Slungt: even faster spoken language understanding with n-grams and tries

Daniel Bermuth, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Bermuth, Daniel, and Wolfgang Reif. 2025. "Slungt: even faster spoken language understanding with n-grams and tries." In ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 6-11 April 2025, Hyderabad, India, edited by Bhaskar D. Rao, Isabel Trancoso, Gaurav Sharma, and Neelesh B. Mehta, 1–5. Piscataway, NJ: IEEE. https://doi.org/10.1109/ICASSP49660.2025.10890387.

Slungt: Even Faster Spoken Language Understanding with N-Grams and Tries

Daniel Bermuth
Institute for Software & Systems Engineering
University of Augsburg
Augsburg, Germany
daniel.bermuth@informatik.uni-augsburg.de

Wolfgang Reif
ISSE
University of Augsburg
Augsburg, Germany
reif@isse.de

***Abstract*—In the domain of *Spoken Language Understanding* (SLU) the primary objective is to extract important information from audio commands, like the intent of what a user wants the system to do and specific entities like locations or numbers. This paper presents a simple method that integrates intents and entities into a *beam search algorithm*, and, in combination with a general-purpose *Speech-to-Text* model, enables the creation of customized SLU-decoders without any additional training. Constructing such decoders is very fast and only takes a few seconds. It is also completely language-independent. In comparative assessments across multiple benchmarks, this method demonstrates comparable performance to several other SLU strategies, while significantly surpassing them in terms of computational speed.**

***Index Terms*—spoken language understanding, speech to intent, offline voice assistant, beam search, ngram and trie decoding**

I. Introduction

When constructing models for *Spoken Language Understanding* (SLU), there exist two primary approaches: one involves employing two separate stages, transcribing the spoken command to text (*Speech-to-Text*, STT), followed by extracting useful information from the transcribed sentence (*Natural Language Understanding*, NLU). The alternative is using a direct SLU approach that integrates both stages into a single model. The first approach offers the advantage of independent training of the two models. The STT-model can often be used across various domains, while the NLU-model can be trained to the given task relatively quickly. On the other hand, the second approach tends to yield higher accuracy, as it circumvents the issue of errors from the STT transcription propagating into the NLU module.

Recent NLU or SLU systems commonly rely on neural networks to extract features. Although these networks achieve high recognition accuracy, they come with the drawback that their training requires a substantial amount of time. This work explores an entirely different approach that eliminates the need for specialized training, enabling the rapid development of customized SLU models.

It uses a customized *beam-search decoding* process instead of neural networks for SLU parsing. The speech recognition part is still based on a neural network, but this only has to be trained once per language on general-purpose STT tasks. In this work *Conformer* [1] models from *NeMo* [2] and *Scribosermo* [3] are used. Those models, after conversion to *tflite* and additional quantization, are especially suitable for running on a RaspberryPi4.

A comparison across various benchmarks demonstrates that this approach performs competitively with other solutions, while being significantly faster. The proposed concept is simple and allows the construction of customized decoder models within a few seconds, and can decode commands faster than real-time on a RaspberryPi4.

The source-code of the presented method for training-free SLU parsing, named *slungt*, as well as the models from *Scribosermo*, can be found at: https://gitlab.com/Jaco-Assistant

Some years ago it was common to use *Finite State Transducers* (FSTs) for speech recognition tasks [4], [5]. Some works already explored the usage of FSTs as static language models for parsing NLU information by adding semantic tags into the FSTs, either from textual inputs [6], [7] or from speech transcription hypotheses of a hidden Markov model [8]. In *finstreder* [9] an approach of embedding semantic tags into the FSTs was presented, which allowed the usage of modern *speech-to-text* neural networks and that outperformed various other approaches in multiple SLU benchmarks. A different approach to detect entities was presented by [10], in which weighted trie structures were used to train classifiers to tag known and unknown entities. If only a textual speech transcription is required, another option, that uses the fact that the vocabulary is restricted and words are normally not combined randomly, would be to use a language model to score possible token combinations. Two common libraries are *ds-ctcdecoder* [11] and *pyctcdecode* [12]. They both use a *beam-search* approach in combination with a *KenLM* [13] language model. This work follows some ideas from *finstreder*, mainly of using a static SLU decoder instead of a learned one, but implements a *beam-search* approach to significantly improve the decoding speed.

