Reproducibility Report for Self-Supervised Quality Estimation for Machine Translation

Chahyon Ku, Daniel Cheng, Sherry Zhao, Shubhkarman Singh
University of Washington
`{chahyon, d0, zhaox43, shubhs2}@uw.edu`

Reproducibility Summary

Scope of Reproducibility

Fine-tuning pre-trained multilingual BERT on domain-specific parallel data from the open gold-standard parallel corpus will have higher sentence-level spearman-correlation and word-level F1-scores on quality estimation tasks than prior unsupervised methods.

Methodology

We adapted and annotated the authors' code from their GitHub repository at `https://github.com/THUNLP-MT/SelfSupervisedQE`. The modifications we made to the code is in our GitHub Repository at `https://github.com/chahyon1998/CSE-481N`.
The authors fine-tune multilingual BERT (mBERT), a "104 languages, 12-layer, 768-hidden, 12-heads, and 110M parameters" transformer model pretrained using a masked language modelling objective [1]. Given a concatenated string of the source and target sentence, the model was fine-tuned to mask 15% of target words and maximize probability of recovering them.
We attempted to train 10 models on the nlpg cluster and 8 models on the HYAK cluster, summing up to about 1250 hours of GPU time. We ran about 60 evaluations with various changes in evaluation parameters and with different models, summing up to about 20 hours of GPU time.

Results

With the setup they provided, we could reproduce sentence-level metrics higher than those reported by the paper and word-level metrics 5% lower than those reported by the paper, which upholds the paper's conclusion that it performs better than the unsupervised baselines.

What was Easy

It was easy to run the author's code without error, especially since it included training and prediction scripts. As it uses Huggingface's Transformers library, we were able to easily access the specific pretraind model the authors used and start fine-tuning. The author's code also included a script for downloading and combining datasets from multiple sources into a single training, development, and test set, letting us set up the training pipeline more easily.

What was Difficult

It was hard to understand what the author's code was doing. None of the command-line arguments were labeled properly and we had to figure out the correct parameters by a combination of manually inspecting the code and trial-and-error. Debugging the training process was also difficult, because fine-tuning mBERT took many days on the nlpg clusters we were provided with.

Communication with Original Authors

We did not communicate with the original authors.

CSE 481N , Reproducibility Project Report.

1 Introduction

Quality estimation (QE) is the task of estimating the quality of machine translated text. While the state-of-the-art machine translation systems have achieved surprising accuracy over the past years, it is still far from perfect and often need human evaluation and post-editing to reach sufficient quality. A common example of this is in multi-language e-commerce settings, where the business wants to translate large amounts of text on the fly while still being able to fix erroneous translations. However, it is expensive for human experts to look through the entirety of all the text and fix errors machine translation systems made. Hence, the quality estimation task aims to locate sentences and words that are more likely to have errors, direct the attention of human experts to specific words or sentences, and ultimately allow human experts to fix errors more efficiently.

There are two parts of the QE-task: sentence-level and word-level quality estimation. In sentence-level quality estimation, the model predicts a scalar of how good of a translation each sentence is. In word-level quality estimation, the model categorically predicts if each word in a translation is a good translation or a bad translation.

The authors of Self-Supervised Quality Estimation for Machine Translation [5] proposes a novel self-supervised approach to solving the QE task. As there is a shortage of QE datasets, datasets with source text, machine-translated text, and human post-edited text, previous work [3] trained quality estimation models on synthetic data. To specify, this method took parallel data, translated the source text using their own machine translation system, and marked differences between their translation and the target text to produce a simulated QE dataset. However, the authors of SelfSupervisedQE [5] proposes an even simpler method which fine-tunes a multilingual masked language model (mBERT) to recover masked target words from parallel data. To specify, their method marks words and sentences with low probability to be recovered as “erroneous." The authors’ main claim is that their method outperforms the SyntheticQE method.

2 Scope of Reproducibility

The paper’s central claim is that their novel method for unsupervised QE is better than previous unsupervised methods in both sentence-level and word-level QE, based on results from established QE datasets. They justify some aspects of their method via ablation tests.

