TRU-Net Based AI Model Implementation by Pure C for Real-time Speech Enhancement

Yonghun Lee[1] and Minjung Kim[1] and Daejin Park[1*]
[1]School of Electronic and Electrical Engineering, Kyungpook National University, Daegu, Republic of Korea
*Correspondence to: boltanut@knu.ac.kr

***Abstract*—This paper presents a real-time speech enhancement AI model based on TRU-Net, implemented in Pure C for deployment on resource constrained edge devices. High-level machine learning frameworks such as PyTorch and TensorFlow can impose significant limitations in edge environments due to their memory footprint, power consumption, and real-time control constraints. Accordingly, this study emphasizes the importance of architecture design that prioritizes efficient memory utilization in addition to computational performance optimization. By implementing the model in Pure C, memory access patterns, buffer sizes, computational methods, and parallel processing structures are precisely controlled, enabling an experimental analysis of the trade-offs between performance and memory usage. Based on these analyses, an optimized NPU architecture is proposed that minimizes memory consumption, improves the efficiency of intermediate tensor and buffer management, and enhances parallel processing performance through loop unrolling. Experimental results demonstrate that the proposed Pure-C–based model significantly reduces per-frame memory usage and achieves more than a 70% reduction in execution time, validating its effectiveness for real-time, low-power edge AI systems.**

***Keywords*—Speech enhancememt; Environmental Noise Reduction; Deep Neural Network; TRU-Net; Pure-C model;**

I. Introduction

In the fields of Artificial Intelligence (AI) and Machine Learning (ML), inference refers to the process by which a trained model makes predictions or draws conclusions from previously unseen data. It enables machines to perform tasks that mimic human intelligence. Among deep learning models, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are commonly employed architectures for processing sequential data and are typically developed using Machine Learning (ML) frameworks such as TensorFlow and PyTorch. While ML frameworks provide high scalability and flexibility by offering a wide range of functions and libraries, they are typically loaded into memory as a whole. Due to this architectural characteristic, ML frameworks often face challenges in finely controlling system-level constraints such as cost, power consumption, and performance [1].

This study was supported by the BK21 FOUR project (4199990113966), the Basic Science Research Program (RS-2018-NR031059, 10%), (RS-2025-24322979, 10%) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education. This work was partly supported by an Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2022-0-01170, 20%) and (No. RS-2023-00228970, 30%) and (No. RS-2025-02218227, 20%) and (No. RS-2022-00156389, Innovative Human Resource Development for Local Intellectualization support program, 10%). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

Advancements in edge AI have enabled real-time decision-making by facilitating the direct execution of inference processes on local devices, thereby obviating the need to transmit data to centralized cloud servers. This transition is primarily driven by the need to reduce latency, enhance user privacy, lower data transmission costs, support real-time responsiveness, and enable operation in offline environments. When deploying AI models on resource-constrained edge devices, recent studies suggest that memory-centric architectural optimizations frequently yield greater efficiency than strategies focused solely on computational throughput [2]. Therefore, it is essential to experimentally evaluate and adjust various system parameters for finding the optimal trade-off between performance and memory usage.

Recently, the demand for edge AI models capable of removing ambient noise and reconstructing clean speech has grown significantly. Applications such as automatic speech recognition (ASR) [3], hearing aids [4], and emergency detection [5] require highly efficient and low-latency processing under resource-constrained environments. Deploying AI models on embedded edge devices necessitates resolving complex trade-offs among execution performance, memory usage, and power consumption. Pure C-based development enables fine-grained control over system resources, including memory access, buffer allocation, computational strategy selection, and parallel processing structures. These capabilities make the approach particularly suitable for embedded systems and edge AI scenarios, where precise resource management is critical.

In this study, a real-time de-noising AI model based on TRU-Net was implemented entirely in pure C for edge deployment. The model consists of multiple hierarchical CNN layers designed to remove environmental noise. To satisfy the strict requirements of cost, power, and performance, CNN inference must be carefully optimized, which is challenging when using high-level, Python-based frameworks. Therefore, an edge-oriented implementation in pure C was adopted.

The proposed model was designed to meet the following key requirements:

1) Maximizing computational efficiency through NPU architecture optimization.
2) Efficient memory utilization by adjusting buffer sizes and managing intermediate tensor storage.
3) Enhancing parallelism via loop unrolling and operation reordering.

