Options:

1. Averaging all plausiblity scores and setting it as a threshold. Threshold would be found by dividing all questions into two equal parts (low and high average). Then we use threshold to pick up randomly confusing and non-confusing questions.
Pros:
Nothing arbitrary at all. Just working with averages.
Cons:
We gonna miss questions that has couple really plausible answers but has a low average at all.
- We can also filter questions if the top answer is higher than the average value for the highest asnwers.

2. **Selection based on Combined approach (two conditions used at same time)**
We consider question as confusing if:

- its avg plausability of answers is higher than *avg plausability threshold* (decided by option 1 above)
- it has at least one answer higher than *top answer threshold (computed by averaging plausability scores of top answers and dividing data into 2 equal parts)*

3. **Selection focused on top plausability scores**
We consider question as confusing if:

- *it has at least one answer higher than top answer's threshold (computed by averaging plausability scores of top answers and dividing data into 2 equal parts)*

Then we take all the answers above that threshold as explanatory options for the target question.

Todo:

- label each question as confusing or not for all PlausibleQA questions to see the distribution, and also distribution for how many answers are picked up in the set of confusing questions
- add the clarify doubts system prompt. something really similar to baseline.
- also formal definiton must be like the example below
- think about the option 4

4. **Real user judgments based selection**

- We randomly select n questions
- We show each of them to users to decide if a questions is confusing or not. In particular, we show the options including gold answer but not show the plausability scores, and we ask them to pick up the correct answer.
- Then we estimate threshold based on top answers' plausability or based on.....

Let $x \in X$ denote an OCRed input sentence, and
let $e \in E$ be the corresponding output of an IE
system (e.g., predicted entity tags or entity links).

The true performance metric (e.g., F1 score) for this input-output pair is denoted by $y \in [0, 1]$, and our objective is to learn a function $p\theta : X \times E \rightarrow [0, 1]$, parameterized by θ, such that:

$\hat{y} = p\theta(x, e) \approx F1(e, x)$ (1)

This formulation casts the performance estimation problem as a regression task, where the model predicts the evaluation score directly from the input-output pair.