2.1 Addressed Claims from the Original Paper

1. The paper claims that their method unsupervised QE achieves better performance in sentence-level and word-level QE compared to prior unsupervised methods such as [3]. They use results from established QE datasets, especially the WMT 2019 English to German development and test sets for both sentence-level and word-level QE.
2. Using Monte Carlo Dropout to sample from Self-supervised models trained using different parameters will improve performance of model in sentence-level and word-level QE.

2.2 Our Claims not from the Original Paper

In addition to running experiments to verify the original paper’s claims, we run some additional experiments to test some hypothesis of our own. However, note that the strength of the evidence we’re able to provide varies between our additional claims.

1. Randomness can play a significant role in the performance metrics of QE, and small difference in performance may just be noise.
2. Since mBERT is trained on the masked language modeling objective with 15% of tokens being masked out, it may be initially intuitive to mask out 15% of the words from the target sentence in training and evaluation of the model. However, for the method introduced by the SelfSupervisedQE paper, increasing the percentage of masked out words actually increases performance.
3. Although adjusting the thresholding parameter for word-level QE can improve the paper’s model’s performance metrics, the differences are slight, and the model can achieve strong results even without using the dev data to pick a threshold.
4. When transferring to other domains, the paper’s method fails to achieve results better than prior unsupervised QE work.

3 Methodology

3.1 Model Descriptions

The authors fine-tune multilingual BERT (mBERT), a "104 languages, 12-layer, 768-hidden, 12-heads, and 110M parameters" transformer model pretrained using a masked language modelling objective [1]. Specifically, they use the cased base variant. [1]

To train the model, the authors concatenate the text of the original sentence and the translated sentence, mask out 15% of the translated sentence's words, and fine-tunes on the typical masked language modeling objective, i.e. maximizing the probability of recovering the masked tokens.

The technique of masking out whole words at a time, as opposed to individual tokens like how mBERT was pretrained, is what the authors call Whole Word Masking (WWM). This technique is necessary since the authors are interested in doing word-level QE, while tokens for mBERT are sub-word units.

During inference, the authors feed each pair of sentence translations into the model multiple times, enabling model dropout and masking out a different 15% of words each time. To be clear, each pair of translated and original sentences is fed into the model N times, and every word of the translated sentence will be randomly masked out M out of those N times. When $N/M = 15\%$, then each translated sentence will have on average 15% of its words masked out. Then, the authors take the average of the predictions for each word when it was masked, and uses the probability of recovering the masked word as a score for the word's translation quality. The authors call this inference technique Monte Carlo Dropout (MCD), in reference to how they sample the model multiple times and enable dropout in the model.

The authors calculate sentence-level translation quality as the average of the word-level translation qualities. For word-level QE, the authors tune a threshold parameter to determine if a word's quality score is high enough to be a correct "OK" translation or a "BAD" translation.

3.2 Datasets

The training data consists of 540k English-German sentence pairs from multiple MT datasets in the IT domain from "the open parallel corpus" (OPUS, https://opus.nlpl.eu/), the WMT 2017 QE dataset [2], and the WMT 2018 automatic post editing dataset [4]. Specifically, the paper uses ada83, GNOME, KDE4, KDEdoc, OpenOffice, PHP, and Ubuntu data from OPUS.

The development and test sets each consist of 1000 QE data points (source sentence, machine translated sentence, and OK/BAD annotation for each translated token) from the WMT 2019 QE dataset [2]. This data was generated by translating the source text using the conference's in-house attention-based NMT system and humans annotating OK/BAD for each word.

The authors provides a shell script that downloads each dataset and combines them into concatenated text files, allowing for us to easily verify if we have all the data the authors used.

Unless otherwise stated, we evaluate and test on the WMT19 English-German QE dataset.

3.3 Hyperparameters

All hyperparameters we use are directly from the paper, unless explicitly stated that we modified it for an experiment. We used the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ to optimize model parameters. During training, we set the effective batch size to 128 [3], the maximum sequence length to 256, the number of training steps to 100,000, the learning rate to 5×10^{-5} and the dropout rate to 0.1. We evaluated our model every 1000 steps and chose the model with the best performance on the development set for inference. For the MC Dropout process, we used the same dropout rate as during training and set $N = 40$ and $M = 6$. So, each prediction masks on average $M/N = 6/40 = 15\%$ of the words. We tuned the threshold τ on the development set to maximize the word-level performance.

