Exploring Regularization in Speech-to-Text Translation

Anonymous ACL submission

Abstract

Although end-to-end speech-to-text translation has many advantages over its cascaded counterpart, it typically suffers from the scarcity of speech-text paired data. Recently, researchers have attempted to overcome this shortcoming by utilizing auxiliary tasks, such as Automatic Speech Recognition (ASR) task and Machine Translation (MT) task, via multi-task learning (MTL). In this paper, we formulate MTL from a regularization perspective and explore how sequences can be regularized within and across modalities. By thoroughly investigating the effect of consistency regularization (different modality) and R-drop (same modality), we show how the two consistency measures respectively contribute to the *total* regularization. We also demonstrate that the coefficient of MT loss in the MTL setting with MT task can serve as a source of regularization. With these three sources of regularization, we introduce the optimal regularization contour in the high-dimensional space, called the regularization horizon. Experiments show that tuning the hyperparameters within the regularization horizon all similarly achieves near state-of-the-art (SOTA) performance on the MuST-C dataset.

1 Introduction

Speech translation (ST) aims to translate input speech into target text in a different language. This task comprises of two sub-tasks – transcribing the source speech into source text and translating the source text into the target text. The traditional approach involves a cascade of two models, each responsible for each sub-task. Recently, an end-to-end approach of directly producing the translated text from the source speech in a single model has gained popularity (Bérard et al., 2016, 2018; Bahar et al., 2021; Wang et al., 2020; Barrault et al., 2023; Radford et al., 2023) because the end-to-end approach mitigates inherent problems with using two models sequentially: error propagation, increased latency and model size.

However, the end-to-end approach is limited by the scarcity of paired speech-text translation data. Many augmentation methods (Bérard et al., 2018; Pino et al., 2019) have been suggested to overcome this shortage, along with the recent approaches (Barrault et al., 2023; Radford et al., 2023) that collect and mine large-scale data. In both cases, acquiring paired data for the speech translation task at scale still remains a challenge.

Distilling knowledge from existing data is another approach to overcoming data constraints. Transfer learning from pre-trained models has shown to be effective in leveraging knowledge from tasks with more abundant data, such as ASR or MT tasks (Wu et al., 2020; Li et al., 2021; Xu et al., 2021; Li and Niehues, 2022; Gállego et al., 2021; Le et al., 2021). Multi-task learning with auxiliary tasks has also recently been widely adopted to overcome the data shortage (Anastasopoulos and Chiang, 2018; Tang et al., 2021; Ye et al., 2021; Fang et al., 2022).

Particularly in an ST and MT multi-tasking setting, the model learns to converge the representation of the two input modalities that are semantically similar and output them in a textual format. Hence, it may be helpful to have the speech and text inputs of similar semantic meaning embedded into a common latent space within the model. In this regard, multi-tasking with the MT task allows more room for explicit representation matching at various latent depths of the model for performance improvement than the ASR task.

In this paper, we seek to explore the effects of ensuring consistency between latent representations at multiple stages of the model's forward pass and across different modalities in an MTL setting with MT task. In short, our contributions are as follows:

- We conduct extensive studies on various methods of consistency training and empirically

verify that applying Kullback-Leibler (KL) divergence loss at the final softmax output is the most effective, regardless of the chosen consistency measure

- We draw similarities between different consistency measures and hypothesize that the MT loss in MTL acts as a regularization force in the presence of consistency objective functions

- We find the contour with optimal total regularization (the regularization horizon) and show that the model achieves similarly near SOTA performance within the regularization horizon

2 Related Works

Multi-Task Learning (MTL) has been explored on many dimensions in ST. Particularly, multi-task learning with MT tasks has been proven to be an effective way to improve the performance of the end-to-end ST model. Fang et al. (2022) interweaves speech and text embeddings through a mixup approach during training while Zhang et al. (2023b) aims to encode speech and text embeddings together by concatenating them before inputting into the model. Other works propose to maximize the effects of multi-task learning by directly enforcing intermediate text and speech embeddings to become similar to one another. However, this is non-trivial as the two embeddings have varying lengths. Han et al. (2021) and Tsiamas et al. (2023) overcome this problem by projecting the embeddings into a fixed-size sequence of vectors. Ye et al. (2022) and Ouyang et al. (2023) align the embeddings via contrastive learning on a sequence level and token level respectively. Despite the differences in their approach, most MTL experiments show that providing the model with careful guidance on matching the two modalities helps improve the model performance. In this paper, we are particularly interested in achieving this with minimal modifications to the architectural design.

Consistency regularization is a regularization method to ensure consistency in results or embeddings produced by a model that is robust to small perturbations in the input. Use cases in computer vision (Zhang et al., 2019) and natural language processing (Xie et al., 2020) have proven their effectiveness. More recently, in ST, Han et al. (2023); Fang and Feng (2023); Lee et al. (2023) utilized the notion of consistency in different ways to achieve SOTA performance on ST benchmark datasets.

R-drop is a consistency training technique that reduces the inconsistency arising from applying dropout during training (Wu et al., 2021). Dropout activates different partials of a given model, yielding different outputs for the same given input. This essentially induces sub-models during training that may be inconsistent with the full model used during inference. R-drop minimizes this discrepancy by ensuring consistency between the sub-models, and hence improves the model performance during inference. This is also similar in spirit to consistency regularization in that the dropout module perturbs the model output by tweaking the activations of the model weights, and R-drop is applied to counter this inconsistency. Gao et al. (2024) explore applications of R-drop to the multi-task learning of ST and MT. Lee et al. (2023) applies R-drop to MT task to prevent catastrophic forgetting when fine-tuning pretrained MT model for ST task. Unlike these two papers, we omit the exploration of consistency within the MT task as we considered optimizing on the MT task as being irrelevant to maximizing ST performance.

3 Method

3.1 Problem Statement

The training data for end-to-end speech translation task ($\mathcal{D}$) comprises of speech ($\mathbf{x_s}$), transcript ($\mathbf{x_t}$), and translation ($\mathbf{y}$), formally $\mathcal{D} = \{(\mathbf{x_s}, \mathbf{x_t}, \mathbf{y})\}$. $\mathbf{x_s} = (x_{s1}, \ldots, x_{sn})$ is the input audio in the raw waveform, $\mathbf{x_t} = (x_{t1}, \ldots, x_{tm})$ is the transcription of the input audio in text form, and $\mathbf{y} = (y_1, \ldots, y_k)$ is the translation of $\mathbf{x_t}$. We seek to explore how $\mathcal{D}_{\text{MT}}$ $\{(\mathbf{x_t}, \mathbf{y})\}$ can be used to enhance the performance on $\mathcal{D}_{\text{ST}}$ $\{(\mathbf{x_s}, \mathbf{y})\}$ through consistency training.

3.2 Model architecture

We use the transformer architecture as the backbone for both ST and MT models. We use pretrained HuBERT (Hsu et al., 2021) base model as our speech encoder to extract a sequence of latent variable tokens from raw audio waveform, which is then fed into the subsequent transformer encoder-decoder structure. The subsequent encoder and decoder are shared for both ST and MT tasks unless otherwise noted. Figure 1 illustrates this process.

