Efficient and Customizable Kernel Generation for LLM Inference Serving

Zihao Ye

Apr 9th, 2025
Guest Lecture at @ CMU Machine Learning Systems
University of Washington & NVIDIA

Major Operators in Large Language Models

- **Attention**
  - KV-Cache storage heterogeneity/sparsity
  - Variable Sequence Length
  - Attention Variant
  - Shared-prefix in KV-Cache structure
- **GEMM**
  - Customized Epilogue
  - Performance critical (close to speed of light)
  - Dynamism: Group GEMM
- **Collective Communication**
  - High-latency, can overlap with computation
- **Logits Post-Processing**
  - Top-K/Top-P/Min-P/... sampling

FlashInfer: non-intrusive, high-performance kernel collectives for LLM Serving Engines.

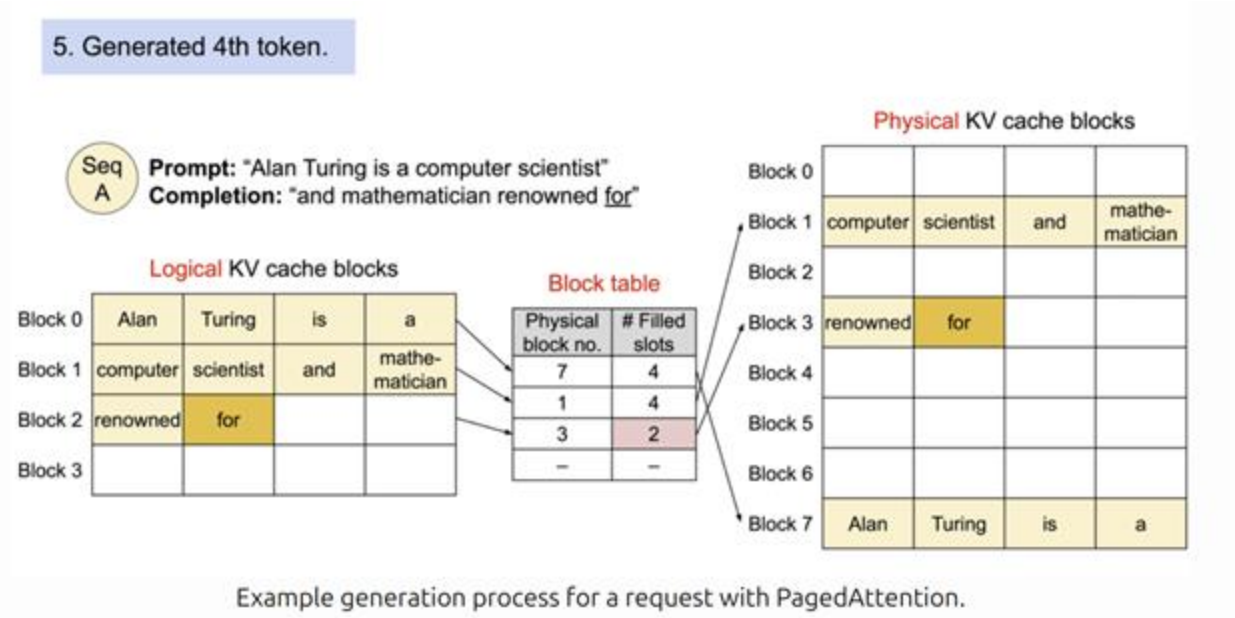
Attentions in LLM Inference Serving: Challenges

- KV-Cache storage heterogeneity
  - Page-Table (vLLM)
  - RadixTree (sglang)
  - Pruning (H2O, …)
  - Speculative Decoding (Tree Attention)
- Attention Variants
  - Different data-types, head dimensions, …
  - Different query/key/value prologue, epilogue
  - Binary size explosion
- Prefix-reusing
  - Prefix caching
- Different phases
  - Prefill/decode/chunked-prefill/speculative decoding
- Runtime dynamism
  - Variable sequence length during generation

Attentions in LLM Serving - KVCache Heterogenity

Page Attention (vLLM): efficient memory management for variable sequence length.

KV-Cache are stored in Paged KV-Cache

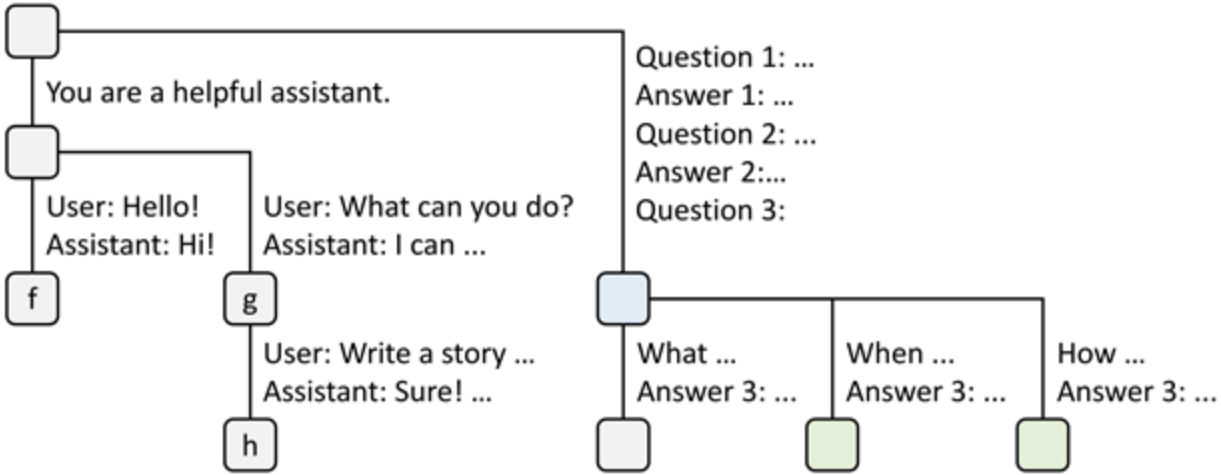


Example generation process for a request with PagedAttention.

Attentions in LLM Serving - KVCache Management

Radix Attention (sglang)

- Designed for prefix-caching and reuse.
- Page_size = 1 for less fragmentation and higher cache hit-rate.
- Duplicate prefix are organized as a Radix Tree.



Attentions in LLM Serving - KVCache Management

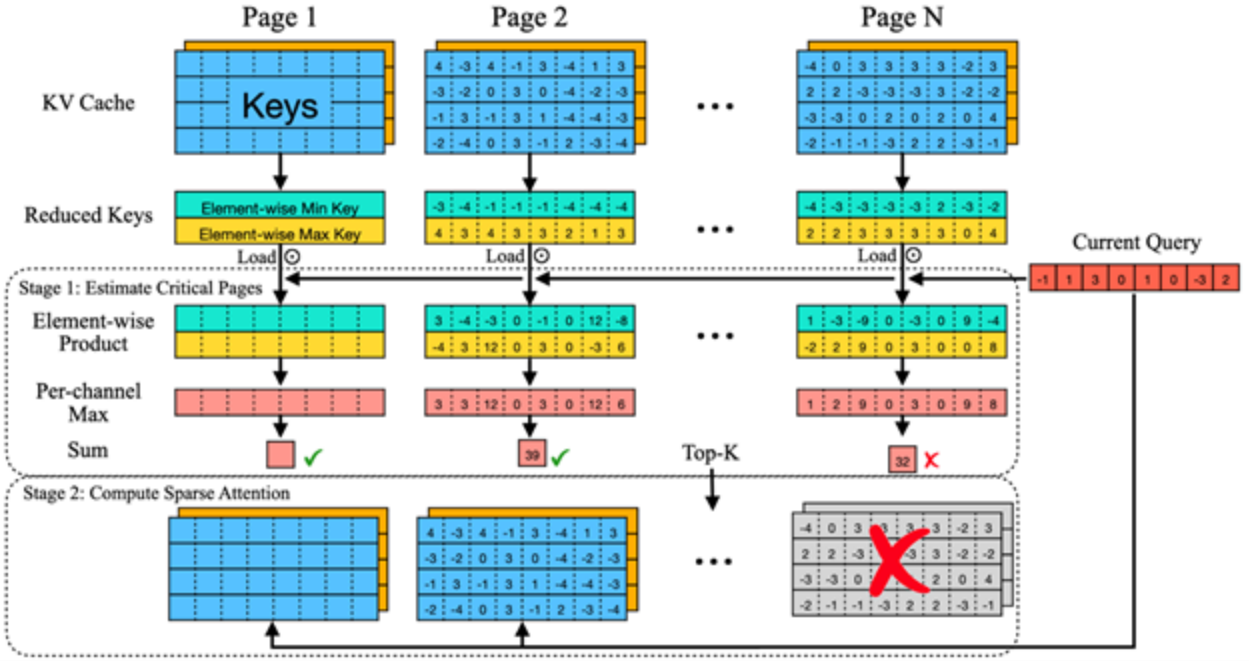
KV-Cache Compression (e.g. Quest):

Stage 1

- Computing the importance mask.

Stage 2

- Compute Sparse Attention with mask.