[1] We use the checkpoint provided by Huggingface at `https://huggingface.co/bert-base-multilingual-cased`.

[2] For further dataset details, see `https://www.statmt.org/wmt19/qe-task.html`

[3] We used gradient accumulation to reach this effective batch size, due to technical limitations of our GPU memory limits

3.4 Implementation

We adapted and annotated the authors' code from their GitHub repository at `https://github.com/THUNLP-MT/SelfSupervisedQE`. The modifications we made to the code is in our GitHub Repository at `https://github.com/chahyon1998/CSE-481N`.

Their code was based on python 3.6, transformers library, pytorch library, and pandas library. However, we got errors with such low versions of python, and ran our experiments with Python 3.9.12, transformers 4.11.3, pytorch 1.11.0, and pandas 1.4.2.

They provided Python scripts for fine-tuning the model and making predictions, as well as their custom tokenizer to use whole word masking. They also provided a Python script to download and organize datasets as mentioned in the above section.

However, they did not provide Python scripts for evaluating any performance metrics. Although the WMT19 QE task provides their own evaluation scripts at `https://github.com/deep-spin/qe-evaluation`, the authors of SelfSupervisedQE use a slightly modified evaluation scheme from the original dataset's so we manually reimplemented our own code for evaluating the results.

We have ran the data-downloading scripts and included the processed data files in the ./data directory. The pretrained model, mBERT, can be downloaded from Huggingface (https://huggingface.co/bert-base-multilingual-cased) and saved into ./bert-base-multilingual-cased. We have also included command-line arguments we identified from the paper as default parameters to the training and prediction scripts. Therefore, running the default experiment of fine-tuning, predicting, and computing the scores is as simple as running "python train.py," "python evaluate.py," and "python compute_scores.py." Extra shell files for running our additional experiments can be found under the directory ./run_experiments. More detailed explanation of the command-line arguments can be found in the README of our Github repository.

3.5 Experimental Setup

We evaluate the performance of models on two metrics from the paper: word-level F1-scores and sentence-level Pearson correlation coefficient.

Word-level metrics are the F1-scores between the ground truth OK/BAD annotations of each word and the output annotations of the model. To specify, let true = ground truth OK and positive = model prediction OK. Then, precision-OK and recall-OK are $\frac{TP}{TP+FP}$ and $\frac{TP}{TP+FN}$. Precision-BAD and recall-BAD flips the truth values to be $\frac{TN}{TN+FN}$ and $\frac{TN}{TN+FP}$. F1-OK and F1-BAD are the harmonic means of each of these values, and F1-MUL is the product of F1-OK and F1-BAD.

Sentence-level metrics are the Pearson correlation coefficient between ground-truth human targeted translation error rate (HTER) and the sentence-level outputs of a model. HTER, the minimum number of edits a human translator needs to make to have an accurate translation, measures the quality of translation of each sentence. The sentence-level outputs of the model is the average of the probabilities that the masked language model will recover each machine translated token.

3.6 Computational Requirements

We fine-tuned multilingual BERT for 100000 steps with an effective batch size of 128. On the nlpg cluster with a single NVIDIA 1080 Ti, each step took 3.5 seconds on average, summing up to 100 hours of GPU time for all 100000 steps. On the HYAK cluster with a single NVIDIA A40, each step took 1 second on average, summing up to 30 hours of GPU time for all 100000 steps. We attempted to train 10 models on the nlpg cluster and 8 models on the HYAK cluster, summing up to about 1250 hours of GPU time. From these 28 models, 12 were successfully trained and used to provide results.

