部存放到 BRAM 中，重排的目的是为了满足脉动阵列在地址空间上连续获取数据的要求，运算过程中，按序读取 A、B 矩阵的分块送入

Shift Buffer，Shift Buffer 将脉动阵列需要的数据在时间上进行对齐，在经过给定的时钟周期之后，14 x 64 脉动阵列中的所有 PE 节点都计算出最终结果，通过控制单元控制将数据按行写回到 BRAM 中，所有分块运算结束后，将 C 写回到 DDR3，再通过 AXI 总线和 PCI-E 接口将浮点结果矩阵传回到 CPU 端。

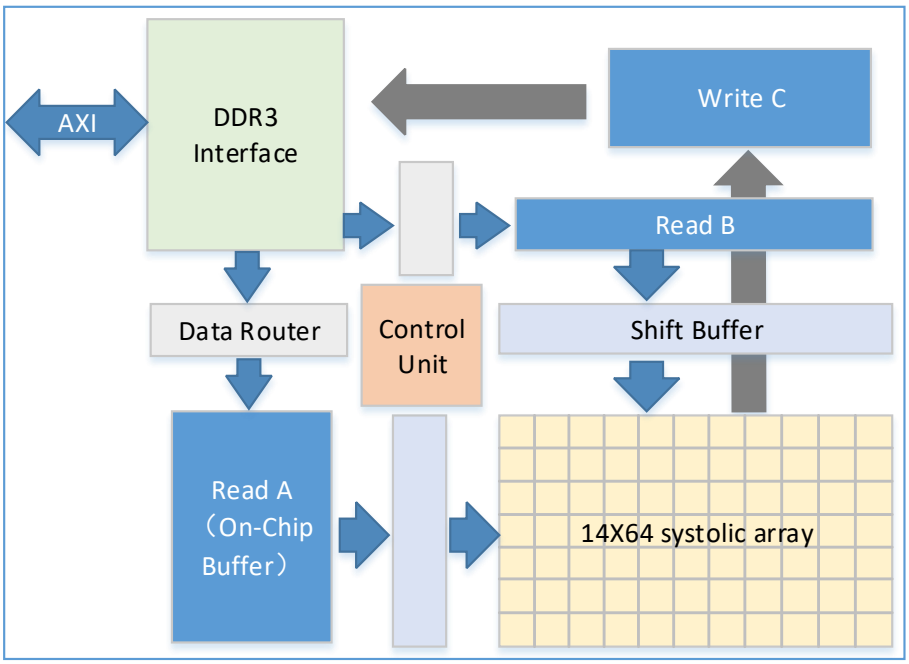

图 4-10 GEMM 加速引擎实现架构

本文基于 Xilinx HLS 工具设计开发了 GEMM 的 IP 核，封装后的 IP 核示意图如下：

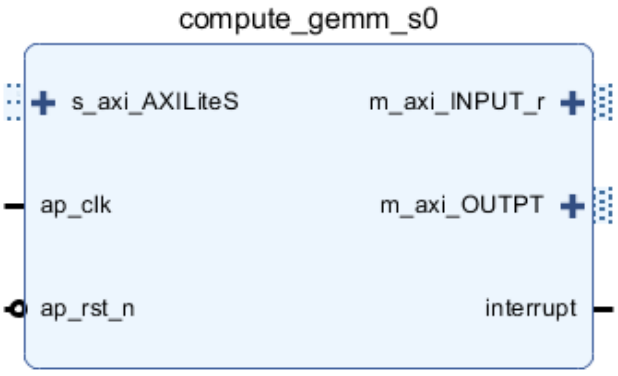

图 4-11 GEMM IP 核封装示意图

数据的输入输出采用 m_axi 协议接口，对 IP 核的控制采用 AXILite 协议接口，IP 核的工作时钟设定为 100MHz，CPU 端和 IP 核的交互流程为：

1）CPU 端将数据传输到 DDR3。

2）CPU 通过 AXILite 协议接口启动 IP 核开始运算。

3）CPU 端通过轮询的方式等待 IP 核计算结束。

4）CPU 端从 DDR3 读取运算结果。

最后，通过 HLS 的资源报告分析，GEMM IP 核在国威 690T FPGA 的资源占用情况如表 4-1 所示

表 4-1 FPGA 资源占用情况

资源	BRAM_18K	DSP48E	FF	LUT
资源消耗	2175 （74%）	1804 （50%）	485186 （56%）	355224 （82%）

4.3 实验测试和评估

本文基于 HLS2018.2 和 Vivado2018.2 工具设计开发了 TDNN 的加速引擎，并在 FT2000A/2 CPU 和国威 690T FPGA 构成的异构计算环境下测试 TDNN-13-SVD-M-Denoise 的前向推理过程，并对推理过程中的端到端时延、PCI-E 数据传输的时间和 IP 核的运算时间进行统计，输入的语音长度在 6-8s 内，测试结果如下图所示：

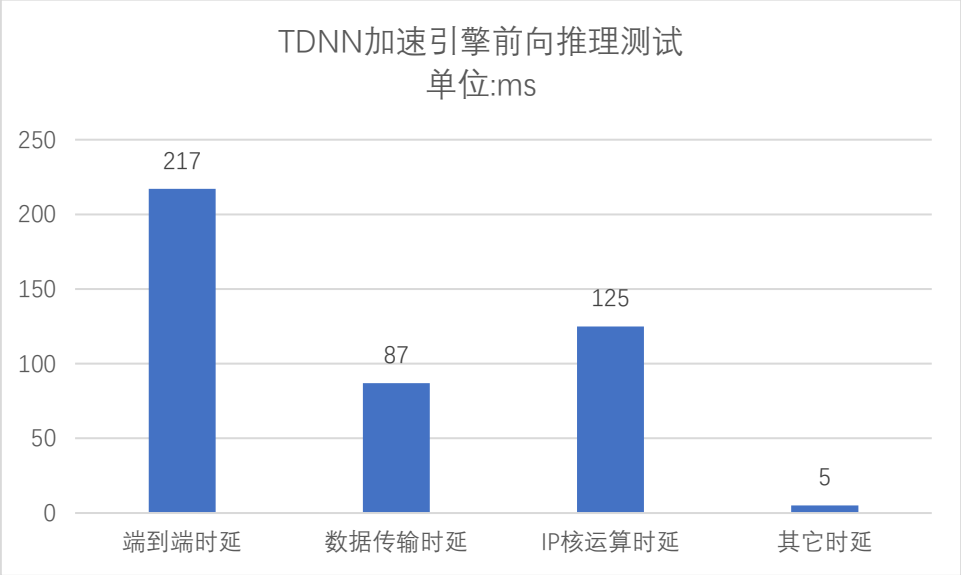

图 4-12 TDNN 加速引擎前向推理测试

从结果可以看出，前向推理端到端的时延近似等于 PCI-E 的数据传输时延和

IP 核的运算时延之和，传输时延较高，占用了总时间的 40%左右，其包括了所有矩阵乘法的 A、B、C 矩阵浮点数据的传输，这是将 TDNN 前向推理过程拆分成多次矩阵乘法的弊端，模型的权重无法复用，相同的权重数据被多次发送到 FPGA。IP 核的运算时延较短，这得益于脉动阵列的高并发性以及从 DDR3 的低带宽读写。

4.4 本章小结

本章主要设计实现了基于 TDNN 网络的前向推理加速引擎，将 TDNN 网络拆分为多次矩阵乘法，并采用脉动阵列方法在 FPGA 上设计实现了加速矩阵运算的 GEMM IP 核，讨论了 IP 核的设计方案、策略以及最终的实现架构，并利用该引擎对 TDNN-13-SVD-M-Denoise 的前向推理进行加速，实验发现数据传输的时延占用了总推理时间的 40%，是该方案的一个缺陷，但是 IP 核的运算时间足够快，可以满足 TDNN 推理实时性的要求。

第五章 面向噪声的嵌入式语音识别系统设计及测试

本文以 Kaldi 为基础设计嵌入式语音识别系统，经实验分析，在 CPU 端运行语音识别，以 OpenBLAS 进行加速后，推理的性能瓶颈仍然在声学模型的 DNN 部分，综合考虑 FPGA 的加速特性和硬件资源限制，利用第四章设计的 TDNN 加速引擎加速 TDNN 网络前向推理，其余部分在 CPU 端运行。基于此，开发了基于嵌入式异构平台的 C++语音识别 API。同时，设计开发了基于 Linux 的录音模块和语音断句模块，搭建了一个小型的离线近场嵌入式语音识别系统，并进行了详细的测试分析。

