Joint Source-Channel Coding for Semantic Text Transmission: Communication with Built-in Translation

Jason Ning, Joshua Ning

Project Statement

In a typical end-to-end (E2E) communication system, source and channel encoder-decoder pairs are designed to compress and protect the signal when transmitted through a noisy channel. Traditionally, source coding and channel coding are designed and implemented separately. Source coding compresses the input signal, removing redundant information in the source to condense the information. Channel coding introduces redundancy to minimize errors when transmitting the information through a noisy channel. Optimality of the source and channel code is well studied, and current methods can achieve theoretical bounds when the block size is sufficiently large. Additionally, as shown by Shannon with his source channel separation theorem, the separation of the source coding and channel coding can be achieved without losing optimality in stationary and ergodic channels.

Joint Source-Channel Coding (JSCC) is an approach that combines source coding and channel coding into a single process. JSCC attempts to combat the challenge of fast-changing channel conditions and finite block length of an E2E system that has the potential to achieve lower latency and higher performance in the low SNR regime.

Deep JSCC is a subfield of JSCC, aiming to solve the problem of JSCC with deep neural networks instead of handcrafted coding schemes. Current state-of-the-art text-based Deep JSCC, DeepSC, [1] is based on transformers, which have to cope with semantic distortion, where the receiver might switch the words in the original message with another similar meaning word. Semantic distortion poses a barrier for trustworthy information transmission and cannot be easily solved because of the underlying DNN architecture.

In our project, we embrace semantic distortion by training the decoder to output text in another language. Translation is an ideal application for Deep JSCC, as it naturally accommodates semantic variations inherent in both communication and language conversion. Since the translation is happening within the communication stack, the proposed method has the potential to be faster than the current local and cloud-based translation pipeline. To the best of our knowledge, no existing work has investigated JSCC for simultaneous transmission and translation, making this an unexplored direction in the field.

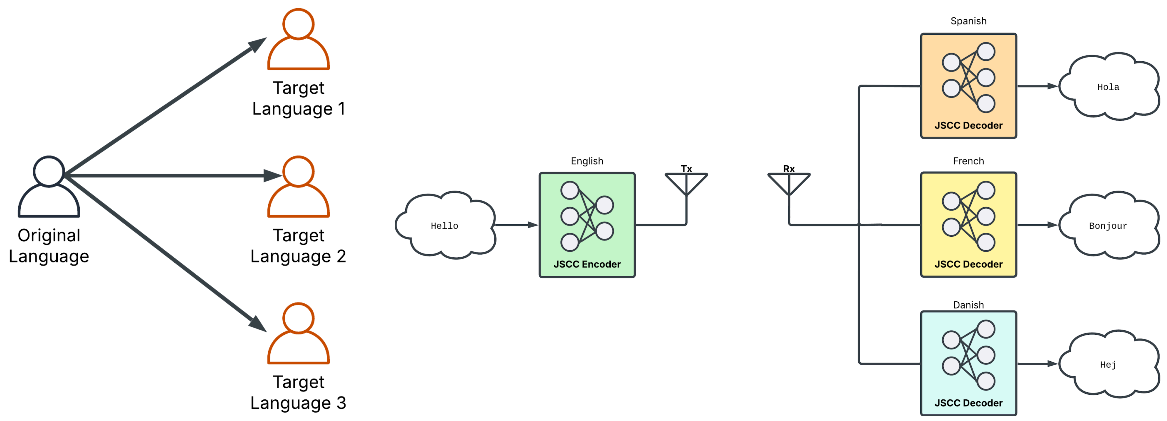


Figure 1: Concept of connecting 1 encoder to multiple decoders to achieve multi-language translation in DeepSC-T

The goal of this project is to train a text-based Deep JSCC model, which we call DeepSC-T (T stands for translation), with a simulated AWGN channel. We want to demonstrate the Deep JSCC's ability to perform translation tasks, as well as the ability to translate 1 input language to multiple output languages simultaneously, shown in Figure 1. We present different training methods and compare their training speed and accuracy. The target embedded system will be a Software Defined Radio (SDR) working with either a CPU or a small GPU like the Jetson Nano.

Our DeepSC-T model was able to achieve a BLEU score > 0.9 and Word Error Rate (WER) < 4%. The speed of encoding/decoding is similar to the DeepSC paper [1], achieving around 7.5 ms E2E latency on an RTX 4070 GPU. We developed a training method that has the ability to train more than 1 decoder, and it has demonstrated a significant decrease in training time compared to conventional methods.