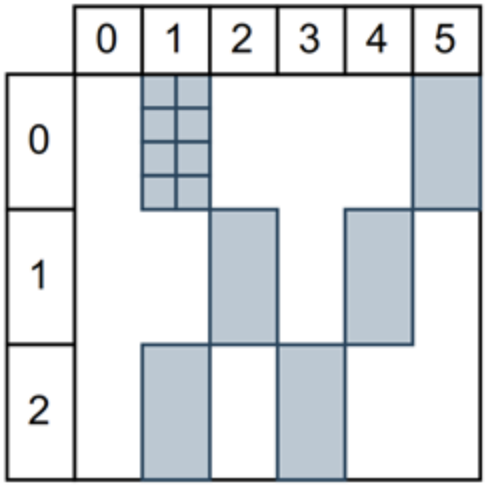


Quest: Query-Aware Sparsity for Efficient Long-Context LLM Inference

KV-Cache Storage

BSR (Block Compressed Row) Format - (B_r, B_c)

- More friendly to GPU Tensor Cores (16x8x16 for Ampere, 64xNx16 for Hopper).

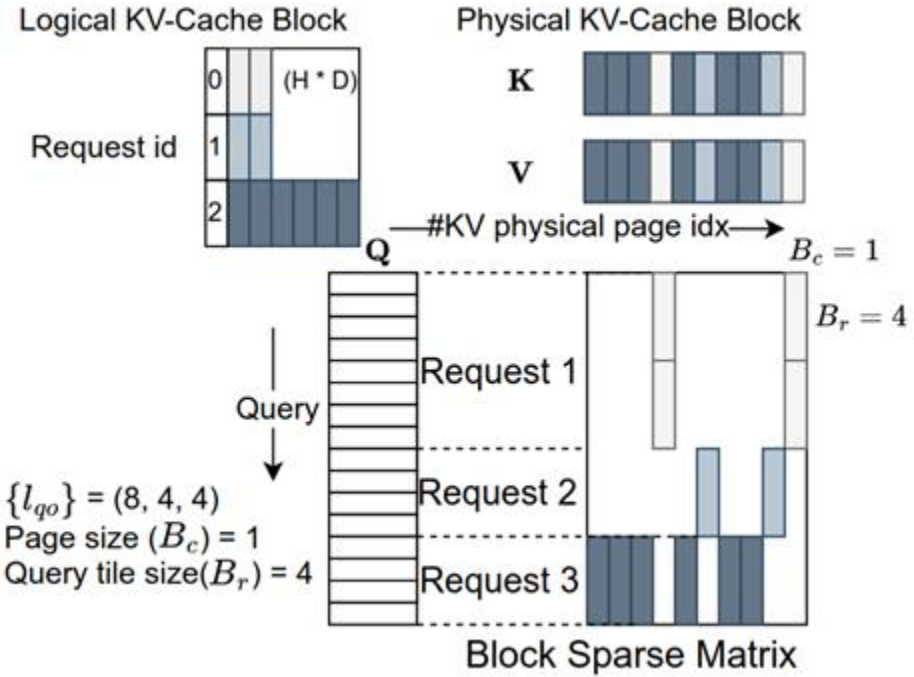


Block Sparse Format
$(B_r, B_c) = (4, 2)$

```
indices[0:2] = [1,5]

indices[2:4] = [2,4]

indices[4:6] = [1,3]

indptr = (0, 2, 4, 6)
indices = (1, 5, 2, 4, 1, 3)
value: Optional[(6, 4, 2)]
```

KV-Cache Storage

Page Attention as Block Sparse Format



KV-Cache Storage

Radix Tree and Token Tree are sparse matrix by nature.

But!!! The tree mask could be highly sparse

- Still a lot of fragmentation in block-sparse representation (128, 128).

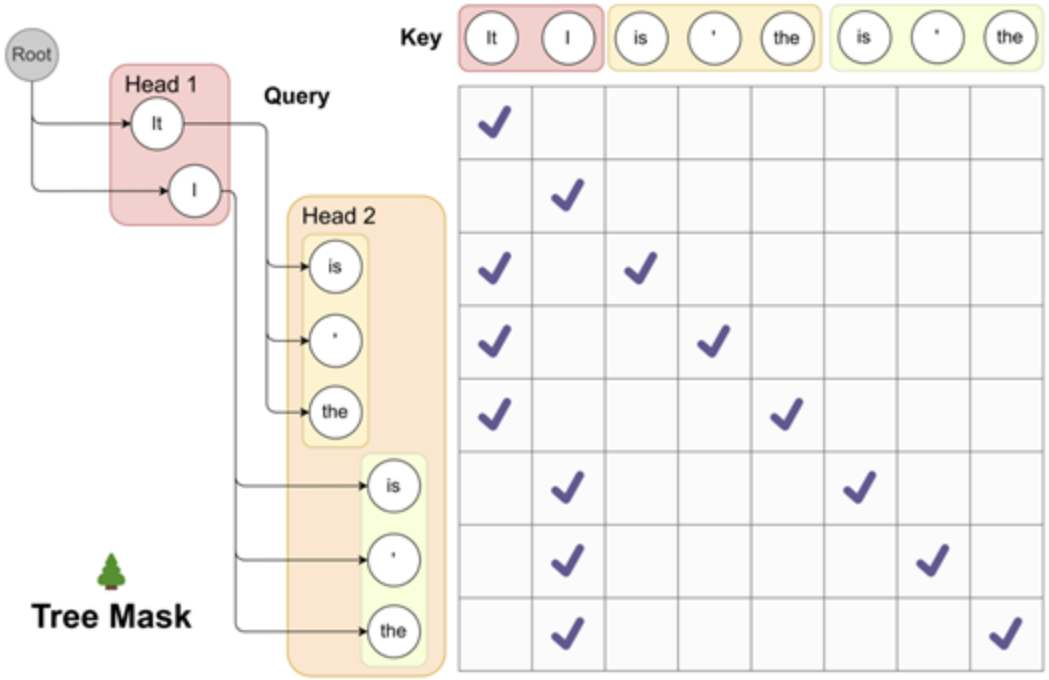


Figure from medusa

KV-Cache Storage

Column vector sparse (block sparse with block columns=1) reduce fragmentations significantly.

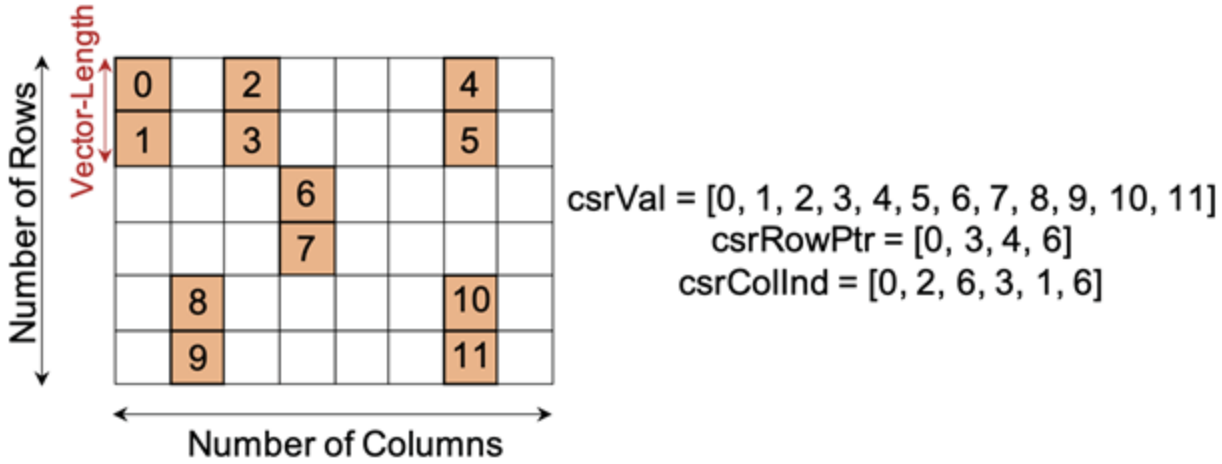


Figure 8: Column Vector Sparse Encoding.

KV-Cache Storage

Sparse rows @ sparse columns is still compatible with Tensor Cores as long as their **last dimension** is *contiguous*. (Octet Tiling, Magicube, TC-GNN, SparseTIR).

FlashInfer workflow:

- Loading rows from global memory (sparse) to shared memory (dense).
- Use dense tensor cores on shared memory. (no waste).

Memory loading is still coalesced because of the contiguous of last dimension.

