1 Question Answering

1.1 *Code Implementation Details*

Task 1
ans_loc():
• To determine the proper start and end positions of a provided context, we first iterate over the range of number of tokens. At each position we check for a few conditions: whether our start character is equal to, or is greater than, our current start location. If so, we define our start location value. Once finding the start location, we continue this process until we find and end character token that is greater than or equal to our end location, and once found we simply exit the loop and return our identified locations.

Task 3
__Init__():
• With regards to initializing our model, everything remains the same except we add 2 Linear layers that we will use for our improved model. The first layer will allow for the initial generation of start_logits, which will then go through some processing and be passed through the second new layer, which will provide us with our end_logits.

Forward():
• For our forward method, we first want to pass in the corresponding parameters through the Transformers language model; here, I set "output_hidden_states" and "output_attentions" as True. Then, I set hidden states to be the last layer of hidden states from our output, which we also pass through a dropout layer. For the original approach, I pass the hidden states through the "self.qa_outputs" layer, from which we can extract both the start and end logits. However for my improved model, I use a slightly different approach (which is elaborated upon in Question 4). In short, I generate the start logits and use them to generate the end logits via 2 new Linear layers. For start and end loss, we simply pass in the logits and positions for both start and end.

Task 5
logits_to_ans_loc():
• For implementing the decoding strategies, I essentially use the start/end location (depending on the strategy) to generate an array of logit combinations, from which I select the "best" one using np.argmax. I continue this process over the designated range, and finish with the correct "st_loc" and "ed_loc," which is finally returned. The iterative process here allows us to continually determine the predicted locations with regards to the strategy. The two greedy algorithms have a similar structure, while the joint strategy slightly differs - further implementation details can be found within the code.

Task 6
Evaluation loop:
• Here, we simply set "input_ids" to be the concatenation of both the question and context encodings' input IDs, while the attention mask is the concatenation of both the question and context encodings' attention masks. To determine the start and end logit predictions, we simply extract them based on the provided size.

Model Results - shown in Figure 1
To summarize, my implementation of the "ans_loc" function passed all three of the test cases. With regards to the training of the model, the training loss at step 100 was 5.66, while the final training loss at step 2700 was **1.34** - lower than the designated loss threshold of 1.5.

```
------------ Test Case 1 ------------
[101, 1192, 1169, 3244, 3739, 1118, 3351, 1126, 151,
[101, 3351, 1126, 151, 1580, 1571, 7739, 102]
['[CLS]', 'You', 'can', 'protect', 'yourself', 'by',
['[CLS]', 'wearing', 'an', 'N', '##9', '##5', 'mask',
The start location is 6, and the end location is 11

Your implementation is correct for case 1

------------ Test Case 2 ------------
[101, 3325, 1114, 22311, 5912, 1105, 6284, 18608, 102
[101, 22311, 5912, 1105, 6284, 1200, 102]
['[CLS]', 'split', 'with', 'Luck', '##ett', 'and', 'R
['[CLS]', 'Luck', '##ett', 'and', 'Rob', '##er', '[SE
The start location is 3, and the end location is 7

Your implementation is correct for case 2

------------ Test Case 3 ------------
[101, 1109, 1993, 1433, 1144, 2097, 24155, 11049, 155
[101, 4805, 1550, 102]
['[CLS]', 'The', 'UK', 'government', 'has', 'spent',
['[CLS]', '250', 'million', '[SEP]']
The start location is 6, and the end location is 8

Your implementation is correct for case 3
```

```
At step 2000, the extraction loss = 1.6501818895339966
At step 2100, the extraction loss = 2.073267698287964
At step 2200, the extraction loss = 1.3850047588348389
At step 2300, the extraction loss = 1.46962308883667
At step 2400, the extraction loss = 1.4806911945343018
At step 2500, the extraction loss = 1.2253282070159912
At step 2600, the extraction loss = 1.0441079139709473
At step 2700, the extraction loss = 1.3374426364898682
Finished Training
```

(a) ans_loc (left), Training Loss (right)

Figure 1: ans_loc and Training Results

1.2 *Questions*

1) My original implementation of my model's Forward method used "lm_output.hidden_states[-1]" to retrieve the output of the top layer from my Transformer model. In order to retrieve the representations from my model's word embedding layer, I kept all other parameters the same but now set "hidden_states" to "lm_output.hidden_states[0]." With this modification, my model demonstrated the following results:

```
At step 2000, the extraction loss = 4.461519241333008
At step 2100, the extraction loss = 4.509395599365234
At step 2200, the extraction loss = 4.507562637329102
At step 2300, the extraction loss = 4.279664039611816
At step 2400, the extraction loss = 4.307399272918701
At step 2500, the extraction loss = 4.3148088455200195
At step 2600, the extraction loss = 4.3123931884765625
At step 2700, the extraction loss = 4.3853349685668945
Finished Training
```

```
Evaluating greedy_left_to_right strategy
EM = 0.03405865657521287
F1 = 0.11415025394362524

Evaluating greedy_right_to_left strategy
EM = 0.04257332071901608
F1 = 0.12106017924099632

Evaluating joint strategy
EM = 0.04039735099337748
F1 = 0.12181199826704663
```

(a) Training (left), Strategies (right)

Figure 2: Training and Strategy Results w/ Word Embedding Layer

As can be observed in Figure 2, the corresponding results for both training and strategy metrics are much worse than the original's models results. This model's final loss was 4.38 compared to the OG model's loss of 1.34; this model's Joint strategy scores were 0.04 for EM and 0.12 for F1, while the OG model's scores were 0.65 for EM and 0.78 for F1. This great discrepancy in results can be attributed to the fact that the extracted representations from the word embedding layer provide only a basic representation of context. A model is better able to "learn" relevant features the more layers it goes through, which allows for it to develop a better "understanding" of different contexts. Our original model's final layer does a great job of representing this understanding, while the word embedding layer does not provide such a comprehensive representation. Because of this, the results of the model's training and the decoding strategies suffer.

2) Strategy Results
Greedy Left to Right
EM = 0.652
F1 = 0.775

Greedy Right to Left
EM = 0.660
F1 = 0.786

Joint
EM = 0.654
F1 = 0.782

To note: all EM scores were greater than the threshold of 0.63, and all F1 scores were greater than the threshold of 0.74.

```
Evaluating greedy_left_to_right strategy
EM = 0.6521286660359508
F1 = 0.7750032056935551

Evaluating greedy_right_to_left strategy
EM = 0.660170293282876
F1 = 0.785901326538265

Evaluating joint strategy
EM = 0.6542100283822138
F1 = 0.7816396956447571
```

Looking at the results, we can see the following trend in terms of my model's performance and representative scores:

Greedy Right to Left $>$ Joint $>$ Greedy Left to Right.

This trend is preserved throughout for both EM and F1 scores, which can provide some insight into what it signifies. The EM score represents how many predicted answers are exactly the same as the annotated answers, and the F1 score represents how many words in the predicted answers overlap the annotated answers. With the Greedy Right to Left strategy having the highest scores, this means that our model slightly prefers/is best at predicting the end indeces of a resulting sequence; this finding makes sense as the strategy is centered around the end (j) of a span. Next in performance is the Joint strategy, which takes the argmax of the result using the start and end. Last in performance was the Greedy Left to Right strategy, which is centered around the start (i) of a span. Ultimately, it is important to note that there is no significant difference between the 3 strategies, as all of them performed well and are pretty similar to one another - there are only minor improvements that can be observed. Hence, we recognize that our model was adequately trained and our "logits_to_ans_loc" function was effective in decoding for all 3 strategies.

3)
a) Here, we will use the "joint" strategy to decode the provided input. Just for some context, the "joint" strategy can be represented through the following statement:

$$Select\ (i,j)\ by\colon i,j = argmax_{i,j}\ S^i_{start} + S^j_{end}\ (i \leq j)$$

What this strategy does is convert predicted start and end logits to an answer span, where the outputs are the index of the start and end tokens. In our example, we will produce the index representation of the substring that maximizes the above statement with i as the start and j as the end indeces. The strategy is called joint here as it focuses on both the start and the end rather than just one direction.

With our model, we know that we will be able to produce the substring "Tom and Jerry" with 100% probability. However, this is not the correct answer to the question as we only want an exact, singular name - either "Tom" or "Jerry." Hence, both names together means that we always get the question incorrect, and so the probability of getting it correct is **0%**.

b) Now, let's look at the probability when sampling start/end positions independently:

P1 = P(string starts with "Tom" and ends with "Tom") = 0.51 x 0.49 = 0.2499
P2 = P(string starts with "Jerry" and ends with "Jerry") = 0.49 x 0.51 = 0.2499
P(correct answer) = P1 + P2 = 0.2499 + 0.2499 = 0.4998 = **49.98%**

c) To allow for our model and strategy to return "Tom" with approx. 50% and "Jerry" with approx. 50% probabilities, we should move our focus away from exclusively start/end indices and towards spans that exclusively include the correct answer. What this will allow us to do is search for particular sections of the answer by limiting the size of the resulting span to *only include 1 or 2 words*.

Looking at the provided probability distribution, we see that only "Tom" and "Jerry" have some significance associated with them, as all other words have P(start) and P(end) of 0. Now if we look at the string "Tom and Jerry," we can discern that its length is 3 - the shortest possible length of a substring that contains both names. Hence, this means that only substrings of word length less than 3 (1 or 2 - as stated above) can include the correct answer. With this smart approach, we can confidently return "Tom" and "Jerry" each with about 50%

probability.

4)
To improve my model's performance, I focused on developing a new way to encode input questions and answers. My approach is based upon the understanding that instead of predicting both start and end positions independently in the same forward pass, I wanted to find a way that we would be able to first predict the start positions, then use this information to help predict the end positions. To do this, I first pass the "hidden_states" through a Linear layer to generate my "start_logits". Then, I concatenate these "start_logits" with the same "hidden_states" - this will allow us to condition "end_logits" on a representation of predicted start locations. So, we pass this tensor through another Linear layer (called "self.end_logit_layer" in my model) to generate "end_logits". Lastly, we squeeze both "start_logits" and "end_logits" so that the dimensionality is correct. Everything else in the model remains the same, and we continue to training.

```
At step 1500, the extraction loss = 1.3940073251724243
At step 1600, the extraction loss = 1.3940073251724243
At step 1700, the extraction loss = 1.8431096076965332
At step 1800, the extraction loss = 0.933843195438385
At step 1900, the extraction loss = 1.0870977640151978
At step 2000, the extraction loss = 1.868841290473938
At step 2100, the extraction loss = 1.2027124166488647
At step 2200, the extraction loss = 1.8418183326721191
At step 2300, the extraction loss = 1.7006287574768066
At step 2400, the extraction loss = 0.9311920404434204
At step 2500, the extraction loss = 1.6487997770309448
At step 2600, the extraction loss = 1.1889636516571045
At step 2700, the extraction loss = 1.0971829891204834
Finished Training
```

```
Evaluating greedy_left_to_right strategy
EM = 0.659035004730369
F1 = 0.7839155169111474

Evaluating greedy_right_to_left strategy
EM = 0.6631031220435194
F1 = 0.7920159203781406

Evaluating joint strategy
EM = 0.6618732261116367
F1 = 0.7902885422706988
```

(a) Training (left), Strategies (right)

Figure 3: Training and Strategy Results w/ Improved Model

Looking at the results in Figure 3, we observe that there are improvements to both the training loss and the resulting scores for our decoding strategies. Our final training loss is **1.09**, which is less than our original training loss of 1.34.

With regards to the strategies, all EM and F1 scores have increased for each strategy. For the Greedy Left to Right strategy, the EM score is now **0.659** (compared to the prev. score of 0.652), while the F1 score is now **0.784** (compared to the prev. score of 0.775); for the Greedy Right to Left strategy, the EM score is now **0.663** (compared to the prev. score of 0.660), while the F1 score is now **0.792** (compared to the prev. score of 0.786); for the Joint strategy, the EM score is now **0.662** (compared to the prev. score of 0.654), while the F1 score is now **0.790** (compared to the prev. score of 0.781).

Hence when we see all of these results, it is clear that though not overwhelming, there certainly is improvement in all aspects of performance when compared to our baseline model.