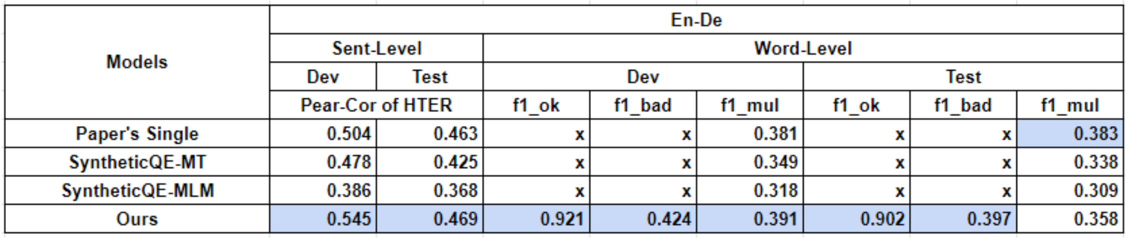
Evaluation was relatively short and took about 20 minutes on a laptop with a single NVIDIA 2080. We ran about 60 evaluations with various changes in evaluation parameters and with different models, summing up to about 20 hours of GPU time.

4 Results

Our work supports that Self-Supervised QE achieves better performance in sentence-level and word-level QE using a single model compared to prior unsupervised methods. Our work doesn't fully support that Self-Supervised QE ensemble model achieves better performance compared to baseline. Our work doesn't support the claim that MC Dropout is conducive to model's performance.

4.1 Comparison between trained model and baselines

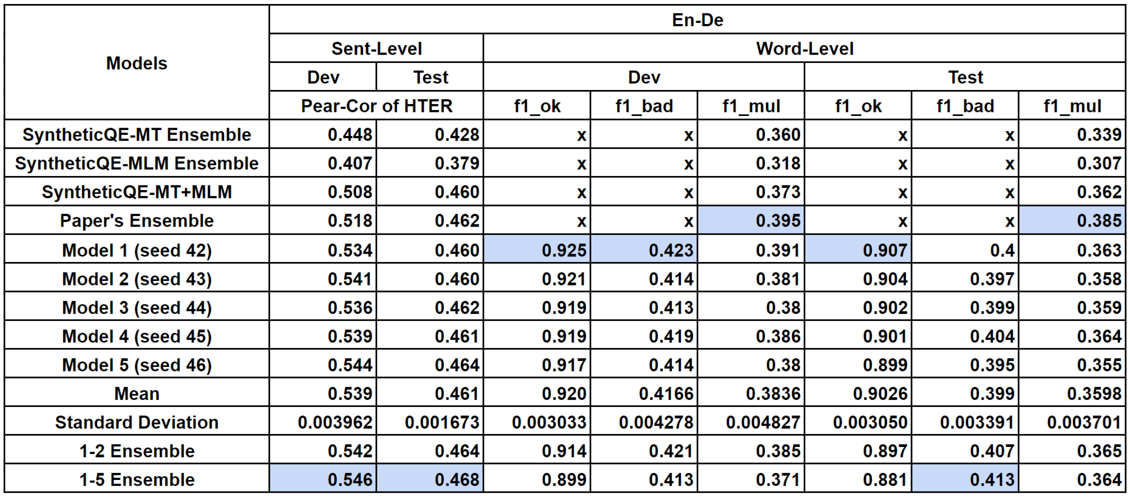
We were successful in reproducing the original paper's results for the single model on the sentence-level and word-level English-German QE dataset. Table 1 shows that for sentence-level QE on both data sets and word-level QE on development set we achieve a higher accuracy than the number presented in the original paper. For the word-level QE test set, we were not able to achieve the number presented in the original paper, only 0.358 compared to the original paper's single model, which got 0.383. However, the number is within 7% of the accuracy in the paper and is still significantly higher than the accuracy of both SyntheticQE models; that upholds the paper's conclusion that it performs better than SyntheticQE baselines.



Models	En-De							
	Sent-Level		Word-Level					
	Dev	Test	Dev			Test		
	Pear-Cor of HTER		f1_ok	f1_bad	f1_mul	f1_ok	f1_bad	f1_mul
Paper's Single	0.504	0.463	x	x	0.381	x	x	0.383
SyntheticQE-MT	0.478	0.425	x	x	0.349	x	x	0.338
SyntheticQE-MLM	0.386	0.368	x	x	0.318	x	x	0.309
Ours	0.545	0.469	0.921	0.424	0.391	0.902	0.397	0.358

Table 1: The result of reproducing a single Self-Supervised QE model and comparison between the trained model with baseline. Numbers in blue are the highest accuracy for sentence-level and word-level test set.

