On-device Streaming Discrete Speech Units

Kwanghee Choi*, Masao Someki*, Emma Strubell, Shinji Watanabe

[1]Language Technologies Institute, Carnegie Mellon University, USA

{kwanghec,msomeki,estrubel,swatanab}@andrew.cmu.edu

Abstract

Discrete speech units (DSUs) are derived from clustering the features of self-supervised speech models (S3Ms). DSUs offer significant advantages for on-device streaming speech applications due to their rich phonetic information, high transmission efficiency, and seamless integration with large language models. However, conventional DSU-based approaches are impractical as they require full-length speech input and computationally expensive S3Ms. In this work, we reduce both the attention window and the model size while preserving the effectiveness of DSUs. Our results demonstrate that we can reduce floating-point operations (FLOPs) by 50% with only a relative increase of 6.5% in character error rate (CER) on the ML-SUPERB 1h dataset. These findings highlight the potential of DSUs for real-time speech processing in resource-constrained environments.

Index Terms: discrete speech units, streaming, on-device, speech recognition

1. Introduction

Self-supervised speech models (S3Ms) have demonstrated remarkable efficacy in various speech-related tasks, including automatic speech recognition (ASR), text-to-speech synthesis, speech translation, speaker identification, and emotion recognition [1–3]. With the advent of S3Ms, many have started to leverage their features for tokenizing speech, *i.e.*, producing discrete speech units (DSUs) [4–10]. DSUs are typically generated by applying clustering methods such as k-means to S3M features, with the resulting clusters treated as DSUs.

The characteristics of DSUs are well adapted to on-device streaming scenarios, offering cheap transmission and easy adaptation to existing large language models (LLMs). Unlike high-dimensional speech features, such as spectral speech features or S3M features, DSUs offer improved data storage and transmission efficiency [4, 10] while maintaining comparable ASR performance [5]. These advantages make DSUs a promising medium for speech data transmission. For instance, a 1-second 16kHz audio signal requires 512kbps in raw form. S3M features (*e.g.*, wav2vec 2.0-large [2]) increase this to 1600kbps. However, DSUs with 2k clusters reduce this to just 0.6kbps, significantly compressing the data by 3-4 orders of magnitude. DSUs also offer a promising way to extend LLMs to speech modalities [8, 9, 11]. A practical application involves generating DSUs on client devices and transmitting them to server-side LLMs for processing. This may also provide benefits for anonymizing personally identifiable information for privacy concerns [12] as S3Ms lose speaker information at later layers [13].
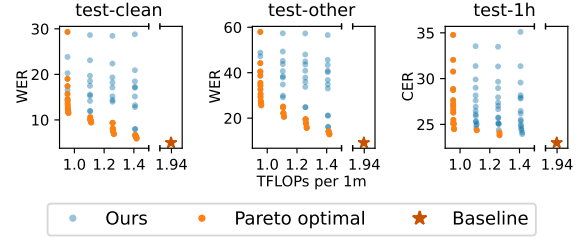

Figure 1: *Pareto tradeoff between ASR performance (WER on Librispeech, CER on ML-SUPERB 1h) and computational efficiency (TFLOPs per 1 minute input) for various modifications on WavLM-large. Results show that we can extract DSUs in a lightweight way with limited impact on ASR performance.*

However, DSUs have yet to be leveraged for streaming on edge devices due to two major challenges. First, conventional S3Ms rely on bidirectional Transformer encoder architectures, with full attention window size to cover the entire input speech, which is not immediately compatible with the streaming setting. Second, state-of-the-art S3Ms typically require hundreds of millions of parameters, which can be computationally infeasible to execute on edge devices with limited computational capacity. It poses a problem for on-device streaming, where GPU memory restricts the total sequence length and prevents real-time generation of DSUs.

In summary, a model for extracting DSUs must work well even with a limited window of future frames while being computationally lightweight. Evidence from previous work suggests this should be possible. For example, prior work has indicated that S3Ms predominantly encode phonetic information, with limited syntactic and semantic content [13–16]. Furthermore, studies have shown a strong correlation between extracted DSUs and phonemes [2,17]. These findings indicate that DSUs can be extracted with a smaller window size. Regarding the model size, we hypothesize that we can reduce the number of required parameters by solely targeting the specific input distribution, similar to the ideas of previous work [18,19]. Typical S3Ms are self-supervised on diverse data to perform well on a wide variety of audio, *i.e.*, "universal audio feature." However, in practice, DSUs would still be useful even if tailored for specific target data and tasks.

In this work, we investigate the feasibility of DSUs for streaming on edge devices by exploring the following two key questions: 1. How much temporal window size is needed? (Section 3); 2. How much model capacity (parameters) is needed? (Section 4). We investigate tradeoffs between computational efficiency and downstream performance in this set-

---

*These authors contributed equally.

ting by varying the attention window size and the number of layers of WavLM-large [3], a representative S3M for extracting DSUs [6]. For each setting, a smaller model is trained by regarding the full model's DSUs as ground truth. We evaluate feasibility by measuring the computational cost, comparing with ASR performance on Librispeech [20] and ML-SUPERB [21].