5.1 实验软硬件环境

本章节所有实验的软硬件环境配置均如表 5-1 所示，实验基于纯国产环境，采用国威 FPGA 板卡（FPGA 芯片型号为 SMQ7VX690TFFG1761）和 FT-2000A/2 搭建异构嵌入式计算环境，两者之间采用 PCI-E 接口进行数据传输通信。其中，声学模型的 TDNN 网络基于国威 FPGA 板卡加速，语音识别的其余部分（特征提取、解码等）基于 FT-2000A/2 运行。

实验基于国产麒麟操作系统完成 CPU 端程序的开发，TDNN 加速引擎基于 Xilinx Vivado HLS 2018.2 和 Xilinx Vivado 2018.2 完成开发。软件的编译环境采用 CMake、gcc 等工具，同时系统依赖于第三方高级 Linux 声音架构库 AlsaLib。

表 5-1 系统软硬件环境

	配置
硬件设置	FT-2000A/2 FTC661@1.0GHz 国威 FPGA 板卡(FPGA 型号：SMQ7VX690TFFG1761) PCI-E 接口：PCI-Ex8 Gen3，最大物理带宽 64Gb/s CPU 端内存 3GB DDR3，FPGA 板载 2 组 64bit 4GB-DDR3
软件设置	Kylin V10 SP1 Xilinx Vivado HLS 2018.2、Xilinx Vivado 2018.2 AlsaLib、CMake=3.5、g++/gcc=9.3.0

5.2 嵌入式语音识别系统整体设计

本文的目标是基于嵌入式平台搭建离线语音识别系统，该系统需要具备从噪声环境持续采集语音的能力，并且有一定的策略可以过滤掉不含人声的音频段，减少模型调用的次数，从而减少系统工作的功耗 。其次，系统可以适应音频中的停顿间隙，根据说话人的说话间隔将音频切分为具备完整语义的语句，这样既保证了识别的响应延迟，也保证了识别的效果。本节详细阐述了系统的设计思路和策略。

5.2.1 系统架构

如图 5-1 所示，系统采用三级流水线的架构搭建，主要由录音单元、断句单元、识别单元以及线程通信单元构成。

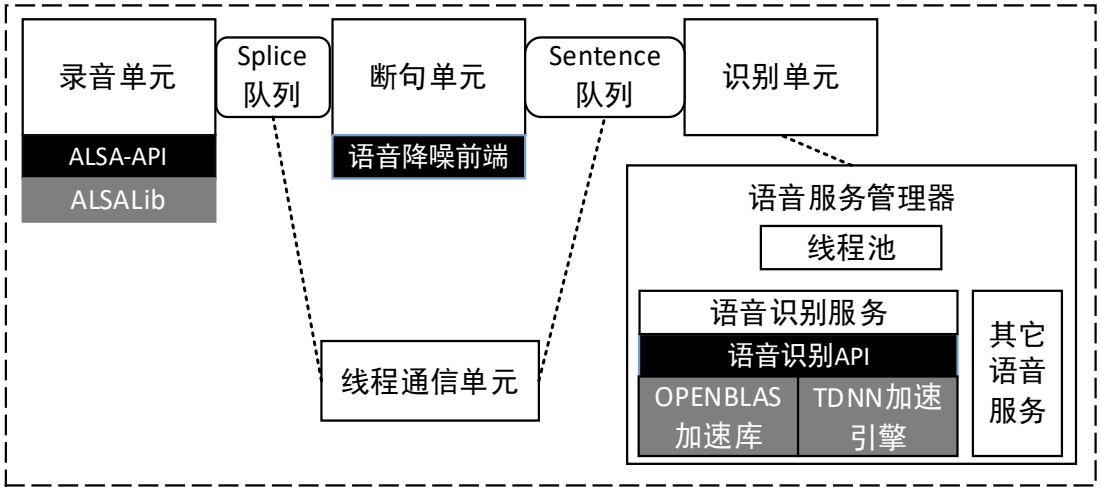

图 5-1 嵌入式语音识别系统架构

录音单元和断句单元内部采用单线程处理，识别单元向语音服务管理器申请语音识别服务，可采用多线程处理。流水线之间以生产者-消费者的模型实现线程间通信，以 splice 队列和 sentence 队列作为通信实体，队列利用锁机制来保证通信的线程安全。录音单元依赖于第三方库 ALSALib，它在 Linux 操作系统上提供音频接口和 MIDI（Musical Instrument Digital Interface，音乐设备数字化接口）的支持，在 ALSALib 之上封装了一层 ALSA-API，提供了音频采集和音频播放的接口，录音单元基于 ALSA-API 实现，在单线程内部循环调用音频采集的过程。断句单元依赖于语音降噪前端提供的 VAD 功能，语音识别服务则依赖于下层的语音识别 API。

5.2.2 语音识别方案分析和 API 设计

本小节首先对无优化的语音识别的推理时间进行分析，找到性能的瓶颈点，然后针对解码过程进行优化。一方面针对 TDNN 网络提供两种加速方案，一种是在 CPU 端利用 OpenBLAS 库进行加速，另一种是在 FPGA 利用 TDNN 加速引擎进行加速。利用前者进行加速时，整个语音识别都工作在 CPU 端，CPU 的负载较高。利用后者进行加速时，系统跨 CPU 和 FPGA 异构计算环境，语音识别中的特征提取和解码流程全部运行在 CPU 端，FPGA 承担了主要的计算负载。另一方面对解码网络进行剪枝优化，提升图遍历的速度。最后，基于这些优化封装了完整的 C++语音识别 API，并针对 API 接口做了实验测试和分析。

5.2.2.1 无加速的语音识别方案

本文利用 Kaldi nnet3 框架部署了第三章设计的轻量化语音识别模型 TDNN-13-SVD-M-Denoise-Trigram，整个语音识别推理过程中比较耗费时间的过程有模型加载、解码、模型销毁，解码是指在构建好的解码网络（图）中，根据输入的音频，生成最优序列的过程。解码流程中包含了声学模型中的 TDNN-13-SVD-M-Denoise 的前向推理。在不做任何优化的前提下，测试 6-8s 的语音，语音识别在 FT2000A/2 CPU 上单核测试的时间分布，如图 5-2 所示。

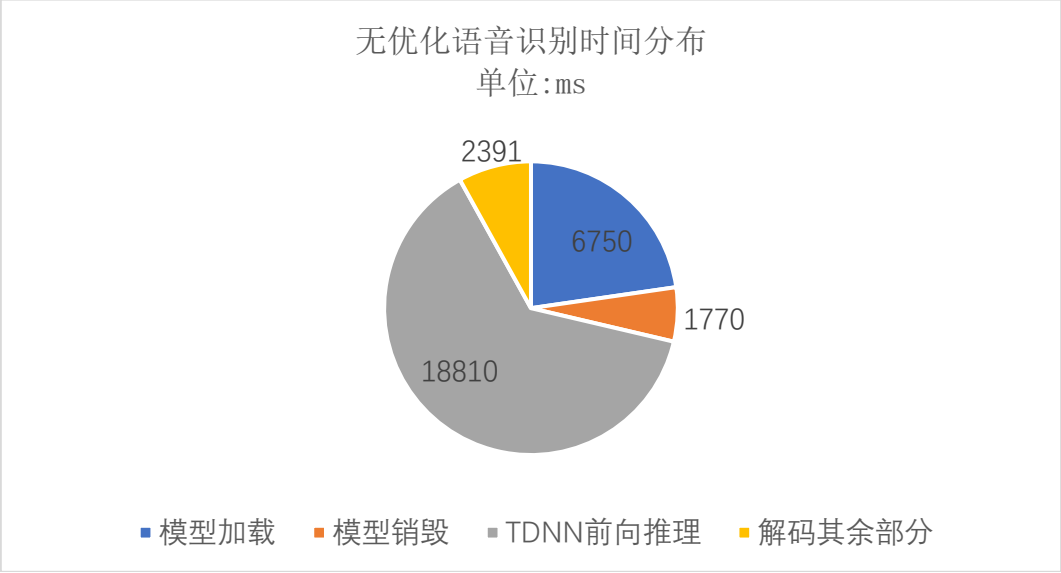

图 5-2 无优化语音识别时间分布

可以看出，最耗费时间的部分在 TDNN 网络的前向推理，模型加载、模型销毁和解码的其余部分同样比较耗费时间。模型加载和销毁的速度与模型本身的大小、I/O 能力等相关，没有进一步优化的空间，因此从 TDNN 前向推理和解码的其余部分进行优化。

