北京智源人工智能研究院
BEIJING ACADEMY OF ARTIFICIAL INTELLIGENCE

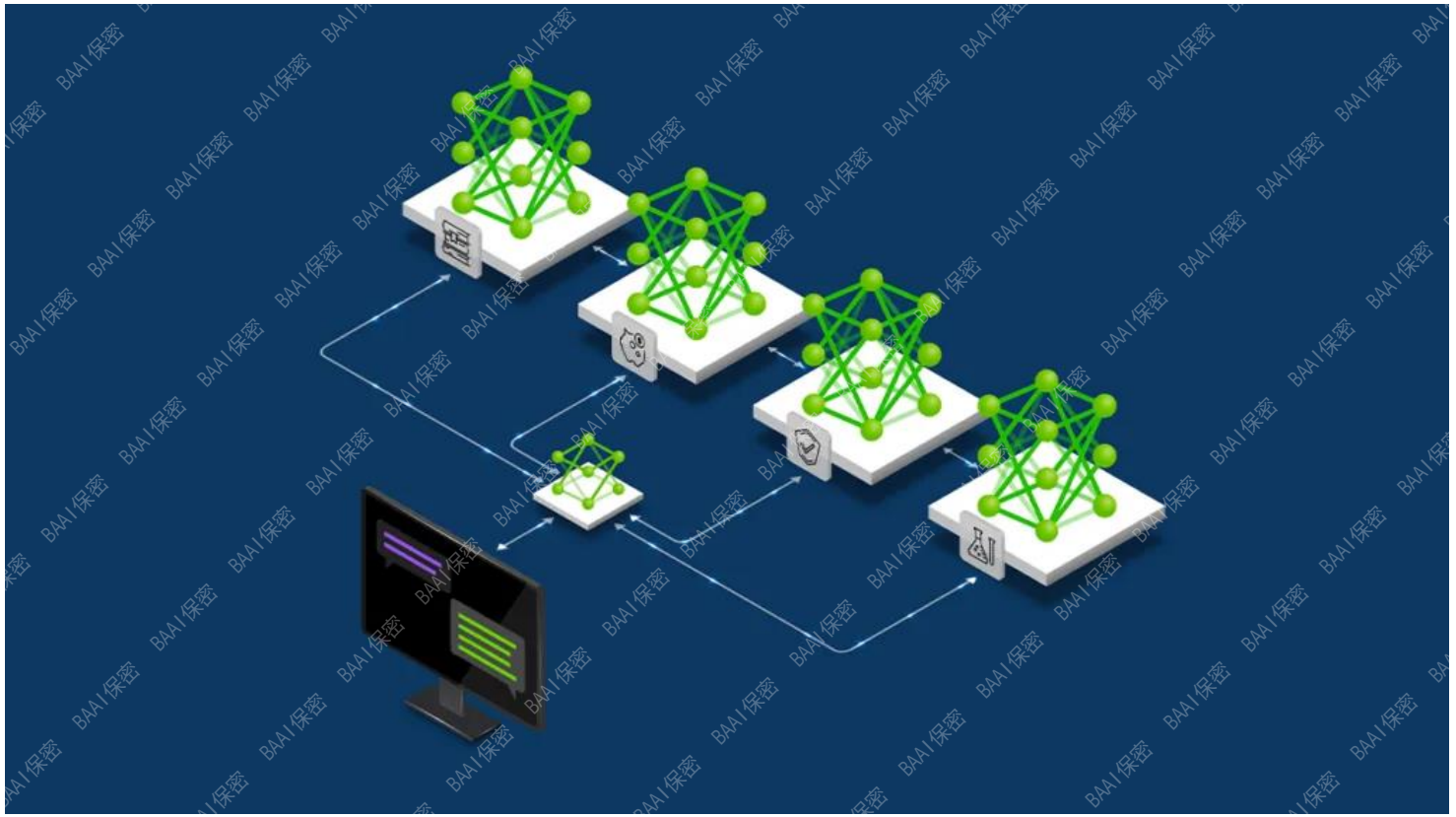
FlagScale大模型高效推理部署优化技术

目录

PART 1：大模型推理部署技术简述

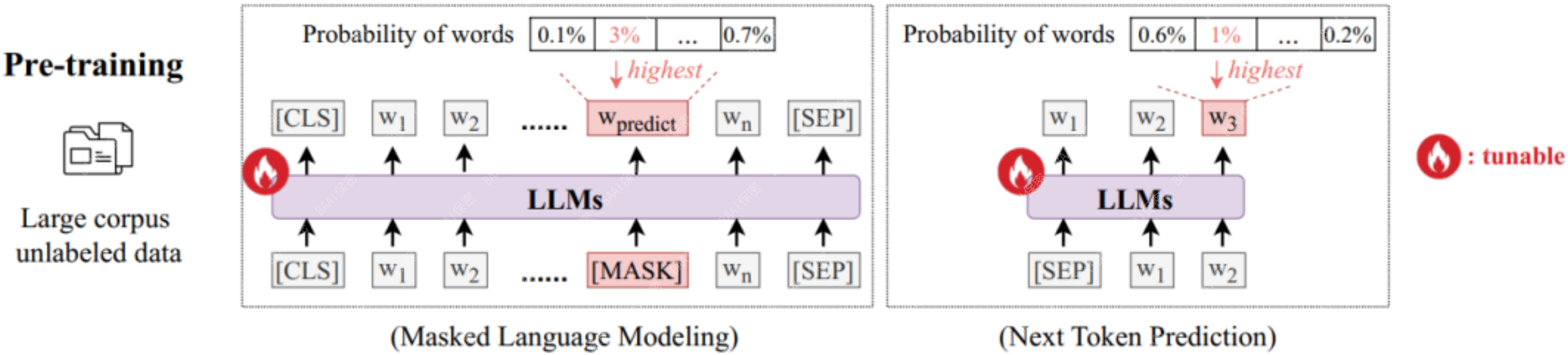
大模型推理部署技术简述— 概述

推理部署：是指将训练完成的大规模人工智能模型（如GPT、BERT、LLaMA、ChatGLM等）**转换成可用于生产环境中的服务或程序**，以便进行实际的**推理任务**，如问答、文本生成、图像识别等。



source: https://developer.nvidia.com/blog/dynamo-0-4-delivers-4x-faster-performance-slo-based-autoscaling-and-real-time-observability/