Our contributions in this paper are summarized as follows:

- To the best of our knowledge, this study is the first to explore DSUs in the on-device streaming setting, exploring the feasibility of real-time lightweight computation of DSUs.
- We explore the trade-off between downstream performance and computational overhead by varying the attention window size and the number of layers of the original S3M, obtaining Pareto optimal trade-off curve.
- By optimizing the model's attention window size and number of layers, we reduce FLOPs by 50% while having only a 6% relative increase in CER on the ML-SUPERB 1h dataset.

2. Experimental Settings

Our work investigates whether DSUs can be accurately predicted using a smaller model. To evaluate the predicted DSUs, we focus on the discrete ASR system. We use the same experimental settings across all experiments.

2.1. Discrete ASR System

Discrete ASR system contains two main modules: speech-to-unit (S2U) module and unit-to-text (U2T) module. S2U module generates DSUs, *i.e.*, speech discretization from continuous S3M features. Given the input raw speech $\mathbf{x} \in \mathbb{R}^t$ with length t, f_{S3M} transforms speech into S3M features:

$$\mathbf{S} = f_{\text{S3M}}(\mathbf{x}) \in \mathbb{R}^{T \times F}, \tag{1}$$

where F is the feature dimension and T is the temporal dimension roughly proportional to t. Then, the pretrained k-means algorithm is applied to yield the stream of DSUs $\mathbf{D} = f_{\text{kmeans}}(\mathbf{S})$. The cascade of S3M and k-means becomes the S2U module:

$$f_{\text{S2U}}(\mathbf{x}) = f_{\text{kmeans}}(f_{\text{S3M}}(\mathbf{x})) = \mathbf{D} = [D_1, D_2, \cdots, D_T], \tag{2}$$

where $D_i \in \{1, 2, \cdots, V\}$ is the DSU at timestep i and V is the vocabulary size, *i.e.*, k in k-means.

After generating DSUs from speech through the S2U module (eq. (2)), U2T module translates DSUs to text transcriptions. To reduce the sequence length, additional post-processing methods are often applied, such as deduplication or byte-pair encoding tokenization [5, 6]:

$$\mathbf{D}' = f_{\text{subword}}(\mathbf{D}) = [D'_1, D'_2, \cdots, D'_{T'}], \tag{3}$$

where $D'_j \in \{1, 2, \cdots, V'\}$ is the post-processed DSU at timestep j. Often, the sequence length T' becomes shorter and vocabulary size V' becomes larger, *i.e.*, $T' \leq T$, and $V' \geq V$.

Finally, the discrete ASR model f_{ASR} is trained to predict the transcription $\mathbf{y}$. The cascade of post-processing methods (eq. (3)) and discrete ASR model becomes the U2T module:

$$f_{\text{U2T}}(\mathbf{D}) = f_{\text{ASR}}(f_{\text{subword}}(\mathbf{D})) = f_{\text{ASR}}(\mathbf{D}') = \hat{\mathbf{y}}, \tag{4}$$

where $\hat{\mathbf{y}}$ is the prediction of $\mathbf{y}$.

In summary, transcription $\mathbf{y}$ is predicted from input speech $\mathbf{x}$ through the cascade of S2U (eq. (2)) and U2T (eq. (4)):

$$\hat{\mathbf{y}} = f_{\text{U2T}}(f_{\text{S2U}}(\mathbf{x})) = f_{\text{ASR}}(f_{\text{subword}}(f_{\text{kmeans}}(f_{\text{S3M}}(\mathbf{x})))), \tag{5}$$

where the DSUs $\mathbf{D} = f_{\text{kmeans}}(f_{\text{S3M}}(\mathbf{x}))$ becomes the communication medium between the two modules. Often, both modules are not trained in an end-to-end manner, but separately trained.

2.2. Training a DSU predictor

Our aim is to train a lightweight DSU predictor $\bar{f}_{\text{S2U}}$ that predicts the DSUs $\mathbf{D}$ given the speech input $\mathbf{x}$, so that it can replace f_{S2U} within the discrete ASR system. To make $\bar{f}_{\text{S2U}}$ to be trainable in an end-to-end manner, we replace the k-means model d with a trainable fully-connected layer (FC):

$$f_{\text{FC}}(f_{\text{S3M}}(\mathbf{x})) = [\hat{\mathbf{D}}_1, \hat{\mathbf{D}}_2, \cdots, \hat{\mathbf{D}}_T], \tag{6}$$

where $\hat{\mathbf{D}}_i \in \mathbb{R}^V$ is the pre-softmax logits at timestep i. The final predicted DSU becomes $\hat{D}_i = \text{argmax}_v \hat{\mathbf{D}}_i[v]$, such that:

$$\bar{f}_{\text{S2U}}(\mathbf{x}) = \text{argmax}_v f_{\text{FC}}(f_{\text{S3M}}(\mathbf{x})) = [\hat{D}_1, \hat{D}_2, \cdots, \hat{D}_T], \tag{7}$$

with a slight abuse of notation for the argmax operator. In summary, we use the original DSUs $\mathbf{D}$ as the ground truth label for the lightweight DSU predictor $\bar{f}_{\text{S2U}}$, so that we can replace the original S2U module f_{S2U} within the discrete ASR system.