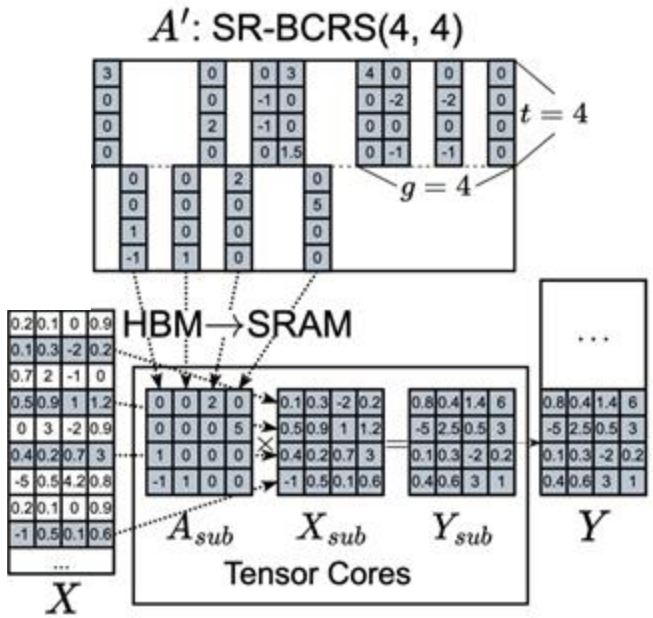
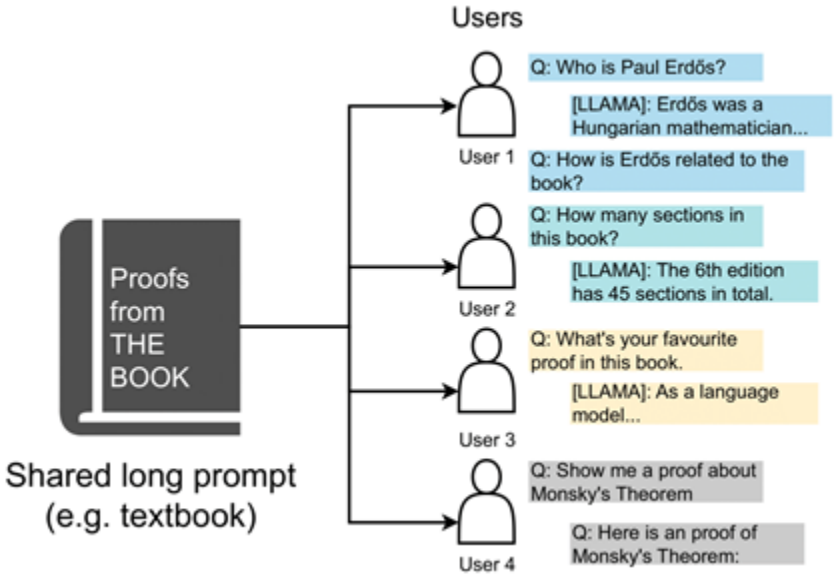
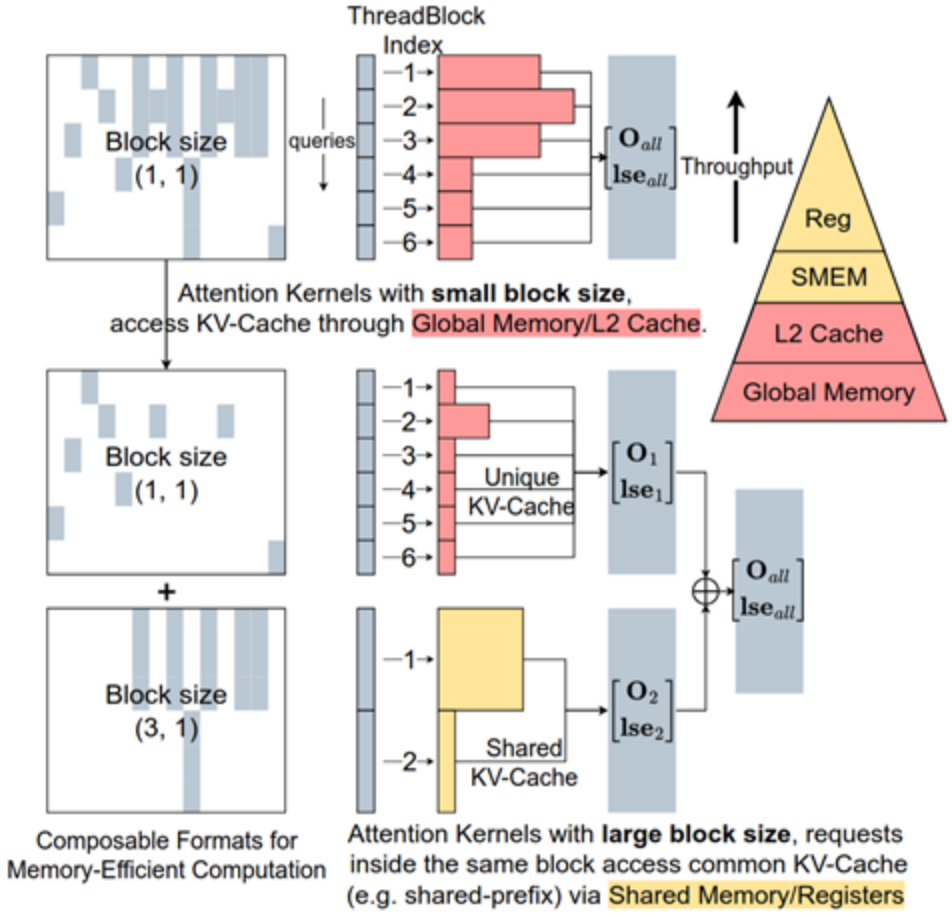


Figure from SparseTIR

Composable Formats

- Lots of shared-prefix in reasoning algorithms.

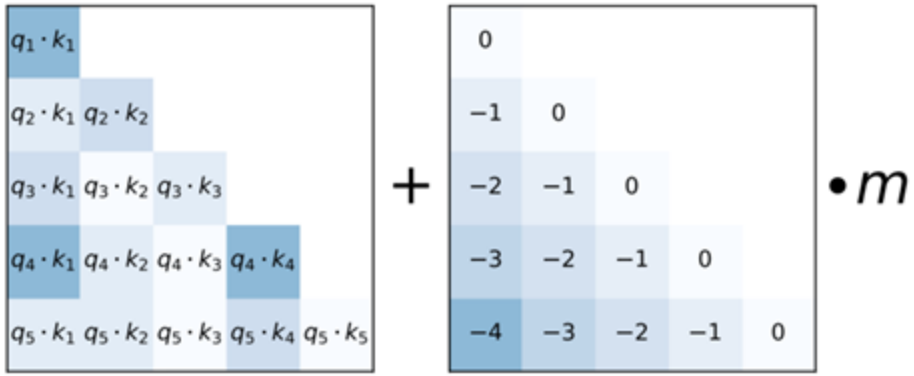


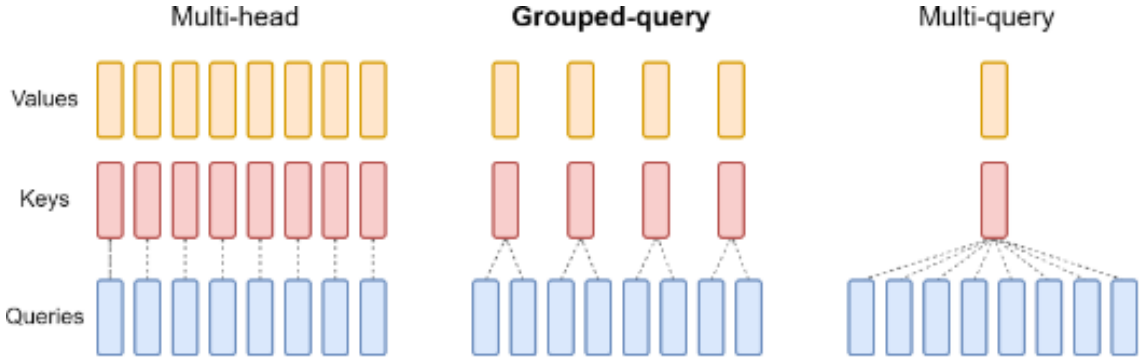


Attentions in LLM Serving - Variants

Lots of attention variants.

- Grouped Query Attention
- Multi-Latent Attention
- Grok/Gemma-style logits soft cap
- ALIBI bias
- ...





Compute Abstraction - Compiler & Runtime

JIT Compiler:

- **Customize** Attention Variants by defining functors.
- Support **continuous/sparse** KV-Cache Storage.
- **Compile-time scheduling** for different problem shapes

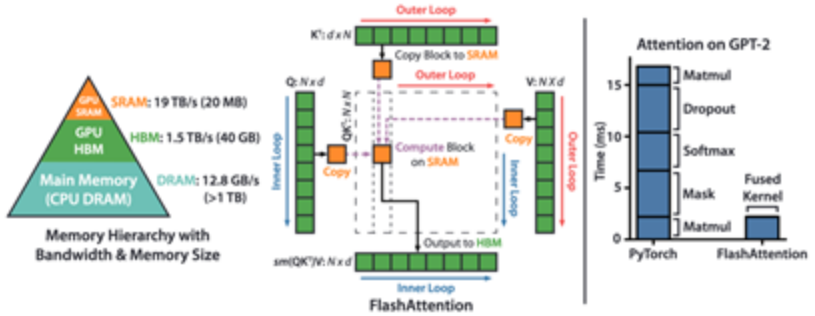
Runtime:

- **Variable-length** dynamic Scheduling
- Programming interface compatible with open-source LLM serving engines.

JIT Compiler

FlashInfer implements highly customizable CUDA/Cutlass template for FlashAttention 2&3 on:

- Batch of requests / single request
- Block Sparse (with **any** block size) KV-Cache / Contiguous KV-Cache.
  - Compatible with mainstream LLM serving frameworks.
- Programmable functors for customizing attention variants (Inspired by FlexAttention & AITemplate from Meta).



