Embedded Technology and Intelligent Systems 嵌入式技术与智能系统,2024, 1(1), 40-49
Published Online August 2024 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2024.11005

Hans汉斯

在MCU端部署GRU模型实现鼾声检测

许鹏*，王印鑫，宋岩

恩智浦半导体有限公司，荷兰埃因霍温

收稿日期：2024年7月4日；录用日期：2024年7月25日；发布日期：2024年8月5日

摘要

本研究旨在开发一种在资源受限的微控制器单元(MCU)上运行的方法，用以进行鼾声检测。不同于使用CNN进行声音检测的方式，我们采用门控循环单元(GRU)模型以对音频数据进行处理和分析。通过采用优化模型结构、模型量化等常用的模型优化方式，我们最终成功将GRU模型适配到低功耗的MCU平台，使其能够在不依赖外部计算资源的情况下，独立完成端侧的鼾声检测任务，无需联网。实验结果表明，该模型在保持较高准确性的同时，能够有效降低系统算力需求，满足移动健康监测设备的实时性与便携性要求。这一研究为鼾症患者的持续监测和睡眠健康管理提供了一种新的解决方案，同时也拓展了深度学习在嵌入式系统中的应用前景。

关键词

微控制器单元(MCU)，门控循环单元(GRU)，鼾声检测，人工智能应用

Deploying GRU Model on MCU for Snore Detection

PengXu*, YinxinWang, YanSong

NXP Semiconductors, Eindhoven, The Kingdom of the Netherlands

Received: Jul. 4th, 2024; accepted: Jul.25th, 2024; published: Aug.5th, 2024

Abstract

This study aims to develop a method running on a resource-constrained microcontroller unit (MCU) for snore detection. Unlike the approach of using CNNs for sound detection, we employ the Gated Recurrent Unit (GRU) model to process and analyze audio data. By adopting common model

*通讯作者。

文章引用:许鹏, 王印鑫, 宋岩.在 MCU 端部署 GRU 模型实现鼾声检测[J].嵌入式技术与智能系统,2024,1(1):40-49.DOI: 10.12677/etis.2024.11005

optimization techniques such as optimizing the model structure and model quantization, we ultimately succeeded in adapting the GRU model to a low-power MCU platform. This allows it to independently perform snore detection tasks on the edge without relying on external computing resources and without the need for internet connectivity. Experimental results indicate that while maintaining high accuracy, the model can effectively reduce the system's computational power requirements, meeting the real-time and portability needs of mobile health monitoring devices. This research provides a new solution for the continuous monitoring and sleep health management of patients with snoring disorders and also expands the application prospects of deep learning in embedded systems.

Keywords

Microcontroller Unit (MCU), Gated Recurrent Unit (GRU), Snore Detection, Artificial Intelligence Application

1. 引言

提起打鼾，可能大家都不会陌生，打鼾是睡眠呼吸障碍的一种常见表现，普遍存在于全球各地，影响着数百万人的生活质量和健康。可以说，打鼾是一个极端令人烦恼的生理现象，验证影响睡眠质量。对于许多患者而言，及时检测打鼾并采取相应措施至关重要。然而，尽管市场上已有多种方法用于鼾声检测，包括使用各种监测设备和进行复杂的睡眠研究，但这些方法大多数依赖于昂贵的设备和专业的操作，这在资源有限的环境中并不实用，也难以普及到广大需要的人群，就显得有些空中楼阁了。因此，开发一种简便、低成本且可靠的鼾声检测方法具有重要的实际意义，可以极大地提高普及度以及大众的认可读[1]。

近年来，人工智能在处理和分析声音数据方面取得了显著进展，尤其是深度学习技术已被成功应用于各种声音识别任务，例如语音识别和异常声音检测。正是这些技术的进步，为更加准确、可靠的鼾声检测提供了新的可能性。在 MCU 单元上实现深度学习模型的尝试，为端侧的计算和边缘智能开辟了新的可能性，使得在低成本的硬件设备上部署复杂的算法变得可行[2]。

本研究旨在探索使用深度学习架构——门控循环单元，并对其进行轻量级设计，以实现对鼾声的高效检测。GRU 模型能够捕捉到时间序列数据中的依赖关系，如鼾声的音频模式，使得其更加适合实时处理和分析。选择 MCU 单元作为运行平台，不仅因为其成本低和易于集成到便携式设备中，而且因为它适合于在资源受限的环境中操作，满足在各种环境下对鼾声进行实时监测的需求。