2.3. System Evaluation

We use the existing discrete ASR system challenge [6], which uses frozen WavLM-Large [3] 21st layers' features to extract DSUs. The challenge uses two datasets: LibriSpeech [20] and ML-SUPERB 1-hour subset [21], which covers English and 143 languages, respectively. We denote the clean and noisy subset of LibriSpeech as test-clean and test-other, and ML-SUPERB test set as test-1h. To measure the downstream performance, we use word error rate (WER) for test-clean and test-other, and character error rate (CER) for test-1h, following [5]. Also, we use tera (10^{12}) floating point operations (TFLOPs) per one minute of input audio to measure the computational cost of the S2U module (eqs. (2) and (7)). It differs from the original challenge, which uses bitrate to focus on the transmission efficiency of the DSU itself, not the overhead of extracting DSUs. We use calflops [22] to calculate FLOPs.

2.4. Experimental Details

Following [6], we use the 12-layer E-branchformer encoder [23] and 6-layer Transformer decoder [24] for the discrete ASR model f_{ASR} (eq. (4)). However, for computational efficiency, we reduce the beam size from 20 to 5. For all of our experiments, we used the AdamW optimizer [25] with learning rate 1e-4 and weight decay 1e-6. We used the step learning rate decay with a rate of 0.9 every 1K steps. We use the variable batch size with 2M frames for 10 epochs, setting patience to 1. Refer to our codebase for more details.[1]

3. Reducing the Attention Window Size

Why does window size matter? S3Ms leverage Transformers with the full attention window, which requires the full audio to produce discrete units, making the streaming scenario impossible. However, this issue can be mitigated by limiting the future window size of the S2U module. The smaller the required future frames, the smaller the theoretical delay of the system for streaming scenarios. Additionally, reducing the input size enables lower computational overhead.
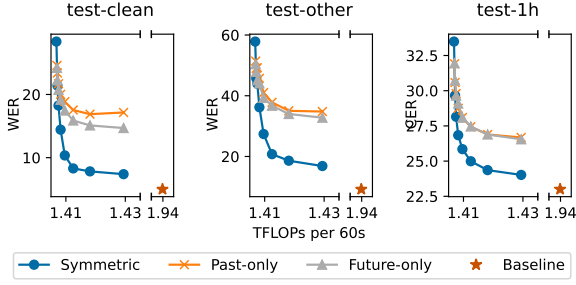

Figure 2: *Results on various attention window sizes when extracting DSUs. Baseline uses the full speech input, where others use sparse attention with symmetric and asymmetric windows.*

Experimental settings. We aim to answer whether the entire input is required or if we can yield similar downstream performance with limited attention window size. We regard the case where we input the full speech as the strong baseline (eq. (2)). We test various window sizes by varying the number of past and future frames. Motivated by the codebase of WavLM [3], we use a streaming mask, *i.e.*, time-restricted self-attention [26, 27], to limit the window size. We fully fine-tune the DSU predictor $\bar{f}_{S2U}$ (eq. (7)) while using the same U2T module f_{U2T} (eq. (4)) as the baseline.

We denote various attention window configurations as the number of left, center, and right frames, *i.e.*, $[l, c, r]$. Given the number of layers n, the temporal receptive field size is $(l+r) \times n + c$ and the theoretical latency is $r \times n + c$.

We always set the center frame as $c = 1$, and vary the number of left and right frames, *i.e.*, l and r. We first test the symmetric windows: $l = r = 1, 2, \cdots, 64$. Additionally, we aim to measure the usefulness of the past or future frames. As such, we compare past-only ($[l = 1, 2, \cdots, 128, c = 1, r = 0]$) and future-only ($[l = 0, c = 1, r = 1, 2, \cdots, 128]$) windows.

Results. The results in Figure 2 align with expectations, showing that performance improves as the attention window size increases. However, the magnitude of improvement diminishes with progressively larger window sizes. It suggests that DSUs capture contextual information from surrounding frames [15]. Additionally, even with the same window size, models that consider only past or future frames perform significantly worse than those that incorporate both. In the case of test-clean, future frames contribute slightly more to performance than past frames, whereas test-other and test-1h show little to no difference, supporting the effectiveness of a symmetric window approach. The results suggest there is little distinction between using past and future information. Further, we explore additional techniques to improve performance in Section 5.

4. Reducing the Number of Layers

Why does the number of layers matter? Existing S3Ms typically consist of a large number of layers. Since the number of layers has a linear relationship with computational cost, reducing them benefits resource-constrained on-device applications.

Also, through this experiment, we aim to estimate the amount of compute required to achieve a certain downstream performance. As it is empirically known that neighboring layers tend to contain similar amounts of information [13–15], fine-tuning S3Ms while removing the final layer one by one is the most straightforward approach.
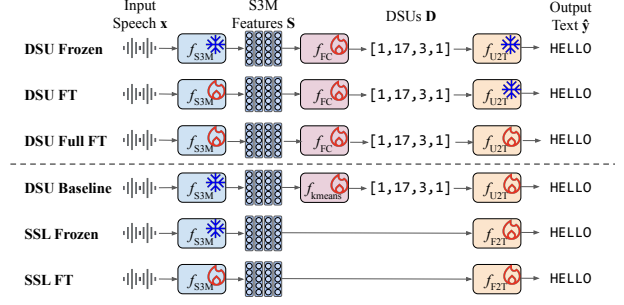