JIT Compiler

$$\text{SigmoidAttn}(\boldsymbol{X}) = \sigma(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d_{qk}})\boldsymbol{V},$$
$$\text{with } \sigma : u \mapsto \text{sigmoid}(u+b) \coloneqq (1+e^{-(u+b)})^{-1}.$$

Customize Attention Variants by user-defined **QueryTransform**, **KeyTransform**, **LogitsTransform**, **LogitsMask** functions. (inspired by **FlexAttention**)

Attention Specification in Python

```
spec_decl = r"""
template <typename Params_, typename KernelTraits_>
struct FlashSigmoid {
  using Params = typename Params_;
  using KernelTraits = typename KernelTraits_;
  static constexpr bool use_softmax = false;
  float scale, bias;
  FlashSigmoid(const Params& params, int batch_idx, uint8_t*
smem_ptr) {
    // Copy from CUDA constant memory to registers
    scale = params.scale;
    bias = params.bias;
  }
  ...
  float LogitsTransform(const Params& params, float
logit_score, int batch_idx, int qo_idx, int kv_idx, int
qo_head_idx, int kv_head_idx) {
    return 1. / (1. + expf(-(logits_score * scale + bias)));
  }
};
"""
attn_spec = AttentionSpec(
  "FlashSigmoid",
  dtype_q, dtype_kv, dtype_o, idtype, head_dim, is_sparse,
  additional_vars=[("scale", "float"), ("bias", "float")],
  additional_tensors=[],
  spec_decl=spec_decl
)
```

Part 1: Kernel Parameters Class

```
template <typename DTypeQ, typename DTypeKV, typename DTypeO,
typename IdType>
struct Params {
  DTypeQ* q;
  DTypeKV* k, v;
  DTypeO* o;
  float* lse;
  IdType* qo_indptr, kv_indptr, kv_indices, kv_seq_lens;
  ...
  // (generated) additional vars
  float scale;
  float bias;
};
```

Part 4: Register custom operators in PyTorch

```
torch::Tensor attention_call(
  torch::Tensor q, torch::Tensor k, torch::Tensor v,
  ...
  float scale, float bias  // (generated) additional vars) {
  ...
  auto kernel = KernelTemplate<FlashSigmoid<Params<{{dtype_q}},
{{dtype_kv}}, {{dtype_o}}, {{idtype}}>, KernelTraits>>;
  ...
}
// Register torch custom ops
TORCH_LIBRARY_IMPL("FlashSigmoid", CUDA, m) {
  m.impl("run", &attention_call);
}
```

Part 2: Kernel Traits class

```
struct KernelTraits {
  static constexpr HEAD_DIM = {{head_dim}};
  static constexpr IS_SPARSE = {{is_sparse}};
  ...
};
```

Part 3: Kernel Body

```
template <typename AttentionSpec>
__global__ KernelTemplate(
  AttentionSpec::Params params) {
  ...
  // Init attention specification class.
  AttentionSpec attn(params, batch_idx, smem_ptr);
  ...
  // Iterate over all elements inside the thread logits tile.
  for (int i = 0; i < size(logits_tile); ++i) {
    // convert register index i to qo_idx, kv_idx, etc.
    qo_idx = get<0>(logits_tile(i));
    kv_idx = get<1>(logits_tile(i));
    ...
    logits_tile(i) = attn.LogitsTransform(
      params, logits_tile(i),
      batch_idx, qo_idx, kv_idx,
      qo_head_idx, kv_head_idx);
  }
  ...
}
```

JIT Compiler

Customize Attention Variants by user-defined **QueryTransform**, **LogitsTransform**, **LogitsMask** functions.

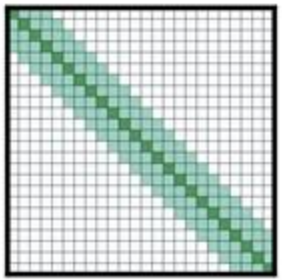
- ALIBI Attention (Relative Attention Additional Bias)
- RoPE Attention

$$\mathrm{RoPE}(\mathbf{q}, \mathrm{offset}(q)) \cdot \mathrm{RoPE}(\mathbf{k}, \mathrm{offset}(k))^T$$

- Sliding Window Attention

$$(\mathbf{q} \cdot \mathbf{k}^T) \cdot (\mathrm{offset}(q) - \mathrm{offset}(k) < w)$$

- Custom Attention Mask
- ...

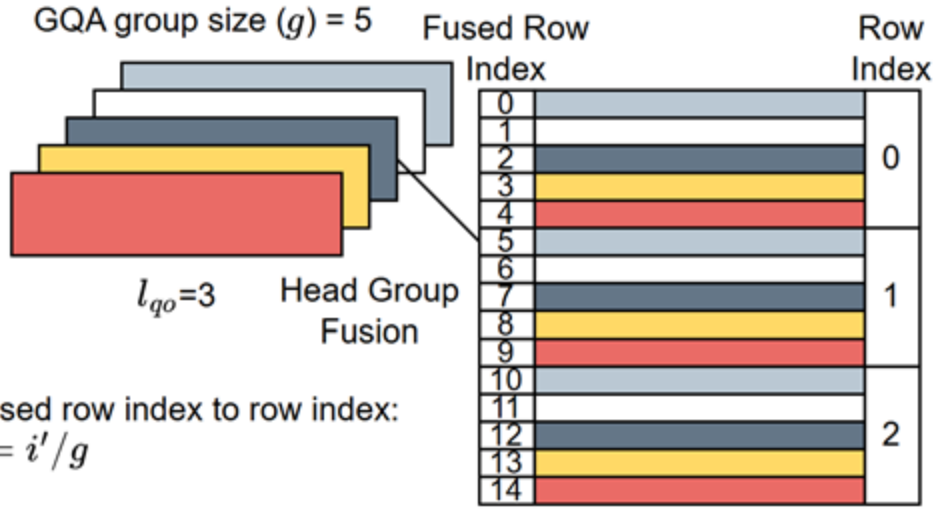


(b) Sliding window attention

Attention Template - Group Query Attention

Head-Group Packing

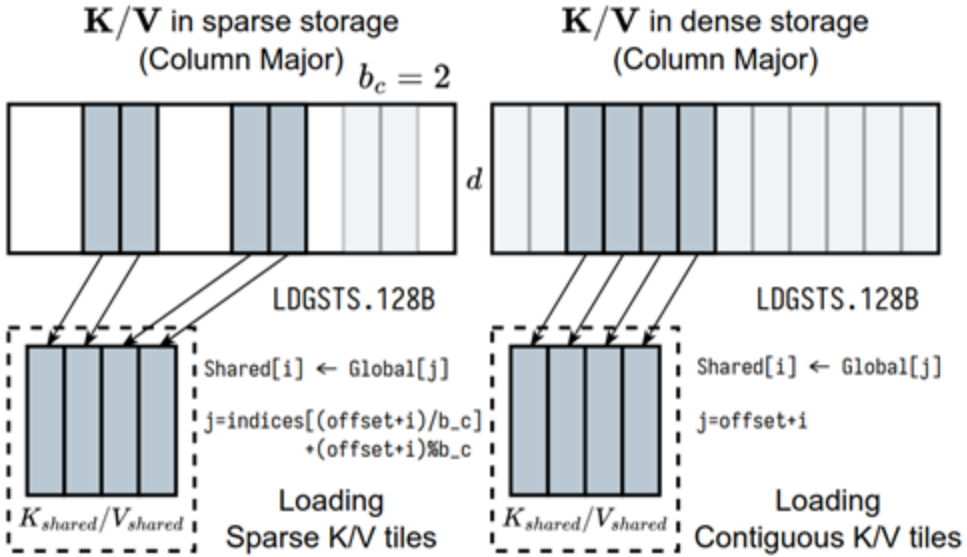
Fuse query rows and head groups to increase the operational intensity of decode/incremental prefill attention operators.