We were also successful in reproducing the original paper's results for the ensemble model on the sentence-level English-German QE dataset. Table 2 shows that for sentence-level QE on development and test set we achieve a higher accuracy than the numbers presented in the original paper. However, for the word-level QE data set, we were not able to achieve the number presented in the original paper with a gap of 6% difference. For the word-level QE on development test, we fail to meet the baseline of SyntheticQE-MT+MLM ensemble model for one of our ensemble model. The result fails to support the paper's conclusion that the ensemble model outperforms the baseline ensemble model.

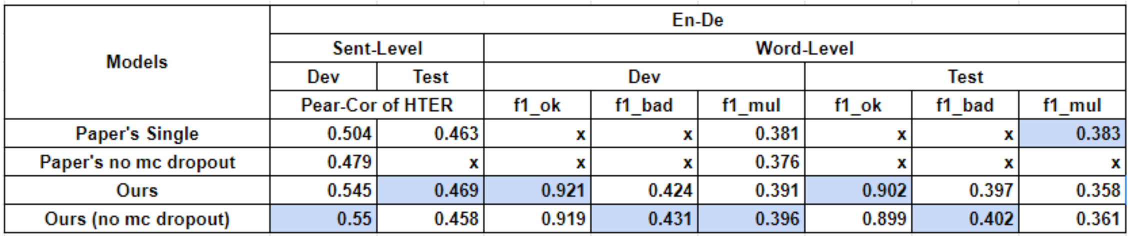


Models	En-De							
	Sent-Level		Word-Level					
	Dev	Test	Dev			Test		
	Pear-Cor of HTER		f1_ok	f1_bad	f1_mul	f1_ok	f1_bad	f1_mul
SyntheticQE-MT Ensemble	0.448	0.428	x	x	0.360	x	x	0.339
SyntheticQE-MLM Ensemble	0.407	0.379	x	x	0.318	x	x	0.307
SyntheticQE-MT+MLM	0.508	0.460	x	x	0.373	x	x	0.362
Paper's Ensemble	0.518	0.462	x	x	0.395	x	x	0.385
Model 1 (seed 42)	0.534	0.460	0.925	0.423	0.391	0.907	0.4	0.363
Model 2 (seed 43)	0.541	0.460	0.921	0.414	0.381	0.904	0.397	0.358
Model 3 (seed 44)	0.536	0.462	0.919	0.413	0.38	0.902	0.399	0.359
Model 4 (seed 45)	0.539	0.461	0.919	0.419	0.386	0.901	0.404	0.364
Model 5 (seed 46)	0.544	0.464	0.917	0.414	0.38	0.899	0.395	0.355
Mean	0.539	0.461	0.920	0.4166	0.3836	0.9026	0.399	0.3598
Standard Deviation	0.003962	0.001673	0.003033	0.004278	0.004827	0.003050	0.003391	0.003701
1-2 Ensemble	0.542	0.464	0.914	0.421	0.385	0.897	0.407	0.365
1-5 Ensemble	0.546	0.468	0.899	0.413	0.371	0.881	0.413	0.364

Table 2: The result of training 5 different models with various seeds from 42-46 along with two ensembles and comparison between the trained model with baseline. Numbers in blue are the highest accuracy for each column.

4.2 Effect of MC Dropout

We were not successful in reproducing the original paper's results for models with Whole Word Masking and without MC Dropout in the ablation study section. The numbers are summarized in Table 3. The accuracy that we achieve by not applying MC Dropout during the evaluation process is even higher than the accuracy of the single model, which is opposite to the author's result that evaluation without MC Dropout yields a lower accuracy than the paper's single model. The result fails to support the paper's assumption that MC Dropout is beneficial for the model's performance.



