Llama2 7b Benchmarks

January 17, 2024

Luc Cary

1 Llama2-7b Wikilingia Experiments

These experiments benchmark the time required for Llama2-7b to generate summaries on Wikilingua articles.

1.0.1 Constraints

The below experiments were run for the task of generating text summaries for each article with the following constraints:

- Number of GPUs: 1
- 300 maximum generation tokens
- No input truncation

1.0.2 Optimizations

The experiments tested multiple optimization strategies on the LLM:

- `4-bit`: 4-bit Quantization
- `8-bit`: 8-bit Quantization
- `flash-attn`: Flash Attention
- `none`: Default model loader (no optimization) (no optimizations)
- `vllm-awq`: vLLM with AutoAWQ quantization

1.0.3 Hardware

For the GPU requirement, these experiments were run on the following instance types:

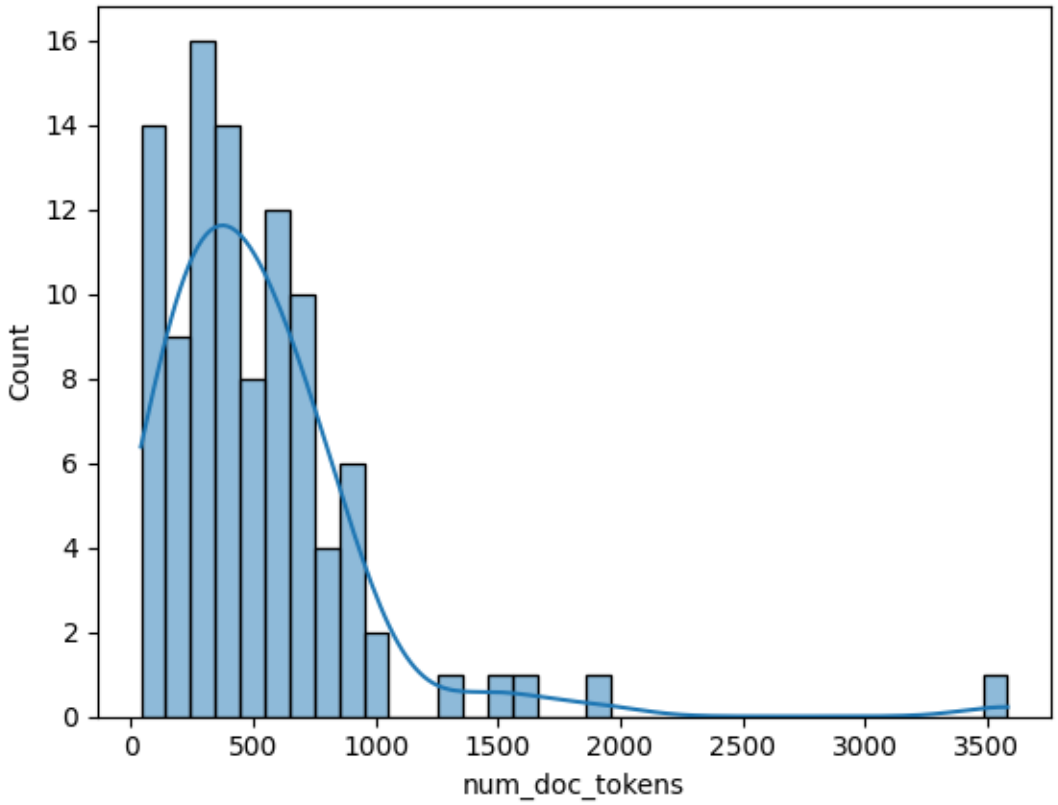
- A100 GPU Server (40GB GPU RAM)
- T4 GPU Server (15GB GPU RAM)

All scripts were run on Google Colab. Before each experiment, all Python dependencies were installed or updated and the runtime was restarted to clear memory.

