# TINYS2I: A SMALL-FOOTPRINT UTTERANCE CLASSIFICATION MODEL WITH CONTEXTUAL SUPPORT FOR ON-DEVICE SLU

*Anastasios Alexandridis* *Kanthashree Mysore Sathyendra* *Grant P. Strimel*
*Pavel Kveton* *Jon Webb* *Athanasios Mouchtaris*

Alexa Maching Learning, Amazon.com, USA
{aanast, ksathyen, gsstrime, kvetonp, webbajon, mouchta}@amazon.com

## ABSTRACT

On-device spoken language understanding (SLU) offers the potential for significant latency savings compared to cloud-based processing, as the audio stream does not need to be transmitted to a server. We present Tiny Signal-to-interpretation (TinyS2I), an end-to-end on-device SLU approach which is focused on heavily resource constrained devices. TinyS2I brings latency reduction without accuracy degradation, by exploiting use cases when the distribution of utterances that users speak to a device is largely heavy-tailed. The model is tailored to process on-device frequent utterances with support for dynamic contextual content, while deferring all other requests to the cloud. Compared to a powerful baseline, we demonstrate that TinyS2I achieves comparable performance, while offering latency gains due to local processing.

***Index Terms***— spoken language understanding, latency reduction, automatic speech recognition, on-device

## 1. INTRODUCTION

Traditionally, voice assistants, such as Siri, Google Assistant, and Alexa, perform SLU on the cloud by transmitting the audio stream to servers. SLU consists of two major tasks: an Automatic Speech Recognition (ASR) model that converts speech to a transcript and a Natural Language Understanding (NLU) model that converts the transcript into a machine interpretation. Recently, the adoption of voice assistants in more devices, such as smart speakers, smart phones, and tablets has motivated the development of on-device SLU systems. On-device SLU comes with the benefit of significant latency reduction and privacy, as the audio stream need not leave the users' devices.

Several approaches exist in the literature on how to design local SLU systems. For ASR, the current trend is to replace traditional Hidden Markov Model (HMM)-based systems with end-to-end approaches [1–5]. End-to-end approaches replace the disjoint acoustic, lexicon, and language models with one fully neural architecture that is more easily compressible. However, when deployed to devices with compute and memory constraints, they still require techniques to achieve low-latency speech recognition. Several techniques have been proposed, especially for the Recurrent Neural Network Transducer (RNN-T) architecture [1] including quantizing the model weights from 32-bits to 8 or 4 bits [6–8], network sparsification [9, 10] or low rank matrix factorization [9]. Additionally, other RNN architectures, such as those based on Simple Recurrent Units (SRU) [11] and CIFG-LSTM [12] have been proposed to replace recurrent layers for on-device ASR. Other works reduce inference time for RNN-T by introducing bifocal and amortized neural architectures [13, 14]. Similarly, for NLU, significant efforts have been made to compress large models [15, 16]. The latest trend is to move to the paradigm of signal-to-interpretation (S2I) [17–23] where the ASR and NLU are fused together into a single all-neural model. This enables parameter sharing and allows to build smaller and simpler end-to-end models without performance degradation.

However, the size reduction offered by these approaches is not sufficient for devices whose memory and compute resources are so limited that they call for SLU models that are only a few megabytes (MB) in size. Furthermore, size reductions come at the cost of lower accuracy, especially on tail-utterances. An alternative approach is to leverage the specific distribution of utterances spoken to the device, and design small-footprint, low-latency SLU models that process a small targeted set of high-frequency utterances on-device. All other utterances are detected as "unsupported" by the model and sent to the cloud for processing. This design not only offers significant latency reductions for high frequency utterances, but also maintains the accuracy of the overall system. Using such a system, users will benefit from latency reductions in the most commonly used utterances, while coverage for all utterances is guaranteed by the cloud model. There are several cases of devices whose distribution is largely heavy-tailed towards a small set of utterances, motivating the design of such a system: users of smart headphones largely use their headphones to play different songs, pause a song, or move to the next/previous song. Users of smart TVs largely use their voice to start/stop a movie, pause, change the volume, etc.