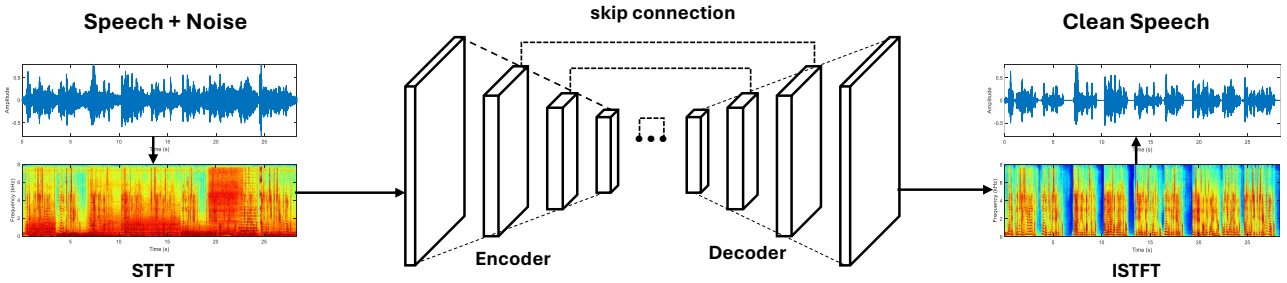

Fig. 1. The speech enhancement framework with Unet.

II. Background

This section describes the overall architecture of the proposed speech enhancement framework as shown in Fig. 1, which is built upon a TRU-Net based encoder–decoder structure.

A. Short Time Fourier Transform

In most deep learning-based audio processing applications, raw audio signals in the time domain are first converted into the time-frequency domain using the Short-Time Fourier Transform (STFT). The STFT transforms a 1D time-domain signal $x(t) \in \mathbb{R}^T$ into a 2D complex-valued spectrogram $X(f,t) \in \mathbb{C}^{F \times T'}$, where F denotes the number of frequency bins and T' is the number of time frames:

$$X(f,t) = \sum_{\tau=0}^{N-1} x(\tau + tH) \cdot w(\tau) \cdot e^{-j2\pi f\tau/N} \tag{1}$$

where $w(\tau)$ is a window function of size N, and H is the hop size. The resulting complex spectrogram $X(f,t)$ contains both magnitude and phase information:

$$X(f,t) = |X(f,t)| \cdot e^{j\phi(f,t)} \tag{2}$$

Either the magnitude spectrogram $|X(f,t)| \in \mathbb{R}^{F \times T'}$, or both real and imaginary parts are used as the input to convolutional or recurrent neural networks.

B. Phase Encoder

In Phase Encoder (PE), phase information is encoded to preserve temporal consistency during later reconstruction. It performs a series of complex-valued convolution operations and amplitude estimation. This module is designed to process the input complex spectrogram and encode the underlying temporal dynamics in a differentiable and learnable manner.

Let the input spectrogram be denoted by a pair of real-valued tensors $\mathbf{X}_r(f,t)$ and $\mathbf{X}_i(f,t)$, representing the real and imaginary parts of the STFT output, respectively. We express the complex-valued input as:

$$\mathbf{X}(f,t) = \mathbf{X}_r(f,t) + j\mathbf{X}_i(f,t) \tag{3}$$

Following the approach of complex convolution proposed in [6], the complex convolution is defined as:

$$\mathbf{Y}_r = \mathbf{X}_r * W_r - \mathbf{X}_i * W_i \tag{4}$$

$$\mathbf{Y}_i = \mathbf{X}_r * W_i + \mathbf{X}_i * W_r \tag{5}$$

Here, W_r and W_i denote the real and imaginary parts of the learnable kernel weights, and $*$ denotes convolution. This structure follows the property of complex multiplication, and has been shown to preserve both magnitude and phase structure [7].

After obtaining the real and imaginary components $\mathbf{Y}_r$ and $\mathbf{Y}_i$, we compute the amplitude (magnitude) spectrum:

$$|\mathbf{Y}(f,t)| = \sqrt{\mathbf{Y}_r(f,t)^2 + \mathbf{Y}_i(f,t)^2} \tag{6}$$

This amplitude is used either as a feature for subsequent encoder deep learning network or directly for inverse transformation via Magnitude Estimation and Adjustment (MEA).

