Personalized Predictive ASR for Latency Reduction in Voice Assistants

Andreas Schwarz[1], *Di He*[2], *Maarten Van Segbroeck*[2], *Mohammed Hethnawi*[1], *Ariya Rastrow*[2]

[1]Amazon Alexa, Germany
[2]Amazon Alexa, USA

{asw, deehe, segbrm, hethnm, arastrow}@amazon.com

Abstract

Streaming Automatic Speech Recognition (ASR) in voice assistants can utilize prefetching to partially hide the latency of response generation. Prefetching involves passing a preliminary ASR hypothesis to downstream systems in order to prefetch and cache a response. If the final ASR hypothesis after endpoint detection matches the preliminary one, the cached response can be delivered to the user, thus saving latency. In this paper, we extend this idea by introducing predictive automatic speech recognition, where we predict the full utterance from a partially observed utterance, and prefetch the response based on the predicted utterance. We introduce two personalization approaches and investigate the tradeoff between potential latency gains from successful predictions and the cost increase from failed predictions. We evaluate our methods on an internal voice assistant dataset as well as the public SLURP dataset.

Index Terms: Voice Assistants, ASR, Latency, Endpointing, Prefetching, Language Modeling, Personalization

1. Introduction

In voice assistants, a request is processed by multiple systems before the response is ready, starting with automatic speech recognition (ASR) and the interpretation of the ASR hypothesis, and ending with the transmission of the generated text to speech (TTS) response to the user, as well as potential execution of external effects such as turning on a smart home device. The steps involved in the generation of the response each contribute varying amounts of latency, adding up to the user-perceived latency (UPL). Besides accuracy, a design goal for voice assistants is minimization of UPL. One major contributor to UPL is algorithmic latency for utterance endpoint detection [1]. Only when the endpoint of the utterance has been detected with sufficient confidence, e.g., based on acoustic and ASR decoder features [2, 3], the final ASR hypothesis can be provided to downstream systems for result generation. Prefetching [4, 5] has been proposed as a way to reduce UPL by already propagating an initial ASR hypothesis before the final endpoint has been detected and caching the result. E.g., a second, lower threshold for endpoint detection may be applied to trigger speculative execution using the preliminary ASR hypothesis. If the recognition after the final endpoint confirms the preliminary hypothesis, the prefetched response can be returned to the user. This allows hiding a part of the downstream systems' latency within the period between the speculative and the final endpoint. A prerequisite is support for speculative execution in the downstream pipeline, e.g., the possibility of executing the response generation speculatively, while postponing any external effects to after the confirmation by the final endpoint.

In this paper, we propose to extend the concept of prefetching. Instead of using a preliminary endpoint detector to trigger prefetching, we use a prediction strategy that generates a complete utterance from a partially observed utterance while the user is still speaking. This promises a larger window for latency savings compared to endpoint-based prefetching. For queries that can be predicted early in the utterance, we could hide the entire downstream system latency between the time of the initial prediction and the final ASR result that matches the prediction. Ideally, the final response playback or intent execution could be carried out immediately after the final ASR result is available, assuming the prediction has been correct. Fig. 1 illustrates this concept: after the user has said "what is", we can predict that the utterance is going to end in "...the weather today", based on the user's frequent usage of this phrase or additional contextual information such as the time of day. We then use this predicted hypothesis to generate a response. As soon as the endpoint has been detected and the final ASR result confirms the prediction, the response can be returned to the user, thus saving latency. In case of a mismatch, the cached response has to be discarded and re-generated, with no effect on latency, but additional compute cost for response generation[1].