We present a system which we call TinyS2I. TinyS2I is a very low footprint SLU system that works as an utterance classifier for a finite set of frequent utterances, which we call "supported utterances". The model classifies each supported utterance to a separate class or identifies an utterance as unsupported and defers processing to the cloud. TinyS2I is able to perform full SLU (thus being an S2I model), since once the utterance is identified there is a one-to-one correspondence between the utterance and its NLU interpretation, thus fully replacing both the ASR and NLU components. A key feature of TinyS2I is that it employs a contextual model to identify dynamic content that appears in a catalog. This model is trained using a multi-modal neural metric learning approach, allowing to incorporate dynamic content such as video or song names.

We applied TinyS2I in a voice-enabled smart TV application for the English language. For our use-case, TinyS2I supports 132 command and control utterances, such as "play", "stop", "volume up/down", "switch to hdmi", "open netflix/hulu", etc. It also supports the contextual utterance "play *VideoName*" where *VideoName* is a movie name that appears on the TV screen. These constitute the supported utterances and all other utterances are identified as unsupported and are sent to the cloud. We show that TinyS2I can pro-

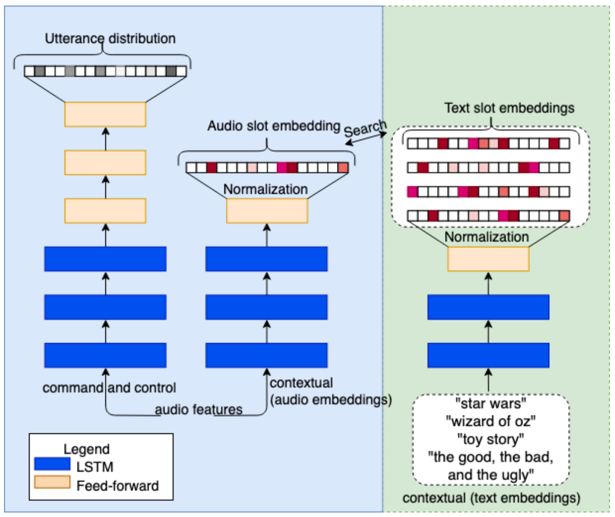

**Fig. 1**. TinyS2I architecture.

**Algorithm 1:** TinyS2I inference

```
Input: audio features, $C$ textual embeddings $c_i$
Output: estimated utterance
1  logits = command_and_control_model(audio features)
2  estimated_class = $\arg\max$(logits)
3  if estimated_class == play VideoName then
4      $a =$ audio_embedding_model(audio_features)
5      $d_i = ||a - c_i||^2, i = 1, ..., C$
6      if $\min(d_i) < t_m$ then
7          video title on screen, slot = $\arg\min d_i$
8      else
9          video title not on screen (unsupported)
10 else
11     if $\ell_{ood} < t_{ood}$ then
12         supported, utterance = $\arg\max$(logits)
13     else
14         unsupported
```

cess the majority of supported utterances on-device (thus improving latency by eliminating the need to transfer audio to the cloud) with high accuracy and correctly identify unsupported ones. We also show that the contextual model is able to accurately identify the dynamic content (video title from the screen), and all with a model whose disk footprint is less than 20 MB.

## 2. MODEL ARCHITECTURE

The TinyS2I architecture is shown in Fig. 1. It consists of two neural models: the command and control model that is responsible for classifying the utterance into one of the supported classes or deciding that an utterance is unsupported, and the contextual model that is responsible for handling the dynamic slot content, such as video names, that appear on the screen. The two models are trained independently, on far-field anonymized datasets. The input audio features are 64-dimensional Log-filter bank energies (LFBE) obtained by segmenting utterances with a window of 25 ms length and a frame rate of 10 ms. We used a left context of 3 frames, resulting in 192-dimensional input features, with a skip rate of 3 frames. The features are normalized with global mean and variance.

