POKT AI Lab

Report II

Authors:
Nicolas Aguirre (nicolas@poktscan.com)
Ramiro Rodríguez Colmeiro (ramiro@poktscan.com)
POKTscan Data Science Team

May 30, 2024

Disclaimer

This work was partially funded by Pocket Scan Technologies LLC, a company operating on the Pocket Network.

Changelog

v1.0	2024-05-30	First version.

Table of Contents

1 Progress Overview

During the month of May, the socket achieved the following milestones:

- Merged issues on the Test-Bench code:
  - Manager:
    - Add buffers logic [1].
    - Implemented circular buffers logic for metrics [2].
    - Task Triggering [3].
  - Sampler: Generate prompts and save to mongo + Tokenizer [4].
  - Requester: Complete requester MVP [5].
  - General: Fixes and enhancements to create the first version of Morse POC [6].

[1] https://github.com/pokt-scan/pocket-ml-testbench/issues/36
[2] https://github.com/pokt-scan/pocket-ml-testbench/pull/38
[3] https://github.com/pokt-scan/pocket-ml-testbench/issues/40
[4] https://github.com/pokt-scan/pocket-ml-testbench/pull/47
[5] https://github.com/pokt-scan/pocket-ml-testbench/pull/37
[6] https://github.com/pokt-scan/pocket-ml-testbench/pull/51

2 Language Models - Tokenizer

In this sections is briefly introduced the concept of *token*, *tokenization* and *tokenizer*, how it works in a language model (LM), an issue when running a decentralized benchmarking, and finally a proposed solution.

2.1 From string to token

Tokenization is the process of representing text in smaller meaningful lexical units. There are three popular ways to do the same, which are: 1) word tokenization, 2) subword tokenization, and 3) character tokenization. Word tokenization is the simplest form of tokenization, where each word is considered a token. Subword tokenization breaks words into subwords and treats them as tokens. Character tokenization considers each character as a token. There is a trade-off between the granularity of the tokens and the number of tokens. That is, while word tokenization has low granularity and high token count, character tokenization has high granularity and low token count. Nowadays, the most popular tokenization method is subword tokenization, which is a compromise between word and character tokenization. It is important to note that prior to training a model, tokenizers must be trained on texts that is representative of the training set (ideally the tokenizer texts should not be part of the training set). With respect to the tokenizer trainin method it's worth to mention that there exist several techniques [3, 2, 4]

On the other hand, the tokenizer is the one in charge of carrying out the tokenization process, which consists (in a nutshell) of a dictionary that contains pairs of tokens and their numerical representation. For this reason, a direct relationship is established between a LM and its tokenizer, which is crucial when evaluating the performance of a LM since one of the main techniques for evaluating a language LM is through the calculation of the probability of occurrence of a sequence of tokens that is conditioned by an initial sequence of tokens. In terms of `lm-eval-harness`[1] these two sequences are also called `context` and `continuation`, as shown in Figure 1.

Without going into the mathematical details, and following the Figure 1, the concatenation of the `context` and `continuation` tokens is sent to the language models, and from the LM returns it can be computed the probability of occurrence of the entire phrase detailed token by token. The LM is considered to respond correctly when the probability of occurrence of the correct `continuation` is the highest with respect to the other continuations. That is, in terms of the Figure 1, the LM responds incorrectly, since C_2 should have been the `continuation` with the highest probability, but the LM returns a higher probability for C_1.

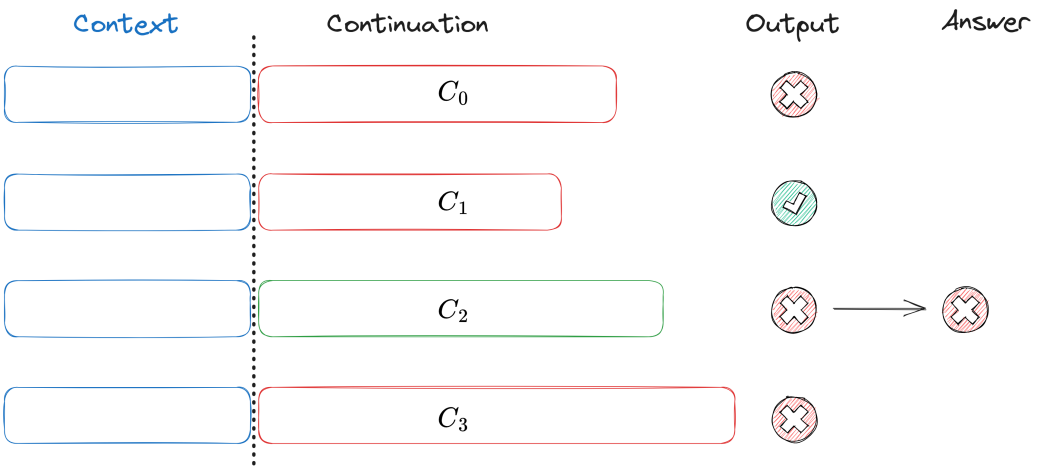

Figure 1: Example of `context` and `continuation` token sequences.

