Brooke-English at SemEval-2023 Task 5: Clickbait Spoiling

Shirui Tang
Pingan / 2040179500@qq.com

Abstract

The task of clickbait spoiling is: generating a short text that satisfies the curiosity induced by a clickbait post. Clickbait links to a web page and advertises its contents by arousing curiosity instead of providing an informative summary. Previous studies on clickbait spoiling has shown the approach that classifing the type of spoilers is needed, then generating the appropriate spoilers is more effective on the Webis Clickbait Spoiling Corpus 2022 dataset. Our contribution focused on study of the three classes (phrase, passage and multi) and finding appropriate models to generate spoilers foreach class. Results were analysed in each type of spoilers, revealed some reasons of having diversed results in different spoiler types. "passage" type spoiler was identified as the most difficult and the most valuable type of spoiler.

1 Introduction

According to the clickbait spoiling generate research paper (Hagen et al., 2022) (Fröbe et al., 2023a) (Fröbe et al., 2023b), clickbait spoiling generation is similar to the task of machine reading comprehension (MRC), where context is the full article/advertises with its title waiting for a reader. Question of MRC corresponds to the clickbait, which is a short text that arouses curiosity without providing much informative summary, aiming at attracting people to click into the link towards the context article. And the answer is a spoiling text that could satisfy people's curiosity raised by the clickbait, therefore people could save the time of reading.

Detailed research about the task and its dataset has been conducted in the referring paper. In order to spoil its content, we are listing a couple of key points already verified by the paper and its dataset, and they would be considered as important priors in our experiments.

- All the answers can be located as a span in the context, using extraction models could produce the correct answers.
- Classifying the spoiler into tree types "phrase", "passage" and "multi" is helpful to the span prediction, under the condition of having a good classifier, otherwise using an unclassified model is a better option.
- DeBERTa (He et al., 2020) model showed a general good performance for most of the spoiler-type and score-metric combinations, it is also a good spoiler-type classifier. Although "multi" type was not evaluated, DeBERTa QA model can be regarded as a good initial model for the task fine tune.

During data processing some flaw on target span positions were found, target fixing procedure was applied, details would be described in Section3.1. Due to this target fix procedure, direct result comparison between this paper and the previous research paper may not be appropriate. On the other hand, "multi" type spoilers are included in our experiment. Within a multi example, multiple text spans that are often locate in different paragraph lines of context. Considering the differences in dataset, training tasks and the starting pretrained model, we designed different features, module layers, losses and postprocesses for each type of spoiler.

Experiment results verified that three types of spoilers shall be trained separately with it's own parameters. Using a classifier with accuracy 0.76, the overall benefit of separation training overcomes the drawback of misclassifing the spoiler and send it to a wrong spoiler extractor. There existed a distribution shift between validation and test dataset, due to a small dataset size and the random dataset splitting. Consistant gap of the measures between validation and test dataset were observed. The reason of observed test performance being much better than validation would be discussed. Evidences and error analysis suggested "passage" type spoiler and BLEU-n may be the causes of troubles.

Experiment results showed mixing all types of spoiler in a training dataset would give lower measures on both BLEU-n and meteor than firstly predicting the spoiler type, then use separate models to extrace spoilers. Replacing the classifier with a hypothetical perfect classifier, would produce the best measures. Separating the spoilers would improve on the signal-noise ratio, which would be shown as a very important aspect of explaining performance differences between models.

The DeBERTa QA model had been pretrained for SQuAD, and was used as initial parameters of my models. Relative paragraph id was used as an additional feature for classifier and "multi" model. Contrastive learning losses were used for "passage", "multi" and "mix" models, because their span token lengths were larger in both mean and variance. Learning the start and end boundary of a longer spoiler, is less similar to a typical SQuAD answer, whereas boundary of a short spoiler (i.e. phrase) typically encloses an entity, which is more similar to a SQuAD answer. More modifications were applied to "passage" and "multi" models due to their smaller similarity answers towards SQuAD.

Inference and training codes are released at [1]

2 Background

Table1 shows a clickbait-spoiler data example, context text is a concatenated string of title and paragraphs, clickbait is a question-like sentence, and its answer is the spoiler. The length of each paragraph is at regular sentence length, but the joined context could contain over 2k tokens, which is a length exceeding the typical max length of transformer encoders. Spoilers could appear anywhere in the context, the position of one spoiler span can be expressed in two ways:

1. as shown in the example, [[span start paragraph id, start offset within the paragraph], [span end paragraph id, end offset within the paragraph]]

2. span position within contxts, [start offset within joined context, end offset within joined context]

For "phrase" and "passage" spoiler types, the answer simply includes 1 span, their difference is mainly on token length. For "multi" spoiler, the answer consists of multiple spans, which then can be concatenated to make the answer after extracted. Most "multi" spoilers are extracted from different lines, but there are a few examples of nested spans.

Dataset paragraph spans has to be converted into context spans, then the context would be splited into shorter texts, called feature texts. Each feature text has a fixed number of tokens overlapping with its neighbour feature texts, in order to reduce the likelihood of having a ground truth span that starts and ends across two consecutive feature texts. The feature texts belonging to an example, are then appended after the example clickbait text, forming the complete input texts, like shown in Fig1

The pretrained DeBERTa QA model predicts the start and end position logits separately and by adding the logit scores $s_{ij} = g_i + g_j$, the span score which starts at token position i and ends at token position j can be calculated. Selecting the span with the maximum score gives the answer span. If the feature does not contain any answers, the [CLS] token position shall have the maximum answer, i.e. $s_{00} > s_{ij} \forall i > 0, j > 0$.