C. Encoder

Tiny Recurrent U-Net (TRU-Net) based on a modified variant of U-Nets [8] is suitable for real-time speech enhancement. This architecture is designed to enable efficient decoupling of computations along the the frequency and time axis, allowing for real-time frame-by-frame processing.

Let $X \in \mathbb{R}^{T \times F}$ be the input spectrogram, where T denotes the number of time frames and F is the number of frequency bins.

U-Net-based models typically apply 2D convolution over both time and frequency axes:

$$Y = \text{Conv2D}(X) \Rightarrow \mathcal{O}(T \cdot F \cdot K_t \cdot K_f)$$

where K_t and K_f are the convolution kernel sizes in the time and frequency dimensions, respectively.

In contrast, the proposed TRU-Net architecture decouples the computation along these two axes. First, a 1D convolution is applied along the frequency axis:

$$Y' = \text{Conv1D}_f(X) \Rightarrow \mathcal{O}(F \cdot K_f)$$

The encoder of TRU-Net comprises a stack of six 1D Convolutional Neural Network (1D-CNN) layers, each implemented as a combination of pointwise and depthwise convoluations. These layers progressively reduce the feature map

dimensionality from 256 to 16 using strided convolutions. This downsampling effectively compresses the frequency resolution, which can result in the loss of fine-grained spectral information [9]. Such degradation is particularly detrimental to speech enhancement tasks, where accurately modeling the spectral content over a wide frequency range is essential.

To address this limitation, a Frequency-axis Self-Attention (FSA) block is applied into the encoder.

Let the input be $X \in \mathbb{R}^{F \times d}$, where F is the number of frequency bins and d is the feature dimension. The self attention mechanism computes:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{7}$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{8}$$

Here, the matrix product $QK^\top \in \mathbb{R}^{F \times F}$ encodes all pairwise interactions among frequency bins. Second, a FSA is applied along the frequency axis:

$$Y'' = \text{SelfAttention}_f(X') \Rightarrow \mathcal{O}(F^2 \cdot d)$$

Assuming d is relatively small and constant, this is often approximated as $\mathcal{O}(F^2)$ in practice. The use of Frequency-axis Self-Attention (FSA) enables the model to extract richer spectral representations by capturing long-range dependencies and global contextual relationships along the frequency axis, thereby enhancing its ability to improve speech quality.

D. Decoder

The decoder is composed of a Time-axis Gated Recurrent Unit (TGRU) block and 1D Transposed Convolutional Neural Network. The output of the encoder is pass through a Time-axis Gated Recurrent Unit (TGRU) layer to aggregate information along the temporal axis. The TGRU block consists of a Gated Recurrent Unit (GRU), a pointwise convolution, batch normalization (BN), and a ReLU in sequence. The Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) architecture specifically designed to model sequential data. It efficiently captures temporal dependencies and contextual information over time while mitigating the vanishing gradient problem through gating mechanisms. A GRU unit maintains a hidden state $\mathbf{h}_t$ at each time step t and updates it based on the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$ as follows:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1}) \tag{9}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1}) \tag{10}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1})) \tag{11}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{12}$$

Here, $\sigma(\cdot)$ denotes the sigmoid activation function, $\odot$ represents element-wise multiplication, and $\mathbf{z}_t$, $\mathbf{r}_t$ are the update and reset gates, respectively. These gates regulate the flow of information, allowing the GRU to selectively retain relevant temporal features and discard outdated ones.

To restore the output size to the original spectrogram size, a 1D-TrCNN block is employed. This block takes two inputs: (1) the output of the previous layer(TGRU) and (2) a skipped tensor from the encoder at the corresponding hierarchical level. These are then combined and upsampled to match the original resolution. The upsampling process refers to reconstructing the reduced temporal or frequency resolution back to its original size, which is a common operation in decoder architectures. By leveraging skip connections, a key feature of the U-Net architecture, this structure facilitates the recovery of temporal and spectral details in the reconstructed output.