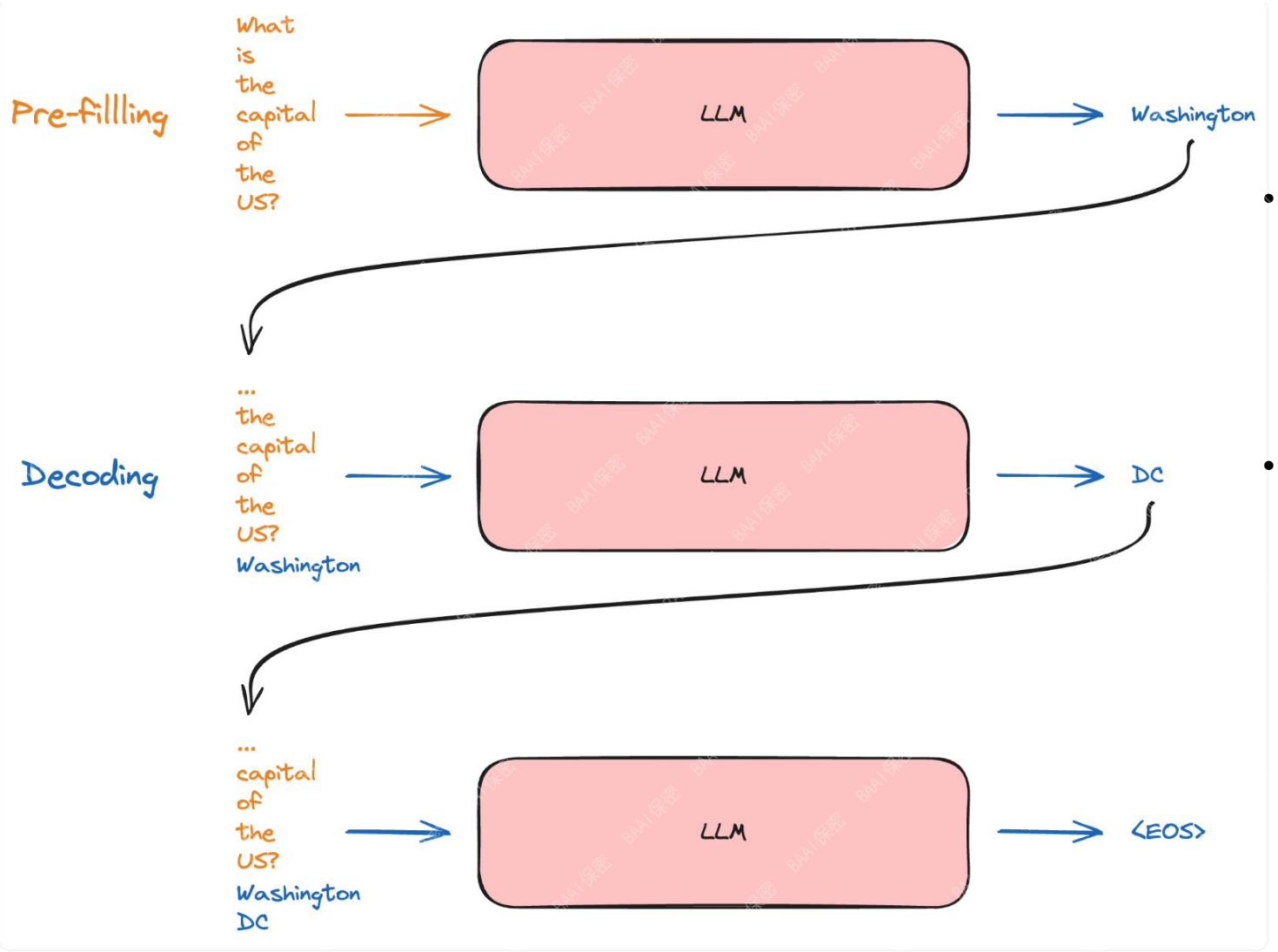
大模型推理部署技术简述 — 概述



source: https://something.plus/article/47b17a6d-c088-464d-bd3b-af1a679755fe

Next-Token Prediction自回归推理：给定一个词序列的上下文，模型预测下一个最可能的词。

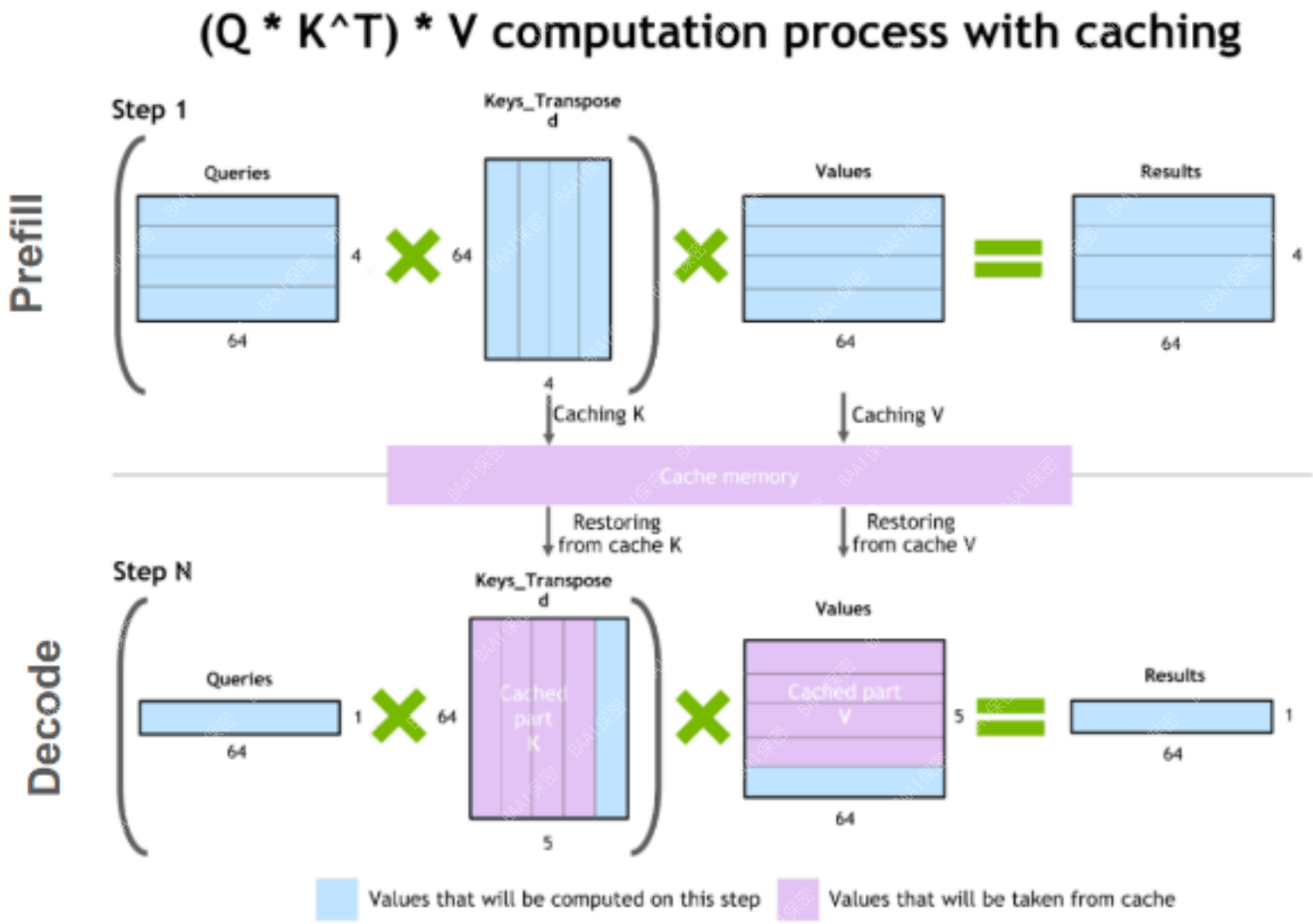
大模型推理部署技术简述— Prefilling & Decoding



- Prefilling阶段：
  根据输入prompt计算第一个输出token。输入是完整的，并行的矩阵计算，可以有效的利用GPU计算能力，属于计算密集型。
- Decoding阶段：