II. Sentence-piece vs char-based inputs

One of the reasons for developing *slungt* was based on preceding experiments with *finstreder* [9], which evaluated the impact of different input types on the performance. In the last years most STT model architectures started to use *sentence-piece* [14] labels instead of *character-based* ones to improve their performance. Basically *sentence-piece* labels are a combination of character and sub-word tokens, like *a, ing, tion, z*. The tokens are generated by a special algorithm that iteratively merges frequently occurring sub-word units. In the end, the most common words are kept as single tokens, but less common words are split into multiple parts.

The author of [15] found that the improved performance *"can be linked to its compression capacity, that is, the capacity of finding a set of words such that they are able to cover the sequence with as few words as possible"*. The experiments with *finstreder* evaluated the impact of a different word distribution, with the following assumption: *"If there are many unknown words, the model needs to combine them from smaller tokens that weren't used as much in training, which should lead to more errors. Choosing the wrong subwords might affect multiple characters at once, whereas a character-based model might only switch a single one. For a later following language model it should be easier to correct errors if the predicted word is not completely wrong"*.

The first experiment was run with the *SmartSpeaker* benchmark [16]. It has the difficulty of containing many artists or music tracks with uncommon names in the commands, like *"play music by [a boogie wit da hoodie]"* or *"I'd like to listen to [Kinokoteikoku]"*. As shown in Table I, the *character-based* model significantly outperforms the *sentence-piece* one in this task. From the differences in the greedy *Word Error Rate* (WER) it can be followed that an important part of this improvement comes from the better correction by the decoder's language model. Regarding the SLU accuracy, a command is considered as correct if the intent and all the slots can be retrieved.

TABLE I: Results on *SmartSpeaker* dataset.

	sentence-piece	**char-based**
Greedy WER	**0.176**	0.220
Finstreder SLU	0.811	**0.879**

The *character-based Conformer* model with 29 output tokens was trained with *Scribosermo*, using the 128 token *sentence-piece* model's weights as initialization. An additional deconvolution layer was added before the output, to reduce the time compression from $1/4$ to $1/2$, to prevent information loss from having fewer timesteps than output characters. Note that if only general-purpose STT tasks are of interest, the *sentence-piece* model outperforms the *character-based* one (for example: WER on *CommonVoice* [17]: 9.4% vs 17.7%).

A similar SLU impact was found using a subset of the *Barista* benchmark [18], which consists of commands for coffee orders. An example would be: *"i'd like a [medium roast] [large] [mocha] with [lots of cream] and [a little bit of brown sugar]"*. In this case, the words are rather common. The STT models use the *Conformer* architecture again, but this time with the *small* instead of the *large* variant. Two *sentence-piece* models, one with 128 and one with 1024 output tokens were evaluated. Table II shows that again the *character-based* model performs better in this task. The *sentence-piece* model with 1024 tokens performs worse than the one with 128 tokens, even though it has a better general greedy decoding WER.

TABLE II: Accurracy on *Barista* dataset.

	sp-128	**sp-1024**	**cb-29**
Finstreder SLU	0.95	0.92	**0.98**

While the *character-based* model showed a better performance in those two benchmarks, this can not be generalized to all scenarios, as can be seen later in the *Experiments* section with the *SmartLights* benchmark. In general, across all evaluated benchmarks, the *character-based* model performed better in tasks with many rare words, while the *sentence-piece* one was slightly better in tasks with many common words (even though the improvement in this direction was not as large).

Now the problem is that *finstreder* with the *character-based* inputs is much slower (between $2\times$ to $10\times$, depending on the task) than with *sentence-piece* inputs. The main reason is that the number of output timesteps of the *character-based* model is twice as large, because it can not merge multiple characters into a single token. This leads to a much larger number of possible paths in the *Finite State Transducer*, which increases the decoding time (the search of the shortest path in the FST graph). In fact, the decoding process was sometimes so slow that real-time decoding was no longer possible on a powerful desktop computer, let alone a RaspberryPi4.

III. SLUNGT'S APPROACH

The FST-based approach of *finstreder* basically builds a graph of possible outputs and a second input graph of most probable tokens, using the CTC-token probabilities as transition costs. Then it combines the two graphs into a single very large graph and searches for the shortest path in it. The resulting path contains the most probable transcription of the input audio. While this approach showed outstanding results in the benchmarks, and is faster than most other alternative solutions, there is still potential for improvement.

Instead of creating a single large graph and searching for the best path in it, *slungt* uses a *beam-search* approach to find the best path step by step, separating the problem along the time axis. This results in a much smaller number of possible paths which are checked, because bad paths can be discarded earlier. The following sections show how intent and entity information can be included directly into the *beam-search* decoding process.

A. Preparing the inputs

The preparation phase is separated into two different steps. As only input a *json*-file following the syntax of *Jaco* [19] is required:

```
{
  "intents": {
    "get-looks": [
      "(is a|are) [---](animal) cute"
    ]
  },
  "lookups": {
    "animal": [
      "atlantic stargazer",
      "aye aye",
      "(hairy frogfish)->striated frogfish"
    ]
  }
}
```

In the first step, a new text file for each intent is generated from the *json*-file, containing all of the intent's possible example sentences. Afterward, an n-gram model is created for each intent file using the *KenLM* [13] toolkit.

```
is a (animal) cute
are (animal) cute
```

In the second step, a vocabulary for each intent is generated, containing all normal words from the intent sentences as well as the lookup values. Note that the lookup values are not split into single words and still contain their entity syntax.

```
is
a
cute
[atlantic stargazer](animal)
[aye aye](animal)
[hairy frogfish->striated frogfish](animal)
```

B. Word Trie

In the decoding process, a trie-based data structure is used to check if a word suggested from the speech-to-text model is part of the vocabulary. A trie is a tree-like data structure where each node represents a single character. The root node is empty, and each node can have multiple children. The children are the next characters of the word. The last character of a word is marked as a leaf node. See Figure 1 for an example. To check if a suggestion is valid, the trie is traversed starting from the root node, following the characters of the suggestion. If a leaf node is reached with the last character of the suggestion, the word is valid and complete. If only a non-leaf node was reached, the suggestion is not complete yet. If a node does not have a child with the next character of the suggestion, the suggestion is invalid. In difference to the standard procedure, a leaf node does

not return a single word, but instead a set of word options. These can include normal words or lookup values or both, if the word is a lookup value in only some of the intent's examples.

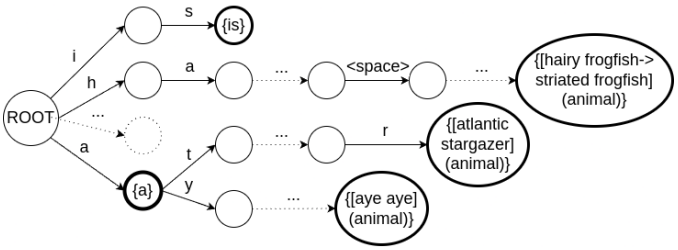

Fig. 1: Example of a word trie with entities.

C. Decoding

The input of the decoding process is the CTC-token probability for each timestep from the speech-to-text model. The process starts with one empty beam per intent. Then for each timestep, the following steps are executed:

1) For each token with a probability above a threshold, a new beam is created from each of the previous beams.
2) For each new beam the new token is added to the beam's temporary text.
3) All beams with the same content are merged.
4) Each temporary text is evaluated with the word trie.
5) If a new word is found, it is scored with the intent's n-gram model and added to the beam's transcription. Entity values are scored as their entity type.
6) The number of beams is pruned to the maximum beam-size, keeping only the best ones.

At the end of the decoding process, unfinished beams are penalized. The beams are sorted again and then returned. The one with the highest score is considered as the final result. The intent was already stored as an attribute while creating the initial beams, and the slots are extracted from the beam's transcription.

