CUED_SPEECH AT TREC 2020 PODCAST SUMMARISATION TRACK

Potsawee Manakul
Engineering Department
University of Cambridge
`pm574@cam.ac.uk`

Mark Gales
Engineering Department
University of Cambridge
`mjfg@eng.cam.ac.uk`

ABSTRACT

In this paper, we describe our approach for the Podcast Summarisation challenge in TREC 2020. Given a podcast episode with its transcription, the goal is to generate a summary that captures the most important information in the content. Our approach consists of two steps: (1) Filtering redundant or less informative sentences in the transcription using the attention of a hierarchical model; (2) Applying a state-of-the-art text summarisation system (BART) fine-tuned on the Podcast data using a sequence-level reward function. Furthermore, we perform ensembles of three and nine models for our submission runs. We also fine-tune the BART model on the Podcast data as our baseline. The human evaluation by NIST[1] shows that our best submission achieves 1.777 in the EGFB scale, while the score of creator-provided description is 1.291. Our system won the Spotify Podcast Summarisation Challenge in the TREC2020 Podcast Track in both human and automatic evaluation.

Keywords Podcast Summarisation · Abstractive Summarisation · Sentence Filtering · Transfer Learning

1 Introduction

This paper describes our submissions to the TREC 2020 Podcast Track shared tasks [1]. There are two tasks in the podcast track: ad-hoc segment retrieval (search), and summarisation. Both tasks use the Spotify Podcast dataset[2]. This work focuses on the summarisation task. The summarisation task involves generating a short text summary containing the most salient information for a given podcast episode with its audio and automatic transcription.

There are two main types of summarisation: *extractive* methods which select and reorder words or sentences in the source; and *abstractive* methods which can generate words and phrases that do not appear in the source. Due to the complex nature of abstractive methods, previous work on spoken document summarisation applied traditional machine learning or extractive methods [2, 3, 4]. Recently, deep learning methods have shown much success and become the standard approach for various natural language processing (NLP) tasks, including abstractive text summarisation [5]. In general, deep learning solves text summarisation by applying the encoder-decoder architecture [6] based on recurrent neural networks (e.g. LSTM, GRU) [7, 8], or self-attention networks [9]. More recently, pre-training large transformer-based language model before fine-tuning the model on the target task has made a considerable improvement and achieved state-of-to-art results on various NLP tasks [10]. Based on this success, we propose that our simplest approach is to fine-tuning state-of-the-art abstractive text summarisation such as BART [11] on the podcast data.

The limitation of pre-trained models is that they are trained on written text corpora, which generally have different characteristics from transcriptions of spoken documents. For instance, a transcription of 30-minute audio contains about 5,000 words[3], which is much longer than typical written text documents that are used during pre-training. As a result, commonly used pre-trained models such as BERT and BART can only take up to 512 or 1024 token due to their fixed maximum positional embedding. Another characteristic of spoken documents is that they contain redundancies in

[1]The National Institute of Standards and Technology (NIST)
[2]`https://podcastsdataset.byspotify.com/`
[3]In this report, we will use the terms *word* and *token* interchangeably

speech (e.g. false start, repetition) and some of the spoken utterances do not convey much information. We consider these issues as the long input sequence problem, and we will discuss it as well as proposing methods to handle in more details in Section 3.1 and Section 3.2.

Furthermore, for this challenge, we explore the use of sequence-level training objectives described in Section 3.3, and ensemble of models described in Section 3.4. In Section 4, we present and discuss the official evaluation of our submissions. Lastly, in Section 5, we conclude our experiments as well as suggesting possible future work.

2 Data

More details about the Podcast dataset can be found in [1]. This section presents the statistics of the podcast data that is useful in our design decisions. The dataset contains around 105,360 episodes from different podcast shows on Spotify. Google Cloud Platform's Cloud Speech-to-text API (GCP-ASR) was used to generate transcriptions. In this work, we do not make use of the audio data. We treat creator-provided descriptions as the summaries, so it should be noted that the quality of our target summaries varies from Bad (B) to Excellent (E). Some statistics are (mean $\pm$ standard deviation):

- The number of words in a transcription: 5,727 $\pm$ 4,153
- The number of words in a summary: 61.1 $\pm$ 63.2

In our *development* stage, we use the *brass* subset of all the data (see [1] for more details about processing steps done to obtain the brass set), resulting in 66,242 episodes in total. We further filtered out episodes with descriptions shorter than 5 tokens, and we process creator-provided descriptions by removing URL links and `@name`. Then, we split the data into train/dev sets of 60,415/2,189 episodes. In addition, 150 episodes were selected with 6 set of summaries for each episode (900 document-summary-grade triplets), and they were graded are on the Bad/Fair/Good/Excellent scale (0-3). We use this subset to train our grader.