5.2.2.2 基于 OpenBLAS 加速 TDNN

OpenBLAS 是一个开源的矩阵计算库，包含了诸多精度和形式的矩阵向量计算方法，就精度而言，包括 float 和 double 两种数据类型的数据，其矩阵调用函数也是不一样。不同矩阵，其计算方式也有所不同，例如，向量与向量之间的计算，向量与矩阵之间的计算，矩阵与矩阵之间的计算。本文使用 OpenBLAS 的 cblas_sgemm 函数接口加速 TDNN 前向推理，其主要利用寄存器分块、SIMD、指令流水线、多线程并行计算等技术加速浮点矩阵乘法运算。本文为了方便对比，不采用 OpenBLAS 的多线程并行加速，设定 OpenBLAS 的工作线程数量为 1，按照 5.2.2.1 节测试了语音识别的时间分布，如图 5-3 所示：

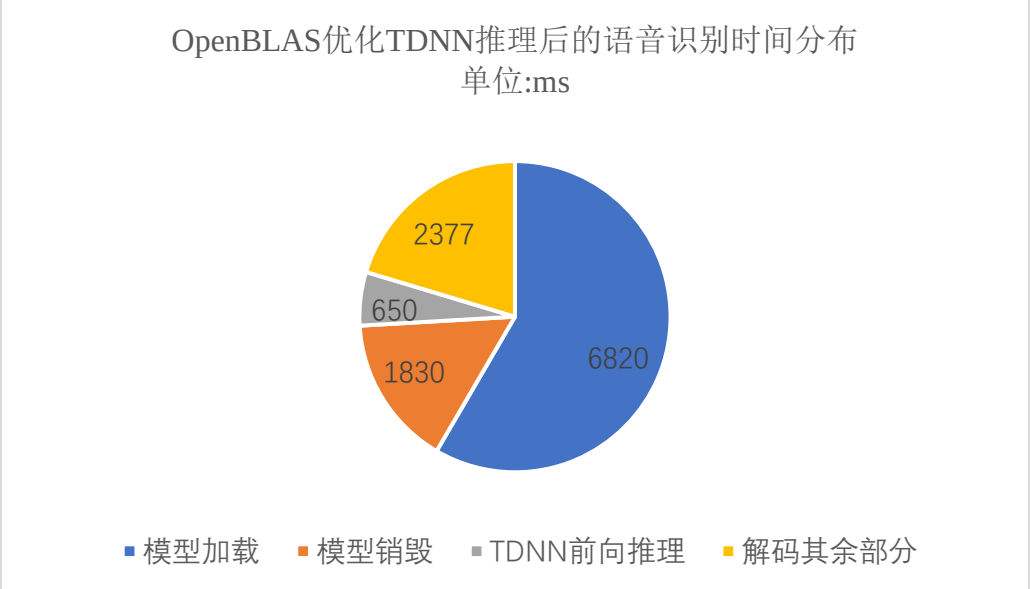

图 5-3 OpenBLAS 优化 TDNN 推理后的语音识别时间分布

可以看出，经过 OpenBLAS 优化之后，TDNN-13 的推理时间由 18810ms 降为 650ms，加速比达到 28.94，其余部分的时间分布基本保持不变，解码的其余部分的时间成为语音识别推理的性能瓶颈，计算过程无法满足实时性要求。

5.2.2.3 解码剪枝优化

本文为了保证语音识别在嵌入式平台的推理速度，采用了基于 WFST 的静态解码方法，核心算法是令牌传递（Token passing），类似于 Viterbi 算法，是一种基于动态规划的算法。本文使用了 Kaidi nnet3 实现的 LatticeFasterOnlineDecoder，其使用 beam search 方法基于展开的 HCLG 静态图搜索，得到最优路径，在遍历图时，可以通过调整剪枝系数 beam 和 lattice beam、最大活跃状态数目 max_active、最小活跃状态数目 min_active 来减少遍历发射弧的数目，从而提升遍历解码图的

速度，代价是牺牲一定的识别精度。本文综合考虑解码速度和识别精度，经实验调参，最终将 beam 参数配置为 13.0（默认为 16.0），lattice beam 参数配置为 13.0（默认为 10.0），max_active 参数配置为 6000（默认为 32 位整型最大值），min_active 参数配置为 20（默认为 200），按照 5.2.2.1 节测试了语音识别的时间分布，如图 5-4 所示：

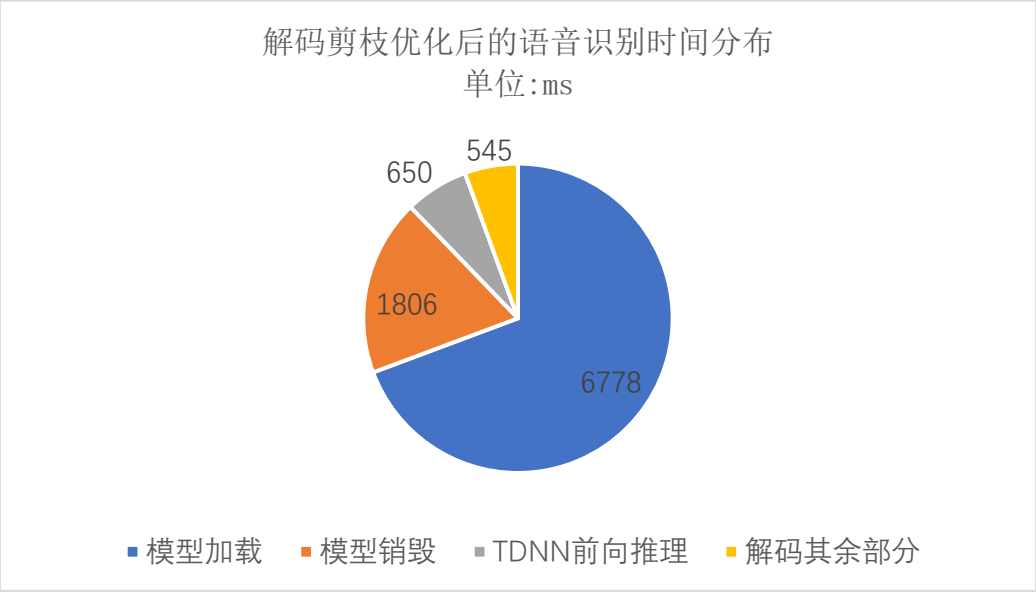

图 5-4 解码剪枝优化后语音识别时间分布

可以看出解码其余流程花费的时间由 2377ms 优化到 545ms，加速比达到 4.36，整个解码过程花费的时间在 1200ms 左右，受限于嵌入式 CPU 的性能，解码的总时间还不满足实时性要求，因此可以采用算力更强的硬件进行加速。

5.2.2.4 基于 TDNN 引擎加速 TDNN 前向推理

本文设计的 TDNN 加速引擎基于运算能力更强的 FPGA 芯片，为语音识别方案扩展了额外的算力。在异构环境下的语音识别流程如图 5-5 所示：

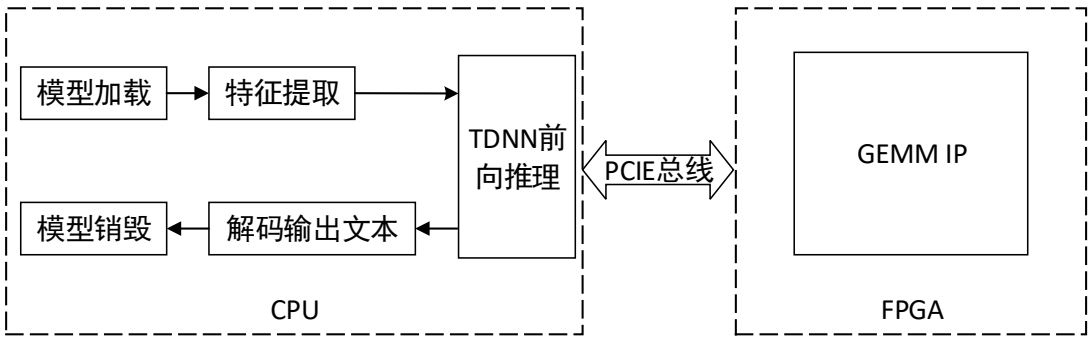

图 5-5 异构计算平台语音识别流程

FPGA 承担了主要的计算负载，对 TDNN 前向推理进行加速，CPU 则负责模型初始化和销毁、特征提取、解码其余部分的流程，语音识别的整个推理过程在两个硬件平台上串行完成，语音特征提取完成后，将特征数据或者中间数据通过 PCI-E 总线传输到 FPGA 端，GEMM IP 完成矩阵向量运算后将结果再通过 PCI-E 总线传回 CPU 端内存，进行后续解码流程。按照 5.2.2.1 节测试了语音识别的时间分布，如图 5-6 所示：

