JoeyS2T: Minimalistic Speech-to-Text Modeling with JoeyNMT

Mayumi Ohta
Computational Linguistics
Heidelberg University, Germany
ohta@cl.uni-heidelberg.de

Julia Kreutzer
Google Research
jkreutzer@google.com

Stefan Riezler
Computational Linguistics & IWR
Heidelberg University, Germany
riezler@cl.uni-heidelberg.de

Abstract
JoeyS2T is a JoeyNMT (Kreutzer et al., 2019) extension for speech-to-text tasks such as automatic speech recognition and end-to-end speech translation. It inherits the core philosophy of JoeyNMT, a minimalist NMT toolkit built on PyTorch, seeking simplicity and accessibility. JoeyS2T’s workflow is self-contained, starting from data pre-processing, over model training and prediction to evaluation, and is seamlessly integrated into JoeyNMT’s compact and simple code base. On top of JoeyNMT’s state-of-the-art Transformer-based encoder-decoder architecture, JoeyS2T provides speech-oriented components such as convolutional layers, SpecAugment, CTC-loss, and WER evaluation. Despite its simplicity compared to prior implementations, JoeyS2T performs competitively on English speech recognition and English-to-German speech translation benchmarks. The implementation is accompanied by a walk-through tutorial and available on https://github.com/may-/joeys2t.

1 Introduction
End-to-end models recently have been shown to be able to outperform complex pipelines of individually trained components in many NLP tasks. For example, in the area of automatic speech recognition (ASR) and speech translation (ST), the performance gap between end-to-end models and cascaded pipelines, where an acoustic model is followed by an HMM for ASR, or an ASR model is followed by a machine translation (MT) model for ST, seems to be closed (Sperber et al., 2019; Ben vitovgli et al., 2021). An end-to-end approach has several advantages over a pipeline approach: First, it mitigates error propagation through the pipeline. Second, its data requirements are simpler since intermediate data interfaces to bridge components can be skipped. Furthermore, intermediate components such as phoneme dictionaries in ASR or transcriptions in ST need significant amounts of additional human expertise to build. For end-to-end models, the overall model architecture is simpler, consisting of a unified end-to-end neural network. Nonetheless, end-to-end components can be initialized from non end-to-end data, e.g., in audio encoding layers (Xu et al., 2021) or text decoding layers (Li et al., 2021).

ASR or ST tasks usually have a higher entry barrier than MT, especially for novices who have little experience in machine learning, but also for NLP researchers who have previously only worked on text and not speech processing. This can also be seen in the population of the different tracks of NLP conferences. For example, the “Speech and Multimodality” track of ACL 2022 had only a third of the number of papers in the “Machine Translation and Multilinguality” track. However, thanks to the end-to-end paradigm, those tasks are now more accessible for students or entry-level practitioners without huge resources, and without the experience of handling the different modules of a cascaded system or speech processing. The increased adoption of Transformer architectures (Vaswani et al., 2017) in both text (Kalyan et al., 2021) and speech processing (Dong et al., 2018; Karita et al., 2019a,b) has further eased the transfer of knowledge between the two fields, in addition to making joint modeling easier and more unified.

Reviewing existing code bases for end-to-end ASR and ST—for example, DeepSpeech (Hannun et al., 2014), ESPnet (Inaguma et al., 2020; Watanabe et al., 2020), fairseq S2T (Wang et al., 2020), NeurST (Zhao et al., 2021) and Speech-Brain (Ravanelli et al., 2021)—it becomes apparent that the practical use of open-source toolkits still requires significant experience in navigating large-scale code, using complex data formats, pre-processing, neural text modeling, and speech processing in general. High code complexity and a
lack of documentation are frustrating hurdles for novices. We propose JoeyS2T, a minimalist and accessible framework, to help novices get started with speech recognition and translation, to accelerate their learning process, and to make ASR and ST more accessible and transparent, that is directly targeting novices and their needs.

We hope that making more accessible implementations will also have trickle-down effects of making the research built on top of it more accessible and more linguistically and geographically diverse (Joshi et al., 2020). This effect has already been observed for the adoption of JoeyNMT for text MT for low-resource languages (Y et al., 2020; Camgoz et al., 2020; Zhao et al., 2020; Zacarias Márquez and Meza Ruiz, 2021; Ranathunga et al., 2021; Mirzakhalov et al., 2021). Furthermore, speech technology has an even higher potential for language inclusivity (Black, 2019; Abraham et al., 2020; Zhang et al., 2022; Liu et al., 2022).