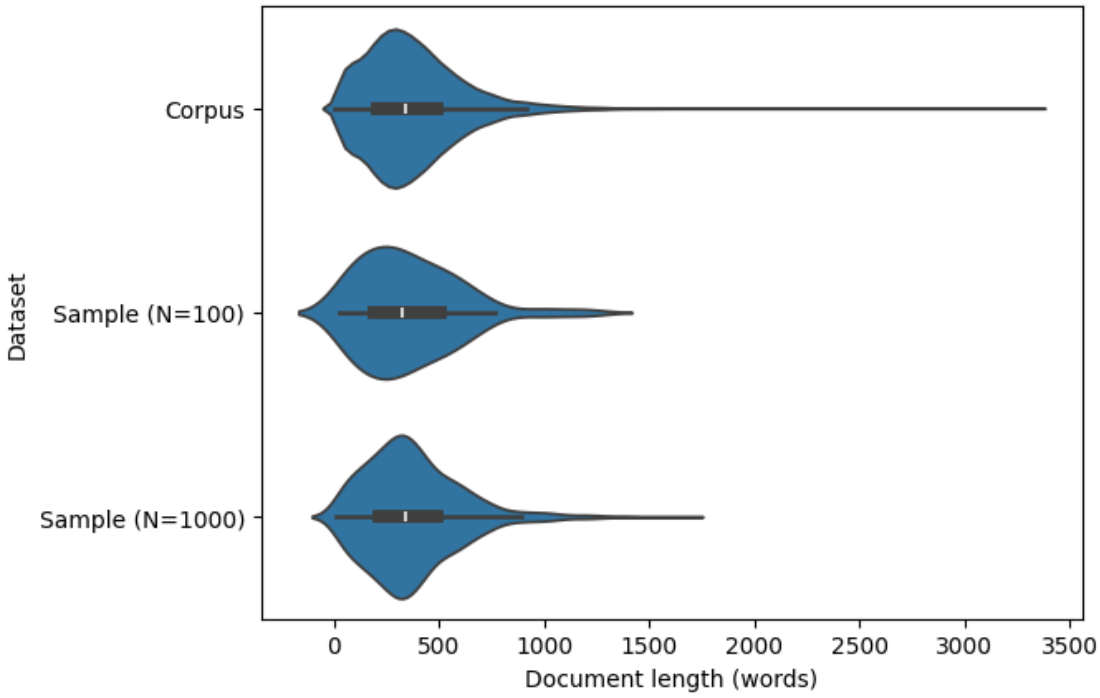
1.0.4 Dataset

- 100 randomly sampled documents from Wikilingua's `english` corpus (same set for all experiments)
- Average document length for the sample: 515.4 tokens

For the set of sampled article URLs in this experiment dataset, see `data/100_articles.txt`.



The above histogram shows the number of tokens per document and kernel density estimate of token count in the sample dataset of 100 documents, using the Llama2 tokenizer to encode text to tokens. Repeated sampling from the broader corpus shows that this sample is representative of the document lengths in the corpus, as shown in the below violin plot comparing counts of words between datasets:



One primary distinguishing feature of the corpus is the existence of some longer documents (up to 3,000 words). Since these documents are rare, the datasets have slightly smaller documents. However, the sample datasets used in this experiment had mean word counts that were similar to the overall Wikilingua corpus, which has a mean of 373 words.

	Dataset	Document length (words)
0	Corpus	373.412169
1	Sample (N=100)	356.250000
2	Sample (N=1000)	370.758000

