Model documentation and write-up

1. Who are you (mini-bio) and what do you do professionally?

Mikhail Kulyabin received an Engineer's degree in mechanical engineering from Bauman Moscow State Technical University (BMSTU) and a M.Sc. degree in computational engineering from Friedrich-Alexander-Nürnberg (FAU), where he is currently pursuing the Ph.D. in computer science. His main research interest lies in applying AI to topics related to ophthalmology.

Gleb Sokolov is an experienced deep learning engineer with a strong background in computer vision. He is a Kaggle grandmaster.

Aleksandr Galaida received a Bachelor's degree in Computer Science from Moscow Institute of Physics and Technology State University (MIPT). He is an experienced computer vision engineer specializing in R&D tasks, having worked for more than five years in the scientific laboratory and product teams.

2. What motivated you to compete in this challenge?

Interest in a new area for us, gaining skill in working with language models.

3. High level summary of your approach: what did you do and why?

The solution consists of two stages:
First stage: train NER segmentation task with four classes ('find', 'proc', 'body', 'none');
Second stage: for each span from the first stage, predict its SNOMED ID.

4. Do you have any useful charts, graphs, or visualizations from the process?

-

5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

Get a database of embeddings:

```
# pseudocode
ID2EMB = {}
for id in all_ids:
    embeds = []
    for synonym in get_all_synonyms(id):
        embed = embedder(synonym)
        embeds.append(embed)
    true_embed = embeds.mean(0)
    ID2EMB[id] = true_embed
```

Match extracted mentions (from stage 1) with database:

```
# pseudocode
for mention in predicted_mentions:
    qvec = embed(mention)
    similarities = qvec @ ID2EMB.values()
    top1_idx = argsort(similarities)
    qID = ID2EMB.keys()[top1_idx]
```

6. Please provide the machine specs and time you used to run your model.

- CPU (model): AMD EPYC 7713
- GPU (model or N/A): NVIDIA A100-SXM4-40GB
- Memory (GB): 200 GB
- OS: Linux
- Train duration: 60 minutes + 24 hours for MLM pretraining (Optional)
- Inference duration: 1 minute

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

-

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

-

9. How did you evaluate performance of the model other than the provided metric, if at all?

We used 4-Fold validation on the annotated dataset.

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

-

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

1. Approach:
First of all, we used a two-stage method:
   1. NER task
   2. Linking

We did not have an opportunity to learn the problem end to end. One should try to integrate these two into one (https://aclanthology.org/2022.naacl-industry.24.pdf ReFinED, for instance). This would require many data to train, but it would be of better quality.

2. Data:
There should be more data! Synthetic data generation with SOTA generative models is a quick and easy way to try a more-data-regime. We did a bunch of early EDA on synthetics but stopped halfway through.

3. Better stages:
If continuing with 2 stage approach, one can improve both stages:

3.1 NER - larger models, more granular segmentation task (0, Body, Procedure, Findings, X, Y, Z). It probably needs some domain expertise.

3.2 Linker:
- We did not train any second-stage model, and it definitely would help.
- Try some other arches of the encoder. We choose late-interaction models and search based only on embeddings, but more options exist (see COLBERT paper, for example).

3.3 add Reranker as 3rd stage:
We build a reranker model based on https://github.com/ncbi/MedCPT.
It helps with the linking accuracy, but it is hard to maintain such a sizeable three-staged solution, and the benefits of IoU are more challenging to track.

For example, on perfect (GT) predictions from the first stage:
SAPBERT linker
Top 1 accuracy for BODY IDs: ~0.62;
Top 5 accuracy for BODY IDs: ~0.75.
If we implement a third stage with reranking candidates from the top 5, the possible improvement is around 0.03 (experimentally).

4. Static dictionary:
As we understand, the first-place team used a simple dict-solution based on human knowledge. This was the right approach for results, but it still needs generalization. A dict-solution would obviously not perform for zero-shot tasks at any capability. So, some integration of static and DL approaches is necessary.

12. What simplifications could be made to run your solution faster without sacrificing significant accuracy?

It is possible to run without ensemble (we used six models for the final score) and without MLM pretraining.