In step (4) a beam's temporary text is evaluated by the trie. If it is not complete yet, nothing happens. If it is not valid, a penalty is added to the beam's score, and if it contains a space, the invalid word is added to the beam's transcription. If it is valid, a new beam is created for each word option returned by the trie. If it also has children, the current beam is kept as well, else it is dropped. Since this step can duplicate beams that might get the same content in a following step, a beam-id is added to each beam to prevent wrong score calculation in the merging step. In the case *sentence-piece* tokens are used, the trie evaluation process needs to include some extra steps, which can be found in the source-code. This is because new tokens might contain multiple spaces, similar to the vocabulary, and therefore new suggested words can not simply be split by spaces, what would be the approach with a standard word trie without tagging.

IV. Experiments

The performance of *slungt* was evaluated in multiple benchmarks, following the same procedure as in *finstreder* [9] and using the same *Conformer* model as well. Some of the experiments with *finstreder* were repeated with improved decoder parameters. A command is considered as correct if the intent and all the slots can be retrieved.

A. Spoken Language Understanding

The *Barista* benchmark was published by *Picovoice* [18] and consists of 620 commands of people ordering coffee in English. The audio is mixed with different volume levels of background noise from cafe and kitchen environments. The results in Figure 2 show that *slungt* shows a slightly better performance than *finstreder* across all noise levels. In both approaches, *sentence-piece (sp)* tokens resulted in better accuracy at high noise levels, whereas *character-based (cb)* ones were better at low noise levels.

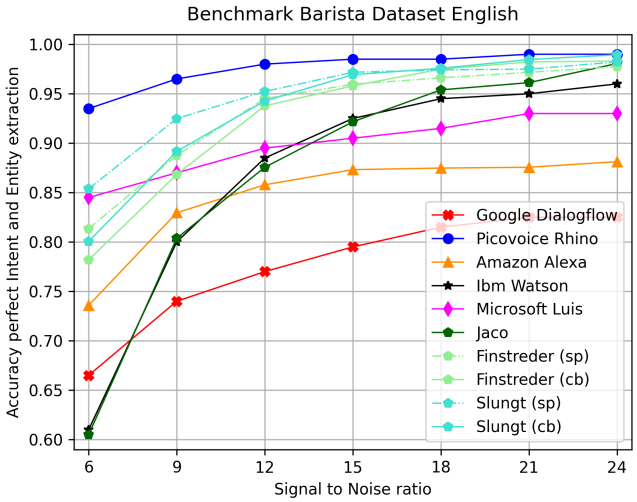

Fig. 2: Coffee orders with noisy backgrounds. The results of *DialogFlow*, *Watson*, *Luis* and *Rhino* have been taken from [18], the results of *Alexa* and *Jaco* from [19], those of *Finstreder* from [9].

The *SmartLights* benchmark from *Snips* [16] tests the capability of controlling lights in different rooms. It consists of 1660 requests which are split into five partitions for a 5-fold evaluation. A sample command could be: *"please change the [bedroom] lights to [red]"* or *"i'd like the [living room] lights to be at [twelve] percent"*. The benchmark results are presented in Table III. The performance of *slungt* is again on par with *finstreder*, and outperforms all other approaches. The *Real Time Factor* (decoding-time / audio-duration) is measured on a desktop computer with an *AMD-3700X* CPU, no GPU, in an end-to-end (audio-to-intent) manner. Due to the optimized decoding approach *slungt* is multiple times faster than *finstreder*, especially when comparing the *character-based* variants.

TABLE III: Results on *SmartLights* dataset.

	Accuracy	**WER**	**RTF**
Snips [16]	0.842	–	–
Alexa [19]	0.792	–	–
Houndify [19]	0.545	0.108	–
Jaco [19]	0.854	0.108	–
AT-AT [20]	0.849	–	–
Finstreder (Conformer sp)	**0.902**	0.075	0.283
Finstreder (Conformer cb)	0.899	0.062	0.597
Slungt (Conformer sp)	0.893	**0.051**	**0.062**
Slungt (Conformer cb)	0.880	0.062	0.063

The *SmartSpeaker* benchmark tests the performance of reacting to music player commands in English as well as in French. The benchmark is from *Snips* [16], too, and is the only one that could be found that includes a language other than English. As found in chapter II, the *character-based* model significantly outperforms the

sentence-piece one, with *slungt* as well. One reason why *slungt* does not completely match *finstreder*, is that the latter is able to correct errors early in the name, if the complete match gets better at the end, whereas *slungt* might have already pruned such a beam.

TABLE IV: Accurracy on *SmartSpeaker* dataset.

	English	French
Snips [16]	0.687	0.751
Jaco [19]	0.627	0.480
Alexa [19]	0.455	**0.889**
Finstreder (sp / cb)	0.811 / **0.879**	0.806 / 0.865
Slungt (sp / cb)	0.791 / 0.847	0.778 / 0.868

FluentSpeechCommands [21] is the most commonly used benchmark in this domain and tests simple voice assistant requests. It includes commands like *"turn up the [bathroom] temperature"*, *"switch the lights on"* or *"go get me my [shoes]"*. Table V shows that *slungt* has a similar performance like other approaches, and falls slightly behind *finstreder* here, as it can not force match the input tokens to command options. But if using the more accurate token probabilities from the *AMT* model ([9], finetuned with training audios, for around 3h on one Nvidia-1080Ti), the difference gets very small.

TABLE V: Accuracy on *FSC* dataset.

Alexa [9]	0.987
Cao et al. [22], *Radfar et al.* [23], *Benazir et al.* [24]	0.990
Slungt (sp / cb)	0.983 / 0.992
Reptile [25], *CMMC* [26]	0.992
AT-AT [20]	0.995
Finstreder (sp / cb) [9]	0.995 / 0.994
Seo et al. [27], *Qian et al.* [28], *Kim et al.* [29]	0.997
Slungt (cb) + AMT	0.997
UniverSLU [30]	0.998
Finstreder (cb) + AMT [9]	0.998

B. Textual inputs

Similar to *finstreder*, the decoding approach of *slungt* can also be used for NLU parsing of textual inputs. For this the textual input is converted to CTC-labels, by assigning a very high probability around 0.99 to the actual character and the other characters get a very low probability around 0.001. Same as *finstreder*, it outperforms traditional approaches on very simple datasets, but falls behind if the datasets are more complex.

TABLE VI: NLU only, tested with textual inputs.

	SmartSpeaker		SmartLights	
	Accuracy	WER	Accuracy	WER
Jaco (Rasa) [9]	0.977	–	**0.960**	–
Finstreder	0.991	0.133	0.909	0.051
Slungt	**0.998**	0.0	0.902	0.027

C. Standard textual beam-search

It is also possible to use *slungt* for plain text decoding, without intent and entity tagging. In this experiment two language model types are tested, one *5-gram* model that is customized to the *SmartLights* domain and only contains words from the benchmark (created with tools from the first version of *Jaco* [19]), and the other is a large *3-gram* general model (from *LibriSpeech* [31], [32]). In comparison to *ds-ctcdecoder* [11] the accuracy is similar, while *slungt* is only a bit slower (*RTF* measured tokens-to-text only). One reason is that *ds-ctcdecoder* does not support *sentence-piece* tokens and can therefore include additional optimizations. Another is that *slungt* needs more logic for intent and entity handling. The implementation of *pyctcdecode* [12] is a bit faster, but also has more transcription errors. As a baseline, the greedy WER of the STT-model on this subset is 12.2%.

TABLE VII: STT only, using different language models.

SmartLights	domain		general	
	WER	RTF	WER	RTF
ds-ctcdecoder	0.054	0.002	**0.102**	0.003
pyctcdecode	0.072	**0.001**	0.134	**0.002**
slungt	**0.052**	0.002	0.103	0.004

D. Limitations

The approach of *slungt* has similar limitations as *finstreder*, but there exists a simpler workaround. In general, it does not work well with open questions or commands and currently can only recognize predefined lookup values. One option to improve this would be to add a default fallback intent with a large vocabulary for general speech-to-text tasks (like the *3-gram* model from before), and then send the textual transcriptions to an online service for further processing (if self-hosting a *Large Language Model* is not possible), if no other intent was matched. In this way a smart home assistant could, for example, have some skills for controlling lights or music, and if the user has a more complicated question like *"where on earth has the most volcanos?"*, the assistant could catch the fallback intent and ask a cloud service for the answer. This would still preserve most of the user's privacy since the audio recording is not sent to some cloud provider. The second benefit of such an approach is that, for the more important local skills, the assistant keeps its high accuracy and low latency.

V. Discussion and Conclusion

In this paper, a simple, fast, and language-independent method for direct *Spoken Language Understanding* without training is presented under the name of *slungt*. For this, a *beam-search* concept is extended to the task of intent and entity extraction. In multiple benchmarks a performance on par with current state-of-the-art algorithms could be achieved.

Similar to *finstreder*, the main advantage over other approaches is that no extra training of the SLU model is required, with the difference that the decoding process of *slungt* is much faster. For the initialization of the decoder only text files describing possible requests are needed, which are easy to create, adjust, and share. In most voice assistants like *Jaco*, *Alexa*, or others, such text files (in slightly differing formats) are already included in the skills.

The preprocessing step of *slungt* is very fast and takes only a few seconds. The implementation is done in *C++* which simplifies using it on edge devices or smartphones. Using *SWIG* a *Python* wrapper was created as well. In comparison to *finstreder*, the compilation process is much faster and less resource-demanding, and now works directly using the shared runners in the *GitlabCI* pipeline. Similar to *finstreder*, the decoding process of *slungt* runs faster than real-time on a RaspberryPi4 (*SmartLights* results with *tflite-sp*: 0.882|0.066|0.460). It now also supports a streamed input to reduce latency even further.

The proposed method is particularly suited for scenarios with personalized domains, frequent skill changes, small datasets, or limited training options. For instance, it is useful in customizable smart home assistants on edge devices or smartphones, where avoiding cloud services is preferred because of unstable internet connections or privacy reasons.

REFERENCES

[1] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al., "Conformer: Convolution-augmented transformer for speech recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[2] Oleksii Kuchaiev, Jason Li, Huyen Nguyen, Oleksii Hrinchuk, Ryan Leary, Boris Ginsburg, Samuel Kriman, Stanislav Beliaev, Vitaly Lavrukhin, Jack Cook, et al., "Nemo: a toolkit for building ai applications using neural modules," *arXiv preprint arXiv:1909.09577*, 2019.

[3] Daniel Bermuth, Alexander Poeppel, and Wolfgang Reif, "Scribosermo: Fast Speech-to-Text models for German and other Languages," *arXiv preprint arXiv:2110.07982*, 2021.

[4] Mehryar Mohri, Fernando Pereira, and Michael Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[5] Takaaki Hori and Atsushi Nakamura, "Speech recognition algorithms using weighted finite-state transducers," *Synthesis Lectures on Speech and Audio Processing*, vol. 9, no. 1, pp. 1–162, 2013.

[6] Christian Raymond and Giuseppe Riccardi, "Generative and discriminative algorithms for spoken language understanding," in *Interspeech 2007-8th Annual Conference of the International Speech Communication Association*, 2007.

[7] Takeshi Homma, Adriano S Arantes, Maria Teresa Gonzalez Diaz, and Masahito Togami, "Maximizing SLU performance with minimal training data using hybrid RNN plus rule-based approach," in *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, 2018, pp. 366–370.

[8] Christian Raymond, Frédéric Béchet, Renato De Mori, and Géraldine Damnati, "On the use of finite state transducers for semantic interpretation," 2005.

[9] Daniel Bermuth, Alexander Poeppel, and Wolfgang Reif, "Finstreder: Simple and fast Spoken Language Understanding with Finite State Transducers using modern Speech-to-Text models," *arXiv preprint arXiv:2206.14589*, 2022.

[10] Silviu Cucerzan and David Yarowsky, "Language independent named entity recognition combining morphological and contextual evidence," in *1999 joint SIGDAT conference on empirical methods in natural language processing and very large corpora*, 1999.

[11] Mozilla Foundation, "Project DeepSpeech," 2021, Available: https://github.com/mozilla/DeepSpeech, [Online, accessed 03-January-2024].

[12] Kensho Inc, "pyctcdecode," 2021, Available: https://github.com/kensho-technologies/pyctcdecode, [Online, accessed 03-January-2024].

[13] Kenneth Heafield, "KenLM: Faster and smaller language model queries," in *Proceedings of the sixth workshop on statistical machine translation*, 2011, pp. 187–197.

[14] Taku Kudo and John Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *arXiv preprint arXiv:1808.06226*, 2018.

[15] Matthias Gallé, "Investigating the effectiveness of BPE: The power of shorter sequences," in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, 2019, pp. 1375–1381.

[16] Alaa Saade, Alice Coucke, Alexandre Caulier, Joseph Dureau, Adrien Ball, Théodore Bluche, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, et al., "Spoken language understanding on the edge," *arXiv preprint arXiv:1810.12735*, 2018.

[17] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M Tyers, and Gregor Weber, "Common voice: A massively-multilingual speech corpus," *arXiv preprint arXiv:1912.06670*, 2019.

[18] Picovoice, "Barista Benchmark," 2021, Available: https://github.com/Picovoice/speech-to-intent-benchmark, [Online, accessed 24-March-2022].

[19] Daniel Bermuth, Alexander Poeppel, and Wolfgang Reif, "Jaco: An Offline Running Privacy-aware Voice Assistant," in *Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction*, 2022, pp. 618–622.

[20] Subendhu Rongali, Beiye Liu, Liwei Cai, Konstantine Arkoudas, Chengwei Su, and Wael Hamza, "Exploring Transfer Learning For End-to-End Spoken Language Understanding," *arXiv preprint arXiv:2012.08549*, 2020.

[21] Loren Lugosch, Mirco Ravanelli, Patrick Ignoto, Vikrant Singh Tomar, and Yoshua Bengio, "Speech model pre-training for end-to-end spoken language understanding," *arXiv preprint arXiv:1904.03670*, 2019.

[22] Yiran Cao, Nihal Potdar, and Anderson R. Avila, "Sequential End-to-End Intent and Slot Label Classification and Localization," in *Proc. Interspeech 2021*, 2021, pp. 1229–1233.

[23] Martin Radfar, Athanasios Mouchtaris, Siegfried Kunzmann, and Ariya Rastrow, "FANS: Fusing ASR and NLU for On-Device SLU," in *Proc. Interspeech 2021*, 2021, pp. 1224–1228.

[24] Afsara Benazir, Zhiming Xu, and Felix Xiaozhu Lin, "Speech Understanding on Tiny Devices with A Learning Cache," in *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*, 2024, pp. 425–437.

[25] Yusheng Tian and Philip John Gorinski, "Improving End-to-End Speech-to-Intent Classification with Reptile," *arXiv preprint arXiv:2008.01994*, 2020.

[26] Beida Zheng, Mijit Ablimit, and Askar Hamdulla, "Cross-Modal Alignment for End-to-End Spoken Language Understanding Based on Momentum Contrastive Learning," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 10576–10580.

[27] Seunghyun Seo, Donghyun Kwak, and Bowon Lee, "Integration of Pre-trained Networks with Continuous Token Interface for End-to-End Spoken Language Understanding," *arXiv preprint arXiv:2104.07253*, 2021.

[28] Yao Qian, Ximo Bianv, Yu Shi, Naoyuki Kanda, Leo Shen, Zhen Xiao, and Michael Zeng, "Speech-language pre-training for end-to-end spoken language understanding," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 7458–7462.

[29] Seongbin Kim, Gyuwan Kim, Seongjin Shin, and Sangmin Lee, "Two-stage Textual Knowledge Distillation to Speech Encoder for Spoken Language Understanding," *arXiv preprint arXiv:2010.13105*, 2020.

[30] Siddhant Arora, Hayato Futami, Jee-weon Jung, Yifan Peng, Roshan Sharma, Yosuke Kashiwagi, Emiru Tsunoo, and Shinji Watanabe, "UniverSLU: Universal Spoken Language Understanding for Diverse Classification and Sequence Generation Tasks with a Single Network," *arXiv preprint arXiv:2310.02973*, 2023.

[31] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[32] OpenSLR, "LibriSpeech language models, vocabulary," 2024, Available: https://www.openslr.org/11, [Online, accessed 17-June-2024].