  模型以自回归的方式逐个生成输出token，直到满足条件为止，每次的输入只有一个token，实际实行矩阵向量计算，不能有效率用GPU计算能力，数据从内存传到GPU的速度决定整体时延，属于内存密集型。

source: https://huggingface.co/blog/tgi-benchmarking

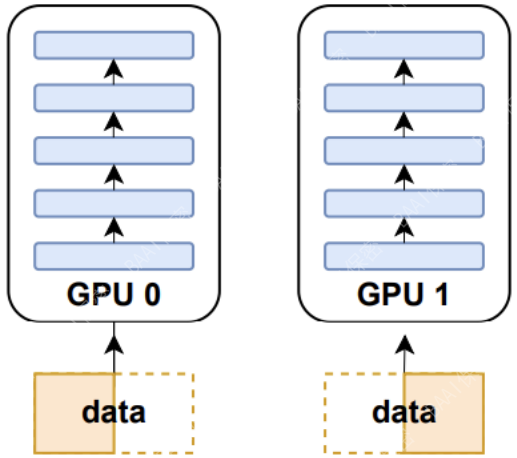
大模型推理部署技术简述— KV Caching



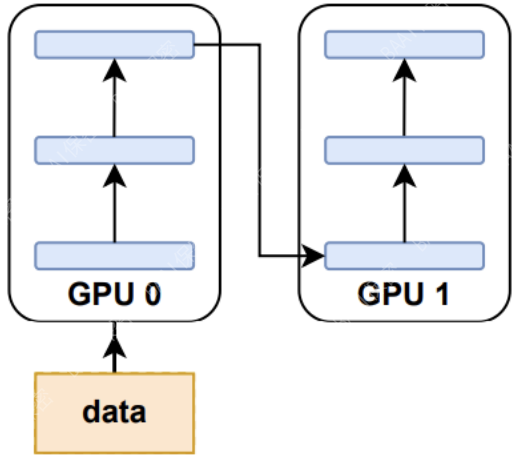
source: https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

- Key-Value缓存：
  每一步解码token的计算都需要依赖当前token和之前所有token之间的注意力，即依赖之前token的key-value，为了避免重复计算，可以把这些key-value都缓存在GPU中，以便在下一次迭代中使用。

