TÉCNICO LISBOA

Efficient Speech Recognition on the Edge with Binary Neural Networks

Frederico Maria de Almeida Santos Roque

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Professor Pedro Filipe Zeferino Aidos Tomás
Professor Nuno Filipe Simões Santos Moraes da Silva Neves

Examination Committee

Chairperson: Professor António Manuel Raminhos Cordeiro Grilo
Supervisor: Professor Pedro Filipe Zeferino Aidos Tomás
Member of the Committee: Professor David Manuel Martins de Matos

December 2024

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

To my supervisors, Dr. Pedro Tomás, Dr. Nuno Neves and Dr. Nuno Roma, you have my deep gratitude for the constant mentoring and counsel throughout these months, without which this work wouldn't have been possible.

I would also like to thank Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento (INESC-ID) for providing me with the necessary tools to carry out this project and Minho Advanced Computing Center (MACC) for providing crucial access to the Deucalion supercomputer.

Finally, to my friends and family, namely my mom, my brother and my girlfriend: Your care, encouragement and unwavering support over all these years helped me reach this stage. Thank you.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project UNIFY (DOI: 10.54499/2022.06780.PTDC).

Abstract

Automatic speech recognition has seen large improvements in the last decade thanks to the use of deep neural networks and the introduction of Transformers and it's derivatives. However, the ever-increasing complexity of new models makes it harder to run them on low-end hardware like edge devices. It tends to be both faster and more energy efficient to off-load neural network computing to the cloud even with the costs associated with transmitting data over the internet. This is problematic for applications that require low latency and low energy consumption.

Binary neural networks are an extreme method of quantization that aims to massively reduce memory and computational requirements while still being competitive with full-precision models regarding accuracy. Binary neural networks have been shown to provide a $38\times$ size reduction and a $58\times$ speedup on ImageNet, making it viable to run state-of-the-art models on the CPU instead of the GPU or a dedicated accelerator. However, the state-of-the-art for these networks is mainly focused on deep convolution networks which have fallen into disuse since the introduction of Transformers.

Hence, this thesis aims to minimize the computational requirements of running a binary neural network on low-end devices for automatic speech recognition by binarizing a state-of-the-art model and optimizing its inference latency and memory consumption. This work proposes Binary Conformer which required binarizing a model with significantly higher complexity than what was attempted before for a task outside the scope of the majority of the current research.

Keywords

Automatic Speech Recognition; Binary Neural Networks; Edge Computing; Speech-to-text.

Resumo

O reconhecimento automático de fala tem registado grandes melhorias na última década graças à utilização de redes neuronais profundas e à introdução dos Transformers. No entanto, a crescente complexidade dos modelos torna mais difícil executá-los em hardware de baixa capacidade, como dispositivos edge. Tende a ser mais rápido e energeticamente mais eficiente transferir a computação das redes neuronais para a nuvem, mesmo considerando os custos associados à transmissão de dados pela internet. Isto é problemático para aplicações que exigem baixa latência e consumo energético.

As redes neuronais binárias são um método extremo de quantização que visa reduzir massivamente os requisitos de memória e computação, mantendo-se competitivas com modelos tradicionais. Foi demonstrado que as redes neuronais binárias proporcionam uma redução de tamanho de $38\times$ e uma aceleração de $58\times$ no ImageNet, tornando viável executar modelos no CPU em vez da GPU ou de um acelerador dedicado. No entanto, o estado da arte destas redes está principalmente focado em redes convolucionais, que caíram em desuso desde a introdução dos Transformers.

Assim sendo, esta tese visa minimizar os requisitos computacionais para executar uma rede neuronal binária em dispositivos de baixa capacidade para reconhecimento automático de fala, através da binarização de um modelo de última geração e da otimização da sua latência de inferência e consumo de memória. Este trabalho propõe o Binary Conformer, que exigiu a binarização de um modelo com uma complexidade significativamente superior ao que foi tentado anteriormente para uma tarefa fora do âmbito da maioria da investigação atual.

Palavras Chave

Computação de Ponta; Conversão de Fala para Texto; Reconhecimento Automático de Fala; Redes Neuronais Binárias.

Contents

List of Figures

List of Tables

Acronyms

ALR	Attention Layer Replacement
ALRR	Attention Layer with Residual Connection Replacement
ASLR	Attention Separate Heads Layer Replacement
ASR	Automatic Speech Recognition
BCNN	Binary Convolutional Neural Network
BN	Batch Normalization
BNN	Binary Neural Network
CER	Character Error Rate
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
CPU	Central Processing Unit
CTC	Connectionist Temporal Classification
ELR	Encoder Layer Replacement
FFT	Fast Fourier Transform
FFNN	Feed Forward Neural Network
GELU	Gaussian Error Linear Unit
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
IPC	Instructions Per Clock
Im2Col	Image to Column
ISA	Instruction Set Architecture
KD	Knowledge Distillation

LLM	Large Language Model
LM	Language Model
LSTM	Long Short-Term Memory
MAC	Multiply and Accumulate
MFCC	Mel Frequency Cepstral Coefficient
MT	MegaTransfers
NaN	Not a Number
NN	Neural Network
OOD	Out-of-Domain
PER	Phoneme Error Rate
RAM	Random Access Memory
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
STE	Straight Through Estimator
TPU	Tensor Processing Unit
WER	Word Error Rate

1

Introduction

Neural Network (NN) models have been achieving incredible milestones in multiple tasks across different fields. However, the ever-increasing complexity and size of new models have been a growing concern. The computational and memory requirements are a bottleneck for the deployment of these models in edge devices. The need for more efficient models becomes more apparent as the number of devices and tasks that could benefit from these models increases. One such task is Automatic Speech Recognition (ASR) which not only requires low latency but, when operating in an energy-constrained environment, could benefit from lower energy consumption as small quantized models consume 0.1% of battery for 10 seconds of inference on a small device like an Apple Watch 3 [14]. Binary Neural Networks (BNNs) are a type of model that is characterized by significantly reduced computational and memory requirements, when compared to standard NNs, for which recent research shows promising results for the mitigation of the accuracy loss incurred from quantization.

1.1 Motivation

Off-loading the inference of conventional NNs to the cloud consumes less battery of the mobile device while achieving better latency [15]. However, running models on cloud servers is extremely expensive because of hardware, energy, and bandwidth costs. For example, GitHub Copilot costs Microsoft, on average, $20 per every $10 subscription with some users reportedly raising that cost up to 80$ [16, 17]. Furthermore, this server-focused approach suffers from the same drawbacks as any other cloud-based service: it requires an internet connection, which can be unreliable or even unavailable in some situations; and it raises privacy and security concerns regarding user data.

ASR is a perfect example of a set of features that could prove extremely useful if it can run locally on very low-end hardware, like on-device live captioning (e.g., TV live captioning of foreign language programs) or helping hearing-impaired individuals to communicate. BNNs can provide $38\times$ size reduction and $58\times$ speedup [18] making it possible to run state-of-the-art models on the CPU instead of the GPU or a dedicated accelerator.

In contrast, in 2021 Google released an end-to-end ASR model running locally [19, 20] on pixel devices by leveraging their proprietary accelerator: the Tensor Processing Unit (TPU) engine, an 8-bit only inference accelerator. The authors claim that, for the first time, a local model ($400\times$ smaller in comparison) had better Word Error Rate (WER) and latency than their conventional model running on the cloud. Since then, newer architectures pushed the state-of-the-art further and, while pixel devices still run at least some ASR tasks locally, claims about their performance were not found. Nonetheless, these advancements paint a clear picture of the direction of the field: the deployment of models on edge devices is powered by smaller quantized NNs that can have their efficiency improved by leveraging dedicated hardware.

1.2 Objectives

The main objective of this thesis is to explore the use of BNNs in ASR and create a model that could be run on an edge device. The main focus was to reduce both energy requirements and computational requirements by building upon current NNs and BNNs research and not to further improve the accuracy of current state-of-the-art models. The model should be able to run on a low-end device and achieve competitive results in terms of accuracy, latency, and energy consumption.

This work proposes Binary Conformer, a binarized variant of the Conformer architecture. It explores the necessary changes to the Conformer Block both as a whole and to individual layers. Based on the current understanding of the effects this extreme form of quantization has on the model's behavior, the changes introduced aimed to reduce computational requirements at inference time, improve information flow during the forward step, improve gradient propagation during the backwards step while aiming to

match as closely as possible the WER and Character Error Rate (CER) of the state-of-the-art models.

Furthermore, this work evaluates the computational performance of the model with a custom inference executable. Then, it dissects the impact each layer and calculation has on inference to identify key hotspots and open the door to further optimizations and research of novel model architectures.

1.3 Contributions

This work has resulted in the following contributions to the fields of ASR, BNNs and running NN models on the edge:

- A new model architecture that successfully integrates Binary Neural Networks into a significantly more complex ASR model than previously achieved.
- The demonstration of the feasibility of applying BNNs to state-of-the-art ASR systems, opening new avenues for efficient speech recognition on resource-constrained devices.
- Development of a custom inference executable that implements both full precision and BNN-specific optimizations. The improved performance is especially noticeable on low-end hardware.
- An in-depth analysis of the complex trade-offs between accuracy, computational requirements and memory requirements when deploying BNNs-based ASRs systems.

1.4 Outline

The remainder of this document is organized as follows. Chapter 2 provides an overview of the state of the art for both ASR and BNNs. Chapter 3 proposes the Binary Conformer architecture, describes the training phase while providing relevant context regarding its limitations. Chapter 4 presents the optimizations implemented on the custom executable and how they affected performance. Finally, Chapter 5 studies the gathered results and identifies the key contributions and limitations of BNNs on the inference step.

2

State of the Art

This chapter describes the State of the Art for the relevant literature for the ASR task. First, a task itself will be introduced followed by an explanation of its evaluation metrics since they will be mentioned when comparing the models. Then, the first section is dedicated to an analysis of the current best model architectures and a comparison between them. These models require a significant amount of computation and memory, therefore, the second section is an overview of selected applicable optimizations. The third section is an analysis of Binary Neural Networks, an extreme form of weight quantization. They are a promising approach to reduce the computational cost of the models, but they are not yet widely used in the literature due to their limitations. After discussing the methodologies, benefits, and drawbacks of this approach, the chapter ends with an overview of current research on BNNs for the ASR task followed by some conclusions.

2.1 Automatic Speech Recognition

ASR is one of the many speech-processing tasks and can be described as the technology allowing a device to recognize and transcribe human speech. Before the introduction of deep learning, the

state-of-the-art models used a Gaussian Mixture Model (GMM)-Hidden Markov Model (HMM) statistical architecture that was composed of: feature extraction — typically Mel Frequency Cepstral Coefficients (MFCCs) — an acoustic model, a language model, and a search based on Bayes decision rule [21]. Currently, the literature has steered towards end-to-end models that learn both acoustic and linguistic features [22].

The two evaluation metrics typically used for this task are WER, or CER when models are trained to recognize individual letters or characters. Both these error rates can be calculated with Equation (2.1) where S is the number of substitutions, I is the number of insertions, D is the number of deletions, and N is the number of characters or words depending on the chosen metric.

$$ErrorRate = \frac{S + I + D}{N} \tag{2.1}$$

2.1.1 Feature Extraction

Pre-processing the raw audio is fundamental, as it compresses the input and extracts the relevant speech features. In all speech-related tasks, but especially in ASR, MFCCs have been overwhelmingly the most used technique for ASR in the literature [23] for more than 4 decades [24]. Even though other feature extraction techniques exist, MFCCs are wildly used because they represent audio features in a way akin to the way humans perceive sounds while also being a very efficient representation of the power spectrum [25]. Six coefficients are sufficient to represent most of the relevant features [24] but, in general, the number chosen is up to 13 [26]. This efficiency is complemented by strong robustness to noise, making this technique ideal for this task. Furthermore, it is possible to increase the number of coefficients, gaining further granularity of the features. For example, the Whisper model uses 80 coefficients [9].

The procedure to generate the MFCCs for a given raw audio clip is the following:

1. The audio signal is divided into short frames (typically 20-40 ms long). Sounds change frequencies rapidly, so it is necessary to analyze small segments where these changes are minimal;

2. Each frame is multiplied by a windowing function to minimize discontinuities at the frame boundaries. This effectively means applying a sliding window of size *frame* to all the audio input;

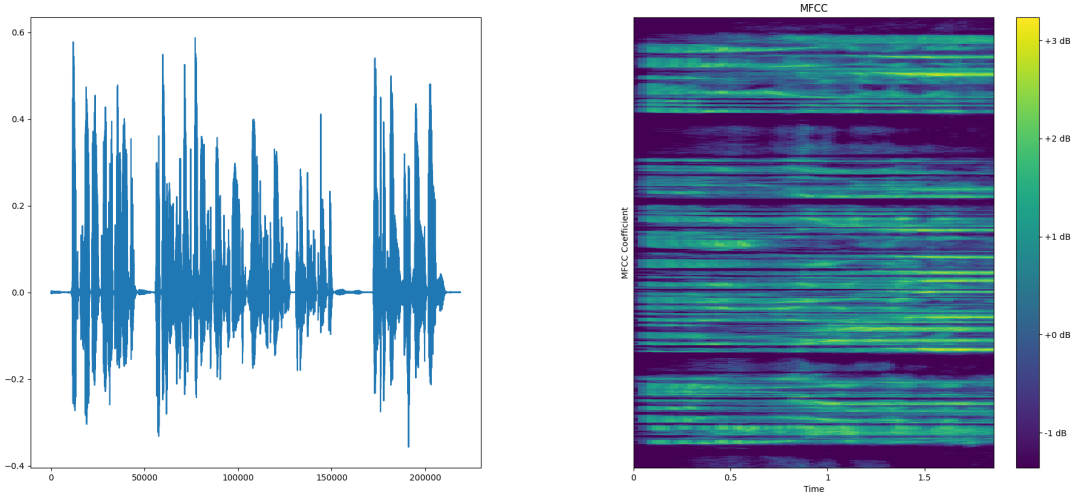
3. A Fast Fourier Transform (FFT) is applied to each frame to convert it from the time domain to the frequency domain;

4. The power spectrum obtained from the FFT is then passed through a set of band-pass filters known as the Mel filter bank. These filters are designed to mimic the human ear's response, which does not perceive frequencies linearly;

5. The energies of each filter in the Mel filter bank are scaled logarithmically. This step complements the previous one in mimicking human's perception of sound;

6. Finally, a Discrete Cosine Transform is applied to the log filter bank energies. This final step creates what can be thought of as a "spectrum of a spectrum" and compresses the representation of the filter banks.

Nonetheless, some state-of-the-art models (e.g., Wav2Vec2 [1]) use the raw audio as input but, on the other hand, include a feature encoding layer, such as several blocks containing a temporal convolution followed by layer normalization and a Gaussian Error Linear Unit (GELU) activation function used on. This approach aims to achieve comparable results while requiring orders of magnitude less labeled training data.



(a) Raw Audio Signal

(b) MFCCs

Figure 2.1: Comparison between the input audio signal and the MFCCs obtained after the feature extraction process.

2.1.2 Model Architectures

This subsection provides a description and comparison of the most relevant model architectures for ASR.

A. Whisper

Whisper is a transformer-based model that takes as input a Log-Mel Spectrogram of 80 coefficients. The overall architecture is as follows: (1) first, the input is processed with two 1D convolution layers and the GELU activation function; (2) then, encoder Transformer blocks are applied; (3) finally, Transformer decoder blocks are applied to the output of the encoder blocks. There are more steps in the architecture (shown in Figure 2.2), but they are not as relevant for this thesis (which only focuses on ASR) since their function is to enable the model to handle multiple tasks.

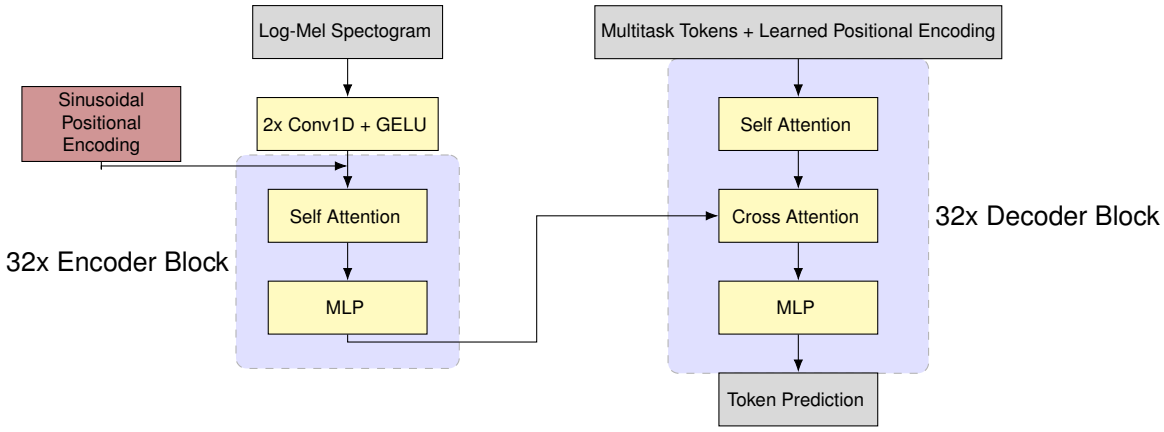


Figure 2.2: Whisper architecture.

Whisper is, in comparison to the other models (see Table 2.2), much bigger but is, according to the authors, capable of a lot more than just ASR. Their goal was to create a model that could be used in a variety of speech-related tasks, to replace as many of those individual models with a single one [9]. Their approach was to train the model on many speech-processing tasks, including multilingual speech recognition, speech translation, spoken language identification, and voice activity detection.

According to the results presented by Radford et al. [9] and the analysis done by Mehrish et al. [7](see Table 2.1), Whisper is not only competitive but also more consistent (i.e., generalizes better) across datasets and settings.

