SHORT–TERM MEMORY CONVOLUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

The real-time processing of time series signals is a critical issue for many real-life applications. The idea of real-time processing is especially important for audio domain as the human audio perception is sensitive to any kind of disturbance in perceived signals, especially the lag between auditory and visual modalities. The rise of deep learning (DL) models complicated the landscape of signal processing. Although they often have superior quality compared to standard DSP methods, this advantage is diminished by higher latency. In this work we propose a method for minimization of latency and memory consumption, called Short-Term Memory Convolution (STMC) and its transposed counterpart. The main advantage of STMC is the low latency comparable to long short-term memory (LSTM) networks. Furthermore, the training of STMC-based models is faster and more stable as the method is based solely on convolutional neural networks (CNNs). In this study we demonstrate an application of this solution to a U-Net model for an audio separation task. We achieved a 5-fold reduction in inference time and a 2-fold reduction in latency without affecting the output quality.

1 INTRODUCTION

In recent years, convolutional neural networks (CNNs) arose to one of the dominant types of models in deep learning (DL). CNN models are effective in tasks where correlations between input elements are confined to areas (or volumes) smaller than the model's receptive field, defined as the size of the input region which is mapped into an output feature (Araujo et al., 2019). The most prominent example is the computer vision, e.g. image classification (Rawat & Wang, 2017), object detection (Zhao et al., 2019), or image segmentation (Minaee et al., 2021).

Moreover, CNN models proved to be effective in certain signal processing tasks, especially in cases where the long-term context is not required, such as speech enhancement (Sun et al., 2021), sound source separation (Stoller et al., 2018), or sound event detection (Lim et al., 2017). Despite the fact that all these tasks may also be performed by recurrent neural networks (RNN), some authors showed that convolutional models can achieve similar performance at a fraction of model parameters (Takahashi & Mitsufuji, 2017). It is also argued that CNNs are easier to parallelize than recurrent neural networks (RNNs) (Gui et al., 2019; Liu et al., 2022; Rybalkin et al., 2021; Kong et al., 2021) due to the time dependence of state computation in RNNs, although examples of recurrent models that address this issue exist, e.g. Simple Recurrent Unit (Lei et al., 2018).

However, the use of CNNs in online signal processing tasks may not be optimal. Unline RNNs, which can process incoming data one sample at the time, CNNs require a chunk of data to work correctly. The minimum chunk size is equal to the size of the receptive field which depends on the kernel sizes, strides, dilataion, and number of covonlutional layers. Additionally, overlaps may be required to reduce the undesired edge effects of padding. Hence, a standard CNN models are characterized by higher latency than RNNs. For example, according to our experience with speech enhancement models, a CNN model operating in the time-frequency domain and fed with a short-time fourier transform (STFT) as an input, require the time of 1–2 magnitudes higher to buffer enough fast fourier transform (FFT) frames than the model processing time. For example, Conv-TasNet (Luo & Mesgarani, 2019) which is based on a sequence of 1-D convolutional blocks, has a receptive field 0.46–3.83 seconds, depending on the configuration of the model.

If no zero-padding is used, the minimum size of a convolutional model input is equal to its receptive field. In practice however, the size of processed data is larger, because processing of an input with

size equal to the receptive field yields only one output element (e.g. a single FFT frame). The input size corresponds to the trade-off between the overall processing time and latency. Typically, it takes less time to process the whole data in a single chunk, than to split the data into smaller chunks and process them sequentially. On the other hand, larger inputs lead to increased latency, because the model waits longer to fill its buffer.

In many audio applications the buffer size is set to match the maximum acceptable latency. In the case of audio-visual signals, humans are able to spot the lag between audio and visual stimulus above 10 ms (Mcpherson et al., 2016). The maximum latency accepted in conversations is 40 ms (Staelens et al., 2012; Jaekl et al., 2015; Ipser et al., 2017). In comparison, the FFT frame and hop sizes in time-frequency domain models are often at least 20–40 ms and 10–20 ms, respectively, and several such frames are needed to run a single inference using a CNN.

1.1 Related works

Many researchers presented solutions addressing the problem of the latency minimization in signal processing models. Wilson et al. (2018) studied a model consisting of bidirectional LSTM (BLSTM), fully connected (FC), and convolutional layers. Firstly, they proposed to use unidirectional LSTM instead of BLSTM and found that it reduces latency by a factor of 2 while having little effect on the performance. Secondly, they proposed to alter the receptive field of each convolutional layer to be causal rather than centered on the currently processed data point. In addition, it was proposed to shift the input features with respect to the output, effectively providing a certain amount of future context, which the authors referred to as *look-ahead*. The authors showed that predicting future spectrogram masks comes at a significant cost, with a reduction of 6.6 dB signal to distortion ratio (SDR) with 100 ms shift between input/output, compared to a zero-look-ahead causal model. It was argued, that this effect occurs because the model is not able to respond immediately to changing noise and speech characteristics.

Romaniuk et al. (2020) further modified the above-mentioned model by removing LSTM and FC layers and replacing the convolutional layers with depth-wise convolutions followed by point-wise convolutions, among other changes in the model. These changes allowed to achieve a 10-fold reduction in the fused multiply-accumulate operations per second (FMS/s). However, the computational complexity reduction was accompanied by a 10% reduction in signal to noise ratio (SNR) performance. The authors also introduced partial caching of input STFT frames, which they referred to as *incremental inference*. When each new frame arrives, it is padded with the recently cached input frames to match the receptive field of the model. Subsequently, the model processes the composite input and yields a corresponding output frame.