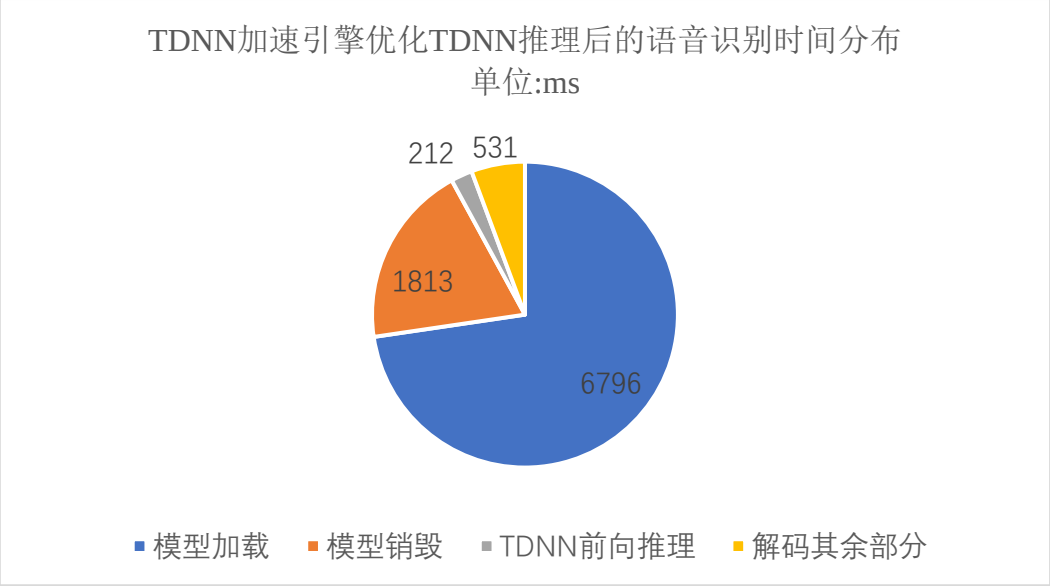

图 5-6 TDNN 引擎优化 TDNN 推理后的语音识别时间分布

TDNN 引擎加速后 TDNN 网络的前向推理时间为 212ms，相比无加速的方案，加速比达到 88.72，TDNN-13 推理速度是 OpenBLAS 方案加速的 3.07 倍。整个解码流程时间在 750ms 左右，达到实时性要求。

5.2.2.5 语音识别 API 设计

综合前面提到的优化策略，最终得到可以在嵌入式平台实时处理的语音识别方案，即在 CPU 端运行模型加载初始化、模型销毁以及核心的解码流程，声学模型的 TDNN 网络可通过 CMake 配置来选择在 CPU 加速或者在 FPGA 加速，在后者加速的语音识别速度更快。虽然模型加载初始化和模型销毁占用了较长的时间，但是模型加载和模型销毁只发生一次，模型一旦在内存中初始化完成，就可以不断地输入语音进行识别。基于以上方案，本文设计实现了语音识别的 C++ API，该 API 不仅提供语音识别的接口，还提供了非固定关键词检索的接口，与固定关键词检索不同的是，非固定关键词检索以语音识别作为前端，可以设置任意类型、任意数量关键词的检索列表，且无需针对新加入的关键词重新训练，C++ API 接口的调用关系和接口说明如下：

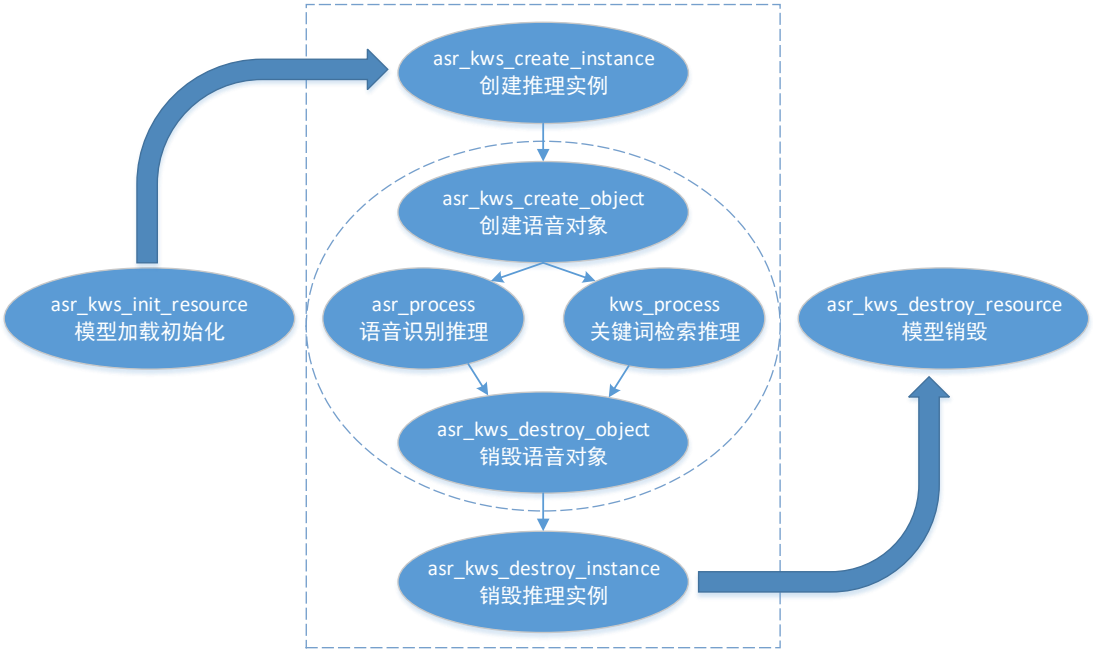

图 5-7 语音识别 API 接口调用关系

1）asr_kws_make_object（创建语音对象）：传入 pcm 格式的数据流、数据长度等，返回语音识别或者关键词检索推理使用的数据对象，关键词检索需要需要额外传入待检索的关键词列表。

2）asr_kws_destroy_object（销毁语音对象）：析构语音识别的数据对象，释放资源。

3）asr_kws_init_resource（模型加载初始化）：将模型加载到内存并初始化共享资源，语音识别和关键词检索可以共享同一份资源。

4）asr_kws_release_resource（模型销毁）：析构共享资源，释放模型占用的内存。

5）asr_kws_create_instance（创建推理实例）：创建一个语音识别或者关键词检索的实例，获得一个句柄，该句柄维系了识别过程中的中间状态变化和中间数据。

6）asr_kws_destroy_instance（销毁推理实例）：析构句柄，释放资源。

7）asr_process（语音识别推理）：利用句柄和语音数据对象完成语音识别的整个流程，返回文本序列。

8）kws_process（关键词检索推理）：利用句柄和语音数据对象完成关键词检索的整个流程，返回文本中检索到的关键词列表。

其中，模型加载初始化和模型销毁在整个调用链中只发生一次，每一个句柄可以被一个线程实体持有，处理任意多条语音数据输入，因此在多线程环境下可

以创建多个句柄，所有的句柄共享同一份模型和共享资源，一旦某个线程持有一个句柄，就可以不断构建语音数据对象并执行语音识别或关键词检索的推理流程。

5.2.3 线程通信单元

线程通信单元基于生产者-消费者模型实现，采用条件变量和互斥锁同步线程的协作，消费者线程睡眠在条件变量上，当生产者线程生产一条数据放入队列后，唤醒消费者线程消费数据。最终销毁生产者-消费者模型时，由主线程发起调用，需要等待消费者处理完队列中所有数据后，主动让消费者线程退出。线程通信单元用于构建 Splice 队列和 Sentence 队列。