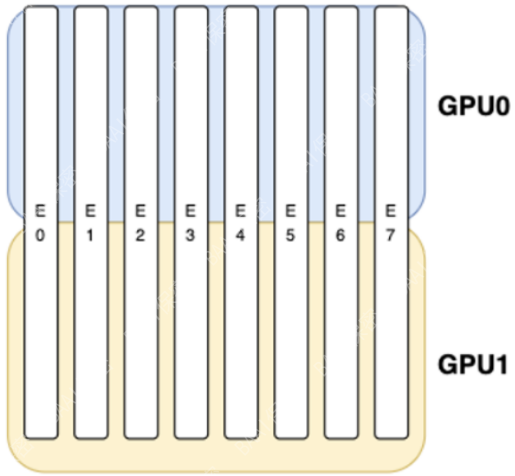
大模型推理部署技术简述— 并行策略



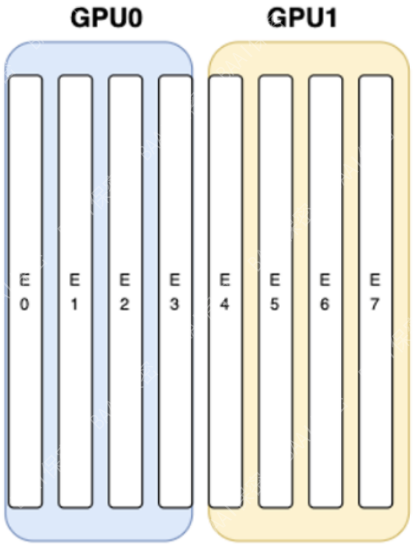
data parallel



pipeline parallel



Tensor Parallel



Expert Parallel

source: https://arxiv.org/pdf/2110.14883, https://nvidia.github.io/TensorRT-LLM/advanced/expert-parallelism.html

并行策略均可以混合使用。

PART 2：大模型推理优化技术

大模型推理优化技术 — vLLM框架特色

特色：

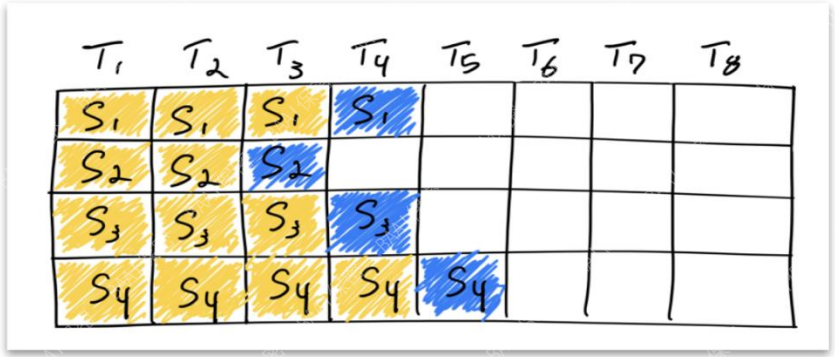
- **Continuous Batching**
  动态调整批处理大小，将请求拆分为预填充和解码阶段，以最大限度地提高 GPU 利用率。

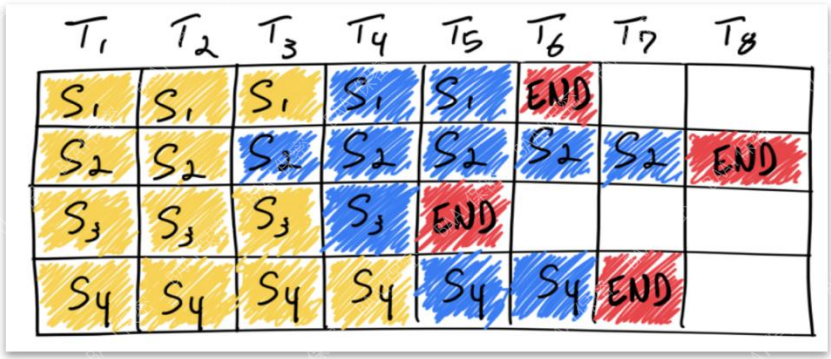
- **PagedAttention**
  借鉴传统的分页操作，把Key-Value拆分成固定块大小，以实现GPU内存的动态分配。

- **Zero Redundancy Tensor Parallelism**
  利用 NCCL/MPI 在多个 GPU 之间进行高效的权重划分和同步，从而提高计算效率。