通过将 GRU 模型适配到基于 MCU 的呼吸辅助装置，我们可以将鼾声检测系统带入患者的日常生活环境中，从而实现实时、非侵入性的监测，提供即时反馈和必要的医疗干预。此外，该方法的发展也将推动深度学习在嵌入式系统中的应用，特别是在处理生理信号和促进移动健康监测方面，为广泛的生物医学应用开辟新的道路。本文接下来将详细介绍 GRU 模型的训练和调优过程，以及如何实现在微控制器单元上部署该模型。

2. GRU 模型简介

GRU 全称 Gated Recurrent Unit，直译叫做门控回归单元。是一种用于处理序列数据的循环神经网络(RNN)模型。RNN 指代递归神经网络(Recurrent Neural Network)，是一种常用于处理序列数据的神经网络架构[3]。与传统的前馈神经网络不同，RNN 具有记忆功能，非常适合于对序列数据进行处理，这是通过计算和维护状态变量来实现的。相较于传统的 RNN，GRU 具有更简单的结构和更高效的训练方式。它通过一种称为门控机制的方式来控制信息的流动，包括更新门和重置门。这些门控机制有助于模型决定在每个时间步上应该记住什么信息，以及应该忘记什么信息，从而更好地处理长序列数据。GRU 模型包括一个更新门和一个重置门。更新门决定了新的输入应如何保留，而重置门则决定了状态信息应如何忽略[4]。

图 1 是一个 GRU 的框图说明。

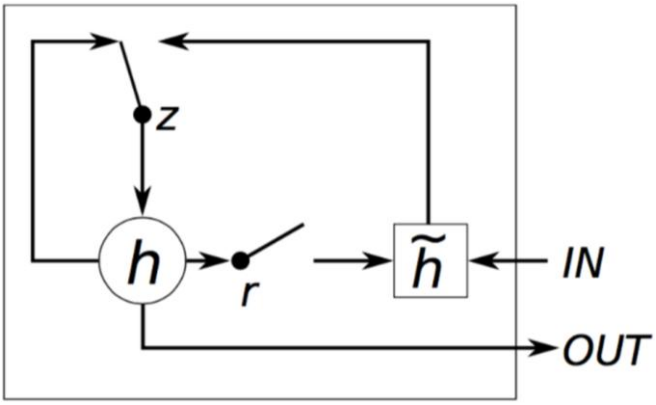

Figure1. GRU model. The switch is soft, is a weighted sum of the inputs, the sum of weights is 1
图 1. GRU 模型示意图。开关其实是软的，是彼此两个输入的带权和，且彼此权重的和为 1

接下来介绍下这两个门控单元的工作流程。首先是重置门 r，负责对状态量 H 进行逐点相乘，记作 $r \cdot H$ 。r 和 H 都是相同维度的向量，相乘的结果起到对记忆的信息筛选淡出的效果。之后将输入和计算后的状态结合成为新的候选状态 $\tilde{h}$ 。最终将结果进行更新，更新就是通过这个 z 门。计算方式是 $z \cdot H_{t-1} + (1-z) \cdot \tilde{H}$ ，也就是说将上一次计算的状态与当前的候选状态 $\tilde{H}$ 进行加权求和，得到本次的最终状态，当然也就是下一次的 H_{t-1} 。如此循环往复，直到训练结束，得到最终的模型[5]。

此外，与长短时记忆网络(LSTM)相比，GRU 模型减少了参数数量，一些情况下更容易训练，并且在计算上也更高效。这使得 GRU 成为处理序列数据时的一种流行选择。由于 GRU 模型的门控结构，它具有一定的记忆能力，能够更好地捕捉时间序列中的重要特征，并且相对于传统的 RNN 模型，GRU 模型在一定程度上缓解了梯度消失的问题，从而更适合处理长序列数据。