5.2.4 录音单元和断句单元

本文设计的语音识别系统并不支持流式语音识别，即只能将一段完整的语音输入到语音识别 API 中进行识别，受音频流长度的影响，处理完一段语音的时间可能较长，而系统的响应时间受限于一段完整语音的处理时延，可能超过用户可接受的范围，因此需要根据说话人说话的停顿间隙将语音进行分段切割，以保证送入语音识别 API 中的语音足够短并且语义也足够完整，在不影响语音识别效果的前提下适量的降低语音识别的端到端时延，以满足实时响应的要求。录音单元内部的线程根据配置可以不断循环录制固定帧数的语音，每一段固定帧数的语音称为一个语音分片 Splice。录音单元作为 Splice 的生产者，不断生产 Splice 放入 Splice 队列，断句单元则作为 Splice 的消费者，其工作策略和流程如图 5-8 所示，语音降噪前端对每一个语音分片进行降噪处理，同时利用 VAD 判定这个 Splice 是不是一个含语音的分片（判断一个分片是否含有语音的规则是：分片中是否存在 VAD 概率超过概率阈值的语音帧），当连续多个 Splice（设定一个阈值）都不含语音，则判定为语音中的停顿，达到语句切分的条件，此时，将之前缓存中累积的 Splice 合成为一个完整的语句 sentence。按以上分析，停顿的间隙时间是单个 Splice 的时间与阈值的乘积，单个 Splice 分片的时间又与录音单元中配置的录音的帧数有关，因此，需要对录音帧数和阈值进行调参，录音帧数和阈值过小可能会把一个完整的句子切分的很零碎，不利于识别，录音帧数和阈值过大会识别不出语音停顿，同时也会显著增加响应时间。在实验中发现，录音帧数调的过低会使得录音的音质效果不佳，原因是录音模块是单线程循环录制，每两段 Splice 之间会有一定的程序间隔，当录音帧数过低时，程序间隔和 Splice 的时间比例会加大，就会导致大量的语音帧被遗漏，影响音质效果（甚至影响到人耳的主管听感）。综上，最终经实验调参得到的最优的录音帧数为 4000，阈值取 1，VAD 的概率阈

值取 0.5。

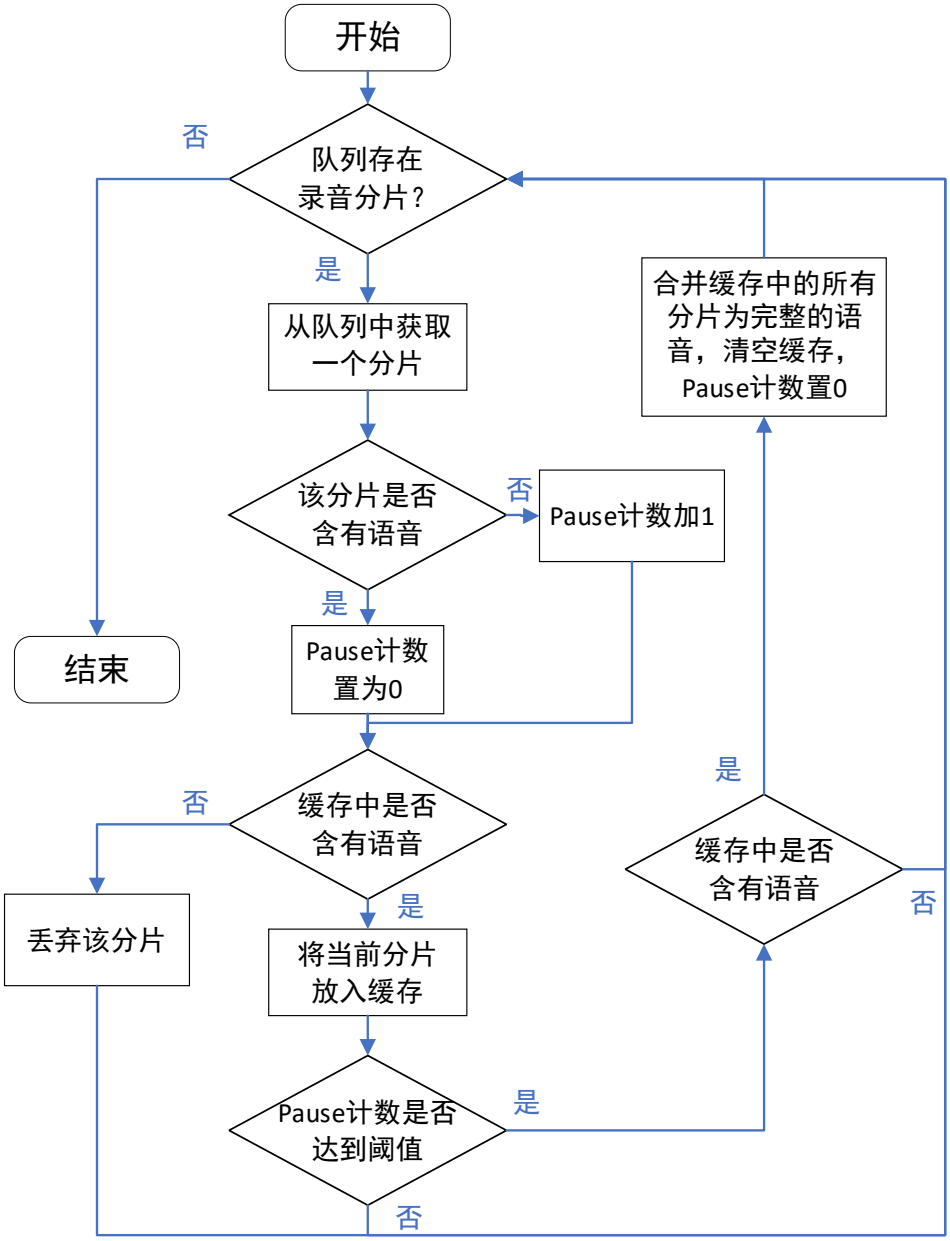

图 5-8 断句策略流程图

5.2.5 语音识别服务和服务管理器

语音服务管理器是一个可扩展的插件服务框架，其内部维护了一个支持高低优先级任务的线程池。可以监管多种语音服务，本文中仅支持语音识别服务和可变关键词检索服务，语音服务的实际计算过程在线程池的某一个工作线程上进行，优先级的作用是保证工作线程优先执行对实时性要求高的服务请求。语音服务管理器规定好了语音服务请求和响应的格式，在语音服务请求中需要指定唯一的请

求标识、请求优先级、请求服务类型、语音数据以及回调函数，请求标识用于将请求和响应对应起来，请求优先级只有两种（HIGH 和 LOW），请求服务类型指明最后申请的服务类型，语音数据的格式由具体的服务规定，回调函数规定了请求执行完毕后以何种方式将服务的结果和响应传回给调用方。所以语音服务的申请是同步的，但是语音服务的执行是异步的。所有语音服务以动态库的形式存放在磁盘上（语音服务插件目录），由语音服务单元在初始化时加载到内存并构建，当需要添加或删除语音服务时，只需要将新的语音服务动态库移入/移出插件目录，无需重新编译语音服务管理器。

语音识别服务是对底层语音识别 API 的封装，虽然断句策略可以根据语音停顿将一段长语音分解为更短的句子，但是在两次停顿之间的连续语音仍然可能过长而导致响应延迟过大，因此语音识别服务提供了对长语音的分段机制，当语音过长时，将 Sentence 切分成多个语音片段 Piece 送入线程池中做语音识别，不同的 Piece 可以在不同的工作线程上并行的识别（多核环境下），因此，需要将 Piece 进行编号，最后将多个 Piece 的识别结果按照编号重组成完整的文本，这一过程在服务中定义了专门的服务线程来处理。语音请求中的回调函数最后也由服务线程生成请求响应之后调用，将语音识别服务的结果传递回调用方。语音分段 Piece 的划分策略如图 5-9 所示：