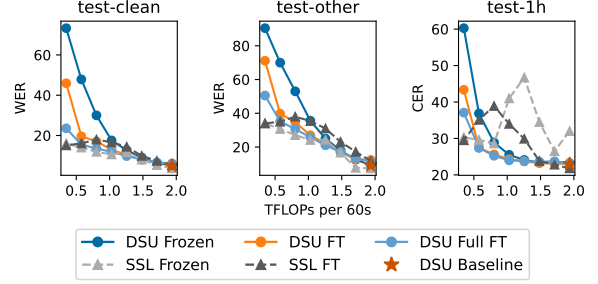Figure 3: *Various baselines for Section 4 and Figure 4.*

Figure 4: *Results using features from various layers when extracting DSUs. Each methods are summarized in Figure 3.*

Experimental settings. To answer the question, we designed a simple experimental setting of reducing Transformer layers one by one and extracting DSUs from them. For f_{S3M} (eq. (1)), we use layers up to layer index 21, 18, $\cdots$, 3, 0, where 0 means using only convolutional features. For the DSU predictor f_{S2U} (eq. (7)), we compare the case where we only fine-tune the FC layer f_{FC} (DSU Frozen) and also fine-tuning the S3M layers f_{S3M} (DSU FT). We use the frozen U2T module f_{U2T} from the challenge baseline. We additionally compare the case where we also fine-tune the U2T module f_{U2T} (DSU Full FT).

We compare with three baselines, where it is summarized in Figure 3: (1) the original challenge baseline (eq. (2), DSU Baseline); (2) using the frozen S3M features directly (SSL Frozen); and (3) also fine-tuning the S3M features (SSL FT). SSL Frozen is the strong baseline for DSU Frozen, which contains the full information inside the frozen n-th layer features. Similarly, SSL FT is the strong baseline for DSU FT, which demonstrates the case when we fully utilize the available model weights.

For SSL FT and SSL Frozen, we follow the settings of [5], which slightly modifies the U2T module f_{U2T}. We feed S3M features to the linear layer and feed through the Transformer directly. We denote the module f_{F2T}, *i.e.*, feature-to-text module. We use the same training settings as Section 3 except for the learning rate scheduler for SSL Frozen and SSL FT; a warmup of 15k was empirically necessary for convergence.

Results. Similar to Section 3, an expected trend is observed in Figure 4: reducing the number of layers leads to performance degradation and fine-tuning more modules leads to better performance. In the lowest layers, SSL features generally perform well, suggesting that relevant information is present but largely lost during the DSU prediction. Nonetheless, DSU-based methods remain generally on par with SSL approaches. In particular, SSL Frozen and SSL FT tend to overfit more easily to the majority language (test-clean and test-other), resulting in degraded performance on other languages (test-1h).
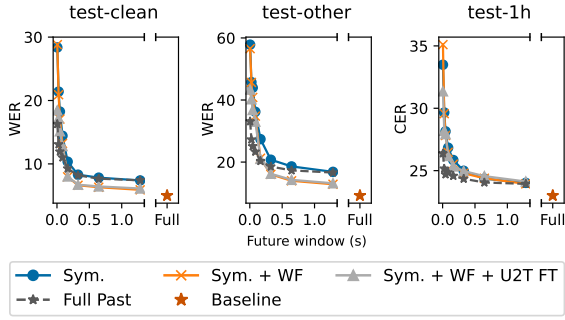
Figure 5: *Results with varying symmetric attention window sizes (Sym.), with applying additional methods, i.e., learnable weights (WF) and fine-tuning U2T module (U2T FT). Baseline uses the full speech input, while Full Past uses full past, but limited future attention window.*

5. Towards the Pareto Optimal

We test various methods to improve downstream performance. Also, by applying such, we produce a Pareto optimal curve that represents the trade-off between the computational overhead and the downstream performance.

Learnable weights for multi-layer features (WF). As demonstrated in [13, 14], different layers encode different types of information. To leverage this, prior work [3, 21, 28] has employed a learnable weighted summation of features across layers, removing the need to search for the optimal layer. Motivated by this approach, we apply the same technique to extract S3M features for DSU prediction.

Fine-tuning unit-to-text module (U2T FT). We hypothesize that DSU predictors with smaller window (eq. (7)) will produce noisier DSU predictions, where the U2T module (eq. (4)) has not been exposed to during training. To address this, we fine-tune the U2T module to enhance its robustness against such noise, resulting in better performance in downstream tasks.

Experimental settings. For WF, we use the exact same training settings as before. For U2T FT, we use the same settings except for the variable batch size of 20K and patience 3. We choose two baselines: (1) the original challenge baseline; (2) limiting only the future window size r, *i.e.*, $[l = \infty, c = 1, r = 1, 2, \cdots, 64]$ (Full Past). We show all methods' future attention window size for extracting DSUs. The theoretical latency, which is proportional to the future window size, is often used in streaming scenarios [29]. Note that symmetric windows' computational cost scale linearly with speech length, whereas both baselines scale quadratically.