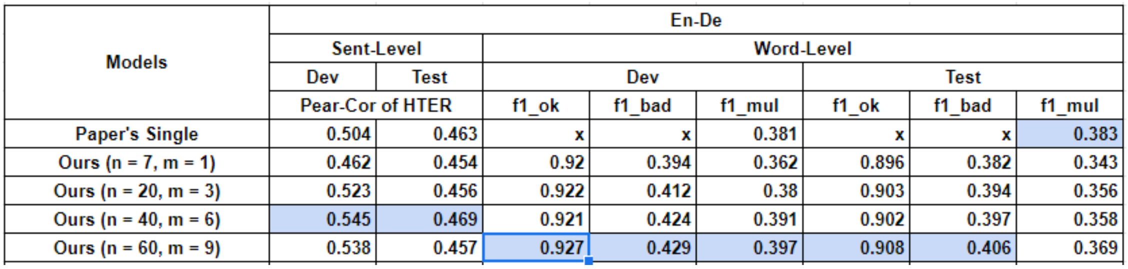
Models	En-De							
	Sent-Level		Word-Level					
	Dev	Test	Dev			Test		
	Pear-Cor of HTER		f1_ok	f1_bad	f1_mul	f1_ok	f1_bad	f1_mul
Paper's Single	0.504	0.463	x	x	0.381	x	x	0.383
Paper's no mc dropout	0.479	x	x	x	0.376	x	x	x
Ours	0.545	0.469	0.921	0.424	0.391	0.902	0.397	0.358
Ours (no mc dropout)	0.55	0.458	0.919	0.431	0.396	0.899	0.402	0.361

Table 3: The result of evaluating a single model without MC Dropout and comparing with the accuracy in the ablation study section.

Additional Results not Present in the Original Paper

4.3 Effect of percentage of words being masked during evaluation

To further see the effect of Monte Carlo Dropout, we first fix the ratio between m and n to be 15% and vary the number of results (n) that we sample from a single model. The model we used for the evaluation is the same model we used to generate the accuracy in Table 1. As we can observe from Table 4, after varying the n, all three sets of parameters will lead to a lower accuracy compared to the initial evaluation when n = 40 and m = 6. However, there is a trend that the F1-mult scores steadily increase when we increase the number of samples. The result reveals that gathering more samples during evaluation will lead to better word-level accuracy. The result indicates that more sampling reduces the randomness of the prediction on word-level QE.



Models	En-De							
	Sent-Level		Word-Level					
	Dev	Test	Dev			Test		
	Pear-Cor of HTER		f1_ok	f1_bad	f1_mul	f1_ok	f1_bad	f1_mul
Paper's Single	0.504	0.463	x	x	0.381	x	x	0.383
Ours (n = 7, m = 1)	0.462	0.454	0.92	0.394	0.362	0.896	0.382	0.343
Ours (n = 20, m = 3)	0.523	0.456	0.922	0.412	0.38	0.903	0.394	0.356
Ours (n = 40, m = 6)	0.545	0.469	0.921	0.424	0.391	0.902	0.397	0.358
Ours (n = 60, m = 9)	0.538	0.457	0.927	0.429	0.397	0.908	0.406	0.369

Table 4: Result of changing n and m during evaluation, fixing the ratio between m and n to be 15%. The initial evaluation is using n = 40 and m = 6.

From there, we fix number of results that we sample from a single model to be n = 40 and vary the ratio between m and n. The model we used for the evaluation is the same model we used to generate the accuracy in Table 1. As we can observe from Table 2, we generally have lower accuracy for extremely small m and extremely large m. This phenomenon is intuitive because masking too few words during the evaluation will make the prediction significantly easier, which leads to higher probability that the model recovers the masked words. On the other hand, masking too many words during the evaluation makes the prediction significantly more challenging and leads to lower probability that the model recovers the mask words. An interesting observation here is that several peaks for accuracy occur when m = 10 and m = 12, which makes the recovery task more difficult compared to m = 6 during training. The result reveals that Quality Estimation tasks are probably more challenging than Machine Translation tasks: masking more words leads to lower probability of the masked word being recovered, which corresponds to the case where a masked word has high recovery probability but the translation is not "OK" in the Quality Estimation criteria (since the recovered word might be wrong).