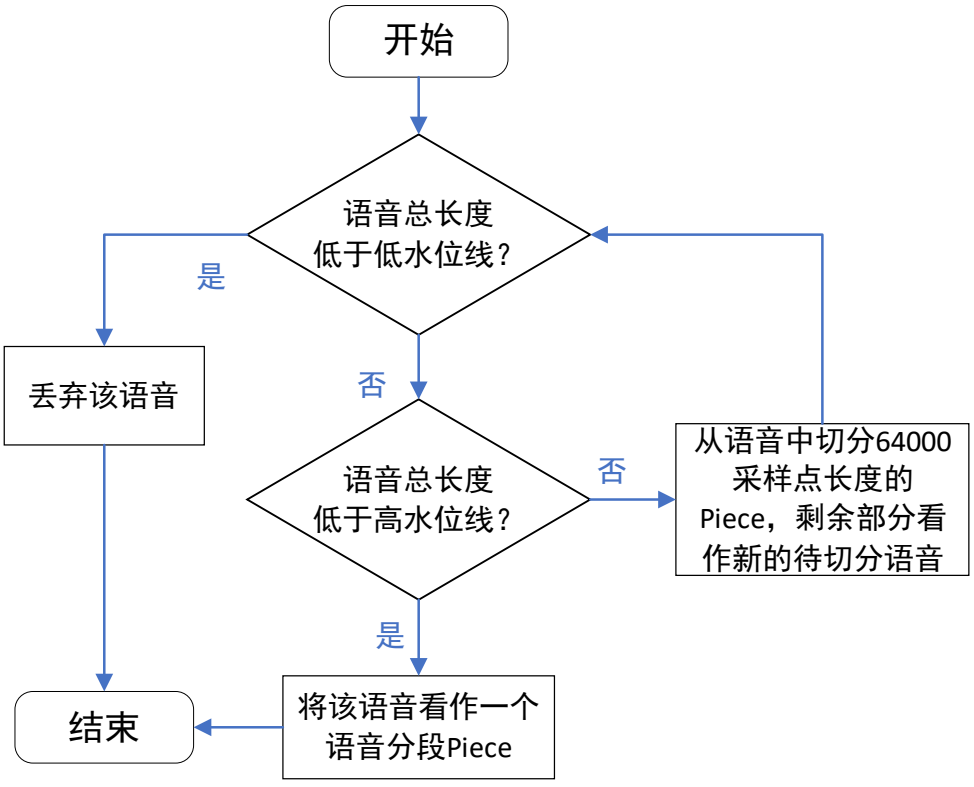

图 5-9 长语音分段策略流程图

1）每一个 Piece 的基础长度为 64000 个采样点，定义了低水位线（16000 采样点）和高水位线（64000 采样点+低水位线）；

2）当整段语音长度低于低水位线则过滤；

3）处于低水位线和高水位线之间则划分到一个 Piece 之中，超过高水位线，则先分一个 64000 长度的 Piece，剩下的语音继续按照第 2、3 点逻辑划分 Piece。

5.2.6 识别单元

断句单元是语句 Sentence 的生产者，识别单元则作为 Sentence 的消费者，从队列中不断获取 Sentence，然后向语音服务管理器申请语音识别服务，以非阻塞的方式从管道获取识别结果，系统可以利用多核 CPU 进一步加速，缩短响应时间，由此带来的一个问题是多核并行处理多条语音引发的文本乱序，即先传入语音服务管理器的请求后接收到结果，识别单元在内部维护了一棵红黑树，用于维护服务请求对应的结果的有序性，算法策略为：

1）每到达一个 Sentence，分配一个唯一且自增的 ID，生成服务请求，并将该 ID 作为 key，将请求注册到红黑树中。

2）每收到一个语音服务传回的结果，将其插入到红黑树中，然后从红黑树中最小 ID 的节点开始遍历，如果请求对应的结果已经得到，则将结果取出在终端上打印对应的文本，直到找到第一个未得到完整结果的节点，结束遍历。

5.3 系统测试和分析

本文基于嵌入式异构计算平台设计实现了面向噪声环境的语音识别 API 以及离线语音识别系统，在 5.1 节的实验环境下，首先对语音识别 API 的接口进行精度和推理性能的测试，然后在目标噪声场景下对离线语音识别系统进行功能性测试，并分析了系统的空间资源占用和功耗。

5.3.1 语音识别 API 接口测试

如上所述，在处理单条语音时，模型加载初始化和模型销毁的时间延长了语音识别的端到端延迟，但是在处理大量语音时，这两个过程的时间被分摊到每一条语音，平均延迟接近语音识别整个解码流程的时间。由于解码剪枝优化牺牲了一定的识别精度，所以采用第三章测试的降噪语音测试集 AISHELL-Denoise-Dev-Test 重新在 API 上测试了 TDNN-13-SVD-M-Denoise-Trigram 模型的精度，语音识别的字错误率如表 5-2 所示：

表 5-2 语音识别 API 字错误率统计

TDNN-13 网络前向推理	解码剪枝	API 语音识别 CER
无加速	采用默认参数	6.81%
无加速	调整为优化后的参数	7.13%
OpenBLAS 加速（float 32）	采用默认参数	6.81%
OpenBLAS 加速（float 32）	调整为优化后的参数	7.13%
TDNN 引擎加速（float 32）	采用默认参数	6.81%
TDNN 引擎加速（float 32）	调整为优化后的参数	7.13%

可以看出，OpenBLAS 和引擎加速没有带来精度损失，解码剪枝优化使 TDNN-13-SVD-M-Denoise-Trigram 的字错误率上升了 0.3%左右，精度损失不大。在保持解码剪枝优化的前提下，选用 6-8s 的语音对 API 的函数接口进行了性能测试，如表 5-3 所示：

表 5-3 语音识别 API 接口性能测试-1

函数接口	TDNN 加速策略	时间（单位：ms）
asr_kws_make_object	——	0.483 ms
asr_kws_destroy_object	——	0.009 ms
asr_kws_init_resource	——	6797.06 ms
asr_kws_destroy_resource	——	1813.72 ms
asr_kws_create_instance	——	0.006 ms
asr_kws_destroy_instance	——	0.004 ms
asr_process	OpenBLAS 加速	1194.13 ms
asr_process	TDNN 引擎加速	752.7 ms
kws_process	OpenBLAS 加速	1198.25 ms
kws_process	TDNN 引擎加速	755.32 ms

5.3.2 系统功能性测试

实验测试时，系统外接 USB 声卡和麦克风，麦克风具备一定的硬件降噪能力，在语音识别模型初始化完成后，提示说话人录入语音，测试时以正常说话交流的语速发声，保持说话时句子之间的停顿。测试语音对应的标签文本如图 5-10 所示：

古时候，有两个兄弟各自带着一只行李箱出远门，一路上，重重的行李箱将兄弟俩都压得喘不过气来，他们只好左手累了换右手，右手累了换左手，忽然，大哥停了下来，在路边买了一根扁担，将两个行李箱一左一右挂在扁担上，他挑起两个箱子上路，反倒觉得轻松了很多。

图 5-10 测试语音标签文本

测试结果如图 5-9 所示：

```
ndsl@sancog:~/ASR-System$ ./ASR-System
ASR-System has been activated, waiting for input...
Mon Mar 14 15:04:43 2022: 古时后
Mon Mar 14 15:04:48 2022: 有两个兄弟各自带着一只行李箱出远门
Mon Mar 14 15:04:49 2022: 一路上
Mon Mar 14 15:04:54 2022: 重重的行李箱将兄弟俩都压得喘不过气来
Mon Mar 14 15:04:56 2022: 他们只好左手累了换右手
Mon Mar 14 15:04:58 2022: 右手累了换左手
Mon Mar 14 15:05:00 2022: 忽然
Mon Mar 14 15:05:01 2022: 大哥停了下来
Mon Mar 14 15:05:03 2022: 在路边买了一根扁担
Mon Mar 14 15:05:07 2022: 将两个行李箱一左一右挂在扁担上
Mon Mar 14 15:05:10 2022: 他挑起两个箱子上路
Mon Mar 14 15:05:12 2022: 反倒觉得轻松了很多
^C
ndsl@sancog:~/ASR-System$
```

图 5-11 系统功能性测试结果

可以看出，系统可以较好的根据语句的停顿间隙进行断句，受限于模型精度，有个别字识别不准确。录入每一句语音之后，系统可以在用户可接受的响应时间内返回识别结果。

5.3.3 空间资源占用

当利用 OpenBLAS 进行加速系统时，设定 OpenBLAS 工作线程数量为 2，FT-2000A/2 CPU 利用率达到 196%（两个核基本都满载），CPU 端内存利用率达到 22.3%。当利用 TDNN 引擎加速时，FT-2000A/2 CPU 利用率达到 67%，CPU 端内存利用率达到 26.7%。FPGA 资源占用情况如表 5-5 所示:

表 5-5 FPGA 资源占用情况

资源	BRAM_18K	DSP48E	FF	LUT
资源消耗	2175 (74%)	1804 (50%)	485186 (56%)	355224 (82%)

5.3.4 系统功耗

本文利用功耗插座测试了 CPU 端主板的功耗，利用 Vivado 自带的功耗分析工具对 FPGA 芯片的功耗（Total On-Chip Power）进行估计，然后可以得到系统总功耗的估计，为了保证 FT-2000A/2 CPU 能够正常与国威 FPGA 板卡通信，CPU 端