Attention Template - Sparse Gathering

Sparse Global to Shared Memcpy

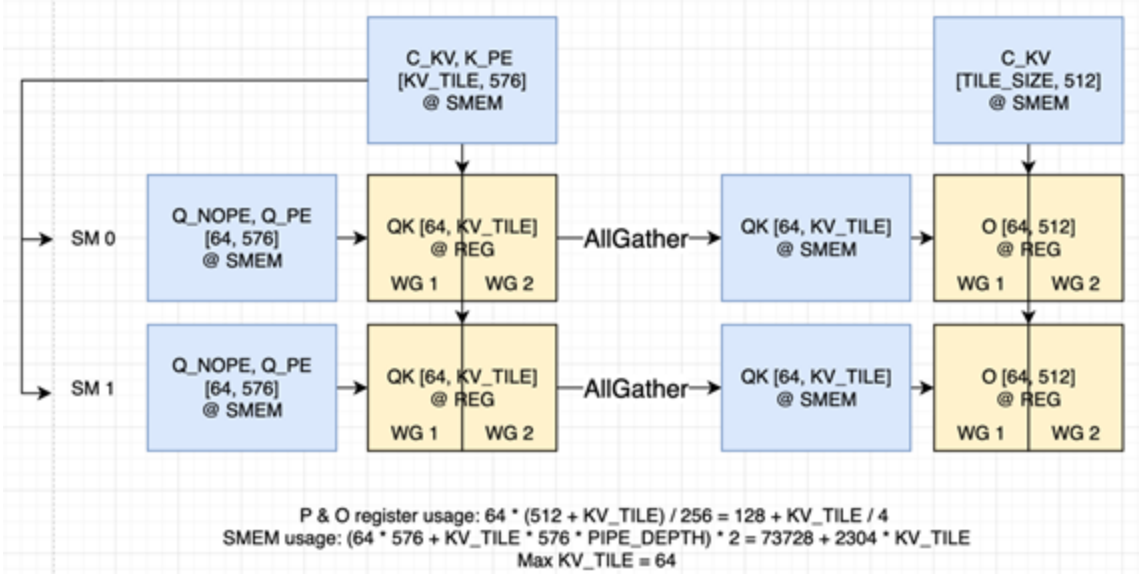
- For contiguous storage, issue TMA instructions for each tile.
- For sparse storage, issue multiple LDGSTS async-copy instructions.
  - TMA2D is not applicable for gather/scatter.



Attention Template (MLA)

Multi-head Latency Attention (MLA) is a special form of Grouped Query Attention (GQA) with large head dimension (**not enough registers for storing output**).

- Split on head dimension



Attentions in LLM Serving - Workload Heterogeneity

Decode:

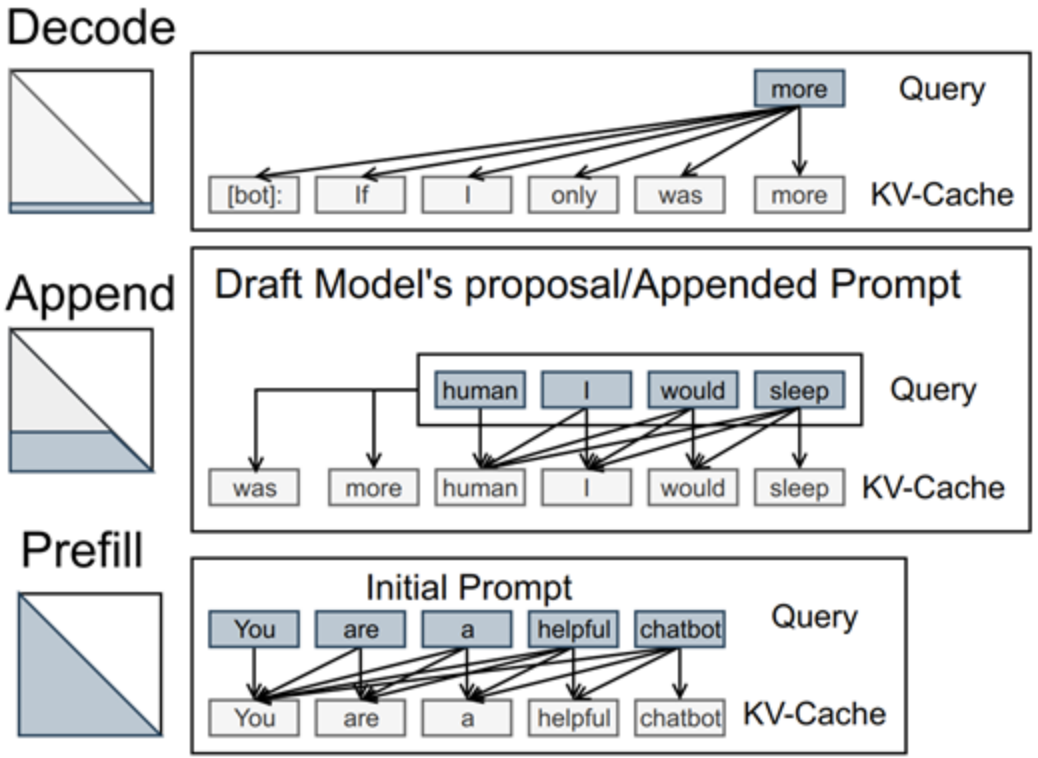
- One token at a time
- len(Q) = 1

Prefill:

- len(Q) = len(KV)

Append (a.k.a incremental prefill)

- Several tokens at a time.
- len(Q) < len(KV)



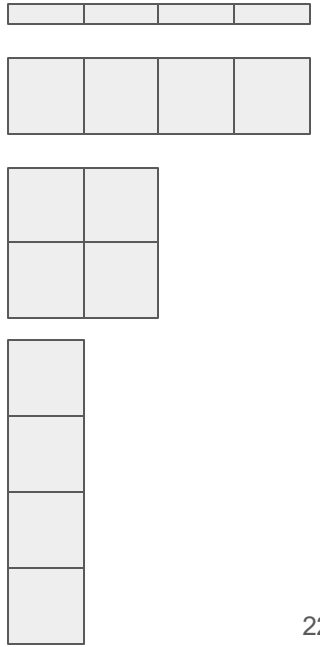
Tile-size selection

Select tile-size according to query length.

- Different tile-size for different problem shapes
  - CUDA-cores (67 TFLOPs/s), tile-size: 1x512
  - Tensor-cores (989 TFLOPs/s), tile-size: 16 x 512
  - Tensor-cores (989 TFLOPs/s), tile-size: 32 x 256
  - Tensor-cores (989 TFLOPs/s), tile-size: 64 x 128
  - Tensor-cores (989 TFLOPs/s), tile-size: 128 x 64
  - ...

- Various pipeline design (for H100 and later)
  - 2 warp groups / 3 warp groups for MLA

Tile Layouts



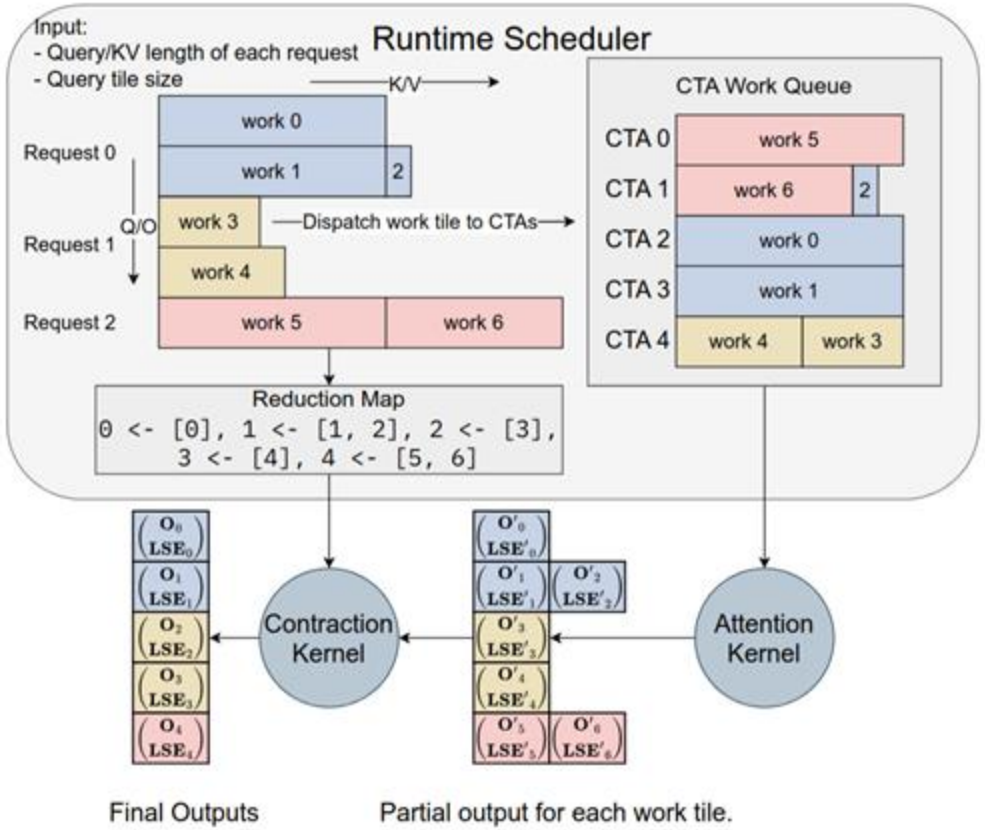
Runtime Scheduler

Cost-model based **deterministic** scheduling for variable length requests.

Inspect sequence length information **ahead-of-time**.