Results. Figure 5 demonstrates that Full Past outperforms Symmetric for smaller windows. However, the difference diminishes beyond 0.5s window size, suggesting a fixed past window with an adjustable future window for latency needs. For WF, it reduces the largest windows' performance drop from 57% and 83% to 18% and 40% in test-clean and test-other. However, it shows limited improvement on ML-SUPERB. On the other hand, U2T FT is effective, especially for smaller windows across all datasets, supporting the above hypothesis that fine-tuning may learn noise in smaller window predictions.

Pareto optimal curve. By integrating all findings and techniques, we construct the Pareto optimal curve in Figure 1. We use symmetric windows of varying sizes ($l = r = 1, 2, \cdots, 64, c = 1$) while varying the number of layers (12, 15, 18, 21). We also apply learnable weights and test both with and without U2T fine-tuning.

Our Pareto curve is shown in Figure 1, demonstrating that one can extract DSUs in a lightweight streaming way with reasonable trade-off on ASR performance. Especially with 0.96 TFLOPs, we achieve 6.3 WER (test-clean), 14.0 WER (test-other), and 24.4 CER (test-1h), close to the baseline performance of 5.0 WER, 9.2 WER, and 23.0 CER.

6. Related Works

Making S3Ms lightweight. Knowledge distillation (KD) is often used to reduce the size of S3Ms, such as DistilHuBERT [30] or FitHuBERT [31]. Techniques like pruning [18, 19] and quantization [32] is also used. These approaches, evaluated on general benchmarks [21], primarily aim to create generally usable S3Ms with lower computational cost, often maintaining the non-streamable Transformer architecture. Our work focuses on efficient streaming DSUs, fine-tuning the model for a specific task and data, rather than yielding general-purpose features.

Speech tokenization. Speech tokenization broadly falls into acoustic tokens [33–35] or DSUs based on S3Ms [2, 3, 17]. Acoustic tokenizers are often trained with the Vector Quantised-Variational AutoEncoder (VQ-VAE) [36]), which emphasizes real-time, high-fidelity acoustic detail. On the other hand, DSUs are known to contain rich phonetic information [13–16] and hence provide structural coherence [7] and better ASR performance [5], although often offline. Rather than relying on VQ-VAE, our work aims to make DSUs directly streamable and lightweight, maintaining their rich knowledge while enabling real-time abilities.

DSUs for KD. [37] uses DSUs to distill S3Ms, but not specifically optimizing for predicting DSUs. DSUs have also been used for KD in voice conversion [38] or to guide acoustic tokenizers such as Speechtokenizer [39] and Mimi [40]. However, these approaches either prioritize building general speech models or introduce complex multi-stage systems. In contrast, our work directly concentrates on enhancing the efficiency of the DSU themselves, specifically for on-device streaming cases.

7. Conclusion

DSUs offer advantages for on-device streaming due to their transmission efficiency and compatibility with LLMs. However, current methods for generating DSUs rely on full speech input and computationally heavy S3Ms. Therefore, we investigated both the attention window size and the number of layers in S3Ms. Our experiments demonstrate the feasibility of creating lightweight and streamable DSUs. Furthermore, we show that weighted feature summation and fine-tuning the unit-to-text model effectively improve performance. Finally, we explored the trade-off between ASR performance and computational overhead, establishing Pareto optimal curve.

8. Acknowledgements

9. References

[1] A. Mohamed, H.-y. Lee, L. Borgholt *et al.*, "Self-supervised speech representation learning: A review," *IEEE J-STSP*, 2022.

[2] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," in *Proc. NeurIPS*, 2020.

[3] S. Chen, C. Wang, Z. Chen *et al.*, "WavLM: Large-scale self-supervised pre-training for full stack speech processing," *IEEE J-STSP*, 2022.

[4] X. Chang, B. Yan, Y. Fujita *et al.*, "Exploration of efficient end-to-end asr using discretized input from self-supervised learning," in *Proc. Interspeech*, 2023.

[5] X. Chang, B. Yan, K. Choi *et al.*, "Exploring speech recognition, translation, and understanding with discrete speech units: A comparative study," in *Proc. ICASSP*, 2024.

[6] X. Chang, J. Shi, J. Tian *et al.*, "The Interspeech 2024 Challenge on Speech Processing Using Discrete Units," in *Proc. Interspeech*, 2024.

[7] Z. Borsos, R. Marinier, D. Vincent *et al.*, "AudioLM: a language modeling approach to audio generation," *IEEE/ACM TASLP*, 2023.

[8] D. Zhang, S. Li, X. Zhang *et al.*, "SpeechGPT: Empowering large language models with intrinsic cross-modal conversational abilities," in *Proc. EMNLP*, 2023.

[9] P. K. Rubenstein, C. Asawaroengchai, D. D. Nguyen *et al.*, "AudioPaLM: A large language model that can speak and listen," *arXiv preprint arXiv:2306.12925*, 2023.

[10] T. Nakamura, K. Choi, K. Hojo *et al.*, "Discrete speech unit extraction via independent component analysis," in *Proc. ICASSP Workshop on Speech and Audio Language Models (SALMA)*, 2024.

[11] J. Zhan, J. Dai, J. Ye *et al.*, "AnyGPT: Unified multimodal LLM with discrete sequence modeling," in *Proc. ACL*, 2024.