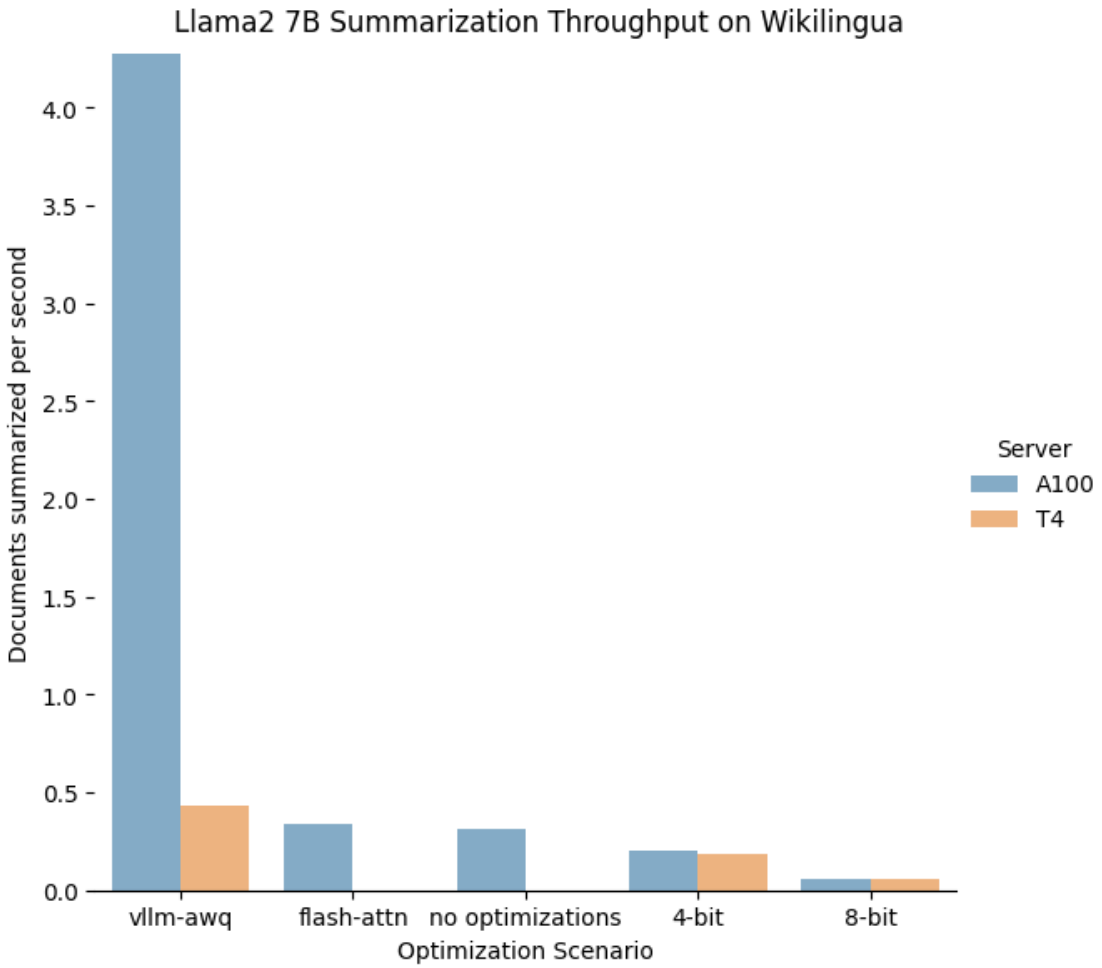
Note that words can consist of multiple tokens, so their relationship is not 1-to-1.

1.1 Results

1.1.1 Summary

The vLLM optimization had the fastest throughput, allowing the Llama2 7-billion parameter model to process 4.273 documents per second. This strategy when executed on the A100 was an order of magnitude faster than any other strategy across all hardware types. Using vLLM has less of an effect on the lower-memory T4 GPU, but still resulted in a 2x speedup over other T4 experiments, making the T4 competitive with other A100 experiments.

See the below chart for a visual comparison of inference speed, measured by the average number of documents processed per second.



The same results can be found below in table format, ordered by the number of documents processed per second by the model:

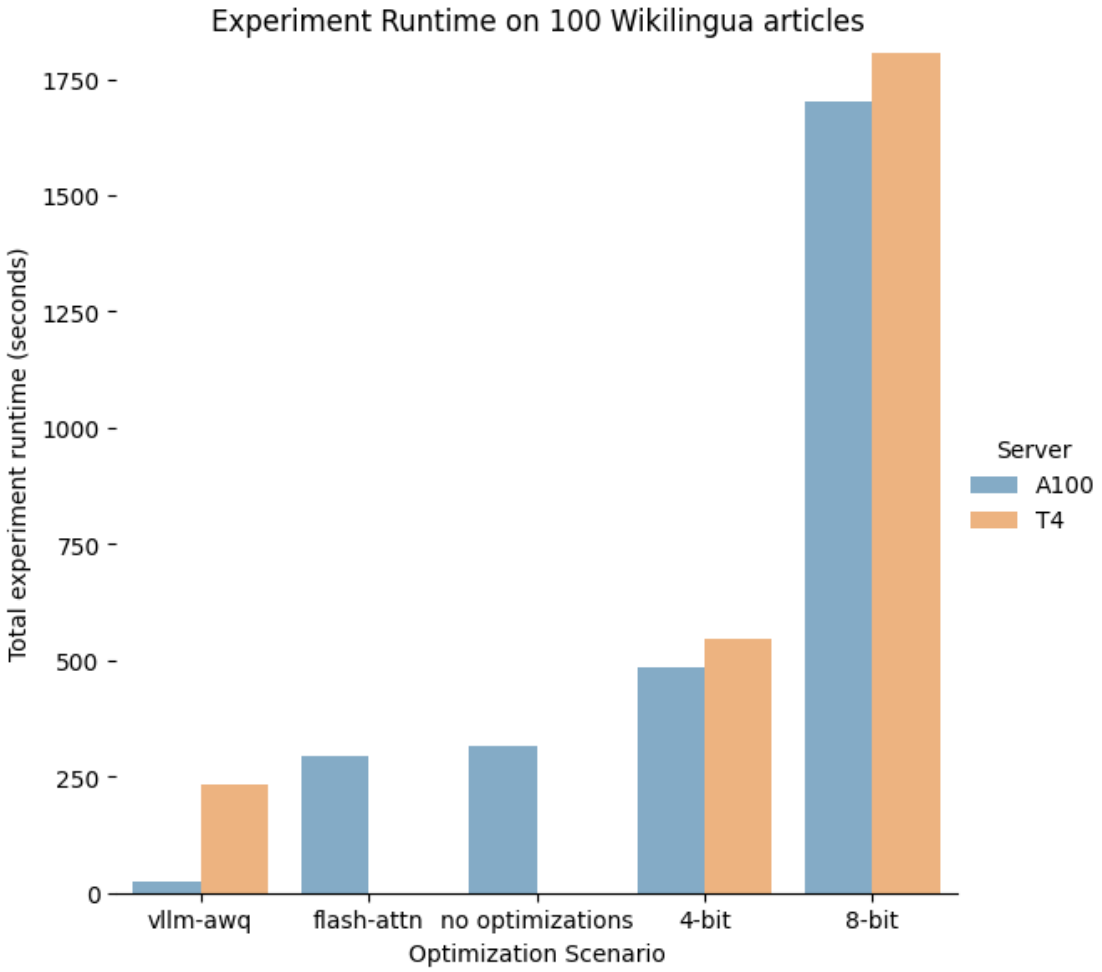
Experiment ID	Optimization	Docs/Second	Instance Type	ROGUE-1 F1
25	`vllm-awq`	**4.273**	`A100`	0.234
28	`vllm-awq`	**0.540**	`T4`	0.254
16	`flash-attn`	0.339	`A100`	0.282
17	`none`	0.317	`A100`	0.280
27	`4-bit`	0.206	`A100`	0.264
22	`4-bit`	0.183	`T4`	0.251
29	`8-bit`	0.059	`A100`	0.267
23	`8-bit`	0.055	`T4`	0.275

1.2 Interpretation

The use of PagedAttention, automatic batching of inputs, quantization of model weights, CUDA execution graph utilization and CUDA kernel optimizations, amongst other techniques, set vLLM aside as the obvious winner in this experiment, particularly for the A100 GPU. Other scenarios like Flash Attention or removal of optimizations were not possible to run on the T4 GPU due to memory constraints and out of memory errors when processing the 100 document dataset without truncation of inputs or decreasing generated output lengths.

For 4-bit and 8-bit quantization of the LLM, the T4 and A100 inference speed was roughly equal. These quantization techniques allowed the T4 GPU to finish generation of summaries by reducing the memory footprint of the model.

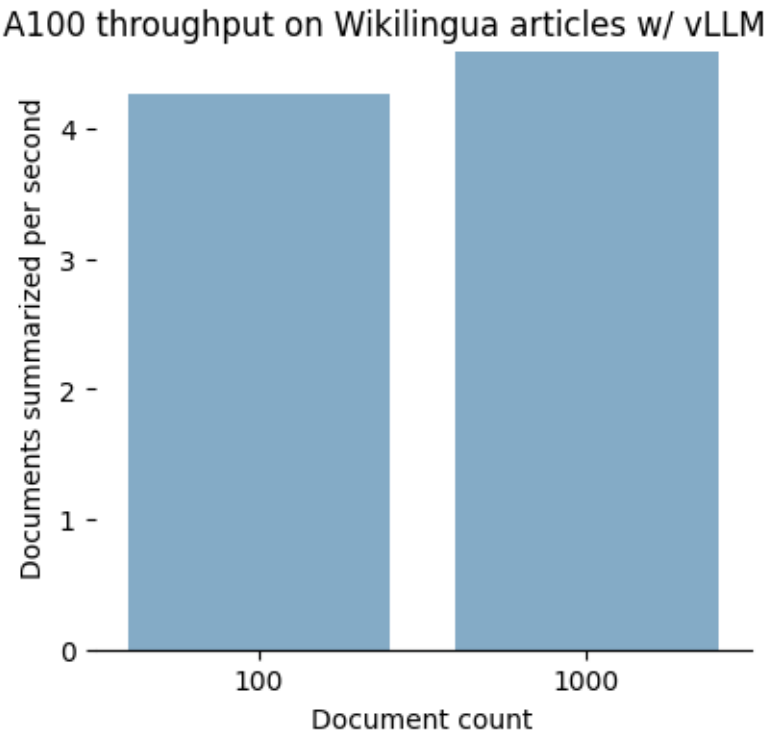
Another way to visualize this information is in the total time to generate summaries for 100 documents (including tokenization and loading):