Goal:

- Load-balancing
- Zero wave-quantization



Compute Abstraction

Integration with LLM Serving Engines

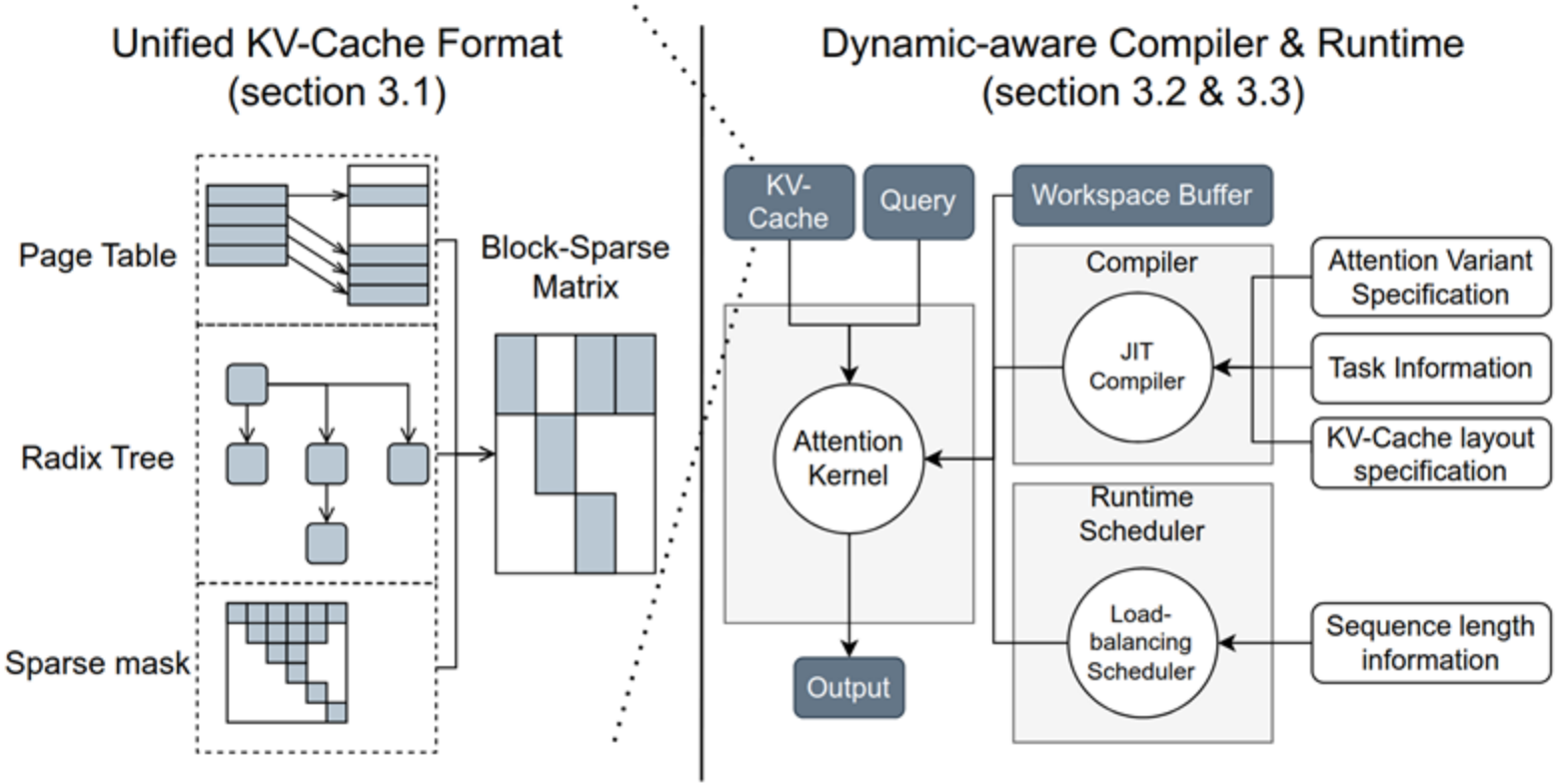
- **Init**: JIT compilation of kernels of each attention configuration (problem size, data-types, attention variants).
- **Plan**: dynamic kernel scheduling
  - Load-balancing
  - Deterministic (reduction in deterministic order).
  - Cost amortized by multiple layers.
- **Run**: execute kernel according to plan information.

Inspector (plan)-Executor (run) mode.

Compatible with `torch.compile` and `CUDAGraph`.

```
# init per configuration
wrapper = AttentionWrapper(task_info)

step = 0
while True:
  step += 1
  # plan per generation step
  wrapper.plan(...)
  for i in range(layers):
    ...
    # run per layer
    wrapper.run(...)
    ...

# For CUDA graphs
g = torch.cuda.CUDAGraph()

# warmup
with torch.cuda.graph(g):
  for i in range(layers):
    ...
    wrapper.run(...)
    ...

step = 0
while True:
  step += 1
  # plan per generation step
  wrapper.plan(...)
  # replay CUDA-Graph
  g.replay()
```

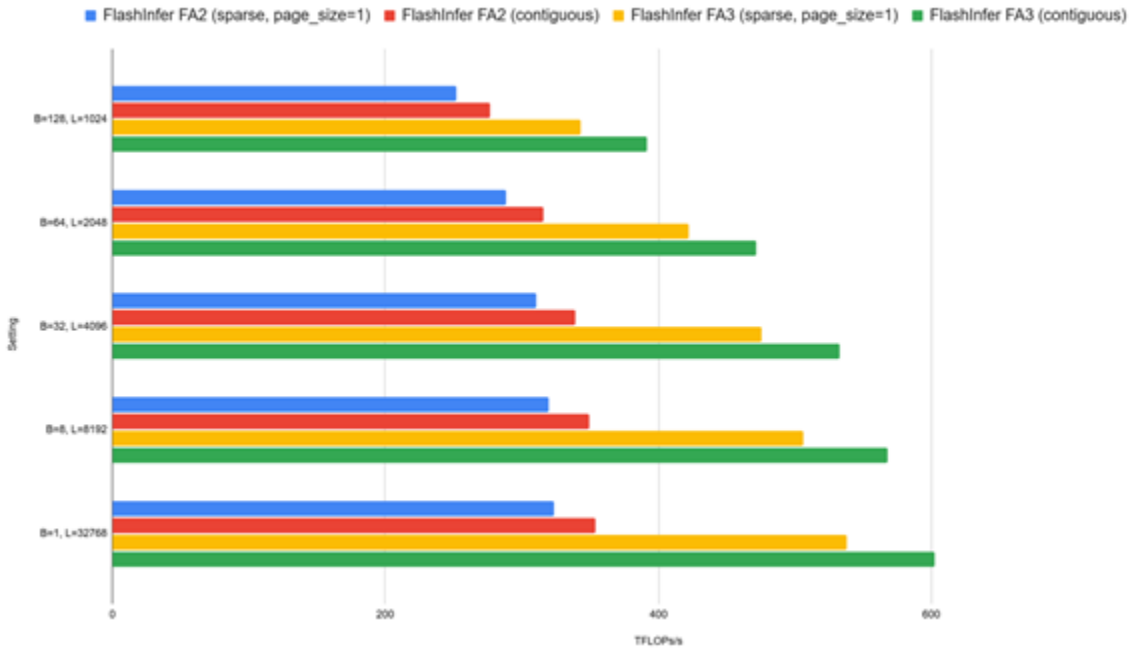
FlashInfer: Storage and Compute Abstractions



Results - Paged Attention

Vector-Sparse Attention (page_size=1) vs Dense Attention

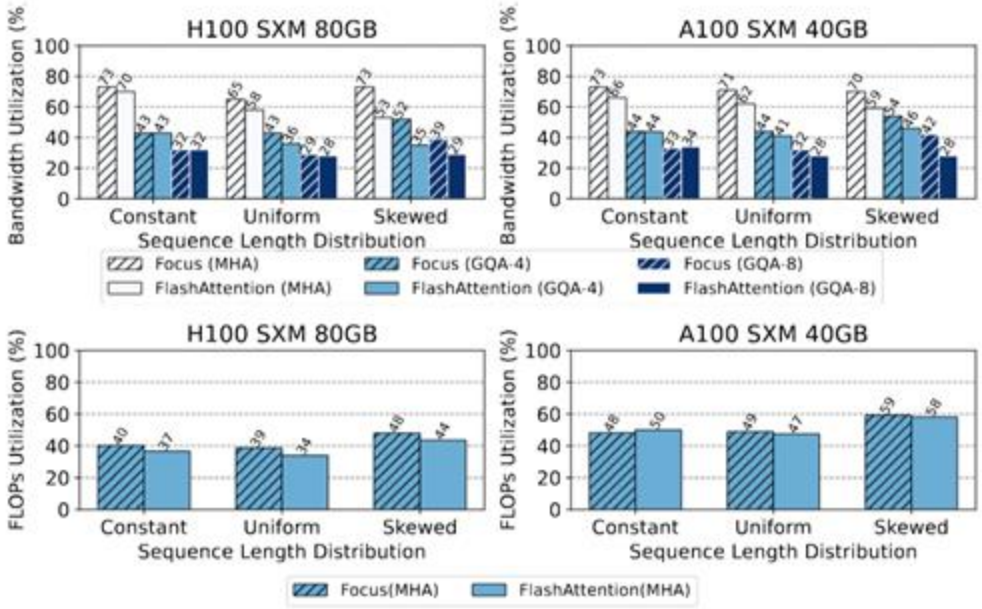
Sparsity at 10% overhead



Results - Load Balancing

Variable Split-K for better load balancing in variable length settings

- Constant/Uniform/Zipf distribution



Results (end-to-end)