[12] A. Hernandez, P. A. Perez-Toro, T. Arias-Vergara *et al.*, "Anonymizing dysarthric speech: Investigating the effects of voice conversion on pathological information preservation," in *Intl. Conf. on Text, Speech, and Dialogue*. Springer, 2024.

[13] K. Choi, A. Pasad, T. Nakamura *et al.*, "Self-supervised speech representations are more phonetic than semantic," in *Proc. Interspeech*, 2024.

[14] A. Pasad, B. Shi, and K. Livescu, "Comparative layer-wise analysis of self-supervised speech models," in *Proc. ICASSP*, 2023.

[15] K. Choi, E. Yeo, K. Chang *et al.*, "Leveraging Allophony in Self-Supervised Speech Models for Atypical Pronunciation Assessment," in *Proc. NAACL*, 2025.

[16] K. Choi, J.-w. Jung, and S. Watanabe, "Understanding probe behaviors through variational bounds of mutual information," in *Proc. ICASSP*, 2024.

[17] W.-N. Hsu, B. Bolte, Y.-H. H. Tsai *et al.*, "HuBERT: Self-supervised speech representation learning by masked prediction of hidden units," *IEEE/ACM TASLP*, 2021.

[18] C.-I. J. Lai, Y. Zhang, A. H. Liu *et al.*, "PARP: Prune, adjust and re-prune for self-supervised speech recognition," *Proc. NeurIPS*, 2021.

[19] Y. Peng, K. Kim, F. Wu *et al.*, "Structured pruning of self-supervised pre-trained models for speech recognition and understanding," in *Proc. ICASSP*, 2023.

[20] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Proc. ICASSP*, 2015.

[21] J. Shi, D. Berrebbi, W. Chen *et al.*, "ML-SUPERB: Multilingual Speech Universal PERformance Benchmark," in *Proc. Interspeech*, 2023.

[22] X. Ye, "calflops: a FLOPs and params calculate tool for neural networks," https://github.com/MrYxJ/calculate-flops.pytorch, 2023, accessed: 2025-05-20.

[23] K. Kim, F. Wu, Y. Peng *et al.*, "E-branchformer: Branchformer with enhanced merging for speech recognition," in *Proc. SLT*, 2022.

[24] A. Waswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," in *Proc. NeurIPS*, 2017.

[25] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. ICLR*, 2019.

[26] D. Povey, H. Hadian, P. Ghahremani *et al.*, "A time-restricted self-attention layer for asr," in *Proc. ICASSP*, 2018.

[27] N. Moritz, T. Hori, and J. Le, "Streaming automatic speech recognition with the transformer model," in *Proc. ICASSP*, 2020.

[28] S. wen Yang, P.-H. Chi, Y.-S. Chuang *et al.*, "SUPERB: Speech Processing Universal PERformance Benchmark," in *Proc. Interspeech*, 2021.

[29] B. Fu, K. Fan, M. Liao *et al.*, "wav2vec-S: Adapting pre-trained speech models for streaming," in *Proc. ACL*, 2024.

[30] H.-J. Chang, S.-w. Yang, and H.-y. Lee, "DistilHuBERT: Speech Representation Learning by Layer-wise Distillation of Hidden-unit BERT," in *Proc. ICASSP*, 2022.

[31] Y. Lee, K. Jang, J. Goo *et al.*, "FitHuBERT: Going thinner and deeper for knowledge distillation of speech self-supervised learning," in *Proc. Interspeech*, 2022.

[32] M. Oujabour, L. B. Letaifa, J.-F. Dollinger, and J.-L. Rouas, "Adaptive compression of supervised and self-supervised models for green speech recognition," in *Proc. ICASSP*, 2025.

[33] N. Zeghidour, A. Luebs, A. Omran *et al.*, "Soundstream: An end-to-end neural audio codec," *IEEE/ACM TASLP*, 2021.

[34] A. Défossez, J. Copet, G. Synnaeve, and Y. Adi, "High fidelity neural audio compression," *TMLR*, 2023.

[35] J. Shi, J. Tian, Y. Wu *et al.*, "ESPnet-Codec: Comprehensive training and evaluation of neural codecs for audio, music, and speech," in *Proc. SLT*, 2024.

[36] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Proc. NeurIPS*, vol. 30, 2017.

[37] D. de Oliveira and T. Gerkmann, "Distilling hubert with lstms via decoupled knowledge distillation," in *Proc. ICASSP*, 2024.

[38] H. Kanagawa and Y. Ijima, "Knowledge distillation from self-supervised representation learning model with discrete speech units for any-to-any streaming voice conversion," in *Proc. Interspeech*, 2024.

[39] X. Zhang, D. Zhang, S. Li *et al.*, "Speechtokenizer: Unified speech tokenizer for speech language models," in *Proc. ICLR*, 2024.

[40] A. Défossez, L. Mazaré, M. Orsini *et al.*, "Moshi: a speech-text foundation model for real-time dialogue," *arXiv preprint arXiv:2410.00037*, 2024.