3. 在 PC 上进行 GRU 模型的训练与测试

3.1. GRU 模型训练

接下来，我们来看看如何训练一个 GRU 模型，模型训练平台选用 Keras，有需要的读者请自行安装 Keras 开发工具。我们这里主要给大家介绍模型搭建部分，这里假设我们的鼾声检测数据集已经准备好了，并将其划分为训练和测试数据集，分别命名为：(x_train, y_train), (x_test, y_test)。直接进行模型构建与训练：

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import GRU, Dense
# 构建 GRU 模型
model = Sequential()
model.add(GRU(128, input_shape=(64, 64), stateful=False, unroll=False))
model.add(Dense(2, activation='softmax'))
# 编译模型
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# 模型训练
model.fit(x_train, y_train, batch_size=128, epochs=10, validation_data=(x_test, y_test))
```

先看下这里所指定的输入，(64, 64)。怎么理解呢？上文讲过 GRU 模型实际上是每次只输入一个维度的数据，即数据应该是(1, 64)。这里好像有点出入。这就不得不解释下，这里的两个 64 分别代表什么。前一个 64 表示 timestep，第二个 64 表示音频特征的维度。那么就可以理解为：一次训练，我们是一次性对 64 组连续的语音特征进行训练，这样能够更好的处理他们之间的相关性。这里的音频特征是由一段固定长度时域数据提取而来。

这里需要注意的是，GRU 模型构建的时候，有两个参数，分别是 stateful 以及 unroll，这两个参数是什么意思呢：

stateful 参数：当 stateful 设置为 True 时，表示在处理连续的数据时，GRU 层的状态会被保留并传递到下一个时间步，而不是每个 batch 都重置状态。这对于处理时间序列数据时非常有用，例如在处理长序列时，可以保持模型的状态信息，而不是在每个 batch 之间重置。默认情况下，stateful 参数为 False。需要注意的是，若设置 stateful 为 True，需要手动管理状态的重置。