FlashInfer w/ leading open-source LLM serving - sglang

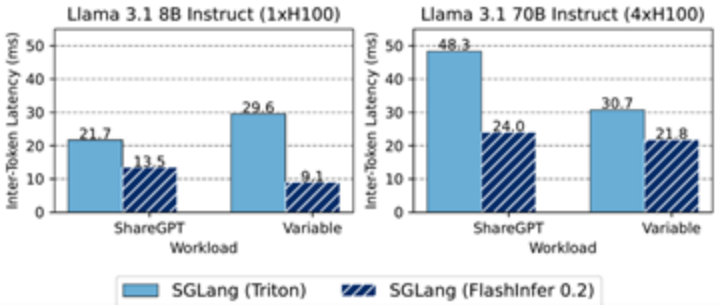
Performance on Llama 3.1 70B on 4xH100

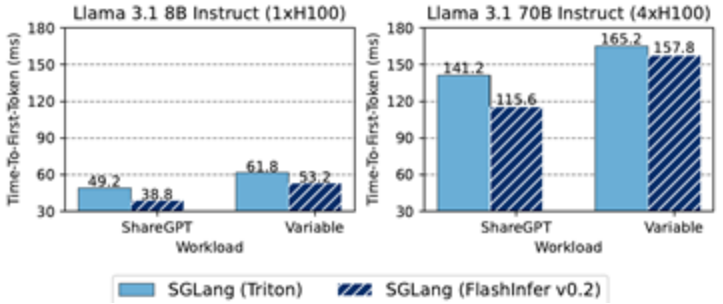
- Sglang (FlashInfer)
- Sglang (Triton)

Workloads

- ShareGPT
- Variable distribution

Up to 2x latency reduction than Triton backend

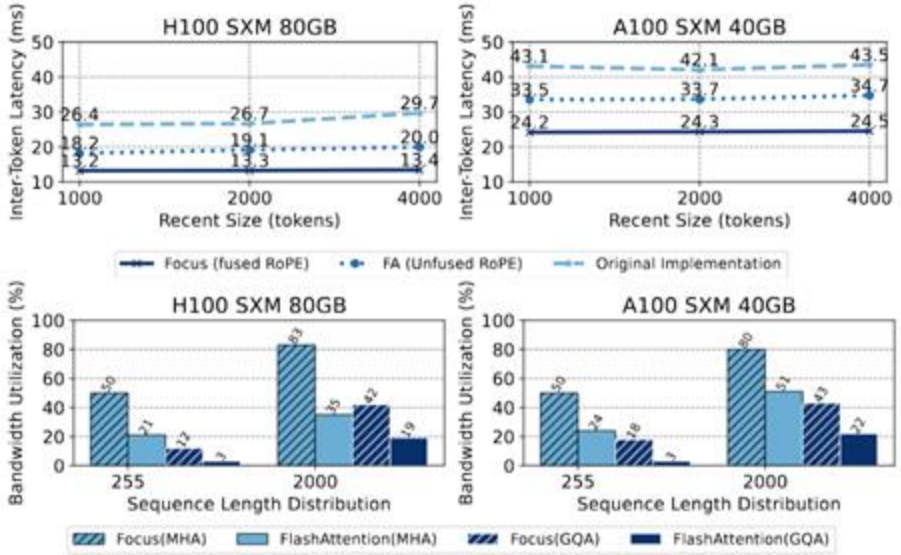




Results - Customized Attention

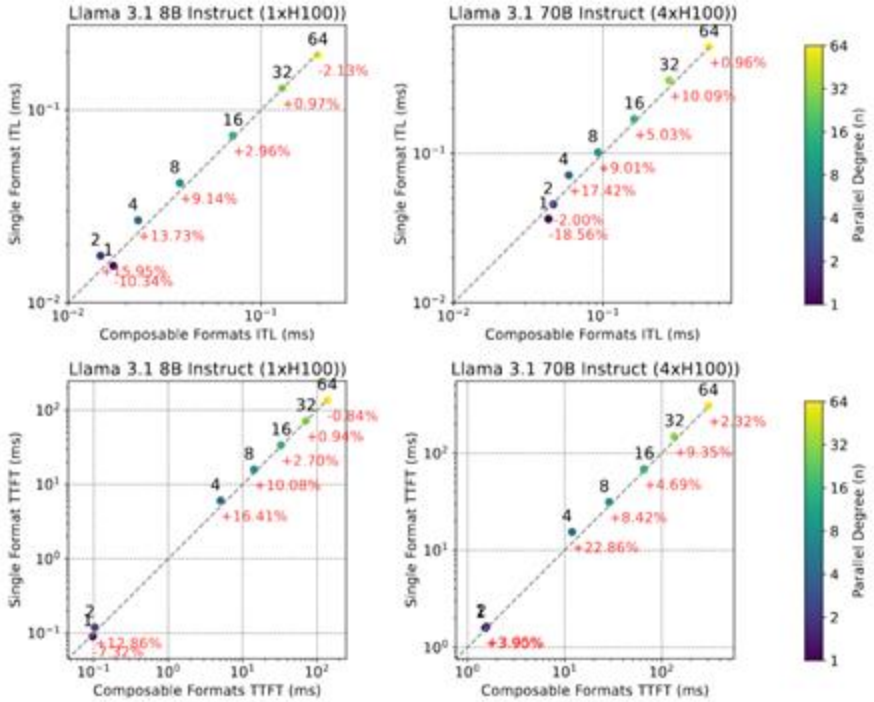
Customized Attentions (fused-RoPE attention in Streaming-LLM)

- 2x acceleration at kernel level.
- 30% end-to-end latency reduction on Streaming-LLM



Results - Parallel Generation

Parallel Generation (with different “n” in OpenAI API)

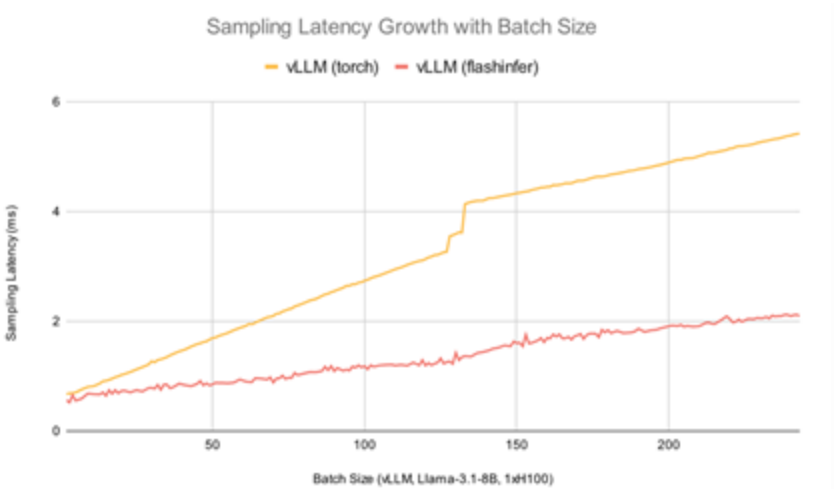


LLM-Specific Operators

- **Top-P/Top-K sampling**
  - Relies on sorting (slow on GPU)
  - Vocabulary size growing in recent years (from 32k to 128k)
  - Lots of data-dependent operators that are hard to fuse.

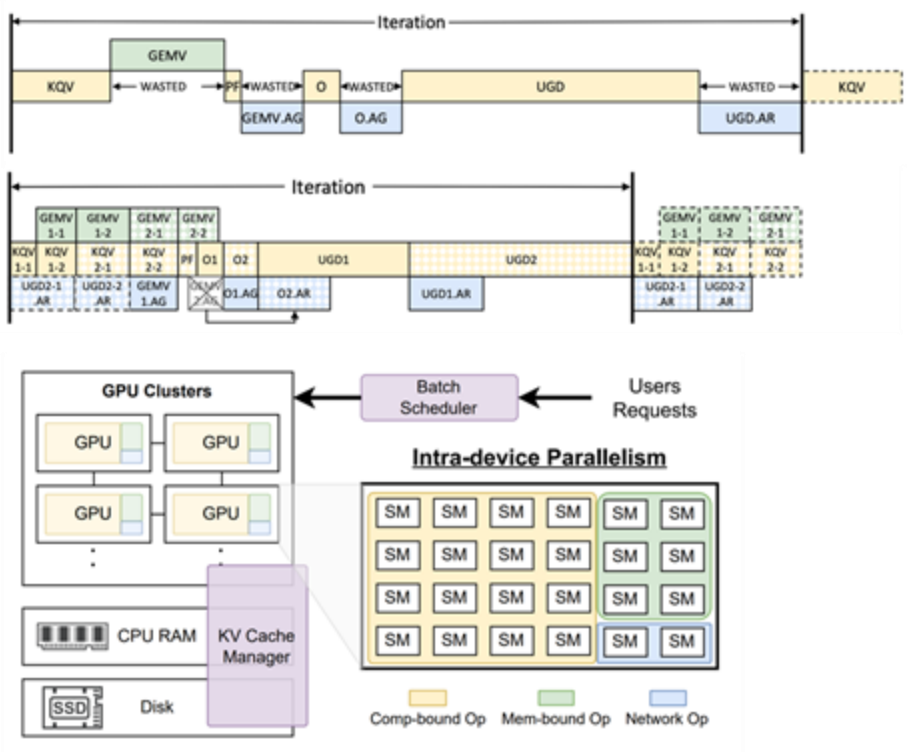
- FlashInfer provides template for **rejection sampling** on GPUs
  - Not relying on sorting
  - Fast convergence (within 10 rounds)
  - **User-defined filter functions** (top-p/top-k/min-p/etc).