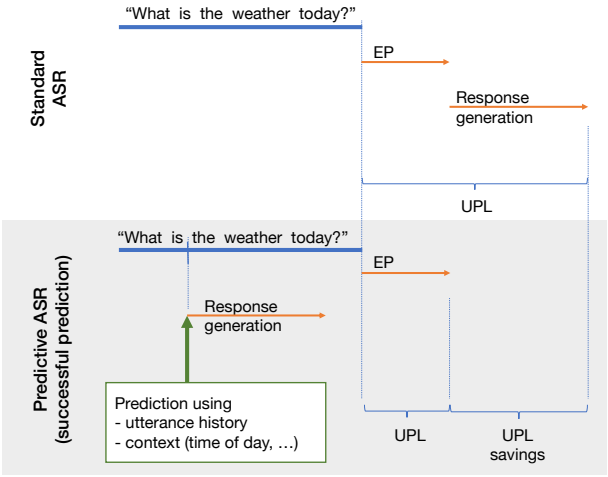

Figure 1: *Illustration of major latency contributions for a voice assistant, comparing standard and predictive ASR. User-perceived latency (UPL) is dominated by endpointing (EP) and response generation. Predictive ASR ideally allows hiding the response generation latency within the time between prediction and endpointing, assuming successful prediction.*

[1]Even in case of mismatch, e.g., the user instead saying "what is the weather today *in Seattle*", a latency benefit may remain due to downstream systems being able to re-use internal caches.

In Sec. 2, we describe our approach for predictive ASR. In Sec. 3, we present our experimental implementation and results.

2. Proposed approach

We consider a simplified voice assistant system architecture consisting of an utterance endpoint detector, a causal streaming ASR model, and a response generator. We make the simplifying assumption that the endpoint detector (EP) and the response generation make up the UPL of the system, neglecting ASR latency[2]:

$$T_{\text{UPL}} = T_{\text{EP}} + T_{\text{Response}}, \tag{1}$$

where T_{EP} is the time between end of the speech and the endpoint decision, and T_{Response} is the time that systems downstream from ASR require to generate a response and execute the intent. In the literature, prefetching has been proposed for executing response generation and endpoint detection in parallel [4], reducing UPL in case of success:

$$T_{\text{UPL, PF}} = \begin{cases} \max(T_{\text{EP}}, T_{\text{PF}} + T_{\text{Response}}) & \text{(successful prefetch)} \\ T_{\text{EP}} + T_{\text{Response}} & \text{(failed prefetch),} \end{cases} \tag{2}$$

where T_{PF} is the prefetching latency, which typically is a positive delay between end of speech and the point in time where we trigger prefetching[3]. We propose to extend prefetching to predict the full utterance transcription before the user has finished speaking the utterance, thus extending the latency saving opportunity. If a correct prediction of the utterance is available by ΔT before the end of speech in the spoken query, we have a negative prefetching latency, i.e., $T_{\text{PF}} = -\Delta T$. We call ΔT the prediction gain. Premature early prefetching can hinder the efficacy of a predictive ASR strategy. To address this, we propose to implement predictive ASR using combination of a prediction model and a policy that uses a prediction confidence model to accept or reject a predicted transcription. We describe these models in the following.

2.1. Prediction Model

We consider the task of predicting the full utterance token sequence, y_{full}, at time t as determining the most probable sequence given all observations available until t, x_t:

$$\hat{y}_{\text{full},t} = \underset{y_{\text{full}}}{\operatorname{argmax}}\, P(y_{\text{full}}|x_t) \tag{3}$$

Observations would typically consist of the partial utterance audio, but could also include previous interactions and any available contextual information, such as the time of day. For simplification we separate the modeling task into a causal ASR model which estimates the partial token sequence up to t, $\hat{y}_t$, and a prediction model which extrapolates this partial token sequence to the most probable complete token sequence:

$$\hat{y}_t = \underset{y}{\operatorname{argmax}}\, P(y|x_t) \tag{4}$$

$$\hat{y}_{\text{full},t} \approx \underset{y_{\text{full}}}{\operatorname{argmax}}\, P(y_{\text{full}}|\hat{y}_t) \tag{5}$$

This allows the use of a standard ASR model and decoder to obtain the partial utterance (4) and a standard generative language model (LM) for the prediction (5). E.g., prediction could re-use the model and partial computation results from an LM which is used for re-scoring [6, 7, 8], or leverage a general pre-trained language model such as Generative Pre-trained Transformer (GPT) [9].