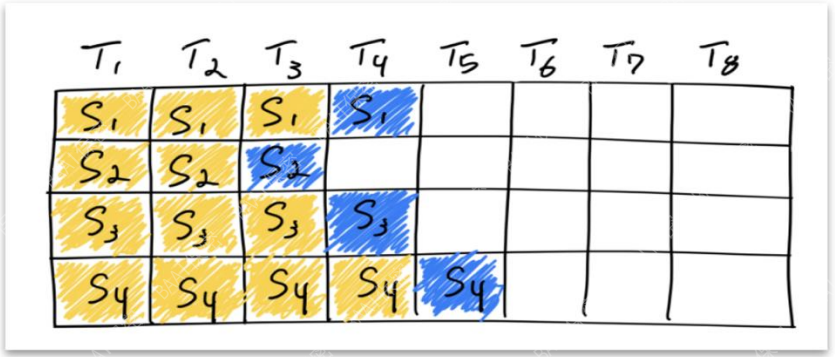
大模型推理优化技术 — continuous batching

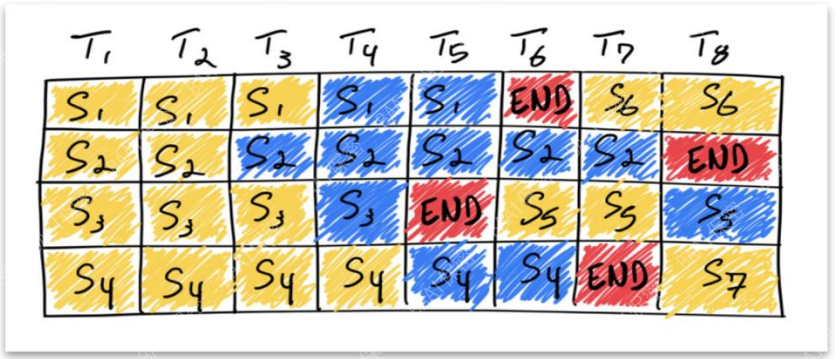
Dynamic Batching 🙅🙅🙅





Continuous Batching 🙏🙏🙏



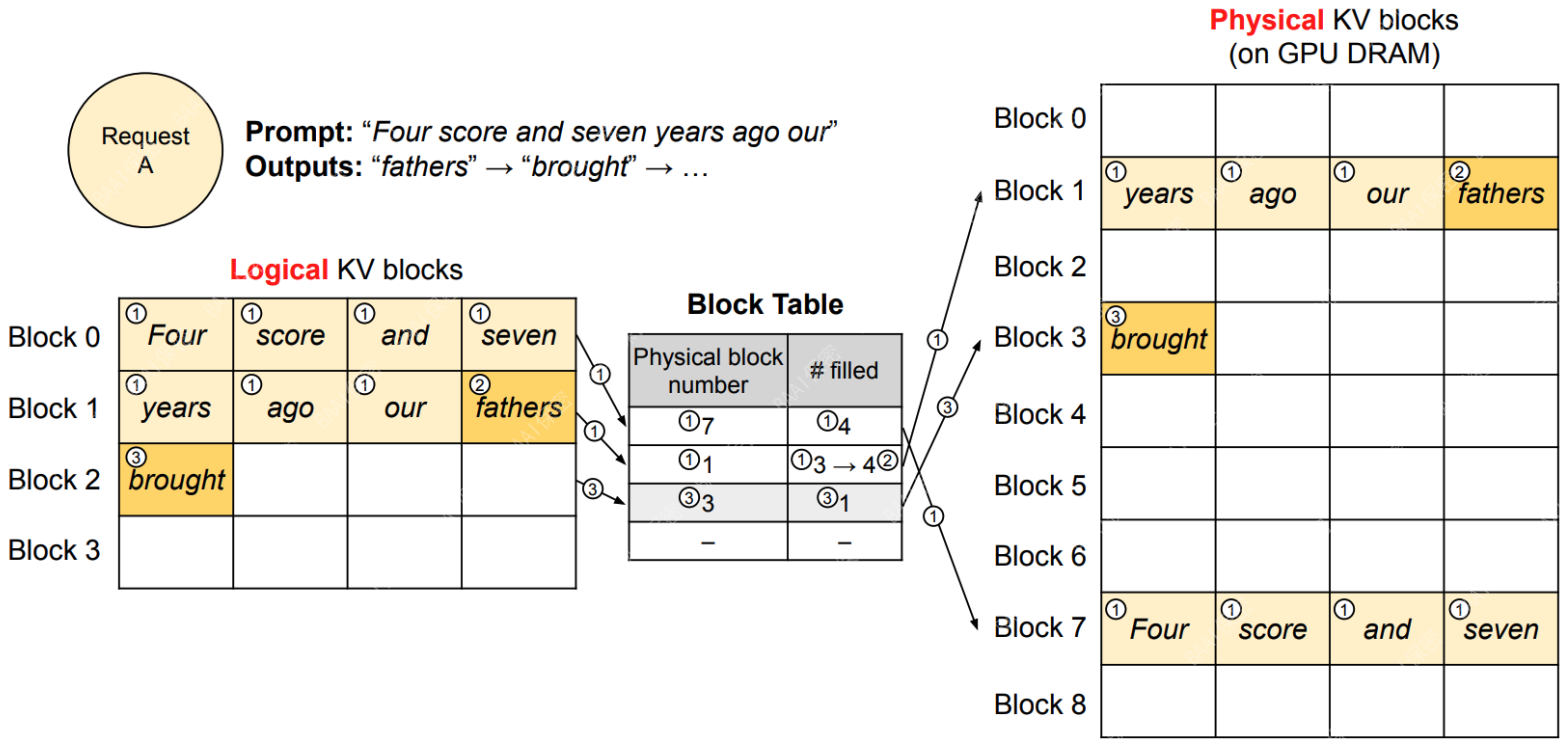


source: https://docs.google.com/presentation/d/1_q_aW_ioMJWUImf1s1YM-ZhjXz8cUeL0IJvaquOYBeA/edit?slide=id.g351f0c84ff2_0_304#slide=id.g351f0c84ff2_0_304

