Evaluating Tokenizers for Memory, Compute and Cache Efficiency: YouTokenToMe v/s TikToken

Abhijeeth, Divy, Smit, Sujay

Problem Statement

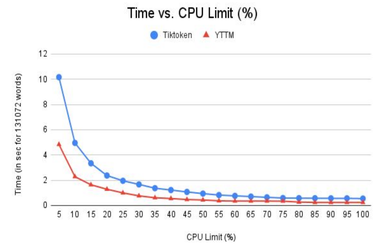
- Compare several existing tokenizers and understand the performance bottleneck
- To analyze compute and memory traffic. This includes profiling cpu usage, cache hits, misses, and memory accesses across cache levels

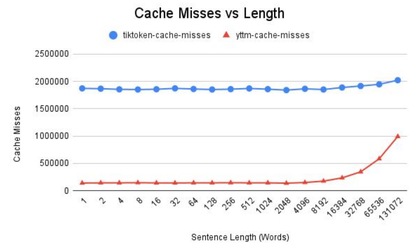
Related work

- Researchers have compared multilingual models with monolingual models by testing them on tasks designed for one language, using different tokenizers
- They use measures like F1 score and accuracy, to evaluate the performance of various downstream models in context with monolingual tokenizers utilized upstream
- There has been some work evaluating the compression factor of various tokenizers. They show how tokenizers with higher compression rates can increase the context lengths of the models and also affects the downstream memory usage and compute usage
- None of the recent work evaluates the memory and cache efficiency of tokenizers
- To verify the system level performance and identify the bottlenecks there is no existing work for the same

Observations

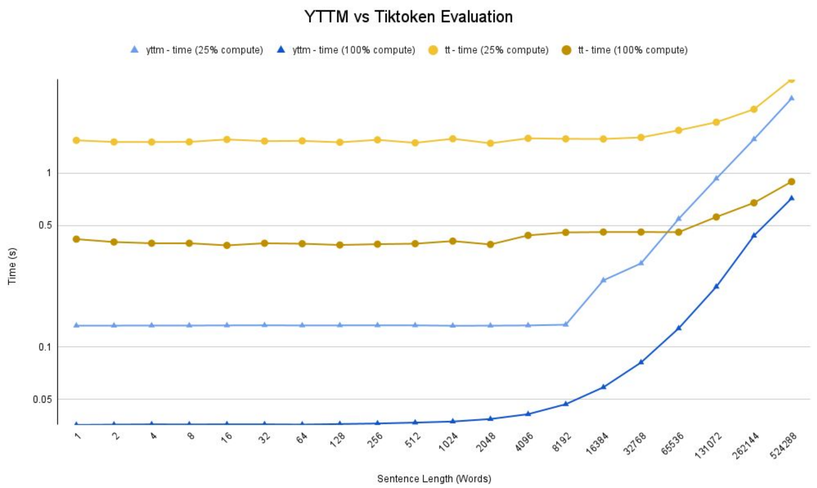
- YouTokenToMe is 10x faster at smaller lengths (till 8k), but the gap reduces as the length increases
- Limiting compute impacts YouTokenToMe more than Tiktoken
- Cache Misses increase more rapidly with sentence length in YouTokenToMe compared to Tiktoken
- In the case of Tiktoken, the time required for tokenization is largely determined by the encoding load time for smaller sentences





- We employed *perf stat* to compute statistics for sentences of varying lengths. To minimize variance, we profiled 50 different sentences, all of identical length
- We used the enhanced *cpu limit* tool from *opsengine* for limiting compute

Early Results



- Setup two open source tokenizers YouTokenToMe and TikToken and profile using *cpulimit, perfstat*

Challenges / WIP

- To scale our approach to sentences having > 64k words, we utilized a file based I/O approach which resulted in additional overhead but scaled well
- Exploring memory usage and analysis for calculating statistics using *cgroups, dvfs*
- Comparing performance across multiple languages like English, Russian and Hindi across varying sentence lengths

WISCONSIN
UNIVERSITY OF WISCONSIN-MADISON