2.2. Prediction Confidence Modeling

After predicting the full utterance text, a decision needs to be made on whether to propagate the result downstream for prefetching or not. The tradeoff to consider here involves the the probability of success, the cost of a failed speculation attempt, and the latency gain in case of success. While these factors could be modeled in a joint loss function, for now, we only model the prediction confidence, i.e., the probability of the predicted utterance $\hat{y}_{\text{full},t}$ matching the final ASR hypothesis $\hat{y}_{\text{full}}$:

$$P(\hat{y}_{\text{full},t} = \hat{y}_{\text{full}}). \tag{6}$$

We apply a threshold to this confidence estimate to decide on whether to act on a prediction, and assume we only act on one prediction per utterance.

In the simplest case, this confidence model could be replaced by the probability given by the prediction model. However, training a dedicated confidence model allows the use of additional features which have not been made available to the prediction model, such as the time since the start of the utterance, the confidence of the ASR model for the partial hypothesis, or additional personalized signals.

2.3. Personalization and Contextualization

Usage patterns of voice assistants are highly individual and dependent on the context of an interaction. E.g., a user may have the habit of asking for the weather at a certain time of the day, or just use the same consistent phrasing for common requests. Further, requests may depend on information such as the location of the device, entries in a user's playlist, or available smart home devices and their state. E.g., if the partial hypothesis is "turn living room", and the user has a device called "living room light" which is turned off, a completion with "...light on" is very likely. We therefore expect that it is beneficial for prediction accuracy to account for personalization and context. We may condition either the prediction model and/or the confidence model on personal and contextual information such as the recently spoken utterances of a user or the current time of day. While conditioning the prediction model itself is likely to be the most general and powerful approach, practical considerations make the use of personalization in the confidence model attractive as well. E.g., in case an existing, general LM is used for prediction, a confidence model could be built with contextual or personalized features and used to select one of several n-best predictions generated from the LM. In this paper, we implement personalization in two ways. First, we use the relative frequency of a prediction in a user's recent utterance history as a feature for the confidence model. Second, we augment the predictions from the LM with additional predictions obtained by prefix matching in the recent utterance history of a user.

3. Experiments

We conduct experiments on two English speech datasets. The first is an internal dataset consisting of de-identified user interactions with a voice assistant. This dataset contains four weeks of continuous interaction data from a number of users, with approx. 1700 users in the training partition, and 200 in the development and test partitions, respectively. We use the last three

[2] An in-depth study of ASR latency contributions can be found in [1].

[3] Note that some ASR models can have a negative partial latency, which would result in a negative delay (see Sec. 4).

weeks for evaluation, thus allowing us to evaluate the effect of personalized prediction and confidence modeling with at least one week of prior historical context for each utterance under test. The second dataset we experiment on is the SLURP dataset [10], which covers similar domains as our internal dataset, however, consists of artificially generated independent utterances and therefore cannot be used for personalization experiments.

3.1. Evaluating Prediction Performance

We distinguish three possible outcomes for an utterance. If the confidence model with a given threshold did not allow any prefetching, there is no impact on latency or cost (*no prefetch*). If prefetching was triggered with a prediction that contains at least one additional word over the partial ASR hypothesis, and the prediction turns out to match the final ASR hypothesis, we have a potential for a latency gain, without an impact on cost (*successful prefetch*). Finally, if we triggered prefetching with an incorrect prediction, there is no impact on latency, but an impact on cost for repeated response generation (*failed prefetch*). We assume at most one prefetching attempt per utterance.

We first evaluate the rate of successful and failed prefetches relative to the total number of utterances. The rate of successful prefetches corresponds to the fraction of utterances which benefit from prefetching. The rate of failed prefetches corresponds to the relative downstream cost increase due to prefetched responses which need to be discarded and re-generated. To quantify the potential latency gain from prefetching for an utterance, we evaluate the prediction gain ΔT as the time between the availability of the prediction and the end of speech of an utterance. In other words, this prediction gain corresponds to the extension of the prefetching window achievable over a hypothetical ASR system with perfect endpointing or perfect causal endpoint-based prefetching.

3.2. System Implementation

3.2.1. ASR Decoding

We use an RNN-T ASR model [11, 12] with an 8×1280 long short-term memory (LSTM) encoder, a 2×1280 LSTM prediction network, a single-layer joint network, and a total of 148M trainable parameters. The model uses 4k word pieces and is trained on an internal voice assistant dataset. In order to generate partial ASR hypotheses, we trigger result generation in fixed intervals of 120 ms. The final ASR result is obtained by decoding the utterance audio after endpoint detection. We note that, while the utterance audio in our dataset contains trailing silence, we compute the prediction gain ΔT of an utterance as the interval between the last frame used for partial decoding, and the last frame containing speech according to a phonetic alignment of the utterance.