unroll 参数：默认情况下，unroll 参数为 False。当 unroll 设置为 True 时，表示在计算时会展开 RNN 的循环。通常情况下，对于较短的序列，unroll 设置为 True 可以提高计算速度，但对于较长的序列，可能会导致内存消耗过大，例如，上述模型如果 unroll=True，会展开 64 次相同的 GRU 所执行的操作，模型中包含大量节点。

我们再看看 stateful 参数。鼾声信号其实是时域音频信号，每个时间步之间实际上是存在有时间前后关系的，即前后的音频片段之间会组成一段完整的音频数据。因此，在实际使用时若要一次只处理单个时间步的数据，就需要设置 stateful 为 True。

3.2. GRU 模型测试

上面通过训练，得到了一个庞然大物。之所以庞大，是因为我们采用了 unroll 参数将其展开，发现模型其实由很多相同的模型块组成，GRU 模块如图 2 所示。

而这个模型块的数量和所设置的时间步是相关的。这里我们要给大家带来 GRU 模型的一个特殊性，那就是实际推理所用到的模型和训练时是不一样的。具体而言，训练时候，我们为了让 GRU 模型能够更好的学习相邻音频之间的关系，需要将模型输入设置为(64, 64)。而实际测试时候，由于模型已经具备了处理相邻音频的能力，我们也就不再需要将模型输入设置为(64, 64)，而是将其设置为(1, 64)，即每次只输入给模型一条音频特征，同时我们设置 stateful=True(请注意，与推理时相反，训练的时候 stateful 为 False！)，即每次模型会记录下当前所计算出来的状态信息，将其用于下一次的计算。这样化整为零，使输入减少到了以前的 1/64。隐藏状态的计算量也就按比例一起减少了。但要注意的是，最后一步用于把隐藏状态映射成输出的 Dense 层的计算次数却按比例增加了。不过，与减少的执行 GRU 模块的计算相比，

增加的计算比减少的计算要少得多。

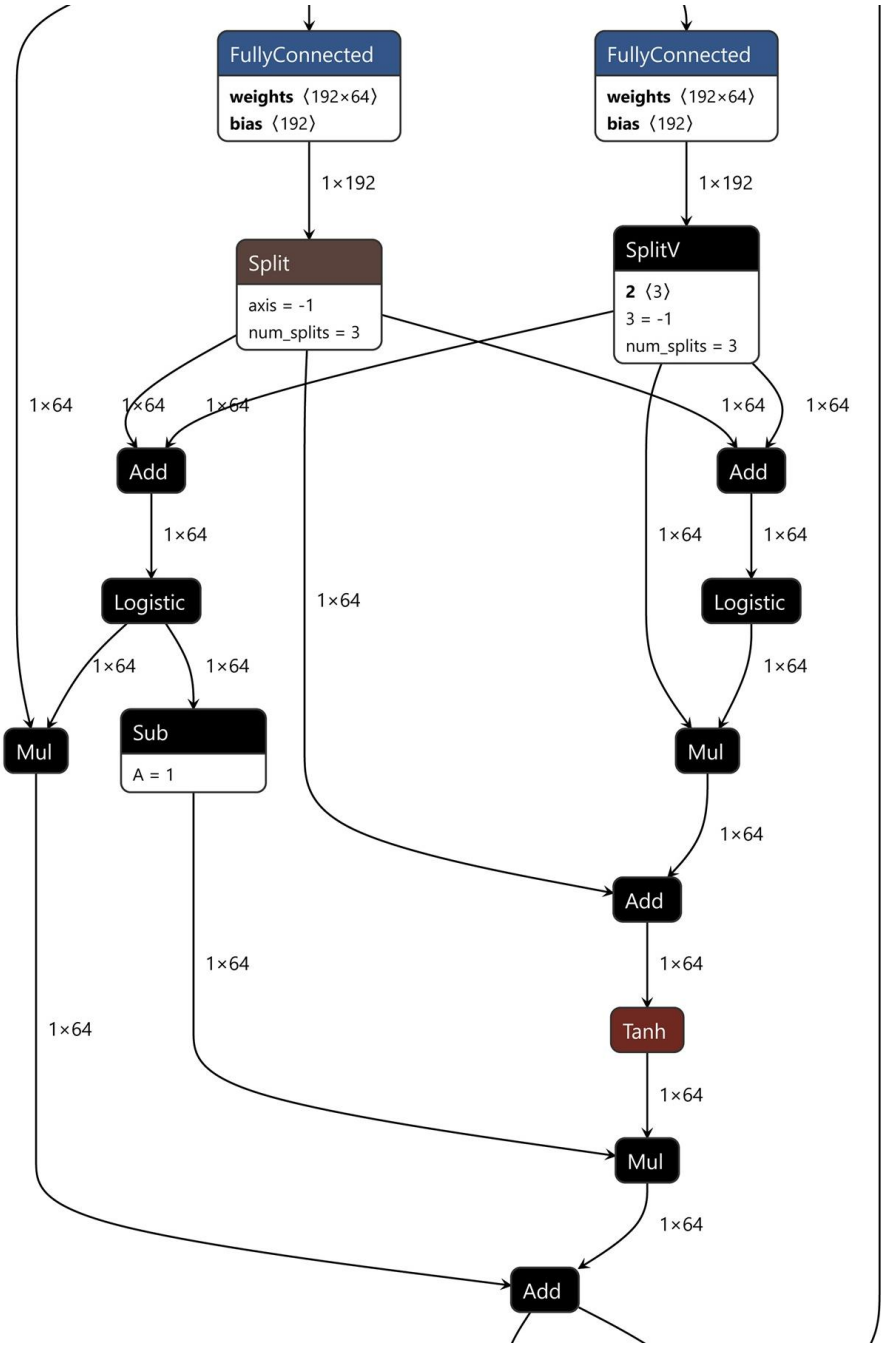

Figure 2. GRU model block
图 2.GRU 模型块

用于推理的模型构建代码修改如下：