Kondratyuk et al. (2021) introduced a new family of CNN for online classification of videos (MoViNets). The authors developed a more comprehensive approach to data caching, namely layer-wise caching instead of input-only caching. MoViNets process videos in small consecutive subclips, requiring constant memory. It is achieved through so-called *stream buffers*, which cache feature maps at subclip boundaries. Using the stream buffers allows to reduce the peak memory consumption up to an order of magnitude in large 3D CNNs. The less significant impact of 2-fold memory reduction was noted in smaller networks. Since the method is aimed at online inference, stream buffers are best used in conjunction with causal networks. The authors enforced causality by moving right (future) padding to the left side (past). It was reported, that stream buffers lead to approximately 1% reduction in model accuracy.

1.2 Novelty

In this work we propose a novel approach to data caching in convolutional layers called Short-Term Memory Convolution (STMC). The STMC layers are based on the following principles:

- Input data is processed in chunks of arbitrary size in an online mode.
- Each chunk is propagated through all convolutional layers, and the output of each layer is cached with shift registers (contrary to input-only caching as in Romaniuk et al. (2020)), providing so-called *past context*.
- The past context is never recalculated by any convolutional layer. When processing a time series all calculations are performed exactly once, regardless of the processed chunk size.

- The conversion of convolutional layers into STMC layers is performed after the model is trained, so the conversion does not affect the training time, nor does it change the model output.
- STMC layers are especially suitable for causal networks because of past context caching.

STMC is similar to the stream buffering technique (Kondratyuk et al., 2021), however it was developed and implemented independently, with a goal to reduce latency and peak memory consumption in CNNs for audio processing. STMC approaches the caching issue differently than MoViNets. It uses shift registers, which cache feature maps of all layers (instead of caching model inputs only) and shift them in the time axis according to the input size. The shift registers enable the processing of data chunks of arbitrary size, where the chunk size may vary in time. Furthermore, shift registers can buffer multiple states of the network, which allows for handling strides and pooling layers. Finally, we propose a transposed STMC, making it suitable for a broader family of networks, including U-nets.

2 METHODS

This section presents the Short-Term Memory Convolutions and their transposed counterpart. The description of STMC is followed by our suggested approach to enforce causality in the model in an efficient manner.

2.1 SHORT-TERM MEMORY CONVOLUTIONS

The STMC and transposed STMC(tSTMC) layers introduce an additional cache memory to store the past outcomes (Fig. 1), in contrast to their standard counterparts. The cache memory provides values calculated in the past to subsequent calls of the STMC layers in order to calculate the final output without any redundant computations. The goal is to process all the data only once.

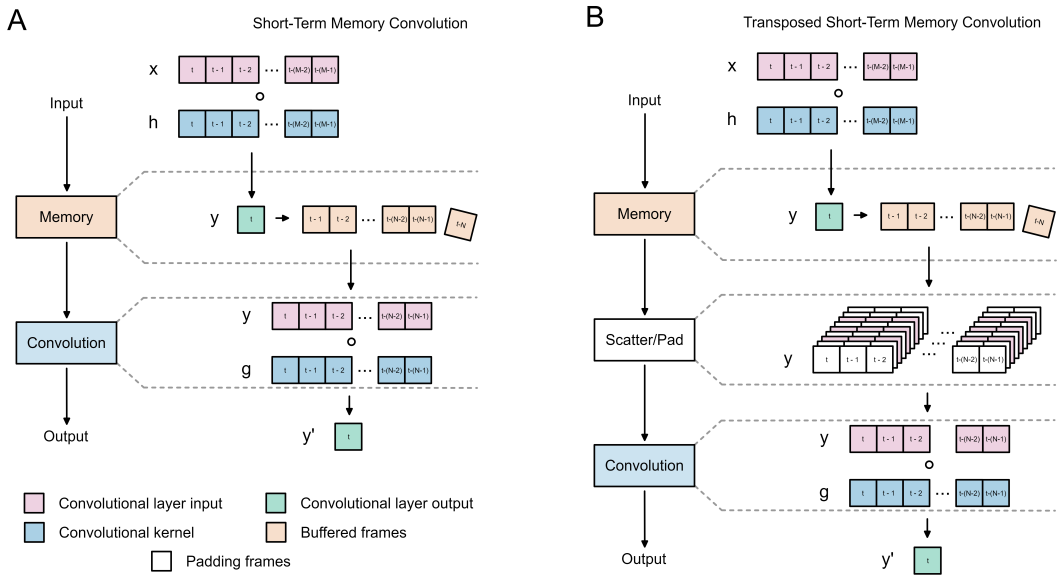

Figure 1: Proposed operations. A) Short-Term Memory Convolution operation. B) Transposed Short-Term Memory Convolution operation.

The design of the STMC shift registers is based on the observation that CNN layers of an online model process input data partially overlapping with data from the previous step. The only difference is that the new data is slightly shifted (Fig. 2A) and only a small portion of the input is actually new. In order to fully reconstruct the output without any redundant computations, the convolution results (feature maps) must be saved at each level of the network (Fig. 2B). The number of saved frames in a particular STMC block is equal to the size of its receptive field. The input of the next

convolutional layer is then padded with a previously saved state. This method retains all the past context from the network in an efficient way. The efficiency stems from the fact that all cached data is already pre-processed by the respective layer.

For the purpose of demonstration let us assume 1D time series data $X \in \mathbb{R}^T$, composed of T samples. Let us also assume X is processed by a convolutional layer with kernel $h \in \mathbb{R}^M$ which can be also represented by a Toeplitz matrix H. We can calculate the output signal $Y_T = (y_{T-M+1}, y_{T-M+2}, ..., y_T)$ as follows:

$$Y_T = \sigma(H \cdot X)$$

where σ is an activation function.

Executing the above operation for each subsequent time step t would have a high computational demand. One of the solutions is the incremental inference (Romaniuk et al., 2020), in which the input data is cropped to match the size of the receptive field, which in this case is $M - 1$.

STMC is a further improvement, because it caches not only the inputs, but all intermediate outputs of all convolutional layers. These outputs are used in a subsequent call of the model by concatenating them with the newest data. This technique ensures that all convolutions process only the newest data with kernels placed at the most recent position. In other words, the convolutional kernels do not need to be slid along the time axis, because past data was already cached, and there is no need to recalculate it again. Only the newest frames $y_T = \sigma(h \cdot X_T)$ need to be calculated:

$$Y_T = (Y_{T-1} \mid \sigma(h \cdot X_T))$$

where $X_T \in \mathbb{R}^M$ are the newest input frames and the $\mid$ represent the operation done by a shift register, e.g.:

$$Z = (z_1, z_2, ..., z_n)$$

$$(Z|\alpha) = (z_2, z_3, ..., z_n, \alpha)$$

The above method holds true for every subsequent convolution, which leads to the conclusion that the STMC effectiveness increases with the depth of the network. The buffer Y_{T-1} can be implemented as a simple shift register of size $M - 1$ (Fig. 1). The method can get significantly more complicated depending on the architecture of NN (see section 2.2 Limitations). For example, the use of strided convolutions requires to buffer multiple states (using multiple shift registers). However, in this particular case, the more complicated implementation of STMC will greatly improve the model's processing speed by mitigating stride-induced increased receptive field as the strided

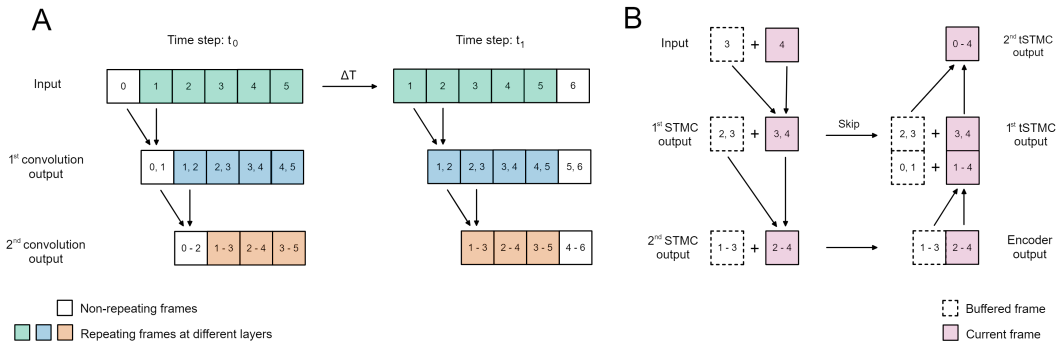

Figure 2: Effect of Short-Term Memory Convolution on the network. A) Convolution redundancy effect in time domain. B) U-Net network with STMC/tSTMC.

convolutions inflate the receptive field by the factor of *stride*. In comparison, the incremental inference would be significantly affected by strided convolutions, because a larger receptive field means increased latency.

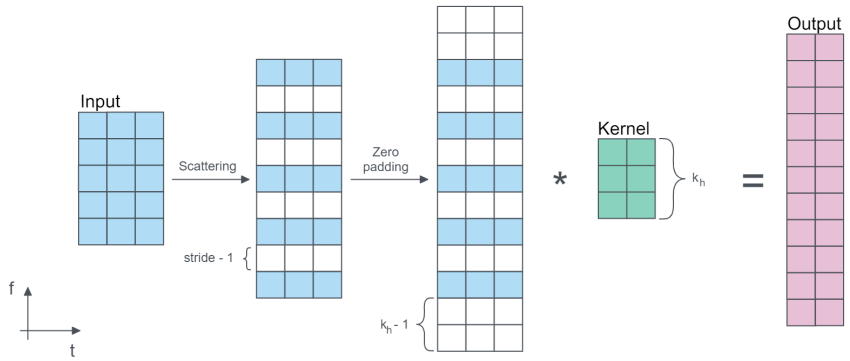

Figure 3: Transposed Short-Term Memory Convolution operation.

STMC can be also applied to transposed convolutions. One approach is to use STMC with standard convolutions with zero-padding to facilitate upsampling. However, a more optimized approach exists, which we will refer to as *transposed STMC* (tSTMC). In principle, at most two additional operations are needed to implement tSTMC, as presented in Fig. 3. The first one is scattering, which inflates the input by introducing zero-valued regions inside the input. Each region size depends on the stride of previous transposed convolution. The second is padding, which depends on the kernel size. The input prepared in this way and convolved with the kernel produces the desired output shape for transposed convolutions.