3.2.2. Prediction

Our prediction model is a word-level 2-layer LSTM LM [13, 14]. It is trained to predict the next token given the previous tokens on a mix of voice assistant utterances and out-of-domain datasets. We use the same type of LM in the ASR second-pass-rescoring stage [6, 7, 8, 15] and could therefore theoretically reuse both the model and partially the computations. The model has a the vocabulary size of 283k and 149M trainable parameters (4M in the LSTM, the remainder in the input embedding). For evaluation on the SLURP dataset, we train a second model by fine-tuning the first model on data from the SLURP training partition. The perplexity of the models is 15 for the internal dataset test partition, and 31 for the SLURP dataset test partition, respectively (we attribute the difference in perplexity to the fact that the SLURP dataset contains artificial sentences which were generated in written form for research purposes). Prediction is implemented using beam search to generate a set of N_{LM} candidate predictions which complete the partial input token sequence until the end-of-sentence symbol. We here chose $N_{\mathrm{LM}} = 4$ as a tradeoff between diversity in prediction candidates and computational effort.

In addition to the prediction from a neural language model, we generate a set of personalized predictions from the past recognized utterances of a user prior to the current utterance (up to 4 weeks of personal usage history). We do this by selecting all previous recognitions which match the prefix of the current partial hypothesis, and combining them with the set of non-personalized predictions obtained from the language model.

3.2.3. Prediction Confidence Modeling

We train an ensemble neural classifier (up to 4 dense layers with width 512) to classify whether a prediction matches the final utterance. We train this classifier on predictions from the training partition of the respective dataset and use the development partition for model selection and early stopping. For each utterance, all predictions generated from all partial ASR hypotheses are included in the training dataset (excluding predictions with zero predicted tokens), resulting in a training dataset size of 5.5M for our internal dataset and 1.2M for the SLURP dataset.

Our baseline model uses as features the log-probability of the prediction conditioned on the partial hypothesis, and the rank of the prediction in the n-best list, as obtained from the LM. We also use simple text features (number of words and characters in the partial hypothesis and prediction) and the time of the prediction relative to the start of the utterance. Note these features have a minor impact compared to the LM features.

We implement a confidence model which accounts for personalization by adding a personalized feature, which is the log-frequency of the prediction relative to other possible completions of the given prefix from a user's history, with a fallback to -10.0 for utterances not seen in the history.

3.3. Results and Discussion

Fig. 2 shows the tradeoff between successful and failed prefetch rates on our internal dataset (test partition) for varying acceptance thresholds. We also show oracle results, where we accept the first prediction which matches the final recognition; this sets an upper bound of 57% predictable utterances with our given prediction model. Plot labels display the prediction gain ΔT after averaging over all successfully predicted utterances. Fig. 3 additionally shows the prediction gain averaged over all utterances (with fallback to 0 for utterances with no or failed prefetching attempt), thus reflecting the average potential latency gain while also taking the prefetching rate into account.

We first note that the relationship between successful and failed prefetches is not monotonic, but successful prefetch rate starts decreasing at some point as the acceptance threshold becomes more permissive. This is due to our chosen limitation to at most one prefetch per utterance, which means we waste the potential for prefetching at a more promising point in time if we decide to prefetch too early. On the other hand, we see that more permissive operating points, corresponding to earlier prefetching, lead to a higher latency gain in case of success.