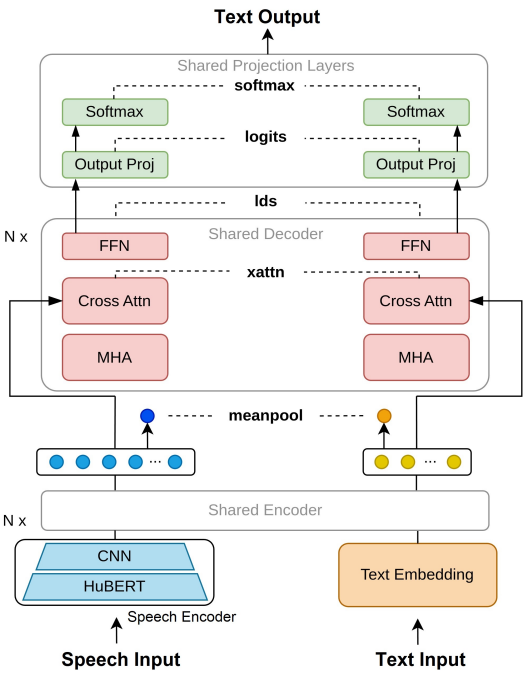

Figure 1: Consistency Training

3.3 Consistency Training

We use the term consistency for two distinct purposes. Firstly, we consider the consistency between two distinct forward passes of the same speech input. Due to the dropout module, every forward pass induces different representations. Secondly, we consider the consistency between the representations of speech and text inputs along the model's forward pass. Notably, each forward pass in this case also passes through different sub-models induced by the dropout module. To avoid confusion, we refer to the former consistency as R-drop and the latter as consistency regularization.

In this paper, we explore how consistency can be enforced in both cases. We do this by introducing the distance between the two as the auxiliary loss function. Specifically, we define the loss function of our training objective as the following.

$$\mathcal{L} = \mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{con}} \tag{1}$$

$\mathcal{L}_{\text{ce}}$ is the cross entropy loss for both ST and MT, defined as:

$$\mathcal{L}_{\text{ce}} = -\alpha_s \mathbf{y} \cdot \log P(\mathbf{y}|\mathbf{x_s}) - \alpha_t \mathbf{y} \cdot \log P(\mathbf{y}|\mathbf{x_t}). \tag{2}$$

$\mathcal{L}_{\text{con}}$ is the consistency loss, which is the addition of consistency regularization loss, $\mathcal{L}_{\text{cr}}$, and R-drop loss $\mathcal{L}_{\text{rdrop}}$, each defined as:

$$\mathcal{L}^{ij}_{\text{cr}} = \alpha^{ij}_{cr} D_j(f^i_s(\mathbf{x_s}), f^i_t(\mathbf{x_t})), \tag{3}$$

$$\mathcal{L}^{ij}_{\text{rdrop}} = \alpha^{ij}_{rd} D_j(f^i_s(\mathbf{x_s}), f^i_s(\mathbf{x_s})). \tag{4}$$

α's are the loss ratio hyperparameters which we fix $\alpha_s = 1$. There are two additional design choices in this scheme – which embeddings do we compare (f^i) and how do we define the distance (D_j). $f^i_{s/t}$ indicates speech/text embeddings at the i-th layer. We consider the encoder output (enc), cross attention (x-attn), last decoder state (lds), logits, and the softmax output as such embeddings. These points of optimization are depicted in Figure 1. Note that we do not apply R-drop to enc and x-attn because the dropouts in the subsequent layers will diverge the representations and dilute the effects of R-drop.

We use three different metrics to measure the distance between the embeddings. They are the mean-square-error (MSE)

$$D_{\text{MSE}}(\mathbf{x}, \mathbf{y}) = \frac{1}{d}||\mathbf{x} - \mathbf{y}||^2, \tag{5}$$

where d is the dimension of $\mathbf{x}$, the cosine similarity (COS)

$$D_{\text{COS}}(\mathbf{x}, \mathbf{y}) = 1 - \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||\,||\mathbf{y}||}, \tag{6}$$

and the Kullback–Leibler (KL) divergence

$$D_{\text{KL}}(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\left(\mathbf{x} \cdot \log \frac{\mathbf{x}}{\mathbf{y}} + \mathbf{y} \cdot \log \frac{\mathbf{y}}{\mathbf{x}}\right). \tag{7}$$

The division and log operation of vectors are done componentwise. Note that the KL divergence loss applied to the softmax output is in effect equal to the online knowledge distillation setting introduced in Tang et al. (2021).

3.4 Cross-modal consistency regularization

In the ST/MT MTL setting, the encoder suffers from the modality discrepancy because the representations for speech and text are not identical in length. We overcome this by applying average pooling to the encoder output embeddings as was done in Ye et al. (2022). However, as noted by Duquenne et al. (2023), the MT and ST loss terms ensure the uniqueness of vectors and hence prevent possible modal collapse. As such, we do not apply contrastive learning. The consistency regularization terms at the decoder level (x-attn, lds, logits, and softmax) can be applied directly since both ST and MT share common output modality, text.

3.5 Additional regularization for speech

Since speech modality has more degrees of freedom compared to text, such as silent interval, repetition, intonation, and non-verbal information, it may require more generalization than text. Therefore, we apply the DropKey scheme (Li et al., 2023) on top of the above consistency measures. Our DropKey scheme masks a single span of keys of length sampled uniformly from 1 to 10 (Shen et al., 2020; Zhang et al., 2022).

4 Experiments

4.1 Datasets

ST Datasets We conduct experiments on the MuST-C (Di Gangi et al., 2019) dataset. Our main investigation on consistency training is conducted on German (De),and we apply our best method to Spanish (Es), French (Fr) and Italian (It).[1]

MT Datasets The transcription and translation pairs in MuST-C dataset is used as our MT dataset. We also use WMT (Buck and Koehn, 2016) for languages (De/Es/Fr) and OPUS100 (Zhang et al., 2020) for It as our external MT dataset during pre-training.

4.2 Experimental setup

Our training set up generally follows the MTL model introduced in Fang and Feng (2023) as they provide a strong baseline to test our methods.

We train our models using the `fairseq` (Ott et al., 2019) framework. Our models are evaluated using BLEU scores (Papineni et al., 2002). During training, models are validated and early-stopped using the dev set of MuST-C. We report our final scores on `tst-COMMON` in Table 4 and use paired bootstrap resampling (Koehn, 2004) to compute the statistical significance of our results. We refer readers to Appendix A for further details.

5 Results

5.1 Consistency Regularization

We first investigate the effects of consistency regularization when varying the compared embeddings and the distance metrics (the i and j's of Eq. (3)). We consider five embeddings across the network with three different distance metrics (Eqs. (5), (6), and (7)) and compare their results with the baseline model trained only with ST and MT cross-entropy losses. Please refer to Figure 1 for the five embeddings' position in the neural network. The BLEU scores for the en-de ST task are shown in Table 1. Note that the expanded setting utilize the external MT dataset during pre-training.