### 2.1. Command and control model

The command and control model is a neural classification model that classifies audio into a predefined set of classes. The output of the network is a vector of $M + 2$ logits, where $M$ is the number of command and control utterances (in our case 132). There are two additional classes, for the unsupported and the contextual "play *VideoName*" utterances.

The model consists of a recurrent encoder with three uni-directional LSTMs of 128 units, followed by two feed-forward layers of 256 units with *tanh* activations, and a final feed-forward layer of $M + 2$ units with a softmax activation. The total number of parameters in the model is 560K. For training, we used a set of 894K utterances that consist of 6K examples for each of command and control utterance, 102K unsupported utterances, and 15K "play *VideoName*" utterances for the contextual class. Experimentally we found that this ratio was enough to train a robust model. We used the Adam optimizer with a cross-entropy loss. The learning rate was set to $10^{-3}$ and we used early stopping with 10 epochs patience.

### 2.2. Contextual model

The contextual model consists of two parts: one that generates an audio embedding from the input audio and another that generates textual embeddings from the entries of a video name catalog. In our use case the catalog consists of video names that appear on the TV screen. We trained the model using a multi-modal neural metric learning approach with the goal that when the video title in the audio and text match, the model will produce embeddings that are similar.

The audio embedding network consists of a recurrent encoder with 3 uni-directional LSTMs with 256 units and a linear feed-forward layer with 256 units. The output of the feed-forward layer is L2-normalized and constitutes the audio embedding. The text embedding network produces a text embedding for each video title on the catalog of video titles that appear on the screen. The video names are converted to wordpieces using a wordpiece model of 2.5K wordpieces. These are the input to an embedding layer that converts them to 128-dimensional representations. The embedding layer is followed by a recurrent encoder with two bi-directional LSTMs of 256 units. The output of this layer is fed to a linear feed-forward layer of 256 units. The output of the feed-forward layer is again L2-normalized.

The audio embedding network consists of 1.5M parameters while the text embedding network consists of 3M parameters. Although the text embedding network is the largest component of the TinyS2I model, this network need not be executed during inference when the user speaks an utterance to the device. Since it only uses information from a catalog (in this case the TV screen), the text embeddings can be calculated once and recalculated only when the screen updates.

For training we used 978K "play *VideoName*" utterances. The model was trained using the Adam optimizer and the loss function was the triplet semi-hard loss [24], which is defined as:

$$L(A, P, N) = \max\left(||A - P||^2 - ||A - N||^2 + \alpha, 0\right) \quad (1)$$

In our case, the anchor input $A$ is the audio embedding, the positive sample input $P$ is the textual embedding for the video title that matches the video title spoken in the audio, and $N$ is the negative textual embeddings (i.e., video titles that do not match the audio). For each positive pair of audio-text, we used 14 negative textual examples from video titles randomly sampled from a large movie catalog. The learning rate was set to $10^{-3}$ and we used early stopping with 10 epochs patience. The margin $\alpha$ was set to 0.6, which we experimentally found to produce good results.

## 3. MODEL INFERENCE

The inference is described in Algorithm 1. The output of the command and control network drives the decision of whether the utterance is contextual, supported, or unsupported. In the case of a contextual ("play *VideoName*") utterance, the Euclidean distance is calculated between the audio embedding and the candidate textual embeddings from the video titles on the screen. Since the embeddings are of unit length, the distance takes values in the range of $[0, 2]$. The final video name slot is selected as the one with the minimum distance, given that it is less than a threshold $t_m$. If the distance is greater than $t_m$ then the user has spoken a "play *VideoName*" utterance for a video title that does not appear on the screen; the utterance is treated as unsupported and deferred to the cloud. If the predicted class is not a contextual utterance, we still need to decide if it is a supported or an unsupported utterance. The decision is based on whether the logit for the unsupported class $\ell_{ood}$ is less than a threshold $t_{ood}$, in which case the utterance is predicted as supported and the utterance class is found based on the logit with the maximum value; otherwise the utterance is predicted as unsupported.