SQuAD dataset was used to pretrain the baseline model, and my models are all fine tuned starting with this baseline version.[2]

In order to merge the results of all features belonging to a single example, a beam search mechanism was used in the baseline postprocess. Pseudocode can be used to describe the postprocess alg1: This postprocess works fine for single span results, but it need to be modified for "multi" spoilers.

For the spoiler type classification task, only the first feature of each example is used as the input to calculate the [CLS] feature, then use a FC classifier to predict the spoiler class.

[1]https://github.com/ttssrr423/SemEval2023-Task5-clickbait-spoiling

[2]https://github.com/pan-webis-de/pan-code/tree/master/semeval23/baselines/transformer-baseline-task-2

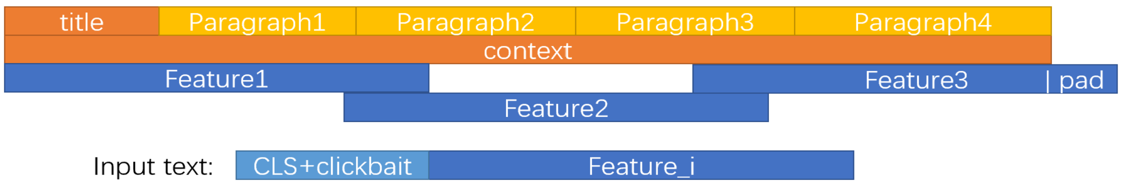

Figure 1: example to input texts

Title	Sprite May Be The Best Hangover Cure, Chinese Researchers Say
Paragraph1	Most of us are familiar with the pounding headaches, ...
Paragraph2	To alleviate the suffering, chugging coffee, downing Advil or simply choosing ...
...	...
ParagraphN	Do you have a tried-and-tested hangover ...
clickbait	This popular soda could cure your hangovers scientists say:
spoiler	Sprite
spoiler-type	phrase
spoiler-pos	[[[2, 125], [2, 131]]]

Table 1: data example

Algorithm 1: baseline postprocess

```
Data: [($g_m^{st}$, $g_m^{ed}$),...]: list of feature logits pair
1  empty reranking list ;
2  for $m^{th}$ feature in example do
3      sort $g_m^{st}$, $g_m^{ed}$ in descending order ;
4      foreach (n, $i = pos_n$) in topN of $g_m^{st}$ do
5          foreach (n, $j = pos_n$) in topN of $g_m^{ed}$ do
6              check for $j > i > 0$ and span text is valid;
7              calculate and adding score $s_{ij}$ to reranking list;
8          end
9      end
10 end
11 return best scored span in reranking list;
```

	phrase	passage	multi	mix
bleu-n	0.592	0.250	0.117	0.367
meteor	0.588	0.385	0.285	0.455

Table 2: spoiler type measures in mix

3 System Overview

In our experiment, dataset repair was conducted before model training, then a single model extracting all types of spoilers, called the mixed model was trained. The mix model was tried with various combinations of input features and loss setups, but the overall result was not satisfactory.

Table2 is the results of the mix model on validation dataset after trained, with detailed bleu-n and meteor on each spoiler type. The performances indicates phrase spoilers are the most similar texts towards a typical QA answer, whereas the multi spoilers have the least similarity. Multi spoilers cannot be produced in the mixed model, it only predicts one answer span.

A classifier was trained, then grid search of confidence threshold of classifier was performed. By comparing the classifier's predicted spoiler type confidence with the thresholds, we could decide whether a type-specified model should be used, and those examples with low type confidences goes into the mixed model. 5 models were trained, three for each individual type, one for mixed, and one classification model.

The three type-specific models all output a score matrix s_{ij}, storing the reranking scores of spans start at token position i and ends at token position j. Their input features, prediction heads, and postprocess are different, depending on their ground truth similarity with the original QA task, more modifications were applied if similarity is low. Multitask training setup was considered but not experimented due to the GPU memory was limited within 24G, so encoder layers do not share parameters.

3.1 Dataset repair

The dataset provided sometimes make mistakes between context span offsets and paragraph offsets, and it may also mismatch with the ground truth text. A fix process on the offsets and the token positions was performed, aligning them with the ground truth spoiler text. If the paragraph id and its spoiler span offsets extracts different text with the ground truth spoiler text, searching around the initial mistaken positions could often fix the mistake. But there were cases that global search had to be done in order to find a match. Matching the paragraph offsets to context offsets while keeping the track of paragraph ids and their absolute position ids within the paragraph requires much carefulness, mistakes can be easily made when the clickbait tokens and the overlap tokens are introduced.