In the *evaluation* stage, the evaluation (test) set consists of 1,027 episodes, and we will test some models and ensembles of models that are trained on the train set of the development stage on this evaluation dataset.

3 Methodology

In this section, we describe our methods using the following notation: $\boldsymbol{\theta}$ is the summarisation model parameter, $\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), ..., (\mathbf{x}^{(J)}, \mathbf{y}^{(J)})\}$ is J transcription-summary pairs. For simplicity of notation, we will express losses and probability distributions for one pair of data $(\mathbf{x}, \mathbf{y})$.

3.1 Baseline: Fine-tuning BART Model

The BART model was proposed in [11]. It has a standard sequence-to-sequence architecture with a bidirectional encoder (similar to BERT [10]) and a causal decoder (i.e. left-to-right) decoder (similar to GPT-2 [12]). Because of the bidirectional nature of the encoder, and the causal nature of the decoder, BART is suitable for conditional text generation tasks. BART is pre-trained using an unsupervised criterion by randomly shuffling the order in the original text as well as filling masked tokens. In this work, we use `bart-large`[4] consisting of token embedding (shared among encoder, decoder, and language model head), positional embedding, 12-layer encoder, 12-layer decoder, hidden size of 1024, 16 heads (406M parameters in total). Each token to the encoder or decoder is embedded by its token_id and position. The BART model is firstly tuned on a news summarisation corpus such as CNN/DailyMail [7] or XSum [13], before being fine-tuned on the podcast data. Training (or fine-tuning) is done by minimising the negative log-likelihood of target sequence as follows:

$$\mathcal{L}_{\text{ml}} = -\log P(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = -\sum_t \log P(y_t|\mathbf{y}_{1:t-1}, \mathbf{x};\boldsymbol{\theta}) \quad (1)$$

The positional embedding uses the absolute position, limiting the length of the input sequence to 1,024 tokens. However, Section 2 shows that the average number of tokens in a podcast transcription is 5,727; thus, a considerable amount of information is lost when truncating the transcription to be within 1,024 tokens. Thus, firstly we propose to expand the positional embedding of the BART model to accommodate sequences longer than 1,024, and we compare it to the vanilla BART model.

[4]`https://huggingface.co/`

Positional embedding expansion

We create a new positional embedding matrix of a size larger than 1,024 tokens; we copy the learned positional embedding weight of the 1st to the 1,024th positions and randomly initialised those beyond 1,024th position. Then, we can train the BART model end-to-end on the podcast data without or with less aggressive truncation.

In this experiment, due to the limited amount of GPU memory (11 GB), we expand the maximum number of tokens to 3,954; we freeze parts of the BART model and fine-tune the last layers of the encoder and decoder as well as the language model head (81M parameters in total). In the first two blocks of Table 1, we show that the non-expanded BART performs better than the expanded BART, and the reasons are likely due to that: (1) since BART uses a learned positional embedding, instead of sinusoidal function in the original Transformer, simply expanding it does not yield a good performance; (2) learning very long sequences could be inefficient. Lastly, we show in the final block of Table 1 that training all parameters of BART but keeping the maximum positional embedding at 1,024 tokens yields the highest ROUGE scores. Therefore, we conclude that expanding positional embedding is *inefficient*, and we will use the vanilla BART model.

BART Configuration			**ROUGE F$_1$**		
MaxLen	**Part frozen**	**#Param**	**R-1**	**R-2**	**R-L**
3954	✓	81M	26.72	7.70	22.87
1024	✓	81M	27.61	8.88	23.99
1024	✗	406M	29.09	9.92	25.37

Table 1: Variants of BART Model Comparison. (1) Expanded BART: BART with maximum positional embedding of 3,954, (2) Non-expanded BART with original maximum positional embedding of 1,024 but parts of the model is frozen the same way as Expanded-BART, (3) Vanilla BART. All systems are trained on the train split of the brass set, and evaluated on the dev split of the brass set.