The threshold $t_{ood}$ is used to control how aggressively we sacrifice some supported utterances by sending them to the cloud, in order to minimize false positives (i.e., unsupported utterances wrongly predicted as supported). Note here that false positives are the ones that degrade users' experiences. When a supported utterance is recognized as unsupported, it will simply be sent to the cloud for processing. Users will experience a higher latency but the utterance will be handled by a powerful, more accurate cloud model. However, when an unsupported utterance is recognized as supported, the recognition will be erroneous and a wrong action will be performed by the device, degrading users' experience. Thus, for this type of system, it is important to minimize the erroneous recognition of unsupported utterances as supported. As our experimental results in Section 4 indicate, TinyS2I is able to achieve high True Negative Rates (TNR), which shows that the majority of unsupported utterances are correctly recognized as unsupported.

## 4. RESULTS AND DISCUSSION

All our experiments are performed on far-field anonymized data. We report the relative performance improvement or degradation compared to a powerful ASR model that is used as a baseline. For our baseline, we used a hybrid DNN-HMM ASR model. The acoustic model is a low-frame-rate model with 2-layer frequency LSTM [25] followed by a 5-layer time LSTM trained on CTC loss, followed by sMBR loss [26]. The LM is a 4-gram model with a vocabulary of 2M words. In our first experiment, we evaluate the command and control model on the 132 command and control utterances and on unsupported utterances. The contextual "play *VideoName*" utterance is evaluated on the subsequent experiments. We used a dataset of 22K test examples with a balanced distribution for the 132 command and control utterances and 26K unsupported utterances. We measure the model's True Positive Rate (TPR) in terms of how well the model accepts supported utterances and processes them on-device, the accuracy (computed on the true positives) of recognition of the NLU interpretation (in the form of an intent, slot values, and slot tags), and the TNR, that shows how well the model correctly rejects unsupported utterances. For all models, the NLU interpretation is found through a one-to-one mapping (exact match rules) between the predicted supported utterances and the intent, slot values and slot tags. We also compare with a small on-device ASR model (Small ASR), with a 2.4M parameter DNN acoustic model and a 3-gram LM with a vocabulary of 3K words specifically tailored for the command and control utterances. The disk footprint of the small ASR model is less than 20MB, comparable to the size of TinyS2I.

**Table 1**. Relative performance improvement for command and control and unsupported utterances.

| Model | $t_m$ | $t_{ood}$ | TPR % | Acc. % | TNR % |
|---|---|---|---|---|---|
| ASR Baseline | N/A | N/A | - | - | - |
| Small ASR | N/A | N/A | -13.62 | -3.08 | -0.73 |
| TinyS2I | 0.6 | 0.001 | -9.42 | 0.30 | 0.36 |
| | | 0.005 | -0.34 | -0.04 | -0.25 |
| | | 0.01 | 1.99 | -0.24 | -0.71 |
| | 0.9 | 0.001 | -9.42 | 0.30 | 0.08 |
| | | 0.005 | -0.34 | -0.04 | -0.54 |
| | | 0.01 | 1.99 | -0.24 | -0.99 |

The relative performance is shown in Table 1. The TinyS2I model achieves very similar performance to the powerful baseline system in terms of accuracy and TNR. TPR varies according to the value of $t_{ood}$, which is expected as this threshold affects the supported/unsupported decision (see Sec. 3) . Threshold $t_m$ does not affect TPR or accuracy, since this threshold is only used for contextual utterances. It, however, slightly affects TNR in cases where an unsupported utterance is estimated as a contextual one. Depending on the value of $t_{ood}$, we observe that our model correctly accepts less, the same, or more supported utterances. As noted previously, even in the cases where we notice degradation in TPR ($t_{ood} = 0.001$), this is not detrimental for users' experience compared to having false positives: users will not experience the latency reductions offered by on-device processing, but the utterance will still be recognized by the cloud. It can also be observed that our TinyS2I model outperforms the small ASR model, showing the benefit of an S2I approach compared to a traditional ASR system of the same model size. In total, the model accepts the vast majority of the command and control utterances and provides similar accuracy and TNR as the powerful baseline.