```
# 构建新模型
new_model = Sequential()
new_model.add(GRU(1, batch_input_shape=(1, 1, 64), unroll=True, stateful=True))
new_model.add(Dense(2, activation='softmax'))
```

new_model.set_weights(model.get_weights())

注意到，最后一行，有一个 set_weights 函数，目的是将之前我们所训练出来的模型参数配置给这个新模型。看一下模型的变化，如图 3 所示。

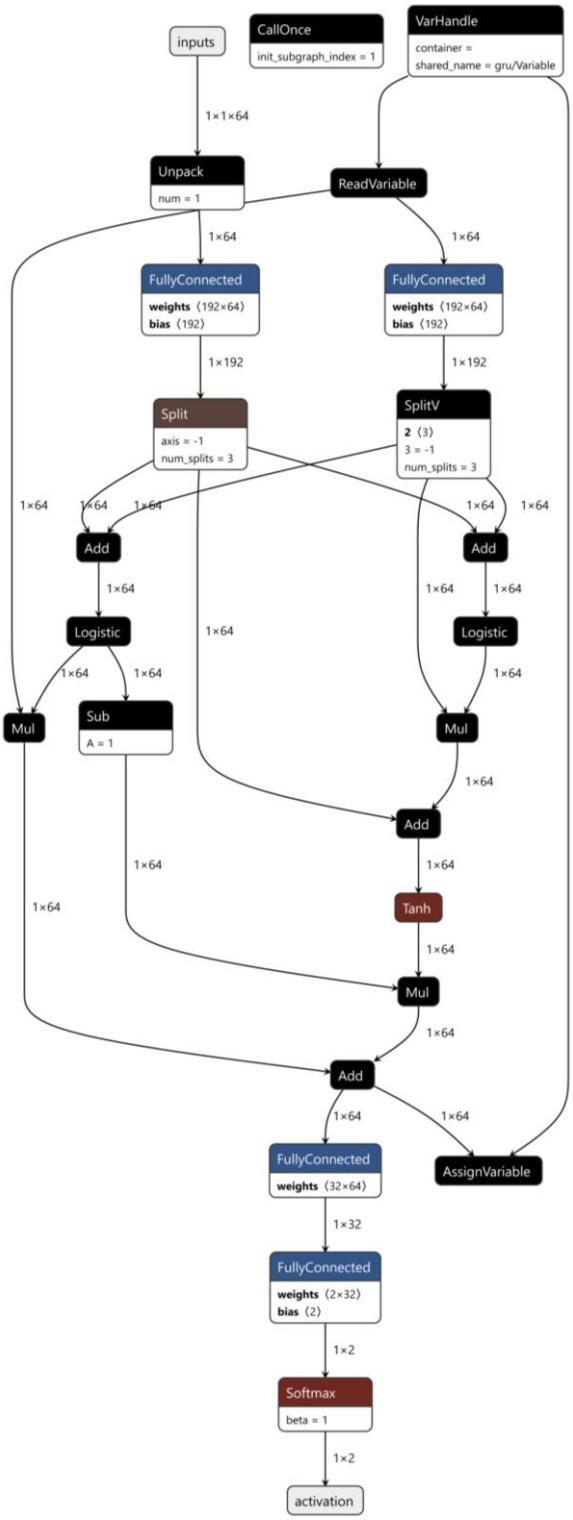

Figure 3. GRU model with time-step 1
图 3.时间步为 1 的 GRU 模型

图中红圈位置有一个 AssignVariable 节点，它的作用就是将本次所运行出来的状态量进行保存，以供下次推理使用。

3.3. GRU 模型量化

在模型部署前需要对模型进行量化，量化的原理就是将用浮点数表示的模型权重，映射为(−128, 127)之间的 8 位整数[6]。这样做的好处是可以减小模型尺寸，可以将其缩减到之前的近 1/4。再者，MCU 平台提供了对于具有 8 位整数类型权重的模型的推理加速能力。不过弊端就是，精度可能会有所下降，所幸的是 GRU 模块大量使用有上、下确界的 Sigmoid 和 Tanh 激活函数，有助于估计中间结果的值域。模型量化可以使用 NXP 公司所开发的 eIQ 工具，打开 eIQPortal 软件，找到 MODELTOOL，如图 4 所示：

Figure 4.eIQ Portal tools
图 4. eIQ Portal 工具

用它打开我们刚才重新制作的模型，点击左上角的 Convert 选择转换工具为 Tensor Flow Lite，并勾选 EnableQuantization，如图 5 所示。

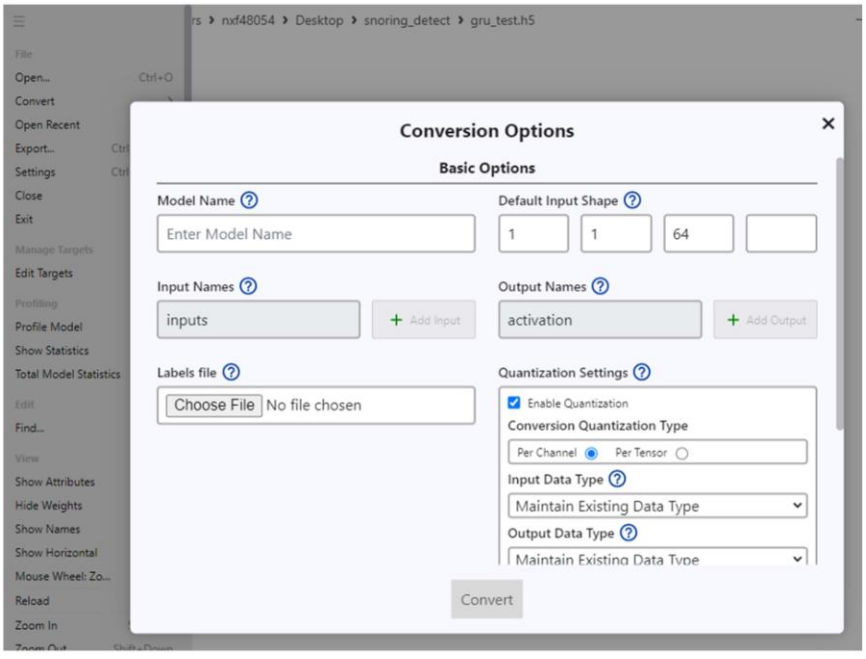

Figure 5. Model quantization
图 5.模型量化页面