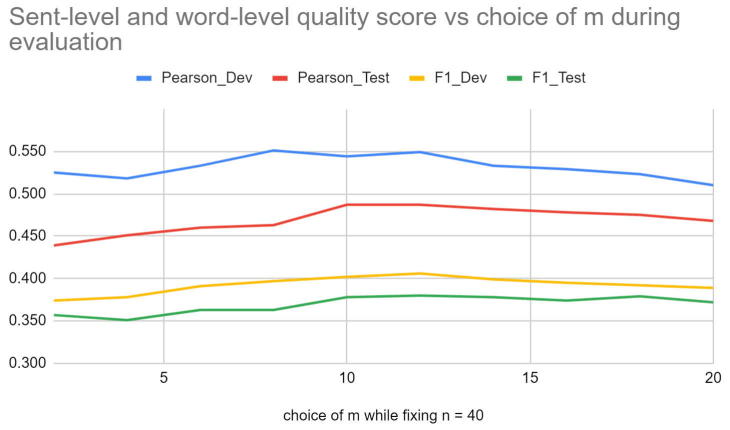
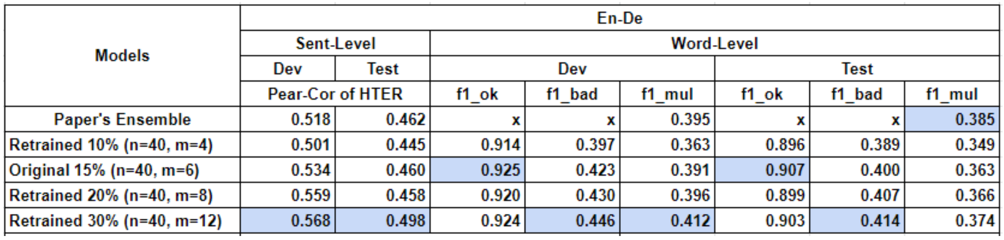


Figure 1: Result of changing n and m during evaluation, fixing n = 40. The initial evaluation is using n = 40 and m = 6. The blue line represents the Pearson-Correlation of HTER on Development set. The red line represents the Pearson-Correlation of HTER on Test set. The yellow line represents the F1-mult score on Development set. The green line represents the F1-mult score on Test set.

We also retrain the model with a different probability of each word being masked out. We do this to see if fine-tuning the model with the appropriate masking probability would prepare it for a different masking probability during inference. Normally, each word in the translated sentence has a 15% chance of being masked during training, but in this experiment, we vary that probability during both training and evaluation. Results for this experiment are shown in Table 5. Interestingly, the retrained models are not always better than the original model evaluated with different M, N.



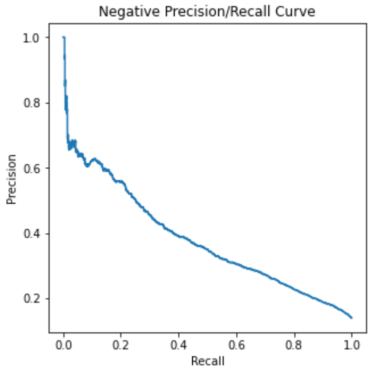
Models	En-De							
	Sent-Level		Word-Level					
	Dev	Test	Dev			Test		
	Pear-Cor of HTER		f1_ok	f1_bad	f1_mul	f1_ok	f1_bad	f1_mul
Paper's Ensemble	0.518	0.462	x	x	0.395	x	x	0.385
Retrained 10% (n=40, m=4)	0.501	0.445	0.914	0.397	0.363	0.896	0.389	0.349
Original 15% (n=40, m=6)	0.534	0.460	0.925	0.423	0.391	0.907	0.400	0.363
Retrained 20% (n=40, m=8)	0.559	0.458	0.920	0.430	0.396	0.899	0.407	0.366
Retrained 30% (n=40, m=12)	0.568	0.498	0.924	0.446	0.412	0.903	0.414	0.374

Table 5: Result of changing N and M during evaluation with models trained with appropriate masking probabilities

5 Discussion

5.1 Larger Implications of the Results

5.1.1 Evaluation Metrics and Precision-Recall Curve for Word-level QE



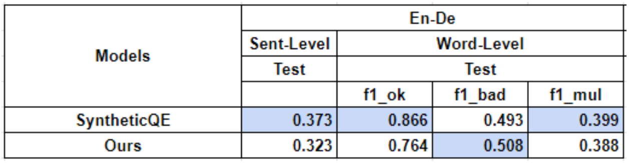
In this task, the data is disproportionately biased towards positive labels. This disbalance is completely expected, since we expect modern machine translation systems to translate most words correctly.