3.2 Spoiler type classification

For the token type classification task, adding the position embeddings after transformer layers gives a better accuracy, like suggested by the DeBERTa paper. A DeBERTa model accepts the first feature of an example, concatenate it with clickbait as the inputs, and trained with cross entropy is the classifier being used for the later extraction experiments. Absolute position ids of input text ($x_{pos} = [0, 1, ...L_{cb}, 0, 1, ..., L_t, 0, ..., L_{p_1}, ..., L_{p_N}]$, where L_{cb}=length of clickbait, L_t=length of title, L_{p_n}=length of n^{th} paragraph) was embedded, and then added after transformers' last layer hidden out. $h_{enc}(x) = DeBERTa(x) + e_{abs}$ where $e_{abs} = Embedding(x_{pos})$ Accuracy of the classifier was around 0.76.

3.3 Phrase extractor

Phrase extractor is the most similar task with the pretrained QA task, so minimum adjustment in the model was made, score metric was identically calculated as $s_{ij} = g_i^{st} + g_j^{ed}$ where $g_i^{st} = logits^{st}(x) \in \mathbb{R}^T$, and $x = (x_w, x_r)$, representing sequence token words and token types. Unlike the classifier model, absolute position ids was not used, $g^{st} = Linear_{st}(DeBERTa(x))$, subscript st indicates the score of start of a span was calculated. Probabilities are $p^{st} = softmax(g^{st})$ and cross entropy loss is the phrase model objective. $Loss_{phrase} = CE(p^{st}, tgt^{st}) + CE(p^{ed}, tgt^{ed})$

Phrase spoilers typically has only a few tokens, therefore relative position in DeBERTa could well capture the n-gram relations if n is not large. Reranking and select the top 1 non-empty span to obtain the answer in postprocess is identical with alg1.

3.4 Passage extractor

The answer span has a larger variance in terms of the span token length, lengths of passage spoilers could vary from as short as phrases to as long as a 200 token paragraph. Score metric was $s_{ij} = g_i^{st} + g_j^{ed}$, the absolute position embeddings were not used. Softmax supresses non-target logits with a strength proportional to the exponential of logits, this involves all the possible relative positions. However the useful supressions are the "top but wrong" logits, their gradients would be more important than the gradient of supressing the long tail logits, because the long tail logits would not

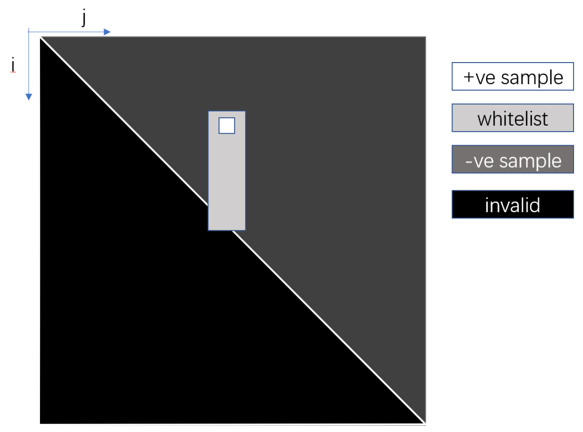

Figure 2: whitelist of a span

threaten the gound truth logit. A contrastive loss was added to the cross entropy loss, in order to emphasis supression of the important negative logits. Total loss is a weighted sum of cross entropy loss same as phrase model plus the contrastive loss.

Contrastive loss $J_c = \frac{1}{K}\sum_{i,j\notin \boldsymbol{C}} \delta_K ReLU(s_{ij} - s^{(+)})$, where $s^{(+)}$ is a positive span's score. Phrase type only has on positive span, the span score of ground truth $s_{ij=gt}$. For those span positions not in the whitelist $\boldsymbol{C}$, choose the top K scores which are greater than ground truth score, then calculate their mean advantages over the ground truth. We hope the ground truth score would have the maximum score, hence achieving $J_c = 0$. $\delta_K = 1$ if $(i, j) \in \{$TopK Span ij set$\}$ else 0. Whitelist $\boldsymbol{C}$ was defined as fig2, it contains the ground truth span position as well as any substring span positions within the ground truth. Also it may include a safty zone/margin, which is 2 token positions neighbouring the start or the end of ground truth span. This margin forbids negative sampling from substrings of ground truth including neighbouring positions $x^{st}_{pos} - 2$ to $x^{ed}_{pos} + 2$. For longer spans, the information extracted would not change much if we add an extra 2 tokens, and we should focus on supressing those spans which share little IOU with ground truth span, meanwhile keeping high s_{ij} scores. The top K negative samples being sampled are from "-ve sample" and "invalid" region. Any spans within the whitelist predicts a span that may not be the ground truth but still gives non-zero BLEU, meteor and BertScore during evaluation, so avoiding negative sampling from whitelist would reduces the noise.

The overall loss for passage model is a weighted sum of the contrastive loss and the cross entropy loss, with weights of 1.0 in both terms. $Loss_{passage} = J_c + CE(p_{st}, tgt_{st}) + CE(p_{ed}, tgt_{ed})$

Postprocess of passage model is similar as alg1, the topN loop was based on score $\{i^{(N)}\} = topN(argsort_i(max_j(s_{ij})))$ and then $\{j^{(N,\{i\})}\} = topN(argsort_j(s_{ij} \mid i \in \{i^{(N)}\}))$. It selects the topN starting i's based on their endings with max span score, and then loop foreach selected i's, select topN endings. Hence we would have $N \times N$ top score candidates in the reranking list. A final reranking of selecting top N from the list would produce N spans. This is a much quicker approximation of ranking all combinations of ij and then choosing top N results. When N=1, the process is identical as $i^*, j^* = argmax_{i,j}(s_{ij})$, which is our desired passage span.