At a maximum, by using a combination of personalized and LM predictions, as well as a confidence model which includes

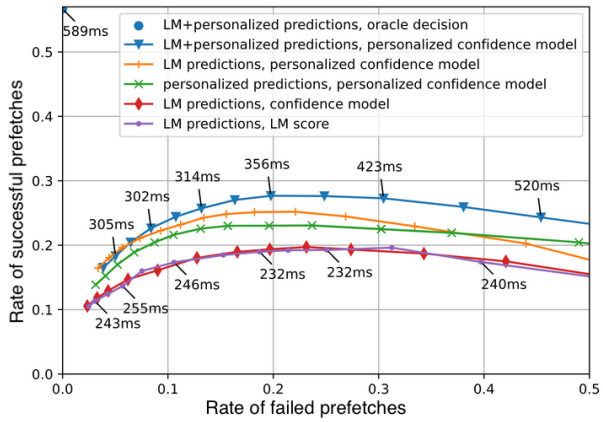

Figure 2: *Rate of successful vs. failed prefetches on the internal voice assistant dataset. Text labels show the prediction gain (time between prediction and end of speech) averaged over the successful prefetches.*

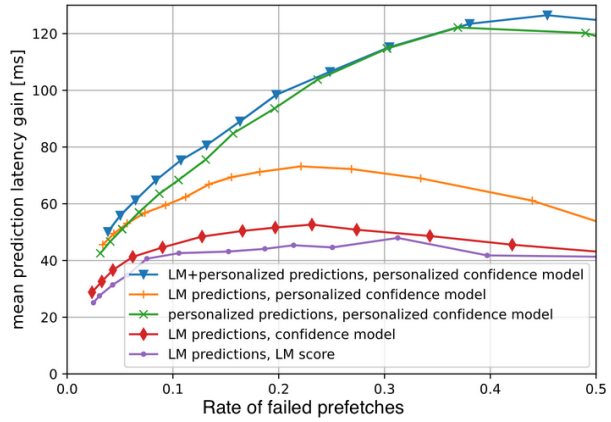

Figure 3: *Mean prediction gain over all utterances vs. rate of failed prefetches (internal voice assistant dataset). Rate of failed prefetches corresponds to downstream cost increase.*

personalized features, we can achieve 28% correctly predicted utterances, at a cost of a 20% downstream execution overhead (due to failed prefetches). The average prediction gain for successful prefetches at this operating point is 356 ms, or 1.7 words. Using the same confidence model, but relying merely on LM predictions or only personalized predictions reduces the success rate. This indicates that the combination of global and personalized modeling is critical for successful prediction. Finally, using LM predictions without any personalized features in the confidence model causes a more significant drop in performance, regardless of whether we use a confidence model or the LM score directly as decision criterion for prefetching.

Fig. 4 shows corresponding results on the SLURP dataset (test partition). For this dataset, we did not estimate the speech endpoint for each utterance, instead we only evaluate prefetching success and failure rates, as well as the average number of predicted words for successful prefetches, which we show in labels. We note that there is no consistent improvement from the use of a confidence model, possibly due to mismatch between the training and test partition distributions, or due to the fact that the confidence model is trained to classify each prediction separately, which is slightly mismatched with our evaluation metric. Overall, prediction accuracy on SLURP is behind our internal dataset, reflecting the higher perplexity which we also observed in the language model. Due to the lack of a per-user history, this dataset also cannot benefit from personalization.

4. Relation to Prior Work

In [4, 5], prefetching for ASR is described as the process of triggering an early ASR hypothesis generation based on an estimate of the end-of-speech probability. This early hypothesis is used to generate results which are then confirmed or discarded based on the final ASR hypothesis. The authors of [5] also present observations that streaming ASR models are capable of producing partial hypotheses with a negative latency, corresponsing to a prefetch window extension of up to 50ms. Further increase in negative latency could be achieved by applying regularization during model training (FastEmit), although at the cost of degrading word error rate (WER) [16]. One major difference in our work is that we use a dedicated prediction model. We therefore do not need to constrain the ASR model training in ways that potentially affect accuracy. Also, prediction using a separate language model allows us to predict up to multiple word tokens of the user utterance. A second difference is that we do not trigger prefetching based on an end-of-utterance probability, but by an estimate of the probability that a prediction matches the final ASR hypothesis.