2.2 On the lack of knowledge of the tokenizer

As previously detailed, the LM receives the concatenation of the `context` and `continuation` token sequences, and then locally separates the `context` and `continuation` probabilities. However, in a decentralized environment, such as that proposed by benchmarking, if you do not have access to the tokenizer, it is not possible to separate the `context` and `continuation` probabilities. The plain text of the concatenation of contex and `continuation` could be sent, but then we could not directly determine which part of the text belongs to the `context` and which part belongs to the continuation, thus preventing the evaluation of the model. Although one could try to reconstruct the `context` and `continuation` phrase from the concatenation of tokens, this is not a trivial task, since the tokenizer could have tokenized differently than the original text was tokenized [1].

2.3 Proposed solution

Faced with this situation, there are three other plausible solutions. The first solution would be to send, in addition to each complete sentence, only the `context` text. This would then allow us to remove those corresponding to the `context` from each of the complete sentences and continue with the evaluation of the model. The disadvantage of this solution is that $N+1$ calls would be made to the model, where N is the number of complete sentences to evaluate, resulting in a higher cost. Furthermore, in the case of evaluations that contemplate the use of conversational models (such as the case of chat), this solution would not be applicable, since internally these models through its toke add special tokens to indicate the beginning and end of a message in a conversation, just as they usually have a text to condition the model's response, called system prompt. The second solution would be online tokenization, calling for example an endpoint *tokenize*. This situation would face similar

problems, as currently discussed in the community [7]. The third solution would involve sending the complet tokenizer associated with an node address. This was detailed in two PRs [8] [9] in both the lm-eval-harness repository and the vLLM repository, popular LM inference engine. The main advantage of this solution is that the issues of the first and second solutions are avoided. Additionally, as previously detailed, the tokenizer is tied to the model. Therefore we can generate a hash of the tokenizer that sends us an address, save it in a database and then verify that the hash of the tokenizer that is sent at the beginning of a session is equal to the hash stored in the database. If for some reason the hash does not match, it is an unequivocal indication that the LM behind the endpoint is not the same, it has changed, therefore the previous results are not valid, metrics are eliminated from the leaderboard, and that address has to be re-evaluated.

[7] https://github.com/huggingface/text-generation-inference/issues/1706
[8] https://github.com/EleutherAI/lm-evaluation-harness/pull/1794
[9] https://github.com/vllm-project/vllm/pull/2643

3 Future Work

During the month of May we focused on merging the parts developed until now individually developed. This included the `Manager`, `Sampler` and `Requester`, plus a single `docker-compose` that includes a localnet composed of three `Lean-nodes` with eight validators each and a `Geo-Mesh` for testing that everything works.

During the month of June we hope to complete the Machine Learning Test-Bench (MLTB) code, including:

1. `Manager`:
    a. Create logic for tokenizer signature task [10].
    b. Clean completed tasks [11].
    c. Add signatures to node [12].
2. `Sampler`:
    a. Create logic for tokenizer signature task [13].
    b. Add asynchronous read operations to the MongoDB collections.
3. `Evaluator`: Create initial code for LMEH processing [14].

[10] https://github.com/pokt-scan/pocket-ml-testbench/issues/42
[11] https://github.com/pokt-scan/pocket-ml-testbench/issues/41
[12] https://github.com/pokt-scan/pocket-ml-testbench/issues/39
[13] https://github.com/pokt-scan/pocket-ml-testbench/issues/43
[14] https://github.com/pokt-scan/pocket-ml-testbench/issues/44

List of Acronyms

LM	language model
MLTB	Machine Learning Test-Bench

Reference List

[1] Stella Biderman et al. *Lessons from the Trenches on Reproducible Evaluation of Language Models*. May 23, 2024. DOI: 10.48550/arXiv.2405.14782. arXiv: 2405.14782[cs]. URL: http://arxiv.org/abs/2405.14782 (visited on 05/29/2024).

[2] Taku Kudo. *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates*. Apr. 29, 2018. DOI: 10.48550/arXiv.1804.10959. arXiv: 1804.10959[cs]. URL: http://arxiv.org/abs/1804.10959 (visited on 05/29/2024).

[3] Taku Kudo and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. Aug. 19, 2018. DOI: 10.48550/arXiv.1808.06226. arXiv: 1808.06226[cs]. URL: http://arxiv.org/abs/1808.06226 (visited on 05/29/2024).

[4] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. June 10, 2016. DOI: 10.48550/arXiv.1508.07909. arXiv: 1508.07909[cs]. URL: http://arxiv.org/abs/1508.07909 (visited on 05/29/2024).