3.2 Sentence Filtering

The memory and time required for full self-attention models grow quadratically with the input sequence length $O(n^2)$, preventing us from fine-tuning the model end-to-end on the entire input transcriptions. Also, in Section 3.1, we show that expanding positional embedding does not improve the performance. Thus, we investigate alternative methods by *filtering* out redundant or less informative sentences in the input transcriptions.

1. *Random*: Randomly select input sentences such that the length does not exceed the maximum length. Note that we keep the same sentence order.
2. *Truncation*: The simplest method is to select the first n tokens in the input transcription.
3. *TextRank* [14]: The algorithm ranks sentences based on their similarity to other sentences. In this work, we represent each sentence by averaging `word2vec` embedding. We select top sentences such that in total there are less than n tokens, and we keep the same sentence order.
4. *Hierarchical Attention (HIER)*: We train the hierarchical encoder decoder model used in [15] on podcast data without truncation. At test time, we sum the sentence-level attention score over all time instances to obtain a measure of the importance of sentence i:

$$v_i = \frac{1}{T}\sum_{t=1}^{T} \alpha^{\mathtt{s}}_{t,i} \tag{2}$$

   where $\alpha^{\mathtt{s}}_{t,i}$ is the sentence-level attention score of the hierarchical model, and t is the decoder time instance. Note that if the target sequence is available (e.g. in training data), we can use teacher forcing decoding to obtain the importance score. However, if the target sequence is not available, we have to rely on a decoding method such as beam search.

Sentence filtering can be applied on the input transcriptions at the training time and/or the test time, so we will look at the following scenarios:

- **Test-time only: Comparing different filtering methods**
  During training, the podcast transcription is truncated to be within the maximum length of 1,024 tokens. We compare the filtering methods described above at test time: Random, Truncate, TextRank, and HIER. The

hierarchical model (HIER) is trained on CNN/DailyMail and fine-tuned on the podcast data. Results in Table 2 show that using the attention of the hierarchical model yields the highest ROUGE scores; truncation is better than random selection; however, the TextRank algorithm yields the worst results.

Filtering Method	**ROUGE F_1**		
Test-time	**R-1**	**R-2**	**R-L**
Random	25.65	6.89	22.10
Truncate	29.09	9.92	25.37
TextRank	24.97	6.40	21.47
HIER	**29.37**	**10.02**	**25.51**

Table 2: Comparison of filtering methods at test time. We use the vanilla BART system with maximum positional embedding of 1,024. At training time, the input transcription is truncated and all model parameters are fine-tuned, but at test time the input transcription is filtered.

- **Training-time and Test-time filtering**
  Here, we investigate another option which is to filter the data at the training time in addition to the test time. We select the hierarchical model (HIER) as our filtering method, and at training time we could either use filtered data to train BART from scratch (Train) or fine-tune it on BART trained on truncated data (Fine-tune). Table 3 shows that the best summarisation performance is achieved when we perform filtering by both training and test times. Another observation is that with the filtered data training a model from scratch is better than continue the training from a model trained on truncated data, and this could be due to a local optimum in training.

Filtering Method		**ROUGE F_1**		
Training-time	**Test-time**	**R-1**	**R-2**	**R-L**
Truncate	Truncate	29.09	9.92	25.37
Truncate	HIER	29.37	10.02	25.51
HIER (Fine-tune)	HIER	29.38	10.04	25.69
HIER (Train)	HIER	**29.74**	**10.25**	**25.71**

Table 3: Effectiveness of sentence filtering at training time and test time.

We have shown the effectiveness of sentence filtering at both training time and test time using the hierarchical model. Note that sentence filtering can be thought of as a less aggressive extractive summarisation method. Typically, extractive summarisation aims to select most N (e.g. $N = 3$) salient sentences by creating pseudo labels [9]. From now on, we will make use of the hierarchical model to filter input transcriptions at both training and test time.