B. Distil-Whisper

Even with the advantages of Whisper, both its size (1550M parameters) and computational requirements remain drawbacks when running them in either low-latency or resource-constrained environments. This challenge pushed the authors to iterate and distill whisper into a smaller model: Distil-Whisper [27]. In this later work, the authors reduced the number of Transformer decoder blocks from 32 to 2, reducing the model's size by 51% and making it 5.8 times faster while keeping WER within 1% of the original model on Out-of-Domain (OOD) short-form audio and outperforming it on OOD long-form audio. The

Table 2.1: Whisper and Wav2Vec WER across multiple datasets without the use of a Language Model. Results taken from Mehrish et al. [7].

Dataset	Wav2Vec2 Large	Whisper Large
Common Voice	29.9	9.0
Fleurs En	14.6	4.4
Tedlium	10.5	4.0
CHiME6	65.8	25.5
VoxPopuli En	17.9	7.3
Switchboard	28.3	13.8
CallHome	34.8	17.6
LibriSpeech Clean	2.7	2.7
LibriSpeech Other	6.2	5.2

performance increase was attributed, by the authors, to fewer hallucinations and repetitions. The results presented by the authors suggest that Knowledge Distillation (KD) (further discussed in Section 2.2.1) can be the best-suited training methodology when trying to optimize models for performance if used correctly (i.e., which parameters can be eliminated without sacrificing accuracy).

Another result relevant to this thesis is that reducing the number of Transformer encoder blocks has a very significant impact on WER: it was suggested that having a deep encoder is the most important aspect of the model to enable strong WER performance.

C. Wav2Vec2

Wav2vec2 is a framework that pre-trains models on unlabeled audio data and then fine-tunes them on a specific dataset [1]. The authors propose this "self-supervised pre-training" method that, they claim, achieves competitive results while requiring 2 orders of magnitude less labeled data. This technique allows the model to learn the structure of speech.

The proposed model consists of two multilayer Convolutional Neural Networks (CNNs) stacked on top of each other (that encode the raw audio) followed by a transformer (that works as a "context layer"). The encoder maps raw audio input to a representation, where each vector covers about 30 milliseconds (ms) of speech. The context network uses those vectors to generate its representations, which cover a larger span of up to a second.

The pre-training consists of using these representations to solve for a self-supervised prediction task: a portion of the convolutional feature encoder outputs are masked before they are fed into the context network, and then the model is trained to predict the true speech representation of the masked portion from a set of distractors. Then the model is fine-tuned on a small labeled dataset: a linear layer is added on top of the context layer and the model is then trained to predict letters or phonemes using the Connectionist Temporal Classification (CTC) loss. During this fine-tuning process, the weights of the convolutional feature encoder (in blue) are frozen.

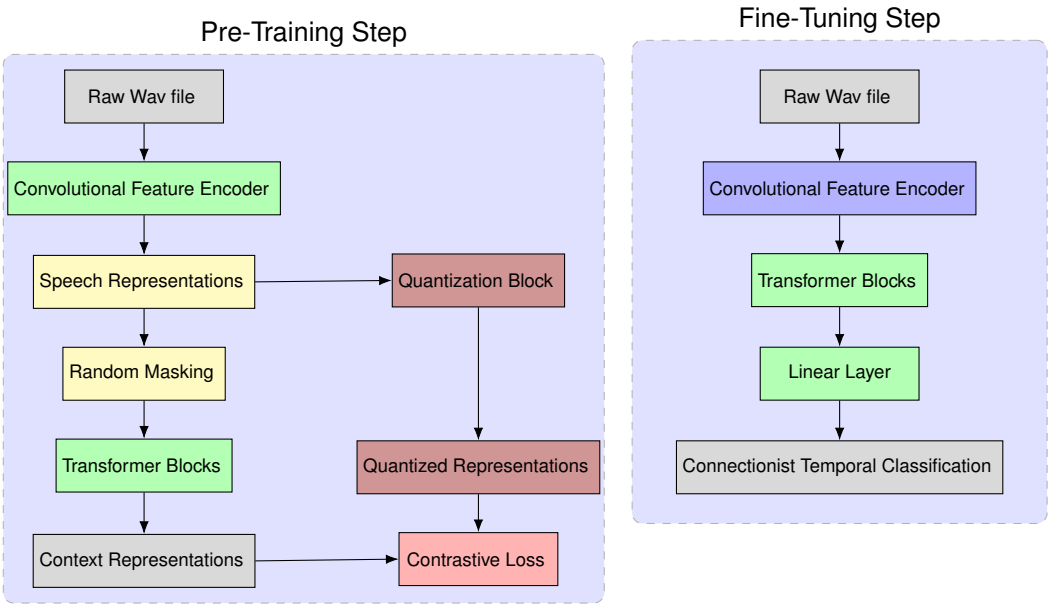


Figure 2.3: Wav2Vec2 [1] Pre-Training and Fine-Tuning processes. Layers in green can have their weights updated. Layers in blue have their weights frozen. The quantization box discretizes the Speech Representations into a finite set of speech representations via product quantization.

The final model uses the Beam Search Decoder presented by Pratap et al. [28] and was tested with multiple Language Model (LM). The results presented by Baevski et al. [1] suggest that the model is competitive with the state of the art and that it can achieve those results with much less labeled data. Those improvements are noticeable not only in scenarios where the amount of data is very limited but also in scenarios where labeled data are abundant, like ASR for English.

D. Conformer

Both transformers and CNNs are essential components of the state-of-the-art ASR models. While CNNs are good at extracting local features, transformers are good at capturing long-range dependencies. Conformers [29], depicted in Figure 2.4, are an architecture that aims to combine the benefits of both CNNs and transformers. The Conformer-Transducer [29] model consists of a convolution subsampling layer that feeds into Conformer blocks (encoder) and a single-layer Long Short-Term Memory (LSTM) (decoder).

According to the authors, Conformers can achieve state-of-the-art results on multiple ASR benchmarks while being more computationally efficient than previous models. Their results show that Conformer-Large outperforms Wave2Vec2-Large on the Librispeech dataset while having 2.6 times fewer parameters.

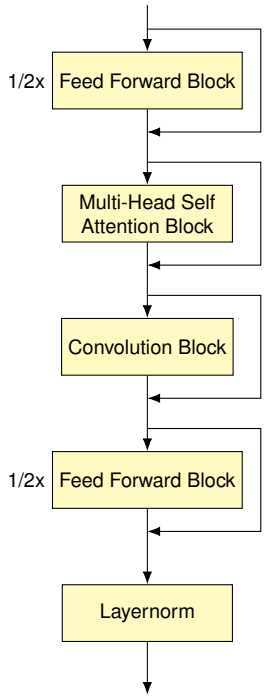


Figure 2.4: Conformer Block architecture.

E. Model Comparison

Table 2.2 provides a comparison of what was described as the best models for this task in the literature [7, 10]. The models are compared on LibriSpeech clean (a dataset derived from read audiobooks from the LibriVox project) and on MyST (a dataset of children's conversational speech) using the results presented by the model's corresponding papers, by Barcovschi et al. [10] and Mehrish et al. [7]. Comparison data on children's speech datasets was included to compare the models on an adjacent domain none of the models had been previously trained but is close enough since the dataset is in English, so the linguistic features are similar. This data seems to indicate that there is still room for improvement in terms of generalization.

All models presented use an encoder-decoder architecture as their foundation. The first thing to notice is that the models can work with or without a LM: a separate model that is trained independently on a large corpus of text and is used to improve predictions by accounting for the probability of a word appearing in a given context. Its inclusion is a trade-off between reducing WER and increasing computational requirements. The results presented in Table 2.2 suggest that the inclusion of a LM improves performance, albeit with diminishing returns proportional to the increase in model size. Furthermore, because it is a separate task and falls outside the scope of the thesis, it will not be further discussed.

A relevant aspect of the metrics for Whisper is that the authors claim that those WER were obtained

Table 2.2: Comparison of the State-of-the-Art models: Conformer with and without a LM [8], Wav2vec2 [1] with and without a LM, and Whisper [9] (and its English-specific variants in brackets) Zero-Shot (ZS) performance. The models are compared on LibriSpeech clean and on MyST [10]. Entries marked with "-" represent results that were not available.

Model	Language Model	# Parameters (M)	# Layers	WER	
				Librispeech (clean)	MyST test
Wav2Vec2 - Base	No LM	95	12	-	-
	Transformer	95	12	2.6	15.41
Wave2Vec2 - Large	No LM	317	24	2.7	-
	Transformer	317	24	2.0	12.50
Conformer - Small	No LM	10.3	16	2.7	-
	LSTM	10.3	16	2.1	21.34
Conformer - Medium	No LM	30.7	16	2.3	-
	LSTM	30.7	16	2.0	24.99
Conformer - Large	No LM	118.8	17	2.1	-
	LSTM	118.8	17	1.9	25.91
Whisper - Tiny (ZS)	No LM	39	4	7.6 (5.6)	40.09 (33.02)
Whisper - Base (ZS)	No LM	74	6	5.0 (4.2)	32.14 (29.15)
Whisper - Small (ZS)	No LM	244	12	3.4 (3.1)	26.22 (26.72)
Whisper - Medium (ZS)	No LM	769	24	3.1 (2.9)	25.11 (28.06)
Whisper - Large V2 (ZS)	No LM	1550	32	2.7	25.00

in a "zero-shot setting". While traditionally "zero-shot" implies a complete lack of prior examples or training related to a task, in this case, they use the term "zero-shot setting" to emphasize the fact that the testing was done on datasets the model did not train on nor was fine-tuned to. This distinction is important because, as the authors claim, it is a more realistic setting for real-world applications and the competitive results imply a better generalization of the model.

According to the findings of Barcovschi et al. [10], the Conformer-Transducer architecture outperforms their Whisper and Wav2Vec2 counterparts while being significantly smaller for the smaller models. But, as the model size increases, Conformer tends to be outperformed. The authors pointed out that this behavior suggests that this architecture loses its generalization capabilities with a parameter increase. Better results were obtained by combining the benefits of the Conformer architecture with Wav2Vec2's pre-training framework [7, 30]. That does appear to be the most promising approach for future development as it achieves state-of-the-art results with a significant reduction in model size while also requiring a lesser amount of labeled data.

2.1.3 Model Training

ASR models commonly employ a CTC [31] loss that helps the model learn the alignment between the audio and text sequences automatically. CTC solves the fundamental problem of alignment between input and output sequences of different lengths without requiring pre-aligned training data. This is particularly relevant for ASR because, on one hand, the input length is bigger that the output length and, on the other hand, the same audio input length may have significantly different output lengths (i.e., number of tokens or characters). Furthermore, precisely aligning input and outputs is complicated and time-consuming making it unviable to do at the scale needed to train NNs. The CTC introduces the concept of a ”path” - a sequence of labels that includes blank tokens (represented by ’-’ on the example) and allows for repeated characters. Furthermore, the path must have the same length as the input sequence, which solves the alignment problem. Then the ”path” is transformed into words by removing the replicated characters and blank labels. For example, for an audio clip with 5 frames (extracted following the process described on section Section 2.1.1) the word ”cat” has multiple valid paths including: ”c-a-t”, ”c-at-”, ”c–at”, ”caa-t”.

2.2 Reducing Model Complexity

The models presented in Section 2.1.2 are complex and have both high memory and computing performance requirements. Reducing these requirements, while maintaining generalization, is a challenge that is researched alongside architecture developments aimed at reducing WER.

This section will present some of the techniques that have been proposed to reduce the complexity of these models. First, optimizations of Transformer and Conformer layers will be presented while explaining the reason they are commonly the focus when trying to increase inference speed. Then, a recent training methodology, known as KD, is presented as a way to use readily-available and already trained models to “distill” their knowledge into a smaller and more efficient network with a significantly lesser impact on accuracy metrics when compared to purely training the smaller model from scratch.

An extreme form of quantization, the binarization of a model’s weights, is also a method that has been proposed to reduce the complexity of a model. But, given the importance of this method for the thesis, it will be presented in a standalone section (Section 2.3).

2.2.1 Knowledge Distillation

KD is a training methodology that aims to transfer the knowledge of a large model (teacher) to a smaller model (student) proposed by Hinton et al. [32]. Traditionally, models are trained with a dataset composed of inputs and labels (i.e., true targets). With KD, The student is trained to match the class probabilities

of the teacher (i.e., soft targets). Nonetheless, a small term encouraging the student to also predict the true targets is usually added to the loss function. The inclusion of this term is considered helpful by the authors because the smaller model cannot exactly match the soft targets.

Using the output probabilities as soft targets aims to, as the authors put it, train to generalize well by also learning the relationship between classes (e.g., in an image recognition task, a cat is more similar to a dog than to a car and that is reflected on the output probabilities).

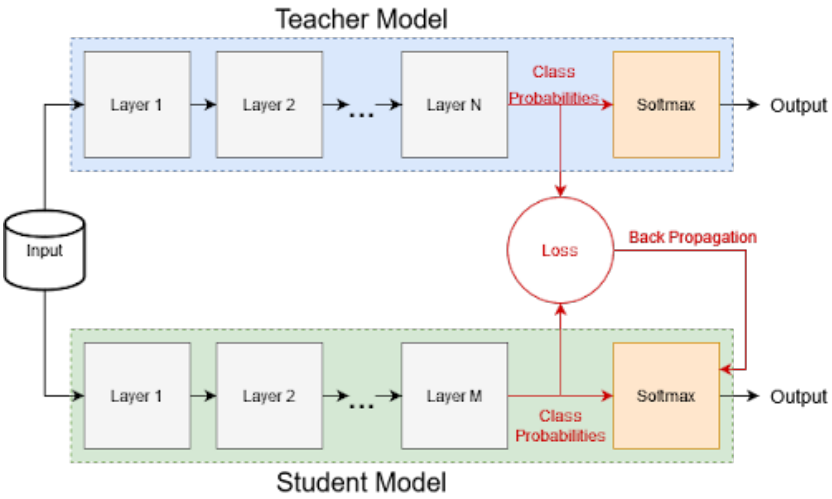


Figure 2.5: Training a NN model using KD.

The possibility to compress larger models into smaller ones was first proposed by Bucila et al. [33] where a model was trained on a dataset generated by an ensemble of models; effectively training it to mimic the outputs of the ensemble and transferring the knowledge. A challenge to this compression is that the ensemble of models can produce outputs with very high confidence, making the influence of classes other than the correct one too small. To overcome it, the authors proposed using the logits (inputs to the softmax function) as soft targets.

The KD approach claimed to be more general [32], increases the temperature (T in Equation (2.2)) of the softmax function of the teacher until it produces a set of suitable soft targets. Those soft targets will be used as a training set for the student model, which will use that same temperature value for its softmax layer.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{2.2}$$

Softmax Layer. Computes the probability (p_i) of each class i from the logit z_i by comparing it to all other logits. T is the temperature and is generally equal to 1.

The importance of having a high-temperature value was studied by Hinton et al. [32] and the results suggest that change is what allows the student to learn the teacher's behavior. The results, presented

for the ASR task, suggest that with KD the distilled model can closely match the ensemble of models. Furthermore, the authors also criticize previous work that built an acoustic model for the ASR task by matching the output probabilities of a pre-trained model [34]: they argue that by using a temperature of 1 the acoustic model only reduced the WER by 28% of the gap between the big and small models.

Finally, an appropriate loss function must be used with this training method due to its unusual characteristics and goals. Given that the goal is for the student model to match the relationships between the labels and that it is impossible for it so have identical probabilities of the teacher model, the most commonly used loss function is the Kullback-Leibler Divergence [32]:

$$KL(P||Q) = \sum_{x} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{2.3}$$

2.2.2 Optimizing Transformers

Transformers, and by extension Conformers, are the most computationally expensive part of the models presented in Section 2.1.2 and, as such, are the focus of most of the research aimed at reducing the complexity of these models. This subsection gives an overview of the proposed techniques and architectural changes that aim to reduce computational requirements.

A – Replacing Attention Layers with Feed-Forward NN was proposed as a way to mimic the behavior of attention layers while maintaining accuracy [2]. First, the model is trained with a traditional Transformer and then the attention layers are replaced with Feed Forward Neural Networks (FFNNs), and the model is retrained using KD (all layers are frozen except the new FFNN). The different approaches are presented in Figure 2.6. It is relevant to note that the number of parameters increases significantly when doing these replacements: removing an attention layer of 60K parameters for a FFNN with several parameters ranging from 290k to 46M. Four replacement options are presented: Attention Layer Replacement (ALR), Attention Layer with Residual Connection Replacement (ALRR), Attention Separate Heads Layer Replacement (ASLR), and Encoder Layer Replacement (ELR).

The relative accuracy presented suggests that ALR and ASLR are the best approaches to replace attention layers (because they managed to match the baseline model) with a special emphasis on ALR for its relative simplicity and competitive performance against ASLR. Furthermore, the authors presented results showcasing that these techniques could be applied with minimal accuracy loss on Encoder Self-Attention, Decoder Self-Attention, and both at the same time. In contrast, for Decoder Cross-Attention the relative accuracy is 40%, which suggests that that layer should not be replaced.