Next, we evaluate the performance on contextual utterances. We used 15K "play *VideoName*" utterances to evaluate how well the model correctly accepts them as contextual utterances (TPR) and how accurately it can detect the correct slot value (i.e., correct video title from the screen). In this experiment, our dataset contains video titles that the model has "seen" during training, and we assume that 15 video titles appear on the screen. We also evaluate how well the model can reject contextual utterances when the video title does not appear on the screen (TNR). In this case, the "play *VideoName*" utterance contains a video title that does not match any of the 15 video titles that are assumed to be shown on the screen. Video titles for negative examples were randomly sampled from a large movie catalog. As our small ASR model is not trained to accurately recognize video titles, it is not included in the contextual evaluation.

**Table 2**. Relative performance for contextual utterances.

| Model | $t_m$ | $t_{ood}$ | TPR % | Acc. % | TNR % |
|---|---|---|---|---|---|
| ASR Baseline | N/A | N/A | - | - | - |
| TinyS2I | 0.6 | 0.001 | -16.39 | 15.91 | -0.58 |
| | | 0.005 | -16.39 | 15.91 | -1.62 |
| | | 0.01 | -16.39 | 15.91 | -2.42 |
| | 0.9 | 0.001 | -14.08 | 15.84 | -3.84 |
| | | 0.005 | -14.08 | 15.84 | -4.89 |
| | | 0.01 | -14.08 | 15.84 | -5.69 |

**Table 3**. Relative performance for zero-shot contextual utterances.

| Model | $t_m$ | $t_{ood}$ | TPR % | Acc. % |
|---|---|---|---|---|
| ASR Baseline | N/A | N/A | - | - |
| TinyS2I | 0.6 | 0.001 | -27.91 | 15.84 |
| | 0.9 | 0.001 | -18.87 | 15.51 |
| | 1.2 | 0.001 | -16.01 | 14.98 |

The results are shown in Table 2. There is a 14-16% degradation in TPR compared to the baseline while, as expected, TPR improves as $t_m$ increases. While this degradation seems severe, it is not detrimental as their utterance is deferred to the cloud for recognition. However, the majority of contextual utterances are still processed on-device and with improved accuracy compared to the baseline. The reason for the accuracy improvement is that the baseline does not utilize any contextual information to recognize movies, showing the benefits of using context in our model. We also observe that $t_{ood}$ does not affect the TPR or accuracy, since it only drives the supported/unsupported decision for the command and control utterances. Finally, there is some degradation in TNR which becomes more evident as the $t_m$ threshold increases.

Table 3 shows the performance on 15K contextual utterances where the video name slot has not been “seen” during training (zero-shot cases). Depending on the $t_m$ threshold, we observe higher degradation in TPR, but only a small variability in accuracy in this zero-shot test set compared to Table 2 (relative improvements remain the same compared to the baseline). This shows that our model was able to learn how to match embeddings of audio and text and generalize well even in unseen video title slots.

We also evaluated the accuracy of the contextual model on contextual utterances for different catalog sizes (i.e., different numbers of video titles that appear on the screen). Table 4 shows the accuracy in identifying the correct video title slot for different catalog sizes, for video titles that are “seen” during training (TEST1), and zero-shot video titles (TEST2). We observe only a small variability in accuracy as the size of the candidate video titles increase, achieving very good performance even when the model has to find the correct match among 200 different textual embeddings.