	Experiment	Loss	Base	Expand
	baseline	$\mathcal{L}_{\text{ce}}$	25.77	28.20
con. reg.	enc-MSE	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{encMSE}}$	26.01	28.35
	enc-COS	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{encCOS}}$	25.93	28.32
	xattn-MSE	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{xattnMSE}}$	25.93	28.45
	xattn-COS	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{xattnCOS}}$	26.11	28.72
	lds-MSE	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{ldsMSE}}$	26.39	28.89
	lds-COS	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{ldsCOS}}$	26.33	28.67
	logits-MSE	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{logitsMSE}}$	26.78	28.88
	softmax-MSE	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{softmaxMSE}}$	**27.45**	29.34
	softmax-KL	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{softmaxKL}}$	27.32	**29.59**
R-drop	lds-MSE	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{rdrop}}^{\text{ldsMSE}}$	26.96	28.84
	lds-COS	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{rdrop}}^{\text{ldsCOS}}$	26.44	28.30
	logits-MSE	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{rdrop}}^{\text{logitsMSE}}$	27.12	28.79
	softmax-MSE	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{rdrop}}^{\text{softmaxMSE}}$	27.49	29.45
	softmax-KL	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{rdrop}}^{\text{softmaxKL}}$	**28.40**	**29.66**

Table 1: BLEU scores with various consistency regularization and R-drop on MuST-C en-de dataset.

The results suggest that introducing consistency regularization indeed enhances the performance of the ST, regardless of the selected embedding or metric. Moreover, enforcing consistency between embeddings that are closer to the final output layer results in better performance for both base and expanded settings. That is, the result of the later rows are better than that of the earlier ones in Table 1. This is consistent with Pham et al. (2019), where they note that enforcing consistency on the embeddings that are nearest to the final output layer offers most freedom for the model to optimize on the most ideal internal representations.

The difference between D_{MSE} and D_{COS} did not show any clear trend for the encoder, cross-attention, and last decoder states. The main difference between the two is that the MSE depends on both the norm of the vectors and the angle between them in d-dimension, while COS is solely a function of the angle. For example, when $\mathbf{x}$ and $\mathbf{y}$ are unit vectors (i.e., $|\mathbf{x}| = |\mathbf{y}| = 1$) the two metrics are identical up to an overall scalar, $D_{\text{MSE}}(\mathbf{x}, \mathbf{y}) = \frac{2}{d} D_{\text{COS}}(\mathbf{x}, \mathbf{y})$. The fact that the two metrics does not have a clear effect on the performance suggests that the consistency is mainly

[1] We use v2.0 for De and v1.0 for Es, Fr, and It

imposed through *aligning* the embedding vectors, and therefore restricting the norm of the embeddings does not provide additional regularization.

The final rows in Table 1 on consistency regularization deals with the softmax output of the network. We apply both the KL divergence and the MSE to this embedding. While softmax-MSE is an uncommon practice, the motivation for this arose from the differences in performance between logits-MSE and softmax-KL. Let us assume the logit outputs of speech and text are $\mathbf{x}$ and $\mathbf{y}$, respectively. Then, $\mathcal{L}_{\text{cr}}^{\text{logitsMSE}} \propto D_{\text{MSE}}(\mathbf{x}, \mathbf{y}) \propto (\mathbf{x} - \mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})$. The softmax output is the normalized exponential of $\mathbf{x}, \mathbf{y}$, and thus $\mathcal{L}_{\text{cr}}^{\text{softmaxKL}} \propto D_{\text{KL}}(\text{softmax}(\mathbf{x}), \text{softmax}(\mathbf{y}))$ roughly scales as $(\mathbf{x} - \mathbf{y}) \cdot (e^{\mathbf{x}} - e^{\mathbf{y}})$. (note that the error comes from the different normalization factors of the softmax, which we neglect in this scaling argument; the exponentials are elementwise) Observing the significant performance difference between logits-MSE and softmax-KL where the essential difference is exponentiating one $(\mathbf{x} - \mathbf{y})$ in the loss, we considered exponentiating both terms which can be done by applying MSE to the softmax output. The exponentiation amplifies the distance between the two vectors and thus has the effect of tying the two embeddings more strongly. However, the difference between softmax-MSE and softmax-KL was less prominent than that of softmax-KL and logits-MSE and had an opposite effect on base and expanded settings. We choose the more conventional softmax-KL as our best model and consider it as the baseline in the following sections.

We would like to comment on the inconsistent effect of features on the base and the expanded settings. As we add various features to the baseline model, the effect is not always similar for the base and extended settings. Regarding this, one should note that the extended setting does not only train additional MT data but also the initialized MT model is pretrained with the additional data. That is, for the extended setting, we train on only a small portion (less than 10%) of the pretrained data and thus the initialized MT model is more robust. This is also why in the expanded setting the response to the additional features differs from the base setting, where the pretrained and fine-tuned data are from the same domain.

5.2 R-drop

We perform similar experiments on R-drop with various embeddings and distance metrics. As mentioned, in R-drop consistency loss, we do not compare the encoder and cross-attention states. The BLEU scores for the en-de ST task are shown in Table 1.

Experiment	Loss	Base	Expand
baseline	$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{rdrop}}^{\text{softmaxKL}}$	28.40	29.66
enc-MSE	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{encMSE}}$	28.07	29.38
enc-COS	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{encCOS}}$	28.15	29.41
xattn-MSE	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{xattnMSE}}$	28.14	29.64
xattn-COS	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{xattnCOS}}$	28.08	29.43
lds-MSE	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{ldsMSE}}$	28.19	29.62
lds-COS	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{ldsCOS}}$	**28.42**	29.50
logits-MSE	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{logitsMSE}}$	28.39	29.43
softmax-MSE	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{softmaxMSE}}$	28.14	29.61
softmax-KL	$\mathcal{L}(\text{baseline}) + \mathcal{L}_{\text{cr}}^{\text{softmaxKL}}$	28.37	**29.73**

Table 2: BLEU scores with both consistency regularization and R-drop on MuST-C en-de dataset. The R-drop consistency is fixed as the softmax-KL and we vary the embeddings and distance metrics of the consistency regularization.

We observe similar trends in R-drop as in consistency regularization. R-drop enhances the performance of the ST task for all embedding candidates and distance metrics, but optimizing on the embeddings that more directly impact the final output leads to better performance. Here, softmax-KL was better than softmax-MSE for both base and expanded settings.

By comparing the best results of consistency regularization and R-drop, the R-drop best results were better for both settings but with different degree of dominance. The gap is quite significant in the base setting, which indicates that with such limited data imposing consistency directly to the target ST task is a more efficient use of the resources. In contrast, when the large external MT dataset is available, enforcing the ST-MT consistency substantially narrows the gap and the two methods show little difference in performance.

5.3 Compounding Consistency Losses

With the consistency regularization and R-drop results at hand, we now turn to the problem of deploying both regularization schemes simultaneously. To observe as many data points as possible, we choose the softmax-KL R-drop as the baseline and add each possible consistency regularization losses shown in Table 1 in tandem. This result is presented in Table 2.