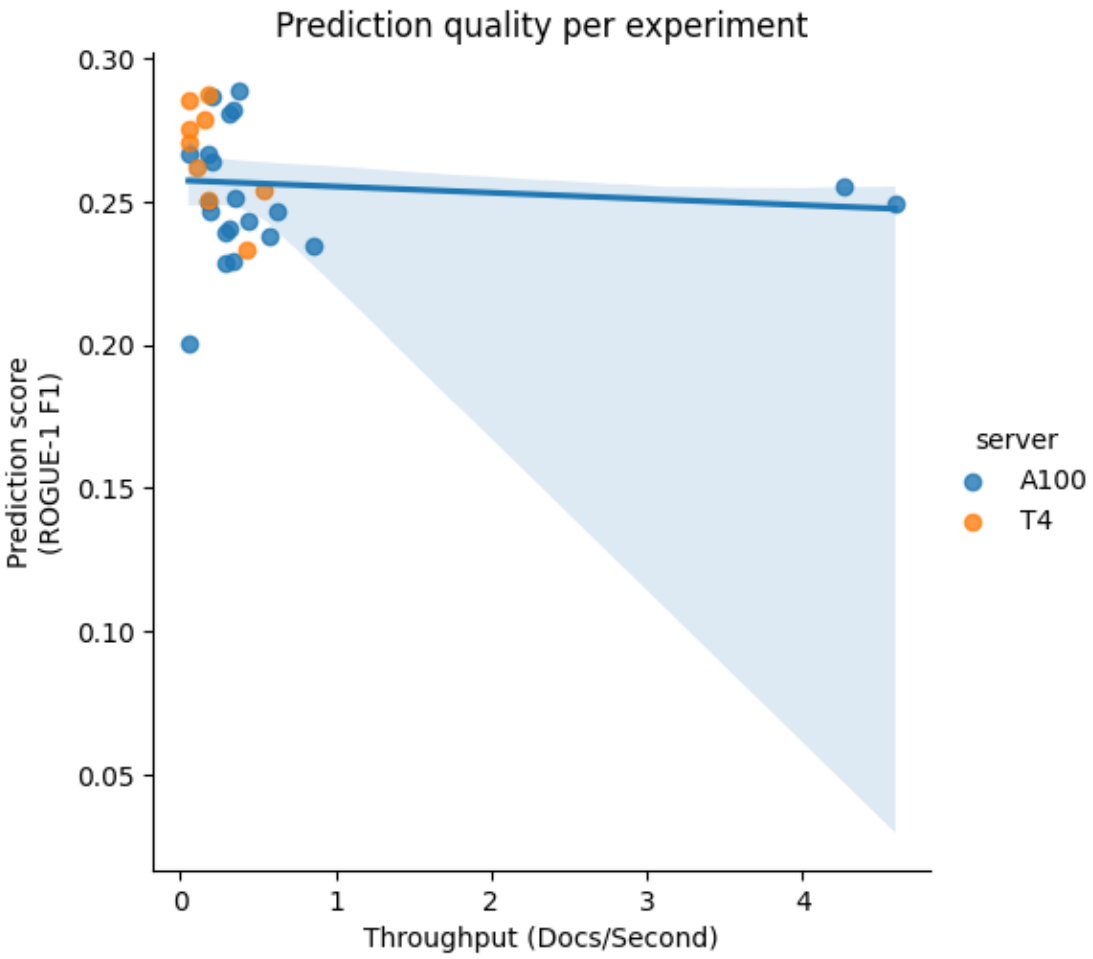
As seen above, the entire experiment runtime, accounting for tokenization, transfer of data to GPU, generation, and decoding, is significantly more time-intensive on the T4, particularly after

quantization. 4-bit quantization dramatically speeds up the experiment compared to 8-bit on each GPU server, but 4-bit and 8-bit quantization techniques reduced latency compared to no optimizations whatsoever, and were mainly shown since it wasn't possible to run the T4 experiment without these techniques.