和 FPGA 板卡都是独立供电的，当 FPGA 从板载 Flash 中加载好 bit 流之后，才启动 CPU 端的银河麒麟操作系统，当采用 OpenBLAS 加速时，不需要使用 FPGA，仅测试 CPU 端的功耗，当采用 TDNN 引擎加速时，还需要考虑 FPGA 芯片的功耗，如图 5-10 所示：

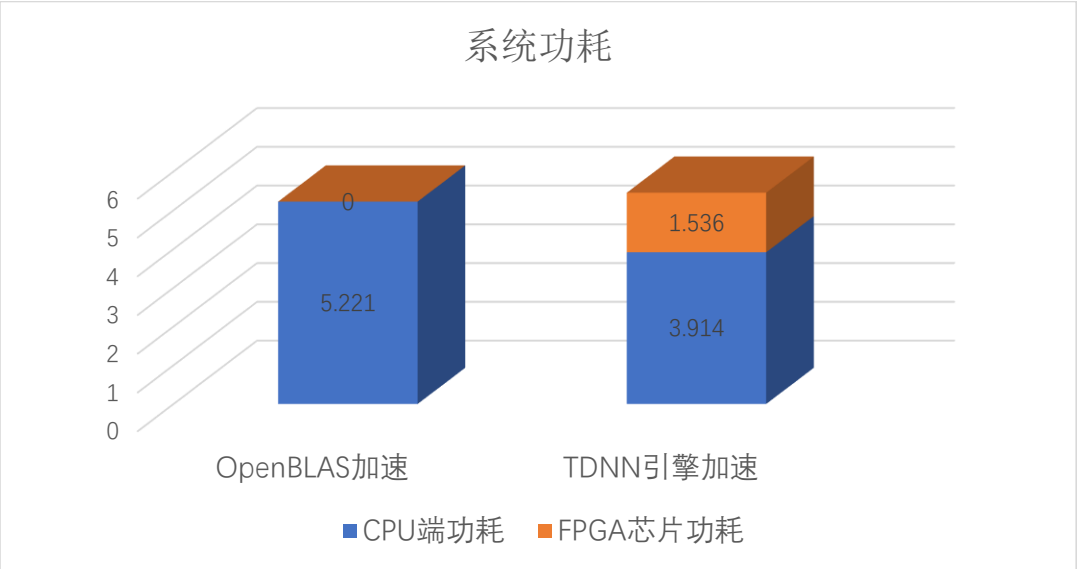

图 5-12 系统功耗分析

可以看出，不论采用哪种加速方案，系统总功耗在 5-6 W 之间，功耗较低，因此系统在离线远端环境下可以工作很长时间。

5.4 本章小结

本章主要介绍了嵌入式语音识别 API 的设计和优化，并结合优化策略设计实现了 API 的函数接口，考虑到优化策略可能带来的精度损失，重新基于 AISHELL-Denoise-Dev-Test 测试语音识别 API 的字错率，优化后 TDNN-13-SVD-M-Denoise-Trigram 在 API 上的字错率上升了 0.3%，达到 7.13%。采用 6-8s 的语音测试了各个 API 函数接口的时间性能，经测试，采用 OpenBLAS 加速语音识别接口推理时间在 1200ms 左右，采用 TDNN 引擎加速语音识别接口推理时间在 750ms 左右，语音识别推理性能满足实时性的要求。本章还利用 API 搭建了一个基于嵌入式平台的离线语音识别系统，在目标噪声环境下对该系统进行了功能性测试，同时测试了系统的空间资源占用和系统功耗，经实验分析，系统的运行功耗在 5 - 6W 左右，利用电池供电可以满足长时间的使用。

第六章 全文总结与展望

6.1 全文总结

本文基于 DNN-HMM 语音识别框架设计了适用于嵌入式平台的、面向噪声的低功耗离线近场语音识别方案。面向目标噪声环境，本文一方面设计实现了轻量化的语音降噪前端，另一方面利用 DNN 本身的抗噪性，采用大量的含噪声语音微调语音识别模型，提升了语音识别在目标噪声环境的鲁棒性。面向嵌入式平台，本文设计了轻量化 TDNN-13-SVD-M-Denoise 作为声学模型的 DNN 网络，采用了奇异值分解技术压缩了模型的权重，结合语言模型 Trigram，最终得到轻量化语音识别模型 TDNN-13-SVD-M-Denoise-Trigram。为了保证系统在嵌入式平台运行的实时性，本文采用两种加速策略对 TDNN-13 模型的前向推理进行加速，一方面利用开源的 OpenBLAS 矩阵向量运算库加速 TDNN 底层的运算，另一方面设计实现了基于异构环境的 TDNN 加速引擎，对 TDNN 底层的矩阵运算进行加速，经实验对比，后者取得了更好的加速效果。为了提升语音识别遍历解码图的速度，调整了解码图的剪枝系数等参数，进一步优化了系统的端到端延时。基于以上的语音识别方案，本文基于 Kaldi nnet3 封装实现了嵌入式语音识别的 C++ API，该 API 同时提供语音识别和关键词检索的接口，可以扩展到多线程环境中使用，共享同一份模型资源。最后，本文基于语音降噪前端和语音识别 API 搭建了一个面向目标噪声环境的离线近场语音识别系统，该系统可以对静音音频和纯背景噪声音频进行过滤，且能够根据语音中的停顿对语句进行分割，系统在 FT-2000A/2 CPU 和国威 FPGA 板卡搭建的异构计算平台上测试，达到了实时性要求，系统的运行功耗较低，即使在离线远端环境下也能满足长时间的使用要求。

6.2 后续工作展望

本文的工作对语音识别在嵌入式平台的移植做了一个较好的开端。未来的工作将从以下几个方面继续深入展开：

1）本文的语音降噪前端对于语音识别在噪声环境的识别效果提升不大（特定噪声环境下字错误率仅下降 0.3-0.5%），一方面原因是语音降噪在抑制噪声的同时，也破坏了音频数据本身，语音失真较多，另一方面原因是语音降噪前端和语音识别的训练过程是分离的，这就使得面向噪声的语音识别无法得到一个全局最佳的效果。后续工作会考虑用语音识别的字错误率来作为指导降噪算法训练时梯

度优化的方向，或者直接考虑从语音识别模型自身结构出发设计出抗噪能力更强的网络模型。

2）本文研究基于DNN-HMM框架的语音识别方案，主要是考虑到目前DNN-HMM 在工业界已经十分成熟，受限于 HMM 的假设条件，DNN-HMM 的建模能力有限，语音识别精度相比现有的端到端语音识别方案还存在一定差距，因此下一步工作将研究端到端的嵌入式语音识别方案。

3）本文设计的 TDNN 加速引擎运算能力较强，但是仅仅只对 TDNN 底层的矩阵乘法进行加速，这意味着每一条语音的识别都会将TDNN模型权重由CPU端传输到 FPGA，PCI-E 总线的压力较大，经实验验证数据传输的时间占用整个 TDNN 推理时间的 40%，显著增加了端到端的处理延时，下一步工作将在 FPGA 上设计整个 TDNN 网络的推理引擎，这样模型权重只需要传输一次，优化数据传输的时间代价。

致 谢

时光荏苒，研究生三年就在教室，实验室和图书馆悄然度过。如今的我就快要从学校这个象牙塔走出去，真正面临社会的挑战和困难，感谢岁月给我带来的智慧，感谢这三年的青春和汗水，让我在奋斗中历练和成长自我。在此我必须由衷地感谢在研究生期间陪我成长的老师、父母和同学。

首先，我要感谢段翰聪教授对我的指导以及在求职上对我的帮助，研究生期间，我们开展了很多次项目组会，段老师总是在会议上教我们从如何从系统的角度去架构项目的框架，同时也教我们在如何在新的领域构思一个实际的应用场景，并解决其中的关键问题。

我还要感谢和我同实验室同届的所有同学们，为了找到满意的工作，我们举办了许多场主题分享交流会，大家根据自己擅长的知识或技术精心准备汇报的内容和问题，每一次的交流会都让我受益匪浅。无论是生活还是学习，他们总是能够给我一些中肯的建议和想法。

我还要感谢我的舍友，遇到困难时，我们互相开导和鼓励彼此。我们也喜欢调侃一些有趣的事情，感谢他的陪伴，我的研究生生活才会如此充实和快乐。

我还要感谢我的家人能给我一个幸福安定的家，他们是我最坚实的后盾，让我随时都能保持轻松愉快的心情去做好科研工作和项目。

最后要特别感谢专家、教授、学者在百忙之中审阅本文，文中的不足之处，请各位老师多多指正，在此表达衷心的感谢。

参考文献

[1] A Hannun∗, Case C, Casper J, et al. Deep Speech: Scaling up end-to-end speech recognition[J]. Computer Science, 2014, 1412.5567.