3.5 Multi extractor

Multi spoiler has the least number of examples in the dataset, but the most modifications was applied on the multi model. Many of the multi spoilers are enumerated items that extracted from different paragraphs, therefore paragraph id information should be incoporated into the model. Paragraph ids were defined as the following:

$$x_p(t) = \begin{cases} -1 & \text{if } 0 \le t < L_{cb} \\ 0 & \text{if } L_{cb} \le t < L_{cb} + L_t \\ d \text{ if } L_{cb} + L_t + \sum\limits_{1 \le e < d} L_p(e) \le t & \end{cases} \quad (1)$$

Tokens in the clickbait with token length L_{cb} were labeled with -1, tokens in title with length L_t were labeled as 0, and tokens in the d^{th} paragraph were labeled with d, starting from paragraph index 1. Clickbait, title and paragraphs are concatenated into a single sequence, whose sequence position was labeled as t.

Relative paragraph id positions between title and paragraphs were calculated, then their embeddings were looked up as $e^{(ctx)}_{mn}$, representing the relative paragraph id embedding of query token in m^{th} paragraph of context, and key token in n^{th} paragraph of context. Context included title and paragraphs, but no clickbait. the embedding for clickbait was a different trainable embedding $e^{(q)}$.

The relative attention score bias between query position i and key position j was calculated as:

$$s_{attn}(i,j) = \frac{1}{\sqrt{(D_H)}}\left[q(i)^T\mathbf{W}_{qk}k(j)+ (\mathbf{W}_{st}q(i))^T e_p(i,j) + (\mathbf{W}_{ed}k(j))^T e_p(i,j)\right] \tag{2}$$

where query, key vectors are slices from DeBERTa out hidden plus absolute position embeddings same as type-classify model, $q(t) = k(t) = h(t) = DeBERTa(x)+e_{abs} \in \mathbb{R}^{D_H}$. $\mathbf{W}_{qk}, \mathbf{W}_{st}$ and $\mathbf{W}_{ed}$ are linear layers with biases. The relative paragraph embeddings looked up is:

$$e_p(i,j) = \begin{cases} e^{(q)} & \text{if } x_p(i) < 0 \text{ OR } x_p(j) < 0 \\ e^{(ctx)}_{mn} & \text{if } x_p(i) = m \text{ AND } x_p(j) = n \end{cases} \tag{3}$$

The score modifier was made antisymmetric as: $m_{ij} = s_{attn}(i,j) - s_{attn}(j,i)$. The output score was $s_{ij} = g^{st}_i + g^{ed}_j + m_{ij}$. The logits g^{st} and g^{ed} are also calculated with absolute position embeddings added after deberta out, like in the spoiler-type classifier.

A token would attendent to tokens not only in its neighbour positions, but also the tokens in its neighbour paragraphs, it would be benefitial for those spoilers locatd in a series of consecutive paragraphs, which is quite oftenly the case.

Cross entropy loss for "multi" was different, because there were multiple (start,end) spans, a regular softmax would mistakenly supress some of the ground truth logits. Hence a "multi" version of softmax was used:

$$Z_- = \sum_{t|tgt(t)=0} exp\{g_t\} \tag{4}$$

$$Z = \sum_t exp\{g_t\} \tag{5}$$

Z_- is the partition function for all negative logits, Z is the normal version of partition function. The multi-softmax probability is:

$$p(t) = \begin{cases} exp\{g_t\}/Z & \text{if } tgt(t) = 0 \\ \dfrac{exp\{g_t\}}{Z_- + exp\{g_t\}} & \text{if } tgt(t) = 1 \end{cases} \tag{6}$$

The multi-softmax ensures any positive logit does not supress the logit of any other positive positions. Also the probabilities are close to the the single-positive softmax probabilities if we ignore the other positive positions temporarily as their probabilities are not being evaluated. Although multi version has a sum probability may not be equal to 1, the logp losses would not be affected much and therefore gradients would still descend at same direction. When there's only one positive logit, the multi-softmax is identical as regular softmax. Cross entropy loss of span start becomes $Loss^{st}_{ce} = \sum_{l=0}^{L} -\delta_{tgt^l_t} log(p^{st}(t))$, where the "multi-hot" delta is 1 when the l^{th} ground truth span starts at position t (or ends at position t for the ed version loss).

Contrastive loss is also different as we need a threshold to know how many spans are produced as the multi answers. The span score at CLS, s_{00} was chosen as the threshold. Negative samples are those span position $pos_-(i,j) = \{(i,j) \notin \boldsymbol{C}\}$ not in whitelist defined same as the "passage" model, and their contrastive loss is:

$$J_{c,-} = 0.5 * \Big(max(ReLU(s^{topK}_- - s_{00}))+ mean(ReLU(s^{topK}_- - s_{00}))\Big) \tag{7}$$

TopK negative sampling is performed to focus on learning the difficult spans. Positive samples produces contrastive losses as:

$$J_{c,+} = 0.5 * (max(ELU(s_{00} - s_+))+ mean(ELU(s_{00} - s_+))) \tag{8}$$

A balanced contribution between max and mean losses was considered for the both positive and negative samplings, only considering max would be a more accurate design but more difficult to train.