It should be noted that vLLM speedups appear to scale to larger datasets. For instance when testing the inference speed from vLLM on A100 with a larger dataset that randomly sampled 1000 documents, the throughput actually improved:



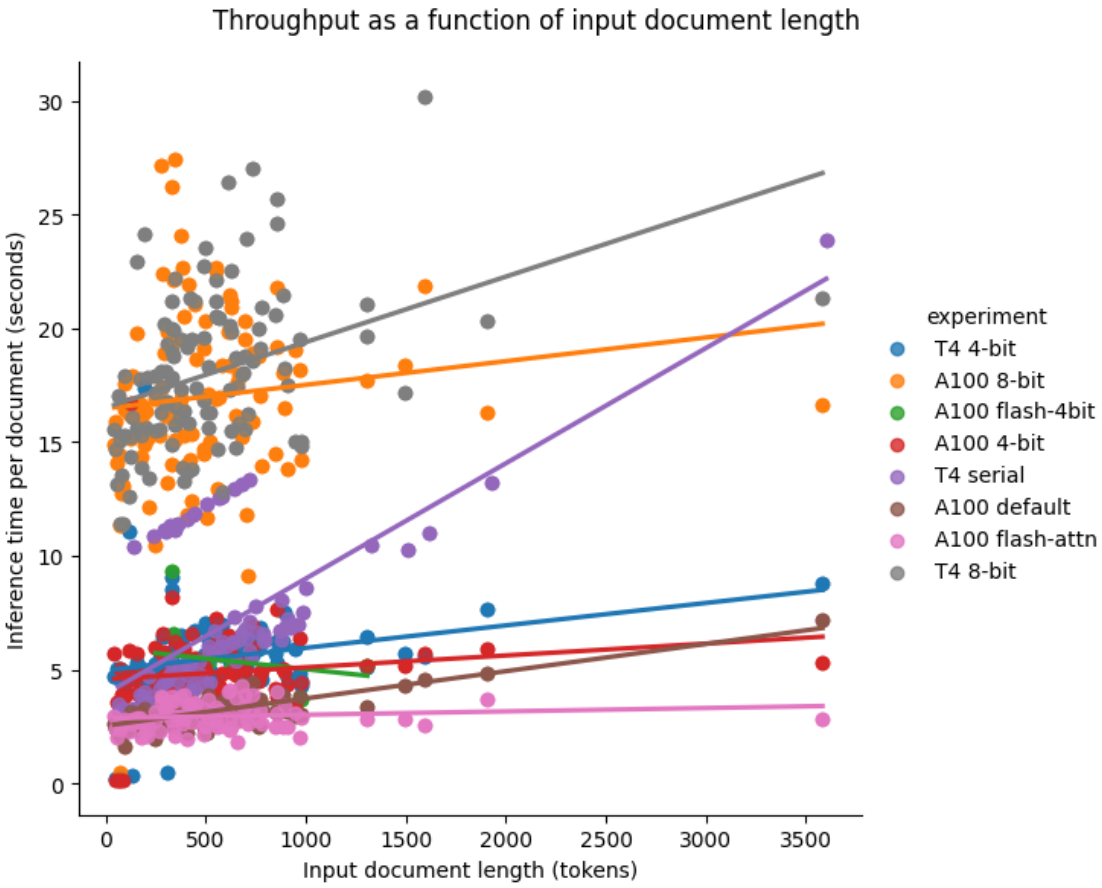
Since quantization and other optimizations discussed above can also impact quality, we look at the quality of the data these models produced next, as seen in the chart below:

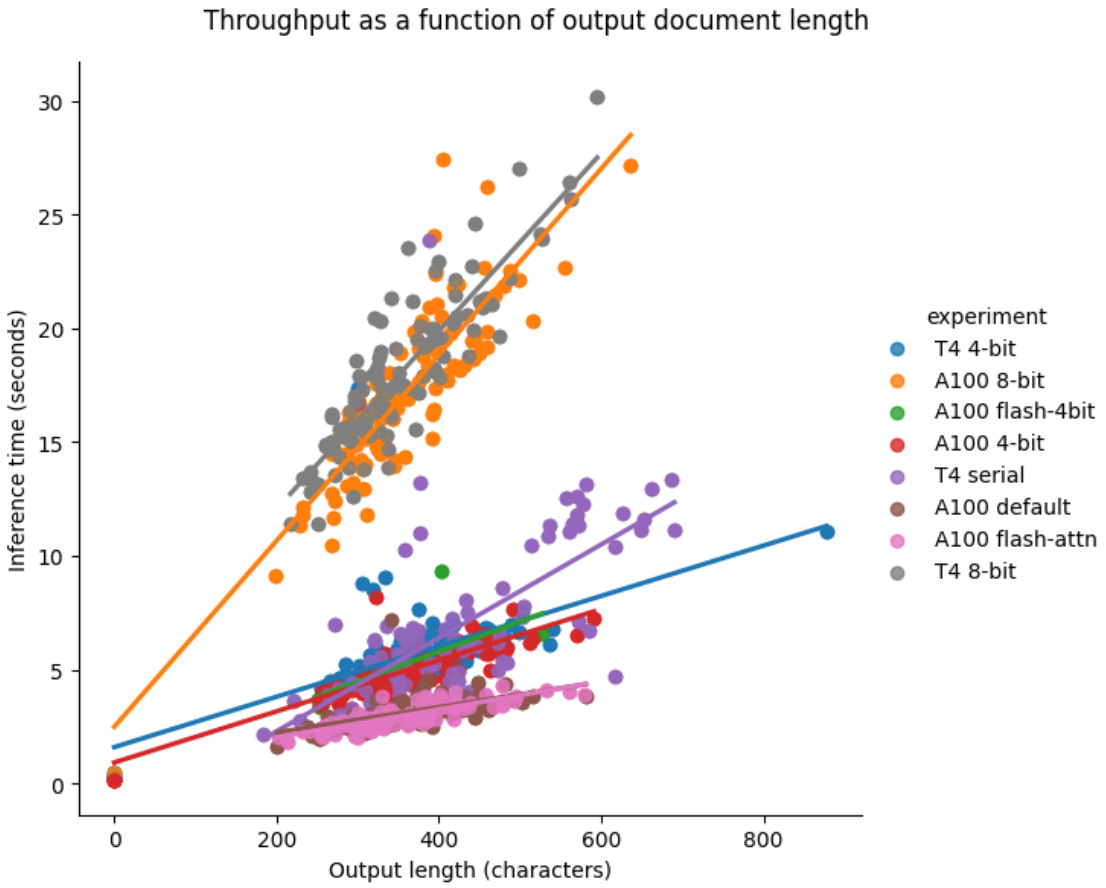


Prediction quality was measured via by computing the ROGUE-1 F1 score for each example. The score is calculated by measuring the overlap of unigrams between the summaries generated by the LLM with each corresponding ground truth summary from the Wikilingua dataset.

There appears to be a slight negative correlation between throughput and prediction quality, but the confidence is very low as throughput increases, due to outliers: the A100 vLLM experiments. There is not enough data, nor are there enough optimization strategies to say with confidence that optimizations necessarily entail a decrease in prediction quality, but certain optimization strategies such as 4-bit and 8-bit quantization of model weights are known to decrease prediction quality by reducing the precision of floating point operations.

Throughput is more influenced by sequence length, as is shown in the following charts:


Throughput as a function of input document length
Inference time per document (seconds)
Input document length (tokens)
0
5
10
15
20
25
30
0
500
1000
1500
2000
2500
3000
3500
experiment
T4 4-bit
A100 8-bit
A100 flash-4bit
A100 4-bit
T4 serial
A100 default
A100 flash-attn
T4 8-bit



Generally, there was a positive correlation between inference speed and input document length as well as speed and generated text length, the degree of which depended on the optimization strategy. This is intuitive, since the longer a given input document (i.e. a Wikilingua article) is, the greater the context that the LLM needs to process, which increases computational load on the server. For Transformer LLMs like Llama2, the attention mechanism has to compute and keep track of attention weights over a larger set of tokens due to the increased length of the prompt document.

For LLM output generation, computational load is also increased by the length of the output due to the attention mechanism as well, which requires keeping a larger set of weights in memory over the larger text. Also, since LLMs generate outputs in a sequence, token by token, the larger the total generated text is, the longer the model needs to spend generating it.

1.3 Failed Experiments

Memory issues plagued the lower-memory T4 instance, which couldn't generate summaries on the Wikilingua documents without optimizations. Quantization (e.g. 4-bit and 8-bit) significantly helped to prevent out of memory errors on the T4 instance type. Memory issues were less of a concern on the larger-memory A100 until experiments with batching. There were many cases where the optimization techniques were insufficient to prevent an out of memory error on the GPU device,

however. And, in addition to memory issues, there were many tests that failed due to hardware or software version incompatibility with the optimization libraries or techniques.