传统的Batch流程为按照当前Batch的最大输入长度分配空间，计算完一个batch之后计算下一个batch，造成空间浪费。

Continuous Batch只需要在每次运行前根据当前阶段动态组Batch，申请当前阶段需要用的显存即可，可以及时调度起来新请求，实现尽可能大的并行度。

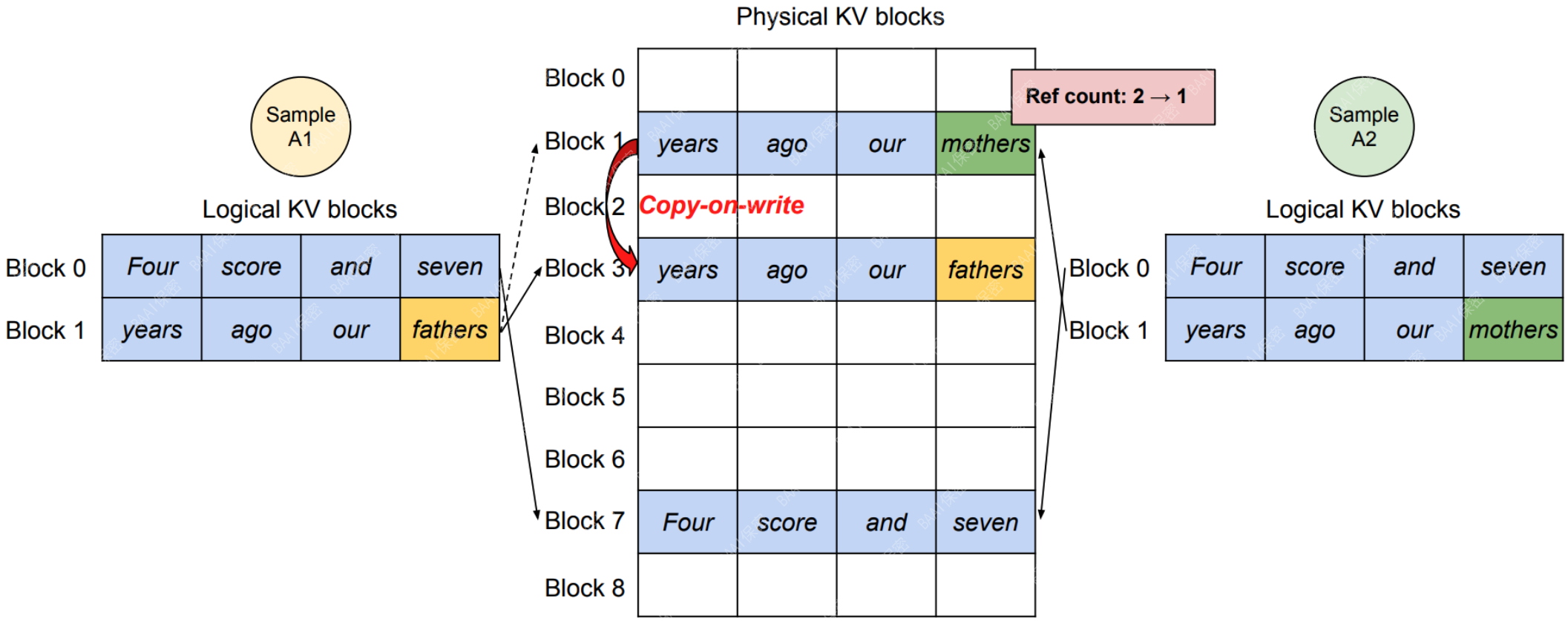
大模型推理优化技术 — PagedAttention



source: https://arxiv.org/pdf/2309.06180

- 接受一条请求，按照设定好的block size划分逻辑块给当前请求，划分完之后映射到物理块上。
- 通过Block Table记录逻辑块到物理块的映射。
- 对每个请求来说，当前请求的KV Cache在逻辑块上是连续的，但是在实际存放的物理块上可以是不连续的。

大模型推理优化技术 — PagedAttention

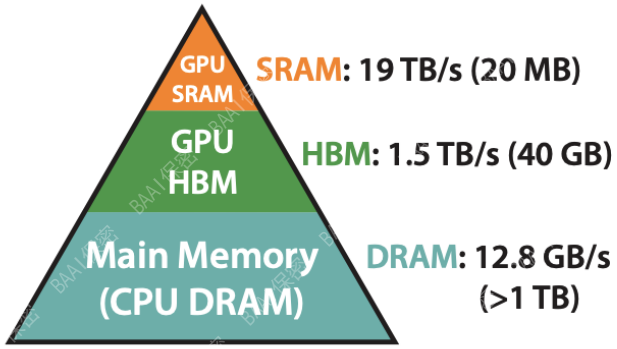


source: https://arxiv.org/pdf/2309.06180

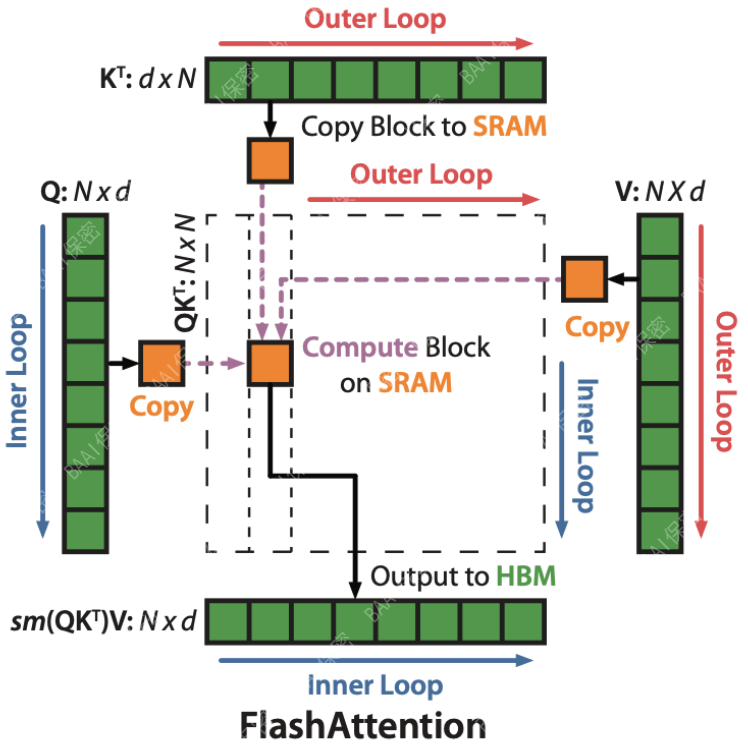
优点：

- 动态申请KV cache大小，避免显存浪费。
- 更好的进行KV Cache复用。

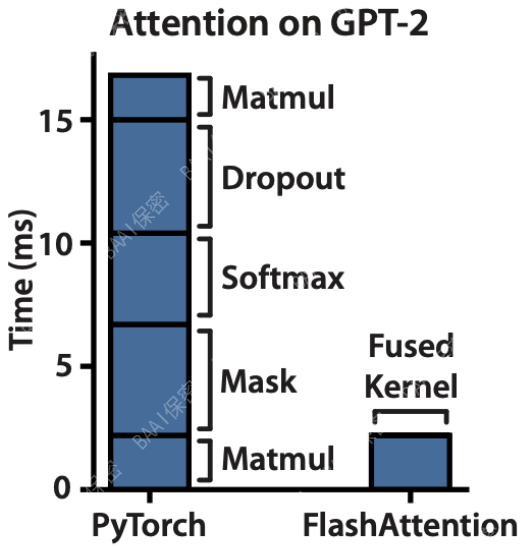
大模型推理优化技术 — FlashAttention



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention



source: https://arxiv.org/pdf/2205.14135

- 优化注意力机制的另一种方法是修改某些计算的顺序，以更好地利用 GPU 的内存层次结构。在实际计算中把多个层融合在一起可以最大限度地减少 GPU 读取和写入内存的次数。
- 分块计算是矩阵乘优化中非常重要的策略，Flash attention利用分块技术每次计算输出矩阵中的一小块。
- Attention计算中，softmax算子需要全局信息，对attention计算真正的难点是对于softmax的分块计算。