3.3 Sequence-level Loss Training

During training, we maximise the likelihood at the token level, while during test time, we use automatic metrics such as ROUGE or we evaluate manually. Both automatic and manual evaluations at test time operate at the sequence level. Thus, optimising at the token level only as done in maximum likelihood training is not expected to yield the best result. We follow [16], which uses reinforcement learning inspired loss:

$$\mathcal{L}_{rl} = (\text{Reward}(\tilde{\mathbf{y}}) - \text{Reward}(\hat{\mathbf{y}})) \sum_t \log P(\hat{y}_t|\hat{\mathbf{y}}_{1:t-1}, \mathbf{x}; \boldsymbol{\theta}) \tag{3}$$

where $\hat{\mathbf{y}}$ is the sequence obtained by sampling $\hat{y}_t \sim P(y|\hat{\mathbf{y}}_{1:t-1}, \mathbf{x}; \boldsymbol{\theta})$ at each time step, and $\tilde{\mathbf{y}}$ is the sequence obtained by greedy search. In [16], training on the sequence-level loss directly is not stable, so we initialise the model using the model weights trained on maximum likelihood. We train the model on the sequence-level loss function in combination with maximum likelihood loss:

$$\mathcal{L} = \gamma\mathcal{L}_{\text{rl}} + (1-\gamma)\mathcal{L}_{\text{ml}} \tag{4}$$

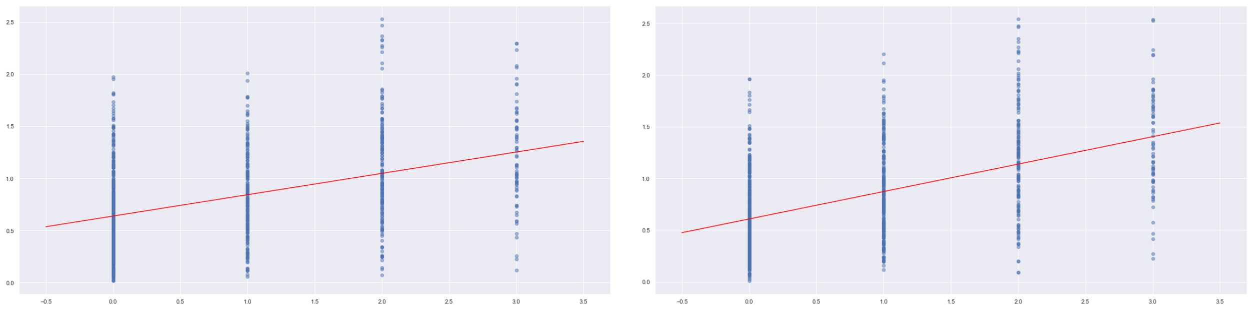
We experiment two types of the reward function to compute $\mathcal{L}_{rl}$:

1. *ROUGE-L:* we follow the ROUGE-L reward training as proposed in [16]. The ROUGE score is computed against the target summary $\mathbf{y}$: $\text{Reward}(\tilde{\mathbf{y}}) = \text{ROUGE-L}(\tilde{\mathbf{y}}, \mathbf{y})$, and $\text{Reward}(\hat{\mathbf{y}}) = \text{ROUGE-L}(\hat{\mathbf{y}}, \mathbf{y})$.

2. *Automatic Grader*: we train a neural grader model ϕ that predicts the grade given a pair of document and summary: $\text{Reward}(\tilde{\mathbf{y}}) = f_{\phi}(\mathbf{x}, \tilde{\mathbf{y}})$, and $\text{Reward}(\hat{\mathbf{y}}) = f_{\phi}(\mathbf{x}, \hat{\mathbf{y}})$, where $f_{\phi}(.)$ gives the grade predicted by model ϕ.

The grader is a convolutional neural network (CNN) model that takes a similarity matrix between input transcription and summary as the input. Each cell (i, j) in the similarity matrix is computed by cosine similarity between the vector representing sentence i in the input, and the vector representing sentence j in the summary. We use Sentence-BERT [17] for the sentence representation. We can use multiple variants of Sentence-BERT models for M-dimensional similarity matrices. In this experiment, we try $M = 1$ using `bert-large-nli-mean-tokens`, and $M = 10$ using nine additional variants. We train the grader model on the part of the podcast data that are annotated with grade information (900 examples from 150 episodes, each episode contains 6 sets of summaries).