Experiments that were unsuccessful due to high runtimes, skipped prompts, or memory issues on either the T4, the A100 or both include:

- DeepSpeed Optimizations
- DeepSpeed ZeRO-Inference
- IBM Foundation Model Stack
- ONNX Inference

The following table shows a variety of different errors from various experiments run on different document counts:

Optimization	GPU Type	Batch Size	Error	Documents
No optimizations	`T4`	1	`OutOfMemoryError`	10
Flash Attention	`T4`	1	`RuntimeError`	10
No optimizations	`T4`	2	`OutOfMemoryError`	10
vLLM (no quantization)	`T4`	dynamic	>90% prompts skipped	100
DeepSpeed	`T4`	dynamic	`OutOfMemoryError`	100
No optimizations	`A100`	4	`OutOfMemoryError`	10
Flash Attention	`A100`	10	`OutOfMemoryError`	100
DeepSpeed	`A100`	dynamic	Garbage outputs	100

The Flash Attention experiments were only run on the A100 GPU since it's incompatible with the T4 machine's older GPU type, since Flash Attention requires Ampere GPUs or newer. Batching, meanwhile resulted in memory errors for all batch sizes (greater than 1) on the T4, and additionally occurred for all batches after 10 documents had been processed on the A100. This suggests that additional memory optimizations - likely of the sort included out-of-the-box in the vLLM library - are required to clear the memory. The above batching experiments used a naive approach of concatenating tensors of prompts based on a fixed batch size, whereas vLLM generates the batch sizes dynamically based on available memory and the total sequence lengths.

DeepSpeed experiments failed with out of memory errors or CUDA compute capabilities version incompatibility issues on the T4 machines in all cases. The DeepSpeed tests showed promising throughput metrics on the A100, but were generating garbage outputs due to a deadlock error which caused the KV cache to reset mid-experiment upon each execution.

I also tested some of the experiments on a V100 code, and noted that Flash Attention and vLLM with AutoAWQ quantization were incompatible with the GPU type or capability version.

1.4 Conclusion

Using vLLM to optimize LLM inference results in dramatic improvements in latency and allow large models to run on smaller hardware at high speeds. For lower-memory GPU instances, such as the T4, quantization is critical to avoiding out of memory errors. In general, there are many compatibility issues in optimization libraries due to different hardware setups and library versions.

It should be noted that runtime and quality fluctuate slightly between each run due to differences in the server environment and non-determinism in the model generation step, respectively. An area of improvement for these experiments is to improve the consistency of the results through repeated executions, running the experiments on larger datasets, and running repeated sampling to calculate more robust benchmarks.

1.5 Appendix

1.5.1 Code

All scripts used to run these experiments can be found in the following GitHub repository:

https://github.com/lcary/ai-summarization-benchmarks/

1.5.2 Cost

Note that using the A100 requires a Colab Pro subscription, and eats up about 2 credits per hour. Additionally, after a certain amount of time using the free T4, Colab pre-empted experiment sessions in faver of paying customers. The most expensive part of this experiment was collecting baseline benchmarks on non-optimized models, since they take the longest to run.

All in all, this experiment costed `$20.00` to run (200 credits at `$10.00` per 100 credits).

1.6 Other Issues

1.6.1 Flash Attention Compatibility

Note that Flash Attention experiments were only run on the A100 GPU since it's incompatible with the T4 machine, due to the T4 having an unsupported, older GPU type (error: `RuntimeError: FlashAttention only supports Ampere GPUs or newer.`). Also, note that the `4-bit` quantization experiments use 16-bit floating points for the computational type via `bnb_4bit_compute_dtype`, while for `8-bit` quantization, the computational type is set to the default.

Note that Flash Attention is also incompatible on V100 GPU servers.

1.6.2 vLLM Memory and Compatibility Issues

The `vllm` experiments run without AutoAWQ quantization consumes the entirety of the T4 machine's GPU memory when optimizing KV cache and during CUDA graph creation, resulting in out of memory issues unless we decrease the maximum model sequence length (prompt and output combined) is decreased, which results in the majority of documents being skipped. This is fixed by AWQ quantization at the expense of throughput. More info on this issue can be found in the `scripts/run_llama_7b_vllm.py` script docstring.

Note vLLM with AWQ is incompatible on V100 GPU servers.