E. Magnitude Estimation and Adjustment (MEA)

Magnitude Estimation and Adjustment (MEA) module refines the enhanced magnitude spectrum by applying phase information from PE. This process improves spectral accuracy, contributing to better reconstruction quality as a post-processing component within the overall speech enhancement network. The process is composed of the following four steps:

- **Magnitude and Phase Decomposition:** The complex STFT input is separated into magnitude and phase components. The phase is obtained via the `atan2` operation applied to the real and imaginary parts.
- **Magnitude Correction via Masking:** Multiple convolutional layers are used to estimate magnitude masks. These masks are activated by ReLU and applied to the original magnitude to generate enhanced magnitude features.
- **Phase Adjustment and Reconstruction:** A separate convolutional path is utilized to predict the phase mask. The estimated mask is added to the original phase, and the adjusted phase is used to reconstruct real and imaginary components using cosine and sine operations.
- **Complex STFT Recomposition:** The final real and imaginary parts are stacked and permuted to form the enhanced complex STFT output, preserving the original sequence and spectral structure.

F. Inverse Short-Time Fourier Transform

Inverse Short-Time Fourier Transform (ISTFT) reconstructs the enhanced time-domain signal from the MEA. The Inverse Short-Time Fourier Transform (ISTFT) is used to reconstruct the enhanced time-domain waveform from the output of the Magnitude Estimation and Adjustment (MEA) module. Specifically, the complex spectrogram $\hat{S} \in \mathbb{C}^{T \times F}$, where T is the number of time frames and F is the number of frequency bins, is transformed back into a one-dimensional time-domain signal $\hat{x} \in \mathbb{R}^L$ using ISTFT.

Given the estimated magnitude $\hat{M}_{t,f}$ and phase $\hat{\phi}_{t,f}$, the complex spectrum is reconstructed as:

$$\hat{S}_{t,f} = \hat{M}_{t,f} \cdot e^{j\hat{\phi}_{t,f}}, \tag{13}$$

where t and f denote the time frame and frequency bin indices, respectively.

The final waveform is then obtained by applying the ISTFT with an overlap-add method:

$$\hat{x}[n] = \text{ISTFT}(\hat{S}_{t,f}), \tag{14}$$

where a synthesis window function and hop size are used to ensure smooth reconstruction.

This process allows the model to generate an enhanced time-domain signal that reflects both the estimated magnitude and corrected phase information from the MEA module.

III. Proposed Methode

Low latency refers to minimizing the time delay between input and output. This is critical for maintaining real-time responsiveness in practical applications. In systems such as hearing aids, simultaneous interpretation, robotic voice control, and real-time meeting noise suppression, excessive latency can degrade performance by causing delayed feedback or unnatural interaction.

To address this issue, we adopt a frame-by-frame processing that does not rely on future context. Each incoming audio frame is processed and output sequentially in real time. In this study, we implement a Pure-C model based on TRU-Net Architecture, ensuring low-latency operation suitable for time-sensitive environments.

Algorithm 1 outlines the proposed frame-by-frame speech enhancement pipeline. The process begins with system initialization and data loading through `load_data()`. This includes reading the input waveform and model parameters.

Each frame is processed sequentially to ensure low-latency inference. The input signal is first transformed to the time-frequency domain via STFT. Phase-related features are extracted and passed through the encoder. The decoder reconstructs the enhanced representation using a TGRU block and transposed convolution.

MEA refines the spectral magnitudes, and ISTFT converts the signal back to the time domain. Finally, the enhanced signal is written using `write_output()`. This structure supports real-time processing without future context.

Algorithm 1: Speech Enhancement Process

```
init()
load_data()
l_1 = frame dimension
for i = 1 to l_1 do
    STFT()
    PHASE_ENCODER()
    ENCODER()
    DECODER()
    MEA()
    ISTFT()
write_output()
```

The proposed model is fully implemented in Pure C, offering several benefits for real-time and embedded speech enhancement. First, C provides low-level control over memory and computation, making it suitable for optimization on DSPs, microcontrollers, and embedded SoCs. Second, it avoids runtime dependencies and dynamic memory allocation, ensuring predictable and reliable execution. Finally, Pure C allows easy integration into existing audio pipelines without external libraries, resulting in a lightweight and portable solution. These features make the model ideal for on-device, real-time speech enhancement in edge environments.