点击 Convert 即可获得量化后的模型。

4. GRU 模型的在端侧的部署

我们选用了 NXP 的 i.MX RT1060 跨界 MCU 部署 GRU 模型。

4.1. i.MX RT1060 平台简介

i.MX RT1060 跨界 MCU 拥有 600MHz 的主频，可以提供卓越的计算能力、多种媒体功能以及实时功能，易于使用。i.MX RT1060 采用主频达 600MHz 的 Cortex®-M7 同时提供一流的安全保障。i.MX RT1060 MCU 支持宽温度范围，适用于消费电子、工业和汽车市场。拥有高达 1MB 片上内存，包括 512KB 的 FlexRAM，能被灵活的分配为 ITCM/DTCM/OCRAM。集成了高级 DCDC 和 LDO 的电源管理模块，简化电源序列。同时提供了多种内存接口，包括 SDRAM，RAWNAND FLASH，NORFLASH，SD/eMMC，Quad/Octal SPI，Hyper RAM/Flash 以及其他接口，可以连接诸如 WLAN，蓝牙，GPS，displays 以及摄像头传感器。同时具有丰富的音视频特性，包括 LCD 显示，显示加速器，摄像头接口，SPDIF 以及 I2S 音频接口。

4.2. 基于 SDK 中 eIQTFLm 工程进行模型集成

首先需要指出的是，GRU 模型相较于一般基于 CNN 的模型相比，其模型部署方法是一致的。特别地，eIQ MODEL TOOL 会把 GRU 模块降解成由最基本的 TensorFlowLite 所支持的算子所组成的模型结构。这样一来，就可以使用我们的 TensorFlowLite 引擎进行推理了。而对应到 MCU 上就是使用 tflite-micro 推理引擎进行推理。Tflite-micro 使用一种算子注册机制决定哪些算子的实现将被链接到最终生成的可执行映像中。这决定了为了将模型部署到 MCU 上，需要根据模型所需节点来修改注册算子的源文件。这里我们已 eIQ 中 label_image 的例子作为基础进行修改。找到工程中的 model_ds_cnn_ops_micro.cpp 文件，它就是刚才提到的注册算子的源文件，原来的例子包含了推理 label_image 所需的算子，我们对其进行修改如下：

