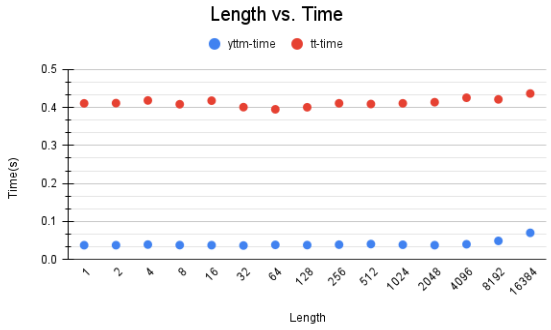
CS 744 Project Check-in

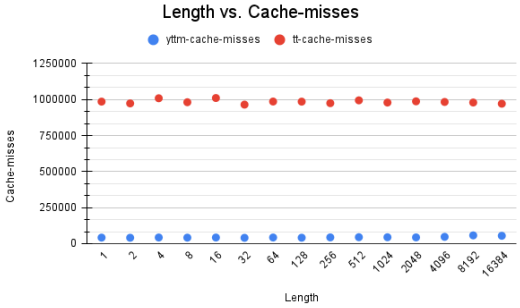
Group Number: 10

a. What have you done so far

- Setup **Tiktoken** and **YouTokenToMe** open-source tokenizers on CloudLab machines
- Scanned through relevant datasets and finalized **Flores-200** in collaboration with a fellow group working on the **HuggingFace** tokenizer
- Trained YouTokenToMe on an English dataset of randomly generated words
- Evaluated YouTokenToMe tokenizer with TikToken tokenizer being the baseline against varying numbers of words per sentence using ***perf-stat*** to collect statistics like **L1, LLC misses and hits**, **tokenization latency**, and the **number of CPU instructions**

Our findings :





b. Any challenges that you have faced so far

Currently, we are passing sentences directly in a Linux command that executes the tokenizer. The command executes successfully for sentences with word lengths up to 16k. However, when dealing with longer word lengths, an error occurs due to the command exceeding its argument limit. Thus, we're exploring solutions to process sentences exceeding the 16k limit.

Our approach (under consideration) is specifying a file name within the Linux command. Subsequently, within the tokenizer script, we would read the file to access the sentences. However, we're still working on isolating the results and statistics specific to the tokenizer, as the current proposed solution would include the file-reading process in the statistics.

Additionally, the YouTokenToMe tokenizer requires the user to specifically train on a desired language's dataset, as it does not offer pre-trained tokenizers.

c. Your timeline (from now till the end of the semester)

We have ~4 weeks remaining for the project. We plan to use it in the following way:

- Week 1: Evaluate performance across Hindi and Russian languages and compare with the English results we have
- Week 2: Experiment with ***dvfs*** (open source package) to limit the compute resource allocated to the tokenizer and limit the memory allocated using **Docker Containers** to determine whether the tokenizers are memory-intensive or compute-intensive. Finally, plotting a roofline analysis graph
- Week 3: Compile and evaluate statistics across parameters (languages, number of tokens, tokenizers) to identify trends and patterns
- Week 4: Document our findings and publish the results

d. Things you need help from the course staff

We are currently utilizing the ***perf-stat*** command line tool in Linux in order to get statistics regarding the performance of our tokenizer in terms of overall **cache hits/misses**. This tool also gives us cache statistics at a more granular level in terms of L1, L2, and LLC (Last Level Cache) cache hits/misses. However, we have been unable to obtain statistics for the L2 cache as of now as we receive a warning that measuring this statistic is not supported whenever we try to evaluate the tokenizer.

We are currently able to pass sentences as a command line argument to the tokenizer programs. However, this approach only allows us to send sentences whose word length is <= 16K.
If we want to send larger sentences, we were considering writing these sentences into a file and then reading them from the file in the tokenizer program. However, this approach would lead to an overhead for reading from the file and may impact the statistics like L1/L3 cache stats and total time taken. Therefore, we need to figure out an approach to isolate the performance of just the encoding part of the tokenizer.

In addition to this, it would also be helpful to get some assistance with evaluating the graphs and results we have obtained so far and also some general guidance on the direction in which to proceed.