Algorithm 2 describes a general 1D convolution process that supports multi-channel inputs and outputs with dilation. For each output position lo and each output channel co, the algorithm accumulates a weighted sum over all input channels and kernel positions. Dilation is applied when accessing input elements to enlarge the receptive field without increasing the kernel size. A bias term is added per output channel, and boundary checks ensure valid input indexing.

Algorithm 2: General 1D Convolution Alogorithm

Input: Input $X[L * C_{\text{in}}]$, weights $W[C_{\text{out}} * C_{\text{in}} * K]$, bias $B[C_{\text{out}}]$

Output: Output $y[L * C_{\text{out}} * C_{\text{in}}]$

L: output length, C_{in}: input channels, C_{out}: output channels

K: kernel size, d: dilation factor

for $lo = 1$ *to* L **do**
  for $co = 1$ *to* C_{out} **do**
    $mac = b[co]$
    for $ci = 1$ *to* C_{in} **do**
      for $k = 1$ *to* K **do**
        if $0 < x\_idx < L$ **then**
          $mac+ = X[ci * lo + d * k] * W[co * ci * k + ci * K + k]$
    $Y[co * lo + lo] = mac$

As illustrated in the general CNN algorithm, convolution operations are typically implemented using six or seven nested `for` loops. These loops iterate over multiple dimensions, including input length, kernel size, channels, and output positions. As a result, convolution accounts for a substantial portion of the computational cost in deep neural networks (DNNs). In this paper, we propose an optimization approach to perform convolution operations more efficiently. The proposed method focuses on enabling parallel execution of multiply-accumulate (MAC) operations, which is one of the key challenges in Neural Processing Unit (NPU) design. To achieve this, several compiler-level optimization techniques are applied:

- **Fixed configuration**: Uses fixed kernel and channel sizes to eliminate dynamic control logic.
- **Branch elimination**: Removes conditional branches to improve execution determinism.
- **Loop unrolling**: Reduces loop control overhead and increases instruction-level parallelism.
- **Loop reordering**: Adjusts loop nesting to enhance data locality and register reuse.
- **Memory reordering**: Optimizes memory access patterns to improve cache efficiency.

TABLE I
LAYER-SPECIFIC OPTIMIZATION STRATEGIES IN TRU-NET

Category	Operation	Key Parameters	Optimization Method	Effect
PE	CNN	$C_{in} = 1$, $C_{out} = 4$, $L = 1$, $K = 3$	Full loop unrolling	Maximum parallelism
Encoder	Point/Depth-wise CNN (6 layers)	$K = 1$, $P = 1$ $K = 3$, $P = 1$	Loop re-ordering	Improved cache locality
Decoder	Pointwise/Transposed CNN (6 layers)	$C = 128$, $K = 1$, $P = 1$ $C = 128$, $K = 3$ or 5, $P = 1$	Memory re-ordering	Improved cache locality
MEA	CNN	$C_{in} = 4$, $C_{out} = 1$, $L = 256$ $K = 3$	Loop unrolling	Simplified structure

Table I summarizes the optimization techniques applied to each major component of the TRU-Net model and their corresponding effects. For each network module, including the Phase Encoder (PE), Encoder, Decoder, and Mask Estimation and Adjustment (MEA), appropriate optimization strategies were selected based on the computational characteristics of the underlying operations. Loop unrolling was primarily employed in the PE and MEA modules to maximize parallelism, while loop re-ordering and memory re-ordering were applied to the Encoder and Decoder convolutional layers with smaller and larger feature dimensions, respectively, to improve data locality and memory access efficiency. As a result, the proposed optimizations effectively reduce runtime while preserving the original network functionality, demonstrating the suitability of the optimized TRU-Net architecture for real-time and resource-constrained embedded environments.

This design benefits from loop unrolling, which removes the overhead associated with loop control and branching. By eliminating conditional branches, the algorithm achieves a predictable control flow that improves pipeline utilization and enhances instruction-level parallelism. In particular, Algorithm 3 performs multiply-accumulate (MAC) operations using 4 parallel MAC, while Algorithm 4 exploiting 8 parallel MAC. This deeper unrolling significantly increases data throughput and better utilizes available hardware resources. The use of a fixed configuration and loop reordering also enables static memory access patterns that improve cache efficiency and reduce memory latency. Furthermore, each output channel computation begins with bias pre-accumulation, reducing redundant memory accesses and allowing for tighter integration of multiply-and-accumulate (MAC) operations.