```
tflite::MicroOpResolver&MODEL_GetOpsResolver()
{
static tflite::MicroMutableOpResolver<15>s_microOpResolver;
s_microOpResolver.AddAdd();
s_microOpResolver.AddAssignVariable();
s_microOpResolver.AddCallOnce();
s_microOpResolver.AddFullyConnected();
s_microOpResolver.AddLogistic();
s_microOpResolver.AddMul();
s_microOpResolver.AddReshape();
s_microOpResolver.AddReadVariable();
s_microOpResolver.AddSub();
s_microOpResolver.AddSplit();
s_microOpResolver.AddSplitV();
```

```
s_microOpResolver.AddSoftmax();
s_microOpResolver.AddTanh();
s_microOpResolver.AddUnpack();
s_microOpResolver.AddVarHandle();
return s_microOpResolver;
}
```

修改好之后，将模型的二进制数据转换成符合 C 数组定义格式的文本形式(例如，使用 xxd 工具)，并替换工程中 model_data.h 里面所包含的原始模型，就完成了所有关于模型的准备工作。当然对于数据预处理部分需要编写对应的 MIC 采集以及特征计算部分，限于篇幅本文就不再展开了。有兴趣的读者可以联系作者。

5. 实验验证

选择 i.MX RT1060 EVK 作为试验平台，整体组装方案以及 UI 设计如图 6 所示：

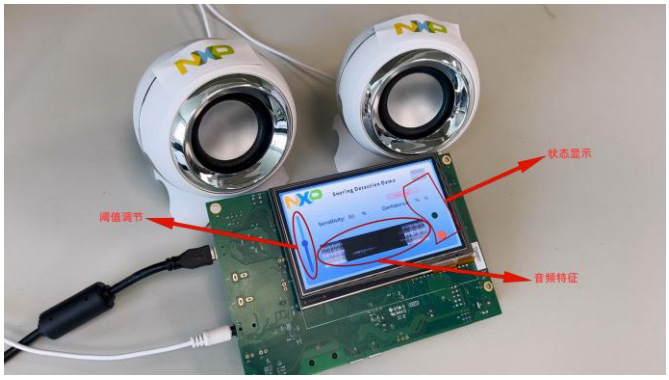

Figure 6. Overall architecture
图 6.整体组装方案

UI 页面主要分为三大显示区，分别是阈值调节区：可以通过滑动条灵活的调节检测阈值，提高系统的抗干扰能力；音频特征区：显示当前音频信号的特征图。状态显示区：显示当前模型预测结果，红灯表示为异常，绿灯表示为正常。

6. 结语

本研究实现了一种能够在资源有限的微控制器单元(MCU)上运行的鼾声检测方法，采用门控循环单元(GRU)模型对音频数据进行处理，并且结合模型优化方案，例如模型结构裁剪，模型量化等，成功将 GRU 模型适配到 MCU 平台，使得系统能够独立完成鼾声检测无需依赖外界计算资源，且无需联网。同时为了丰富产品显示，设计了一个人机友好的 UI 界面，可以实时显示系统识别状态，并可以调节系统检测阈值等。实验表明，该模型在保持高准确性的同时，降低了系统算力需求，满足移动健康监测设备的实时性和便携性。这一研究为鼾症患者提供了新的睡眠健康管理解决方案，并拓展了深度学习在嵌入式系统中的应用前景。

参考文献

[1] Nguyen, M. and Huang, J. (2022) Snore Detection Using Convolution Neural Networks and Data Augmentation.In: Long, B.T., Kim, H.S., Ishizaki, K., Toan, N.D., Parinov, I.A. and Kim, YH., Eds.,*Proceedings of the International*

Conference on Advanced Mechanical Engineering, Automation, and Sustainable Development 2021 (*AMAS*2021). AMAS 2021. Lecture Notes in Mechanical Engineering. Springer, Cham. https://doi.org/10.1007/978-3-030-99666-6_15

[2] Xie, J., Aubert, X., Long, X., van Dijk, J., Arsenali, B., Fonseca, P., *et al*. (2021) Audio-Based Snore Detection Using Deep Neural Networks. *Computer Methods and Programs in Biomedicine*, **200**, Article 105917. https://doi.org/10.1016/j.cmpb.2020.105917

[3] Goodfellow, I., Bengio, Y. and Courville, A. (2016) Deep Learning. MIT Press, 367-415.

[4] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., *et al*. (2014). Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the* 2014 *Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), Doha, October 2014, 1724-1734. https://doi.org/10.3115/v1/d14-1179

[5] Zhang, A., Lipton, Z.C., Li, M. and Smola, A.J. (2023) Dive into Deep Learning.Cambridge University Press.

[6] Krishnamoorthi, R. (2018) Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper.arXiv:1806.08342.https://doi.org/10.48550/arXiv.1806.08342