B – Fast Conformers [35] is a new Conformer architecture that, according to the authors, achieves 2.8x faster inference speed and ensures scalability to a billion parameters while maintaining accuracy

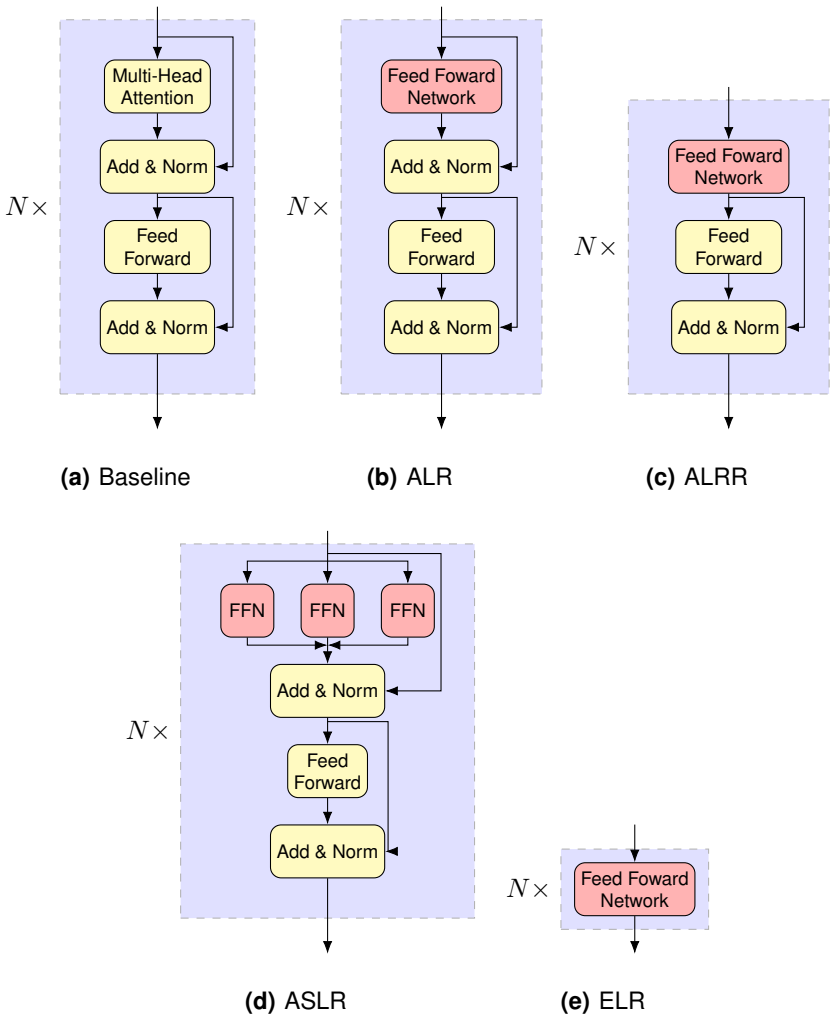


Figure 2.6: Different encoder self-attention replacement approaches [2]. Layers kept are yellow and replacements are red.

and allowing for longer input sequence lengths. Based on the previous work of Efficient Conformers [36] and Squeezeformer [37], this newer approach introduced the following changes to the original Conformer [29] design: (a) increasing down-sampling from 4x to 8x at the start of the encoder; (b) replace the convolution subsampling layers with depth-wise separable convolutions [38]; (c) reduce the number of convolution filters in the down-sampling block from 512 to 256; and (d) reduce the convolution kernel size from 31 to 9.

C – SummaryMixing [3] is another alternative to the self-attention layers for ASR with linear-time complexity. The authors state that this method works specifically for ASR because current models do not capture fine-grained pair-wise relations. Furthermore, they claim that using SummaryMixing reduces inference times by up to 28% and memory usage by 2 times. This layer creates ”global summary” vector

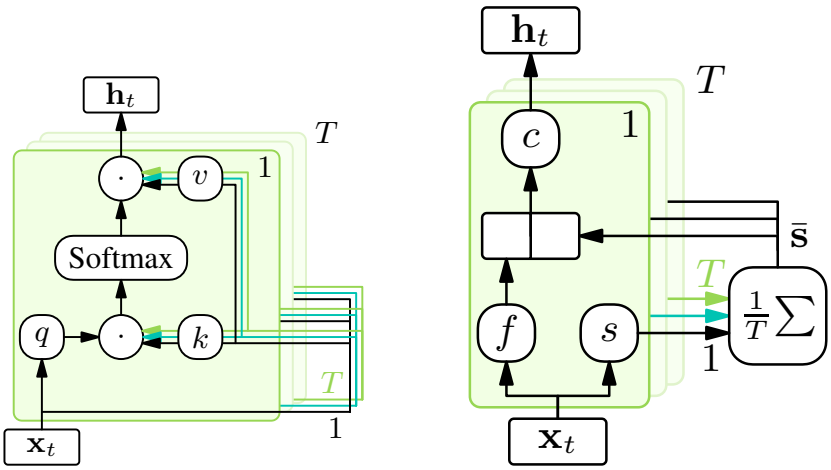
$\bar{s}$ that represents the entire input sequence through averaging:

$$\bar{s} = \frac{1}{T} \sum_{t=1}^{T} s(x_t). \tag{2.4}$$

Akin to how self-attention works, it then creates a hidden representation h_t by concatenating $\bar{s}$ with the local features at each time step:

$$h_t = c(f(x_t), \bar{s}). \tag{2.5}$$

In this way, instead of computing attention weights between every pair of tokens, each token only needs to attend to this single global summary vector and its own local features



(a) Self-Attention **(b)** SummaryMixing

Figure 2.7: Comparison of the self-attention cell and the newly proposed SummaryMixing cell taken from [3].

2.3 Binary Neural Networks

Quantization is a common technique used to reduce the memory footprint, computational complexity, and energy consumption of neural networks by reducing the precision of the model's weights (e.g., 32bit to 16bit float, or to 32bit or 8bit integer). This section will focus on an extreme form of quantization, BNNs, where both weights and activations are constrained to +1 or -1 and, thus, represented by a single bit [39,40]. By reducing precision to a single bit, the memory footprint of the weights of the model can be reduced by $32\times$ and both the computational requirements and energy consumption are greatly reduced by having fewer memory accesses and having much simpler operations that replace multiplications and accumulations: bit-wise *XNOR* and *popcount*.

The literature presents two main approaches to binarization: Deterministic Binarization and Stochas-

tic Binarization.

The Stochastic Binarization [39] function is given by Equation (2.6) where x is the full precision weight, x^b is the binarized weight and σ is the "hard sigmoid" (see Figure 2.8) function given by Equation (2.7). It is used instead of the "soft" version because it is less computationally intensive [39].

$$x^b = \begin{cases} +1 \text{ with probability } p = \sigma(x) \\ -1 \text{ with probability } 1 - p \end{cases} \tag{2.6}$$

$$\sigma = clip\left(\frac{x+1}{2}, 0, 1\right) = \max(0, \min(1, \frac{x+1}{2})) \tag{2.7}$$

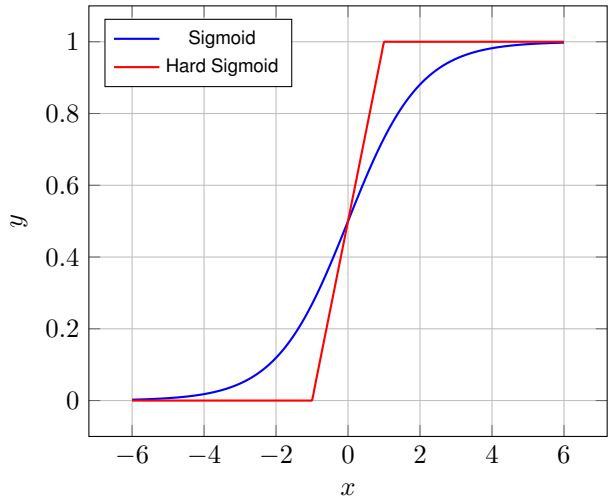


Figure 2.8: Comparison of Sigmoid and Hard Sigmoid Functions

Deterministic Binarization [40] is given by Equation (2.8) where x is the full precision weight and x^b is the binarized weight. From this point on, unless explicitly stated otherwise, this is the binarization method utilized.

$$x^b = Sign(x) = \begin{cases} +1 \text{ if } x \geq 0 \\ -1 \text{ otherwise} \end{cases} \tag{2.8}$$

If the values of the weights and activations are limited to $+1$ and -1, they can be represented by 0 and 1 respectively and, thus, the multiplication is effectively a bit-wise XNOR operation as illustrated by Table 2.3. Furthermore, Multiply and Accumulate (MAC) operations (i.e., dot product) can be replaced by XNOR-(pop)count operations (i.e., XNOR dot product), which are much simpler and faster to compute. After counting the number of set bits (popcount), the output has to be binarized by counting the number of set bits and comparing it to the number of unset bits as shown by Equation (2.9) where x is the input vector, w is the weight matrix, y^b is the binarized y, N is the total number of bits of y and Z is the number of set bits of y.

$$
\begin{aligned}
& y = x \odot w, \\
& y^b = Sign(Z - (N - Z)) = Sign(2Z - N) \equiv Sign\left(Z - \frac{N}{2}\right)
\end{aligned}
\tag{2.9}
$$

Table 2.3: Bit-wise XNOR operation replaces multiplication for BNNs.

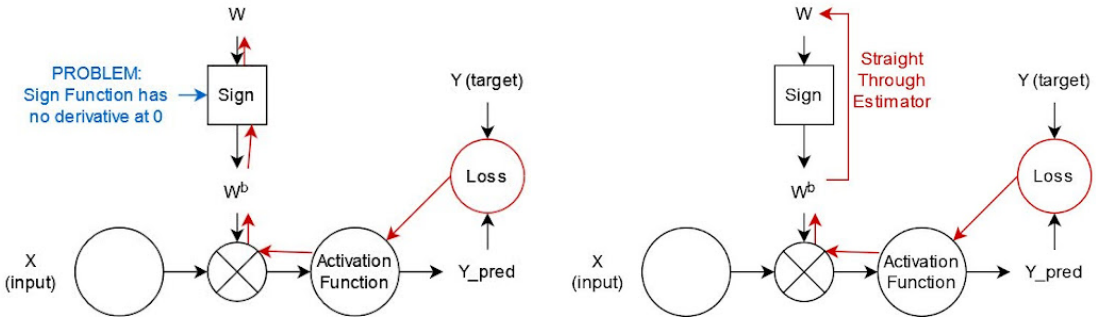
x	w	$x \times w$	x^b	w^b	$x \odot w$
-1	-1	1	0	0	1
-1	1	-1	0	1	0
1	-1	-1	1	0	0
1	1	1	1	1	1

To train NN models, one of the most common methods is the Stochastic Gradient Descent (SGD) backpropagation that can be decomposed in 3 steps.

(a) **Forward Pass:** for a given input, obtain its output by computing the unit activations layer by layer from the first to the last layer;

(b) **Backward Pass:** using the output of the forward pass, compute the gradient of the loss function concerning the weights of the network layer by layer from the last to the first layer;

(c) **Parameter Update:** compute the gradient of each parameter of each layer and update the weights' value.



(a) Backpropagation **(b)** Backpropagation with STE

Figure 2.9: The sign function is not compatible with the backpropagation step (left) so the Straight Through Estimator (STE) is used as an approximation (right).

While the forward and the backward propagations can use the binarized weights and activations, the parameter update step requires full precision weights to work at all [39]. SGD is based on small updates to the weights, which are not possible with binary weights, and, in addition, the sign function

is not derivable at $x = 0$ and, where it is derivable, $\nabla Sign(x) = 0$ making it incompatible with the backpropagation. To overcome this issue, full precision weights (also known as latent weights) are kept to provide a way to use SGD, and the STE [41] is used to bypass the gradient of the sign function resulting in an identity function, as shown by Equation (2.10) where E is the loss function at the output, w is the full precision weight and w^b is the binarized weight. Then, the updated weights are binarized again and a new training step can be performed. After training, the full precision weights are discarded.

$$\begin{aligned} w^b &= Sign(w), \\ \frac{\partial E}{\partial w} &= \frac{\partial E}{\partial w^b} \end{aligned} \tag{2.10}$$

The binarization of the full precision weights introduces a new problem: the binary weights can only be $+1$ or -1 while the full precision weights could "explode" without having any impact on the binary weights. To overcome this issue, Courbariaux et al. [40] proposed a STE that takes into account the saturation effect given by Equation (2.11) that cancels the gradient when w is too large. This STE effectively clips the full precision weights to $[-1, 1]$. The derivative $1_{|r|\leq 1}$ can be represented by the "hard tanh" (see Figure 2.10) given by Equation (2.12).

$$w = w^b 1_{|r|\leq 1} \tag{2.11}$$

$$HardTanh(x) = \max(-1, \min(1, x)) \tag{2.12}$$

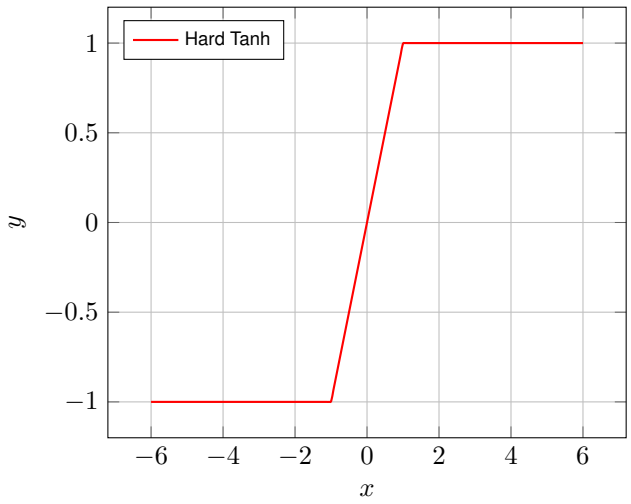


Figure 2.10: The Hard Tanh Function

Finally, in BNNs both weights and activation functions of all layers are binarized meaning that the input and output of all layers are binary, with one exception: the input of the first layer. Binarizing the

inputs leads to a significant accuracy loss, and it is recommended to handle continuous-valued inputs as fixed-point numbers [40] using Equation (2.13) where x is an input vector with m bits of precision, x_1^m is the most significant bit of the first input and w^b is a binarized weight vector and s is the weighted sum.

$$s = x \cdot w^b = \sum_{n=1}^{m} 2^{n-1}(x^n \cdot w^b) \tag{2.13}$$

2.3.1 Layer Optimizations

A – CNN Filter Repetition In contrast to traditional CNNs, CNN-based BNNs have a particularity that can be exploited to accelerate calculations: there is a finite number of unique filters for a given window size given by Equation (2.14) where n is the number of dimensions of the convolutional layer and k_i is the kernel size along dimension i. This allows us to compute the convolution of a filter and reuse the result for all the identical filters.

$$\#UniqueFilters = 2^{\Pi_{i=1}^{n} k_i} \tag{2.14}$$

B – Batch Normalisation The use of Batch Normalization (BN) to improve performance is claimed to be particularly valuable for BNNs [39, 40]. BN is a technique that normalizes the activations of a layer by subtracting the mean and dividing by the standard deviation of the activations of the layer:

$$BN(x) = \gamma \times \frac{x - \mu}{\sqrt{\sigma + \epsilon}} + \beta \tag{2.15}$$

ϵ is added in the denominator for numerical stability and is an arbitrarily small constant. This technique has been shown to improve the performance of NNs by reducing the internal covariate shift and may help regularize the model [42]. However, BN is relatively computationally expensive and Courbariaux et al. [40] claim that the use of the Shift Batch Normalization (an approximation of the conventional BN) not only is significantly faster but also did not incur an accuracy loss. Furthermore, at inference time, the quantization and BN steps can be fused into a single one step by using a threshold τ instead of 0 in the $sign$ function:

$$sign(BN(x)) = \begin{cases} +1 \text{ if } BN(x) \geq 0 \\ -1 \text{ otherwise} \end{cases} = \begin{cases} +1 \text{ if } \gamma \times \frac{x-\mu}{\sqrt{\sigma+\epsilon}} + \beta \geq 0 \\ -1 \text{ otherwise} \end{cases} = \begin{cases} +1 \text{ if } x \geq \tau \\ -1 \text{ otherwise} \end{cases} \tag{2.16}$$

$$\tau = \frac{\gamma}{-\beta\sqrt{\sigma + \epsilon}} + \mu \tag{2.17}$$

2.3.2 Rethinking Latent Weights

The use of latent weights in the training phase of BNNs adds another layer of complexity to the training process because the weight updates are performed with full precision, but those small changes do not affect the network's behavior unless a change of sign occurs. Helwegen et al. [43] proposed a different point of view on the nature of those full precision weights: latent weights do not exist, and those parameters can be described as the sign and inertia of the binary weights as shown by Equation (2.18) where w is the full precision weight, w^b the binarized weight and $m = |w|$. They consider the inertia as an optimizer variable, akin to momentum.

$$w = sign(w) \cdot |w| = w^b \cdot m, w \in -1, +1, m \in [0, \infty) \tag{2.18}$$

Furthermore, the authors challenge the common perception that the binary weights are an approximation of the latent weights (that was originally analogized to Dropout [39, 40]) by performing an experiment that suggests that the binary weights are the true weights of the network: after training, they compared the accuracy of the BNN with a network using the latent weights while keeping the binarization of activations; according to their results, the latter would perform identically or worse than the BNN.

Based on this new perspective, Helwegen et al. [43] propose a new training method that does not use latent weights and instead uses the exponential moving average of gradients: Binary Optimizer. The exponential moving average m_t is given by Equation (2.19) where g_t is the gradient at time t and γ is the adaptivity rate. Finally, the update rule (whether to bit flip a weight) is given by Equation (2.20) where w_t^i is the weight at time t, m_t^i is given by Equation (2.19) and τ is the threshold that determines if a bit flip occurs.

$$m_t = (1-\gamma)m_{t-1} + \gamma g_t = \gamma \sum_{r=0}^{t} (1-\gamma)^{t-r} g_r \tag{2.19}$$

$$w_t^i = \begin{cases} -w_{t-1}^i & \text{if } m_t^i \geq \tau \text{ and } sign(m_t^i) = sign(w_{t-1}^i) \\ w_{t-1}^i & \text{otherwise} \end{cases} \tag{2.20}$$

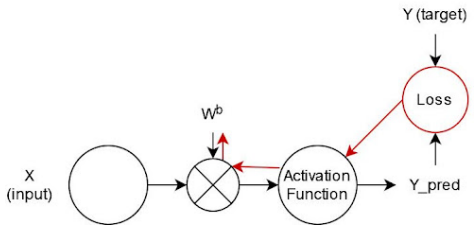


Figure 2.11: The binary optimizer removes latent weights.

The authors draw an analogy between γ and the learning rate of conventional NNs: reducing γ

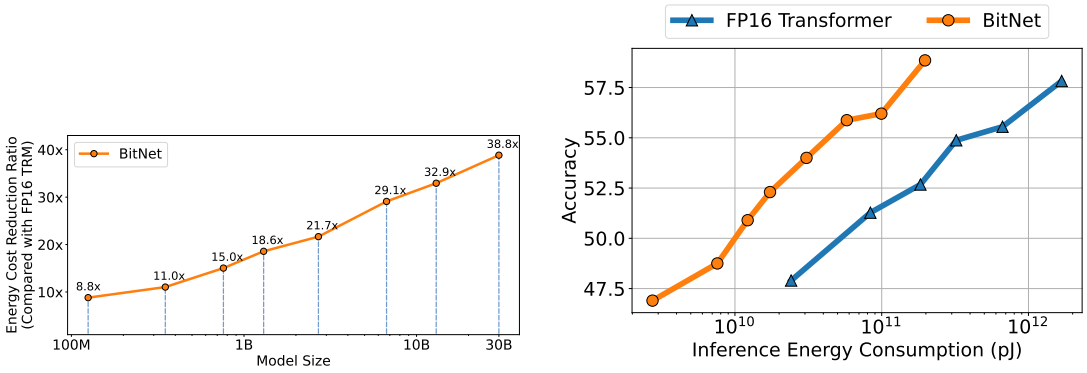
increases the consistency of the gradients required to cause a sign change (i.e. a bit flip) [43]. They also suggest that similar to using ADAM or Momentum in conventional NNs, a non-zero threshold τ avoids rapid back and forth of weights when the gradient reverses on a bit flip.

This novel approach effectively eliminates the need for latent weights and, as a result, the training process is simplified and the number of hyperparameters is reduced to 2. Furthermore, Bop reduces the number of parameters required for training (thus reducing memory requirements) to a single full precision value rather than 2 and 3 for Momentum and ADAM respectively.

The results presented by the authors show that Bop can achieve state-of-the-art results on the ImageNet dataset and outperforms the state-of-the-art BNNs on the CIFAR-10 dataset.

2.3.3 Binary Transformers

While most of the literature regarding BNNs focuses on CNNs, there is a growing interest in binarizing Transformers due to their higher energy consumption and memory requirements. In October 2023, Wang et al. [4] proposed BitNet, a 1-bit Transformer architecture that uses binary weights and activations. While their approach is tailored to Large Language Models (LLMs), their conclusions show potential for this type of architecture in other domains. The results presented suggest that, when training from scratch, BitNet can achieve competitive accuracy and perplexity while significantly reducing memory requirements and energy consumption.



(a) Energy Consumption Ratio

(b) Energy Consumption Comparison

Figure 2.12: Energy consumption and accuracy comparison between models based on FP16 Transformers and BitNet. Images taken from [4].

2.3.4 Binary LSTMs

Ardakani et al. [44] proposed a method to successfully train LSTM networks with binary weights. The recurrent transition of a LSTM is given by:

$$f_t = \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f), \tag{2.21}$$

$$i_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i), \tag{2.22}$$

$$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o), \tag{2.23}$$

$$g_t = \tanh(W_{gh}h_{t-1} + W_{gx}x_t + b_g), \tag{2.24}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t, \tag{2.25}$$

$$h_t = o_t \circ \tanh(c_t), \tag{2.26}$$

where $\{W_{fh}, W_{ih}, W_{oh}, W_{gh}\} \in \mathbb{R}^{d_h \times d_h}$, $\{W_{fx}, W_{ix}, W_{ox}, W_{gx}\} \in \mathbb{R}^{d_x \times d_h}$ and $\{b_f, b_i, b_o, b_g\} \in \mathbb{R}^{d_h}$ denote the recurrent weights and bias. The parameters $h \in \mathbb{R}^{d_h}$ and $c \in \mathbb{R}^{d_h}$ are hidden states. The logistic sigmoid function and Hadamard product are denoted as σ and $\circ$, respectively. The updates of the LSTM parameters are regulated through a set of gates: f_t, i_t, o_t, and g_t.

The method proposed consists of representing each weight w as $w = \alpha w^b$ where α is a fixed scaling factor for all the weights used to normalize them. Then the weights are binarized following the Stochastic Binarization described in Equation (2.6). Finally, they batch normalize the vector-matrix multiplications of Equation (2.21).

The results presented suggest that the proposed work can achieve competitive performance with minimal accuracy loss.

2.4 Binary Neural Networks Applied to ASR

The use of BNNs in ASR is still in its infancy and there is a lack of literature regarding this topic because most of the research was focused on CNNs which are more suitable for image-processing tasks.

Xiang et al. [45] implemented an ASR model consisting of binary linear layers, obtaining comparable results to the baseline model (that followed the same architecture) on the TIMIT dataset while having an inference time $4\times$ faster. While this work showcased the potential of BNNs in ASR, the authors did not explore the use of layers beyond binary linear layers (e.g. binary convolutions), and the results were obtained using a model architecture of their choice and not based on a state-of-the-art model. The use of Binary Convolutional Neural Networks (BCNNs) was explored by Qian et al. [11] in conjunction with KD to distill the knowledge from a teacher model to a student binary model. The results suggest that

the BNN had $5-7\times$ faster inference while remaining competitive with the full precision model (a relative 15% accuracy degradation).

Guo et al. [13] explored the use of Binary Very Deep CNNs paired with CTC in ASR for the Chinese language. The results present further accentuate the potential of BNNs in ASR by achieving only a 13% accuracy degradation while reducing the memory size of the model by $14\times$ (in terms of MBytes).

Table 2.4: Comparison between the state-of-the-art BNN models in ASR. Qian et al. [11] uses the TIMIT [12] dataset while Guo et al. [13] uses the chinese language dataset ST CMD, published by an AI company Surf Technology. No information about this last dataset could be found on-line beyond what the authors revealed in their work.

Neural Network	Number of Parameters	Model Size	PER - Test Set
Qian et al. [11]	9.141M	7.05MByte (estimated)	20.1%
Guo et al. [13]	5.787M	2.07MByte	17.90%

2.5 Summary

This chapter presented the state-of-the-art for the ASR task and BNNs and how the latter can be applied to the former. It also presented the main challenges that arise when training BNNs and the solutions proposed by the literature. While BNNs are still in their infancy, the results presented in the literature suggest that they have the potential to be used in ASR and achieve competitive results while reducing memory requirements and inference time. The results presented suggest a promising path of research: binarize the Conformer model (or one of its more recent variants) and use these bigger models as teachers to distill knowledge into the binarized model.

On the other hand, although most of the research uses the deterministic approach to train BNNs it could be interesting to explore the use of the stochastic approach to train BNNs with the methodology presented in Section 2.3.2 the results with the deterministic approach.

3

Binary Conformer

3.1 Overview

This work proposes a novel model architecture that balances computational efficiency with speech recognition accuracy through strategic quantization and architectural optimizations. The architecture, illustrated in Figure 3.1, uses BNNs while maintaining full-precision layers where necessary. Furthermore, this work also proposes changes to the modules of the Conformer block and a new module to replace the Relative Multi-Head Attention module. The primary objective is to maximize computational performance through extensive quantization of model layers while keeping the increase in WER within acceptable bounds. This approach required careful consideration of which layers could be effectively binarized without significantly degrading model performance. Furthermore, some layers or activation functions became redundant and were removed. All binary layers binarize their input with the $Sign(x)$ function. The Binary Conformer model proposed can be divided into its encoder and decoder. The encoder consists of a convolution subsampling (which reduces the time dimension by $4\times$), followed by a Binary Linear layer (with dimension equal to the enconder dimension), a Rectified Linear Unit (ReLU) activation and a BN, followed by N Binary Conformer Blocks in succession. The decoder is an LSTM.

The outputs of the encoder and decoder are concatenated and used as input to a full-precision linear layer (with dimension equal to the sum of the encoder and decoder dimensions) and a ReLU followed by a full-precision linear layer (with dimension equal to the number output dimension).

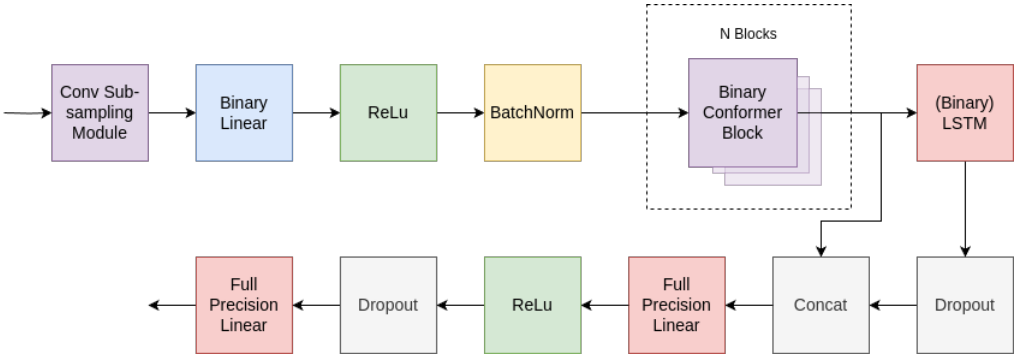


Figure 3.1: Binary Conformer Architecture. Layers marked with "(Binary)" are layers that can be binarized but, due to time constraints during training, could not be fully tested.

The layers kept in full-precision are the input layer of the model (i.e., the first layer of the convolution subsampling), the LSTM and the linear layers that follow it. As explained in Chapter 2, the first and last layers of the model should be kept in full-precision. The second to last linear layer joins the output of the LSTM and the encoder keeping it in full-precision had a negligeable impact on inference latency while having a significant impact on the accuracy metrics. The LSTM could be binarized and that was the initial plan. However, due to time constraints, that layer was kept in full-precision to allow for a more in-depth analysis of the effects of binarizing the encoder. Furthermore, this layer accounts for a small portion of the inference time.

To compensate for the inherent limitations of binary networks, several modifications were applied focusing on improving information flow during the forward pass. These changes were specifically designed to maintain rich feature representations despite the reduced precision in binary layers. A key modification was the strategic increase in model dimensions while reducing model depth, resulting in a net increase in the total number of parameters. This adjustment provided the model with greater representational capacity at each layer, helping to offset the constraints imposed by binarization. Another crucial modification was the inclusion of additional shortcuts to preserve information flow.

The training procedure was adapted to accommodate the dual nature of the architecture, which combines both full-precision and binary layers. This hybrid approach required specialized training techniques to ensure proper gradient flow through binary layers while maintaining the advantages of traditional floating-point training where appropriate.

Table 3.1: Comparison between the Conformer and Binary Conformer architectures.

	Conformer - Small	Binary Conformer
Number of layers	16	12
Sub-sampling dim	144	256
Encoder dim	144	256
Decoder dim	320	640
Number of parameters (Total)	10.4M	25.2M
Number of parameters (Binary)	0	4.2M
Number of parameters (FP32)	10.4M	21.0M
Model size (Theoretical)	39.29MByte	18.83MByte

3.2 Binarizing Individual Layers

Binarizing a full-precicion layer is more complex than simply swapping one for the other due to the downsides of quantization. In fact, at first glance, the binarized model appears more complex than its full-precision counterpart. Every full-precision layer is replaced by its binary counterpart followed by a ReLU activation function a BN and the $Sign(x)$ activation function. Due to the use of shortcuts that take the output of the BN layers, the $Sign(x)$ activation function is applied to the inputs of binary layers rather than to the outputs of the BN.

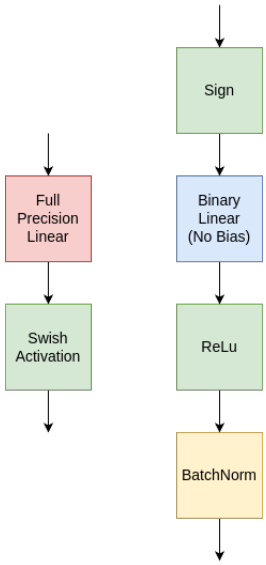


Figure 3.2: Comparison of a full-precision layer (left) to the binarized version (right).

This setup is very peculiar compared to ”traditional” full-precision layers, but each modification is crucial to maximize the model's accuracy while exploiting as much as possible from the benefits of binary layers.

The importance of the inclusion of BN layers cannot be overstated and proved crucial for managing the numerical range [46]. They fulfill a wide range of roles in addition to their expected benefits

(i.e., ensuring stable training). As such, they were placed after every binary layer (and their activation function). Firstly, they allow for the removal of biases of the subsequent binary layer since they already have a learnable offset. This simplifies the computation of those layers. Furthermore, that bias can help in implementing the effect of a "shifted Hard Tanh" [47]. This effectively means that each layer has two activation functions: the ReLU function before the BN and the Sign function after. This creates an "unbalanced activation distribution" that significantly improves the accuracy of the model [47]. Finally, the BN layers also work as a scaler to compensate for the approximation error caused by the quantization [48]. In fact, with all quantized networks it is typically proposed to include a scaling factor to improve accuracy. However, in this case, the inclusion of a scaler not only would add expensive full precision operations and increase the size of binary layers, but it would also reduce the model's accuracy [48].

Another key change is the use of a ReLU instead of any other activation function and to place it before the BN as proposed by Bulat et al. [49].

$$Relu(x) = max(0, x) = \frac{x + |x|}{2} \tag{3.1}$$

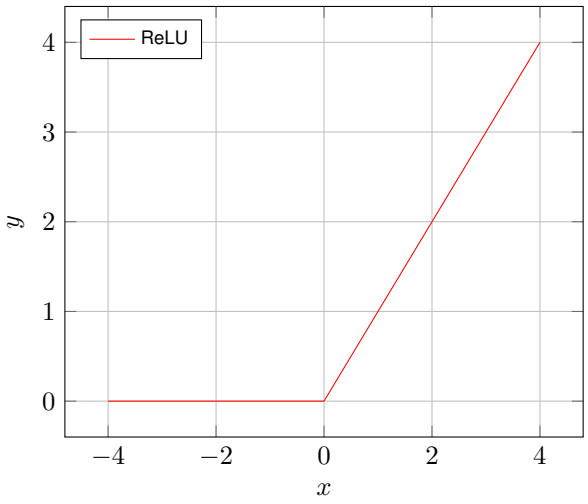


Figure 3.3: The ReLU activation function.

Even though the authors applied only to CNNs, empirical testing showed that this method could be applied to all binary layers with great success. On one hand, it causes the "unbalanced activation distribution" mentioned previously improving the model's accuracy. On the other hand, it significantly less computationally expensive than other activation functions. However, traditionally the ReLU is placed after the BN. But doing so in with BNN would be catastrophic because applying the $Sign(x)$ function to the output of a ReLU would create a tensor of exclusively the values $+1$ (0 is treated as a positive number when using BNNs).

While Shift-Based Batch Normalization has been proposed in some binary neural network imple-

mentations, standard Batch Normalization was chosen for the architecture. This decision was based on empirical observations showing that the computational benefits of shift-based operations were marginal at inference time and were outweighted by the also marginal decrease in recognition accuracy.

Finally, at inference time, the binary layer, ReLU and BN can be optimized for faster inference. Furthermore, when creating the model, these ”blocks” follow each other in succession and, in those cases, binary layers, ReLU, BN and $Sign(x)$ can be fused into a single layer. Thus, significantly improving inference latency. However, the fusion cannot happen when the output of the BN is also used in a shortcut.

3.3 Binary Conformer Blocks

The transition from the original Conformer to Binary Conformer involved several crucial modifications to the core Conformer block. The most significant change was the replacement of standard feed-forward and convolutional modules with their binary counterparts. The Binary Conformer Blocks consist of two macaron-like Binary Linear layers with residual connections sandwiching the Binary SummaryMixing and convolution modules, followed by a LayerNorm.

Another key change, is the replacement of the extremely computationally intensive Relative Multi-Head Attention module with the Binary SummaryMixing module. Initially, the swap was made with a full precision SummaryMixing module which reduced inference latency by both being a less complex layer but also removing the need of positional encodings. However, the much simpler nature of SummaryMixing allowed the use of the proposed Binary SummaryMixing module which further improved the model's latency and reduced the model's size.

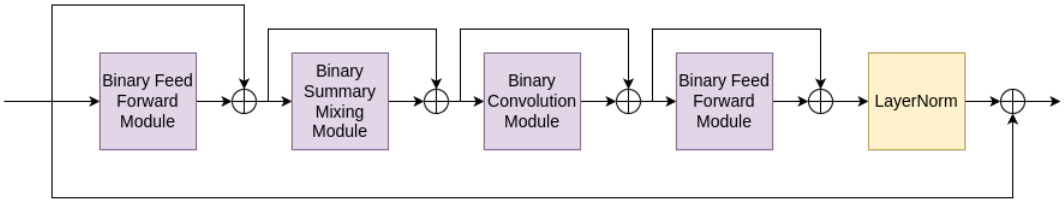


Figure 3.4: Binary Conformer Block architecture.

Finally, the dimensionality of the model was increased while the number of blocks was decreased (i.e., widening the model while reducing its depth). These modifications were carefully balanced to maximize inference performance while minimizing the increase in error rates. A comparison summary between the original Conformer and Binary Conformer is presented with Table 3.1.

The Binary Conformer, despite having $2.5\times$ more parameters, achieves significantly reduced model size due to the binary representation of weights in most layers. However, the model size could be further

reduced if some of the optimizations used when the model is loaded were applied when saving the model (e.g., only saving τ for the BN when possible).