These optimizations collectively yield a highly efficient implementation suitable for low-power embedded platforms such as Micro-Controller, DSP, and NPU. The predictable timing, low memory overhead, and compact code structure support real-time, low-latency inference required in on-device speech enhancement systems.

Algorithm 3: Proposed 1D Convolution Alogorithm in Phase Encoder(PE)

Input: Input $X[L * C_{\text{in}}]$, weights $W[C_{\text{out}} * C_{\text{in}} * K]$, bias $B[C_{\text{out}}]$
Output: Output $y[L * C_{\text{out}} * C_{\text{in}}]$

$mac = B[0]$
$mac+ = X[0] * W[0]$
$mac+ = X[1] * W[1]$
$mac+ = X[2] * W[2]$
$Y[0] = mac$
. . . . . .
$mac = B[3]$
$mac+ = X[0] * W[9]$
$mac+ = X[1] * W[10]$
$mac+ = X[2] * W[11]$
$Y[3] = mac$

Algorithm 4: Proposed 1D Depthwise Convolution Alogorithm in Encoder

Input: Input $X[L * C_{\text{in}}]$, weights $W[C_{\text{out}} * C_{\text{in}} * K]$, bias $B[C_{\text{out}}]$
Output: Output $y[L * C_{\text{out}} * C_{\text{in}}]$

Function `MAC_FUNCTION`(x, w, b) **:**
  return $b + x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2$
for $co = 1$ *to* C_{out} **do**
  `Y[0] = MAC(X[1:2], W[0:1], B[0])`
  for $lo < 1$ *to* $L/8$ **do**
    $Y =$ `MAC(X[0:2], W[0:2], B[0])`
    `Y[1]= 1`
    . . . . . .
    $Y =$ `MAC(X[6:8], W[0:2], B[0])`
    `Y[7] = 1`
    $Y =$ `MAC(X[7:9], W[0:2], B[0])`
    `Y[8] = 1`
    `X+=8 Y+=8`
  W += 3; B += 1;

IV. CONCLUSION

In this work, we presented a Pure-C implementation of a real-time speech enhancement model based on TRU-Net, an efficient deep neural network architecture specifically designed for embedded devices. Although modern High-Level Synthesis (HLS) are theoretically capable of applying loop unrolling and pipeline optimizations, they often struggle with deeply nested

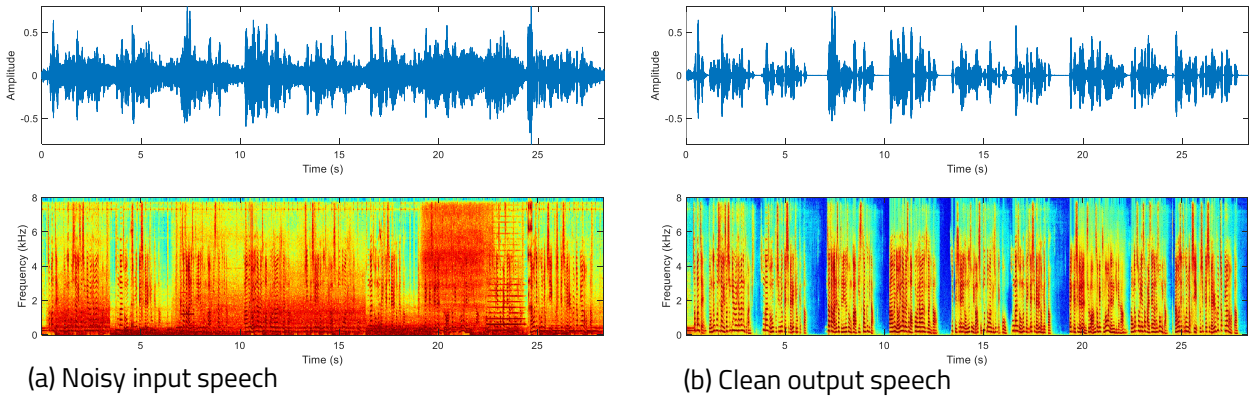

Fig. 2. Speech enhancement results : (a) noisy speech (b) clean speech

loop structures commonly found in convolution operations. In such cases, automatic optimization fails to fully exploit available parallelism and memory access patterns. Therefore, manual optimization strategies, including careful restructuring of loop ordering and memory layout, are essential. These approaches are particularly effective for edge devices, where computational resources, memory bandwidth, and power budgets are severely constrained. By tailoring the computation to the underlying hardware architecture, significant reductions in latency and energy consumption can be achieved, enabling efficient deployment of deep learning models in real-time, resource-limited environments.