Intuitively speaking, the word-level F1 metrics for the QE task make absolutely no sense in terms of what's necessary for real-world usage of production neural QE systems. In fact, it may be more reasonable to examine the negative F-scores. Furthermore, solely using F-scores hides lots of the intricacies of a model's performance.

Figure 5.1.1 displays the negative precision/recall curve for the default model. The end use-case of neural QE is to provide human experts a tool for figuring out which sentences or words of a translation require human examination to verify and correct. In that sense, precision of negative labels corresponds to the proportion of the time that the human expert will actually have to fix the translation, if they were to only examine translated words that the QE system tells them to. On the other hand, recall of negative labels indicates the proportion of badly translated words that will be caught by the neural QE system. For example, in that sense, according to Figure 5.1.1, if we want to catch 80% of the badly translated words, then the human experts would only be fixing the words and doing something roughly 20% of the time, assuming human experts are only examining words flagged by this QE model and are perfect in making fixes.

5.1.2 Why WMT 2020 and not WMT 2019?

The paper analyzed performance on the WMT 19 QE task, a task from 3 years ago, even though the previous work it claims to have outperformed, SyntheticQE, is originally based on the WMT 2020 QE task. We thought there would be value in exploring how this method will perform in the WMT 2020 QE task.



Models	En-De			
	Sent-Level	Word-Level		
	Test	Test		
		f1_ok	f1_bad	f1_mul
SyntheticQE	0.373	0.866	0.493	0.399
Ours	0.323	0.764	0.508	0.388

Figure 2: Result of SyntheticQE vs. SelfSupervisedQE on WMT 2020 QE task.

5.2 What was Easy

It was easy to run the author's code without error. As it uses Huggingface's Transformers library, we were able to easily access the specific pre-traind model the authors used and start fine-tuning. The author's code also included a script for downloading and combining datasets from multiple sources into a single training set, letting us set up the training pipeline more easily.

5.3 What was Difficult

It was hard to understand what the author's code was doing. None of the command-line arguments were labeled properly and we had to figure out the correct parameters by trial-and-error. There were very few comments describing the code and no specifications before methods, which made it harder to understand key choices the authors made. In the paper, specific values for multiple parameters, including the 15% masking during evaluation and the thresholding values used, was not backed with reasoning.

Debugging the training process was also difficult, because fine-tuning mBERT took forever on the nlpg clusters we were provided with. Training a new model took on average 2 to 4 days to complete, which limited the total number of changes we could implement.

5.4 Recommendations for Reproducibility

We believe commenting the code and adding the evaluation scripts would have greatly reduced our efforts to reproducing the paper. Being unsure of what each part of the code did and what effect each arguments had, we had to experiment with various parameters to see if it accurately reflects the descriptions from the paper. We also had to find and reimplement evaluation code from external repositories, adding some uncertainty to whether we are reproducing exactly the paper's exerimental setup.

Communication with Original Authors

We did not in any form communicate with the original authors.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] L. Specia and V. Logacheva. WMT17 quality estimation shared task training and development data, 2017. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[3] Y.-L. Tuan, A. El-Kishky, A. Renduchintala, V. Chaudhary, F. Guzmán, and L. Specia. Quality estimation without human-labeled data. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 619–625, Online, Apr. 2021. Association for Computational Linguistics.

[4] M. Turchi, M. Negri, and R. Chatterjee. WMT18 APE shared task: En-DE NMT train and dev data, 2018. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[5] Y. Zheng, Z. Tan, M. Zhang, M. Maimaiti, H. Luan, M. Sun, Q. Liu, and Y. Liu. Self-supervised quality estimation for machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3322–3334, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics.