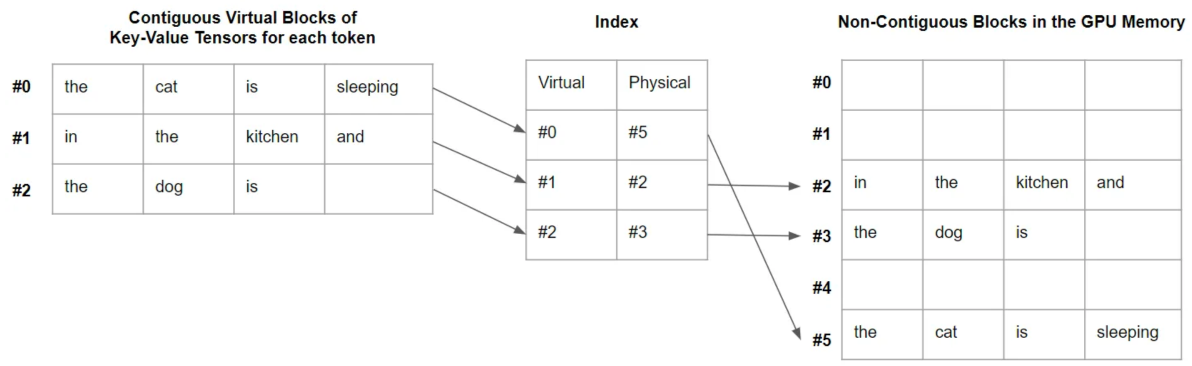
# vLLM: PagedAttention for 24x Faster LLM Inference

A more efficient way to compute Transformer's attention during inference

Benjamin Marie · Follow
Published in Towards Data Science
6 min read · 2 days ago

Listen Share



PagedAttention for a prompt "the cat is sleeping in the kitchen and the dog is". Key-Value pairs of tensors for attention computation are stored in virtual contiguous blocks mapped to non-contiguous blocks in the GPU memory. — Image by the author

Almost all the large language models (LLM) rely on the Transformer neural architecture. While this architecture is praised for its efficiency, it has some well-known computational bottlenecks.

During decoding, one of these bottlenecks is in the computation of the attention with pairs of key-value tensors for each token of the input. All these tensors must be stored in memory.

*Note: I won't explain in this article what is the role of these key-value pairs. It's one of the most complicated and interesting aspects of the Transformer architecture. If you don't know about it, I strongly recommend reading The Illustrated Transformer by Jay Alammar.*

As LLM accepts longer and longer inputs, e.g., the LLM Claude accepts 100k token-long inputs, the memory consumed by these tensors can become very large.

Naively storing all these tensors in memory leads to memory over-reservation and fragmentation. This fragmentation can make memory access very inefficient, especially for long sequences of tokens. As for over-reservation, the system does it to make sure it has allocated enough memory for the tensors, even if it doesn't consume all of it.

To alleviate these issues, UC Berkeley proposes PagedAttention.

PagedAttention is implemented in vLLM (Apache 2.0 license) which is deployed by LMSYS, an organization for open research founded by students and faculty from UC Berkeley with the help of UCSD and CMU.

In this article, I explain what PagedAttention is and why it significantly speeds up decoding. I show towards the end of the article how to get started with vLLM to exploit PagedAttention for inference and serving LLMs on your computer.

## PagedAttention for Transformer

Kwon et al. (2023) propose PagedAttention.

The goal is to store key-value tensors more efficiently in the non-contiguous spaces of the GPU VRAM.

In short, the idea behind PagedAttention is to create contiguous virtual blocks mapped to physical blocks in the GPU memory.

Each block is designed to store key-value pairs' tensors for a predefined number of tokens. All the blocks are virtually contiguous and mapped to physical non-contiguous blocks, allocated on demand during inference, in the fragmented GPU memory. A simple index table is also created in memory to associate virtual with physical blocks.

The kernel of PagedAttention fetches as needed these blocks. This is efficient because the system fetches smaller numbers of key-value tensors due to the limited size of the blocks.

Let's take the following prompt for illustration:

*the cat is sleeping in the kitchen and the dog is*

We have key-value tensors for each token. With PageAttention, we can (arbitrarily) set the block size at 4. Each block contains 4 key-value tensors, except the last one which contains only 3 key-value tensors. The blocks are virtually contiguous but are not necessarily contiguous in the GPU memory, as illustrated by the figure in the introduction of this article.

For the computation of attention, for each query token, the system fetches the block one by one, as illustrated below.

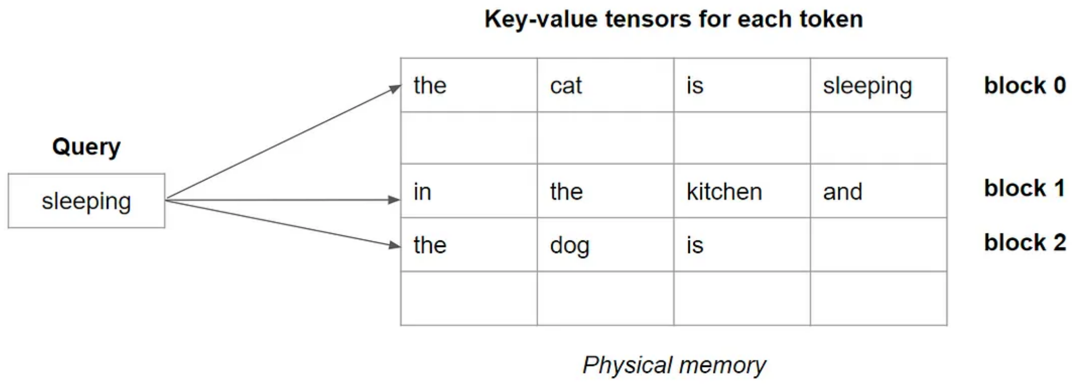


Illustration of virtual blocks containing key-value tensors for up to 4 tokens — Image by the author

By fetching key-value tensors by blocks, instead of the entire sequence of tensors, the computation of attention is much faster.

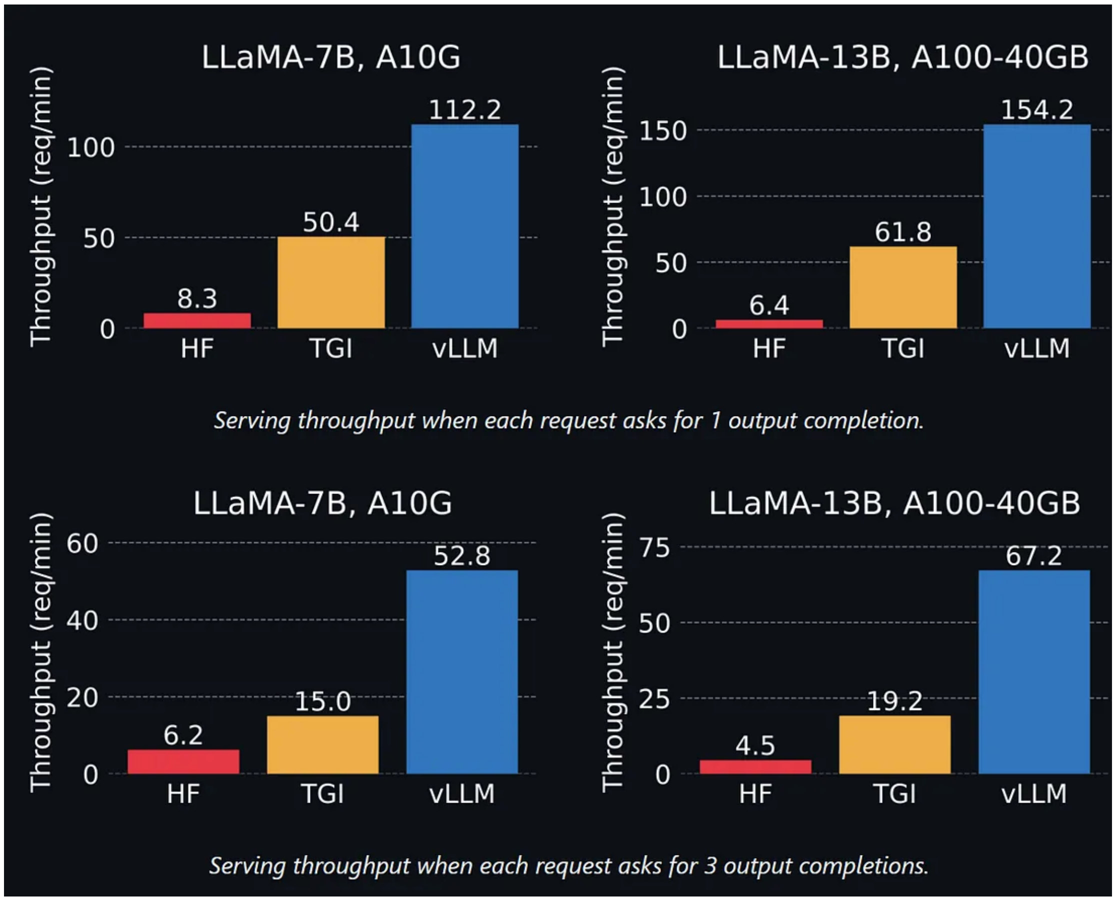
## Parallel Sampling for Inference

Another advantage of PagedAttention is that the virtual blocks can be shared when sampling during inference. All the sequences generated in parallel via sampling or beam search can use the same virtual blocks, avoiding duplicates.

In their experiments, LMSYS observed a 55% reduction in memory usage for beam search decoding.

## PagedAttention performance reported by LMSYS

Before trying it by ourselves, let’s have a look at the performance reported by the authors (UC Berkely/LMSYS) when using PagedAttention implemented in vLLM compared to the text generation inference library developed by Hugging Face.



Performance of LLaMa models for output completion tasks for the original Hugging Face library (HF), text generation inference library (TGI), and vLLM with PagedAttention (vLLM) — Plots by UC Berkeley and LMSYS

vLLM looks much faster according to these results, especially in the case of multiple output completions. The difference between TGI and vLLM increases with bigger models. This is expected since bigger models require more memory and are thus more impacted by memory fragmentation.

Overall, vLLM is up to 24x faster than the Hugging Face Transformers library.

*Note: Actually, I'm also impressed by the improvement from HF to TGI. I didn't cover TGI yet on my blog but I'll probably write a guide about it. TGI is used in production at Hugging Face. While it seems much slower than vLLM, TGI has other advantages such as the support for many more models and features.*

## How to set up vLLM on your computer

*Note: vLLM doesn't support CUDA 12 yet. Use a lower version, such as 11.8.*

In this section, I will only go through the basics of how to set up and run vLLM on your computer. For more advanced usage, you can have a look at the vLLM documentation.

As I write this article, vLLM only supports a few types of models:

- GPT-2
- GPT-NeoX and Pythia based

- LLaMa based
- OPT based

You can add the support of other models by following these instructions.

In the code below, I use Dolly V2 (MIT license). It is a chat model based on Pythia and trained by DataBricks.

I chose the smallest version with 3 billion parameters. It can run a consumer GPU with 24 GB of VRAM, e.g., an nVidia RTX 3080/3090.

The most straightforward way to install vLLM is with pip:

```
pip install vllm
```

*Note: This should take up to 10 minutes.*

But in my case, on both my computer and Google Colab, pip failed to install the vllm library. The authors of vLLM confirm that there is a problem with some nvcc versions and environments. Nonetheless, for most configurations, pip should install vLLM without any problem.

If you are in the same situation as me, the workaround is simply to use a Docker image. This one worked for me:

```
docker run --gpus all -it --rm --shm-size=8g nvcr.io/nvidia/pytorch:22.12-py3
```

*Note: Once in the docker, the authors recommend removing Pytorch before installing vLLM: pip uninstall torch. Then, "pip install vllm" should work.*

Then, we can start writing Python.

We first need to import vllm, and then we load the model with vllm. The inference is triggered by llm.generate().

```
from vllm import LLM

prompts = ["Tell me about gravity"] #You can put several prompts in this list
llm = LLM(model="databricks/dolly-v2-3b")  # Load the model
outputs = llm.generate(prompts)  # Trigger inference
```

You can also use vLLM for serving LLMs. It works similarly to TGI. It's also much more simple than running the NVIDIA Triton inference server that I described in a previous article.

You first need to start the server:

```
python -m vllm.entrypoints.openai.api_server --model databricks/dolly-v2-3b
```

*Note: The server will listen on port 8000. Make sure it is available or change it in the vLLM configuration file.*

Then, you can query the server with prompts as follows:

```
curl http://localhost:8000/v1/completions \
    -H "Content-Type: application/json" \
    -d '{
        "model": "databricks/dolly-v2-3b",
        "prompt": "Tell me about gravity",
        "max_tokens": 200
    }'
```

And that's it! You have a very efficient LLM server running on your computer.

## Conclusion

PagedAttention significantly speeds up inference. It's another step toward more affordable AI with LLM.

In further experiments, I confirmed that vLLM is especially efficient with batches of prompts. To fully take advantage of vLLM, consider optimizing your batching strategy for inference.

While beam search with large beams may have been prohibitive with standard attention computation, beam search with PagedAttention is faster and more memory efficient.

One of my next experiments will be to combine PagedAttention with QLoRa to reduce memory usage. It should be straightforward. It would make running LLMs on consumer hardware even more efficient.

. . .

*If you like this article and would be interested to read the next ones, the best way to support my work is to become a Medium member using this link:*

**Join Medium with my referral link - Benjamin Marie**

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story…

medium.com

*If you are already a member and want to support this work, just follow me on Medium.*

Data Science | Programming | Machine Learning | Technology | Artificial Intelligence