As shown in Fig. 2, the proposed speech enhancement model effectively improves the perceptual quality and spectral clarity of noisy speech signals. The noisy input contains substantial background noise distributed over the entire time–frequency domain, which obscures speech components and degrades intelligibility. In contrast, the enhanced output demonstrates a clear reduction of background noise, particularly in the low- and mid-frequency regions, while preserving more distinct speech structures. In addition to improving speech quality, the proposed approach achieves a significant reduction in execution time. As shown in Fig. 3, the overall runtime is reduced by nearly 75% compared to a conventional CNN-based model. This result indicates that the proposed architecture not only maintains competitive speech enhancement performance but also provides substantial acceleration, making it more suitable for real-time and resource-constrained applications.

It should be noted that this study primarily focuses on optimizing the encoder layers to improve computational efficiency. In future work, the optimization will be extended to the entire network, including the decoder, with the goal of achieving a real-time factor (RTF) of 0.85 and conducting comprehensive quantitative comparisons with state-of-the-art speech enhancement models.

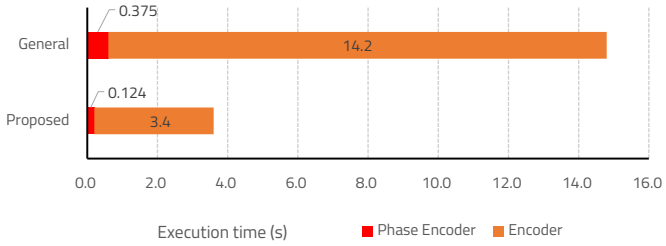

Fig. 3. The run time comparison between general and proposed CNN algorithm

REFERENCES

[1] L. Baresi, G. Quattrocchi, and N. Rasi, "Resource management for tensorflow inference," in *Service-Oriented Computing*, H. Hacid, O. Kao, M. Mecella, N. Moha, and H.-y. Paik, Eds. Cham: Springer International Publishing, 2021, pp. 238–253.

[2] C. Surianarayanan, J. J. Lawrence, P. R. Chelliah, E. Prakash, and C. Hewage, "A survey on optimization techniques for edge artificial intelligence (ai)," *Sensors*, vol. 23, no. 3, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/3/1279

[3] Y. Zhang, M. Pezeshki, P. Brakel, S. Zhang, C. Laurent, Y. Bengio, and A. C. Courville, "Towards end-to-end speech recognition with deep convolutional neural networks," *CoRR*, vol. abs/1701.02720, 2017. [Online]. Available: http://arxiv.org/abs/1701.02720

[4] G. Park, W. Cho, K.-S. Kim, and S. Lee, "Speech enhancement for hearing aids with deep learning on environmental noises," *Applied Sciences*, vol. 10, no. 17, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/17/6077

[5] J. Sharma, O.-C. Granmo, and M. Goodwin, "Emergency detection with environment sound using deep convolutional neural networks," in *Proceedings of Fifth International Congress on Information and Communication Technology*, X.-S. Yang, S. Sherratt, N. Dey, and A. Joshi, Eds. Singapore: Springer Singapore, 2021, pp. 144–154.

[6] Y. Hu, Y. Liu, S. Lv, M. Xing, S. Zhang, Y. Fu, J. Wu, B. Zhang, and L. Xie, "Dccrn: Deep complex convolution recurrent network for phase-aware speech enhancement," in *Proc. Interspeech*, 2020.

[7] D. S. Williamson, Y. Wang, and D. Wang, "Complex ratio masking for monaural speech separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 3, pp. 483–492, 2016.

[8] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation." Springer International Publishing, 2015, pp. 234–241.

[9] M. H. Claudia Barlli, Roger Mundri, "The relationship between downsampling and frequency resolution," 2013. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0082748