2.2 Limitations

The first limitation of STMC is that can be applied only to convolutional and pooling layers. In contrast to RNNs and LSTMs which take only the most recent data as inputs, convolutional models requires longer data segments as inputs. STMC simply caches the overlapping parts of the input segments together with all intermediate results within the network between subsequent calls to the model in the online inference.

The second limitation is that the exact implementation of STMC depends on the model's architecture. This work presents how to implement STMC to U-nets with convolutions, transposed convolutions, and skip connections. While it is possible to implement STMC to layers with strides and dilations, it would considerably increase the complexity of a network. E.g. strided convolutions require multiple shift registers per layer.

Finally, STMC should be used with causal convolutions, which typically have a lower than standard convolutions due to the lack of future context. In addition, causal convolutions are effectively "anti-causal" in the transposed convolutions, because the deconvolution output depends solely on the current and future input values (contrary to a causal case where output depends only on the past and the current input). This effect can be countered by shifting the model inputs with respect to model outputs, which works for stacked convolutional layers, but some additional modifications may be required in the case of more sophisticated architectures.

The causality requirement is explained in detail in Appendix A.

3 Experiment

In this section we describe the experiment, which evaluates the proposed method from the point of view of inference time and memory consumption. We chose an audio separation task solved by a U-net network as the case study. The task is one of the examples in which the online inference is often required and the latency must be kept low. In addition, the U-net consists of both convolutions and transposed convolutions, so it is more challenging to use due to the "anti-causality" of the decoder, as

described in Section A. In total, seven U-nets are compared: U-Net, U-Net$_{\text{INC}}$, U-Net$_{\text{STMC}}$, Causal U-Net, Causal U-Net$_{\text{INC}}$, Causal U-Net$_{\text{STMC}}$, and Causal U-Net$_{\text{STMC+tSTMC}}$. The models differ in causality and in STMC applied to the encoder and decoder. In addition, both U-Net and Causal U-Net process the whole data in a single inference, so they are effectively off-line models. The rest of the models work in an online mode, processing data in chunks.

3.1 Models and training process

U-Net As our benchmark model, we incorporated a network based on the U-Net architecture (Ronneberger et al., 2015). The original architecture was simplified by removing max pooling operations, so the encoding is done solely by convolutions. We adapted the model for the audio separation task and achieved satisfying results with a model of depth 7. Each convolutional layer comprises a number of kernels of length 3 in the time domain, which leads to the receptive field of 29 for an assembled network. The kernel size in the frequency domain is set to 5. As the benchmark network consists of convolutional layers which are locally centered (Favero et al., 2021) on a current time frame, the receptive field of 14 for both past and the future context can be derived. The STFT input with real and imaginary parts as separate channels is utilized, resulting in a complex mask for the input signal as the output of the network. A STFT window of size 1024 and a hop of 256 are used.

Causal U-Net To show the influence of causality we also trained Causal U-Net models. In all the considered causal models we used kernels of size 2 in the time domain to reduce the computation cost and the receptive field, as well as to compress the model size. The preliminary tests showed that the output quality of the casual model is not affected by the change. Such an approach allows to preserve only vital calculations and removes additional padding operations. The receptive field of the network is effectively reduced by half. To achieve causality, the output of the model was shifted by 7 STFT frames. To align the data inside of the network, all skip connections also needed to be shifted by the amount dependant on their depth inside the model.

Incremental inference We applied the incremental inference method (Romaniuk et al., 2020) to U-Net and Causal U-Net by cropping the original input size to the size of receptive field, yielding 29 STFT frame input for U-Net and 15 STFT frames for the Causal U-Net. We will refer to this network as U-Net$_{\text{INC}}$ and Causal U-Net$_{\text{INC}}$.

Short-Term Memory Convolutions Similarly to the incremental inference, STMC was applied to both U-Net and Causal U-Net networks. Firstly, we applied STMC inside the encoder and inside the decoder with zero-padding and output cropping (i.e. without tSTMC). We will denote these networks as U-Net$_{\text{STMC}}$ and Causal U-Net$_{\text{STMC}}$. To show the difference in inference time of STMC with cropping and with tSTMC, we also separately trained causal model consisting of both STMC and tSTMC. This network is denoted by Causal U-Net$_{\text{STMC+tSTMC}}$. The input size in the temporal axis for this method was set to 3 and 2 for U-Net and Causal U-Net, respectively.

Dataset and training conditions The Deep Noise Suppression (DNS) Challenge - Interspeech 2020 dataset (Reddy et al., 2020), licensed under CC-BY 4.0, was adopted for both training and validation of networks. During training the models have seen around 100M randomly sliced 3-second audio samples. All models (U-Net, Causal U-Net and Causal U-Net$_{\text{STMC+tSTMC}}$) were trained on Nvidia A100 with batch size of 128 and Adam optimizer with learning rate of 1e-3. Each model was trained for 100 epochs which took about 90 hours per model.

All models were converted into TFLite representation with default converter settings.

3.2 Evaluation on desktop

The TFLite Model Benchmark Tool with C++ binary was adopted as the evaluation tool for inference time and peak memory consumption. Each model was tested in 1,000 independent runs on 8 cores of the Intel(R) Xeon(R) Gold 6246R CPU with 3.40 GHz clock, for which the average inference times and average peak memory consumption were calculated. Additionally, the average SI-SNRi metric was calculated to ensure the proposed optimization methods do not reduce the output's quality.

The latency L was calculated based on the following equation:

$$L = i + l_c + l_f + t_{STFT} + t_{iSTFT}$$

where i is the inference time, l_c is the length of the current audio chunk, l_f is the length of the future audio chunk needed to fill the receptive field of the network, t_{STFT} and t_{iSTFT} is the time spent on calculating short-time Fourier transform and its inverse, respectively. The length of future audio l_f is zero in the case of causal models.

3.3 Evaluation on mobile device

To confirm the portability of the proposed method to other devices, models were also evaluated on a mobile device using the same setup as on the desktop. We used Samsung Fold 3 SM-F925U equipped with Qualcomm Snapdragon 888. The models were analyzed using the TFLite Model Benchmark Tool, but this time 4 cores were used.

4 Results

4.1 Results on desktop

The results for all models are presented side-by-side in Table 1. Comparing to standard convolutions, the incremental inference (INC) method achieves an 8.00-fold and a 29.74-fold latency reduction in the case of non-causal and causal setting, respectively. Our method (STMC) outperforms INC with a further 1.32-fold latency reduction for non-casual and a 2.52-fold reduction for causal variants. STMC achieved also approximately 5.68-fold reduction in inference time, compared to INC. The impact of STMC on the non-causal U-Net is lower as most of the computational cost of U-Net$_{\text{STMC}}$ is due to the need for waiting for the future context (convolutional kernels centered with respect to the output). The implementation of transposed STMC in the decoder of the Causal U-Net model gives only a slight improvement in latency (0.57 ms), but it reduces the peak memory consumption by approximately 20%. The model size is similar for standard, INC, and STMC variants. The causal models are approximately 33% smaller than the non-causal ones, which is caused by the smaller kernels as described in Section 3.1.

Table 1: Results of evaluation on desktop using TFLite Model Benchmark Tool. *Difference in metrics due to separate learning of models.

Name	SI-SNRi (dB)	Inference time (ms)	Latency (ms)	Peak memory (MB)	Size (KB)
U-Net	**8.00**	1 381.02 ± 17.62	4 403.22	170.36	14 025
U-Net$_{\text{INC}}$	**8.00**	151.92 ± 12.25	550.54	48.97	14 025
U-Net$_{\text{STMC}}$	**8.00**	86.33 ± 4.33	416.03	51.31	14 035
Causal U-Net	7.58	962.61 ± 25.91	3 984.81	138.18	**9 366**
Causal U-Net$_{\text{INC}}$	7.58	55.64 ± 2.47	133.98	31.57	9 367
Causal U-Net$_{\text{STMC}}$	7.58	10.37 ± 0.69	52.07	27.50	9 376
Causal U-Net$_{\text{STMC+tSTMC}}$	7.44*	**9.80 ± 0.97**	**51.50**	**22.20**	9 373

The collected data is also visualised in Fig. 4. The inference time distributions have many outliers (Fig. 4C), in some cases reaching up to 14-15 ms. However, even these extreme values are several times lower than the average inference times of INC models, and almost 2 orders of magnitude lower compared to the baseline models. As convolution is the most resource intensive task in the analyzed models, we also investigated the inference time of each convolutional layer for Causal U-Net$_{\text{STMC}}$ and Causal U-Net$_{\text{STMC+tSTMC}}$ (Fig. 4D and 4E).

4.2 Results on mobile device

The results of mobile evaluation are collected in Table 2. They further confirm the conclusions drawn from desktop results. Since the models used in this test are the same as in the case of desktop evaluation, we did not include the model size and SI-SNRi metrics in the table.

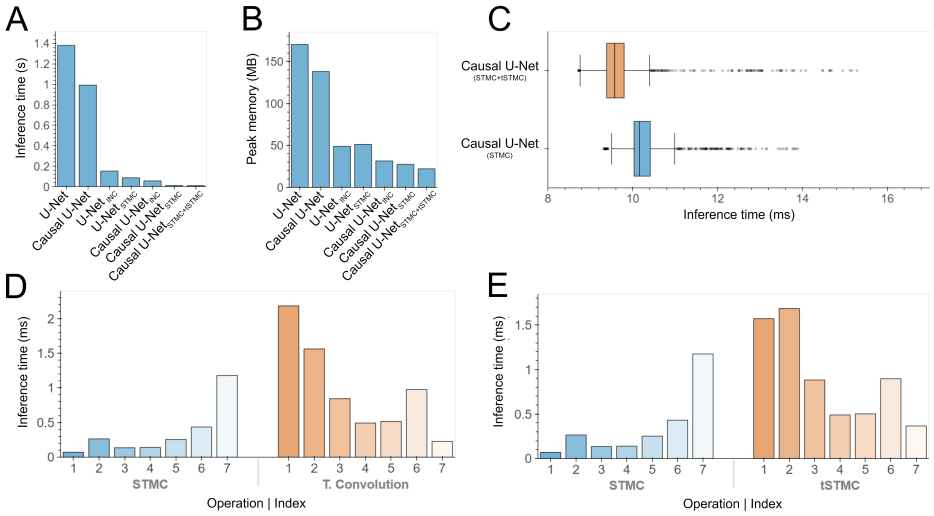

Figure 4: Visualisation of evaluation results. A) Inference time of each model. B) Peak memory usage of each model. C) Box plot of inference time of causal STMC model and causal tSTMC model. D) Inference time of each convolution for STMC model. E) Inference time of each convolution for tSTMC model.

Table 2: Results of evaluation on mobile using TFLite Model Benchmark Tool.

Name	Inference time (ms)	Latency (ms)	Peak memory (MB)
U-Net	262.44 ± 26.81	3 284.64	164.93
U-Net$_{\text{INC}}$	38.51 ± 4.34	323.72	44.14
U-Net$_{\text{STMC}}$	19.26 ± 1.02	288.96	46.59
Causal U-Net	240.95 ± 30.4	3 263.15	133.60
Causal U-Net$_{\text{INC}}$	14.02 ± 0.71	50.74	26.53
Causal U-Net$_{\text{STMC}}$	3.15 ± 0.10	28.85	21.54
Causal U-Net$_{\text{STMC+tSTMC}}$	**2.77** ± **0.26**	**28.47**	**16.11**

The inference time and latency achieved on the mobile device are significantly lower than on the desktop, despite the same evaluation method used on both platforms. The inference time reduction factor between Causal U-Net$_{\text{INC}}$ and Causal U-Net$_{\text{STMC+tSTMC}}$ is 5.06. The usage of tSTMC on this device has a relatively higher impact than on the desktop, as in this case it reduces the inference time by 12.06% (compared to 5.50%). Similarly to the desktop, the lowest peak memory consumption was achieved by Causal U-Net$_{\text{STMC+tSTMC}}$.

5 Conclusion

In this paper, we presented the method for inference time, latency, and memory consumption reduction. The method relies on buffering of the convolutional layers' outputs for each iteration. The overall goal of the proposed method is to process all data only once and re-use the past results of all convolutional layers. The method was tested on an audio separation task on two devices: desktop computer and mobile phone. In total, 7 variants of a U-Net architecture were compared, differing in causality and buffering types. On both test platforms, approximately a 5-fold reduction of inference time was achieved compared to the incremental inference, which is considered the state-of-the-art method for online processing using convolutional models. The STMC method has also 44-61% lower latency and 30-39% lower peak memory consumption, depending on the test platform. All the causal models used in the test achieved approximately 6% reduction in SI-SNRi compared to their non-causal counterparts. The cause of the score reduction is attributed to the lack of future context and not the used caching method.

The STMC method proved to significantly reduce the number of operations inside CNNs, which may be crucial for many DL applications, e.g. deploying models on embedded devices. The method can be applied to most of the existing CNN architectures and can be combined with several different methods used for time series processing, e.g. dilated causal convolutions (van den Oord et al., 2016) or temporal convolutions (Lea et al., 2017). Using STMC and their transposed counterpart – tSTMC – does not require any retraining of the model. It is, however, strongly advised to use causal models in order to reach near real-time processing.

The main limitation of STMC is that it requires initialization. Before the first inference, the shift registers are filled with zeros. The network output will be degraded until all shift registers are filled with past data, which takes no more time than the receptive field length itself. For example, the U-Net model from the presented case study is fully initialized after 0.512 seconds.

The described approach is also relevant for applications beyond the audio separation task presented in this work. Since the STMC method leads to the reduction of inference time, latency, and peak memory consumption, it can be used for systems where the size of data is too large for the available memory or computational power, e.g.: medical imaging (Blumberg et al., 2018), radio astronomy imaging (Corda et al., 2020), FPGA (Ma et al., 2018), video analysis (Lea et al., 2017), checksum computation (Filippas et al., 2022). We also believe that the described method may be useful for low latency systems on chip (SoC) for online processing applications (Park et al., 2022).

REFERENCES

André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. doi: 10.23915/distill.00021.

Stefano B. Blumberg, Ryutaro Tanno, Iasonas Kokkinos, and Daniel C. Alexander. Deeper image quality transfer: Training low-memory neural networks for 3d images. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pp. 118–125. Springer International Publishing, 2018. ISBN 978-3-030-00928-1.

Stefano Corda, Bram Veenboer, Ahsan Javed Awan, Akash Kumar, Roel Jordans, and Henk Corporaal. Near memory acceleration on high resolution radio astronomy imaging. In *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–6, 2020. doi: 10.1109/MECO49872.2020.9134089.

Alessandro Favero, Francesco Cagnetta, and Matthieu Wyart. Locality defeats the curse of dimensionality in convolutional teacher-student scenarios. In *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=sBBnfOFtPc`.

Dionysios Filippas, Nikolaos Margomenos, Nikolaos Mitianoudis, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos. Low-cost online convolution checksum checker. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(2):201–212, 2022. doi: 10.1109/TVLSI.2021.3119511.

Tao Gui, Ruotian Ma, Qi Zhang, Lujun Zhao, Yu-Gang Jiang, and Xuanjing Huang. Cnn-based chinese ner with lexicon rethinking. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 4982–4988. International Joint Conferences on Artificial Intelligence Organization, July 2019. doi: 10.24963/ijcai.2019/692.

Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Alberta Ipser, Vlera Agolli, Anisa Bajraktari, Fatimah Al-Alawi, Nurfitriani Djaafara, and Elliot D. Freeman. Sight and sound persistently out of synch: stable individual differences in audiovisual synchronisation revealed by implicit measures of lip-voice integration. *Scientific Reports*, 7(1): 46413, April 2017. ISSN 2045-2322. doi: 10.1038/srep46413.

Philip Jaekl, Jakob Seidlitz, Laurence R. Harris, and Duje Tadin. Audiovisual delay as a novel cue to visual distance. *PLOS ONE*, 10(10):1–12, October 2015. doi: 10.1371/journal.pone.0141125.

Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. Movinets: Mobile video networks for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16020–16030, June 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Kondratyuk_MoViNets_Mobile_Video_Networks_for_Efficient_Video_Recognition_CVPR_2021_paper.html.

Jun Kong, Leixin Zhang, Min Jiang, and Tianshan Liu. Incorporating multi-level cnn and attention mechanism for chinese clinical named entity recognition. *Journal of Biomedical Informatics*, 116:103737, 2021. ISSN 1532-0464. doi: 10.1016/j.jbi.2021.103737.

Colin Lea, Michael D. Flynn, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks for action segmentation and detection. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pp. 1003–1012. Institute of Electrical and Electronics Engineers Inc., November 2017. doi: 10.1109/CVPR.2017.113.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4470–4481. Association for Computational Linguistics, October–November 2018. doi: 10.18653/v1/D18-1477.

Hyungui Lim, Jeongsoo Park, and Yoonchang Han. Rare sound event detection using 1D convolutional recurrent neural networks. Technical report, DCASE 2017 Challenge, September 2017.

Yingying Liu, Peipei Li, and Xuegang Hu. Combining context-relevant features with multi-stage attention network for short text classification. *Computer Speech & Language*, 71:101268, 2022. ISSN 0885-2308. doi: 10.1016/j.csl.2021.101268.

Yi Luo and Nima Mesgarani. Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 27(8):1256–1266, August 2019. ISSN 2329-9290. doi: 10.1109/TASLP.2019.2915167.

Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. Optimizing the convolution operation to accelerate deep neural networks on fpga. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1354–1367, 2018. doi: 10.1109/TVLSI.2018.2815603.

Andrew Mcpherson, Robert Jack, and Giulio Moro. Action-sound latency: Are our tools fast enough? In *The International Conference on New Interfaces for Musical Expression*, April 2016.

Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vh-0sUt8HlG.

Shervin Minaee, Yuri Y. Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. doi: 10.1109/TPAMI.2021.3059968.

Jun-Seok Park, Changsoo Park, Suknam Kwon, Hyeong-Seok Kim, Taeho Jeon, Yesung Kang, Heonsoo Lee, Dongwoo Lee, James Kim, YoungJong Lee, Sangkyu Park, Jun-Woo Jang, SangHyuck Ha, MinSeong Kim, Jihoon Bang, Suk Hwan Lim, and Inyup Kang. A multi-mode 8k-mac hw-utilization-aware neural processing unit with a unified multi-precision datapath in 4nm flagship mobile soc. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pp. 246–248, 2022. doi: 10.1109/ISSCC42614.2022.9731639.

Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9):2352–2449, 2017. doi: 10.1162/neco_a_00990.

Chandan KA Reddy, Vishak Gopal, Ross Cutler, Ebrahim Beyrami, Roger Cheng, Harishchandra Dubey, Sergiy Matusevych, Robert Aichner, Ashkan Aazami, Sebastian Braun, et al. The interspeech 2020 deep noise suppression challenge: Datasets, subjective testing framework, and challenge results. In *Proc. Interspeech 2020*, 2020.

Michal Romaniuk, Piotr Masztalski, Karol Piaskowski, and Mateusz Matuszewski. Efficient Low-Latency Speech Enhancement with Mobile Audio Streaming Networks. In *Proc. Interspeech 2020*, pp. 3296–3300, 2020. doi: 10.21437/Interspeech.2020-2443.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

Vladimir Rybalkin, Jonas Ney, Menbere Kina Tekleyohannes, and Norbert Wehn. When massive gpu parallelism ain't enough: A novel hardware architecture of 2d-lstm neural network. *ACM Trans. Reconfigurable Technol. Syst.*, 15(1), November 2021. ISSN 1936-7406. doi: 10.1145/3469661.

Nicolas Staelens, Jonas De Meulenaere, Lizzy Bleumers, Glenn Van Wallendael, Jan De Cock, Koen Geeraert, Nick Vercammen, Wendy Van den Broeck, Brecht Vermeulen, Rik Van de Walle, and Piet Demeester. Assessing the importance of audio/video synchronization for simultaneous translation of video sequences. *Multimedia Systems*, 18(6):445–457, November 2012. ISSN 1432-1882. doi: 10.1007/s00530-012-0262-4.

Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. In *The 19th International Society for Music Information Retrieval Conference*, 2018. URL `http://ismir2018.ircam.fr/doc/pdfs/205_Paper.pdf`.

Yuhang Sun, Linju Yang, Huifeng Zhu, and Jie Hao. Funnel Deep Complex U-Net for Phase-Aware Speech Enhancement. In *Proc. Interspeech 2021*, pp. 161–165, 2021. doi: 10.21437/Interspeech.2021-10.

Naoya Takahashi and Yuki Mitsufuji. Multi-scale multi-band densenets for audio source separation. In *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 21–25, 2017. doi: 10.1109/WASPAA.2017.8169987.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. In *Proc. 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, pp. 125, 2016. URL `https://arxiv.org/pdf/1609.03499.pdf`.

Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1747–1756. JMLR.org, 2016. URL `http://proceedings.mlr.press/v48/oord16.pdf`.

Kevin Wilson, Michael Chinen, Jeremy Thorpe, Brian Patton, John Hershey, Rif A. Saurous, Jan Skoglund, and Richard F. Lyon. Exploring tradeoffs in models for low-latency speech enhancement. In *Proceedings of the 16th International Workshop on Acoustic Signal Enhancement*, 2018.

Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019. doi: 10.1109/TNNLS.2018.2876865.

A Efficient causality

Non-causal convolutions can be used in online inference by either waiting enough time for their receptive field to be filled with "future" data, which increases latency, or by replacing the future data with zeros or some other values, e.g. symmetrically reflecting the data. The latter approach, however, degrades the output quality at the most recent frames, which coincidentally are the most important for the online inference. Therefore, causality is typically required for robust online inference using convolutional networks. Causality is also desired for STMC, because STMC layers cache the past context as described in Section 2.1.

Technically, the convolution operation does not take any assumptions on causality. In many applications, especially in computer vision, the kernels are centered with respect to the current output

element of interest. For example, let C be a convolutional layer with 3×1 dimensional kernel and valid padding. Let A and B be sample input and output tensors with dimensions of 5×1 and 3×1 respectively. We then have:

$$CA^T = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} b_2 \\ b_3 \\ b_4 \end{bmatrix} = B^T$$

When the convolution is used this way, both the past and future context is needed. One of the methods for enforcing causality is by using masked convolutions, in which the kernels are multiplied by non-symmetric masks (Van Den Oord et al., 2016) before being convolved with the input. However, the masked convolutions are more resource consuming than standard convolutions due to additional multiplications. Therefore, in signal processing it is more desired to enforce causality by shifting the input with respect to the output (van den Oord et al., 2016). In other words, the elements of tensor B can be re-indexed to reflect temporal dependency, in a way that b_i does not depend on any a_j for $[j > i)$:

$$CA^T = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} b_3 \\ b_4 \\ b_5 \end{bmatrix} = B^T$$

$$b_i = k_1 a_{i-2} + k_2 a_{i-1} + k_3 a_i$$

An interesting phenomenon occurs when such a "causal" convolution is transposed, because then get:

$$C'B^T = \begin{bmatrix} k'_1 & 0 & 0 \\ k'_2 & k'_1 & 0 \\ k'_3 & k'_2 & k'_1 \\ 0 & k'_3 & k'_2 \\ 0 & 0 & k'_3 \end{bmatrix} \begin{bmatrix} b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \\ a'_4 \\ a'_5 \end{bmatrix} = A'^T$$

$$a'_i = k'_3 b_i + k'_2 b_{i+1} + k'_1 b_{i+2}$$

which is effectively "anti-causal", i.e. the deconvolution output depends solely on current and future input values (contrary to a causal case where output depends only on past and current input). This effect can be countered by shifting model inputs with respect to model outputs. This approach works for stacked convolutional layers, but in the case of more sophisticated architectures some modifications may be required. For example, U-Net models have skip connections between convolutions and transposed convolutions. The data passed through each skip connection must also be properly shifted with respect to the outputs of transposed convolutions. The shift value depends on the network architecture and may be different at each level.

B ADDITIONAL EVALUATION

To show that our method has also an impact in small CNN models we applied to a GhostNet (Han et al., 2020) architecture trained for an audio classification task. We tested the inference time and peak memory consumption on CPU (Intel(R) Xeon(R) Gold 6246R CPU with 3.40 GHz clock), GPU (Nvidia A100), and on CPU with XNNPACK optimization. For comparison, we also tested LSTM and MobileViT (Mehta & Rastegari, 2022) networks. The networks used in this test were not quantized.

GhostNet Our classification network consist of 3 ghost bottleneck blocks and ends with a convolutional layer and average pooling. As the input it takes a mel spectrogram of size 94 in the time axis and 256 in the frequency axis.

LSTM We tested the RNN architecture with 2 LSTM blocks with 64 and 32 units and 2 fully connected layers with 16 and 1 units. As the input it takes 1 time bin of mel spectrogram with 256 frequency bins.

MobileViT We also put to the test the MobileViT network. In contrast to original implementation described in (Mehta & Rastegari, 2022) we changed the number of input channels to 1 and the number of output classes to 1.

Table 3: Results of additional evaluation.

Network	Inference time (ms)	Peak memory (MB)	Model size (MB)	Number of parameters	Input size
GhostNet	4.706 ± 6.167	9.76	0.040	8 296	94×256×1
GPU GhostNet	0.912 ± 0.504	—			
XNNPACK GhostNet	1.086 ± 5.612	11.49			
STMC GhostNet	0.973 ± 0.866	7.59	0.044	8 300	1×256×1
GPU STMC GhostNet	0.602 ± 0.321	—			
XNNPACK STMC GhostNet	0.484 ± 2.447	8.10			
LSTM	0.095 ± 0.010	7.50	0.385	95 137	1×256
GPU LSTM	0.099 ± 0.074	—			
XNNPACK LSTM	0.119 ± 0.348	7.70			
MobileViT XXS	52.948 ± 11.456	38.11	5.407	1 306 049	256×256×1
GPU MobileViT XXS	419.203 ± 90.989	—			