- Next step: compile the entire post-processing pipeline (w/ repetition penalty, etc.)



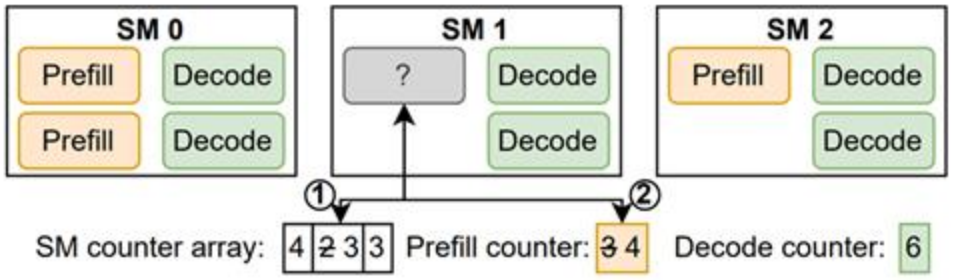
Next Steps - Holistic Kernel

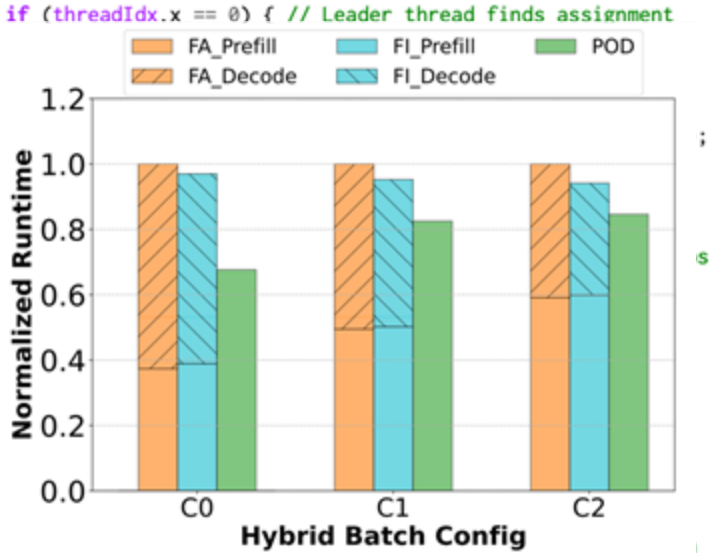
- Imbalance between I/O, Compute, Network bandwidth usage.
- Overlap Compute-Bound, I/O-Bound, Networking operators
  - Split requests into nano-batches.
  - Dispatch different workload to different SM on GPUs.
  - Use multi-stream to isolate the operators.
- Kernel generation in Nanoflow:
  - SM-Constrained (Horizontally fused) GEMM/Communication/Attention operators.
  - Customization for different models.
  - Using Cutlass as backbone GEMM generator.



Next Steps - Holistic Kernel

- Example: Chunked Prefill
  - Piggybacking decode with prefill, require attention kernels.
  - Single kernel with fixed tile size
    - Sub-optimal for prefill or sub-optimal for decode
  - One prefill kernel + one decode kernel
    - Low-occupancy for prefill kernel
  - POD-Attention (Aditya et al.)
    - Compile both large/small tile size.
    - Runtime dispatch to one implementation.

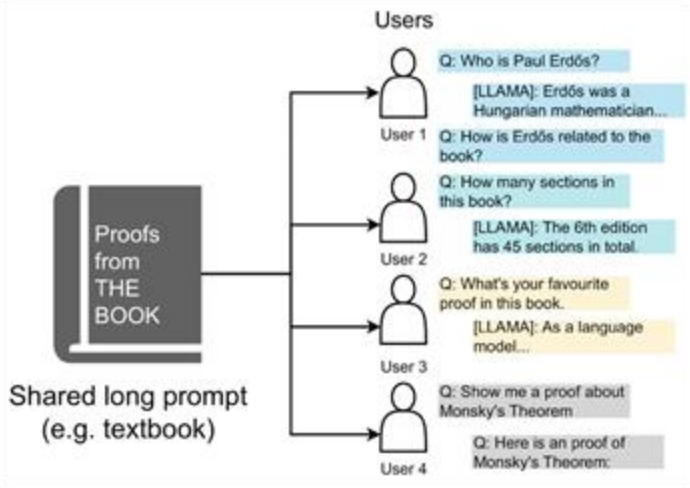


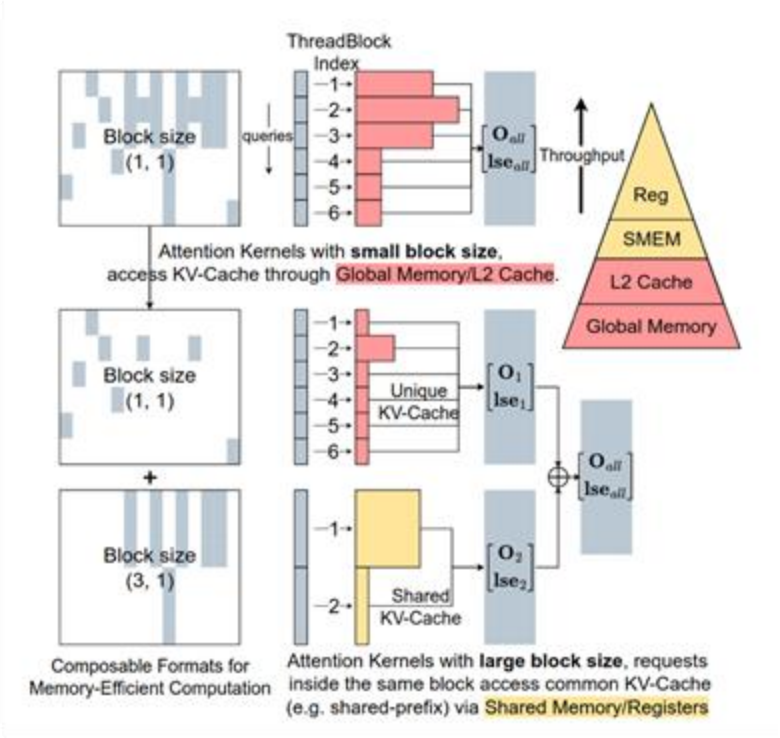



```
if (threadIdx.x == 0) { // Leader thread finds assignment
// Fetch the assigned CTA and operation.
int cta_id = shared_mem[0];
const int op = shared_mem[1];
__syncthreads();
// Perform the appropriate operation
if (op == PREFILL) prefill_op(cta_id);
else decode_op(cta_id)
```

Next Steps - Holistic Kernel

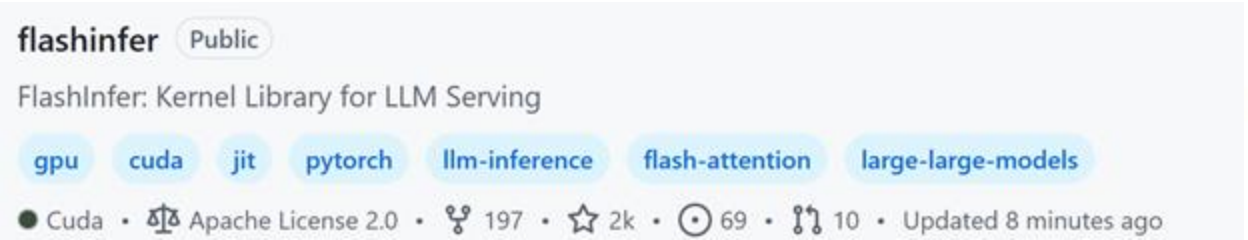
- Lots of opportunities for **prefix-cache reuse** in agent-based system
  - Can be formulated as attention on several different sparse matrix, each with different block size.
- **Low-occupancy for each kernel.**
- **Holistic Kernel**
  - Compile fused kernel for all tile sizes kernels.
  - Software scheduling for dispatching and load-balancing.





Community

- Originated as a research project.
- After over a year of development, the project now has 50+ contributors from both industry and academia.



- Widely adopted by leading open-source LLM engines, including MLC-LLM, vLLM, sglang, Hugging Face TGI (Text Generation Interface), Nanoflow, and others.

SGL vLLM Text Generation Inference MACHINE LEARNING COMPILATION NanoFlow

FlashInfer - Kernel Library for LLM Serving

Position: Expanding on existing operators in PyTorch, cuBLAS, and similar libraries for efficient LLM inference, with focus on:

- Attention Operators
  - Customizable Attention Templates optimized for LLM Serving scenario.
- GEMMs
  - SM-Constraint GEMM and Grouped GEMM (call cutlass under the hood).
- LLM-specific operators (Sampling, etc)
- Holistic Kernel Generation.

Mission: open-source CUDA kernel generator/library for foundation models.

- Welcome to join the community!
- Slack workspace for open discussions 👉

Backup Slides