10. Appendix

Table 1: *We provide the exact values of Figure 2. We denote various attention window configurations as the number of left, center, and right frames, i.e., $[l, c = 1, r]$. As mentioned in Section 2.4, our baseline differs from [6] only by the number of beam size, where we reduce from 20 to 5. As mentioned in Section 2.3, we use WER for test-clean and test-other, and CER for test-1h.*

Method	l	r	test-clean	test-other	test-1h	TFLOPs
Baseline [6]	∞	∞	5.0	9.1	22.9	1.936
Baseline (Ours)	∞	∞	5.0	9.2	23.0	1.936
Symmetric	0	0	28.37	57.84	33.49	1.407
	1	1	21.40	45.76	29.66	1.407
	2	2	18.25	43.96	28.15	1.408
	4	4	14.43	36.21	26.84	1.408
	8	8	10.35	27.39	25.85	1.410
	16	16	8.30	20.76	24.99	1.412
	32	32	7.83	18.60	24.36	1.418
	64	64	7.39	16.85	24.02	1.429
Past-only	1	0	24.59	51.42	31.91	1.407
	2	0	23.39	49.00	30.58	1.407
	4	0	21.70	49.24	29.80	1.408
	8	0	20.02	45.73	28.75	1.408
	16	0	18.86	41.03	28.10	1.410
	32	0	17.54	37.82	27.44	1.412
	64	0	16.88	35.04	26.89	1.418
	128	0	17.15	34.79	26.68	1.429
Future-only	0	1	24.20	50.39	31.95	1.407
	0	2	22.23	47.84	30.73	1.407
	0	4	20.73	45.08	29.65	1.408
	0	8	19.16	44.09	29.09	1.408
	0	16	17.45	39.29	28.08	1.410
	0	32	15.88	36.72	27.46	1.412
	0	64	15.10	33.99	26.89	1.418
	0	128	14.69	32.77	26.55	1.429

Table 2: *We provide the exact values of Figure 4. Layers denote the number of layers used. As mentioned in Section 2.3, we use WER for test-clean and test-other, and CER for test-1h. For SSL Frozen and SSL FT, we modified the configuration of the existing baseline.*

Method	Layer	test-clean	test-other	test-1h	TFLOPs
Baseline	21	5.0	9.2	23.0	1.936
DSU Frozen	21	5.11	9.41	23.08	1.936
	18	6.30	12.51	23.23	1.709
	15	9.03	17.98	23.40	1.483
	12	12.67	24.99	24.09	1.256
	9	17.56	35.60	25.51	1.029
	6	29.99	53.02	28.89	0.802
	3	47.87	69.99	36.85	0.576
	0	73.29	90.42	60.26	0.349
DSU FT	21	6.11	12.35	23.26	1.936
	18	6.34	12.85	23.10	1.709
	15	8.00	16.88	23.21	1.483
	12	10.37	22.06	23.72	1.256
	9	12.96	26.99	24.31	1.029
	6	17.05	34.70	25.63	0.802
	3	19.60	39.80	27.48	0.576
	0	45.94	71.13	43.35	0.349
DSU Full FT	21	5.94	11.48	23.25	1.936
	18	6.65	13.40	23.72	1.709
	15	8.09	17.02	23.61	1.483
	12	9.77	21.05	23.73	1.256
	9	11.85	25.79	23.99	1.029
	6	13.56	30.68	25.25	0.802
	3	15.46	35.16	27.40	0.576
	0	23.58	50.51	37.08	0.349
SSL Frozen	21	5.3	11.8	21.8	1.936
	18	7.4	17.2	22.8	1.709
	15	9.9	23.1	23.8	1.483
	12	14.4	31.0	29.9	1.256
	9	16.6	35.4	34.0	1.029
	6	18.0	38.0	38.9	0.802
	3	16.1	35.0	35.1	0.576
	0	15.1	34.0	29.5	0.349
SSL FT	21	3.7	7.4	32.0	1.936
	18	5.3	7.7	26.5	1.709
	15	8.0	16.7	34.5	1.483
	12	11.7	24.3	46.7	1.256
	9	10.6	24.2	41.0	1.029
	6	11.9	27.1	28.6	0.802
	3	14.0	30.7	29.6	0.576
	0	16.1	34.0	30.3	0.349

Table 3: *We provide the exact values of Figure 5. We denote various attention window configurations as the number of left, center, and right frames, i.e., $[l, c = 1, r]$. Sym., WF, and U2T FT denotes symmetric attention window, learnable weights, and fine-tuning U2T module, respectively. Baseline uses the full speech input, while Full Past uses full past, but limited future attention window. As mentioned in Section 2.3, we use WER for test-clean and test-other, and CER for test-1h.*