Project Challenges

There are three challenges to this project. First, we need to prove that translation is possible for the DeepSC framework by building our own implementation called DeepSC-T. Second, we want to improve the training efficiency by evaluating the effects of transfer learning and our custom training algorithms (**Algorithm 1**). Third, we aim to reduce the GPU usage during the training phase for faster training.

Embedded Systems

The target embedded system is a software-defined radio (SDR). The SDR imposes a limitation on our machine learning model because the transmit power needs to be normalized between -1 and 1. We implemented the model so that the power normalization is done before the transmission to the channel to ensure compatibility of our model on any SDR. Furthermore, SDR works with a computer that does not guarantee GPU availability. Therefore, we performed inference and profiling of our model on both CPU and GPU to verify feasibility.

Machine Learning

We constructed the DeepSC-T based on the original DeepSC architecture shown in **Figure 2**. It mainly consists of a Transformer Encoder, a Dense Layer, a Channel Layer, and a Transformer Decoder. The Transformer Encoder obtains the semantics of the sentence, compressing the information, acting as a source encoder. The Dense Layer adapts the output of the transformer according to the current channel condition, which is similar to the channel encoder in function. The decoder performs the inverse function of the encoder, trying to estimate the input S.

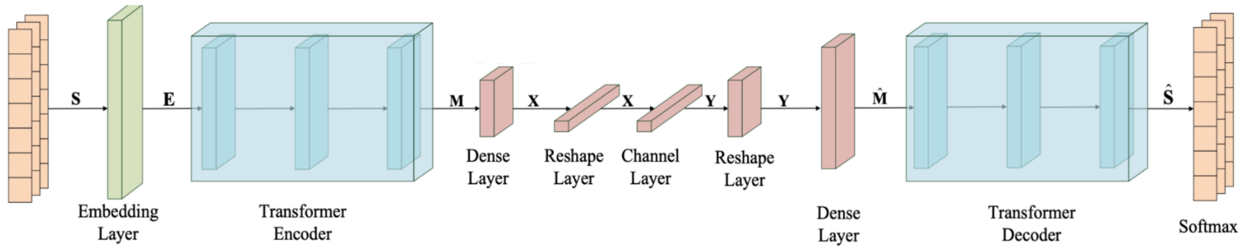


Figure 2: Architecture of DeepSC [1]. We implemented this architecture from scratch to build DeepSC-T

Hardware and Software Integration

We planned to use GNU Radio to interface our implementation of Deep JSCC with the SDR and test the performance of our models in real-world scenarios. However, due to the lack of time, this step was not completed by the project deadline. The implementation should be relatively simple, and a block diagram is shown below for the GNU Radio Architecture.

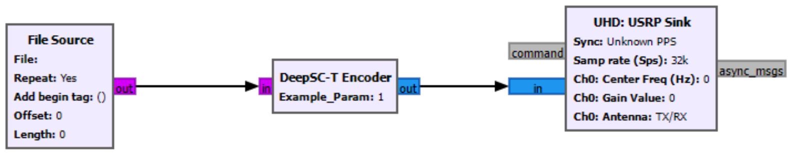


Figure 3: An illustration of DeepSC-T encoder implementation in GNURadio. A complex number is sent to the channel, the complex number represents the I/Q component of a signal.

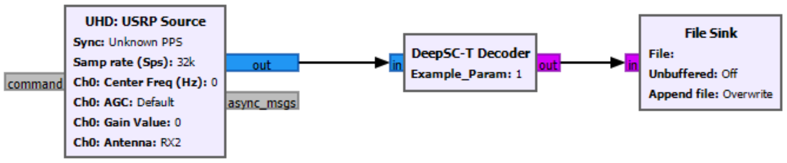


Figure 4: An illustration of DeepSC-T decoder implementation in GNURadio. An estimation of the input is represented with a "string" type.

Implementation

We implemented DeepSC-T from scratch using the PyTorch library in Python. The dataset used is Europarl [2], a multilingual dataset consisting of many European languages. Data preprocessing and tokenization were done prior to training to ensure each training sample contains less than 27 tokens (25 tokens for the source sentence and 1 token for the start of sentence and 1 token for the end of sentence).

The number of decoders in DeepSC-T is user-definable, enabling it to be trained and evaluated in several different ways. Similar to DeepSC(traditional method), DeepSC-T can train with a single encoder and decoder. Furthermore, the users can train DeepSC-T with decoders for many languages simultaneously, following **Algorithm 1**. The ability to train multiple decoders simultaneously offers some performance benefits, as we will discuss in the evaluation section.