As illustrated in Figure 1, we perform 9-fold cross-validation on the subset of podcast data that has grade information, and we achieve Pearson Correlation Coefficient (PCC) of 0.421 for 1-channel input, and that of 0.517 for 10-channel input. Subsequently, we train our CNN model on all the data (900 graded episodes) using 1-channel input due to limited computational resource when using the model for reward optimisation.



(a) 1-channel input, PCC=0.421 (b) 10-channel input, PCC=0.517

Figure 1: 9-fold cross validation experiment on the subset of podcast data that has grade information.

In Table 4, we show that using ROUGE-L as the reward is more effective, and there is an improvement over the baseline. When using our automatic grader; however, the summarisation model achieves higher ROUGE-2 and ROUGE-L scores than the baseline, but this model does not outperform the model trained using ROUGE-L. We believe that the outputs of the automatic-grader-reward model may be better than those of the ROUGE-L-reward model, but further human analysis is required to make a conclusion. Hence, we decided to use the ROUGE-L-reward model to our final ensemble, which is described in the next section.

		ROUGE F_1		
System	**Reward**	**R-1**	**R-2**	**R-L**
ML-trained	-	29.74	10.25	25.71
$\gamma = 1.0$	ROUGE-L	**29.97**	10.32	25.86
$\gamma = 0.9$	ROUGE-L	29.92	**10.42**	25.90
$\gamma = 1.0$	Grader	29.70	10.35	**25.97**
$\gamma = 0.9$	Grader	29.73	10.20	25.80

Table 4: The training data is filtered using HIER model, and the baseline is the model trained on filtered data using only $\mathcal{L}_{\text{ml}}$ criterion. Note that to fit the training on one 11GB GPU, the first three layers of the encoder and decoder are frozen.

3.4 Ensemble of models

We obtain different set of model's weights from different data shuffles, and checkpoints. Given M systems $\{\boldsymbol{\theta}^1, ..., \boldsymbol{\theta}^M\}$, we combine the predictive distribution on the input $\mathbf{x}$ at token-level as follows:

$$P(\mathbf{y}^*|\mathbf{x}) = \prod_{t=1}^{T} P(y_t^*|\mathbf{y}_{1:t-1}^*, \mathbf{x}) = \prod_{t=1}^{T} \left[\frac{1}{M} \sum_{m=1}^{M} P(y_t^*|\mathbf{y}_{1:t-1}^*, \mathbf{x}; \boldsymbol{\theta}^m) \right] \tag{5}$$

where $\mathbf{y}^*$ denotes the hypothesis sequence.

Having investigated various approaches on our own development set, in this part we evaluate their performance on the **evaluation** (test) set of 1,027 episodes. We also build ensembles of models by using different data shuffles, and parameters at different checkpoints. During the evaluation stage, we found that fining-tuning the BART summarisation model tuned on the XSum dataset yields a better result than fine-tuning the same model tuned on the CNN/DailyMail as shown in Table 5.

System	**Filtering**		**ROUGE F_1**		
	Train	**Test**	**R-1**	**R-2**	**R-L(†)**
Baseline*	✗	✗	26.57	9.14	23.43 (18.44)
Baseline	✗	HIER	26.96	9.53	23.70 (18.57)
Filtered	HIER	HIER	26.96	9.75	23.71 (18.90)
Filtered(XSum)*	HIER	HIER	27.10	9.96	23.92 (18.91)
Filtered(XSum)+$\mathcal{L}_{rl}$	HIER	HIER	27.91	10.25	24.67 (19.48)
Ensemble(3shuffles)*	HIER	HIER	28.56	10.83	25.23 (19.98)
Ensemble(3checkpoints)	HIER	HIER	28.12	10.44	24.87 (19.66)
Ensemble(3shuffles$\times$3checkpoints)*	HIER	HIER	**28.57**	**10.86**	**25.28 (19.98)**

Table 5: ROUGE F_1 scores on the evaluation set (1,027 episodes). We use our processed creator-provided descriptions as described in Section 2 as the reference for computing ROUGE scores. We split the summaries into sentences to compute longest n-grams only within sentences; hence the first ROUGE-L numbers, and the ROUGE-L numbers in (†) are obtained when the summary is treated as one sequence. We compare our ROUGE-L scores against NIST-evaluated scores in Table 7. *The summaries of these systems were submitted.

4 NIST Evaluation Results

We submitted four set of summaries from the following systems:

1. CUED-baseline: run_id=`cued_speechUniv3` - BART model fine-tuned on truncated podcast data.
2. CUED-filtered: run_id=`cued_speechUniv4` - BART model fine-tuned on the transcriptions filtered using the hierarchical model.
3. CUED-ensemble3: run_id=`cued_speechUniv2` - Ensemble of 3 BART models (3 random seeds $\times$ 1 checkpoints), each trained on filtered transcription (using hierarchical model) data + $\mathcal{L}_{rl}$ criterion
4. CUED-ensemble9: run_id=`cued_speechUniv1` - Ensemble of 9 BART models (3 random seeds $\times$ 3 checkpoints), each trained on filtered transcription (using hierarchical model) data + $\mathcal{L}_{rl}$ criterion

We are also provided with results of two baselines from Spotify:

1. SPFT-desc: creator-provided descriptions as summaries
2. SPFT-filt: filtered creator-provided descriptions as summaries

179 episodes were selected randomly, and NIST annotators evaluated them on the Bad/Fair/Good/Excellent scale, and answered eight yes/no question (Q1-Q8 in Table 6. For example, Q1 is "Does the summary include names of the main people (hosts, guests, characters) involved or mentioned in the podcast?".

Table 6 shows that our CUED-ensemble3 submission receives the highest human rating at 1.777 in average, compared to our CUED-baseline at 1.564 and SPFT-desc at 1.291. When using an automatic evaluation, CUED-ensemble3 also achieves the highest score shown in Table 7 (note that our ROUGE scores, computed using processed episode descriptions, suggest the performance of CUED-ensemble3 and CUED-ensemble9 are similar). Furthermore, the per episode analysis (in Figure 2) shows that CUED-ensemble3 performs better than an *average* model[5] on 84.36% of the judged episodes, compared to 79.33% for CUED-baseline, and 60.34% for SPTF-filt.

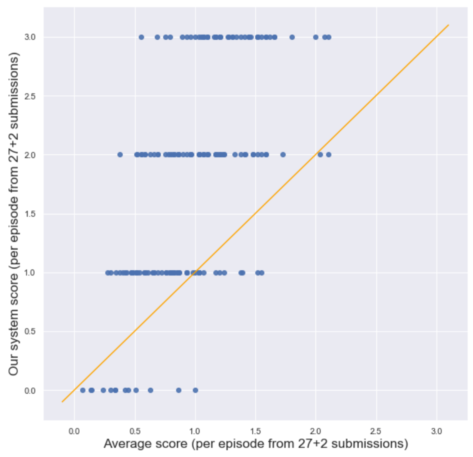
[5] We obtain the score of an average model by computing the mean score of each episode

System	**Avg**	**%E**	**%EG**	**%EGF**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6(↓)**	**Q7**	**Q8**
SPTF-desc	1.291	15.6	39.7	73.7	55.9	34.1	72.1	55.3	63.7	3.4	77.7	52.0
SPTF-filt	1.307	18.4	39.7	72.6	58.1	38.0	70.4	53.1	60.3	2.8	78.2	60.9
CUED-baseline	1.564	21.8	50.3	84.4	**63.7**	**44.1**	82.7	61.5	74.9	**6.7**	88.3	70.4
CUED-filtered	1.687	25.7	56.4	86.6	62.0	43.0	87.2	**63.1**	79.3	7.8	**88.8**	73.7
CUED-ensemble3	**1.777**	26.3	**58.7**	**92.7**	**63.7**	39.1	**89.9**	**63.1**	**80.4**	7.8	88.3	**77.1**
CUED-ensemble9	1.704	**27.4**	58.1	84.9	62.0	40.8	86.0	60.3	**80.4**	10.1	86.6	76.5