Since all results presented in this paper are relative to a baseline, we also provide some absolute numbers to give a perspective on the operating point of our models. Note that, in all results, the contextual model had over 95% accuracy in correctly finding the video name slot and the command and control model had over 95% accuracy in supported utterances, over 80% TPR and over 95% TNR. The results again show that the majority of supported and contextual utterances are processed on-device (offering latency reductions) and the recognition is performed accurately, while the model correctly identifies the utterances it does not support.

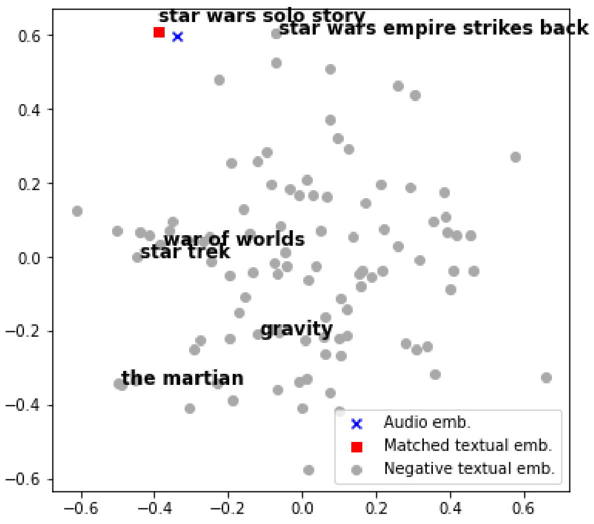

**Fig. 2**. Visualization of audio and text embeddings using T-SNE.

**Table 4**. Relative improvement in accuracy (%) for contextual slots for different catalog sizes.

| Model | Catalog size | TEST1 | TEST2 |
|---|---|---|---|
| ASR Baseline | N/A | - | - |
| TinyS2I | 15 | 15.91 | 15.84 |
| | 50 | 15.85 | 15.64 |
| | 100 | 15.66 | 14.97 |
| | 200 | 15.52 | 14.34 |

Finally, Figure 2 shows a visualization of the embeddings using T-SNE [27]. It shows the audio embedding for “play star wars solo story” and 100 textual embeddings from video titles that are assumed to be shown on the screen. It is noteworthy to observe how close the matched textual embedding is to the audio embedding, which validates that the model is able to produce audio-text embeddings that are similar for the same video title. It is interesting that similar textual embeddings to the audio appear close together, showing that the model was able to learn semantics between audio and text.

## 5. CONCLUSION

Supporting only a limited set of frequent utterances allows us to build very small and fast models for SLU processing. We presented one such model, called TinyS2I, and showed that such a model can achieve almost the same performance as a powerful baseline ASR model which is too large to fit on the device, while outperforming small on-device ASR models designed for the same use-case. The size of the TinyS2I model is less than 20 MB, which makes it a perfect candidate for heavily resource-constrained devices in terms of compute power and memory.

## 6. REFERENCES

[1] Alex Graves, "Sequence transduction with recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2012.

[2] Alex Graves, Abdel rahman Mohamed, and Geoffrey E. Hinton, "Speech recognition with deep recurrent neural networks," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013, pp. 6645–6649.

[3] Alex Graves and Navdeep Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, Bejing, China, 22–24 Jun 2014, vol. 32, pp. 1764–1772.

[4] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2016.

[5] Hagen Soltau, Hank Liao, and Haşim Sak, "Neural Speech Recognizer: Acoustic-to-Word LSTM Model for Large Vocabulary Speech Recognition," in *INTERSPEECH*, 2017, pp. 3707–3711.

[6] Raziel Álvarez, Rohit Prabhavalkar, and Anton Bakhtin, "On the efficient representation and execution of deep acoustic models," in *INTERSPEECH*, 2016.

[7] Hieu Duy Nguyen, Anastasios Alexandridis, and Athanasios Mouchtaris, "Quantization aware training with absolute-cosine regularization for automatic speech recognition," in *INTERSPEECH*, 2020.

[8] Yi Yang, Andy Chen, Xiaoming Chen, Jiang Ji, Zhenyang Chen, and Yan Dai, "Deploy large-scale deep neural networks in resource constrained iot devices with local quantization region," in *arXiv preprint arXiv:1805.09473*, 2018.

[9] Michael H. Zhu and Suyog Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," in *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, 2018.

[10] Ruoming Pang, Tara Sainath, Rohit Prabhavalkar, Suyog Gupta, Yonghui Wu, Shuyuan Zhang, and Chung-Cheng Chiu, "Compression of end-to-end models," in *INTERSPEECH*, 2018.

[11] Tao Lei, Y. Zhang, Sida I. Wang, Huijing Dai, and Yoav Artzi, "Simple recurrent units for highly parallelizable recurrence," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[12] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[13] Jonathan Macoskey, Grant P. Strimel, and Ariya Rastrow, "Bifocal neural asr: Exploiting keyword spotting for inference optimization," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ISCASSP)*, 2021.

[14] Jonathan Macoskey, Grant P. Strimel, Jinru Su, and Ariya Rastrow, "Amortized neural networks for low-latency speech recognition," in *INTERSPEECH*, 2021.

[15] Hamidreza Saghir, Samridhi Choudhary, Sepehr Eghbali, and Clement Chung, "Factorization-aware training of transformers for natural language understanding on the edge," in *INTERSPEECH*, 2021.

[16] Kanthashree Mysore Sathyendra, Samridhi Choudhary, and Leah Nicolich-Henkin, "Extreme model compression for on-device natural language understanding," in *International Conference on Computational Linguistics (COLING)*, 2020.

[17] Yuan-Ping Chen, Ryan Price, and Srinivas Bangalore, "Spoken language understanding without speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 6189–6193.

[18] Natalia Tomashenko, Antoine Caubrière, Yannick Estève, Antoine Laurent, and Emmanuel Morin, "Recent advances in end-to-end spoken language understanding," in *International Conference on Statistical Language and Speech Processing*. Springer, 2019, pp. 44–55.

[19] Parisa Haghani, Arun Narayanan, Michiel Bacchiani, Galen Chuang, Neeraj Gaur, Pedro Moreno, Rohit Prabhavalkar, Zhongdi Qu, and Austin Waters, "From audio to semantics: Approaches to end-to-end spoken language understanding," in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 720–726.

[20] Marco Dinarelli, Nikita Kapoor, Bassam Jabaian, and Laurent Besacier, "A data efficient end-to-end spoken language understanding architecture," *arXiv preprint arXiv:2002.05955*, 2020.

[21] Milind Rao, Anirudh Raju, Pranav Dheram, Bach Bui, and Ariya Rastrow, "Speech to semantics: Improve asr and nlu jointly via all-neural interfaces," *arXiv preprint arXiv:2008.06173*, 2020.

[22] Martin Radfar, Athanasios Mouchtaris, and Siegfried Kunzmann, "End-to-end neural transformer based spoken language understanding," in *INTERSPEECH*, 2020, pp. 799–803.

[23] Anirudh Raju, Gautam Tiwari, Milind Rao, Pranav Dheram, Bryan Anderson, Zhe Zhang, Bach Bui, and Ariya Rastrow, "End-to-end spoken language understanding using rnn-transducer asr," *arXiv preprint arXiv:2106.15919*, 2021.

[24] Florian Schroff, Dmitry Kalenichenko, and James Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[25] Bo Li and Tara Sainath et al., "Acoustic modeling for google home," in *INTERSPEECH*, 2017.

[26] Karel Veselý, Arnab Ghoshal, Lukas Burget, and Daniel Povey, "Sequence-discriminative training of deep neural networks," in *INTERSPEECH*, 2013.

[27] Laurens Van der Maaten and Geoffrey Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.