The overall loss of multi model is $Loss_{multi} = (J_{c,+} + J_{c,-}) + 0.5 * (Loss^{st}_{ce} + Loss^{ed}_{ce})$, the loss gives less weight on the multi-CE loss and more on contrastive loss compare with "passage".

Postprocess of multi spoiler is also quite different, top1 search cannot be used. If we use topN search, there would be another problem. The n candidate spans with highest scores may overlap to each other due to they have close start and end positions. We want to group similar answer spans, and choose the best answer foreach group, so that the final answers are texts with "orthogonal" contents.

A greedy clustering of candidate answer was designed as alg2

IOU of token count between current and previous spans were used as a measure of deciding whether a new answer line (or a new group) should be added. The complete postprocess for multi

Algorithm 2: multi span orthogonize

```
Data: [($sp_1$, $sp_2$),...]: list of topN spans in a feature
empty groups list $\boldsymbol{G}$;
for $sp_n \in$ topN span list do
    foreach group id $i_g \in \boldsymbol{G}$ do
        if $\exists sp_{o<n} \in \boldsymbol{G}[i_g]$ and IOU($sp_o$, $sp_n$)>0.6 then
            append $sp_n$ after group list $\boldsymbol{G}[i_g]$
        end
    end
    if $sp_n$ not added to any previous group then
        create a new group in $\boldsymbol{G}$
    end
end
set result list as empty ;
foreach group id $i_g \in \boldsymbol{G}$ do
    $sp_{i_g}^{top1} = max_{sp.score}(\boldsymbol{G}[i_g])$ ;
    add $sp_{i_g}^{top1}$ to result list ;
end
return result list ;
```

Algorithm 3: postprocess for multi spoilers

```
Data: $\boldsymbol{E}$=[($feat_1$, $feat_2$),...]: list of features in an example
define Group(list) as alg2 ;
empty candidate list $\boldsymbol{D}$;
for $feat_v \in \boldsymbol{E}$ do
    calculate scores and spans $(\boldsymbol{s}^v, \boldsymbol{sp}^v) = model_{multi}(feat_v)$ ;
    calculate $\boldsymbol{F}_{cand}$ ={$sp_{1:N}$}= topN list of "passage" postprocess ;
    calculate $\boldsymbol{F}_{gp}$=Group($\boldsymbol{F}_{cand}$) ;
    foreach $sp \in \boldsymbol{F}_{gp}$ do
        append ($sp$, $s_{00}^{(v)}$) to $\boldsymbol{D}$
    end
end
calculate $\boldsymbol{O} = Group^*(\boldsymbol{D})$ ;
sort $\boldsymbol{O}$ with span score $s_{sp}$ in descending ;
select top 6 answers from $\boldsymbol{O} := \boldsymbol{O}^{(top6)}$ ;
delete any $sp_r \in \boldsymbol{O}^{(top6)}$ if $s_{sp_r} < 0.9 s_{00}^{v|r}$ until $|\boldsymbol{O}| \leq 2$ ;
rearrange $\boldsymbol{O}$ with ascending paragraph id;
return $\boldsymbol{O}$
```

spoiler of an example is alg3. The input is a list of features belong to a same example like shown in Fig1, with subscript v representing its id.

The output of "multi" postprocess keeps at least 2 answer lines and at most 6 lines. Statistic showed most of the ground truth multi spoilers contains 2 to 6 answer lines. Spans belonging to any feature of the example are sorted with their scores s_{ij}^r, r is a loop index of $\boldsymbol{O}$ which is the example's output of alg3. We try to keep those spans which are allowed to be slightly lower (the 0.9 factor) than the threshold of its feature score $s_{00}^{v|r}$, in the hope of increasing the recall. The final $Group^*$ process is slightly different from the feature $Group$ process in terms of $Group^*$'s IOU is calculated on text offsets instead of token position ids, this makes calculation easier outside the feature loop.

3.6 Mixed extractor

Mixed model is similar to the "phrase" model, but its loss is a combination of CE with multi-softmax (CEM) and contrastive loss. $Loss_{mix} = Loss_{CEM}(g^{st}, tgt^{st}) + Loss_{CEM}(g^{ed}, tgt^{ed}) + 0.5 * J_c$ where $J_c = \frac{1}{K} \sum_{i,j \notin \boldsymbol{C'}} \delta_K ReLU(s_{ij} - max(s^{(+)}))$. Noticing the difference of contrastive loss here and in the "passage" model, the positive scores are wrapped with a max, meaning we have abandomed predicting multi answers, in order to avoid the distruption of learning second best span. Only one best span would be produced in the post-process. The score metric was $s_{ij} = g_i^{st} + g_j^{ed}$, where absolute position embedding was not used after DeBERTa output. The safty margin in whitelist defination was set to 0, to make the span boundary clearer for short "phrase" spoilers. For the short phrase spoilers, extra 2 tokens would make a great difference in BLEU and meteor, so whitelist only contains substrings from token start $i = x_{pos}^{st}$ to end $j = x_{pos}^{ed}$ of ground truth.

4 Experimental Setup

800 examples were splited into 400 for validation and 400 for test, whereas the training example has a number of 9138. Train batch size was 4 and gradient accumulation steps was 5. Max token sequence length was set to 384. Negative sample in contrastive loss chose topK as $K = 20$, postprocess in multi answer grouping chose topN as $N = 30$. During the training of all the models, the following techniques based on experience were used to improve training performance:

1. Lookahead optimizer was being used, with AdamW being the base optimizer, lr was 1e-5, weight decay was 1e-2.
2. Learning rates for different transformer layers were grouped and separately set. Layer 1 to 8 have a lower learning rate of $LR/1.6$, layer 9 to 16 have an unchanged LR, and layer 17 to 24 have a higher $LR * 1.6$.
3. Linear schedule with warmup were used, with warmup step=60.
4. Adversarial training was applied after the first epoch, AWP had an attack lr of 1e-4.
5. Epoch parameter set with the best evaluation metric was chosen, with max train epoch num=10.

Differences of setups for training all the models are listed in Table3. Rows are the model options, including: 1) absolute token positions in each paragraph; 2) paragraph ids used to calculate relative paragraph embedding and the antisymmetric modifier; 3) multi-softmax in cross entropy; 4&5) weights of cross entropy loss and contrastive loss; 6) width of margin in terms of number of tokens when defining the whitelist; 7) max number of tokens of a single answer span allowed; 8) top N number during postprocess; 9) number of answer sapans after grouping similar answers.

	type	phrase	passage	multi	mix
abs pos	✓	✗	✗	✓	✗
para-id modifier	✗	✗	✗	✓	✗
multi -softmax	NA	✗	✗	✓	✓
CE weight	NA	1.0	1.0	0.5	1.0
contrastive weight	NA	0.0	1.0	1.0	0.5
whitelist margin	NA	NA	2	2	0
span max #tokens	NA	40	200	40	200
postprocess topN	NA	1	1	30	1
#grouped answers	NA	NA	NA	[2 to 6]	NA

Table 3: model setups

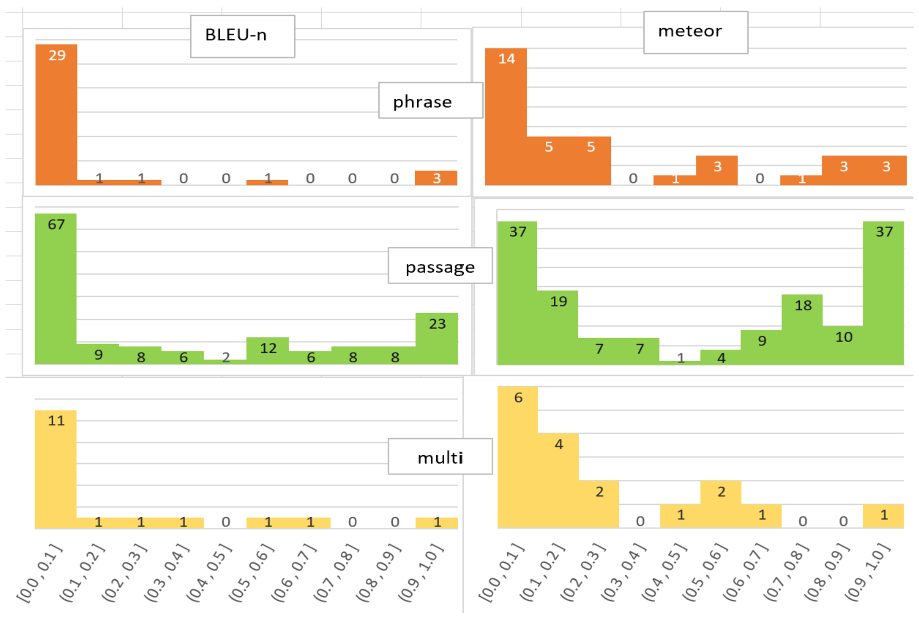

Figure 3: measure metric distribution for "passage" predictions

$\Delta = 0.808 - 0$ **prd:** handling their money and talking about it with their partners **GT:** talk about their money
$\Delta = 0.844 - 0.235$ **prd:** the NES Classic Edition is not available for pre-order just yet **GT:** not available for pre-order
$\Delta = 0.709 - 0.120$ **prd:** Roberts continued, I think she's fabulous," before trailing off, saying, "but she doesn't seem ... she seems cooler than ..." **GT:** I think she's fabulous,
$\Delta = 0.814 - 0.263$ **prd:** the canisters Bruce Wayne/Batman used have "Pb" — the abbreviation for lead in the Periodic Table of Elements — etched on them. **GT:** the canisters Bruce Wayne/Batman used have "Pb"
$\Delta = 0.701 - 0.158$ **prd:** I think this one is straight down the pike for the fans, said Spielberg, who would not reveal plot details, with one exception: "The one thing I will tell you is I'm not killing off Harrison [Ford] at the end of it." **GT:** not killing off Harrison [Ford] at the end

Table 4: divergence between bleu-n and meteor for validation predicted as "passage"

5 Results

Type classification result is shown in Table5. Reason of such a big difference is due to the imbalanced "passage" examples in the two dataset, and we would discuss it later with more details. We would show test and validation results together as the evidences.

	classifier
val acc	0.7625
test acc	0.7925

Table 5: spoiler type model results

If the spoiler type can be perfectly predicted, which is just a hypothetical scenario, the performance of models for different types can be evaluated separately.

	phrase	passage	multi	mix	all
val B	0.644	0.343	0.276	0.367	0.450
test B	0.617	0.402	0.388	0.412	0.494
val M	0.610	0.500	0.570	0.455	0.557
test M	0.583	0.530	0.621	0.470	0.570

Table 6: poiler generation knowing the types

Metric B is BLEU-n, n is in range [1,4] and not exceeding the length of number of words in ground truth text. Metric M is meteor. Since there were only 400 in validation and 400 in test dataset, the metrics evaluated had a large variance. Results indicated that test dataset may contain less hard examples than validation dataset, therefore their results were better except for the phrase spoilers. There were not enough time for me to k-fold all the models to make a more convincing result, but the overall BLEU-n and meteor for "all" spoiler types (combined result for all 3 spoiler types using the classifier model) are larger in the test dataset, so overfitting did not occure. The column "mix" showed result of training all types of spoilers with one single model. Clearly separating the spoiler types and train a model for each type would increase the "all" performance in both datasets.

Using a classifier already trained, confidences of each spoiler type can be obtained. A grid search on validation dataset of thresholds for "phrase", "passage" and "multi" was performed. If an example has its top1 type probability above the confidence threshold of the corresponding type, then the type-specified model would extract the answer, otherwise the "mix" model would extract the answer instead.

Thresholds of phrase=0.75, passage=0.45, multi=0.7 were chosen because it had a balanced good performance in validation dataset. Results of two-stage prediction using such a threshold were shown in Table7

	phrase	passage	multi	mix	all
val B	0.647	0.297	0.330	0.367	0.411
test B	0.726	0.341	0.510	0.412	0.467
val M	0.563	0.453	0.560	0.455	0.502
test M	0.618	0.488	0.725	0.470	0.538

Table 7: type-predict then spoiler generation

Spoiler type counts and their type prediction confusion matrices using the [0.75, 0.45, 0.7] thresholds are shown in Table8 and Table9.

	phrase	passage	multi	mix	tot
phrase	106	36	0	35	177
passage	2	132	5	13	152
multi	1	15	44	11	71

Table 8: spoiler type confusion matrix on test

	phrase	passage	multi	mix	tot
phrase	78	35	2	43	158
passage	6	149	4	11	170
multi	5	17	37	13	72

Table 9: spoiler type confusion matrix on validation

The large difference of performance between validation and test datasets was mainly due to the distribution shift. Validation dataset has a larger proportion of "passage" spoilers, therefore increasing the difficulty of making accurate type predictions. If we remove the thresholds and analyse the confusion matrix without the "mix" model, we could discover that both "phrase" and "multi" are most likely to be misclassfied to "passage". Both "phrase" and "multi" examples had a proportion of 23% which were predicted as false "passage", whereas only 12% passages were predicted as false "phrase" or "multi". This implied "passage" data is naturally noisy in the sence that some of its examples showed moderate similarities with the other two types. Unlike the "phrase" and "multi" examles which showed good orthogonality in terms of type similarity, "passage" is more like a "mixture" of all types of spoilers. Some "passage" had a short spoiler text which is shorter than the mean length of "phrase" spoilers, therefore it is reasonable for the model to misclassify "phrase" and "multi" to a "passage" if confidence level is low. That explains why we need high confidence thresholds for "phrase"(0.75) and "multi"(0.7), these two models show much better meteor performance compare with "passage" model when they are extracting their own type of spoilers.

In Table6 we could conclude the "passage" had a similar noise level as the "mix", because "passage" had performances which were closest to "mix" model. Low confidence threshold of "passage"(0.45) reflected the fact that "passage" model was accepting the "noisy" (or hard) examples, and it was still a slightly better model compre to the "mix" in most cases.

Comparing the number of all types of spoilers in validation and test dataset, we could exaplain why test results are better than validation results in nearly all the tables, because validation has 170 passage spoilers out of 400, whereas test has a fewer number of 152. Even though best snapshots of models on the validation dataset was chosen, the deterioration of performances due to the distribution shift of having more "passages", is still prominent. Spoiler type prediction accuracy confirms the fact that validation dataset was harder to be correctly classified due to these extra "passage" examples.

We have explained the causes of bad performances of "passage" model on validation dataset, so bad case analysis was done on the errors. Fig3 showed the detailed distribution of measure metrics of examples predicted as "passage" type. The type labels in the figure are the ground truth types, the x-axis are measure metrics, y-axis are the example counts. Hence the primary reason of bad performance of "passage" predictions in Table7 can be identified as wrongly classifying "phrase" and "multi" examples to a false "passage".

Table10 showed the bleu-n and meteor of predicted spoilers with the "passage" model for each ground truth spoiler type on validation dataset. When ground truth type is "passage", the results were close to the corresponding results in Table6, which are the correctly classified results. However if the ground truth type is "phrase" or "multi", their results would be significantly lower than the results in Table6. Summing the true and false "passages", the predicted bleu-n and meteor in Table7 would be produced, giving the worst performance model which was the passage type model.

true type	bleu-n	meteor
phrase	0.174	0.286
passage	0.350	0.509
multi	0.133	0.297

Table 10: effect of false "passage" predict

Fig3 also revealed the fact that BLEU-n is not as smooth as meteor. The term "smooth" means the histogram of the metric is a more even distribution, rather than a polarized distribution, which has concentrated measure values at the low end [0,0.1] or the high end (0.9, 1.0]. This means BLEU-n is encoding less information of similarity between ground truth and predicted spoiler, because it has a lower entropy.