3.4 Binary Convolution Sub-Sampling Module

The proposed Binary Convolution Sub-Sampling module, despite its name, includes a full-precision layer. The first layer of the model should always be in full-precision, thus the first 2-D convolution is in full-precision. The module consists of a 3×3 2-D convolution, a ReLU activation function and a BN followed by a 3×3 Binary 2-D convolution, a ReLU activation function and a BN. Both convolutions have a stride of 2, creating an output with a time dimension reduced by $4\times$ (i.e., the module compresses the time dimension from 10ms frames to 40ms frames).

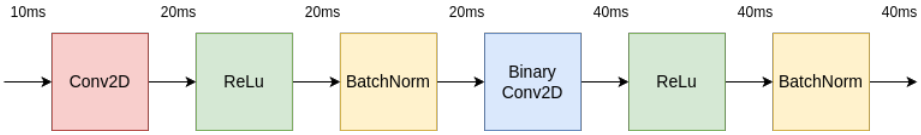


Figure 3.5: The Convolution Sub-sampling module.

Another crucial architectural change was implemented in the subsampling mechanism but was reverted on the final model. In the original Conformer, the subsampling output dimensionality was directly tied to the model's hidden dimension. In Binary Conformer, they were decoupled due to performance concerns. The modified subsampling layer would operate with a lower dimensionality in comparison, which is then projected to the model's dimensionality through the linear layer present between the subsampling module and the encoder blocks.

Reducing the dimensionality of the subsampling layers had an adverse impact on the accuracy of the model while also significantly reducing the inference latency. The trade-off did not seem worthwhile since the performance goals were achieved and sacrifices had to be made to decrease the error rates.

The subsampling module is the most computationally expensive module of the whole model and accounts for approximately 21% of the total execution time. Further improvements to this layer can be explored such as the Fast Conformer mentioned in Chapter 2 but could not be tested for this work. Since the model was trained using KD, using the Fast Conformer sub-sampling module would halve the output time dimension of the student which would then be incompatible with the output dimensions of the teacher model. However, this avenue could be explored if Binary Conformer is trained from scratch.

3.5 Binary Feed-Forward Module

The Binary Feed-Forward module suffered the lesser amount of changes from its original design. The Swish activation function was removed since it is more computationally intensive.

$$f(x) = x \times sigmoid(\beta x) = \frac{x}{1 + e^{-\beta x}} \tag{3.2}$$

The Binary Feed-Forward module consists of a LayerNorm, followed by a Binary linear layer (with an expansion factor of 4), a ReLU activation function and a BN, followed by a Binary linear layer (projection back to the encoder dimension), a ReLU activation function and a BN.

The residual connection was changed from the original half-step residual connection (i.e., applying $\times 0.5$ to the residual connection) to a full residual connection. This change improved both information and gradient flows.

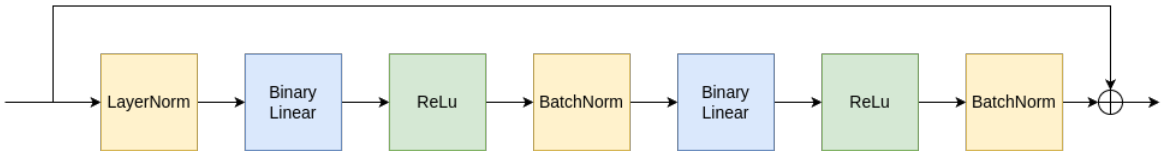


Figure 3.6: The feed-forward module.

3.6 Binary Convolution Module

The Binary Convolution Module went through multiple iteration, but simpler and more direct binarization approach was the more successful. Point-Wise convolution layers, which perform 1×1 convolutions across channels, might initially appear of limited value when constrained to binary weights. This constraint, effectively reduces the effect of those layers to applying one of 2 masks to their input. In some cases, the Point-Wise convolutions can be detrimental to the model's ability to learn [49]. However, the concerns over this limitation are addressed when looking at the Binary Convolution module as a whole. The Point-Wise Convolutions serve as dimensional expansion layers before the Depth-Wise convolution and as a dimensional compression after it. The Binary Convolution module consists of a LaryernNorm, followed by a 1-D Binary Point-Wise convolution (with a number of filters double the size of the encoder dimension), a ReLU and a BN, followed by a 1-D Binary Depth-Wise convolution (with a kernel size of 31 and dimension equal to the previous layer), a ReLU and a BN, followed by a second 1-D Binary Point-Wise convolution (with a number of filters equal to the encoder dimension), a ReLU and a BN.

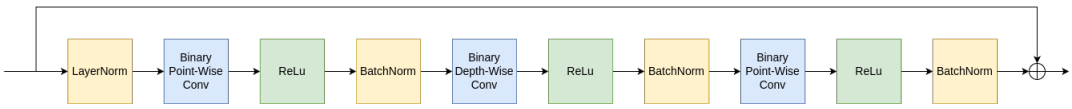


Figure 3.7: The Convolution Module.

3.7 Binary SummaryMixing Module

The Binary SummaryMixing module represents a significant optimization over the attention mechanism originally used (Relative Multi-Head Attention), which has quadratic time and memory complexities (i.e., $O(N^2)$). Thus, those layers are extremely computationally expensive even when processing shorter sequences. On the other hand, SummaryMixing provides a more efficient alternative with linear complexities ($O(N)$).

The advantages of swapping the attention mechanism for SummaryMixing extend beyond its computational efficiency. The module's simpler architecture makes it more straightforward to implement and optimize. A key innovation in this work is the binarization of the SummaryMixing module, creating Binary SummaryMixing. This binarization further reduces both inference latency, memory usage and the model's size. The combination of linear complexity and binary operations makes Binary SummaryMixing particularly well-suited for edge devices.

However, as mentioned in Chapter 2, SummaryMixing can be used as a replacement of the attention module in this model due to the particularities of ASR.

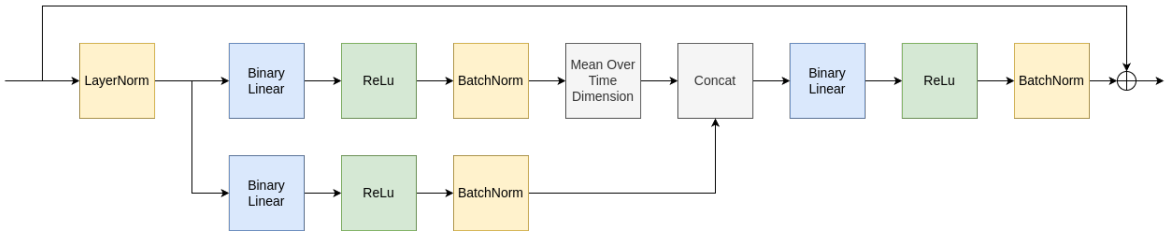


Figure 3.8: The SummaryMixing module.

3.8 Training the Binary Conformer

Training a state-of-the-art BNN from scratch for ASR presents several significant challenges that make it virtually impossible and fundamentally suboptimal. Firstly, BNNs require more training epochs than their full-precision counterparts. For this task, this would translate to several hundred epochs (each taking around 10h with the available hardware) since Conformer models take around 150 epochs to train. Even beyond this practical reason, KD was the better approach to training. Not only did it reduce the amount

of training epochs needed, but it is also a very effective method to train quantized models. Furthermore, attempts at training the model from scratch ended with it getting stuck at a local minimum.

The teacher model chosen for the KD process was a Conformer model from TensorFlowASR [50] (version r.1.0.3) trained on LibriSpeech [51] and openly available on GitHub. The teacher model has worse accuracy metrics than the ones presented by the original authors of the Conformer architecture [8] which limited the maximum accuracy the student model could achieve. The loss function chosen to train the model was the Kullback–Leibler divergence with a temperature of 2 (see (2.3)).

Table 3.2: Teacher model characteristics.

	Teacher Model
WER - Test Clean	7.87%
WER - Test Other	15.73%
CER - Test Clean	2.83%
CER - Test Other	7.67%
Number of Parameters	10.441M
Model Size	42.0MByte
Number of layers	16
Encoder Dimensions	144
Decoder Dimensions	320
Number of Subwords	1031
Batch Size	2
Trained Epochs	120

The proposed model has both binary and full-precision weights, so a hybrid optimizer was chosen to train it. This optimizer includes two optimizers Adam for the full-precision weights and the Binary Optimizer for the binary weights. The model was trained for 5 epochs with a batch size of 32 (which translates to 8788 steps per epoch) on a single node of the Deucalion supercomputer with 4 NVidia-a100-80GB Graphics Processing Units (GPUs). Each epoch had a training time of around 10 hours. The Binary Optimizer had a constant gamma rate of 1^{-4} and a threshold of 1^{-8}. The Adam optimizer had beta1 of 0.8, a beta2 of 0.98, an epsilon of 1^{-9} and the learning rate was a Transfomers Learning Rate Schedule with 10000 warm-up steps.

Unexpectedly, the approach of using a hybrid optimizer did not yield decisively better results compared to using only ADAM as an optimizer.

$$lr(step) = d_{model}^{-0.5} \cdot \min(step^{-0.5}, step \cdot warmup\_steps^{-1.5}) \tag{3.3}$$

d_{model} is the model dimension, in this case the dimension of the encoder. Regularization was removed from all layers given its adverse effects on BNNs. Moreover, all dropout layers were removed for the same reason, except when used between two consecutive full-precision layers. The momentum parameter was set to 0.9 for all BN layers due to the non-continuous nature of flipping weights.

The dataset chosen to train the student model was LibriSpeech [51], which consists of over 283

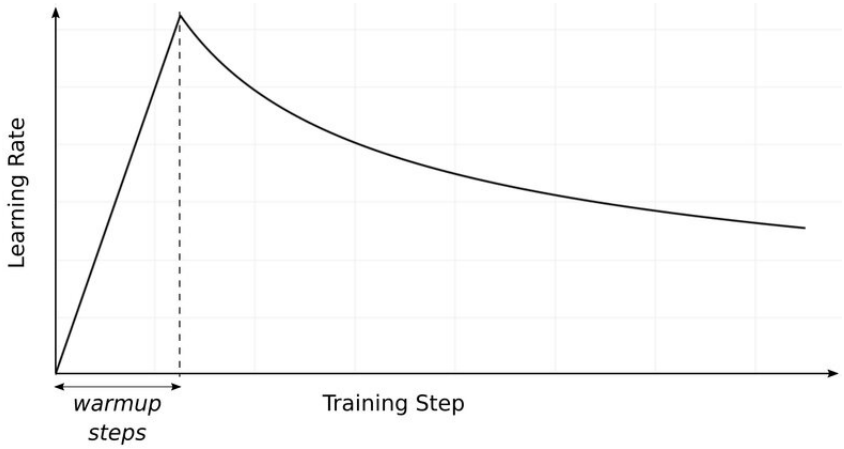
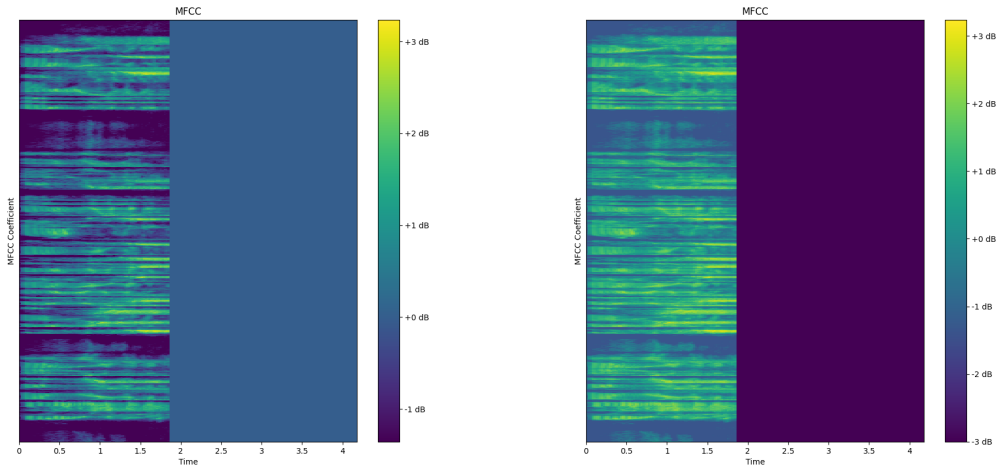


Figure 3.9: Transformer Learning Rate Schedule. Image take from [5].

thousand audio clips of variable length with a sampling rate of 16kHz. Inputs are created from these samples by, firstly, applying the feature extraction process (see Chapter 2). 80 MFCCs were generated per frame have a size of 25ms and a stride of 10ms. Each input is then normalized.

Every sample of the dataset has a different length, so each input had to be padded to match the biggest sample length of each batch. Conventionally, the inputs are padded with 0. However, since the feature extraction process already normalized the values and the padding was applied after, the padding value was set to -3 since it falls outside the normal range of values.



(a) Input padded with 0

(b) Input padded with -3

Figure 3.10: Comparison between the inputs padded with 0 and padded with -3. When using the value 0, the padding appears to contain relevant information when that is not the case.

3.9 Summary

This chapter proposed the novel Binary Conformer architecture that strategically combines binary and full-precision layers to optimize computational efficiency while maintaining comparable WER and CER for ASR. The architecture employs a hybrid encoder-decoder structure, where the encoder is mostly fully binarized while the decoder was kept with full precision. Following the current state-of-the-art, the input and output layers were kept with full-precision to preserve model performance.

The successful implementation of the binarization of the Conformer model required several crucial architectural modifications. Each binary layer is accompanied by BN to manage numerical ranges, provide scaling and use its bias to both use it as a learnable offset (removing the need to include a bias on binary layers) and create an equivalent effect to using a "Shifted Hard Tanh" with a learnable threshold value. ReLU activations were placed before BN layers to create beneficial unbalanced activation distributions. They also replaced the original activation functions with a much simpler computation that can be further optimized with layer fusion (explored in Chapter 4). To compensate for the loss of information caused by the reduced precision in binary layers, the architecture increased model width while reducing depth, maintaining representational capacity through a greater number of parameters at each layer. Additional shortcuts were incorporated throughout the network to preserve information (inference) and gradient (training) flow between layers.

The Binary Conformer introduces several specialized modules that optimize the original Conformer architecture. The Binary Convolution Sub-sampling Module combines full-precision and binary convolutions for efficient feature extraction. The Binary Feed-Forward Module simplifies the original design by removing the Swish activation and modifying residual connections. The Binary Convolution Module maintains both point-wise and depth-wise convolutions. Most notably, the computationally intensive Relative Multi-Head Attention mechanism was replaced with a Binary SummaryMixing module, which achieves linear computational complexity instead of quadratic.

Training this hybrid architecture required careful adaptation of standard approaches. KD from a full-precision teacher model was essential, as training from scratch proved impractical and suboptimal. The training process employed a hybrid optimizer that combined Adam for full-precision weights with the Binary Optimizer for binary weights. The learning rate scheduling and regularization approaches were modified to accommodate the characteristics of binary layers, and special consideration was given to padding values in the input data to maintain clear distinctions between actual features and padding.

Through these comprehensive modifications and optimizations, the proposed architecture demonstrates that strategic binarization of neural networks can significantly reduce model size and computational requirements while maintaining acceptable performance for speech recognition tasks. This approach offers a promising direction for deploying sophisticated speech recognition systems on resource-constrained devices.

4

Inference Optimization With a Custom Runtime

The development of a custom runtime for model inference proved to be a critical component of the work. The language chosen was Zig [52], a 100% inter-compatible language with C with a great way of creating generic functions through what is called ”comptime” [53]. This approach not only significantly improved latency and reduced memory usage compared to TensorFlow but also enabled a detailed examination of each optimization's impact, guiding both further optimizations and architectural changes to the model. The combination of binary operation optimizations, efficient memory management, and architectural considerations resulted in substantial speedup over naive implementations.

4.1 Binary Representation and Operations

The first optimization involved changing the binary representation from (-1,1) $\rightarrow$ (0,1) to (1,-1) $\rightarrow$ (0,1) and replacing XNOR operations with XOR operations. The reason behind this modification is due to the

ARMV8 Instruction Set Architecture (ISA) which lacks a vector XNOR instruction [54] but does include a vector XOR instructions. By changing the representation, each Multiply and Accumulate (MAC) goes from taking 3 instructions (XOR, NOT and popCount) to taking only 2 (XOR and popCount). The change has no cost associated given that it is performed when reading the saved models' weights.

Table 4.1: Changes to the representation of weights and XNOR operations to XOR operations. Conventional representation on the right and updated representation on the left.

w	w	x^b	w^b	$x \odot w$
-1	-1	0	0	1
-1	1	0	1	0
1	-1	1	0	0
1	1	1	1	1

w	w	x^b	w^b	$x \oplus w$
-1	-1	1	1	0
-1	1	1	0	1
1	-1	0	1	1
1	1	0	0	0

4.2 Custom Packed Binary Arrays

Custom packed arrays of single-bit unsigned integers were developed to represent weights and intermediate tensors. Initially, the chosen underlying type for the arrays was 64-bit unsigned integers since the target platforms are 64-bit architectures. This approach allowed to interpret these arrays as arrays of 64-bit unsigned integers and then perform the equivalent of 64 multiply-and-accumulate operations using only one XOR and one popcount instructions:

$$\text{dot\_product}(a, b) = \text{popcount}(a \oplus b). \tag{4.1}$$

This computation is particularly efficient because modern CPU architectures provide hardware-level support for popcount operations through dedicated instructions.

However, using 64-bit unsigned integers is not universally optimal, even on 64-bit architectures. In fact, while it was the clear choice when running on the AMD Ryzen 5 5600H, when running the runtime on the RaspberryPi5's Broadcom BCM2712, using 32-bit unsigned integers yielded a small advantage.

Therefore, the choice of the underlying type for the custom bit-packed array is heavily depended on the target platform.

4.3 Layer Fusion

Another key advantage of BNNs is that their peculiar characteristics allow the fusion of layers into a single one with much greater computational efficiency. In fact, even though Binary Conformer replaces every full-precision layer with a block of three layers (Binary layer, ReLU and BN), that increase in layers

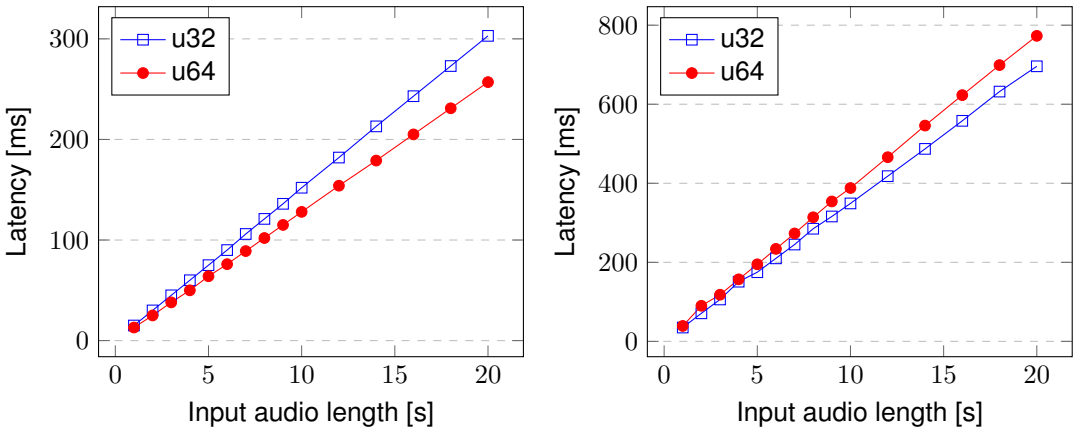


Figure 4.1: Effect of the underlying type of the custom array on the single core inference latency. Tested using the runtime's custom allocator on AMD Ryzen 5 5600H (left) and RaspberryPi5's Broadcom BCM2712 (right).

is offset, in part, by fusing the block (and the next $Sign(x)$) into a single step. However, this fusion can only occur effectively if the next layer is another binary layer and the output of the BN is not used in a residual connection.

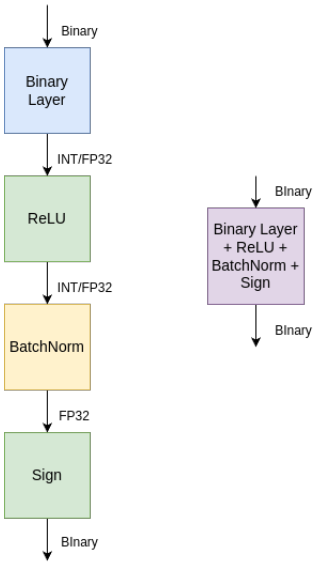


Figure 4.2: Layer fusion at inference time.

As explained in Chapter 2, at inference time, the BN can be simplified and fused wit the previous binary layer and the following $Sign(x)$ activation function. The simplification creates a new parameter (τ) that works as a new threshold value for the binarization (Equation (2.16) and Equation (2.17)). It is equivalent to implementing a "shifted Hard Tanh" activation function with a learnable shift.

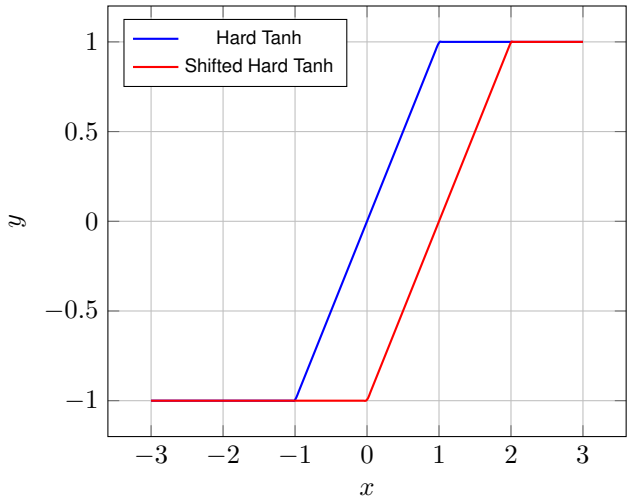


Figure 4.3: The Sifted Hard Tanh Function.

4.4 Pre-computation of common operations and quantization of parameters

When a model's weights are loaded, several mathematical operations that remain constant throughout inference can be pre-computed and stored, reducing the computational overhead during actual inference time. This optimization approach involves careful analysis and rearrangement of the mathematical equations that define each layer's operations. One such example is the threshold value τ of BN layers, mentioned in the previous section.

Quantization opportunities arise particularly in cases when layer fusion is possible. In these scenarios, the precision requirements of intermediate computations can be reduced without impacting the final binary output. Once again, one such example is the threshold value τ of BN layers. Since, τ is used as the binarization threshold and the MAC operations of binary layers are always integers, τ can be converted to an integer (rounding away from 0).

4.5 Optimized Memory Access Patterns

After thouroughly profiling the naive implementations of the runtime, it was clear that it was a memory bound task and improving memory access patterns and cache hit rate was paramount. Therefore, several strategies were employed to optimize memory access. Firstly, all layer weights and tensors use 1D-array representation instead of a matrix (i.e., an array of pointers of arrays of pointers for 3 dimensions). On one hand, using a single array improves data locality (since all memory is allocated in a contiguous chunk), therefore, also improving cache hit rate. On the other hand, allows to easily

reorder the dimensions and weights (e.g., channels first or channels last, Figure 4.4) improving temporal and cache locality. These changes simplify cache accesses making them more predictable which will maximize the effectiveness of the pre-fetcher of the Central Processing Unit (CPU).

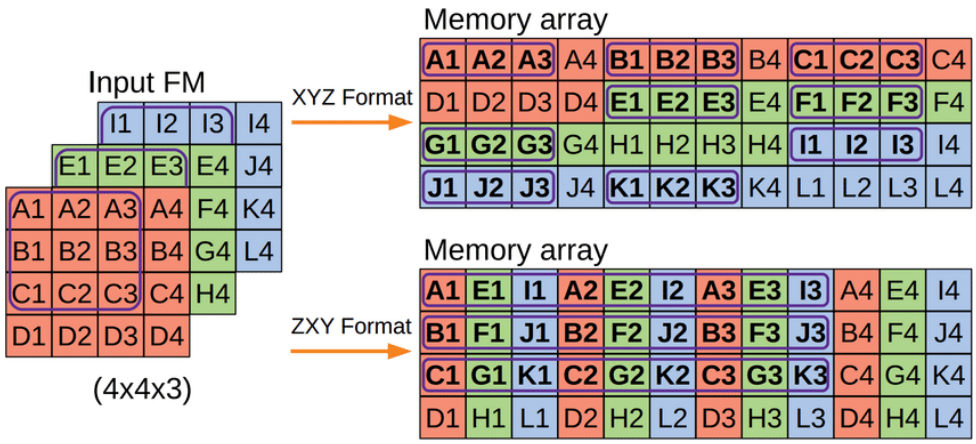


Figure 4.4: Comparison between channels last (XYZ format) and channels firt (ZXY format). Image taken from [6].

Additionally, each loop was individually profiled and, through empirical testing, the access patterns of arrays inside loops were altered to maximize performance. These changes were geared towards to layers with more irregular access patterns like the attention layer. Furthermore, in some cases, implementing more for-loops with fewer computations per loop, improved cache hit rates by accessing fewer arrays simultaneously.

Finally, both binary and full-precision convolutional layers benefited from the Image to Column (Im2Col) transformation [55] (Figure 4.5), which reformats the input data to enable more efficient matrix multiplication. It first creates an intermediary matrix of patches of size $kernel\_size \times kernel\_size \times channels$ which will allows to perform a conventional matrix multiplication.

4.6 Compiler optimizations

The first optimization was using the optimized float mode available in Zig [56]. This is equivalent to `-ffast-math` in GCC. Using optimized floats enables a set of aggressive floating-point optimizations that can significantly improve performance at the cost of strict IEEE compliance. It allows the compiler to make assumptions about floating-point arithmetic that may not always hold true, such as assuming that Not a Number (NaN) and infinity don't occur, treating mathematically equivalent operations as computationally equivalent, and reordering floating-point operations. This can lead to faster code execution but may produce slightly different results compared to strictly compliant floating-point computations. Empirical testing showed that this change did not change the predictions at inference time but significantly reduced the inference latency.

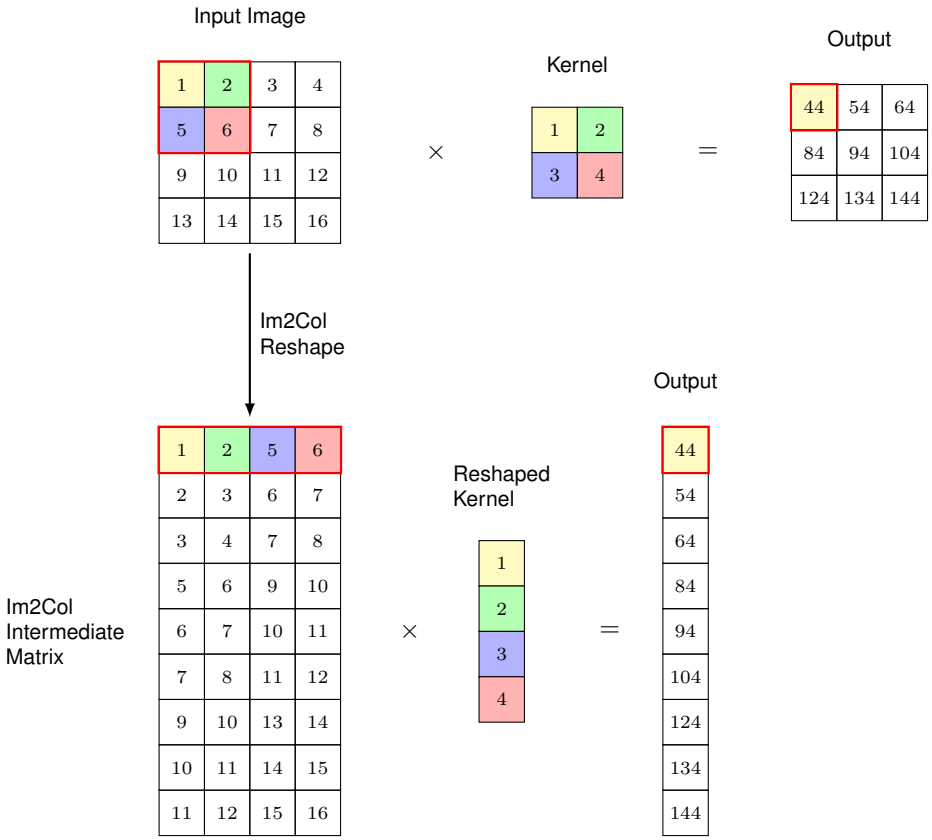


Figure 4.5: Im2Col transformation example.

Further increases in performance were achieved by carefully selecting the compilation flags. Using `-OReleaseFast` (equivalent to `-O3` in C) was a crucial yet obvious choice. On the other hand, giving more specific information about the target device (i.e., using `-target x86_64-linux.6.9.3...6.9.3-gnu.2.35` instead of the generic `-target x86_64-linux`) also increased performance. However, giving more specific details to the compiler was not always the optimal choice. In fact, using a generic CPU target (i.e., `-mcpu x86_64_v3` for the AMD Ryzen 5 5600H, and `-mcpu generic`) yielded significantly higher performance than using the specific flags appropriate for each of them.

4.7 Floating-Point Optimizations

Even though the focus of this work was on the binary layers, full-precision also had to be optimized achieve the goal of improving overall performance of the runtime. The other significant hurdle in optimizing full-precision layers was improving the Instructions Per Clock (IPC) of the matrix multiplications. While significant improvements could be achieved by re-implementing most of the know optimization methods, it was near impossible to match the optimization level of OpenBlas [57] on dot products and

matrix multiplication. Not only is OpenBlas highly optimized on a software level, but it also implements optimizations based on the specific hardware architecture. Nonetheless, empirical testing showed that, on some rare occasions, not using this library was the optimal choice since it had lower performance.

4.8 Memory Allocator

Each layer creates an output tensor and, possibly, some internal temporary helper tensors. This implies allocating and freeing a large quantity of memory. By using Zig's Arena Allocator on top of `libc`'s `malloc()` and setting the `retain_capacity` flag, it allows the allocator to be ”pre-heated” by allocating a large enough buffer to accomodate all previously done allocations. The arena allocator is a memory management strategy that allocates memory in large, contiguous blocks and allows for efficient allocation of smaller objects within these blocks. It provides fast allocation by simply incrementing a pointer and doesn't require individual deallocation of objects. Instead, the entire arena can be freed at once when it's no longer needed. When running the prediction function in a loop with identical audio chunk sizes (i.e., it's typical use case), the allocator will:

1. on the first iteration, allocate memory normally (while counting the amount of Bytes allocated) and free all memory at once at the end of the iteration when the arena is freed;
2. on the second iteration will allocate a large enough buffer to hold all allocations needed previously, no frees will happen;
3. on the third iteration and then on, the arena is ”pre-heated” so no more allocations or frees occur, significantly reducing inference latency.

Furthermore, changing the allocator or creating a custom one is trivial in Zig. The only code that needs changes is the allocator declaration, the rest of the code is automatically optimized (e.g., when using the arena allocator, all calls to `free()` are removed at compile time).

The downside of using such an allocator is that it causes an explosion of memory usage (as explored in Chapter 5) when dealing with larger inputs. As such, either a hybrid approach or using the conventional C allocator present in `libc` with `malloc()` could be more suited in situations where memory usage is a limiting factor concern.

4.9 Summary

The custom runtime development process not only yielded significant performance improvements but also provided invaluable insights into the behavior of binary neural networks in the context of ASR. These

optimizations and architectural decisions pave the way for further research into additional architecture changes to the model and software optimizations.

This chapter detailed the development and optimization of a custom runtime implemented in Zig for efficient model inference. The runtime's design integrated multiple optimization strategies that worked in concert to achieve significant performance improvements over both naive implementations and TensorFlow.

The runtime optimized representations by switching from (-1,1) $\rightarrow$ (0,1) to (1,-1) $\rightarrow$ (0,1) and leveraged the custom packed binary arrays, with target-specific underlying types (32-bit or 64-bit). This approach enabled efficient computation of multiple multiply-and-accumulate operations using single XOR and popcount instructions. Layer fusion techniques further enhanced performance by combining binary layers, ReLU, BN, and $Sign(x)$ activation into a single step, when binary layers are chained one after the other and the output of the BN is not used on a residual connection.

Memory optimization played a crucial role in the runtime's performance. Profiling and decompiling the runtime guided both the optimizations implemented and changes to the model's architecture with the aim of minimizing inference latency. Tensor were transformed by changing their representation to 1D-arrays (improving data locality) and dimension reordering (improving cache utilization and temporal locality). The implementation of Im2Col transformations significantly sped-up convolution operations while sacrificing a bit of memory usage.

These memory optimizations were complemented by layer fusion and pre-computation and quantization of constant operations, particularly in BN layers, where the threshold value can be computed during model loading and quantized to integer values without compromising output accuracy.

Compiler-level optimizations formed another crucial aspect of the runtime's performance enhancement. The runtime uses optimized float mode and compilation flags were carefully selected and tested to achieve the best results. Finally, the runtime integrated OpenBlas for certain full-precision operations, while maintaining custom implementations where they proved more efficient.

5

Results

This chapter is dedicated to the analysis of the results produced during the development of this work. Firstly, a comparison between the WER and CER of Binary Conformer, the teacher model and the original Conformer will be drawn. Then, the remainder of the chapter is dedicated to performance (latency and memory wise) comparisons between different Binary Conformer both on different modes of the custom runtime and on TensorFlow and the teacher model on TensorFlow. Finally, the trade-offs between model accuracy and computational requirements will be explained.

All results were collected on one of two devices, always with the same settings. The first one is a laptop with an AMD Ryzen 5 5600H CPU, with 32GB of Random Access Memory (RAM) running at 3200 MegaTransfers (MT) per second, running POP-OS22.04. Test were run on high performance mode. The second one is a RaspberryPi5 with a Broadcom BCM2712 CPU, with 8GB of RAM running at 4267 MT per second, running Ubuntu22.04. The fan was manually set to full speed to ensure the CPU wouldn't thermal throttle.

All tests were run on a single core by using the `taskset` linux command and memory usage was gathered using `/usr/bin/time -v`. The choice of testing using a single core was made due to the objectives of this work. The goal was to optimize a state-of-the-art model in such a way that it could be

viable to use it on edge devices. Those devices usually do not have a GPU nor sufficient cores to allow for a multi-threaded approach without impacting other running processes.

Each data point is an average of 5 runs for that particular configuration. Neither inference times nor memory usage statistics include the pre-processing of the raw audio signal. Inference times were calculated per prediction and not total running time of the program. This means, it does not include program setup, loading the model or creating the input sample. Data-points were only taken after a first prediction that compiles the model. Likewise, data-points for the custom runtime were only taken after pre-heating the custom allocator (when it was used) which takes 2 predictions. The underlying type for the custom packed array was u64 and u32 when testing the runtime on the AMD Ryzen 5 5600H and Broadcom BCM2712 respectively.

5.1 Model Accuracy

The accuracy of the Binary Conformer was tested using the test set of LibiSpeech and compared to its teacher model on that same test set. The state-of-the-art BNN models in ASR were tested on different datasets and use phonemes (which are typically easier to fit to but are less useful on real applications) instead of subwords as output. Therefore, a direct comparison is of limited value. Nonetheless, as reference, Qian et al. [11] and Guo et al. [13] achieved Phoneme Error Rate (PER) of 20.1% and 17.9% respectively.

Unfortunately, there wasn't enough time to effectively train the model which is reflected on the number of epochs trained. On one hand, the best architecture was discovered very late and yet surpassed the error rates of other iterations that were trained over 25 epochs. Furthermore, the time to train such a complex architecture on a difficult task like ASR with a very large dataset was greater than anticipated. In fact, with the initial setup, each epoch took over 22 hours to train with a batch size of 8, thus making the iterative process not only very slow but also ineffective due to such a small batch size. Additionally, even after finding a good candidate architecture for the Binary Conformer, training BNNs require many more epochs than their full-precision counterparts. Since the Teacher model took over 100 epochs to fully train, the Binary Conformer is expected to require at least the same amount. Although KD helps to reduce the number of epochs needed for training, it is unclear how large that reduction is. At the latter quarter of the thesis, access to the Deucalion supercomputer was extremely beneficial and reduced the training time of each epoch to 10 hours and allowed the training of bigger sizes (i.e., 32). Even then, testing too many iterations of the Binary Conformer without any of them proving clearly superior early enough significantly limited the accuracy potential of the trained model.

All these factors significantly impacted the results obtained which seem. Even with such limited training, Binary Conformer is a very promising architecture that shows promising results if it could be

extensively trained. However, while evaluating these results, it is relevant to keep in mind that, contrary to other tasks, in ASR error rates do not have an upper bound (i.e., can reach values significantly above 100%).

Table 5.1: Comparison between the original Conformer Small model, the Teacher model and the Binary Conformer model.

	WER Test Clean (%)	CER Test Clean (%)	WER Test Other (%)	CER Test Other (%)
Conformer Small	2.7	-	6.3	-
Teacher Model	7.87	2.83	15.73	7.67
Binary Conformer	**56.7**	**34.2**	**72.0**	**53.0**

Binary Conformer achieved significantly worse error rates when compared to Teacher Model. However, given the very limited training time the model had, these are extremely promising results for this architecture. If allowed to further train, the error rates would significantly improve and close the gap.

5.2 Inference Latency

In both test setups, using the custom runtime represents a meaningful improvement of inference latency over both the Binary Conformer and the Teacher model running on TensorFlow, particularly on shorter input lengths. That is particularly relevant since the typical use case would fall within the range of short inputs. The improvements are even more noticeable on the lower power cores of the RaspberryPi5. Furthermore, the latency is low enough to allow for real-time predictions.

Another important thing to note is the worse inference latency of Binary Conformer running on TensorFlow when compared to the Teacher model. This is expected since, in TensorFlow, all weights are treated as full-precision weights, thus not leveraging the benefits of using binary layers. In fact, in TensorFlow, Binary Conformer is treated simply as a model with $2.5\times$ the number of parameters of the Teacher model.

5.2.1 AMD Ryzen 5 5600H

Running the Binary Conformer model on the custom runtime with the custom allocator offers an average improvement of $3.3\times$ over using TensorFlow. The maximum improvement was a $6.2\times$ speed-up with the 1s input and the minimum improvement was $2.7\times$ with the 14s input. When using the `libc` `malloc()` the average improvement is $3.2\times$, the maximum improvement was a $6.2\times$ speed-up with the 1s input and the minimum improvement was $2.6\times$ with the 14s input.

Running the Binary Conformer model on the custom runtime with the custom allocator offers an average improvement of $1.29\times$ over the Teacher model on TensorFlow. The maximum improvement was

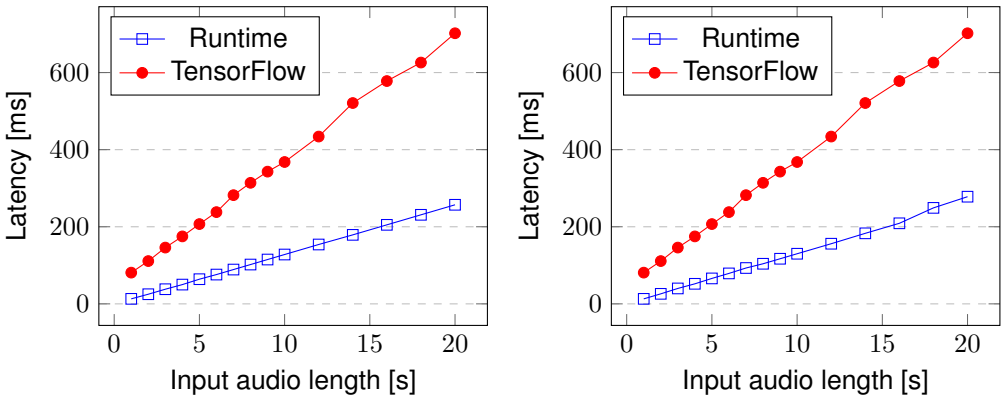


Figure 5.1: Single core inference latency of the Binary Conformer architecture using the custom allocator (left) and `libc malloc()` (right) compared to Binary Conformer on TensorFlow on AMD Ryzen 5 5600H.

a $1.61\times$ speed-up with the 1s input and the minimum improvement was $1.22\times$ with the 5s input. When using the `libc malloc()` the average improvement is $1.26\times$, the maximum improvement was a $1.61\times$ speed-up with the 1s input and the minimum improvement was $1.17\times$ with the 7s input.

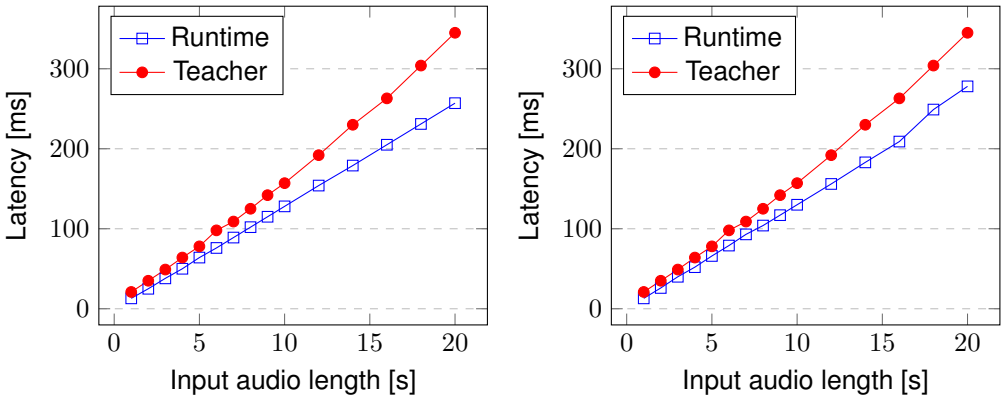


Figure 5.2: Single core inference latency of the Binary Conformer architecture using the custom allocator (left) and `libc malloc()` (right) compared to the Teacher model on TensorFlow on AMD Ryzen 5 5600H.

5.2.2 Broadcom BCM2712

Running the Binary Conformer model on the custom runtime with the custom allocator offers an average improvement of $5.2\times$ over using TensorFlow. The maximum improvement was an $8.5\times$ speed-up with the 1s input and the minimum improvement was $4.7\times$ with the 14s input. When using the `libc malloc()` the average improvement is $5.7\times$, the maximum improvement was a $9.9\times$ speed-up with the 1s input and the minimum improvement was $4.9\times$ with the 15s input.

A relevant aspect of these results is the fact that, with this test setup, using the custom runtime with the `libc malloc()` unexpectedly improved inference latency. This indicates that that allocator is the better suited for edge devices. This is particularly interesting because it is also the more memory efficient of the two (see Figure 5.7 and Figure 5.8). Thus, it is not necessary to sacrifice memory usage for better performance.

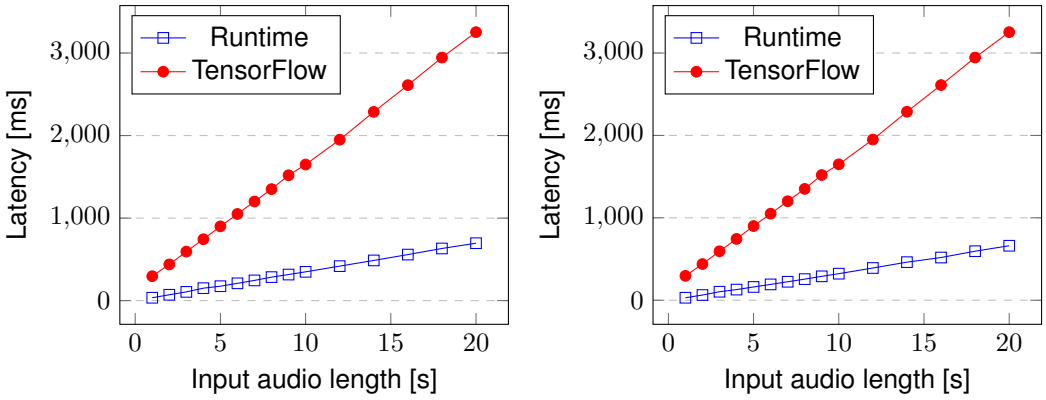


Figure 5.3: Single core inference latency of the Binary Conformer architecture using the custom allocator (left) and `libc malloc()` (right) compared to Binary Conformer on TensorFlow on Broadcom BCM2712.

Running the Binary Conformer model on the custom runtime with the custom allocator offers an average improvement of $2.5\times$ over the Teacher model on TensorFlow. The maximum improvement was a $2.8\times$ speed-up with the 4s input and the minimum improvement was $2.2\times$ with the 14s input. When using the `libc malloc()` the average improvement is $2.7\times$, the maximum improvement was a $3.3\times$ speed-up with the 1s input and the minimum improvement was $2.4\times$ with the 3s input.

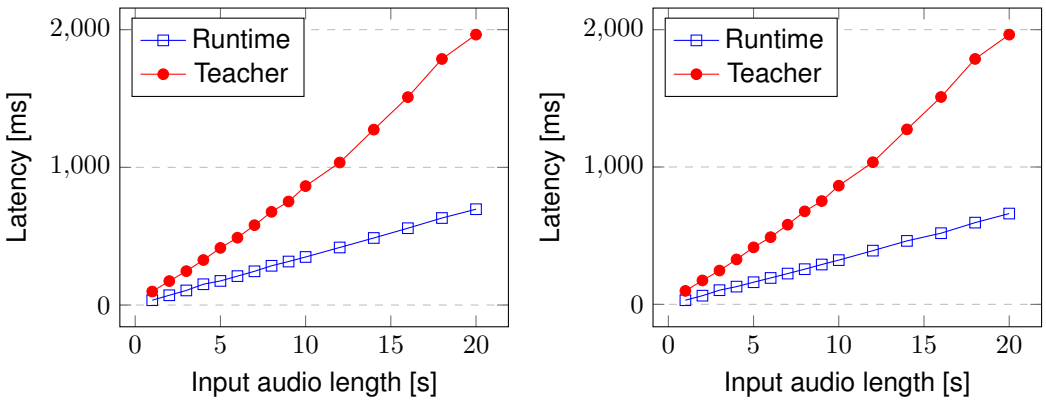


Figure 5.4: Single core inference latency of the Binary Conformer architecture using the custom allocator (left) and `libc malloc()` (right) compared to the Teacher model on TensorFlow on Broadcom BCM2712.

5.3 Memory usage

The remarks made about latency are mostly identical than those that are drawn about memory usage but with even bigger improvements. In both test setups, using the custom runtime represents an improvement of peak memory usage of 1 to 2 orders of magnitude over both the Binary Conformer and the Teacher model running on TensorFlow, particularly on shorter input lengths. That is particularly relevant since the typical use case would fall within the range of short inputs.

Another important thing to note is the much bigger peak memory usage of Binary Conformer running on TensorFlow when compared to the Teacher model. This is expected for the same reason mentioned in the previous section.

5.3.1 AMD Ryzen 5 5600H

Running the Binary Conformer model on the custom runtime with the custom allocator offers an average improvement of $37.8\times$ over using TensorFlow. The maximum improvement was $143.8\times$ with the 1s input and the minimum improvement was $10.6\times$ with the 15s input. When using the `libc malloc()` the average improvement is $86.6\times$, the maximum improvement was $259.0\times$ with the 1s input and the minimum improvement was $29.4\times$ with the 15s input.

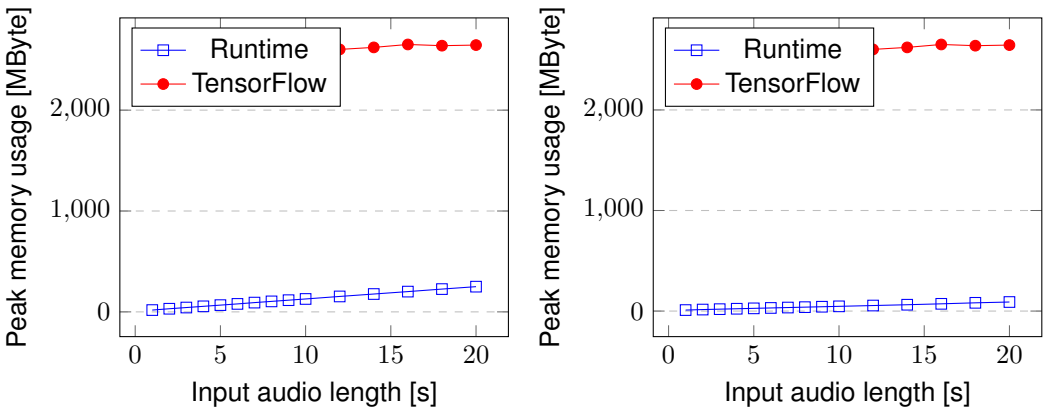


Figure 5.5: Single core peak memory usage of the Binary Conformer architecture using the custom allocator (left) and `libc malloc()` (right) compared to Binary Conformer on TensorFlow on AMD Ryzen 5 5600H

Running the Binary Conformer model on the custom runtime with the custom allocator offers an average improvement of $12.6\times$ over the Teacher model on TensorFlow. The maximum improvement was $47.6\times$ with the 1s input and the minimum improvement was $3.58\times$ with the 15s input. When using the `libc malloc()` the average improvement is $29.1\times$, the maximum improvement was $85.7\times$ with the 1s input and the minimum improvement was $9.9\times$ with the 15s input.

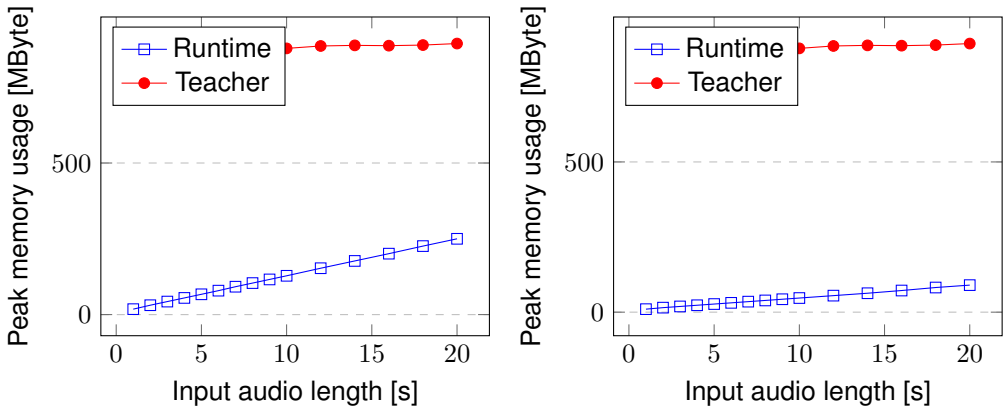


Figure 5.6: Single core peak memory usage of the Binary Conformer architecture using the custom allocator (left) and `libc malloc()` (right) compared to the Teacher model on TensorFlow on AMD Ryzen 5 5600H.

5.3.2 Broadcom BCM2712

Running the Binary Conformer model on the custom runtime with the custom allocator offers an average improvement of $31.8\times$ over using TensorFlow. The maximum improvement was $117.4\times$ with the 1s input and the minimum improvement was $9.3\times$ with the 15s input. When using the `libc malloc()` the average improvement is $70.2\times$, the maximum improvement was $185.9\times$ with the 1s input and the minimum improvement was $25.6\times$ with the 15s input.

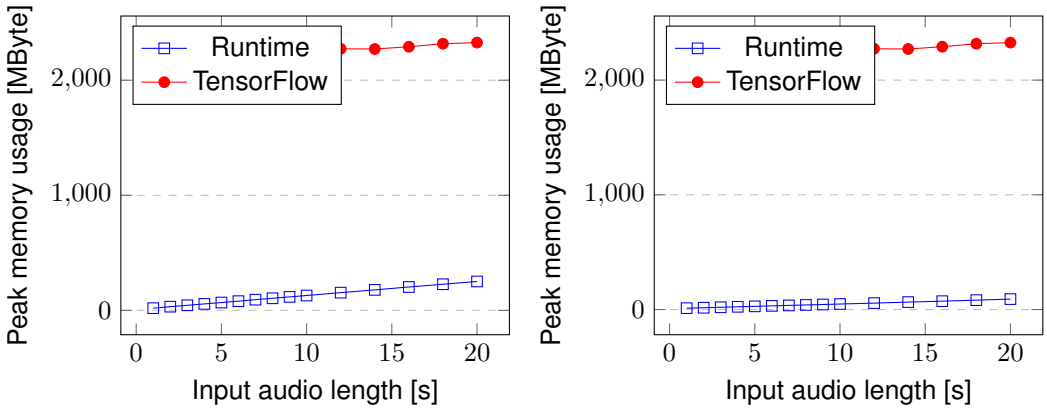


Figure 5.7: Single core peak memory usage of the Binary Conformer architecture using the custom allocator (left) and `libc malloc()` (right) compared to Binary Conformer on TensorFlow on Broadcom BCM2712.

Running the Binary Conformer model on the custom runtime with the custom allocator offers an average improvement of $9.8\times$ over the Teacher model on TensorFlow. The maximum improvement was $35.5\times$ with the 1s input and the minimum improvement was $2.8\times$ with the 15s input. When using the

`libc malloc()` the average improvement is $21.6\times$, the maximum improvement was $56.3\times$ with the 1s input and the minimum improvement was $7.8\times$ with the 15s input.

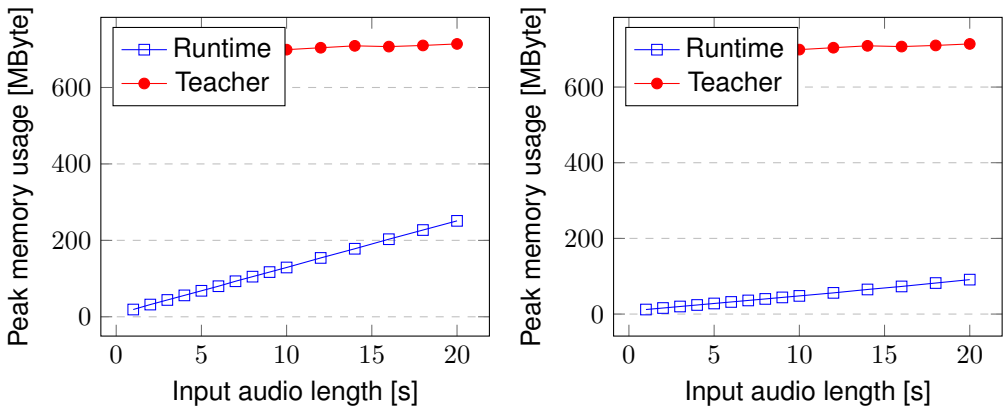


Figure 5.8: Single core peak memory usage of the Binary Conformer architecture using the custom allocator (left) and `libc malloc()` (right) compared to the Teacher model on TensorFlow on Broadcom BCM2712.

5.4 Energy Consumption

Energy consumption was measured on the RaspberryPi 5 by using the `pmic_read_adc` utility that reads the tension and current used by the system. It measures all power consumption with the exception of certain peripherals. Therefore, to ensure correct estimations, no peripherals were connected and the cooling fan was set to a fixed speed. With the values collected with that utility, it is possible to estimate the power consumption (in Watts) of the RaspberryPi. When running the tests, the current power consumption was polled and averaged. Multiplying that average with the time it takes to complete a single inference calculates the total energy consumption of that inference (in Joules).

Firstly, several measurements were taken to get the idle power consumption. This value will then be subtracted to the value obtained during testing to get the power consumption of the inference alone. Secondly, the measurements were taken while running the custom runtime and the Teacher model on TensorFlow. Both were limited to using a single core. Since both tasks had 100% utilization of the core, both had the same power consumption while running (i.e., the power consumption of a single core at full throttle). The idle power consumption was measured at $2.54W$ and the power consumption when testing was $5.07W$.

To obtain the total energy consumption for a single inference, we can use the inference times and multiply them by $5.07 - 2.54 = 2.53W$. However, since this is simply multiplying the values obtained in the previous section by a constant, the comparisons of energy consumption between the custom

runtime and Binary Conformer and the Teacher Model on Tensorflow are identical in to the comparison on inference times previously made. Therefore, it is concluded that the custom runtime achieves an average of $2.5\times$ improvement in energy consumption compared to the Teacher model.

5.5 Summary

Binary Conformer showed promising results to improve and enable ASR models on edge device. This chapter presented statistics of what was considered the adequate range of input lengths considering their typical use case. While the error metrics trail behind state-of-the-art full-precision models, it is a clear step forward to reduce the computational requirements of these models.

As expected, the lower power Broadcom BCM2712 CPU saw the biggest benefits from using the binary model achieving a relative inference latency of 37% when compared to the teacher model while also only using 4.6% as much memory. Furthermore, this directly equates to a relative energy consumption of 37%. In addition to these benefits, the model size is also significantly smaller (a $2\times$ reduction) making it easier to transfer and store.

Table 5.2: Comparison of the trade-offs between the teacher model and both runtime modes on AMD Ryzen 5 5600H.

	Relative WER Test Clean	Relative CER Test Clean	Relative WER Test Other	Relative CER Test Other	Relative Inference Latency	Relative Peak Memory Usage
Teacher Model	100	100	100	100	100	100
Binary Conformer using Custom Allocator	720.5	1208.5	457.7	691.0	37.0	4.6
Binary Conformer using malloc()	720.5	1208.5	457.7	691.0	37.0	4.6

Table 5.3: Comparison of the trade-offs between the teacher model and both runtime modes on Broadcom BCM2712. Since the custom allocator performed worse than using `libc malloc()` on both metrics, results are omitted.

	Relative WER Test Clean	Relative CER Test Clean	Relative WER Test Other	Relative CER Test Other	Relative Inference Latency	Relative Peak Memory Usage
Teacher Model	100	100	100	100	100	100
Binary Conformer using malloc()	720.5	1208.5	457.7	691.0	37.0	4.6

6

Conclusion

NN models have demonstrated remarkable achievements across various domains, but their growing complexity and size have presented significant challenges for edge device deployment. This thesis addressed these challenges by exploring BNNs in the context of ASR, successfully proposing, developing and implementing Binary Conformer, a novel architecture that significantly reduces computational and memory requirements while maintaining competitive performance.

The primary motivation behind this work was the need for efficient, on-device ASR systems that could be used on low-power devices without sacrificing either inference latency or accuracy. The goal was to explore the use of BNNs on much more complex architectures and prove it is a viable course. While cloud-based solutions offer certain advantages, their dependence on internet connectivity and associated costs make them impractical for many applications. On the other, locally run NN require extra hardware accelerators that will not be available on edge device. This work demonstrates that BNNs can provide a viable alternative, enabling local processing on resource-constrained devices while addressing privacy concerns and connectivity requirements.

6.1 Main Contributions

This thesis achieved several significant milestones. Firstly, it demonstrated that BNNs can be effectively applied to complex ASR tasks and models, expanding their application beyond previous state-of-the-art models that were heavily centered around CNN. Secondly, it proposed Binary Conformer, a binarized variant of the Conformer architecture, which incorporated crucial modifications that enhanced information flow and gradient propagation while preserving accuracy. Finally, through a custom inference executable, notable performance improvements were achieved, especially on low-end hardware, validating the approach's practical use-cases. Furthermore, it provides insightful knowledge that can guide both the creation of improved architectures and optimizations to the inference runtime.

6.2 Future Work

The results of this work suggest that BNNs represent a promising direction for deploying sophisticated NN on edge devices. These findings indicate that, with appropriate architectural modifications and optimization techniques, BNNs can achieve substantial reductions in computational and memory requirements of complex models while maintaining acceptable accuracy levels for practical applications.

Looking forward, this research opens several avenues for future work. Firstly, the model should be further trained to explore its full potential. Secondly, the decoder layer (i.e., the LSTM) now represents a significant proportion of the inference time. Binarizing it and pairing with Binary Conformer seems a promising path to further reduce computational requirements of the model. Thirdly, a more in-depth grid-search to find the best hyper-parameters could further improve the model's accuracy without requiring architecture changes. Finally, further analysis of the custom runtime could reveal more optimizations that could accentuate its current advantages.

Bibliography

[1] A. Baevski, H. Zhou, A. rahman Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *ArXiv*, vol. abs/2006.11477, 2020. [Online]. Available: https://arxiv.org/pdf/2006.11477.pdf

[2] V. Bozic, D. Dordevic, D. Coppola, J. Thommes, and S. P. Singh, "Rethinking attention: Exploring shallow feed-forward neural networks as an alternative to attention layers in transformers," *ArXiv*, vol. abs/2311.10642, 2023. [Online]. Available: https://arxiv.org/pdf/2311.10642.pdf

[3] T. Parcollet, R. van Dalen, S. Zhang, and S. Bhattacharya, "Summarymixing: A linear-complexity alternative to self-attention for speech recognition and understanding," 2024. [Online]. Available: https://openreview.net/forum?id=PoBB8n52oi

[4] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, "Bitnet: Scaling 1-bit transformers for large language models," *ArXiv*, vol. abs/2310.11453, 2023. [Online]. Available: https://arxiv.org/pdf/2310.11453.pdf

[5] J. L. García-Nava, J. Flores, V. Téllez, and F. Calderon, "Fast training of a transformer for global multi-horizon time series forecasting on tensor processing units," 07 2022.

[6] P. Miranda, D. Garigali Pestana, J. Lopes, R. Duarte, M. Véstias, H. Neto, and J. Sousa, "Configurable hardware core for iot object detection," *Future Internet*, vol. 13, p. 280, 10 2021.

[7] A. Mehrish, N. Majumder, R. Bharadwaj, R. Mihalcea, and S. Poria, "A review of deep learning techniques for speech processing," *Information Fusion*, vol. 99, p. 101869, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253523001859

[8] A. Gulati, C.-C. Chiu, J. Qin, J. Yu, N. Parmar, R. Pang, S. Wang, W. Han, Y. Wu, Y. Zhang, and Z. Zhang, Eds., *Conformer: Convolution-augmented Transformer for Speech Recognition*, 2020.

[9] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," *ArXiv*, vol. abs/2212.04356, 2022. [Online]. Available: https://arxiv.org/pdf/2212.04356.pdf

[10] A. Barcovschi, R. Jain, and P. Corcoran, “A comparative analysis between conformer-transducer, whisper, and wav2vec2 for improving the child speech recognition,” in *2023 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, 2023, pp. 42–47.

[11] Y. Qian and X. Xiang, “Binary neural networks for speech recognition,” *Frontiers of Information Technology & Electronic Engineering*, vol. 20, pp. 701–715, 2019. [Online]. Available: https://doi.org/10.1631/FITEE.1800469

[12] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, “Darpa timit acoustic phonetic continuous speech corpus cdrom,” 1993.

[13] L. Guo, Y. Deng, L. Tang, R. Fan, B. Yan, and Z. Xiao, “A chinese speech recognition system based on binary neural network and pre-processing,” in *2023 6th World Conference on Computing and Communication Technologies (WCCCT)*, 2023, pp. 129–134.

[14] S. Gondi and V. Pratap, “Performance and efficiency evaluation of asr inference on the edge,” *Sustainability*, vol. 13, p. 12392, 11 2021.

[15] T. Guo, “Cloud-based or on-device: An empirical study of mobile deep inference,” in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 184–190.

[16] T. Dotan and D. Seetharaman, “Big tech struggles to turn ai hype into profits,” *The Wall Street Journal*, 2023, (Accessed 10 October, 2024). [Online]. Available: https://www.wsj.com/tech/ai/ais-costly-buildup-could-make-early-products-a-hard-sell-bdd29b9f

[17] M. Tyson, “Microsoft lost $20 for every $10 copilot ai subscription: Report,” *Tom’s Hardware*, 2023, (Accessed 10 October, 2024). [Online]. Available: https://www.tomshardware.com/news/microsoft-lost-money-on-ai

[18] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[19] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, Q. Liang, D. Bhatia, Y. Shangguan, B. Li, G. Pundak, K. C. Sim, T. Bagby, S.-y. Chang, K. Rao, and A. Gruenstein, “Streaming end-to-end speech recognition for mobile devices,” in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6381–6385.

[20] T. N. Sainath, Y. He, B. Li, A. Narayanan, R. Pang, A. Bruguier, S.-y. Chang, W. Li, R. Alvarez, Z. Chen, C.-C. Chiu, D. Garcia, A. Gruenstein, K. Hu, A. Kannan, Q. Liang, I. McGraw, C. Peyser,

R. Prabhavalkar, G. Pundak, D. Rybach, Y. Shangguan, Y. Sheth, T. Strohman, M. Visontai, Y. Wu, Y. Zhang, and D. Zhao, "A streaming on-device end-to-end model surpassing server-side conventional model quality and latency," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6059–6063.

[21] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[22] J. Li, "Recent advances in end-to-end automatic speech recognition," *ArXiv*, vol. abs/2111.01690, 2021. [Online]. Available: https://arxiv.org/pdf/2111.01690.pdf

[23] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE Access*, vol. 7, pp. 19 143–19 165, 2019.

[24] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[25] R. Deo, A. Rathi, T. Rathod, J. Mevada, and V. Sagvekar, "Review of feature extraction techniques," in *Proceedings of the 3rd International Conference on Advances in Science & Technology (ICAST)*, 2020.

[26] Z. K. Abdul and A. K. Al-Talabani, "Mel frequency cepstral coefficient and its applications: A review," *IEEE Access*, vol. 10, pp. 122 136–122 158, 2022.

[27] S. Gandhi, P. von Platen, and A. M. Rush, "Distil-whisper: Robust knowledge distillation via large-scale pseudo labelling," *ArXiv*, vol. abs/2311.00430, 2023. [Online]. Available: https://arxiv.org/pdf/2311.00430.pdf

[28] R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2letter: an end-to-end convnet-based speech recognition system," *ArXiv*, vol. abs/1609.03193, 2016. [Online]. Available: https://arxiv.org/pdf/1609.03193.pdf

[29] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," *ArXiv*, vol. abs/2005.08100, 2020. [Online]. Available: https://arxiv.org/pdf/2005.08100.pdf

[30] Y. Zhang, J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le, and Y. Wu, "Pushing the limits of semi-supervised learning for automatic speech recognition," *ArXiv*, vol. abs/2010.10504, 2020. [Online]. Available: https://arxiv.org/pdf/2010.10504.pdf

[31] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural ’networks,” vol. 2006, 01 2006, pp. 369–376.

[32] G. E. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *ArXiv*, vol. abs/1503.02531, 2015. [Online]. Available: https://arxiv.org/pdf/1503.02531.pdf

[33] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’06. New York, NY, USA: Association for Computing Machinery, 2006, p. 535–541. [Online]. Available: https://doi.org/10.1145/1150402.1150464

[34] J. Li, R. Zhao, J.-T. Huang, and Y. Gong, “Learning small-size DNN with output-distribution-based criteria,” in *Proc. Interspeech 2014*, 2014, pp. 1910–1914.

[35] D. Rekesh, S. Kriman, S. Majumdar, V. Noroozi, H. Juang, O. Hrinchuk, A. Kumar, and B. Ginsburg, “Fast conformer with linearly scalable attention for efficient speech recognition,” *ArXiv*, vol. abs/2305.05084, 2023. [Online]. Available: https://arxiv.org/pdf/2305.05084.pdf

[36] M. Burchi and V. Vielzeuf, “Efficient conformer: Progressive downsampling and grouped attention for automatic speech recognition,” in *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2021, pp. 8–15.

[37] S. Kim, A. Gholami, A. E. Shaw, N. Lee, K. Mangalam, J. Malik, M. W. Mahoney, and K. Keutzer, “Squeezeformer: An efficient transformer for automatic speech recognition,” *ArXiv*, vol. abs/2206.00888, 2022. [Online]. Available: https://arxiv.org/pdf/2206.00888.pdf

[38] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1800–1807.

[39] M. Courbariaux, Y. Bengio, and J. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” *CoRR*, vol. abs/1511.00363, 2015. [Online]. Available: http://arxiv.org/abs/1511.00363

[40] M. Courbariaux and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1,” *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[41] Y. Bengio, N. Léonard, and A. C. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *ArXiv*, vol. abs/1308.3432, 2013. [Online]. Available: https://arxiv.org/pdf/1308.3432.pdf

[42] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *ArXiv*, vol. abs/1502.03167, 2015. [Online]. Available: https://arxiv.org/pdf/1502.03167.pdf

[43] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder, “Latent weights do not exist: Rethinking binarized neural network optimization,” in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d 'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/9ca8c9b0996bbf05ae7753d34667a6fd-Paper.pdf

[44] A. Ardakani, Z. Ji, S. C. Smithson, B. H. Meyer, and W. J. Gross, “Learning recurrent binary/ternary weights,” in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HkNGYjR9FX

[45] X. Xiang, Y. Qian, and K. Yu, “Binary deep neural networks for speech recognition,” 08 2017, pp. 533–537.

[46] J. Bethge, H. Yang, M. Bornstein, and C. Meinel, “Binarydensenet: Developing an architecture for binary neural networks,” *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 1951–1960, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:207901458

[47] H. Kim, J. Park, C. Lee, and J.-J. Kim, “Improving accuracy of binary neural networks using unbalanced activation distribution,” 06 2021, pp. 7858–7867.

[48] J. Bethge, H. Yang, M. Bornstein, and C. Meinel, “Back to simplicity: How to train accurate bnns from scratch?” *ArXiv*, vol. abs/1906.08637, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195218776

[49] A. Bulat and G. Tzimiropoulos, “Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources,” 03 2017.

[50] H. L. Nguyen. TensorFlowASR. (Accessed 10 October, 2024). [Online]. Available: https://github.com/TensorSpeech/TensorFlowASR/tree/r1.0.3

[51] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, “Librispeech: An asr corpus based on public domain audio books,” in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5206–5210.

[52] Zig Software Foundation. Zig. (Accessed 10 October, 2024). [Online]. Available: https://ziglang.org/

[53] ——. (Accessed 10 October, 2024). [Online]. Available: https://ziglang.org/documentation/0.13.0/#comptime

[54] Arm Limited, *Arm Architecture Reference Manual for A-profile architecture*, (Accessed 10 October, 2024). [Online]. Available: https://developer.arm.com/documentation/ddi0487/latest/

[55] V. Sze, Y. hsin Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, pp. 2295–2329, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:3273340

[56] Zig Software Foundation. (Accessed 10 October, 2024). [Online]. Available: https://ziglang.org/documentation/0.13.0/#setFloatMode

[57] Z. Xianyi. Openblas. (Accessed 10 October, 2024). [Online]. Available: http://www.openblas.net