Experiment	Base	Expand
baseline 1 ($\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{softmaxKL}}$)	27.32	29.59
$+ (\alpha_t = 0.5)$	27.41	29.34
$+ (\alpha_t = 0.1)$	**27.68**	**29.68**
$+ (\alpha_t = 0.0)$	27.44	29.64
$+ \mathcal{L}_{\text{cr}}^{\text{encMSE}}$	27.43	29.36
+ (DropKey)	27.34	29.25
baseline 2 ($\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{softmaxKL}} + \mathcal{L}_{\text{rdrop}}^{\text{softmaxKL}}$)	28.37	29.73
$+ (\alpha_t = 0.5)$	**28.56**	29.72
$+ (\alpha_t = 0.1)$	28.10	29.37
$+ (\alpha_t = 0.0)$	28.18	29.30
$+ \mathcal{L}_{\text{cr}}^{\text{encMSE}}$	28.45	**29.84**
+ (DropKey)	28.18	29.70

Table 3: BLEU score with various combinations that impose regularization to the ST model. We use two baselines, one with the softmax-KL consistency regularization and the other with an additional softmax-KL R-drop.

We notice that the BLEU scores of the experiments are surprisingly stable. The variance of the data in Table 2 is 0.018 for base and 0.014 for expand. In contrast, the same columns in Table 1 has variance of 0.32 and 0.18 for base and expand, respectively, showing an order of magnitude difference. While we claimed that enforcing consistency close to the final output is more effective for both consistency regularization and R-drop, and also both base and expanded settings in Table 1, such trend does not appear in Table 2. This suggests that the two regularizations does not simply add up but rather saturates after a certain level of regularization. We conjecture the reason for this behavior stems from the inherent similarity between consistency regularization and R-drop. As seen in Eqs. (3) and (4), the only difference is comparing speech-text embeddings and speech-speech embeddings. Having both loss terms results in the same form of loss but just with different modality, and thus imposing the same *type* of regularization.

There is a similar concept called the "m-time R-drop" in the original R-drop literature (Wu et al., 2021). m-time R-drop ensures consistency between m sub-models by defining the loss function as the sum of the pairwise KL-divergences (that is, of total $m(m-1)/2$ terms). Therefore the conventional R-drop is a 2-time R-drop. Compounding the two consistency losses is conceptually similar to the 3-time R-drop where one of the sub-model is of text modality. And as discussed in section 4.3 of Wu et al. (2021), increasing m does not improve the performance but rather quickly saturates, analogous to the saturation observed in Table 2.

5.4 Other Regularizations

We seek for other means and combinations of regularization on the model. We try three additional regularization methods: i) tune α_t, ii) impose consistency regularization at multiple points of the network, and iii) add a DropKey scheme. For this experiment, we use two different baselines. One is adding softmax-KL consistency regularization to the cross entropy loss (con. reg. softmax-KL row of Table 1) the other is adding both softmax-KL consistency regularization and softmax-KL R-drop (softmax-KL row of Table 2). The results are presented in Table 3.

We first discuss the effect of α_t, the coefficient of the MT cross-entropy loss. While tuning α_t can be construed as a hyperparameter search, our experiments show that the term has a direct effect on the *total* regularization as well. In essence, decreasing α_t decreases the strength of the cross-entropy loss relative to the regularization loss, which effectively increases regularization. We elaborate and present more evidence on the regularization aspect of tuning α_t in the Discussion section.

We observe that by decreasing α_t from 1.0 to 0.0, the BLEU score mostly increases initially but eventually decreases. This is a typical behavior of regularization – when imposed, the performance improves as the model tends not to overfit, but adding too much regularization leads to performance drop. For the baseline 1, the peak performance occurred at $\alpha_t = 0.1$. For the baseline 2, the BLEU score peaked at $\alpha_t = 0.5$ for base setting and decreased monotonically for the expanded setting (there is, of course a possibility that the peak of the expanded setting has occurred between $0.5 < \alpha_t < 1.0$). This is consistent with our argument since with the R-drop loss, baseline 2 already has larger regularization than the baseline 1. Therefore when decreasing α_t the total regularization is greater in the baseline 2, and would have a peak at larger α_t.

We also explored imposing consistency regularization more than once in the network. Since our baseline includes $\mathcal{L}_{\text{cr}}^{\text{softmaxKL}}$, we added $\mathcal{L}_{\text{cr}}^{\text{encMSE}}$ which is farthest from the softmax layer. The experiment showed mixed results, where we attribute the performance drop to the fact that $\mathcal{L}_{\text{cr}}^{\text{encMSE}}$ effectively decreases the degree of freedom for the model to optimize the representations. The addi-

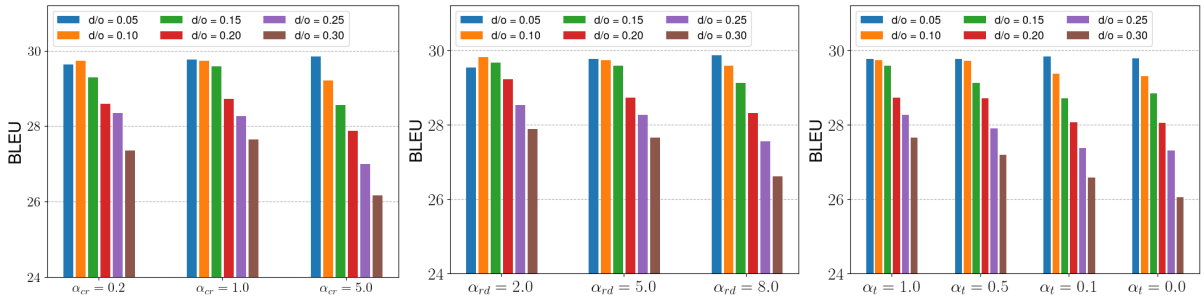

Figure 2: The effect of dropout rate on the ST BLEU score, when varying the loss rate coefficients. The form of the loss function is that of the baseline 2 in Table 3. The default values are $\alpha_{cr} = 1.0$, $\alpha_{rd} = 5.0$, and $\alpha_t = 1.0$. One can again verify that decreasing α_t has the same effect as increasing $\alpha_{cr/rd}$, which is increasing regularization.

tion of DropKey mostly showed performance drop, which may similarly be due to excessive regularization.

6 Discussion

6.1 Consistency in MTL as Means of Regularization

Here, we give an alternative explanation of how decreasing α_t serves as a regularization in the context of the analogy between consistency regularization and R-drop.

As MTL benefits the transfer of knowledge between tasks, it also improves the model's generalizability (Caruana, 1997). In the MTL setting with shared outputs, we can consider the text input for the MT task as a corrupted version of the speech input. Consistency regularization can be thought of as encouraging model consistency between a normal input and a corrupted input – similar in spirit to the Cutoff approach (Shen et al., 2020). The consistency regularization objective function thereby amplifies the cost of overfitting and further enhances the model's generalizability.

Within this context, R-drop and consistency regularization explored in our paper resemble each other; both regularize the inconsistency between the outputs of a corrupted forward pass (either the model is corrupted or the data is corrupted). This is also mentioned in Wu et al. (2021) as they describe the similarity between R-drop and the Cutoff.

With the consistency regularization term in place, α_t becomes a determinant of how corrupted the output of MT forward pass is. A lower α_t value yields less optimal outputs, which can be considered as increased perturbations to the model, effectively increasing the regularization. Hence, α_t value acts as a factor contributing to the amount of *total* regularization. The role of α_t in consistency regularization is analogous to the role of the dropout rate in R-drop. Increasing the dropout rate in R-drop achieves greater perturbation in the output, yielding stronger regularization.

6.2 The Regularization Horizon

As we have established α_t as a knob for regularization, we can now collect all sources of regularization in our scheme (consistency regularization, R-drop, and α_t) and define the concept of *total regularization*. Here, we leave out DropKey as a source since it is difficult to tune the strength of the method. There are three sources in the sense of loss terms, but considering the independent parameters there are actually four – α_{cr}, α_{rd}, α_t, and the dropout rate (d/o). The total regularization would be a function of the four, $f(\alpha_{cr}, \alpha_{rd}, \alpha_t, \text{d/o})$.

Determining the analytic form of f is a formidable task. However, from our experiments, we can infer several core properties of the function. First, f monotonically increases with α_{cr}, α_{rd}, and d/o, and monotonically decreases with α_t. Second, while α_t and d/o have little correlation with others (that is, the composite effect easily adds up for those parameters), the effect of having both α_{cr} and α_{rd} saturates as seen in Table 2.

In general, regularization has some optimal value. Some regularization benefits the model by preventing overfitting, but excessive regularization hurts the model's performance on its original task. Therefore the performance will increase to a peak value and then decrease as one increases regularization to an un-regularized model. With the total regularization as a function of four parameters, the optimal intensity of regularization would be represented as a three-dimensional surface in the four-dimensional parameter space. We refer to this optimal surface as the "regularization horizon", beyond which the model performance begins to

Model	Base					Expanded				
	en-de	**en-es**	**en-fr**	**en-it**	**avg**	**en-de**	**en-es**	**en-fr**	**en-it**	**avg**
XSTNet (Ye et al., 2021)	25.5	29.6	36.0	25.5	29.2	27.8	30.8	38.0	26.4	30.8
ConST (Ye et al., 2022)	25.7	30.4	36.8	26.3	29.8	28.3	32.0	38.3	27.2	31.5
CRESS (Fang and Feng, 2023)	27.2	31.9	37.8	27.3	31.1	29.4	33.2	40.1	27.6	32.6
FCCL (Zhang et al., 2023a)	25.9	30.7	36.8	26.4	30.0	29.0	31.9	38.3	27.3	31.6
Siamese-PT LARGE (Le et al., 2023)	27.9	31.8	**39.2**	**27.7**	**31.7**	-	-	-	-	-
Consistency is Key (Lee et al., 2023)	-	-	-	-	-	29.3	32.3	39.5	-	-
SimRegCR (Gao et al., 2024)	27.9	**32.1**	39.0	**27.7**	**31.7**	29.2	33.0	40.0	**28.2**	32.6
Ours										
$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{softmaxKL}} + \mathcal{L}_{\text{cr}}^{\text{encMSE}} + \mathcal{L}_{\text{rdrop}}^{\text{softmaxKL}}$ + d/o= 0.05	**28.4**	31.8	38.2	27.1	31.4	**29.9**	**33.5**	**40.4**	27.7	**32.9**
$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{softmaxKL}} + (\alpha_t = 0.1)$	27.7	31.7	37.5	27.0	31.0	29.7	33.3	40.0	27.6	32.7
$\mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{cr}}^{\text{softmaxKL}} + (\alpha_t = 0.1)$ + (DropKey)	27.5	31.9	37.7	27.1	31.1	29.4	33.4	39.7	27.5	32.5

Table 4: BLEU scores across 4 languages in MuST-C dataset. We choose three parameter sets which we believe to be in the vicinity of the regularization horizon, and compare it with other baseline models in literature.

rapidly collapse.

We further observe the dynamics of each regularization components with the model performance. In Figure 2, we visualize the combined regularization effects of the three α values and dropout rates. We fix the default value as $\alpha_{cr} = 1.0$, $\alpha_{rd} = 5.0$, and $\alpha_t = 1.0$, and tune α and the dropout rate in each figure. The $\alpha_{cr} = 0.2$ and $\alpha_{rd} = 2.0$ shows the typical behavior of regularization discussed in the previous paragraph – increase to a peak and decreases. However, larger values of α's BLEU score monotonically decrease with d/o. This is because the total regularization is already too large even with d/o= 0.05 and has passed the horizon. For example, let us consider the in the middle panel with varying α_{rd}. One can expect the regularization horizon to be in the vicinity of d/o= 0.10 for $\alpha_{cr} = 0.2$, in between d/o= 0.05 and 0.10 for $\alpha_{cr} = 1.0$, and near d/o= 0.05 for $\alpha_{cr} = 5.0$.

Turning to the rightmost panel with various α_t's, the peak performance is observed at the lower end of the dropout rates, and the model performance degrades more rapidly with smaller α_t. Decreasing α_t adds to the regularization effect, thus, the additional regularization force by increased dropout rates causes more rapid degradation of model performance. This similar behavior of α_t to the other two panels is another evidence that decreasing α_t has the same effect as increasing α_{cr} or α_{rd}.

Finally, the peak BLEU score in all cases are similar. This indicates that the performance of the model is simply a function of total regularization, and does not depend on individual terms. Models with parameters exactly on the regularization horizon will have identical performance.

6.3 Multilingual Speech Translation

We identify several points on the regularization horizon, which are combinations of consistency terms that maximize the model performance within the architectural framework of MTL. Finally, we apply our findings across 4 languages (De/Es/Fr/It) and observe that they are consistent across different languages and across base/expanded settings. As reported in Table 4, our model shows near-SOTA performance in the base setting, and achieves SOTA performance in the expanded setting.

7 Conclusion

In this paper, we explore various consistency training strategies to maximize the model performance in ST in the MTL setting. We show that applying consistency objective functions closer to the final output is more effective for both within and across modalities. We also establish that α_t in consistency regularization serves a similar role as the dropout rate in R-drop. From experiments, we draw the relationship between R-drop, consistency regularization, and α_t, and suggest the *total* regularization and the regularization horizon on which the model performance peaks with optimal amount of total regularization. We select several points near the regularization horizon and show that they achieve near-SOTA performance in MuST-C dataset.

8 Limitations

While we observe indications of the model performance peak, the notion of regularization force is not clearly quantified. As the optimal amount of regularization may vary depending on the task, applications of our findings to other domains will re-

quire extensive search space, making the discovery of regularization horizon expensive. In the future, we plan to investigate a systematic approach to derive the regularization horizon and minimize the need for a rigorous search.

References

Antonios Anastasopoulos and David Chiang. 2018. Tied multitask learning for neural speech translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 82–91.

Parnia Bahar, Tobias Bieschke, Ralf Schlüter, and Hermann Ney. 2021. Tight integrated end-to-end training for cascaded speech translation. In *2021 IEEE Spoken Language Technology Workshop (SLT)*, pages 950–957. IEEE.

Loïc Barrault, Yu-An Chung, Mariano Coria Meglioli, David Dale, Ning Dong, Mark Duppenthaler, Paul-Ambroise Duquenne, Brian Ellis, Hady Elsahar, Justin Haaheim, et al. 2023. Seamless: Multilingual expressive and streaming speech translation. *arXiv preprint arXiv:2312.05187*.

Alexandre Bérard, Laurent Besacier, Ali Can Kocabiyikoglu, and Olivier Pietquin. 2018. End-to-end automatic speech translation of audiobooks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6224–6228. IEEE.

Alexandre Bérard, Olivier Pietquin, Laurent Besacier, and Christophe Servan. 2016. Listen and translate: A proof of concept for end-to-end speech-to-text translation. In *NIPS Workshop on end-to-end learning for speech and audio processing*.

Christian Buck and Philipp Koehn. 2016. Findings of the wmt 2016 bilingual document alignment shared task. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 554–563.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28:41–75.

Mattia A Di Gangi, Roldano Cattoni, Luisa Bentivogli, Matteo Negri, and Marco Turchi. 2019. Must-c: a multilingual speech translation corpus. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2012–2017. Association for Computational Linguistics.

Paul-Ambroise Duquenne, Holger Schwenk, and Benoit Sagot. 2023. SONAR: sentence-level multimodal and language-agnostic representations.

Qingkai Fang and Yang Feng. 2023. Understanding and bridging the modality gap for speech translation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15864–15881.

Qingkai Fang, Rong Ye, Lei Li, Yang Feng, and Mingxuan Wang. 2022. Stemm: Self-learning with speech-text manifold mixup for speech translation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7050–7062.

Gerard I Gállego, Ioannis Tsiamas, Carlos Escolano, José AR Fonollosa, and Marta R Costa-jussà. 2021. End-to-end speech translation with pre-trained models and adapters: Upc at iwslt 2021. In *Proceedings of the 18th International Conference on Spoken Language Translation (IWSLT 2021)*, pages 110–119.

Pengzhi Gao, Ruiqing Zhang, Zhongjun He, Hua Wu, and Haifeng Wang. 2024. An empirical study of consistency regularization for end-to-end speech-to-text translation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 242–256.

Chi Han, Mingxuan Wang, Heng Ji, and Lei Li. 2021. Learning shared semantic space for speech-to-text translation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2214–2225.

Yuchen Han, Chen Xu, Tong Xiao, and Jingbo Zhu. 2023. Modality adaption or regularization? a case study on end-to-end speech translation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1340–1348.

Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 388–395.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

Hang Le, Juan Pino, Changhan Wang, Jiatao Gu, Didier Schwab, and Laurent Besacier. 2021. Lightweight adapter tuning for multilingual speech translation.

In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 817–824.

Phuong-Hang Le, Hongyu Gong, Changhan Wang, Juan Pino, Benjamin Lecouteux, and Didier Schwab. 2023. Pre-training for speech translation: Ctc meets optimal transport. In *International Conference on Machine Learning*, pages 18667–18685. PMLR.

Hojin Lee, Changmin Lee, and Seung-won Hwang. 2023. Consistency is key: On data-efficient modality transfer in speech translation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13572–13581.

Bonan Li, Yinhan Hu, Xuecheng Nie, Congying Han, Xiangjian Jiang, Tiande Guo, and Luoqi Liu. 2023. Dropkey for vision transformer. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22700–22709. IEEE Computer Society.

Xian Li, Changhan Wang, Yun Tang, Chau Tran, Yuqing Tang, Juan Pino, Alexei Baevski, Alexis Conneau, and Michael Auli. 2021. Multilingual speech translation from efficient finetuning of pretrained models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 827–838.

Zhaolin Li and Jan Niehues. 2022. Efficient speech translation with pre-trained models. In *Conference on Neural Information Processing Systems (NeurIPS) 2022: 2nd workshop on Efficient Natural Language and Speech Processing (ENLSP), 2nd December 2022, New Orleans*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53.

Siqi Ouyang, Rong Ye, and Lei Li. 2023. WACO: Word-aligned contrastive learning for speech translation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3891–3907.

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

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Ngoc-Quan Pham, Jan Niehues, Thanh-Le Ha, and Alex Waibel. 2019. Improving zero-shot translation with language-independent constraints. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 13–23.

Juan Pino, Liezl Puzon, Jiatao Gu, Xutai Ma, Arya D McCarthy, and Deepak Gopinath. 2019. Harnessing indirect training data for end-to-end automatic speech translation: Tricks of the trade. In *Proceedings of the 16th International Conference on Spoken Language Translation*.

Matt Post. 2018. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, page 186. Association for Computational Linguistics.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR.

Dinghan Shen, Mingzhi Zheng, Yelong Shen, Yanru Qu, and Weizhu Chen. 2020. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *arXiv preprint arXiv:2009.13818*.

Yun Tang, Juan Pino, Xian Li, Changhan Wang, and Dmitriy Genzel. 2021. Improving speech translation by understanding and learning from the auxiliary text translation task. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4252–4261.

Ioannis Tsiamas, Gerard I Gállego, José AR Fonollosa, and Marta R Costa-jussà. 2023. Efficient speech translation with dynamic latent perceivers. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Changhan Wang, Yun Tang, Xutai Ma, Anne Wu, Dmytro Okhonko, and Juan Pino. 2020. Fairseq s2t: Fast speech-to-text modeling with fairseq. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 33–39.

Anne Wu, Changhan Wang, Juan Pino, and Jiatao Gu. 2020. Self-supervised representations improve end-to-end speech translation.

Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, Tie-Yan Liu, et al. 2021. R-drop: Regularized dropout for neural networks. *Advances in Neural Information Processing Systems*, 34:10890–10905.

Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. Unsupervised data augmentation for consistency training. *Advances in neural information processing systems*, 33:6256–6268.

Chen Xu, Bojie Hu, Yanyang Li, Yuhao Zhang, Shen Huang, Qi Ju, Tong Xiao, and Jingbo Zhu. 2021. Stacked acoustic-and-textual encoding: Integrating the pre-trained models into speech translation encoders. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2619–2630.

Rong Ye, Mingxuan Wang, and Lei Li. 2021. End-to-end speech translation via cross-modal progressive training. In *Proc. of INTERSPEECH*.

Rong Ye, Mingxuan Wang, and Lei Li. 2022. Cross-modal contrastive learning for speech translation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5099–5113.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639.

Han Zhang, Zizhao Zhang, Augustus Odena, and Honglak Lee. 2019. Consistency regularization for generative adversarial networks. In *International Conference on Learning Representations*.

Hao Zhang, Dan Qu, Keji Shao, and Xukui Yang. 2022. Dropdim: A regularization method for transformer networks. *IEEE Signal Processing Letters*, 29:474–478.

Hao Zhang, Nianwen Si, Yaqi Chen, Wenlin Zhang, Xukui Yang, Dan Qu, and Wei-Qiang Zhang. 2023a. Improving speech translation by cross-modal multi-grained contrastive learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:1075–1086.

Linlin Zhang, Kai Fan, Boxing Chen, and Luo Si. 2023b. A simple concatenation can effectively improve speech translation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1793–1802.

A Training details

Model For our speech encoder, we use HuBERT base model, pretrained[2] on 960 hours of Librispeech (Panayotov et al., 2015). The HuBERT model is followed by 2 layers of convolutional subsamplers each with kernel size 5, stride 2, padding 2, resulting in 512 hidden dimensions. The subsequent ST encoder and ST decoder are shared with the MT task and are initialized by pretraining on the MT dataset. The MT pretrained model is a 6 by 6 encoder-decoder model with 8 attention heads, 512 hidden dimensions, and 2048 feed-forward hidden dimensions.

Data preprocessing For audio input, we use the raw 16kHz waveform. For text input, we use the tokenizer trained on both the transcription and translation of our ST dataset with a vocabulary size of 10k using unigram in SentencePiece (Kudo and Richardson, 2018).

Training We train our models using the `fairseq` (Ott et al., 2019) framework. We use Adam (Kingma and Ba, 2015) as our optimizer for training. The learning rate is scheduled using an inverse square root scheduler with maximum learning rate of 1×10^{-4} and 4000 warm-up steps. FP16 training is used as provided in fairseq, without gradient clipping. We train our models on 2 Nvidia V100 GPUs with gradient accumulation of 4 steps, leading to an effective batch size of up to 16 million audio tokens. R-drop requires additional GPU space, as such, we half the batch size for each gradient accumulation and use 4 V100 GPUs to preserve the effective batch size.

Evaluation Our models are validated using case-sensitive detokenized BLEU scores (Papineni et al., 2002) on MuST-C `dev` set using sacreBLEU (Post, 2018). We stop training when the validation BLEU score does not improve for 10 checkpoints, and the averaged last 10 checkpoints is selected as our final model. We report our scores using the same BLEU score metric on `tst-COMMON` of MuST-C. We use paired bootstrap resampling (Koehn, 2004) to compute the statistical significance of our results and report them together with the BLEU scores in Table 4.

B Dataset

- ST

  As noted in the main paper, we downloaded

[2] https://github.com/facebookresearch/fairseq/blob/main/examples/hubert/README.md

and processed MuST-C data as described in the link below.

Link: github/facebookresearch/fairseq

- MT (Pre-training)
  We generally followed the same pre-training schemes as described in the repository below.

  Link: github/ictnlp/CRESS

C Dataset Statistics

We show the statistics of the training dataset used in our paper.

Lang	ST	hours/sents	MT	sents
En-De	MuST-C	420/250k	WMT16	3.9M
En-Es	MuST-C	504/270k	WMT13	14.2M
En-Fr	MuST-C	492/280k	WMT14	31.2M
En-It	MuST-C	465/258k	OPUS100	0.7M

Table 5: Statistics of ST and MT data

D Data for Figure 2

Here, in Table 6, we show the data we have plotted in the main text Figure 2.

δ	0.05	0.10	0.15	0.20	0.25	0.30
$\alpha_{cr} = 0.2$	29.64	29.73	29.30	28.60	28.35	27.35
$\alpha_{cr} = 1.0$	29.77	29.73	29.59	28.72	28.26	27.65
$\alpha_{cr} = 5.0$	29.85	29.21	28.56	27.88	27.00	26.17
$\alpha_{rd} = 2.0$	29.54	29.81	29.67	29.22	28.53	27.89
$\alpha_{rd} = 5.0$	29.77	29.73	29.59	28.72	28.26	27.65
$\alpha_{rd} = 8.0$	29.86	29.58	29.11	28.31	27.56	26.62
$\alpha_t = 1.0$	29.77	29.73	29.59	28.72	28.26	27.65
$\alpha_t = 0.5$	29.77	29.72	29.13	28.71	27.90	27.20
$\alpha_t = 0.1$	29.83	29.37	28.71	28.07	27.37	26.59
$\alpha_t = 0.0$	29.78	29.30	28.84	28.05	27.31	26.06

Table 6: The data for the bar chart in main text (Figure 2). The default values are $\alpha_{cr} = 1.0$, $\alpha_{rd} = 5.0$, and $\alpha_t = 1.0$.‘

E Code

- Main Framework - Fairseq
  We used fairseq as our main framework and implemented our codes as a user directory - a functionality provided by fairseq whereby users can easily run their own implementations. We also use some of the code implemented by github/ictnlp/CRESS for our model structure.

- Custom modifications
  We made custom modifications to the original fairseq repository. The changes are detailed in the code snippet appended at the end of the document. These changes allow us to retrieve intermediate embeddings and calculate the consistency objective functions $\mathcal{L}_{\text{con}}$ and $\mathcal{L}_{\text{rdrop}}$. The custom modifications also include the implementation of the DropKey method.

- Calculating the losses
  Based on the custom modifications, we compute the respective loss values for each set of experiments. We provide a pseudocode for this at the end of the document as well.

- Pseudocode of training code

```
def train(alphas=[alpha_s, alpha_t, alpha_con, alpha_rdrop]):
    for data in dataloader:
        speech_input, text_input, label = data
        speech_output, speech_embeddings = model(speech_input, label)
        speech_output_2, speech_embeddings_2 = model(speech_input, label)
        text_output, text_embeddings = model(text_input, label)
        L_ST = compute_loss(speech_output, label)
        L_MT = compute_loss(text_output, label)
        L_con = compute_loss(speech_embeddings, speech_embeddings_2)
        L_rdrop = compute_loss(speech_embeddings, text_embeddings)
        loss = sum(i * j for i, j in zip([L_ST, L_MT, L_con, L_rdrop], alphas))
```

- Custom modifications to fairseq

```
diff --git ./fairseq/models/speech_to_text/s2t_transformer.py custom/fairseq/models/speech_to_text/s2t_transformer.py
index 50fae2ff..75b1a878 100644
--- ./fairseq/models/speech_to_text/s2t_transformer.py
+++ custom/fairseq/models/speech_to_text/s2t_transformer.py
@@ -428,6 +438,52 @@ class S2TTransformerEncoder(FairseqEncoder):


 class TransformerDecoderScriptable(TransformerDecoder):
+    def forward(
+        self,
+        prev_output_tokens,
+        encoder_out: Optional[Dict[str, List[Tensor]]] = None,
+        incremental_state: Optional[Dict[str, Dict[str, Optional[Tensor]]]] = None,
+        features_only: bool = False,
+        full_context_alignment: bool = False,
+        alignment_layer: Optional[int] = None,
+        alignment_heads: Optional[int] = None,
+        src_lengths: Optional[Any] = None,
+        return_all_hiddens: bool = False,
+    ):
+
+        x, extra = self.extract_features(
+            prev_output_tokens,
+            encoder_out=encoder_out,
+            incremental_state=incremental_state,
+            full_context_alignment=full_context_alignment,
+            alignment_layer=alignment_layer,
+            alignment_heads=alignment_heads,
+            get_alignments=get_alignments,
+        )
+
+        if not features_only:
+            x = self.output_layer(x)
+        return x, extra
+
     def extract_features(
         self,
         prev_output_tokens,
@@ -436,17 +492,26 @@ class TransformerDecoderScriptable(TransformerDecoder):
         full_context_alignment: bool = False,
         alignment_layer: Optional[int] = None,
         alignment_heads: Optional[int] = None,
+        get_alignments: bool = False,
     ):
         # call scriptable method from parent class
-        x, _ = self.extract_features_scriptable(
+        x, attn = self.extract_features_scriptable(
...
             alignment_heads,
         )
-        extra = {"encoder_out": encoder_out} if incremental_state is None else None
+        extra = {
+            "after_cross_attn": attn["after_cross_attn"],
+            "attn": attn["attn"],
+            }
+        if incremental_state is None:
+            extra["encoder_out"] = encoder_out
+
         return x, extra


diff --git ./fairseq/models/transformer/transformer_decoder.py custom/fairseq/models/transformer/transformer_decoder.py
index 744c73f4..49e2e88a 100644
--- ./fairseq/models/transformer/transformer_decoder.py
+++ custom/fairseq/models/transformer/transformer_decoder.py
@@ -259,6 +259,11 @@ class TransformerDecoderBase(FairseqIncrementalDecoder):
         full_context_alignment: bool = False,
         alignment_layer: Optional[int] = None,
         alignment_heads: Optional[int] = None,
+        need_after_cross_attn: bool = False,
+        dropqkv_type: str = None,
+        dropqkv_rate: float = 0.0,
+        dropqkv_length: int = 0,
     ):
         """
         Similar to *forward* but only return features.
@@ -327,7 +332,9 @@ class TransformerDecoderBase(FairseqIncrementalDecoder):
         self_attn_padding_mask: Optional[Tensor] = None
         if self.cross_self_attention or prev_output_tokens.eq(self.padding_idx).any():
             self_attn_padding_mask = prev_output_tokens.eq(self.padding_idx)
-
+        ## IMPL: ST-CR
```

```diff
+        after_cross_attn = []
+        cross_attn_alignments = []
         # decoder layers
         attn: Optional[Tensor] = None
         inner_states: List[Optional[Tensor]] = [x]
@@ -344,9 +351,21 @@ class TransformerDecoderBase(FairseqIncrementalDecoder):
                 incremental_state,
                 self_attn_mask=self_attn_mask,
                 self_attn_padding_mask=self_attn_padding_mask,
-                need_attn=bool((idx == alignment_layer)),
-                need_head_weights=bool((idx == alignment_layer)),
+                need_attn=bool((idx == alignment_layer)) or get_alignments,
+                need_head_weights=bool((idx == alignment_layer)) or get_alignments,
+                need_after_cross_attn=need_after_cross_attn,
+                dropqkv_type=dropqkv_type,
+                dropqkv_rate=dropqkv_rate,
+                dropqkv_length=dropqkv_length,
             )
+            ## IMPL: ST-CR
+            if need_after_cross_attn or get_alignments:
+                if need_after_cross_attn:
+                    after_cross_attn.extend(layer_attn["after_cross_attn"])
+                layer_attn = layer_attn["attn"]
             inner_states.append(x)
             if layer_attn is not None and idx == alignment_layer:
                 attn = layer_attn.float().to(x)
@@ -366,8 +385,8 @@ class TransformerDecoderBase(FairseqIncrementalDecoder):

         if self.project_out_dim is not None:
             x = self.project_out_dim(x)
-
-        return x, {"attn": [attn], "inner_states": inner_states}
+        ## IMPL: ST-CR
+        return x, {"attn": [attn], "inner_states": inner_states, "after_cross_attn": after_cross_attn}

     def output_layer(self, features):
         """Project features to the vocabulary size."""
diff --git ./fairseq/modules/transformer_layer.py custom/fairseq/modules/transformer_layer.py
index 19e035de..17df43dd 100644
--- ./fairseq/modules/transformer_layer.py
+++ custom/fairseq/modules/transformer_layer.py
@@ -15,6 +15,7 @@ from fairseq.modules import LayerNorm, MultiheadAttention
 from fairseq.modules.fairseq_dropout import FairseqDropout
 from fairseq.modules.quant_noise import quant_noise

+from modules.drop_qkv import DropQKV

 class TransformerEncoderLayerBase(nn.Module):
     """Encoder layer block.
@@ -393,6 +394,11 @@ class TransformerDecoderLayerBase(nn.Module):
         self_attn_padding_mask: Optional[torch.Tensor] = None,
         need_attn: bool = False,
         need_head_weights: bool = False,
+        need_after_cross_attn: bool = False,
+        dropqkv_type: str = None,
+        dropqkv_rate: float = 0.0,
+        dropqkv_length: int = 0,
     ):
         """
         Args:
@@ -409,7 +415,16 @@ class TransformerDecoderLayerBase(nn.Module):
         """
         if need_head_weights:
             need_attn = True
+        dropqkv = None
+        if dropqkv_type is not None and dropqkv_rate != 0.0:
+            dropqkv = DropQKV(
+                dropout=dropqkv_rate, droptype=dropqkv_type,
+                dropout_length=dropqkv_length,
+            )

+        ## IMPL: ST-CR
+        after_cross_attn = []
+        cross_attn_alignments = []
         residual = x
         if self.normalize_before:
             x = self.self_attn_layer_norm(x)
@@ -483,18 +498,26 @@ class TransformerDecoderLayerBase(nn.Module):
                     saved_state["prev_key_padding_mask"] = prev_attn_state[2]
                 assert incremental_state is not None
                 self.encoder_attn._set_input_buffer(incremental_state, saved_state)
-
+            query, key, value = x, encoder_out, encoder_out
+            if dropqkv is not None:
+                query, key, value, mask = dropqkv(query, key, value,
+                        padding_mask=encoder_padding_mask, training=self.training
+                )
             x, attn = self.encoder_attn(
-                query=x,
-                key=encoder_out,
-                value=encoder_out,
+                query=query,
+                key=key,
+                value=value,
                 key_padding_mask=encoder_padding_mask,
                 incremental_state=incremental_state,
                 static_kv=True,
-                need_weights=need_attn or (not self.training and self.need_attn),
-                need_head_weights=need_head_weights,
+                need_weights=(need_attn or (not self.training and self.need_attn)) or get_alignments,
```

```
+                 need_head_weights=need_head_weights or get_alignments,
             )
             x = self.dropout_module(x)
+            ## IMPL: ST-CR
+            after_cross_attn.append(x)
             x = self.residual_connection(x, residual)
             if not self.normalize_before:
                 x = self.encoder_attn_layer_norm(x)
@@ -526,6 +549,15 @@ class TransformerDecoderLayerBase(nn.Module):
             else:
                 self_attn_state = [saved_state["prev_key"], saved_state["prev_value"]]
             return x, attn, self_attn_state
+        others = dict()
+        others["attn"] = attn
+        if need_after_cross_attn:
+            others["after_cross_attn"] = after_cross_attn
+        ## IMPL: ST-CR
+        if need_after_cross_attn:
+            return x, others, None
         return x, attn, None
```