Method	l	r	test-clean	test-other	test-1h	TFLOPs
Baseline	∞	∞	5.0	9.2	23.0	1.936
Sym.	0	0	28.37	57.84	33.49	1.407
	1	1	21.40	45.76	29.66	1.407
	2	2	18.25	43.96	28.15	1.408
	4	4	14.43	36.21	26.84	1.408
	8	8	10.35	27.39	25.85	1.410
	16	16	8.30	20.76	24.99	1.412
	32	32	7.83	18.60	24.36	1.418
	64	64	7.39	16.85	24.02	1.429
Sym.+WF	0	0	28.81	56.55	35.10	1.407
	1	1	20.91	45.66	29.57	1.407
	2	2	17.07	40.83	27.82	1.407
	4	4	13.23	34.81	26.47	1.408
	8	8	7.96	21.24	25.52	1.408
	16	16	6.63	16.04	24.79	1.410
	32	32	6.32	13.97	24.39	1.412
	64	64	5.91	12.86	23.94	1.418
Sym.+WF+U2T FT	0	0	18.45	43.36	31.36	1.407
	1	1	17.23	40.24	28.19	1.407
	2	2	15.15	36.79	27.93	1.407
	4	4	12.82	33.04	26.18	1.408
	8	8	8.02	21.03	25.34	1.408
	16	16	6.72	16.24	24.97	1.410
	32	32	6.45	14.28	24.56	1.412
	64	64	6.12	13.17	24.13	1.418
Full Past	∞	0	16.28	33.12	26.38	1.671
	∞	1	13.08	27.26	25.16	1.672
	∞	2	11.94	25.09	24.70	1.672
	∞	4	11.01	23.35	24.85	1.672
	∞	8	9.27	20.34	24.58	1.672
	∞	16	8.20	18.55	24.35	1.673
	∞	32	7.67	17.32	24.05	1.674
	∞	64	7.41	16.62	23.94	1.677

Table 4: *We provide the exact values for each individual points of Figure 1. We apply both symmetric window and learnable weights (Sym. + WF) while varying the window size and the number of layers. As mentioned in Section 2.3, we use WER for test-clean and test-other, and CER for test-1h.*

Layer	l	r	test-clean	test-other	test-1h	TFLOPs
21	0	0	28.81	56.55	35.10	1.407
	1	1	20.91	45.66	29.57	1.407
	2	2	17.07	40.83	27.82	1.407
	4	4	13.23	34.81	26.47	1.408
	8	8	7.96	21.24	25.52	1.408
	16	16	6.63	16.04	24.79	1.410
	32	32	6.32	13.97	24.39	1.412
	64	64	5.91	12.86	23.94	1.418
18	0	0	28.45	57.28	33.48	1.256
	1	1	22.37	48.00	29.66	1.256
	2	2	18.21	43.61	27.96	1.256
	4	4	14.36	35.94	26.66	1.256
	8	8	9.32	24.81	25.70	1.257
	16	16	8.09	19.52	25.09	1.258
	32	32	7.39	17.59	24.39	1.261
	64	64	6.92	15.71	24.10	1.265
15	0	0	28.63	57.35	33.54	1.105
	1	1	23.09	49.32	29.91	1.105
	2	2	18.52	43.81	28.17	1.105
	4	4	15.30	38.43	26.93	1.105
	8	8	11.90	29.92	25.91	1.106
	16	16	10.62	25.09	25.28	1.107
	32	32	9.90	22.18	24.52	1.109
	64	64	9.43	20.82	24.36	1.113
12	0	0	29.32	57.95	34.77	0.953
	1	1	23.82	48.84	30.77	0.954
	2	2	20.27	47.22	28.75	0.954
	4	4	16.55	40.36	27.24	0.954
	8	8	14.18	34.35	26.29	0.954
	16	16	13.04	30.58	25.49	0.955
	32	32	12.36	27.50	25.02	0.957
	64	64	11.92	26.21	24.52	0.960

Table 5: *We provide the exact values for each individual points of Figure 1. We apply all symmetric window, learnable weights, and fine-tuning U2T module (Sym. + WF + U2T FT) while varying the window size and the number of layers. As mentioned in Section 2.3, we use WER for test-clean and test-other, and CER for test-1h.*

Layer	l	r	test-clean	test-other	test-1h	TFLOPs
21	0	0	18.45	43.36	31.36	1.407
	1	1	17.23	40.24	28.19	1.407
	2	2	15.15	36.79	27.93	1.407
	4	4	12.82	33.04	26.18	1.408
	8	8	8.02	21.03	25.34	1.408
	16	16	6.72	16.24	24.97	1.410
	32	32	6.45	14.28	24.56	1.412
	64	64	6.12	13.17	24.13	1.418
18	0	0	19.34	44.96	31.33	1.256
	1	1	17.11	40.86	28.96	1.256
	2	2	15.18	37.84	27.37	1.256
	4	4	14.00	33.75	27.02	1.256
	8	8	9.37	24.83	25.76	1.257
	16	16	8.08	19.45	25.15	1.258
	32	32	7.76	17.85	25.02	1.261
	64	64	6.97	15.88	23.82	1.265
15	0	0	19.70	45.20	31.37	1.105
	1	1	17.48	40.70	29.00	1.105
	2	2	15.57	37.85	27.61	1.105
	4	4	14.17	34.81	26.47	1.105
	8	8	11.98	29.16	26.10	1.106
	16	16	10.21	24.65	25.35	1.107
	32	32	9.91	22.18	24.87	1.109
	64	64	9.60	20.58	25.04	1.113
12	0	0	19.01	44.19	32.05	0.953
	1	1	17.42	40.54	28.87	0.954
	2	2	15.69	38.70	27.64	0.954
	4	4	14.60	35.54	26.95	0.954
	8	8	13.54	32.64	26.58	0.954
	16	16	12.43	29.28	25.17	0.955
	32	32	11.79	27.19	25.05	0.957
	64	64	11.50	25.63	24.50	0.960