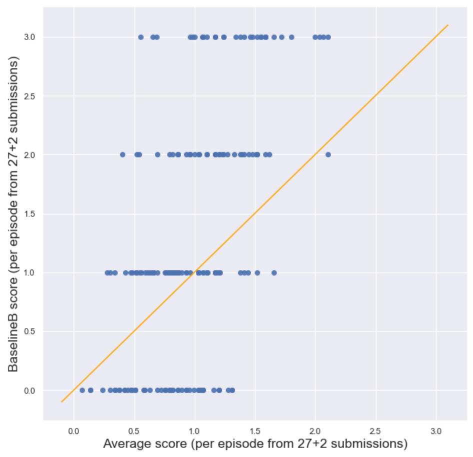
Table 6: Human Evaluation Results. Avg = average of E=3, G=2, F=1, B=0. %E is the percentage of episodes graded excellent, %EG is the percentage of episodes graded excellent or good, %EGF is the percentage of episodes graded excellent, good, or fair. Q1-Q8 are the percentage of *yes* answers.

	ROUGE F_1	
System	**R-L(NIST)**	**R-L(ours)***
CUED-baseline	18.120	18.44
CUED-filtered	18.486	18.91
CUED-ensemble3	**19.670**	19.98
CUED-ensemble9	19.661	19.98

Table 7: Automatic Evaluation Results using the ROUGE-L (F_1) metric. *Our reference for computing ROUGE-L is based on our processed version of episode descriptions (after remove URL links, etc), and we believe that the processed version should better reflect what the models are trained as it is less noisy.



(a) CUED-ensemble3 (84.36%)



(b) SPTF-filt, (60.34%)

Figure 2: Per summary analysis of our submissions. There are in total 27+2 sets of summaries. X-axis is the average score of all submissions, and Y-axis is the system score. The number in brackets is the number of times that the system achieves a higher score than the average score.

5 Conclusion and Future Work

Our best system consists of (1) using the hierarchical model to filter transcriptions at both training time and test time, (2) fine-tuning the BART model initialised on XSum without expanding its positional embedding, (3) optimising the sequence-level loss $\mathcal{L}_{rl}$ after optimising the token-level maximum likelihood, (4) ensemble of models. Both automatic and human evaluations show that this approach outperforms our baseline. Our system-generated summaries obtain higher human ratings than the target summaries (episode descriptions), suggesting training the model using episode descriptions as target summaries can be improved. For instance, an automatic grader can be used to select if an episode description is appropriate as the target or not, allowing one to use semi-supervised learning approaches. Furthermore, the long sequence problem can be better addressed by either improving sentence filtering or using other neural architectures without full self-attention mechanism. Lastly, there is additional information in audio that are not utilised in this work.

Acknowledgments

This paper reports on research supported by ALTA institute, Cambridge Assessment English, University of Cambridge. Thanks to Yiting Lu for evaluating our system-generated summaries before submission.

References

[1] Ann Clifton, Aasish Pappu, Sravana Reddy, Yongze Yu, Jussi Karlgren, Ben Carterette, Rosie Jones, Maria Eskevich, and Gareth Jones. Overview of the TREC 2020 Podcasts Track. In *The 29th Text Retrieval Conference (TREC 2020) notebook*. NIST, 2020.

[2] Sz-Rung Shiang, H.-Y Lee, and L.-S Lee. Supervised spoken document summarization based on structured support vector machine with utterance clusters as hidden variables. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 2728–2732, 2013.

[3] F. Liu and Y. Liu. Towards abstractive speech summarization: Exploring unsupervised and supervised approaches for spoken utterance compression. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(7):1469–1480, 2013.

[4] Guokan Shang, Wensi Ding, Zekun Zhang, Antoine Tixier, Polykarpos Meladianos, Michalis Vazirgiannis, and Jean-Pierre Lorré. Unsupervised abstractive meeting summarization with multi-sentence compression and budgeted submodular maximization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[7] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar GuÌ‡lçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. Association for Computational Linguistics, August 2016.

[8] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, July 2017.

[9] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[11] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[12] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[13] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[14] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.

[15] Potsawee Manakul, Mark J.F. Gales, and Linlin Wang. Abstractive Spoken Document Summarization Using Hierarchical Model with Multi-Stage Attention Diversity Optimization. In *Proc. Interspeech 2020*, pages 4248–4252, 2020.

[16] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[17] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.