Human analysis of differences between BLEU-n and meteor was done by sorting $\Delta = abs\left(meteor - BleuN\right)$ in descending order for the "passage" predictions of validation dataset. Table4 showed top5 examples that the two metrics showed a largest divergence. The predicted passages tend to include the ground truth as a substring, and it is reasonable to penalize the tedious unwanted texts, but there are cases like "talk about their money" and "handling their money and talking about it with their partners", which convey essentially the same information only with slightly different way of saying. Silly mistakes for both metrics happens in short phrase spoilers. Meteor failed with single-token predicted and target spoiler, for an example $text_{prd} = text_{gt}$="ottoman", has bleu-n 1.0 and meteor 0.5. This directly caused blue-n larger than meteor value in phrase examples. Bleu-n failed with substrings such as $text_{tgt}$="Mitzi Gaynor Beverly Hills, California" and $text_{prd}$="Mitzi Gaynor", which has bleu-n 0.0 and meteor 0.4. Non-english names also failed for both measures, such as $text_{tgt}$="Carrer Avinyo" and $text_{prd}$="Carrer Avinyó", producing bleu-n 0.0 and meteor 0.25.

These are common mistakes made by metrics, the predictions are not absolutely wrong. Extracting a short phrase or passage from a long article would always encounter such a problem that a spoiler could have "isotops" which bring the same information but expressed differently. Largest common string was used during dataset repair, maybe it is also a good metric for short texts. Bert score may also be able to reduce these pseudo mistakes. It would be interesting to make a further study on designing a suitable metric.

Because of the deadline time and the GPU memory limitation, multi-task setup was considered but not experimented, it is reasonable to hypothesize that sharing the first few layers of encoder could be benifitial, further study would also be focused on this approach.

6 Conclusion

Clickbait spoiling generation can be regarded as an MRC problem, its model accepts similar input features as SQuAD. The spoiler answer types and the article paragraph layout information were important because these are newly added components towards a typical QA problem. Designs considering multi-paragraph answers produced the largest improvement in the "multi" model results using the mix as a baseline, this can be concluded by comparing Table2 and Table6, revealing improvements on bleu-n for phrase, passage and multi are 0.022, 0.115 and 0.285 respectively, and the improvements on meteor are 0.052, 0.093 and 0.159. Separating different types of models brings improvements in all models. Despite having the least similarity with the starting QA task and the least number of training examples, multi type spoiler extractor achieved the largest improvement.

Phrase type spoiler extraction is the easiest task, it has little improvement after the type splitting, and in most cases the absolute performance is the best among the all types. It is the most similar task as the starting pretrained task, experiments of adding extra input features, loss design or post-process showed little influence on its performance. The influences are too small to be distinguished within a 400 example 1-fold validation, k-fold experiments shall be done to make a convincing weak improvement, making phrase type spoiler task not as attracted as the other types.

Passage type spoiler is the most difficult in the sence that its data is the noisiest. There is not a clear feature indicating the spoiler should be a "passage" type instead of the other two types, and there is not a good general description of the passage ground truth texts. It is not a named entity, not a list of enumeration items, not a complete sentence, but it is any possible substring of the context. Therefore we encounter the most unknown inputs and outputs, results of passage model is the closest towards mix-type results, confirming the randomness of passage spoilers.

Since passage type has a relative large proportion in the dataset and a relative poor performance, more study effort was put on passage model. Fig3 revealed the most important drawback for the passage model is misclassifing spoiler type and use it to predict a "phrase" or "multi" example. Tuning the thresholds showed a trade-off between task generalization and specialization. If a model is good at a specific type of task, then as shown in Table10, its performance in a general task (where passage is the closest to the general mix) would be bad, hence we need a high threshold. Designing the training for a general task like passage spoiler extraction, is challenging and valuable, further study would be focus on this.

Bad case study also revealed some fails for both bleu-n and meteor for short spoilers with only a few tokens. For longer spoiler texts, changing the way of expressing and difference in degree of penalty of substring, are the two major causes of diversified results between the two measures. A measuring metric to bring the gaps closer shall be studied in future.

References

Maik Fröbe, Tim Gollub, Matthias Hagen, and Martin Potthast. 2023a. SemEval-2023 Task 5: Clickbait Spoiling. In *17th International Workshop on Semantic Evaluation (SemEval-2023)*.

Maik Fröbe, Matti Wiegmann, Nikolay Kolyada, Bastian Grahm, Theresa Elstner, Frank Loebe, Matthias Hagen, Benno Stein, and Martin Potthast. 2023b. Continuous Integration for Reproducible Shared Tasks with TIRA.io. In *Advances in Information Retrieval. 45th European Conference on IR Research*

(ECIR 2023), Lecture Notes in Computer Science, Berlin Heidelberg New York. Springer.

Matthias Hagen, Maik Fröbe, Artur Jurk, and Martin Potthast. 2022. Clickbait spoiling via question answering and passage retrieval. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7025–7036, Dublin, Ireland. Association for Computational Linguistics.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention.