Daria Galteva

JetBrains AI Evaluation Internship Test Task

1) Preparing the dataset

First, I had to decide how exactly to split files:

- Should I aim for completing the whole block or just a current line;
- Should comments be completed;
- Should I sample a point in file or sample a line and then sample a point inside that line;
- What if special tokens like <fim_middle> or eos token are used in the code;

Considering the specifics of fill-in-the-middle task I decided to end the "middle" part at the end of the line or, if cursor is inside of an expression in brackets and the closing bracket is on that line, at the position of closing bracket => filling until the end of line or until the end of expression.

I sampled points from files uniformly, even though that means that we will have more samples from longer lines (otherwise prior probabilities of samples from shorter lines would be bigger).

To deal with special tokens it would be possible to replace them during preprocessing and then to put them back after generation (if used), but I did not implement it.

Code to sample a dataset from a list of files is in **get_samples.py** and it should work with default parameters.

Dataset structure:

Dataset is saved in json format: list of dictionaries for each file that was used for dataset creation:

[{"filename": str, "text": str, "samples": list}, …]

Each sample is a dictionary with start position of middle part in the code, end position of middle part and a middle part itself (even though it is not really needed, because it can be extracted from "text" field of corresponding file entry):

"samples": [{"middle": str, "middle_start": int, "middle_end": int}, …]

After running a model, fields "generation_result" and "metrics" with a dictionary of metrics are added.

After annotating, fields "annotations" with text annotations and "label" with float label are added.

Example of accessing chrf++ metric of first sample from first file:

dataset[0]['samples'][0]['metrics']['chrf++']

2) Running a generation pipeline and computing metrics

To generate the middle part, I used all code above and below the missed part in the form <fim_prefix>prefix<fim_suffix>suffix<fim_middle>. **run_model.py** will print a warning if tokenized input + *max_new_tokens* will exceed the model limit.

I tried a few different values for *max_new_tokens,* and it seems that *tiny_starcoder* prefers to generate longer outputs or generate code that is already in suffix. A lot of the generated examples would be perfect if generation stopped earlier. But some examples were longer than, for example, 10 tokens, so I limited *max_new_tokens* to 25.

For evaluation I selected the following metrics:

- Exact_match – indicates that user can just press tab and be satisfied (but not all the cases, because exact_match is sensitive to any minor changes). But if it is 1, it is really good.
- ChrF – character-level n-gram F-score, good for code generation task because works on character level
- ChrF++ - also considers word n-grams of order 2. According to an article referred to in huggingface documentation ChrF++ produces scores that correlate better with human judgements than chrF.
- Syntactic correctness – whether resulting completed code is syntactically correct. Instead of this some automatic tests can be used.
- Levenshtein ratio – it is based on Levenstein distance. The idea is that the more we need to correct generated code, the worse the result is.
- LCS – same idea as before, if a large piece of code matches, then it is probably good (and maybe we can just delete the rest).
- Normalized LCS – solves the problem of LCS sensibility to the length of reference text.

3) Annotation

For annotation I used script **annotate.py**. It can be run with the resulting dataset for a little better visualization. It displays the code with the middle part colored blue and generated output colored green. It also prints all the metrics. Existing annotations and labels can be edited.

As I already mentioned, the model often generated correct output but then continued to generate code from suffix. Most of these examples have labels 0.6 (bigger than 0.5 but still not so good).

To get binary labels some threshold can be used (I annotated examples with a threshold of 0.5 in mind, so more than 0.5 are in my opinion somewhat good).

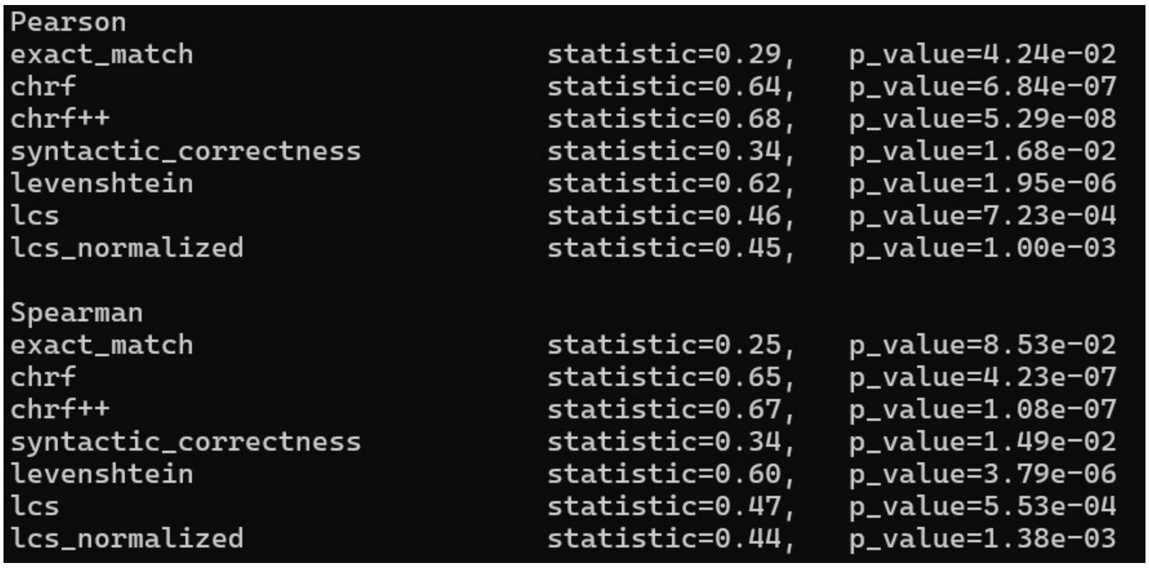
To improve generation, outputs can be cleaned at least from the code that is exactly like in the first few lines in the suffix.

4) Correlation

Script **correlation.py** should be run with parameter *--file_name resulting_dataset.json*

I measured Pearson and Spearman correlation (Figure 1) between labels that I assigned and metrics. Because of the nature of the labels (I would say that they represent my ranking of the samples) I trust Spearman correlation more, but it corresponds to Pearson quite good as well.

ChrF++ correlates best with my judgement, slightly better than ChrF as expected. I guess, because of how I annotated examples (with the idea of how much user should correct proposed code), Levenshtein ratio correlates quite good as well. I would say that I trust ChrF++, but in my opinion all these metrics should be somehow considered.



```
Pearson
exact_match                     statistic=0.29,   p_value=4.24e-02
chrf                            statistic=0.64,   p_value=6.84e-07
chrf++                          statistic=0.68,   p_value=5.29e-08
syntactic_correctness           statistic=0.34,   p_value=1.68e-02
levenshtein                     statistic=0.62,   p_value=1.95e-06
lcs                             statistic=0.46,   p_value=7.23e-04
lcs_normalized                  statistic=0.45,   p_value=1.00e-03

Spearman
exact_match                     statistic=0.25,   p_value=8.53e-02
chrf                            statistic=0.65,   p_value=4.23e-07
chrf++                          statistic=0.67,   p_value=1.08e-07
syntactic_correctness           statistic=0.34,   p_value=1.49e-02
levenshtein                     statistic=0.60,   p_value=3.79e-06
lcs                             statistic=0.47,   p_value=5.53e-04
lcs_normalized                  statistic=0.44,   p_value=1.38e-03
```

Figure 1-correlation results

python3 annotate.py --file_name resulting_dataset.json can be used to visualize the dataset. Samples are displayed one by one; it is enough to press enter 2 times (because the code will propose a possibility to redact annotations and labels.