[2] Davis K H. Automatic Recognition of Spoken Digits[J]. J.acoust.soc.america, 1952, 24(6): 669.

[3] Denes P. The design and operation of the mechanical speech recognizer at University College London[J]. Journal of the British Institution of Radio Engineers, 1959, 19(4): 219-229.

[4] Dtw C. Information Retrieval for Music and Motion. Dynamic Time Warping[J]. Springer Berlin Heidelberg, 2007, 10.1007.

[5] Sakoe H, Chiba S. Dynamic Programming Algorithm Optimization for Spoken Word Recognition[J]. IEEE Transactions on Acoustics Speech and Signal Processing, 1978, 26(1): 43-49.

[6] Velichko V M, Zagoruyko N G. Automatic Recognition of 200 Words[J]. International Journal of Man-Machine Studies, 1970, 2(3): 223-234.

[7] Nasrabadi, Feng. Vector quantization of images based upon the Kohonen self-organizing feature maps[C]. IEEE International Conference on Neural Networks. IEEE, 1988: 1-108.

[8] Baum L E. An inequality and associated maximization technique in statistical estimation for probablistic functions of Markov processes[J]. Inequalities, 1972, 3.

[9] Rabiner L R. A tutorial on hidden Markov models and selected applications in speech recognition[J]. Proc IEEE, 1989, 77.

[10] Lee K F, Hon H W. An overview of the SPHINX speech recognition system[J]. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1990, 38(1): 35-45.

[11] Young S. The HTK Hidden Markov Model Toolkit: Design and Philosophy[J]. CUED Technical Report F_INFENG/TR152, Cambridge University Engineering Dept, 1993, 2: 2-44.

[12] Hinton G E, Salakhutdinov R R. Reducing the Dimensionality of Data with Neural Networks[J]. Science, 2006, 313: 504-507.

[13] Hinton G E, Osindero S, Teh Y W. A Fast Learning Algorithm for Deep Belief Nets[J]. Neural Computation, 2006, 18(7): 1527-1554.

[14] Mohamed A R, Dahl G, Hinton G, et al. Deep Belief Networks using discriminative features for phone recognition[C]. Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, 2011: 5060-5063.

[15] Hinton G, Deng L, Yu D, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups [J]. IEEE Signal Process . Mag, 2012, 29(6): 82-97.

[16] Li J, Zhang H, Cai XY, et al. Towards end-to-end speech recognition for Chinese Mandarin using long short-term memory recurrent neural networks[C]. Annual Conference of the International Speech Communication Association, 2015: 3615-3619.

[17] Chorowski J, Bahdanau D, Serdyuk D, et al. Attention-Based Models for Speech Recognition[J]. Computer Science, 2015, 10(4):429-439.

[18] Vaswani A, Shazeer N, Parmar N, et al. Attention Is All You Need[J]. Computer Science, 2017, 1706.03762.

[19] Dong L, Shuang X, Bo X. Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition[C]. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2018: 5884-5888.

[20] Watanabe S, Hori T, Karita S, et al. ESPnet: End-to-End Speech Processing Toolkit[J]. Computer Science, 2018, 1804.00015.

[21] Povey D, Ghoshal A, Boulianne G, et al. The Kaldi Speech Recognition Toolkit[J]. Idiap, 2012.

[22] Ravanelli M, Parcollet T, Bengio Y. The PyTorch-Kaldi Speech Recognition Toolkit[J]. Computer Science, 2019, 1811.07453.

[23] 王作英, 肖熙. 基于段长分布的 HMM 语音识别模型[J]. 电子学报, 2004, 32(1): 46-49.

[24] Wang D, Zhang X. THCHS-30: A Free Chinese Speech Corpus[J]. Computer Science, 2015, 1512.01882.

[25] Bu H, Du J, Na X, et al. AISHELL-1: An Open-Source Mandarin Speech Corpus and A Speech Recognition Baseline[C]. O-COCOSDA, 2017: 1-5.

[26] Du J, Na X, Liu X, et al. AISHELL-2: Transforming Mandarin ASR Research Into Industrial Scale[J]. 2018, 1808.10583

[27] Amodei D, Ananthanarayanan S, Anubhai R, et al. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin[J]. Computer Science, 2015, 1512.02595.

[28] Zhang WD, Zhang F, Chen W, et al. Fault state recognition of rolling bearing based fully convolutional network[J]. Computing in Science & Engineering, 2019, 21(5): 55–63.

[29] Zhang S, Lei M, Yan Z, et al. Deep-FSMN for Large Vocabulary Continuous Speech Recognition[C]. International Conference on Acoustics, Speech and Signal Processing. Canada: IEEE, 2018: 5869-5873.

[30] Pertilä P, Parviainen M. Time difference of arrival estimation of speech signals using deep neural networks with integrated time-frequency masking[C]. 2019 IEEE International Conference on Acoustics, Speech and Signal Processing. Brighton: IEEE, 2019: 436–440.

[31] Moore AH, Xue W, Naylor PA, et al. Noise covariance matrix estimation for rotating microphone arrays[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2019, 27(3): 519–530.

[32] Park J, Chang JH. State-space microphone array nonlinear acoustic echo cancellation using multi-microphone near-end speech covariance[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2019, 27(10): 1520–1534.

[33] Ephraim Y, Malah D. Speech enhancement using a minimum mean-square error log-spectral amplitude estimator[J]. IEEE Transactions on Acoustics Speech & Signal Processing, 1985, 33(2): 443-445.

[34] Pascual S, Bonafonte A, J Serrà. SEGAN: Speech Enhancement Generative Adversarial Network[J]. Computer Science, 2017, 1703.09452.

[35] Qian YM, Bi MX, Tan T, et al. Very deep convolutional neural networks for noise robust speech recognition[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2016, 24(12): 2263–2276.

[36] Tan T, Qian YM, Hu H, et al. Adaptive very deep convolutional residual network for noise robust speech recognition[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2018, 26(8): 1393–1405.

[37] Sriram A, Jun H, Satheesh S, et al. Cold fusion: Training Seq2Seq models together with language models[J]. Computer Science, 2017, 1708.06426.

[38] Zhou P, Yang WW, Chen W, et al. Modality attention for end-to-end audio-visual speech recognition[C]. IEEE International Conference on Acoustics, Speech and Signal Processing. Barcelona: IEEE, 2019: 6565–6569.

[39] Makino T, Liao H, Assael Y, et al. Recurrent neural network transducer for audio-visual speech recognition[C]. IEEE Automatic Speech Recognition and Understanding Workshop. Singapore: IEEE, 2019: 905–912.

[40] Chin H, Kim J, Kim I. Realization of speech recognition using DSP(Digital Signal Processing)[C]. IEEE International Symposium on Industrial Electronics. Proceedings. ISIE, 2001: 1760-1770.

[41] Yuan M, Tan L, Ching P C. Speech recognition on DSP: issues on computational efficiency and performance analysis[J]. Microprocess. Microsystems, 2006, 30(3): 155-164.

[42] Manikandan J, Venkataramani B, Girish K, et al. Hardware Implementation of Real-Time Speech Recognition System Using TMS320C6713 DSP[C]. International Conference on VLSI Design. India, 2011: 250-255.

[43] Amudha V, Venkataramani B, et al. Software/Hardware co-design of HMM based isolated digit recognition system[J]. 2009, 4(2): 154-159.

[44] Lee M, Hwang K, Park J, et al. FPGA-based Low-power Speech Recognition with Recurrent Neural Networks[J]. Computer Science, 2016: 1610.00552.

[45] Hasim Sak, Andrew W. Senior A, et al. Long short-term memory recurrent neural network architectures for large scale acoustic modeling[C]. Conference of the international speech communication association, 2014: 338-342.

[46] Waibel A, Hanazawa T, Hinton G, et al. Phoneme recognition using time-delay neural networks[J]. Readings in Speech Recognition, 1990, 1(3): 393-404.

攻读硕士学位期间取得的成果