`vllm-triton-backend`: How To Get State-of-the-art Attention Performance on NVIDIA and AMD With Just Triton

Dr. Burkhard Ringlein
IBM Research
AI Platform Group, Zurich

2025-10-23, PyTorch Conference, San Francisco

IBM

The year in open-source LLM inference so far …

(just no guarantee for completeness)

DeepSeek-R1 Mixtral Bamba Llama 4 gpt4-oss Qwen3-next Granite4

Models

Inference Technologies

chunked prefill, prefix caching → **vLLM V1**

State-space kernels

Attention sinks

FP8 / FP4

Linear attention

Mixture of Experts + Attention + SSM → “Hybrid KV-cache”

…How to keep up?

Can we make portable kernels?

+ vLLM

→ vLLMs Triton attention backend is here to help you!

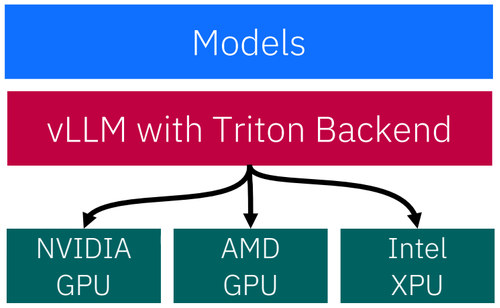
Hardware

H100 MI300 Blackwell CNDA4 / MI350X Intel Gaudi 3

Agenda: How to achieve performance portability for LLM kernels

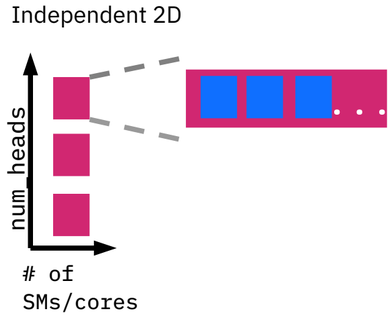
(Or, how to catch the train...all the times!)

1. Some background: Triton

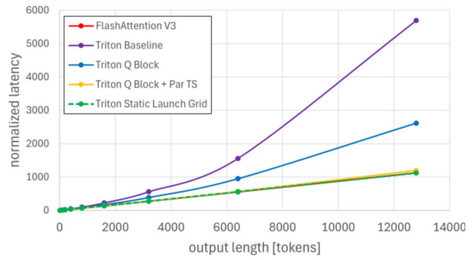


2. Triton-only Attention Backend in vLLM

3. Insights in Kernel Development & Performance Portability



4. Results

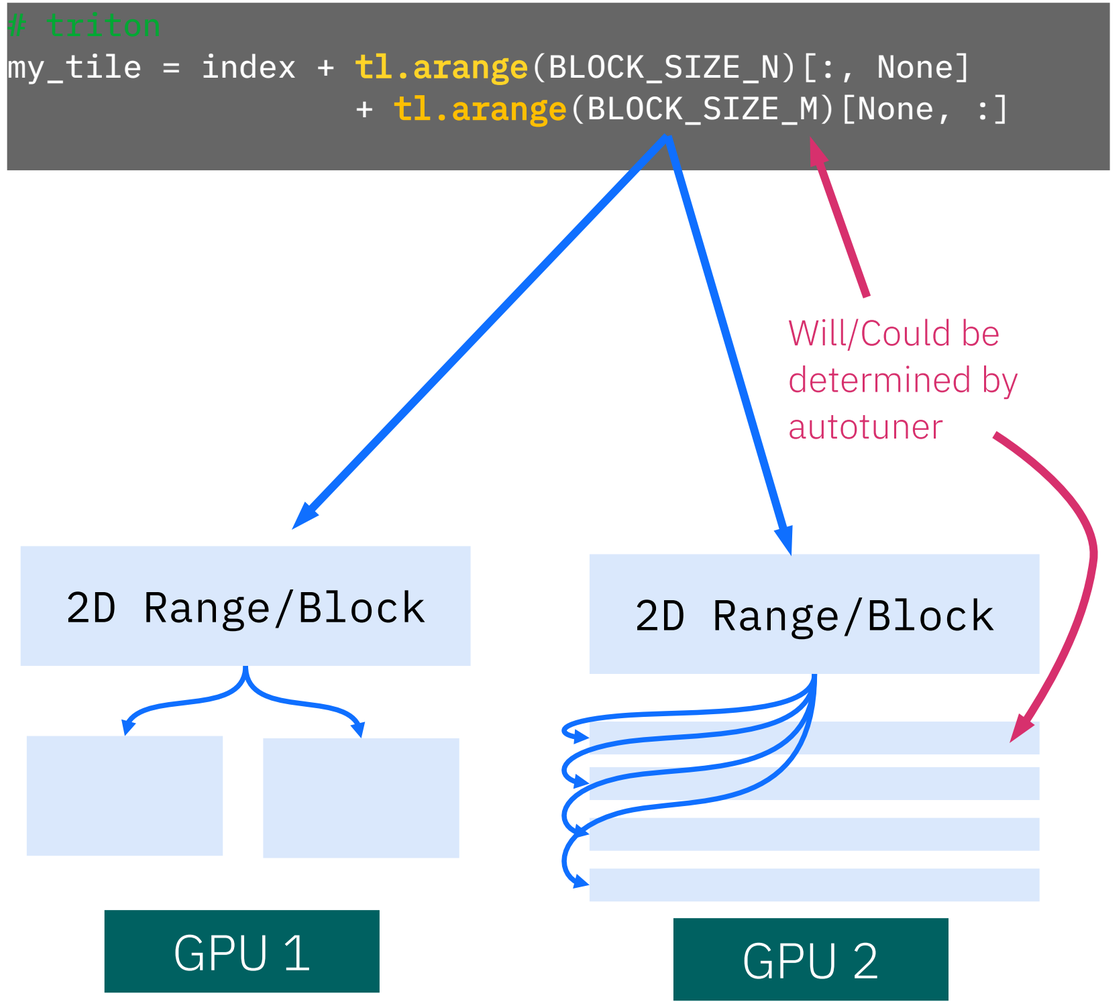


Conclusion?

→ Goal: (1) Sketch the way to performance portability using the 100% open-source solutions
(2) ...getting you excited about the Triton Attention Backend in vLLM! 😉

Performance Portability: What is it and how does Triton help?

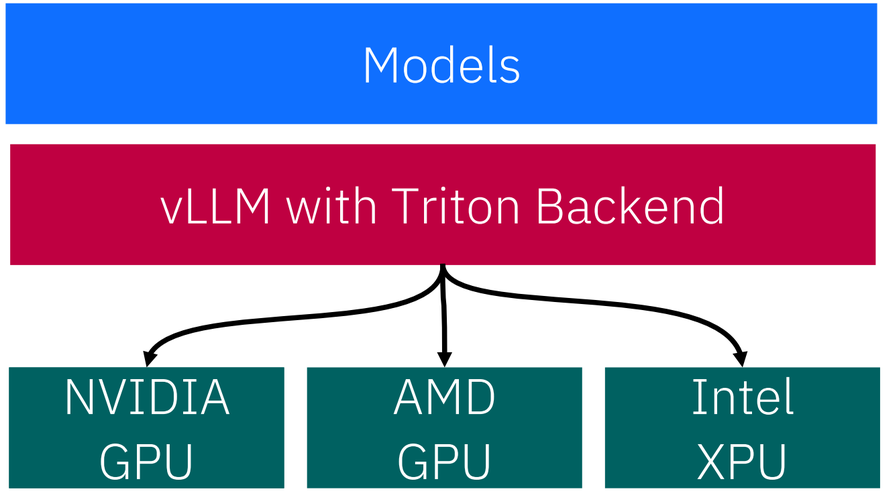
- “Performance portability refers to the ability of computer programs and applications to operate effectively across different platforms.” (wikipedia.org)
- In a Nutshell: Tritons tiled programming model & abstraction level:
  - Allows to express complicated optimizations *(we’ll come to that…)*
  - While staying hardware agnostic
- Hyperparameters (called “configurations”) allow simple but effective adaptations for different hardware (maybe also with **autotuning**)



Introducing: vllm-triton-backend

- We (IBM Research & RedHat) built a production-ready vLLM attention backend written entirely in Triton
  - Contains multiple kernels for prefill & decoding (paged attention)
  - Packaged together with heuristics to adapt for different scenarios
- (a Backend in vLLM encapsulates different attention-implementations, usually wrappers for external libraries)
- → First (and only) triton-only backend in vLLM
- Runs on NVIDIA, AMD & Intel
- Delivering state-of-the-art performance
  - Is the default for AMD-based deployments
  - 100.7% end-to-end latency vs FlashAttention3 using vLLM V1 on H100
  - (PRs of latest changes still pending)
- In the meantime: *a lot of help from the community!*

(Thanks a lot!)



Installation:

```
# integrated in vllm
pip install vllm
        (no further dependencies)
```

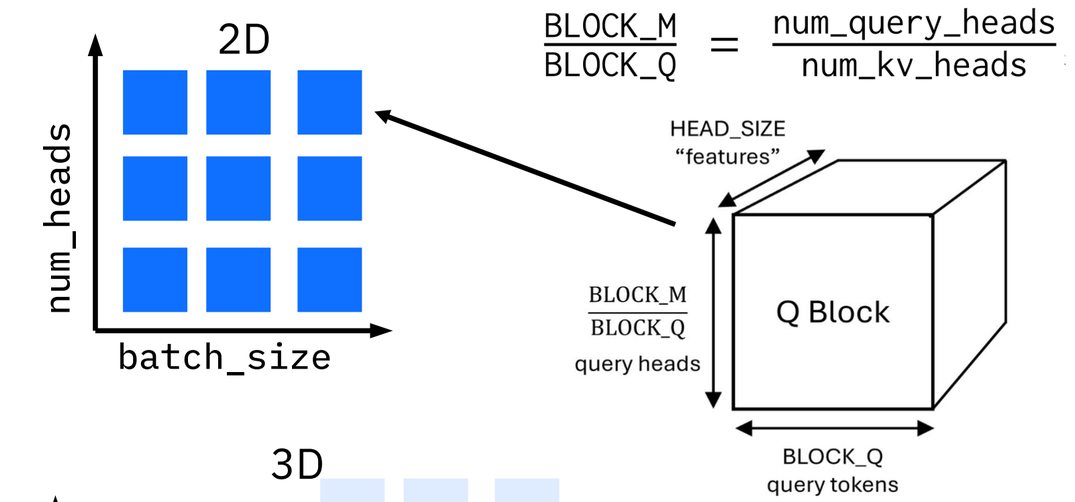
Run:

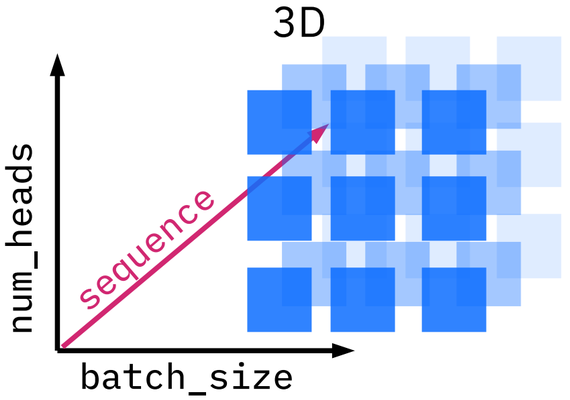
```
export VLLM_ATTENTION_BACKEND=TRITON_ATTN_VLLM_V1
# only in cases where it is not the default

vllm serve meta-llama/Llama-3.1-8B-Instruct
```

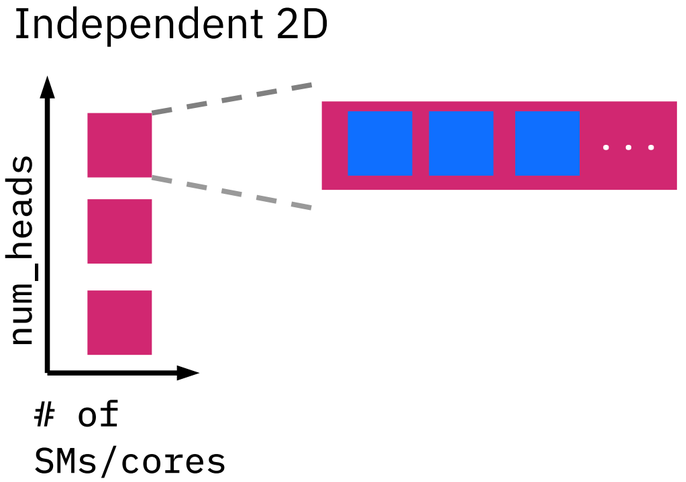
Key Insights: Kernel Development

- Optimize for GQA and maximize tile sizes for `tl.dot`:
  - Load all query heads for one KV head
  - Fill tiles with multiple query tokens
- For decode-heavy batches: add attional parallelization → “Parallel Tiled Softmax” / 3D
- Minimize launch overhead of Triton kernels
  - Can be up to 200µs → dominates for small/medium requests
  - CUDA/HIP graphs also for attention layer
    - Support added during summer to vLLM V1
    - Requires launch grids of kernels that are independent of the batch size
    - Requires persistent kernels → add outer loop
- Derive heuristics for kernel configurations (simple decision-trees) from extensive micro-benchmarking



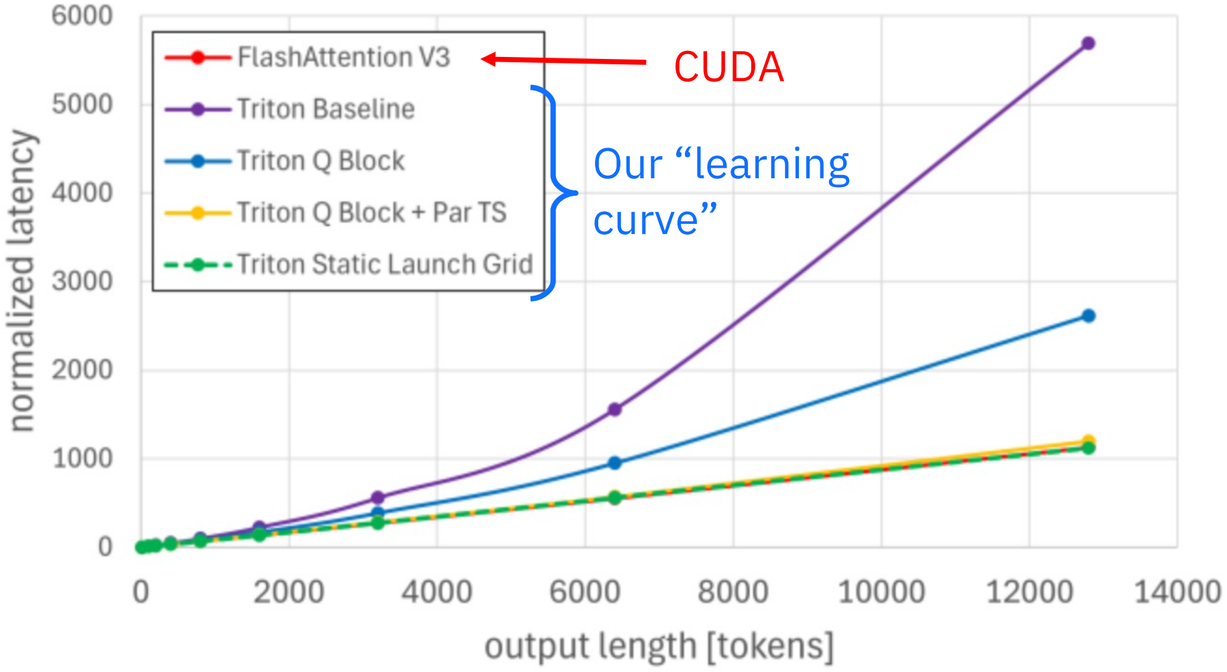


(different heuristics for size of 3rd dimension)

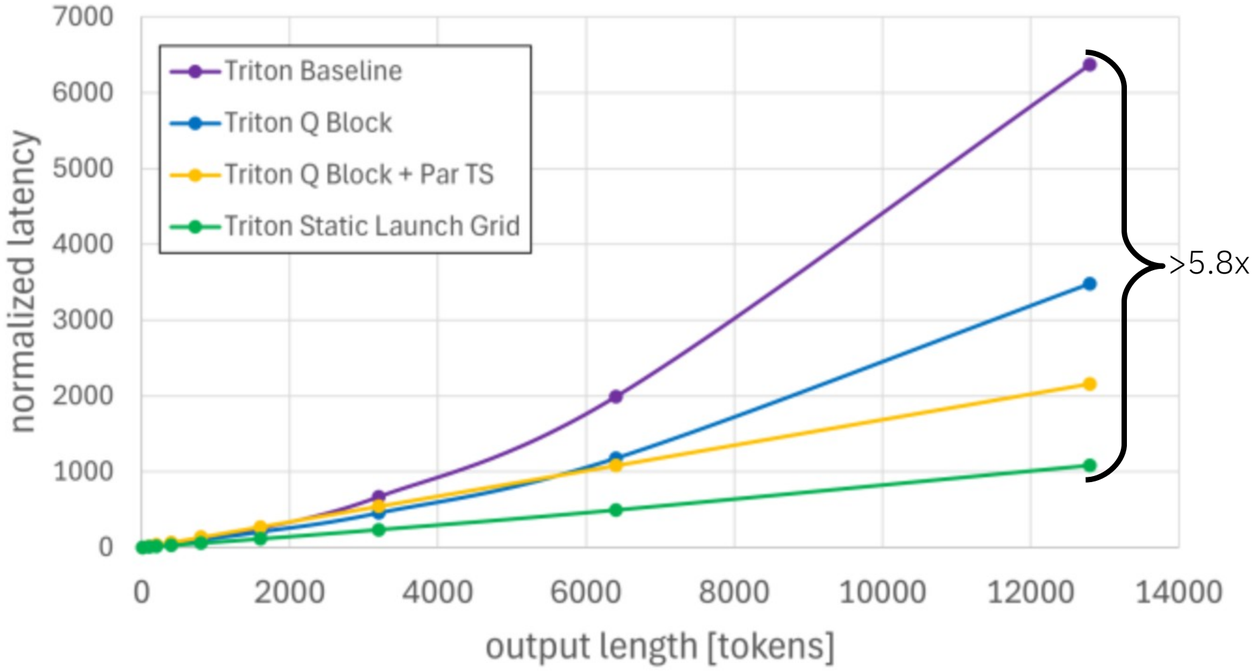


Result: State-of-the-art-performance → on NVIDIA & AMD GPUs

End-to-end experiment: vLLM **latency** benchmark results for **Llama-3.1-8B** with **batch size 1** and **input length of 500 tokens** with random data. The Triton Static Launch Grid experiment is using full CUDA/HIP graphs, all other piece-wise graphs.



(a) H100



(b) MI300

- On NVIDIA H100: Our kernel achieves 100.7% of the famous "FlashAttention3" library (for long decode requests)
  - ~800 LoC Triton vs. ~70'000 LoC CUDA
- On AMD MI300: We are the state-of-the-art 😉 and achieved a speedup of 5.8x in the last 6 month
  - Using the same (!) triton kernels → achieving performance portability

Conclusion?

- We developed a triton-only feature-complete attention backend for vLLM
  - that achieves state-of-the-art performance on AMD & NVIDIA (and also runs on Intel)
  - with a single-source kernel
- Community-maintained kernels in an open-source high-level language help to “catch the train” in LLM kernel development by enabling performance portability

```
pip install vllm
# no further
#  dependencies
```

- Big thanks to the team: Jan van Lunteren, Thomas Parnell, Chih-Chieh Yang, Sara Kokkila Schumacher, Sage Moore, Lukas Wilkinson, Luka Govedič

Want to know more?

- **Upcoming paper**: → With all details and further lessons learned

→ Paper: *“The Anatomy of a Triton Attention Kernel”*

- **vLLM office hour** about vllm-triton-backend: 20th of November
- vLLM meetup in Zurich: 6th of November
- github.com/foundation-model-stack/vllm-triton-backend
  - Including all Micro-benchmarks & developer setup
  - Including optimizations that are not yet in vLLM
- ***… Contribute!*** (Play around…and we/vLLM welcomes PRs)

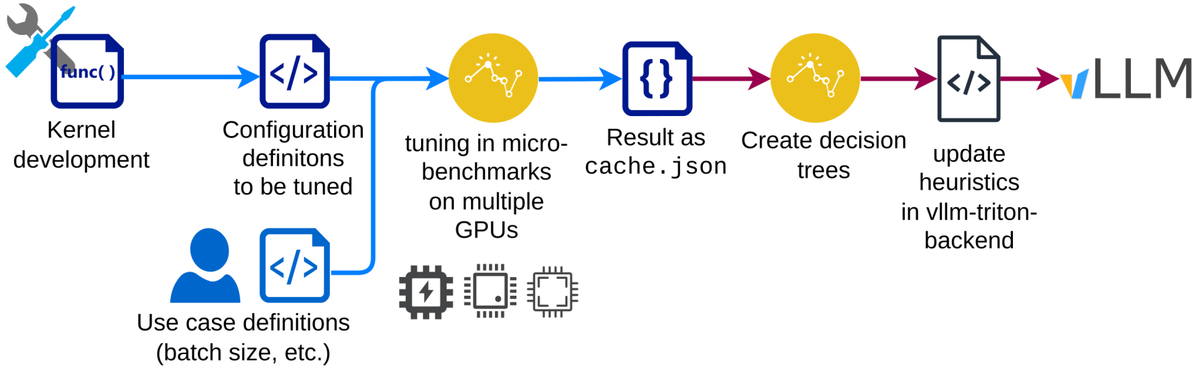
…looking forward to all your questions!

Dr. Burkhard Ringlein

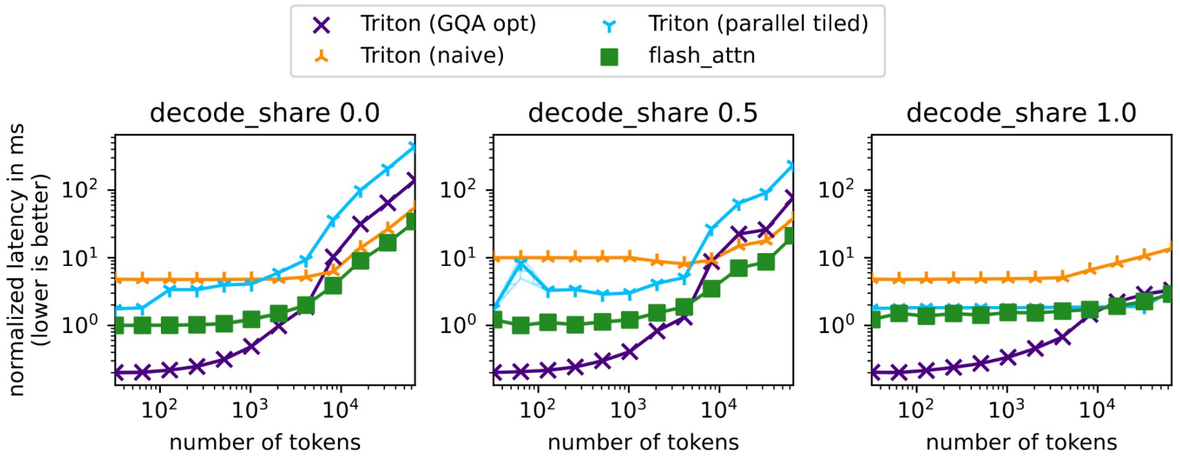
ngl@zurich.ibm.com

burkhard-ringlein

Appendix



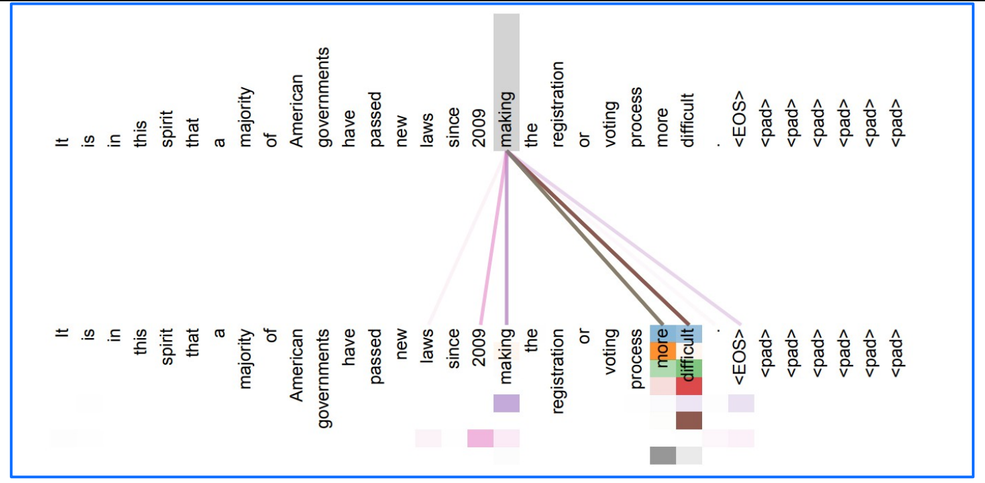

```python
if torch.version.hip:
    TILE_SIZE_PREFILL = 64
    NUM_STAGES_PREFILL = 1
    NUM_WARPS_DECODE_3D = 4
else:  # cuda platform
    TILE_SIZE_PREFILL = 16
    NUM_STAGES_PREFILL = 4
    NUM_WARPS_DECODE_3D = 2
```



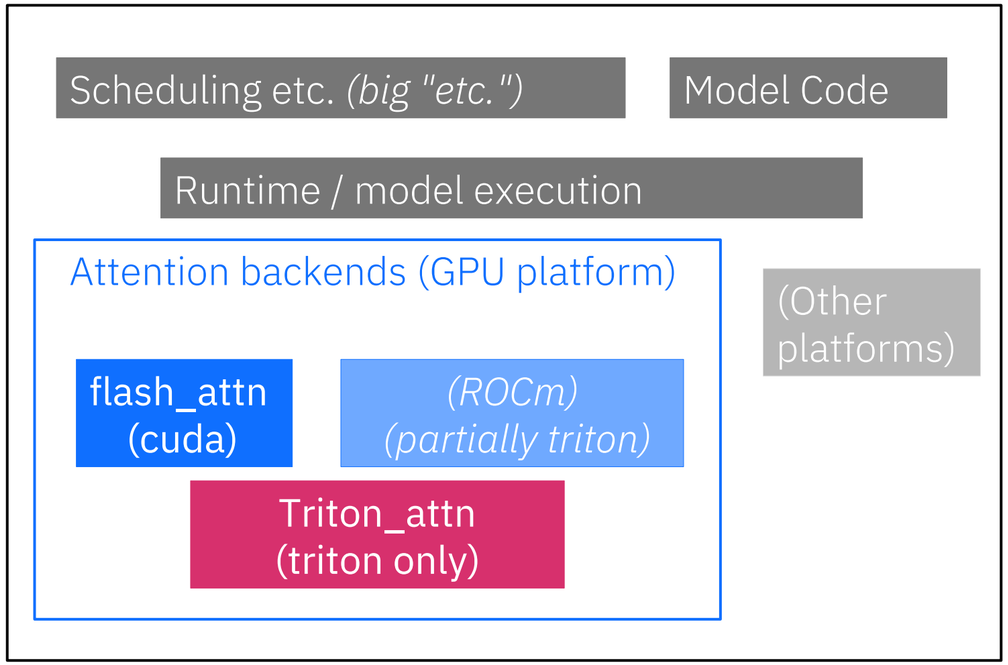
Key-Insights: Autotuning & Heuristics

- Tuning block and tile sizes for different models & GPUs can increase performance
- However:
  - Tuning still has a lot of overhead (even if done ahead-of-time)
  - Looking up best config before every kernel call adds latency
  - Ahead-of-time tuning can barley cover all possible use cases
- → Train heuristics based on extensive micro-benchmarks
  - Simplifies kernel launch
  - Generalizes well to unseen workloads
  - (simple if-else trees for now)
  - Compatible with CUDA graphs
  - Use extensive Micro-benchmarks
- Future work: Implement ML-based decision Trees

A. Vasawani et al., Attention Is All You Need, https://arxiv.org/abs/1706.03762



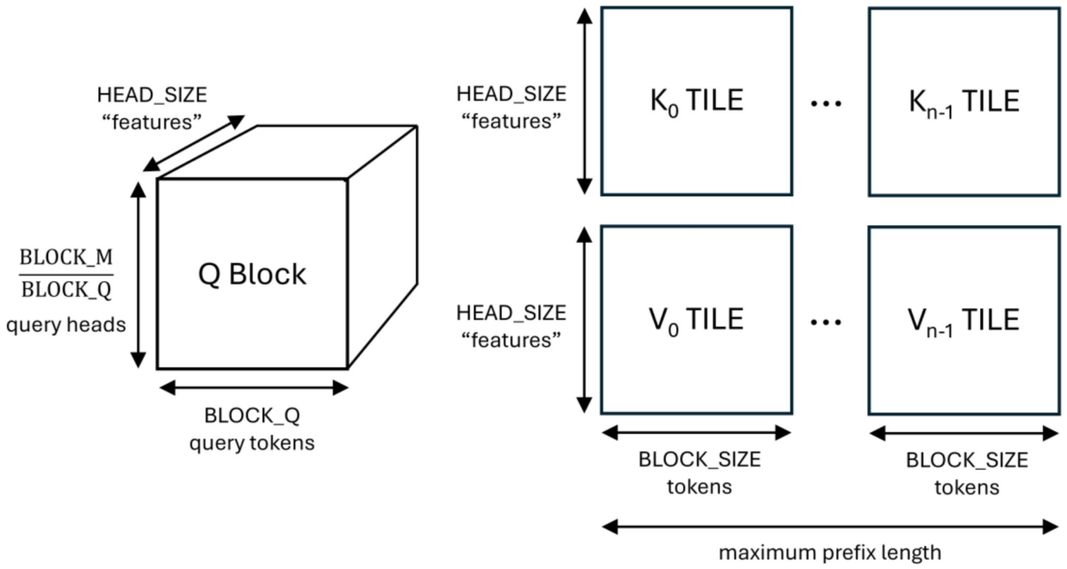
vLLM V1



vLLM and its Backends

- Execution of LLMs are dominated by the attention kernels
  - Attention as in “Attention Is All You Need”
  - → vLLM encapsulates the different attention implementations/libraries as Backends
  - (Different backends available for vLLM V0 and vLLM V1)
- A backend offers different APIs for metadata preparation and the “`forward`” operation
  - Also in the process to introduce different backends for SSMs, MoE, etc.
  - Backends are sometimes Model or GPU specific… *(talking about catching the train again…)*
  - Backends usually introduce external dependencies
- How to ensure performance portability?
- No triton-only backend exists → until recently…

“Q-Blocks”

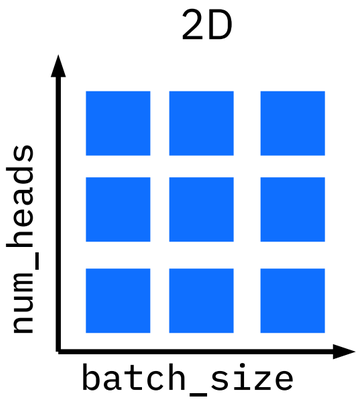


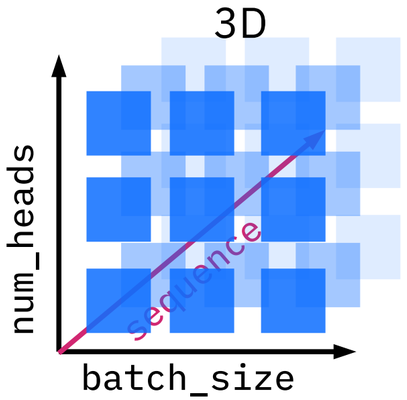
$$\frac{\texttt{BLOCK\_M}}{\texttt{BLOCK\_Q}} = \frac{\texttt{num\_query\_heads}}{\texttt{num\_kv\_heads}}$$

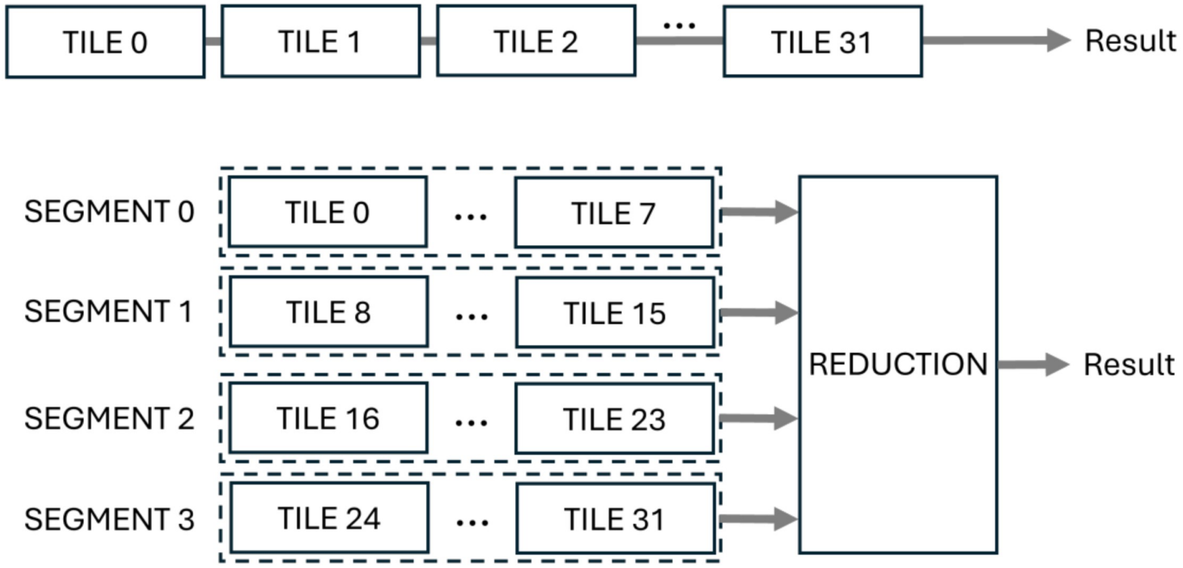
- Primary optimization for GQA (done in May)
  - Ensures that each KV-head is only loaded once
- But tuning also allows to optimally select multiple query tokens to make the tile sizes for tl.dot optimal
  - e.g. Llama 3: 32 query heads, 8 kv heads
  - 4 query tokens in parallel:
  - → H100: BLOCK_M = 4, BLOCK_Q = 4
  - → MI300: BLOCK_M = 64, BLOCK_Q = 16 (depending on the kernel version)

```
S += scale * tl.dot(Q, K)
```

- tl.dot: [BLOCK_M, PAGE_SIZE]







“3D kernel” or Parallel Tiled Softmax

- So far, we parallelize along **`num_heads`** and **`batch_size`** → does not utilize SMs fully
- Long-context requests requires additional parallelization
- → New kernel that partitioning into many tiles per single request
  - reduce partial results later (online softmax)
  - Despite twice launch overhead, it leads to significant gains for long requests
- Tradeoff: additional launch overhead vs more parallelism:
  - Heuristics for number of decodes
  - Full cuda graphs required