applied sciences

MDPI

Article

Online Speech Recognition Using Multichannel Parallel Acoustic Score Computation and Deep Neural Network (DNN)- Based Voice-Activity Detector

Yoo Rhee Oh *,† **, Kiyoung Park** † **and Jeon Gyu Park**

Artificial Intelligence Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon 34129, Korea; pkyoung@etri.re.kr (K.P.); jgp@etri.re.kr (J.G.P.)

* Correspondence: yroh@etri.re.kr

† These authors contributed equally to this work.

Received: 15 May 2020; Accepted: 9 June 2020; Published: 14 June 2020

Abstract: This paper aims to design an online, low-latency, and high-performance speech recognition system using a bidirectional long short-term memory (BLSTM) acoustic model. To achieve this, we adopt a server-client model and a context-sensitive-chunk-based approach. The speech recognition server manages a main thread and a decoder thread for each client and one worker thread. The main thread communicates with the connected client, extracts speech features, and buffers the features. The decoder thread performs speech recognition, including the proposed multichannel parallel acoustic score computation of a BLSTM acoustic model, the proposed deep neural network-based voice activity detector, and Viterbi decoding. The proposed acoustic score computation method estimates the acoustic scores of a context-sensitive-chunk BLSTM acoustic model for the batched speech features from concurrent clients, using the worker thread. The proposed deep neural network-based voice activity detector detects short pauses in the utterance to reduce response latency, while the user utters long sentences. From the experiments of Korean speech recognition, the number of concurrent clients is increased from 22 to 44 using the proposed acoustic score computation. When combined with the frame skipping method, the number is further increased up to 59 clients with a small accuracy degradation. Moreover, the average user-perceived latency is reduced from 11.71 s to 3.09–5.41 s by using the proposed deep neural network-based voice activity detector.

Keywords: concurrent clients; low-latency; parallel decoding; voice-activity detector (VAD); automatic speech recognition (ASR); bidirectional long short-term memory (BLSTM)

1. Introduction

1.1. Problem Definition

Deep learning with GPU and considerable speech data has greatly accelerated the advance of speech recognition [1–5]. In line with this advancement, automatic speech recognition (ASR) systems have been widely deployed in various applications such as dictation, voice search, and video captioning [6]. The research on an ASR deployment can be classified into (a) an on-device system and (b) a server-based system. For an on-device deployment, various optimizations have been proposed, such as efficient and light architectures of an acoustic model (AM) or a language model (LM), network pruning methods, parameter quantization, speaker-dependent models, and compiler optimizations [7–12]. Therefore, there exists a trade-off between ASR accuracy and real-time performance. For a server-based deployment where an ASR system is based on a server-client model, the server performs speech recognition using high-performance resources, whereas the client can be applied in various devices, including embedded devices. Therefore, the trade-off between accuracy and real-time performance

can be relaxed when compared to on-device deployment. However, issues on response latency or multiple concurrent clients can arise when utilizing state-of-the-art ASR systems [13–22].

In order to employ one of the state-of-the-art AMs, a bidirectional long short-term memory (BLSTM), this paper focuses on the deployment of an online server-based ASR, where the overall structure is shown in Figure 1. Whenever an audio segment is captured from a client device, the client sends the audio segment to an ASR server (which is connected over a computer network), and then receives the decoded text from the server. On the ASR server, there exist a main thread and a decoder thread for each connected client. The main thread communicates with the connected client, manages the decoder thread, extracts the speech feature vectors from the received audio segments, and buffers the feature vectors into a ring buffer. The decoder thread performs speech recognition using the extracted feature vectors. More detailed description is explained in Section 2.1.

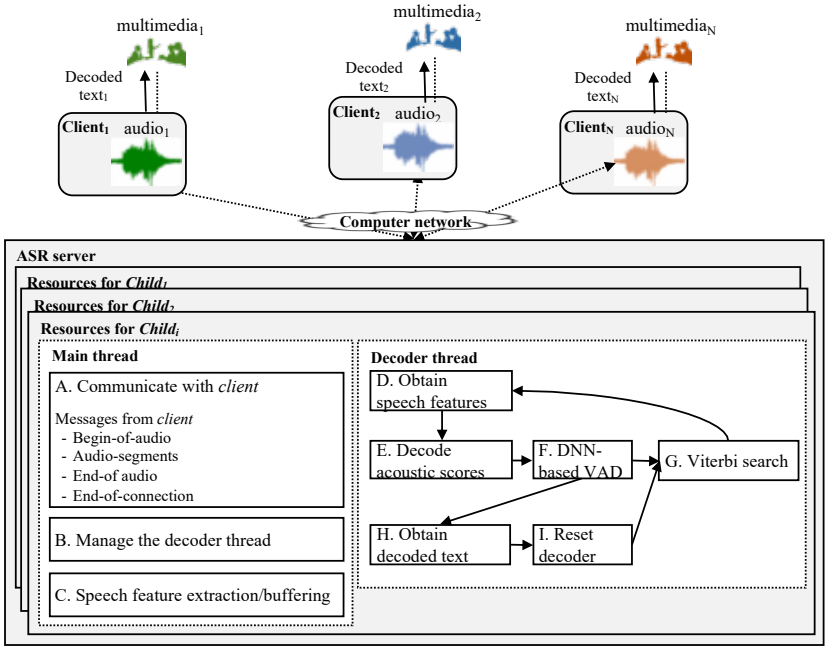


Figure 1. Overall structure of the server-client-based online automatic speech recognition (ASR) system used in this paper.

In short, our goal is to deploy an online ASR system using a BLSTM AM where the AM is usually used in an offline ASR system since the vanilla AM requires a whole input speech during decoding. For a real-time online ASR system with more concurrent clients, we aim to solve the two challenges (a) a long decoding time and (b) long response latency, while the accuracy performance is maintained.

1.2. Literature Review

The BLSTM AM of an ASR system achieves better performance than a deep neural network (DNN)- or long short-term memory (LSTM)-based AM. However, the use of BLSTM AM is limited due to practical issues such as massive computational complexity, the use of a long speech sequence, and long latency. Therefore, many previous studies on online ASR assumed a BLSTM AM to be an offline AM having better performance but long latency [23–29]. In addition, there has been considerable research on fast training and decoding for a BLSTM AM. References [30–32] proposed a context-sensitive-chunk (CSC) BLSTM, a windowed BLSTM, and latency-controlled (LC) BLSTM, respectively. The CSC BLSTM considers a CSC to be an isolated audio sequence where the CSC comprises a chunk of fixed length and the past and future chunks. The windowed BLSTM is a variant of the CSC BLSTM with jitter training, and it was found that the limited context is adequate rather than an entire utterance. The LC BLSTM is decoded in a similar manner to a conventional unidirectional

LSTM; however, it additionally decodes a fixed number of future frames for BLSTM. Reference [33] reduced the computational complexity of the LC BLSTM by approximating forward or backward BLSTMs as simple networks. Reference [34] utilized the LC BLSTM in an online hybrid connectionist temporal classification (CTC)/attention end-to-end (E2E) ASR. Reference [35] proposed a CTC-based E2E ASR using a chunk-based BLSTM where the chunk size was randomized over a predefined range, and showed accuracy improvement and low latency.

Moreover, there has been considerable research on online ASR systems for increasing the concurrency. Reference [13] proposed a CTC-based E2E ASR system, Deep Speech 2, and used an eager batch dispatch technique of GPU, 16-bit floating-point arithmetic, and beam pruning for real-time and low latency. Reference [14] proposed an online ASR architecture based on time-delay neural network (TDNN) and unidirectional LSTM layers. Reference [15] proposed a recurrent neural network (RNN)-based LM for an online ASR using a quantization of history vectors and the CPU-GPU hybrid scheme. Reference [16] proposed beam search pruning and used the LC BLSTM for an RNN transducer (RNNT)-based E2E ASR. Reference [17] proposed a monotonic chunkwise attention (MoChA) model using hard monotonic attention and soft chunk-wise attention. Reference [18] employed the MoChA-based approach in an E2E ASR using unidirectional LSTM and attention layers for streaming. Reference [19] proposed an online E2E ASR system based on a time-depth separable convolution and CTC for low latency and better throughput. Reference [20] proposed TDNN-based online and offline ASR by proposing a parallel Viterbi decoding based on an optimized, weighted finite-state transducer decoder using GPU, batching multiple audio streams, and so forth.

This paper adopts a CSC BLSTM AM that can decode not a whole input speech but segments of an input speech in order to reduce the decoding latency. To solve the long decoding time, unlike previous works for an online ASR which mostly degrade the accuracy performance [14,15,17,19], we focus on accelerating a GPU parallelization while the accuracy performance is maintained. To solve the long response latency to a user, we also utilize a voice activity detector.

1.3. Proposed Method and Its Contribution

This paper presents an online multichannel ASR system employing a BLSTM AM, which is hardly deployed in industries even though it is one of the best performing AMs. Accordingly, our online ASR system is based on the server-client model where the server can perform speech recognition using high-performance resources and the client interacts with the user and server via various devices. For the baseline server-side ASR system, we utilize a CSC BLSTM where the sizes of a chunk and its left and right contexts of a CSC are 20, 40, and 40 ms, respectively [30]. In fact, the baseline ASR performs in real time; however, the number of concurrent clients is limited. For more concurrent clients, we propose a method for accelerating the GPU parallelization and reducing the transmission overhead between the CPU and GPU based on the fact that a CSC is regarded as an isolated sequence. Moreover, we propose a DNN-based voice activity detector (DNN-VAD) for a low-latency response even though the client sends no end-of-utterance message. To evaluate the proposed online ASR system, we use the test data recorded from Korean documentary programs. The performance in terms of ASR accuracy, real time, and latency is measured as a function of the syllable error rate (SyllER), number of concurrent clients, and the elapsed time in seconds until a user receives the recognized text of they spoke.