Training DeepSC-T with multiple decoders is done with **Algorithm 1**. A dataset and decoder are selected at the start of the epoch in a round robin fashion. Then the encoder and selected decoder are trained on the selected dataset. After iterating through all the decoders in the list, evaluation is performed on the model to assess convergence.

Algorithm 1 Round-Robin Decoder Training

```
1:  Initialize array dataset
2:  Initialize array decoder
3:  Initialize numEpochs
4:  for counter ← 0 to numEpochs−1 do
5:      idx ← counter mod len(decoder)
6:      currentDataset ← dataset[idx]
7:      currentDecoder ← decoder[idx]
8:      Train currentDecoder with currentDataset
9:      if (counter + 1) mod len(decoder) == 0 then
10:         for each dec in decoder
11:             Evaluate dec
12:
13:     end if
14: end for
```

Algorithm 1: A decoder is selected in a round robin fashion from a set of decoders. The decoder is paired up with the encoder during training, and we choose another decoder for the next epoch.

To save VRAM on the GPU, the user also has the option to unload the decoders that are not currently training to the CPU. This step allows for a larger batch size and faster training time.

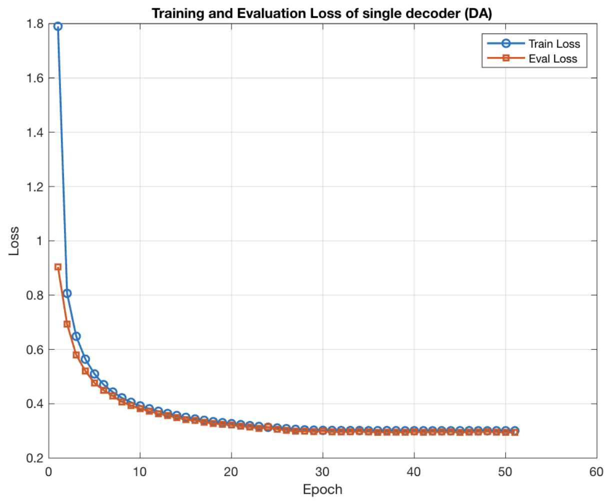
To test the modularity of DeepSC-T, we apply transfer learning techniques. An initial encoder and decoder pair are trained for translation across a source and target language(s). After model convergence, the encoder is frozen, and a new decoder is trained with a new dataset for a new target language.

Evaluation

We trained and evaluated several models on the Europarl dataset and demonstrated that DeepSC-T has acceptable translation accuracy. We also compare the convergence speed of the model when using **algorithm 1** with traditional training methods. Additionally, we evaluated the speed of convergence for the transfer learning method with the encoder from both **algorithm 1** and traditional training methods.

First, we evaluate the convergence speed of DeepSC-T when there is a single encoder and decoder, we call this DeepSC-T1, with 1 denoting the number of decoders. As shown in **Figure 5**, the train loss and evaluation loss converge after around 30 epochs. This proves that DeepSC-T1 can indeed be used for translation. In **Figure 6**, we compare the convergence speed of DeepSC-T1 across two languages, Danish and French. We found that both models converge at around 30 epochs, however, the evaluation loss of French is smaller than the evaluation loss of Danish. We think this could be attributed to the fact that the French language has a smaller dictionary size, hence, easier to decode. The dictionary sizes for each language are listed in **Table 1**

We then evaluated the convergence speed of DeepSC-T3 (single encoder and three decoders). **Figure 7** plots the evaluation loss of DeepSC-T3 when trained with three different languages, Danish, Spanish, and French. We see that the three decoders all converge after around 30 epochs. As shown in **Algorithm 1**, only one decoder is trained in each epoch. This implies that the decoders of DeepSC-T3 converged around 3 times faster than DeepSC-T1. Additionally, as shown in **Table 2** and **Figure 8**, DeepSC-T3 achieves this faster convergence without much compromise in common translation metrics such as the BLEU and WER scores. In **Figure 8**, the Danish evaluation loss for DeepSC-T3 is higher than DeepSC-T1, especially at earlier epochs. We believe this is mainly an artifact due to the evaluation method. Since Danish was the first encoder selected during the training phase of DeepSC-T3 following **Algorithm 1**, the encoder would have been optimized for the Spanish and French decoders before the first evaluation phase. This led to worse evaluation loss for the Danish decoder, which is more pronounced in the earlier epochs.



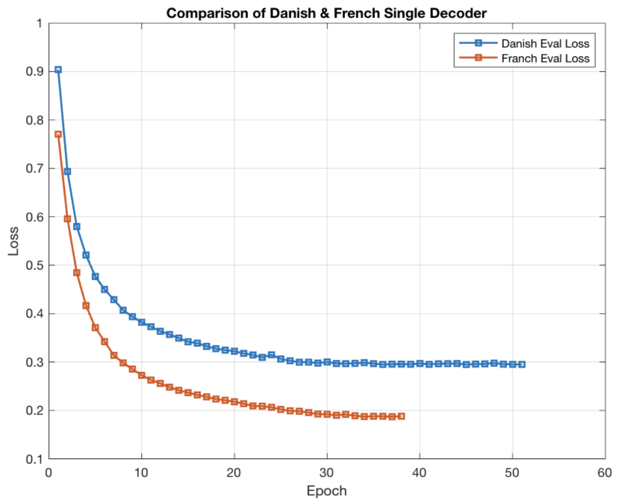
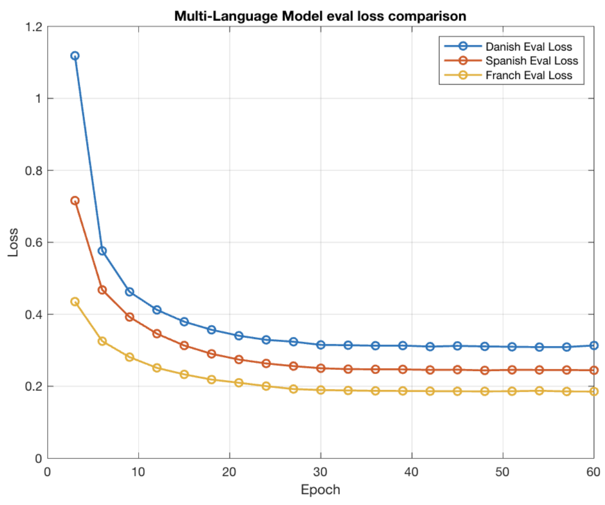


Figure 5 (left): Train and evaluation loss of DeepSC-T1.DeepSC-T1 on the English to Danish dataset is ~30 epochs. **Figure 6** (Right): The Evaluation loss of DeepSC-T1 on the English to Danish dataset and the English to French dataset is ~30 epochs. English to French was able to achieve lower loss due to a smaller dictionary size.



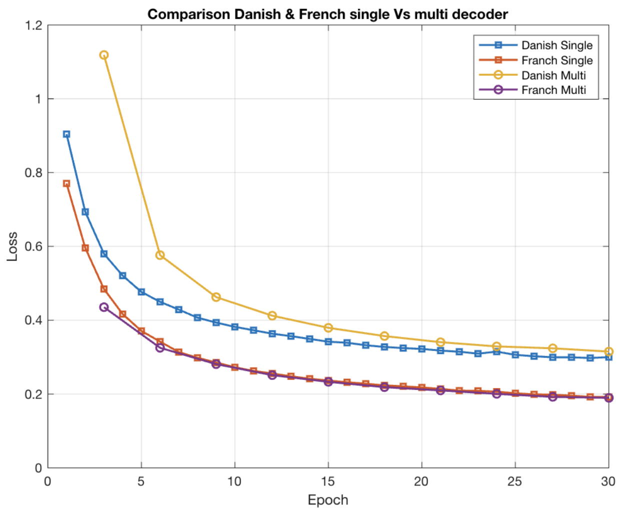


Figure 7 (left): The evaluation loss of DeepSC-T3 on English to Danish, Spanish, and French datasets. Convergence speed of DeepSC-T3 is ~30 epochs, which represents a 3x increase in convergence speed as compared to DeepSC-T1, since only one decoder is trained per epoch.
Figure 8 (Right): The evaluation loss of DeepSC-T3 compared to DeepSC-T1. Both converge after ~30 Epochs.

Furthermore, we evaluated the convergence of transfer learning methods. We froze the encoder for both DeepSC-T1 and DeepSC-T3 obtained from the above methods and trained a new decoder on the dataset for English to Italian. We call DeepSC-T1 and DeepSC-T3 trained with transfer learning DeepSC-T1TL and DeepSC-T3TL, respectively. We want to see if transfer learning could benefit from faster convergence. **Figure 9** shows that there is no benefit of using transfer learning methods.

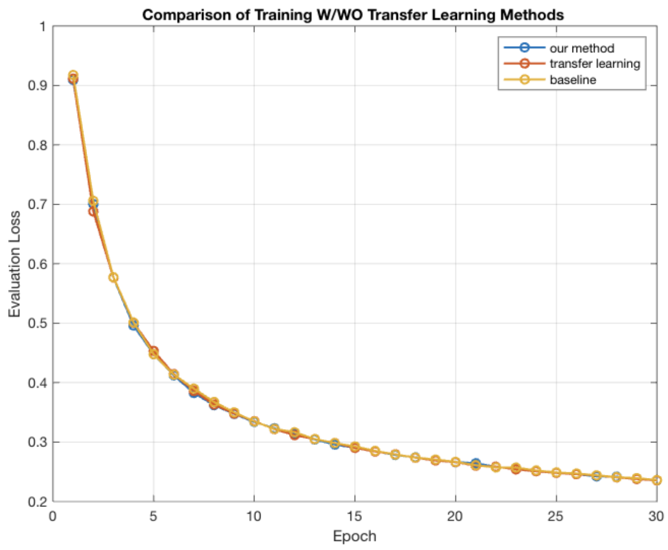


Figure 9: Comparison between DeepSC-T3TL (our method, blue), with DeepSC-T1TL (transfer learning, red), and training encoder and decoder from scratch (baseline, yellow)

	English (en)	Danish (da)	Spanish (es)	French (fr)	Italian (it)
Dictionary Size	53,151	143,410	88,480	70,726	89,224

Table 1: Dataset Dictionary Size Across Different Languages

	DeepSC-T1 en->it Epoch 30	DeepSC-T1 en->fr Epoch 30	DeepSC-T3 en->fr Epoch 27	DeepSC-T3TL en->it Epoch 30
BLEU Score	0.9015	0.9212	0.9206	0.9003
WER Score	0.03814	0.0307	0.0309	0.0386

Table 2: Performance Metrics Comparison of Transfer Learning

On single-sample inference, DeepSC-T1 uses less than 0.5 GB of VRAM. The average time (on 10,000 samples) to encode is 3.1 ms, and the mean time to decode is 4.4 ms on the GPU (RTX 4070). On the CPU (AMD Ryzen 7 7700X 8 Core), it takes 2.4 ms to encode and 7.6 ms to decode.

Source Code

Source code can be found at the following GitHub repository. Follow the README file to obtain the Europarl dataset, and we also provide this Google Drive Link to our pre-trained models.

Limitations and Future Works

Deep JSCC methods for videos and images are more explored topics because the large amount of data stream tends to consume more bandwidth. Deep JSCC models for text are criticized for their inability to decode the original message. This was also observed in our implementation of the DeepSC architectures. Furthermore, we observed that the word error is not limited to the semantic distortion mentioned in the original paper [1]; most of the errors appear to be random.

As future work, we want to benchmark commonly used source and channel encoding/decoding schemes and compare them with the CPU run-time of our DeepSC-T models. We also want to explore other hybrid model architectures, such as CNN + Transformer, to decrease model complexity, or Vision Transformer to support image transmission. Lastly, we want to dedicate more time to looking into transfer learning and its evaluation methods. The result from **Figure 9** is quite surprising because we expect the transfer learning models to converge faster.

Individual Contributions

Jason Ning: Paper research, build DeepSC architecture from scratch, plot data in MATLAB to generate figures, evaluation metrics such as the BLEU and WER score, draw illustrations, debug, and work on the report.

Joshua Ning: Paper research, data set gathering, data pre-processing, data loaders and train/eval loops, transfer learning, profile DeepSC-T on both the CPU and GPU, draw illustrations, debug, and work on the report.

References:

[1] H. Xie, Z. Qin, G. Y. Li, and B.-H. Juang, ‘Deep Learning Enabled Semantic Communication Systems’, IEEE Transactions on Signal Processing, vol. Early Access, 2021.

[2] P. Koehn, ‘Europarl: A Parallel Corpus for Statistical Machine Translation’, in Proceedings of Machine Translation Summit X: Papers, sep # ‘ 13-15’, year = ‘2005’, address = ‘Phuket, Thailand’, url = ‘https:/aclanthology.org/2005.mtsummit-papers.11/’, pages = ‘79--86’, abstract = ‘We collected a corpus of parallel text in 11 languages from the proceedings of the European Parliament, which are published on the web. This corpus has found widespread use in the NLP community. Here, we focus on its acquisition and its application as training data for statistical machine translation (SMT). We trained SMT systems for 110 language pairs, which reveal interesting clues into the challenges ahead.’ 2005, pp. 79–86.