2 Speech-to-Text Modeling

Automatic speech recognition and translation require mapping a speech feature sequence \( X = \{x_t \in \mathbb{R}^d\} \) to a text token sequence \( Y = \{y_t \in \mathcal{V}\} \). The continuous speech signal in its raw wave form is pre-processed into a sequence of discrete frames that are each represented as \( d \)-dimensional speech feature vectors \( x_t \), e.g., log Mel filterbanks at the \( i \)-th time frame. In contrast, a textual sequence is naturally composed of discrete symbols that can be broken down into units of different granularity, e.g. characters, sub-words, or words. These units then form a vocabulary, so in the above formulation \( y_t \) is the \( t \)-th target token from the vocabulary \( \mathcal{V} \). The goal of S2T modeling is then to find the most probable target token sequence \( \hat{Y} \) from all possible vocabulary combinations \( \mathcal{V}^* \):

\[
\hat{Y} = \arg\max_{Y \in \mathcal{V}^*} p(Y \mid X). \tag{1}
\]

2.1 Why End-to-End Modeling?

In conventional HMM modeling, the posterior probability \( p(Y \mid X) \) from Eq. 1 is decomposed into three components by introducing the HMM state sequences \( S = \{s_t\} \):

\[
p(Y \mid X) \approx \underbrace{p(X \mid S)}_{\text{Acoustic Model}} \underbrace{p(S \mid Y)}_{\text{Lexical Model}} \underbrace{p(Y)}_{\text{LM}}. \tag{2}
\]

The components correspond to an acoustic model \( p(X \mid S) \), a lexical representation model \( p(S \mid Y) \), and a language model \( p(Y) \). For practitioners, this means that three individual models need to be implemented, trained and combined. This comes with a large overhead, since each of them requires dedicated linguistic resources and experience in training and tuning. Attention-based deep neural networks have reduced this burden significantly since they implicitly model all three components in a single neural network, mapping \( X \) directly to \( Y \) (Chorowski et al., 2015; Chan et al., 2016).

2.2 Optimization

Most approaches to sequence-to-sequence learning tasks like MT use the cross-entropy (Xent) loss for optimization, and break the sequence prediction task down to a token-level objective. The posterior probability from above is modeled as the product of output token probabilities conditioned on the entire input sequence \( X \) and the target prefix \( y_{<t} \):

\[
p_{\text{xent}}(Y \mid X) := \prod_t p(y_t \mid y_{<t}; X). \tag{3}
\]

A popular alternative in ASR is to employ Connectionist Temporal Classification (CTC) loss (Graves and Jaitly, 2014). CTC uses a Markov assumption to model the transition of states similar to conventional HMM:

\[
p_{\text{ctc}}(Y \mid X) := \sum_{\mathcal{A}} \prod_t p(a_t \mid X), \tag{4}
\]

where \( \mathcal{A} \) denotes the set of valid alignments from \( X \) to \( Y \), \( a_t \in \mathcal{A} \) is one possible alignment at the \( t \)-th time step, and marginalizing the conditional probability \( p(a_t \mid X) \) over all valid possible alignments yields the sequence-level probability.

This CTC formulation is suitable to learn monotonic alignments between audio and text, and it also can handle very long sequences efficiently by solving dynamic programming on the state transition graph. The assumption of conditional independence at different time steps is a potentially harmful simplification which is compensated for by a token-level objective and by jointly minimizing cross-entropy and CTC loss (Hori et al., 2017; Watanabe et al., 2017). The final optimization objective in the JoeyS2T implementation is a logarithmic linear combination of the label-smoothed cross-entropy loss and the CTC loss defined above:

\[
\mathcal{L}_{\text{total}} := (1 - \lambda) \log p_{\text{xent}}(Y \mid X) + \lambda \log p_{\text{ctc}}(Y \mid X), \tag{5}
\]

where \( \lambda \in [0, 1] \) is an interpolation parameter.
3 Design Principles

Simplicity: We devoted considerable effort to keep JoeyS2T’s module structure simple and flat. It directly employs the PyTorch (Paszke et al., 2019) backend and has a low level of abstraction (details in Section 4.6). JoeyS2T has a minimal list of external dependencies that can be easily installed via the `pip` tool. Even for pre-processing, external dependencies on tools such as Kaldi (Povey et al., 2011) are avoided. For filterbank feature extraction, we use TorchAudio\(^3\) which is seamlessly integrated into PyTorch. In contrast to other toolkits, speech modules extended in JoeyS2T are only built for speech-to-text modeling. It does not implement speech enhancement, nor speaker detection or speech generation. While this might appear like a limitation, we believe that the reduction of functionalities to a carefully identified minimum for ST and ASR is the key for increased accessibility.\(^4\)

Accessibility: We also have written extensive documentation and walk-through tutorials to help newcomers become more familiar with speech technologies. JoeyS2T also provides pretrained models including configuration files which lower the barrier to get started. To guarantee the accessibility of the code, we open-sourced JoeyS2T under a very permissive license (Apache 2.0). The JoeyS2T developer community actively supports user questions and requests. We maintain an open platform to discuss bug fixes, possible extensions etc. All contributions are first automatically controlled by the internal unit tests and will manually be reviewed by our team.

Reproducibility: To ensure that the reported results are comparable and reproducible, we release models trained on publicly available data. Our evaluation metrics are described in detail (tokenization, punctuation handling etc.). All pre- and post-processing scripts are published with a data download path and explicit hyperparameter configurations. We track all code changes in our repository and provide version information which is often a critical factor for reproducibility as bug fixes can affect evaluation scores.

\(^3\)https://pypi.org/
\(^4\)https://github.com/pytorch/audio

4 Implementation and Usage

4.1 Hyperparameter Configuration

JoeyS2T sets up experiments based on a YAML-style configuration file which declares the whole pipeline, just like JoeyNMT. Processes are run in a Python interface without relying on external Bash or Perl scripts. In the configuration file, users can choose between the tasks MT (Machine Translation) or S2T (Speech-to-Text) in order to inform JoeyS2T about the input data type: audio or text. The hyperparameters of speech-related modules such as SpecAugment, 1d-Conv etc. can also be specified in the same configuration file.\(^5\)

4.2 Data Loading and Pre-processing

Source Audios: We separated computationally heavy pre-processing steps from model training, e.g., the conversion from raw wave forms to spectrograms by Fourier transformation. We employ the TorchAudio API to extract audio features in the pre-processing scripts. JoeyS2T includes modules for Cepstral Mean Variance Normalization (CMVN) (Viikki and Laurila, 1998) and SpecAugment (Park et al., 2019) by default. These are applied minibatch-wise before the input data are fed into the encoder.

Data Loading: As a precautionary measure to avoid memory allocation errors (which can happen for large audio inputs) we implemented on-the-fly data loading: we only store the path to the data in the iterator, and load the actual spectrogram features into memory every time a minibatch is constructed.

Target Texts: For target texts, we expect users to prepare a tokenization model independently and to specify the path to the trained tokenizer. Besides rule-based character-level tokenization and basic white space splitting, we currently support subword-nmt tokenizers (Sennrich et al., 2016) and SentencePiece tokenizers (Kudo and Richardson, 2018). Users can specify tokenizer options in JoeyS2T’s configuration file. During training, JoeyS2T applies text tokenization on the fly. Since the text length can be calculated only after tokenization, instance filtering by length is applied in this step. Thanks to this flexible on-the-fly tokenization, dynamic data augmentation methods i.e., BPE Dropout (Provilkov et al., 2020), SwitchOut (Wang

\(^5\)Sample configuration files for different datasets are available at https://github.com/may-/joey2t/configs
et al., 2018) or ADA (Lam et al., 2021) can be easily integrated.

4.3 Architectures
JoeyS2T supports a Transformer-based encoder-decoder architecture (see Figure 1). We reuse the self-attention encoder and decoder layers of JoeyNMT, and modify them in order to support speech-specific components.

Input Representations: Instead of converting token embeddings from discrete one-hot encodings to continuous vectors (as done for text input), we directly feed the sequence of filterbank vectors to the encoder. The embedding size in text-based JoeyNMT thus corresponds to the filterbank frequency size in JoeyS2T.

Encoder: The biggest difference to the original text-to-text Transformer architecture is the 1-dimensional convolution layer (1d-Conv) placed before the self-attention encoder. It compresses potentially redundant features along the time dimension in order to capture phonetic structures. Each 1d-Conv layer has a stride of 2. This further downsamples the sequence by a factor of $2^l$, where $l$ is the number of 1d-Conv layers. The reduction of the input length is essential for computation speed: Speech feature sequences are usually much longer than text token sequences, and the computational complexity of one self-attention block is $O(u^2 \cdot d)$ (Vaswani et al., 2017), where $u$ is the maximal input length (number of tokens in textual input, or number of time frames in speech input), and $d$ is the embedding size.

Decoder: We reuse the decoder construction of the original JoeyNMT code, but add one additional linear layer for the CTC loss on top of the self-attentive decoder layers.

Inference: We support greedy and beam search based on the token probability distributions. All inference enhancements introduced in JoeyNMT v2.0 such as repetition penalty, n-gram blocker, probability scoring, attention visualization of cross-attention heads in transformer layers, etc. are supported by JoeyS2T as well.

4.4 Evaluation Metrics
JoeyS2T supports Character F-score (ChrF) (Popović, 2015), BLEU (Papineni et al., 2002) and Word Error Rate (WER) based on Levenshtein distance (Navarro, 2001) as evaluation metrics for ASR and ST. We import sacrebleu\footnote{https://github.com/mjpost/sacrebleu} (Post, 2018) forChrF and BLEU, and editdistance\footnote{https://github.com/roy-ht/editdistance} (Hyryrö, 2001) for WER. In addition, perplexity and accuracy can be monitored during training on Tensorboard (Abadi et al., 2015).

4.5 Documentation and Tutorial
We follow the documentation strategy of JoeyNMT, which means that all extended functions have their own docstring and in-line comments for tensor shapes. Unit tests covering essential modules are automatically triggered on every commit to the repository.

In the hands-on tutorial, we present working examples for ASR and ST as Jupyter notebooks.\footnote{Demo video: https://youtu.be/bpBtq2jLolQ} The walk-through tutorial is self-contained and explains the whole pipeline: installation steps, data downloading, data pre-processing, configuration, model training/fine-tuning, inference and evaluation. We will keep the tutorial up to date with potential future API changes.

4.6 Code complexity
JoeyNMT exhibits the spirit of minimalism by aiming to achieve 80% of the output quality with 20% of a common toolkit’s code size (80/20 principle; (Pareto, 1896)). Table 3 gives statistics on code complexity. 

| nrefs:1|case:mixed|eff:no|tok:13a|smooth:exp|version:2.1.0
|----|---|---|---|---|---
|https://github.com/mjpost/sacrebleu
|https://github.com/roy-ht/editdistance
|Demo video: https://youtu.be/bpBtq2jLolQ
|https://github.com/espnet/espnet/tree/master/espnet2 (commit hash 039cc5d)
|https://github.com/pytorch/fairseq/tree/main/fairseq (commit hash ad3bec5)
|https://github.com/may-/joeys2t/tree/main/joeynmt (commit hash a80802a)
complexity. In terms of the numbers of Python files and code lines, JoeyS2T is 10–11 times more compact than ESPnet (Inaguma et al., 2020; Watanabe et al., 2020) and fairseq (Wang et al., 2020). However, both ESPnet and fairseq are general-purpose toolkits, covering a wide range of tasks beyond MT, ASR or ST, such as language modeling or speech synthesis, while JoeyS2T is designed for a speech-to-text tasks only. Yet JoeyS2T’s comment-to-code ratio is much higher than that of the competitors.

JoeyS2T offers a flat code structure in order to make debugging along the stack trace easier.
5 Experimental Results on Benchmarks

Despite its simplicity, JoeyS2T achieves a performance on standard benchmarks that is comparable to other high-functional speech-to-text toolkits.

5.1 ASR on LibriSpeech

LibriSpeech (Panayotov et al., 2015) is the de-facto standard English ASR benchmark that contains 960 hours of audiobooks in Project Gutenberg. The corpus is publicly available under the CC BY 4.0 license and many works set their goal to achieve state-of-the-art WER on its test splits.

Tables 1 present the results of models trained on 100h and 960h audio, respectively. JoeyS2T shows comparable performance with current Transformer-based models, which are generally outperformed by Conformer (Gulati et al., 2020) models.

5.2 ST on MuST-C

MuST-C (Cattoni et al., 2021) is a publicly available speech translation corpus built from English TED Talks. It consists of English transcriptions and translations into 14 languages, contributed by volunteers. We trained our model on the English-German subset of version 2, and evaluated the model both on version 1 and version 2, tst-COMMON, and tst-HE splits.

MuST-C is a challenging dataset due to its spontaneous speech that contains hesitations, disfluent utterances, etc. on the source side. Furthermore, the ground-truth target texts derived from the subtitles are also noisy. There are some additional descriptions of non-verbal information, i.e., “(applause)” “(laughter)”, or “♫ (music)”. Those are not actually pronounced in the source, but provided in the target, which makes learning more difficult. We normalized such noisy expressions and specified them as special tokens during the subword training, so that they are not tokenized into subwords but kept as single tokens. For the sake of reproducibility, we provide a preprocessing script for all normalization steps.

For ST tasks, we first pretrained ASR models and MT models using the gold transcriptions. Then we initialized the encoder layers of an end-to-end ST model with the pretrained ASR encoders and the decoder layers with the pretrained MT decoders, and further trained it on the end-to-end ST task.

The ST results can be found in Table 2. JoeyS2T shows competitive results, both in end-to-end scenarios and in a cascade using the same pre-trained models. We also include the ASR and MT pretraining results for reference.

6 Conclusion & Future Work

We described JoeyS2T, an extension of the JoeyNMT toolkit to the spoken language processing tasks ASR and ST. JoeyS2T is characterized by its minimalist design, prioritization of simplicity, accessibility and reproducibility in its code and documentation. The code is self-contained and requires minimal prior experience with speech or language processing. In benchmark evaluations, JoeyS2T performed comparable or superior to other ASR or ST code bases, while having much lower code complexity.

While its functionality is kept minimal, support for state-of-the-art architectures such as wav2vec and Conformer might be desired for future extensions.

Limitations

The limitations of our work mainly concern the reproducibility of comparable state-of-the-art re-
sults. First, there are many different preprocessing variants which are quite complex (length filtering, speed shift, lowercasing, punctuation normalization etc.) and not always clearly documented. Second, the same problem appears in evaluation. There is no commonly accepted evaluation scheme (including lower-cased vs. true-cased results, with or without punctuation, etc.). While the sacrebleu library is a first step to addressing this problem in MT, we believe that the speech processing community also needs such efforts to standardize speech-to-text evaluation.

Since the goal of our work is not to present a new state-of-the-art in speech-to-text modeling, we did not invest a large effort into hyperparameter tuning, but only varied three different random seeds in our setup, and used the default settings for competitor systems.

Acknowledgements

We would like to thank the members of the StatNLP group at Heidelberg University and the AIMS Senegal students for their feedback on the tutorial. Furthermore, we appreciate the discussions with the Masakhane community in the early stages of the toolkit development. We also thank Yaraku Inc. for the opportunity to publish JoeyS2T tutorial articles.

References


Wilhelmina Nekoto, Vukosi Marivate, Tshinondiwa Matsila, Timi Fasubaa, Taiwo Fagbohungbe, Solomon Oluwole Akinola, Shamsuddeen Muhammad, Salomon Kabongo Kabenamuula, Salomey Osei, Freshia Sackey, Rubungo Andre Niyongabo, Ricky Macharm, Perez Ogayo, Orevaoghene Ahia, Musie Meressa Berhe, Mofetoluwa Adeyemi, Masabata Mokgesi-Selinga, Lawrence Okegbemi, Laura Martinus, Kolawole Tajudeen, Kevin Degila, Kelechi Ogueji, Kathleen Siminyu, Julia Kreutzer, Jason Webster, Jamaal Toure Ali, Jade Abbott,
tional Linguistics.


Alex Graves and Navdeep Jaitly. 2014. Towards end-
to-end speech recognition with recurrent neural net-
works. International conference on machine learn-
ing, pages 1764–1772.

Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei  
Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruom-
Speech Communication Association, pages 5036–5040, Shanghai, China. ISCA.

Awni Hannun, Carl Case, Jared Casper, Bryan Catan-
zero, Greg Diamos, Erich Elsen, Ryan Prenger, San-


Heikki Hyyrö. 2001. Explaining and extending the bit-


Jacob Kahn, Ann Lee, and Awni Hannun. 2020. Self-
training for end-to-end speech recognition. In 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 7084–7088. IEEE.


Shigeki Karita, Nelson Yalta, Marc Delcroix, Atsunori Ogawa, and Tomohiro Nakatani. 2019b. Improving transformer-based end-to-end speech recognition with connectionist temporal classification and language model integration. In INTER-
SPEECH.


Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword token-


eering and Informatics (CISP-BMEI), pages 439–444. IEEE.

Xian Li, Changhan Wang, Yun Tang, Chau Tran, Yuqing Tang, Juan Pino, Alexei Baevski, Alexis Conneau, and Michael Auli. 2021. Multilingual speech trans-
lation from efficient finetuning of pretrained models.