5. Conclusion

We proposed a predictive ASR system for voice assistants which prefetches downstream results based on predictions of the full utterance from a partially spoken utterance. The proposed system allows obtaining the correct complete hypothesis on average 300 ms before the end of speech (i.e., up to 300 ms latency reduction) for 23% of the utterances in a voice assistant dataset, while incurring only an 8% increase in downstream cost due to failed prefetching attempts. The maximum successful prefetch rate of the current system of 28% might be further increased by lifting the limitation to a single prefetch attempt per utterance, instead allowing multiple parallel or sequential prefetches (incurring higher downstream cost). We further found personalization of predictions to be a critical factor, and expect that an even larger effect could be achieved by conditioning the prediction model itself on the usage history, as well as exploiting additional contextual features such as the time of day or approximate device location.

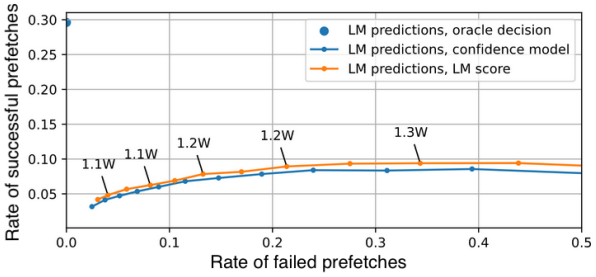

Figure 4: *Rate of successful vs. failed prefetches on the SLURP dataset. Text labels show the average number of predicted words for successful prefetches.*

6. References

[1] Y. Shangguan, R. Prabhavalkar, H. Su, J. Mahadeokar, Y. Shi, J. Zhou, C. Wu, D. Le, O. Kalinli, C. Fuegen, and M. Seltzer, "Dissecting user-perceived latency of on-device e2e speech recognition," in *Proc. Interspeech*, 2021.

[2] R. Maas, A. Rastrow, C. Ma, G. Lan, K. Goehner, G. Tiwari, S. Joseph, and B. Hoffmeister, "Combining acoustic embeddings and decoding features for end-of-utterance detection in real-time far-field speech recognition systems," in *Proc. ICASSP*, 2018.

[3] S. yiin Chang, R. Prabhavalkar, Y. R. He, T. Sainath, and G. Simko, "Joint endpointing and decoding with end-to-end models," in *Proc. ICASSP*, 2019.

[4] S. yiin Chang, B. Li, D. J. Rybach, W. Li, Y. R. He, T. N. Sainath, and T. D. Strohman, "Low latency speech recognition using end-to-end prefetching," in *Proc. Interspeech*, 2020.

[5] B. Li, A. Gulati, J. Yu, T. N. Sainath, C.-C. Chiu, A. Narayanan, S.-Y. Chang, R. Pang, Y. He, J. Qin, W. Han, Q. Liang, Y. Zhang, T. Strohman, and Y. Wu, "A better and faster end-to-end model for streaming ASR," in *Proc. ICASSP*, 2021.

[6] A. Deoras, T. Mikolov, and K. Church, "A fast re-scoring strategy to capture long-distance dependencies," in *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[7] S. Kumar, M. Nirschl, D. Holtmann-Rice, H. Liao, A. T. Suresh, and F. Yu, "Lattice rescoring strategies for long short term memory language models in speech recognition," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2017.

[8] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The Microsoft 2017 conversational speech recognition system," in *Proc. ICASSP*, 2018.

[9] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," *OpenAI*, 2018.

[10] E. Bastianelli, A. Vanzo, P. Swietojanski, and V. Rieser, "SLURP: A Spoken Language Understanding Resource Package," in *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[11] A. Graves, "Sequence transduction with recurrent neural networks," in *Proc. Representation Learning Workshop on International Conference on Machine Learning (ICML)*, 2012.

[12] A. Schwarz, I. Sklyar, and S. Wiesler, "Improving RNN-T ASR Accuracy Using Context Audio," in *Proc. Interspeech*, 2021.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Proc. Interspeech*, 2012.

[15] A. Raju, D. Filimonov, G. Tiwari, G. Lan, and A. Rastrow, "Scalable multi corpora neural language models for ASR," in *Proc. Interspeech*, 2019.

[16] J. Yu, C.-C. Chiu, B. Li, S. yiin Chang, T. N. Sainath, Y. R. He, A. Narayanan, W. Han, A. Gulati, Y. Wu, and R. Pang, "Fastemit: Low-latency streaming asr with sequence-level emission regularization," in *Proc. ICASSP*, 2021.