The contributions of this paper are summarized as follows:

Fast BLSTM-based online ASR system A BLSTM based AM is commonly regarded as an offline ASR system since a vanilla BLSTM can be decoded after an overall input speech is obtained. This paper successfully deploys a CTC-BLSTM-AM-based online ASR by using the proposed multichannel parallel acoustic score computation and DNN-VAD methods. It is noted that the proposed system can be employed for any language if proper acoustic and language models are prepared for the language though our experiments are conducted only for Korean in this paper.

Parallel acoustic score computation using multichannel data Even though a BLSTM-based online ASR system is successfully deployed using CTC-BLSTM, the number of concurrent clients is still limited due to the massive computational complexity. The proposed acoustic score computation method increases the number of concurrent clients by merging multichannel data and by processing the data in parallel, which accelerates the GPU parallelization and reduces the data transfer between CPU and GPU. From the ASR experiments, the number of concurrent clients is increased from 22 to 44 by using the proposed parallel acoustic score computation method.

DNN-based voice-activity detector An online ASR system needs to send the recognized text to a user as soon as possible even before the end of sentence is detected to reduce response time. To this end, we propose a DNN-based voice-activity detector that detects short pauses in a continuous utterance. The benefits of the proposed method are (a) to reduce user perceived latency from 11.7 sec to 5.41 sec and 3.09 sec respectively depending on the parameter with little degradation in accuray and (b) this is done by the use of the acoustic model score used in ASR without any auxiliary training or the additional computation while decoding.

ASR performance The proposed method maintains the accuracy performance whereas many previous works degrade ASR performance.

Combination with additional optimization An additional optimization method that causes performance degradation can be applied to the proposed ASR. For instance, we utilize a frame skip method during Viterbi decoding to our ASR system. From the ASR experiments, the number of concurrent clients is increased from 44 to 59 while the syllable error rate is degraded from 11.94% to 12.53%.

The rest of this paper is organized as follows. Section 2 describes our configuration of a server-client-based ASR system, an acoustic score computation CTC BLSTM AM, and a baseline multichannel acoustic score computation using a CSC BLSTM AM. Next, Section 3 proposes a fast multichannel parallel acoustic score computation to support more clients and Section 4 proposes a DNN-VAD method for a low-latency response. Section 5 shows our experimental setup and its performance comparison. Finally, we conclude our findings in Section 6.

2. Baseline Multichannel Acoustic Score Computation of a CSC BLSTM AM

In advance to propose parallel multichannel acoustic score computation, we first introduce our server-client-based ASR system and the use of a CTC BLSTM AM [30]. Then, we present a baseline multichannel acoustic score computation method that is used in our CTC BLSTM-based online ASR.

2.1. Configuration of a Server-Client-Based ASR System

This section gives a detailed description on our server-client-based ASR system in terms of main thread a decoder thread.

2.1.1. Main Thread of the Online ASR Server

When a client requests a connection to the ASR server, a main thread is created for the client, which parses the messages from the client where the messages can be a begin-of-audio, audio segments, an end-of-audio, and an end-of-connection. If the begin-of-audio message is received, the main thread creates a decoder thread and prepares speech recognition. If an audio segment is obtained, the main thread extracts a 600-dimensional speech feature vector for each 10 ms audio segment. That is, it extracts 40-dimensional log mel filterbanks for the 10 ms audio segment, and then stacks the features of the past seven frames and the future seven frames. The extracted features are queued into a ring buffer $\mathcal{R}_{feat}$, to perform speech recognition whenever the decoding thread is ready. If the end-of-audio message is received, the main thread waits to finish the decoder thread and terminates the decoder thread and itself.

2.1.2. Decoder Thread of the Online ASR Server

Whenever a decoder thread is in idle or ready state, the thread checks whether the size of the speech features in $\mathcal{R}_{feat}$ is larger than a pre-defined minibatch size, $\mathcal{T}_{bat}$. $\mathcal{T}_{bat}$ indicates the length of the audio samples to be decoded at a time, and is defined during the initialization stage of the ASR server. In this paper, we set $\mathcal{T}_{bat}$ as 200 frames, each of which is 2-s long [14,26,36].

If the decoder thread detects speech features that are larger than $\mathcal{T}_{bat}$, the thread obtains the speech features with a length of ($\mathcal{T}_{bat}$+$\mathcal{T}_{cxt}$), while those with a length of $\mathcal{T}_{bat}$ are popped from $\mathcal{R}_{feat}$. $\mathcal{T}_{cxt}$ is the length of the audio samples for left or right context information [30]. In this paper, we set $\mathcal{T}_{cxt}$ as 40 frames. The obtained features are used to calculate the acoustic scores of a CSC BLSTM AM where the multichannel parallel decoding is proposed in Section 3. The acoustic scores are used in two ways: (a) DNN-VAD and (b) Viterbi decoding. DNN-VAD aims to automatically send the decoded text to the client even though no end-of-audio message is received from the client, where DNN-VAD is proposed in Section 4. If DNN-VAD detects any short pause in an utterance, the thread finds optimal texts up to the time, sends the texts to the client, and resets the search space of Viterbi decoding. Viterbi decoding estimates the probabilities of possible paths using the acoustic scores and an n-gram LM. The processes are repeatedly performed until the decoder thread receives an end-of-audio or end-of-connection message.

2.2. The Use of a CSC BLSTM AM

To efficiently incorporate a BLSTM AM, our ASR system utilizes a CSC-based backpropagation through time (BPTT) and decoding [30]. That is, a CSC comprises an audio chunk of fixed length and its left/right chunks. The chunk size is relatively smaller than the input audio. The ASR system employing the CSC BLSTM AM can reduce the training time and decoding latency as the CSC-based approach regards a CSC as an isolated sequence.

Let us assume the speech features of an overall input sequence of length $\mathcal{T}_{tot}$, as shown in Figure 2a. As mentioned in Section 2.1.1, the decoder thread obtains a speech feature vector with length $\mathcal{T}_{bat}$+$\mathcal{T}_{cxt}$. The features of length $\mathcal{T}_{bat}$ and those of length $\mathcal{T}_{cxt}$ are used as the features to be decoded at the time and its future context, which are drawn in yellow and in gray, respectively, in Figure 2b. As shown in Figure 2c, each speech feature vector is split into a set of chunks of length $\mathcal{T}_{chunk}$. Then, CSC vectors are generated by appending the past features of length $\mathcal{T}_{cxt}$ and the future features of length $\mathcal{T}_{cxt}$ for each split chunk, as shown in Figure 2d. The length ($\mathcal{T}_{win}$) of a CSC vector and the number ($\mathcal{N}_{seg}$) of CSC vectors are defined in Equation (1) and Equation (2), respectively.

$$\mathcal{T}_{win} = \mathcal{T}_{chunk} + 2 \times \mathcal{T}_{cxt}, \tag{1}$$

$$\mathcal{N}_{seg} = \mathcal{T}_{bat} / \mathcal{T}_{chunk}. \tag{2}$$

For an efficient use of GPU parallelization, the CSC vectors are merged into a CSC matrix in the form of $\mathcal{N}_{seg} \times \mathcal{T}_{win}$, as shown in Figure 2e. By using the CSC matrix, the acoustic scores are calculated using a CSC BLSTM AM. In this paper, $\mathcal{T}_{chunk}$ and $\mathcal{T}_{cxt}$ are set as 20 and 40 frames, respectively. Therefore, $\mathcal{T}_{win}$ and $\mathcal{N}_{seg}$ are 100 frames and 10.

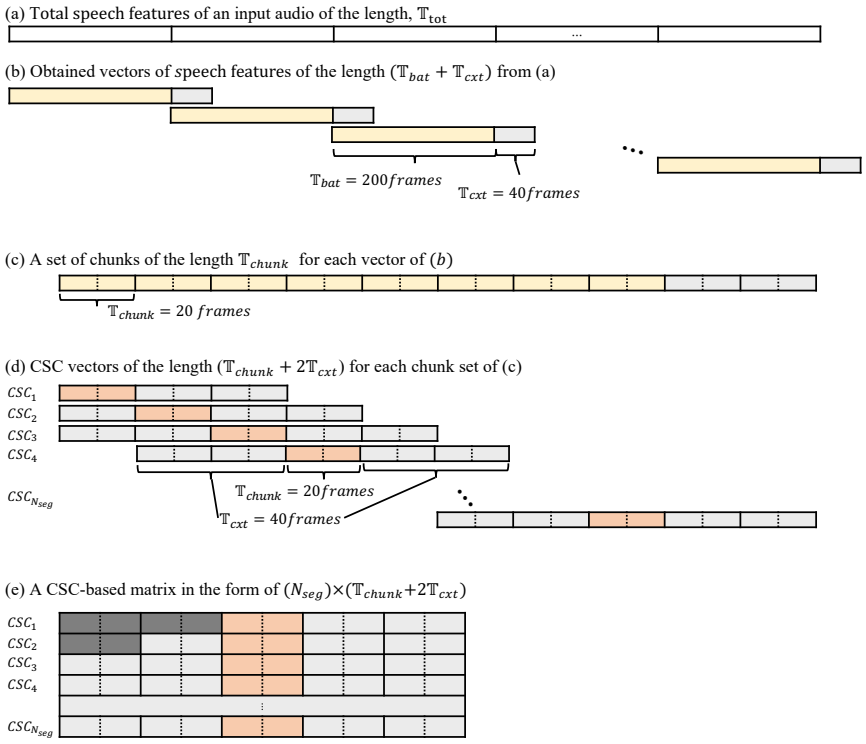


Figure 2. Context-sensitive-chunk (CSCs) of our ASR system with $\mathcal{T}_{bat}$ of 200 frames, $\mathcal{T}_{chunk}$ of 20 frames, and $\mathcal{T}_{cxt}$ of 40 frames: (**a**) extracted speech features from the overall input sequence, (**b**) a set of speech feature vectors where each vector is obtained from $\mathcal{R}_{feat}$ and the size is ($\mathcal{T}_{bat}$+$\mathcal{T}_{cxt}$), (**c**) a set of chunks where the chunk size is $\mathcal{T}_{chunk}$, (**d**) a set of CSCs where the CSC size is ($\mathcal{T}_{cxt}$+$\mathcal{T}_{chunk}$+$\mathcal{T}_{cxt}$), and (**e**) a CSC matrix for a speech feature vector.

2.3. Baseline Multichannel Acoustic Score Computation

When the decoder thread is initialized, the run-time memories are allocated. As shown in Figure 3a, the run-time memories comprise (a) three types of CPU memories, $\mathcal{M}^{CPU}_{feat\_vec}$ for a speech feature vector, $\mathcal{M}^{CPU}_{feat\_mat}$ for a CSC matrix, and $\mathcal{M}^{CPU}_{prob\_vec}$ for acoustic scores, and (b) two types of GPU memories, $\mathcal{M}^{GPU}_{feat\_mat}$ for a CSC matrix and $\mathcal{M}^{GPU}_{prob\_vec}$ for acoustic scores.

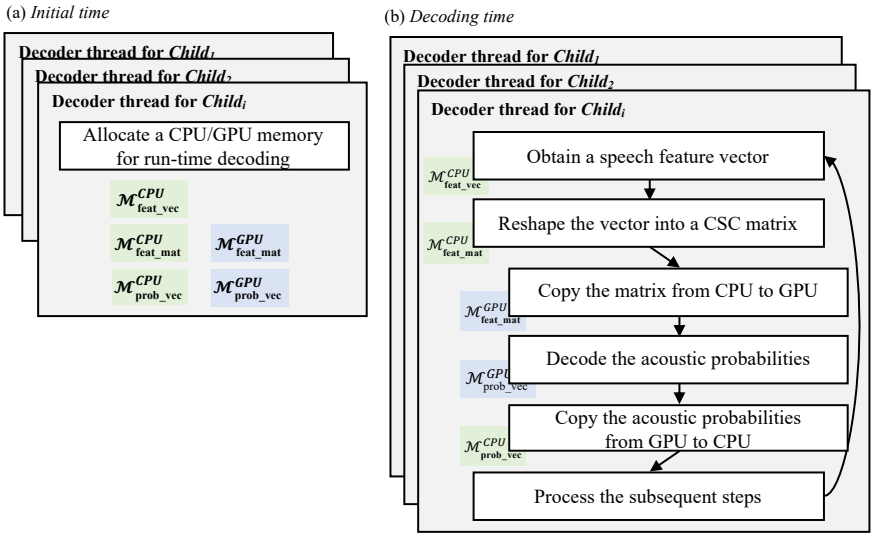


Figure 3. Main procedure of the baseline multichannel acoustic score computation based on CSC bidirectional long short-term memory acoustic model (BLSTM AM). The green- and blue-shaded boxes indicate the CPU and GPU memories, respectively.

Moreover, the sizes of the run-time memories are summarized in Table 1. That is, the size of $\mathcal{M}^{CPU}_{feat\_vec}$ is $\mathcal{N}_{bat}$, which is defined as

$$\mathcal{N}_{bat} = \mathcal{T}_{bat}/\mathcal{T}_{shift} \times \mathcal{N}_{feat\_dim}, \tag{3}$$

where $\mathcal{T}_{shift}$ and N_{feat_dim} are the sizes of frame shift and the speech feature dimension, respectively, which are 10 and 600 frames, as described in Section 2.1.1. Therefore, $\mathcal{N}_{bat}$ is 12000. The size of $\mathcal{M}^{CPU}_{feat\_mat}$ or $\mathcal{M}^{GPU}_{feat\_mat}$ is $\mathcal{N}_{win} \times \mathcal{N}_{seg}$ and that of a CSC, $\mathcal{N}_{win}$, is defined as

$$\mathcal{N}_{win} = \mathcal{T}_{win}/\mathcal{T}_{shift} \times \mathcal{N}_{feat\_dim}. \tag{4}$$

thus, $\mathcal{N}_{win}$ is 6000. The size of $\mathcal{M}^{CPU}_{prob\_vec}$ and $\mathcal{M}^{GPU}_{prob\_vec}$ is $\mathcal{N}_{node}$, which is the number of output nodes of a BLSTM AM and is set as 19901 in this paper..

Table 1. Summarization of the allocated run-time memories at CPU and GPU of the baseline multichannel acoustic score computation.

Thread	Name	Device	Size
Child$_i$	$\mathcal{M}^{CPU}_{feat\_vec}$	CPU	$\mathcal{N}_{bat}$
Child$_i$	$\mathcal{M}^{CPU}_{feat\_mat}$	CPU	$\mathcal{N}_{win} \times \mathcal{N}_{seg}$
Child$_i$	$\mathcal{M}^{CPU}_{prob\_vec}$	CPU	$\mathcal{N}_{node}$
Child$_i$	$\mathcal{M}^{GPU}_{feat\_mat}$	GPU	$\mathcal{N}_{win} \times \mathcal{N}_{seg}$
Child$_i$	$\mathcal{M}^{GPU}_{prob\_vec}$	GPU	$\mathcal{N}_{node}$

Whenever a decoder thread is in idle state and $\mathcal{R}_{feat}$ contains speech features more than $\mathcal{T}_{bat}$, the decoder thread obtains a speech feature vector of length $\mathcal{T}_{bat}$. The speech feature vector is reformed into CSC vectors, which are merged into a CSC matrix. The CSC matrix is then transmitted from CPU to GPU. On GPU, the CSC matrix is normalized using a linear discriminant analysis (LDA)-based transform [37] and the acoustic scores of the matrix are calculated using the BLSTM AM. Next, the acoustic scores are transmitted from GPU to CPU and used in the subsequent steps, such as DNN-VAD and Viterbi decoding. The explained procedures are shown in Figure 3b.

For each feature with a duration of $\mathcal{T}_{bat}$ per client, the transmission sizes are $\mathcal{N}_{win} \times \mathcal{N}_{seg}$ and $\mathcal{N}_{node}$ for a CSC matrix from CPU to GPU and for acoustic scores from GPU to CPU, respectively, as shown in Table 2. Moreover, the transmission frequency is increased by $\mathcal{N}_{channel}$ times if the number of concurrent clients is $\mathcal{N}_{channel}$. However, the frequent data transfer tends to degrade the overall computational performance of a system and causes a small utilization of the GPU [38,39].

Table 2. Summarization of the frequency and transmission sizes between CPU and GPU for each feature with a duration of $\mathcal{T}_{bat}$ if the number of concurrent clients is $\mathcal{N}_{channel}$, when using the baseline multichannel acoustic score computation.

Transmission	Frequency	Size
from CPU to GPU	$\mathcal{N}_{channel}$	$\mathcal{N}_{win} \times \mathcal{N}_{seg}$
from GPU to CPU	$\mathcal{N}_{channel}$	$\mathcal{N}_{node}$

3. Proposed Fast Multichannel Parallel Acoustic Score Computation

Using the baseline acoustic score computation, the number of concurrent clients is restricted due to the frequent data transfer between GPU and CPU and the low parallelization of the GPU [39]. To support more concurrent clients in real time, this section proposes a fast multichannel parallel acoustic score computation method by accelerating the GPU parallelization and reducing the transmission overhead.

As shown in Figure 4, the proposed fast multichannel parallel acoustic score computation is performed with one decoding thread per client and an additional worker thread, whereas the baseline method is performed with no worker thread. When an online ASR server is launched, the server creates a worker thread for a GPU parallel decoding and initializes the maximum number ($\mathcal{N}^{CPU}_{parallel}$) of concurrent clients and that ($\mathcal{N}^{GPU}_{parallel}$) of the GPU parallel decodings. Once the worker thread is initialized, the run-time memories are allocated, as shown in Figure 4a. These memories comprise (a) three types of CPU memories—$\mathcal{M}^{CPU(W)}_{feat\_vec\_f}$ for input feature vectors, $\mathcal{M}^{CPU(W)}_{feat\_vec}$ for feature vectors to be decoded, and $\mathcal{M}^{CPU(W)}_{prob\_vec}$ for acoustic scores, and (b) three types of GPU memories, $\mathcal{M}^{GPU}_{feat\_vec}$ for feature vectors to be decoded, $\mathcal{M}^{GPU}_{feat\_mat}$ for a CSC-based matrix, and $\mathcal{M}^{GPU}_{prob\_vec}$ for acoustic scores. As shown in Figure 4b, one run-time CPU memory is also allocated when a decoder thread for a client is initialized. The run-time CPU memory $\mathcal{M}^{CPU}_{prob\_vec}$ is for the acoustic scores. Moreover, the sizes of the run-time memories are summarized in Table 3.

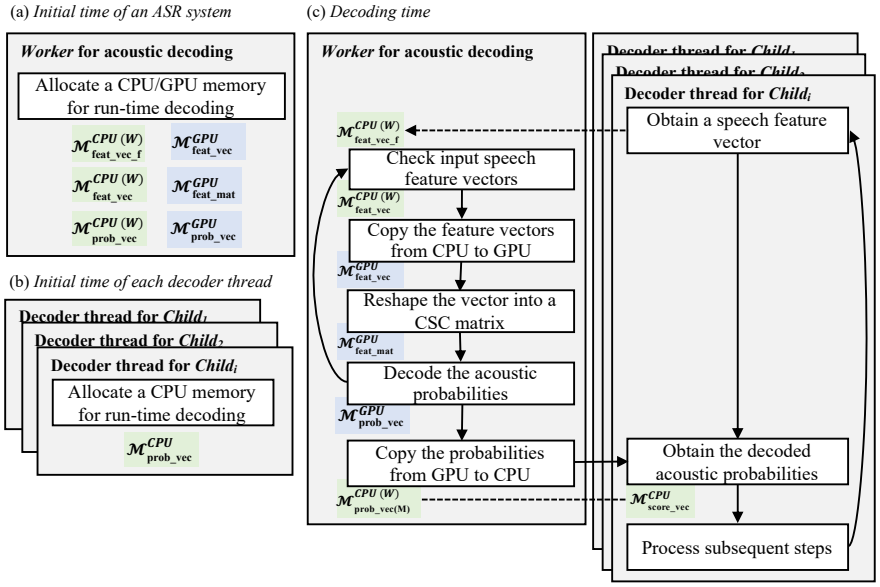


Figure 4. Main procedure of the proposed fast multichannel parallel acoustic score computation using a CSC BLSTM AM. The green- and blue-shaded boxes indicate a CPU memory and a GPU memory, respectively.

Table 3. Summarization of the allocated run-time memories at CPU and GPU of the proposed fast multichannel parallel acoustic score computation.

Thread	Name	Device	Size
Worker	$\mathcal{M}^{CPU(W)}_{feat\_vec\_f}$	CPU	$\mathcal{N}_{bat} \times \mathcal{N}_{channel}$
Worker	$\mathcal{M}^{CPU(W)}_{feat\_vec}$	CPU	$\mathcal{N}_{bat} \times \mathcal{N}^{GPU}{}_{parallel}$
Worker	$\mathcal{M}^{GPU}_{feat\_vec}$	GPU	$\mathcal{N}_{bat} \times \mathcal{N}^{GPU}_{parallel}$
Worker	$\mathcal{M}^{GPU}_{feat\_mat}$	GPU	$\mathcal{N}_{win} \times \mathcal{N}_{seg} \times \mathcal{N}^{GPU}_{parallel}$
Worker	$\mathcal{M}^{GPU}_{prob\_vec}$	GPU	$\mathcal{N}_{node} \times \mathcal{N}^{GPU}_{parallel}$
Worker	$\mathcal{M}^{CPU(W)}_{prob\_vec}$	CPU	$\mathcal{N}_{node} \times \mathcal{N}^{GPU}_{channel}$
Child$_i$	$\mathcal{M}^{CPU}_{prob\_vec}$	CPU	$\mathcal{N}_{node}$

Whenever a decoder thread is in idle state and $\mathcal{R}_{feat}$ contains speech features more than $\mathcal{N}_{bat}$, the decoder thread obtains a speech feature vector of length $\mathcal{T}_{bat}$ and stores it in the CPU memory of the worker thread, $\mathcal{M}^{CPU(W)}_{feat\_vec\_f}$. For instance of an i-th client, the vector is stored at $\mathcal{M}^{CPU(W)}_{feat\_vec\_f}$ with the offset index of

$$\mathcal{N}_{bat} \times i. \tag{5}$$

Then, the decoder thread waits for the acoustic scores to be calculated by the worker thread. On the other hand, whenever the worker thread is in idle state and there are buffered speech feature vectors in $\mathcal{M}^{CPU(W)}_{feat\_vec\_f}$, the worker thread pops k speech feature vectors from $\mathcal{M}^{CPU(W)}_{feat\_vec\_f}$ in first-in, first-out (FIFO) order where k is the number of feature vectors to be decoded at a time and the maximum is $\mathcal{N}^{GPU}_{parallel}$. The obtained k vectors are merged at $\mathcal{M}^{GPU}_{feat\_vec}$, and then transmitted from CPU to GPU ($\mathcal{M}^{GPU}_{feat\_vec}$). On GPU, the transmitted vectors are reformed into CSC-based vectors, which are

merged into a CSC-based matrix in the cascaded form of the CSC-based matrix of each speech feature vector, as follows:

$$\begin{bmatrix} CSC^{(1)} \\ \vdots \\ CSC^{(k)} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} CSC^{(1)}_{1,1} & \cdots & CSC^{(1)}_{1,i} & \cdots & CSC^{(1)}_{1,N_{win}} \\ \vdots & & \vdots & & \vdots \\ CSC^{(1)}_{N_{seg},1} & \cdots & CSC^{(1)}_{N_{seg},i} & \cdots & CSC^{(1)}_{N_{seg},N_{win}} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} CSC^{(k)}_{1,1} & \cdots & CSC^{(k)}_{1,i} & \cdots & CSC^{(k)}_{1,N_{win}} \\ \vdots & & \vdots & & \vdots \\ CSC^{(k)}_{N_{seg},1} & \cdots & CSC^{(k)}_{N_{seg},i} & \cdots & CSC^{(k)}_{N_{seg},N_{win}} \end{bmatrix} \end{bmatrix}, \tag{6}$$

where $CSC^{(i)}$ indicates the CSC-based matrix of the i-th speech feature vector of $\mathcal{M}^{GPU}_{feat\_vec}$. Then, the matrix is normalized using an LDA-based transform [37] and the acoustic scores are calculated into $\mathcal{M}^{GPU}_{prob\_vec}$ using the CSC BLSTM AM. The acoustic scores are in the following cascaded form:

$$\begin{bmatrix} prob^1 & \cdots & prob^k \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} prob^{(1)}_1 & \cdots & prob^{(1)}_i & \cdots & prob^{(1)}_{N_{node}} \end{bmatrix} & \cdots & \begin{bmatrix} prob^{(k)}_1 & \cdots & prob^{(k)}_i & \cdots & prob^{(k)}_{N_{node}} \end{bmatrix} \end{bmatrix}, \tag{7}$$

where $prob^i$ means the acoustic scores of the i-th speech feature vector of $\mathcal{M}^{GPU}_{feat\_vec}$. Next, the acoustic scores are transmitted from GPU to CPU. For instance of $prob^i$, if it is for the m-th client, $prob^i$ is stored at $\mathcal{M}^{CPU(W)}_{prob\_vec}$ with an offset index of

$$\mathcal{N}_{node} \times m. \tag{8}$$

If the waiting decoder thread detects the acoustic scores at $\mathcal{M}^{CPU(W)}_{prob\_vec}$ with the corresponding offset index of Equation (8), the decoder thread copies them into its local memory ($\mathcal{M}^{CPU}_{prob\_vec}$) and proceeds the subsequent steps as the baseline method. The descried procedures are shown in Figure 4c.

As shown in Table 4, the transmission sizes are $\mathcal{N}_{bat} \times k$ for k speech feature vectors from CPU to GPU and $\mathcal{N}_{node} \times k$ for acoustic scores from GPU to CPU, respectively, when the worker thread is ready and $\mathcal{M}^{CPU(W)}_{feat\_vec\_f}$ contains k speech feature vectors. In addition, the frequency is varied according to the size of k from $\mathcal{N}_{channel}/\mathcal{N}^{GPU}_{parallel}$ to $\mathcal{N}_{channel}$.

Assuming that the number of concurrent clients is $\mathcal{N}_{channel}$ and the number of decoded speech feature vectors obtained by the proposed method is $1 \leq k \leq \mathcal{N}^{GPU}_{parallel}$, the main differences between the baseline and proposed acoustic score computation methods are as follows:

Decoding subject(s) The decoder thread of each client calculates acoustic scores in the baseline method, whereas the additionally used worker thread does so in the proposed method.
Transmission frequency The transmission occurs $2 \times \mathcal{N}_{channel}$ times in the baseline method and $2 \times k$ times in the proposed method. Therefore, the proposed method reduces the transfer frequency by k.
Transmission size For a transmission from CPU to GPU, the baseline method transmits $\mathcal{N}_{win} \times \mathcal{N}_{seg}$ for each client, whereas the proposed method $\mathcal{N}_{bat} \times k$ does so for each decoding turn. The total transmission data size to GPU is reduced by the proposed method. On the other hand, for the transmission from GPU to CPU, the baseline method transmits $\mathcal{N}_{node}$ for each client, whereas the proposed method transmits $\mathcal{N}_{node} \times k$ for each decoding turn. The total transmission size to CPU is equal.
Decoding size at a time The baseline method decodes one speech feature vector, whereas the proposed method decodes k vectors. This leads to more GPU parallelization.

Table 4. Summary of the frequency and transmission sizes between CPU and GPU for each feature with a duration of $\mathcal{T}_{bat}$ if the number of concurrent clients is $\mathcal{N}_{channel}$ and the number of decoded speech feature vectors is $k \leq GPU_{parallel}$, when using the proposed fast multichannel parallel acoustic score computation.

Transmission	Min. Frequency	Max. Frequency	Size
from CPU to GPU	$\mathcal{N}_{channel}/\mathcal{N}^{GPU}_{parallel}$	$\mathcal{N}_{channel}$	$\mathcal{N}_{bat} \times k$
from GPU to CPU	$\mathcal{N}_{channel}/\mathcal{N}^{GPU}_{parallel}$	$\mathcal{N}_{channel}$	$\mathcal{N}_{node} \times k$

4. Proposed DNN-Based VAD Method for Low Latency Decoding

Viterbi decoding involves two processes: one estimates probabilities of states in all possible paths, and the other finds an optimal path by backtracking the states with the highest probability. The ASR system yields the results only after both processes are completed, usually at the end of an utterance.

In an online ASR system recognizing long utterances, the end point of an utterance is not known in advance and deciding the back-tracking point affects user experience in terms of response time. If the backtracking is performed infrequently, the user will receive a delayed response, and in the opposite case, the beam search will not find the optimal path that reflects the language model contexts.

In our system, VAD based on an acoustic model for ASR is used to detect short pauses in continuous utterance, which will trigger backtracking. Especially, the acoustic model is built with a deep neural network; hence, we call it DNN-VAD. Here, DNN includes not only a fully connected DNN but also all types of deep models, including LSTM and BLSTM. As explained in previous sections, in our ASR system, BLSTM is used to compute a posterior probability of each state of triphone for each frame. By re-using these values, we can also estimate the probability of non-silence for a given frame with little additional computational cost.

Each output node of the DNN model can be mapped into states of nonsilence or silence phones. Let the output of the DNN model in the i-th node be o_i. Then, the speech probability of the given frame is computed as

$$\log P_{\text{nonsil}} = \max_i o_i, \text{where } i \in \text{non-silence states} \quad (9)$$

$$\log P_{\text{sil}} = \max_i o_i, \text{where } i \in \text{silence states} \quad (10)$$

$$\text{LLR}_{\text{nonsil}} = \log P_{\text{nonsil}} - \log P_{\text{sil}}(t), \quad (11)$$

where LLR is log likelihood ratio. Each frame at time t is decided to be a silence frame if LLR_{nonsil} is smaller than the predefined threshold.

$$s(t) = \begin{cases} 1 & \text{if } \text{LLR}_{\text{nonsil}}(t) < \mathcal{T}_l \\ 0 & \text{otherwise.} \end{cases} \quad (12)$$

In addition, for smoothing purpose, the ratio of silence frames in a window of length $(2W+1)$ is computed and compared to the predefined threshold $\mathcal{T}_r$.

$$\hat{s}(t) = \begin{cases} 1 & \text{if } \sum_{w=-W}^{W} s(t+w)/(2W+1) > \mathcal{T}_r \\ 0 & \text{otherwise.} \end{cases} \quad (13)$$

All computations in from Equation (9) to Equation (13) are performed within each minibatch, and the frame at t when $\hat{s}(t-1) = 1$ and $\hat{s}(t) = 0$ is regarded as the short pause in an utterance. As will be explained in the Section 5.3, the frequent backtracking reduces the response time but also degrades the recognition accuracy. Thus, a minimal interval is set between the detection of short pauses to control the trade-off.

5. Experiment

We select Korean as the target language for the experiments conducted on the proposed methods (Though our experiments are based on Korean speech recognition, the proposed method can be applied to a CTC BLSTM based speech recognition for any language), and all experiments are performed on two Intel(R) Xeon(R) Silver 4214 CPU @ 2.20 GHz and single NVIDIA GeForce RTX 2080 Ti. Section 5.1 describes the corpus and baseline ASR system and compares the performance of the ASR systems employing different AMs. Next, Sections 5.2 and 5.3 present the performances of the proposed parallel acoustic score computation method and the DNN-VAD method, respectively.

5.1. Corpus and Baseline Korean ASR

We use the 3440-h Korean speech and its transcription data to train the baseline Korean ASR system. The speech data comprise approximately 19-million utterances, which are recorded in various environments, such as speaker, noise environment, recording device, and recording scripts. Each utterance is sampled at a rate of 16 kHz and no further augmentation methods are adopted. To evaluate the proposed methods, we prepare a test set recorded from documentary programs. The recordings include voices of narrators and interviewee with and without various background music and noises. The recordings are manually split into 69 segments, each of which are 29.24-s long, on average, and 33.63 min in total.

Each utterance of the training speech data is converted into 600-dimensional speech features. With the extracted speech features, a CSC BLSTM AM is trained using a Kaldi toolkit [40], where the chunk and context sizes of the CSC are 20 and 40 ms, respectively. The AM comprises one input layer, five BLSTM layers, a fully connected layer, and a soft-max layer. Each BLSTM layer comprises 640 BLSTM cells and 128 projection units, while the output layer comprises 19901 units. For the language model, 38 GB of Korean text data is first preprocessed using text-normalization and word segmentation methods [41], and then, the most frequent 540k sub-words are obtained from the text data (For Korean, a sub-word unit is commonly used as a basic unit of an ASR system [42,43]). Next, we train a back-off trigram of 540k sub-words [41] using an SRILM toolkit [44,45].

During decoding, the minibatch size is set to 2 s. Although a larger minibatch size increases the decoding speed owing to the bulk computation of GPU, the latency also increases. We settle into 2 s of minibatch size as a compromise between decoding speed and latency [14,26,36].

To compare the baseline ASR system, we additionally train two types of AMs—(a) DNN-based AM and (b) LSTM-based AM. A DNN-based AM comprises one input layer, eight fully connected hidden layers, and a soft-max layer. Each hidden layer comprises 2048 units and the output layer comprises 19901 units. An LSTM-based AM consists of one input layer, five LSTM layers, a fully connected layer, and a soft-max layer. Each LSTM layer consists of 1024 LSTM cells and 128 projection units, and the output layer consists of 19901 units. The ASR accuracy performance is measured using SyllER, which is calculated as following,

$$SyllER = \frac{S + D + I}{N} \times 100, \tag{14}$$

where S, D, I, and N are the numbers of substuted syllables, deleted syllables, inserted syllables, and reference syllables, respectivey. As shown in Table 5, the BLSTM AM achieves an error rate reduction (ERR) of 20.66% for the test set when compared to the DNN-based AM. In addition, it achieves an ERR of 11.56% for the test set when compared to the LSTM-based AM. Therefore, we employ the BLSTM AM as our baseline ASR system for achieving better ASR accuracy performance (As for the comparision, the Korean speech recognition experiments with the same data set using the Google Cloud API achieved an average SyllER of 14.69%).

Table 5. Comparison of SyllER (%) of the Korean online ASR systems employing deep neural network (DNN), LSTM, and BLSTM AMs for the test set. Our baseline ASR system uses the BLSTM AM.

	SyllER	**ERR with**	
		DNN	**LSTM**
DNN	15.05	-	-
LSTM	13.50	10.30	-
BLSTM	11.94	20.66	11.56

Next, we evaluate the multichannel performance of ASR systems employing the three AMs by examining the maximum number of concurrent clients where an ASR system can be performed in real time. That is, multiple clients are parallelly connected to an ASR server and each client requests to decode the test set. We then measure the real time factor for each client using the following equation

$$\text{RTF}_i = \frac{\text{The processing time for the client } i}{\text{The total duration of the test set}}. \tag{15}$$

Next, we confirm that the number of concurrent clients are performed in real time if the average real-time factors for the concurrent clients are smaller than 1.0. As shown in Table 6, the BLSTM AM supports 22 concurrent clients for the test set, whereas the DNN- or LSTM- based AMs support more concurrent clients. Hereafter, an experimental comparison is performed with only LSTM-based AM as our ASR system is optimized for uni- or bidirectional LSTM-based AMs.

Table 6. Comparison of the multichannel performance of the Korean online ASR systems employing DNN-, LSTM-, and BLSTM AMs for the test set. The evaluation metric is the maximum number of concurrent clients where an ASR system can be performed in real time.

	Max. Clients
DNN	27
LSTM	33
BLSTM	22

Moreover, we evaluate the CPU and GPU usages (%) of the ASR systems using the baseline acoustic score computation method of Section 2 for (a) the LSTM-based AM and (b) the BLSTM AM. The experiments are performed with the test set. From Figure 5, the averaged usages of the CPU and GPU are 83.26% and 51.71% when the LSTM-based AM is employed, and 27.34% and 60.27% when the BLSTM AM is employed. The low usages of GPU can result from the frequent data transfer and low GPU parallelization. Moreover, the low usage of CPU can be observed for the BLSTM AM as CPU takes a long time to wait for the completion of acoustic score computation.

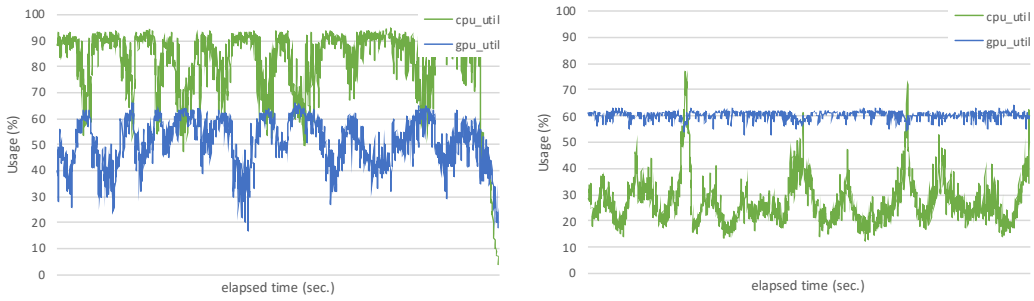


Figure 5. Comparison of CPU and GPU usages (%) of ASR systems using the baseline acoustic score computation method for (a) LSTM-based AM and (b) BLSTM AM with the test set. The green and blue lines indicate the CPU and GPU usages.

5.2. Experiments on the Proposed Fast Parallel Acoustic Score Computation Method

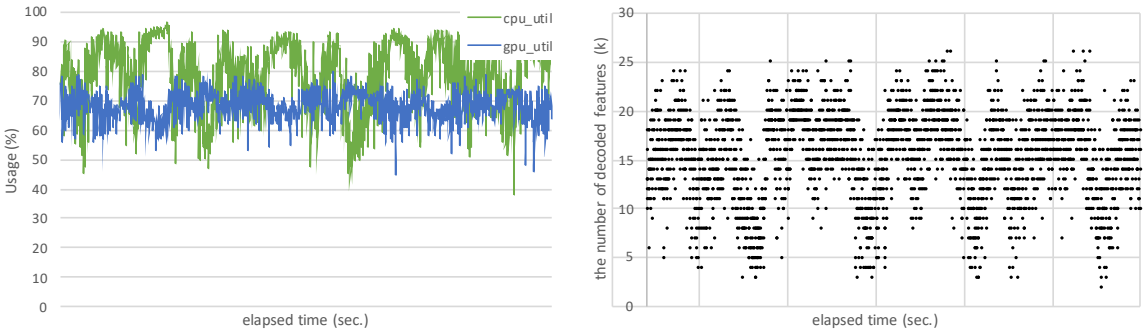
The proposed fast parallel acoustic score computation method can be utilized by replacing the baseline acoustic score computation method with no AM changes in the baseline Korean ASR system. As shown in Table 7, the ASR system using the proposed method of Section 3 achieves the same SyllER when compared to the system using the baseline method of Section 2.3. No performance degradation can be obtained since we only modify the way how the acoustic scores are calculated by accelerating a GPU parallelization. Moreover, the ASR system using the proposed method supports 22 more concurrent clients for the test set, when compared to the system using the baseline method. Therefore, we conclude that the proposed acoustic score computation method increases the concurrent clients with no performance degradation.

Table 7. Comparison of SyllER (%) and the multichannel performance of the Korean online ASR systems using the baseline and proposed acoustic score computation methods of BLSTM AM, with the test set.

	SyllER (%)	**Max. Clients**
baseline	11.94	22
proposed	11.94	44

To analyze the effects of the proposed acoustic score computation method, we compare the CPU and GPU usages (%) of ASR systems using the baseline and proposed methods for the test set. The averaged usages of the CPU and GPU are 78.58% and 68.17%, respectively, when the proposed method is used. By comparing Figure 5b and Figure 6a, the averaged usages of the CPU and GPU are improved by 51.24% and 7.90%, respectively. It can be concluded that the proposed method reduces the processing time of GPU and the waiting time of CPU by reducing the transfer overhead and increasing the GPU parallelization. In addition, we examine the number (k) of parallel decoded feature vectors at each time stamp when using the proposed method, as shown in Figure 6b. The parallel decoded feature vectors varies from 2 to 26, which is depend on the subsequent step, an optimal path search of Viterbi decoding.

For further improvement in the multichannel performance, some optimization methods can be applied, such as beam pruning. In this study, we apply a simple frame skip method during a token passing-based Viterbi decoding. That is, a token propagation is only performed in the odd time stamps during Viterbi decoding. From the experiments of the proposed acoustic score computation, the ASR system combined with the frame skip method supports up to 59 concurrent clients, although the SyllERs are degraded by 4.94% for the set, when compared to the ASR system without the frame skip method, as shown in Table 8.



(**a**) (**b**)

Figure 6. Performances of an ASR system using the proposed fast parallel acoustic score computation method of BLSTM AM with the test set, in terms of (**a**) CPU and GPU usages (%) and (**b**) the number (k) of parallel decoded feature vectors of the proposed method for each acoustic score computation time. The green and blue lines indicate the CPU and GPU usages, respectively.

Table 8. Comparison of SyllER (%) and the multichannel performance of the Korean online ASR systems using the proposed acoustic score computation method of BLSTM AM with/without the frame skip technique on search space expansion during Viterbi decoding, with the set set.

	SyllER (%)	**Max. Clients**
proposed decoding	11.94	44
+ skip frame	12.53	59

Again, we measure the CPU and GPU usages (%) of the ASR systems employing the proposed method with/without the frame skip method with the test set, as shown in Figure 7a. The averaged usages of the CPU and GPU are measured as 47.24% and 71.87%. Note that the frame skip method unburdens the CPU load, and thus, the GPU usage is accordingly improved. Moreover, Figure 7b compares the number of active hypotheses during Viterbi decoding.

In addition, we examine the number (k) of parallel decoded feature vectors at each time stamp when using the proposed method, as shown in Figure 7c. The parallel decoded feature vectors varies from 22 to 35. When compared to the Figure 6b, the parallel decoded vectors are increased due to the reduced computation during Viterbi decoding when combining the frame skip method.

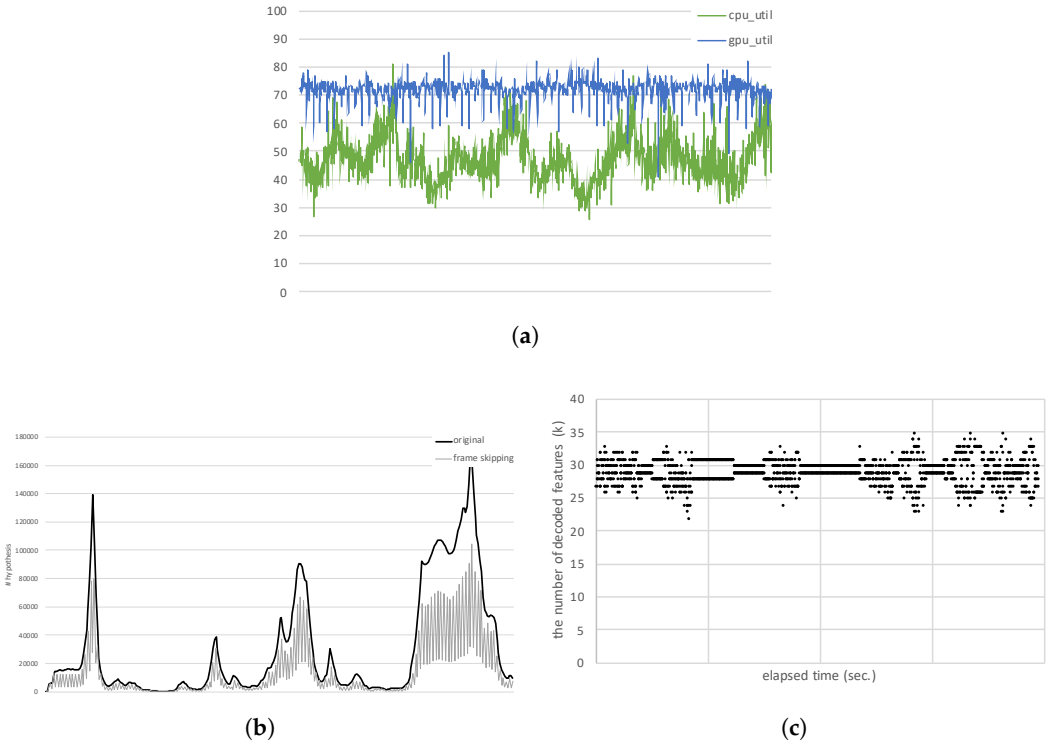


Figure 7. Performances of an ASR system using both the proposed fast parallel acoustic score computation method of BLSTM AM and the frame skip method, with the test set, in terms of (**a**) CPU and GPU usages (%), (**b**) number of hypothesis during Viterbi decoding, and (**c**) the number (k) of parallel decoded feature vectors of the proposed method for each acoustic score computation time. The green and blue lines indicate the CPU and GPU usages, and the straight and dotted lines indicate the proposed decoding without/with frame skip.

5.3. Experiments on DNN-VAD

DNN-VAD is used to reduce the waiting time for a user to receive the ASR results of what they said by triggering backtracking at the possible pause among user utterances. However, frequent backtracking at an improper time can degrade the recognition performance. Hence, in the experiments, the minimum interval between two consecutive backtracking points is set to various values. Table 9 shows the segment lengths divided by DNN-VAD and the recognition accuracy with and without DNN-VAD for test set 1. For example, when the minimum interval is limited to 6 s, an utterance is split into 3.6 segments of 8.2 s each, on average, and the word error rate (WER) is 11.13, which is slightly reduced compared to the case in which VAD is not used, where WER is 11.02.

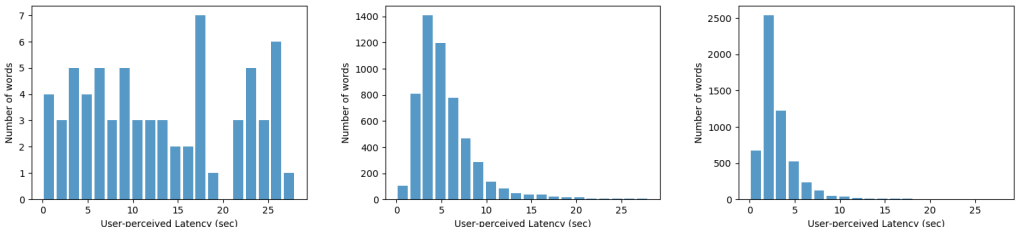
As the minimum interval reduces, the number of segments increases and the length of each segment increases, which means more frequent backtrackings and smaller user-perceived latencies. The accuracy degrades only slightly, which means that the backtracking point is selected reasonably. The internal investigation confirms that the segments are split mostly at the pause between phrases.

Table 9. Average number and duration of split segments and syllable error rates (%) with and without DNN-voice-activity detector (VAD)

Use of DNN-VAD	Minimum Interval(sec)	Number of Segments	Mean and Std. Dev. of Segment Durations(sec)	WER(%)
No	-	29.24	-	11.02
Yes	6	3.6	8.2 ± 3.1	11.13
Yes	4	4.7	6.3 ± 2.9	11.23
Yes	2	6.6	4.4 ± 2.6	11.34
Yes	1	8.3	3.5 ± 2.3	11.29

To measure the waiting time from the viewpoint of users, the user-perceived latency suggested in Reference [19] is used. User-perceived latency is measured for each word uttered, and estimated empirically by measuring the difference in the timestamp of when a transcribed word is available to the user and that in an original audio. The aligned information in the recognition result is used as the timestamp of a word.

The average user-perceived latency is 11.71 s for test set 1 without DNN-VAD, which is very large since all results are received after the end of segments are sent to the server. When DNN-VAD is applied, the average latency is reduced to 5.41 s with a minibatch of 200 frames and 3.09 s with a minibatch of 100 frames. For a detailed analysis, the histogram of latency for each word is shown in Figure 8.



(**a**) (**b**) (**c**)

Figure 8. Histogram of the user-perceived latency for all words in test set 1. (**a**) Without VAD (**b**) WithVAD, minibatch 200 (**c**) WithVAD, minibatch 100.

6. Conclusions

In this paper, we presented a server-client-based online ASR system employing a BLSTM AM, which is a state-of-the-art AM. Accordingly, we adopted a CSC-based training and decoding approach for a BLSTM AM and proposed the following: (a) the parallel acoustic score computation to support more clients concurrently and (b) DNN-VAD to reduce the waiting time for a user to receive the recognition results. On the designed server-client-based ASR system, a client captures the audio signal from a user, sends the audio data to the ASR server, receives a decoded text from the server, and presents it to the user. The client can be deployed in various devices, from low to high performance. On the other hand, a server performs speech recognition using high-performance resources. That is, the server manages the main thread and decoder thread for each client and an additional worker thread for the proposed parallel acoustic computation method. The main thread communicates with the connected client, extracts speech features, and buffers them. The decoder thread performs speech recognition and sends the decoded text to the connected client. Speech recognition is performed in three main steps: acoustic score computation using a CSC BLSTM AM, DNN-VAD to detect a short pause in a long continuous utterance, and Viterbi decoding to search an optimal text using an LM. To handle more concurrent clients in real time, we first proposed the acoustic score computation method by merging the speech feature vectors collected from multiple clients, to reduce the amount of data transfer between the CPU and GPU, and calculating the acoustic scores with the merged data to increase GPU parallelization and to reduce the transfer overhead between the CPU and GPU. Second, we proposed DNN-VAD to detect a short pause in an utterance for a low latency response to a user. The Korean ASR experiments conducted using the broadcast audio data showed that the proposed acoustic score computation method increased the maximum number of concurrent clients from 22 to 44. Furthermore, by applying the frame skip method during Viterbi decoding, the maximum number of concurrent clients was increased to 59, although SyllER was degraded from 11.94% to 12.53%. Moreover, the average user-perceived latencies were reduced to 5.41 and 3.09 s with a minibatch of 200 frames and 100 frames, respectively, when the proposed DNN-VAD was used.

Author Contributions: The authors discussed the contents of the manuscript. Methodology, validation, writing, editing, and formal analysis, Y.R.O. and K.P.; project administration, J.G.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.2019-0-01376, Development of the multi-speaker conversational speech recognition technology)

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AM	acoustic model
ASR	automatic speech recognition
BLSTM	bidirectional long short-term memory
BPTT	backpropagation through time
CSC	context-sensitive-chunk
CTC	connectionist temporal classification
DNN	deep neural network
E2E	end-to-end
ERR	error rate reduction
LC	latency-controlled
LDA	linear discriminant analysis
LLR	log likelihood ratio
LM	language model
LSTM	long short-term memory
LVCSR	large vocabulary continuous speech recognition
MoChA	monotonic chunkwise attention
RNN	recurrent neural network
RNN-T	RNN transducer
SyllER	syllable error rate
TDNN	time-delay neural network
VAD	voice activity detector

References

1. Deng, L. Industrial Technology Advances: Deep Learning—From Speech Recognition to Language and Multimodal Processing. In *APSIPA Transactions on Signal and Information Processing*; Cambridge University Press: Cambridge, MA, USA, 2016.
2. Zhang, Z.; Geiger, J.; Pohjalainen, J.; Mousa, A.E.D.; Jin, W.; Schuller, B. Deep Learning for Environmentally Robust Speech Recognition: An Overview of Recent Developments. *ACM Trans. Intell. Syst. Technol.* **2018**, *9*, doi:10.1145/3178115.
3. Hatcher, W.G.; Yu, W. A Survey of Deep Learning: Platforms, Applications and Emerging Research Trends. *IEEE Access* **2018**, *6*, 24411–24432, doi:10.1109/ACCESS.2018.2830661.
4. Nassif, A.B.; Shahin, I.; Attili, I.; Azzeh, M.; Shaalan, K. Speech Recognition Using Deep Neural Networks: A Systematic Review. *IEEE Access* **2019**, *7*, 19143–19165, doi:10.1109/ACCESS.2019.2896880.
5. Alam, M.; Samad, M.D.; Vidyaratne, L.; Glandon, A.; Iftekharuddin, K.M. Survey on Deep Neural Networks in Speech and Vision Systems. *arXiv* **2019**, arXiv:1908.07656.
6. Kim, J.Y.; Liu, C.; Calvo, R.A.; McCabe, K.; Taylor, S.; Schuller, B.W.; Wu, K. A Comparison of Online Automatic Speech Recognition Systems and the Nonverbal Responses to Unintelligible Speech. *arXiv* **2019**, arXiv:1904.12403.
7. Park, J.; Boo, Y.; Choi, I.; Shin, S.; Sung, W. Fully Neural Network Based Speech Recognition on Mobile and Embedded Devices. In *Advances in Neural Information Processing Systems 31*; Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2018; pp. 10620–10630.

8. Shangguan, Y.; Li, J.; Liang, Q.; Alvarez, R.; Mcgraw, I. Optimizing Speech Recognition For The Edge. *arXiv* **2019**, arXiv:1909.12408.
9. Sim, K.C.; Zadrazil, P.; Beaufays, F. An Investigation into On-Device Personalization of End-to-End Automatic Speech Recognition Models. In Proceedings of the 2019 Interspeech, Graz, Austria, 15 September 2019; pp. 774–778, doi:10.21437/Interspeech.2019-1752.
10. Ismail, A.; Abdlerazek, S.; El-Henawy, I.M. Development of Smart Healthcare System Based on Speech Recognition Using Support Vector Machine and Dynamic Time Warping. *Sustainability* **2020**, *12*, doi:10.3390/su12062403.
11. He, Y.; Sainath, T.N.; Prabhavalkar, R.; McGraw, I.; Alvarez, R.; Zhao, D.; Rybach, D.; Kannan, A.; Wu, Y.; Pang, R.; et al. Streaming End-to-end Speech Recognition for Mobile Devices. In Proceedings of the ICASSP 2019—2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 17 May 2019; pp. 6381–6385, doi:10.1109/ICASSP.2019.8682336.
12. Dong, P.; Wang, S.; Niu, W.; Zhang, C.; Lin, S.; Li, Z.; Gong, Y.; Ren, B.; Lin, X.; Wang, Y.; et al. RTMobile: Beyond Real-Time Mobile Acceleration of RNNs for Speech Recognition. *arXiv* **2020**, arXiv:2002.11474.
13. Amodei, D.; Ananthanarayanan, S.; Anubhai, R.; Bai, J.; Battenberg, E.; Case, C.; Casper, J.; Catanzaro, B.; Chen, J.; Chrzanowski, M.; et al. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. In Proceedings of the International Conference on Machine Learning, Lille, France, 11 July 2015.
14. Peddinti, V.; Wang, Y.; Povey, D.; Khudanpur, S. Low Latency Acoustic Modeling Using Temporal Convolution and LSTMs. *IEEE Signal Process. Lett.* **2018**, *25*, 373–377, doi:10.1109/LSP.2017.2723507.
15. Lee, K.; Park, C.; Kim, N.; Lee, J. Accelerating Recurrent Neural Network Language Model Based Online Speech Recognition System. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AL, Canada, 20 April 2018; pp. 5904–5908, doi:10.1109/ICASSP.2018.8462334.
16. Jain, M.; Schubert, K.; Mahadeokar, J.; Yeh, C.F.; Kalgaonkar, K.; Sriram, A.; Fuegen, C.; Seltzer, M.L. RNN-T For Latency Controlled ASR With Improved Beam Search. *arXiv* **2019**, arXiv:cs.CL/1911.01629.
17. Kim, K.; Lee, K.M.; Gowda, D.N.; Park, J.; Kim, S.; Jin, S.; Lee, Y.Y.; Yeo, J.; Kim, D.; Jung, S.; et al. Attention Based On-Device Streaming Speech Recognition with Large Speech Corpus. In Proceedings of the 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Sentosa, Singapore, 18 December 2019; pp. 956–963.
18. Kim, C.; Kim, S.; Kim, K.; Kumar, M.; Kim, J.; Lee, K.M.; Han, C.W.; Garg, A.; Kim, E.; Shin, M.; et al. End-to-End Training of a Large Vocabulary End-to-End Speech Recognition System. In Proceedings of the 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Sentosa, Singapore, 18 December 2019; pp. 562–569.
19. Pratap, V.; Xu, Q.; Kahn, J.; Avidov, G.; Likhomanenko, T.; Hannun, A.; Liptchinsky, V.; Synnaeve, G.; Collobert, R. Scaling Up Online Speech Recognition Using ConvNets. *arXiv* **2020**, arXiv:2001.09727.
20. Braun, H.; Luitjens, J.; Leary, R. GPU-Accelerated Viterbi Exact Lattice Decoder for Batched Online and Offline Speech Recognition. *arXiv* **2019**, arXiv:1910.10032.
21. Kim, J.; Lane, I. Accelerating multi-user large vocabulary continuous speech recognition on heterogeneous CPU-GPU platforms. In Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Shanghai, China, 25 March 2016; pp. 5330–5334, doi:10.1109/ICASSP.2016.7472695.
22. Yang, Y.P. An Innovative Distributed Speech Recognition Platform for Portable, Personalized and Humanized Wireless Devices. *IJCLCLP* **2004**, *9*, 77–94.
23. Tang, J.; Zhang, S.; Wei, S.; Dai, L.R. Future Context Attention for Unidirectional LSTM Based Acoustic Model. In Proceedings of the 2016 Interspeech, San Francisco, CA, USA, 8 September 2016; pp. 3394–3398, doi:10.21437/Interspeech.2016-185.
24. Mirco Ravanelli.; Dmitriy Serdyuk.; Yoshua Bengio. Twin Regularization for Online Speech Recognition. In Proceedings of the 2018 Interspeech, Hyderabad, India, 2 September 2018; pp. 3718–3722, doi:10.21437/Interspeech.2018-1407.
25. Li, J.; Wang, X.; Zhao, Y.; Li, Y. Gated Recurrent Unit Based Acoustic Modeling with Future Context. In Proceedings of the 2018 Interspeech, Hyderabad, India, 2 September 2018; pp. 1788–1792, doi:10.21437/Interspeech.2018-1544.
26. Li, J.; Shan, Y.; Wang, X.; Li, Y. Improving Gated Recurrent Unit Based Acoustic Modeling with Batch Normalization and Enlarged Context. In Proceedings of the 2018 11th International Symposium on Chinese Spoken Language Processing (ISCSLP), Taipei, Taiwan, 26 November 2018; pp. 126–130.

27. Li, S.; Lu, X.; Takashima, R.; Shen, P.; Kawahara, T.; Kawai, H. Improving CTC-based Acoustic Model with Very Deep Residual Time-delay Neural Networks. In Proceedings of the 2018 Interspeech, Hyderabad, India, 2 September 2018; pp. 3708–3712, doi:10.21437/Interspeech.2018-1475.
28. Moritz, N.; Hori, T.; Roux, J.L. Unidirectional Neural Network Architectures for End-to-End Automatic Speech Recognition. In Proceedings of the 2019 Interspeech, Graz, Austria, 15 September 2019; pp. 76–80, doi:10.21437/Interspeech.2019-2837.
29. Moritz, N.; Hori, T.; Roux, J.L. Streaming End-to-End Speech Recognition with Joint CTC-Attention Based Models. In Proceedings of the 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Sentosa, Singapore, 18 December 2019; pp. 936–943, doi:10.1109/ASRU46091.2019.9003920.
30. Chen, K.; Yan, Z.J.; Huo, Q. Training Deep Bidirectional LSTM Acoustic Model for LVCSR by a Context-Sensitive-Chunk BPTT Approach. In Proceedings of the 2015 Interspeech, Dresden, Germany, 6 September 2015; pp. 3600–3604.
31. Mohamed, A.; Seide, F.; Yu, D.; Droppo, J.; Stoicke, A.; Zweig, G.; Penn, G. Deep bi-directional recurrent networks over spectral windows. In Proceedings of the 2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), Scottsdale, AZ, USA, 10 July 2015; pp. 78–83, doi:10.1109/ASRU.2015.7404777.
32. Zhang, Y.; Chen, G.; Yu, D.; Yaco, K.; Khudanpur, S.; Glass, J. Highway long short-term memory RNNS for distant speech recognition. In Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Shanghai, China, 25 March 2016; pp. 5755–5759, doi:10.1109/ICASSP.2016.7472780.
33. Xue, S.; Yan, Z. Improving latency-controlled BLSTM acoustic models for online speech recognition. In Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 5 March 2017; pp. 5340–5344, doi:10.1109/ICASSP.2017.7953176.
34. Miao, H.; Cheng, G.; Zhang, P.; Li, T.; Yan, Y. Online Hybrid CTC/Attention Architecture for End-to-End Speech Recognition. In Proceedings of the 2019 Interspeech, Graz, Austria, 15 September 2019; pp. 2623–2627, doi:10.21437/Interspeech.2019-2018.
35. Audhkhasi, K.; Saon, G.; Tüske, Z.; Kingsbury, B.; Picheny, M. Forget a Bit to Learn Better: Soft Forgetting for CTC-Based Automatic Speech Recognition. In Proceedings of the 2019 Interspeech, Graz, Austria, 15 September 2019; pp. 2618–2622, doi:10.21437/Interspeech.2019-2841.
36. Scovell, J.; Beltman, M.; Doherty, R.; Elnaggar, R.; Sreerama, C. Impact of Accuracy and Latency on Mean Opinion Scores for Speech Recognition Solutions. *Procedia Manuf.* **2015**, *3*, 4377–4383.
37. Haeb-Umbach, R.; Ney, H. Linear discriminant analysis for improved large vocabulary continuous speech recognition. In Proceedings of the 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing, San Francisco, CA, USA, 26 March 1992; Volume 1, pp. 13–16, doi:10.1109/ICASSP.1992.225984.
38. Boyer, M.; Meng, J.; Kumaran, K. Improving GPU Performance Prediction with Data Transfer Modeling. In Proceedings of the 2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum, Boston, MA, USA, 24 May 2013; pp. 1097–1106, doi:10.1109/IPDPSW.2013.236.
39. Li, X.; Zhang, G.; Huang, H.H.; Wang, Z.; Zheng, W. Performance Analysis of GPU-Based Convolutional Neural Networks. In Proceedings of the 2016 45th International Conference on Parallel Processing (ICPP), Philadelphia, PA, USA, 16 August 2016; pp. 67–76, doi:10.1109/ICPP.2016.15.
40. Povey, D.; Ghoshal, A.; Boulianne, G.; Goel, N.; Hannemann, M.; Qian, Y.; Schwarz, P.; Stemmer, G. The Kaldi Speech Recognition Toolkit. In Proceedings of the 2011 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), Waikoloa, Hawaii, USA, 11 December 2011.
41. Chung, E.; Park, J.G. Sentence-Chain Based Seq2seq Model for Corpus Expansion. *ETRI J.* **2017**, *39*, 455–466.
42. Kwon, O.W.; Park, J. Korean Large Vocabulary Continuous Speech Recognition with Morpheme-Based Recognition Units. *Speech Commun.* **2003**, *39*, 287–300, doi:10.1016/S0167-6393(02)00031-6.
43. Park, H.; Seo, S.; Rim, D.J.; Kim, C.; Son, H.; Park, J.; Kim, J. Korean Grapheme Unit-based Speech Recognition Using Attention-CTC Ensemble Network. In Proceedings of the 2019 International Symposium on Multimedia and Communication Technology (ISMAC), Quezon City, Philippines, 19 August 2019; pp. 1–5, doi:10.1109/ISMAC.2019.8836146.

44. Stolcke, A. SRILM—An extensible language modeling toolkit. In Proceedings of the International Conference on Spoken Language Processing (ICSLP), Jerusalem, Israel, 26 November 2002; pp. 901–904.
45. Jeon, H.B.; Lee, S.Y. Language Model Adaptation Based on Topic Probability of Latent Dirichlet Allocation. *ETRI J.* **2016**, *38*, 487–493.