大模型推理优化技术 — FlashAttention

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

显存需求从O(N^2)降低到O(N)

Algorithm 1 FlashAttention

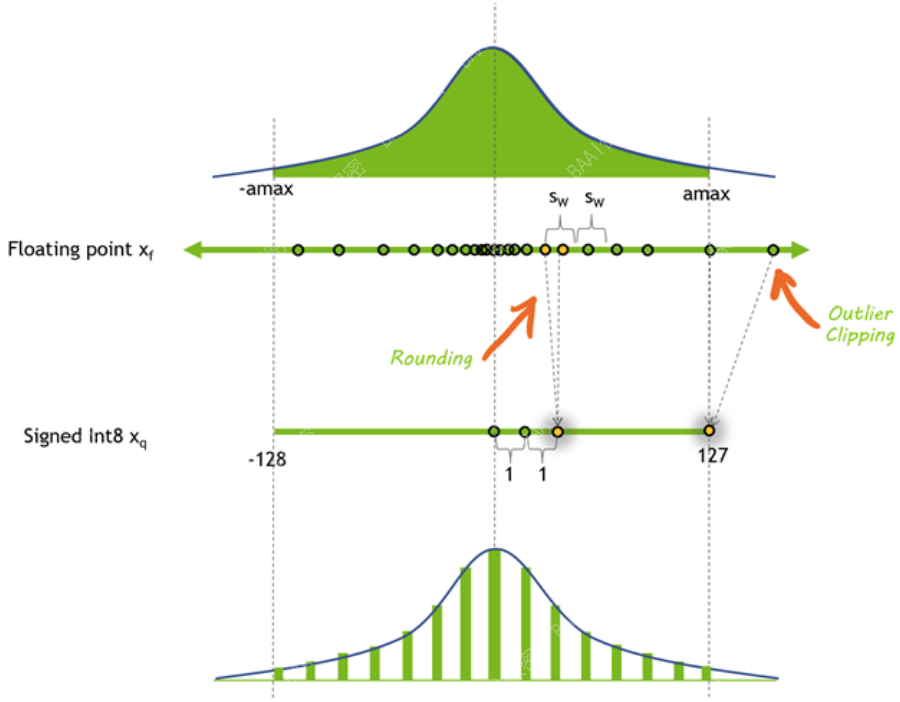
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min\left(\lceil \frac{M}{4d} \rceil, d\right)$.
2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
3: Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
4: Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ of size B_r each.
5: **for** $1 \le j \le T_c$ **do**
6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
7: **for** $1 \le i \le T_r$ **do**
8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
10: On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
11: On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
12: Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
13: Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
14: **end for**
15: **end for**
16: Return $\mathbf{O}$.

source: https://arxiv.org/pdf/2205.14135

PART 3：大模型压缩推理优化

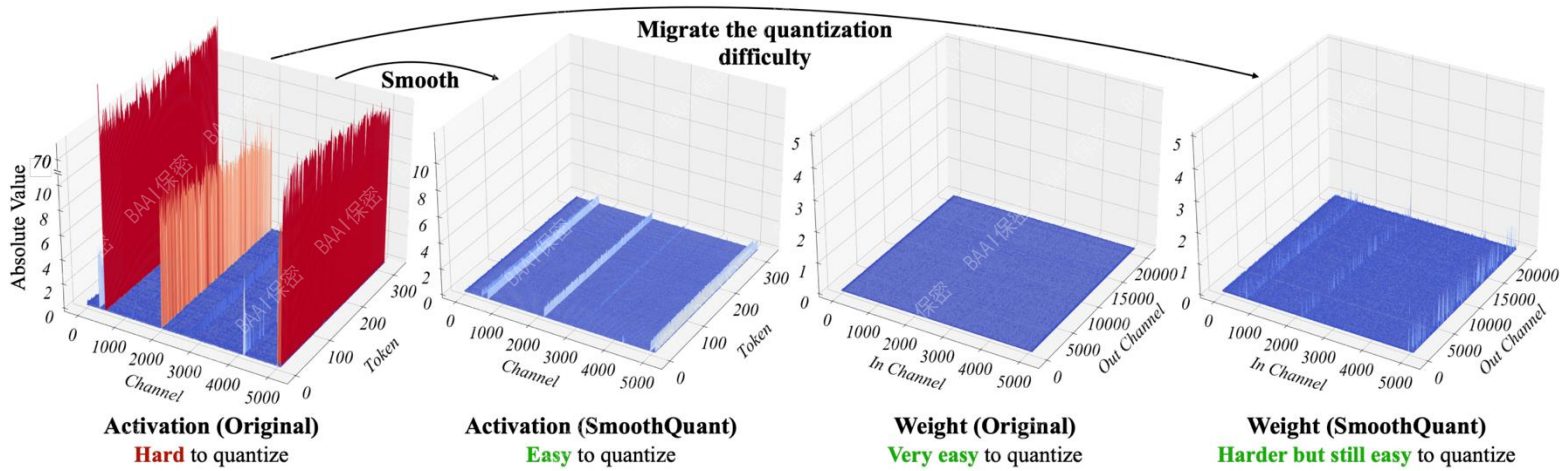
大模型压缩推理优化 — 量化



source: https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

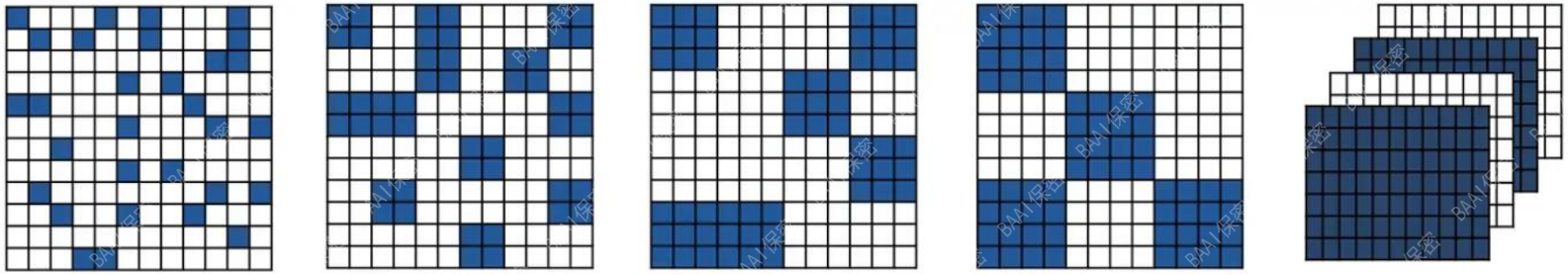
- 量化：
  把模型权重和激活从高精度映射到低精度，降低显存占用并加快计算。

- 大模型量化难点：
  大模型的激活向量通常包含异常值，增加了数据的动态范围，让低比特量化变得更具挑战性。
  找出异常值的位置，保持高精度计算。
  把激活异常值量化压力转移到权重上。



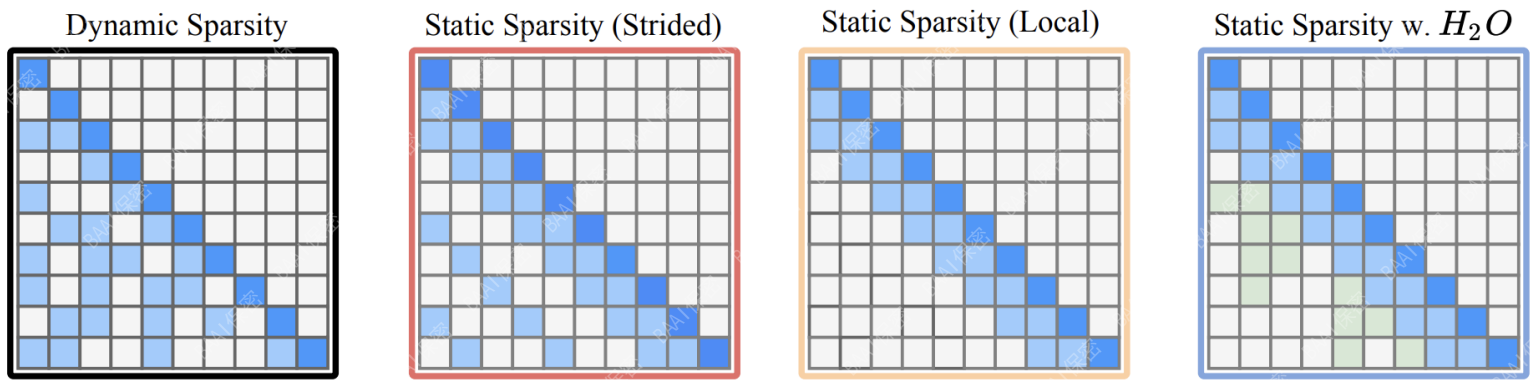
source: https://arxiv.org/pdf/2211.10438

大模型压缩推理优化 — 剪枝



source: https://zhuanlan.zhihu.com/p/622519997

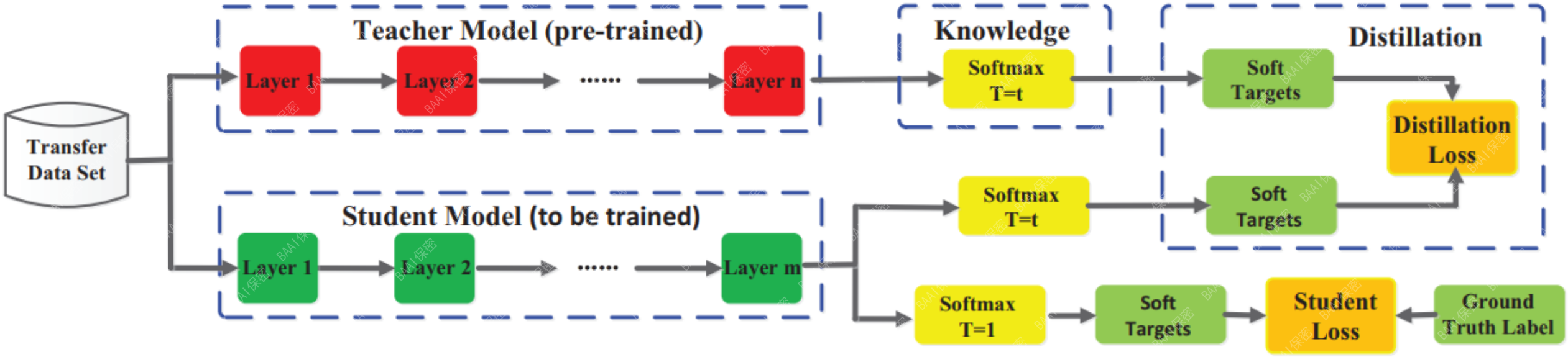
权重剪枝：把张量按照不同维度进行裁剪，降低模型参数量，并减少实际进行计算的张量大小。从左到右稀疏粒度越来越粗。最左边为非结构化稀疏，需要推理库甚至底层硬件特殊支持，最右边为结构化稀疏，仅需要在模型配置上进行修改即可。



针对KV cache不同的剪枝策略。

source:https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf

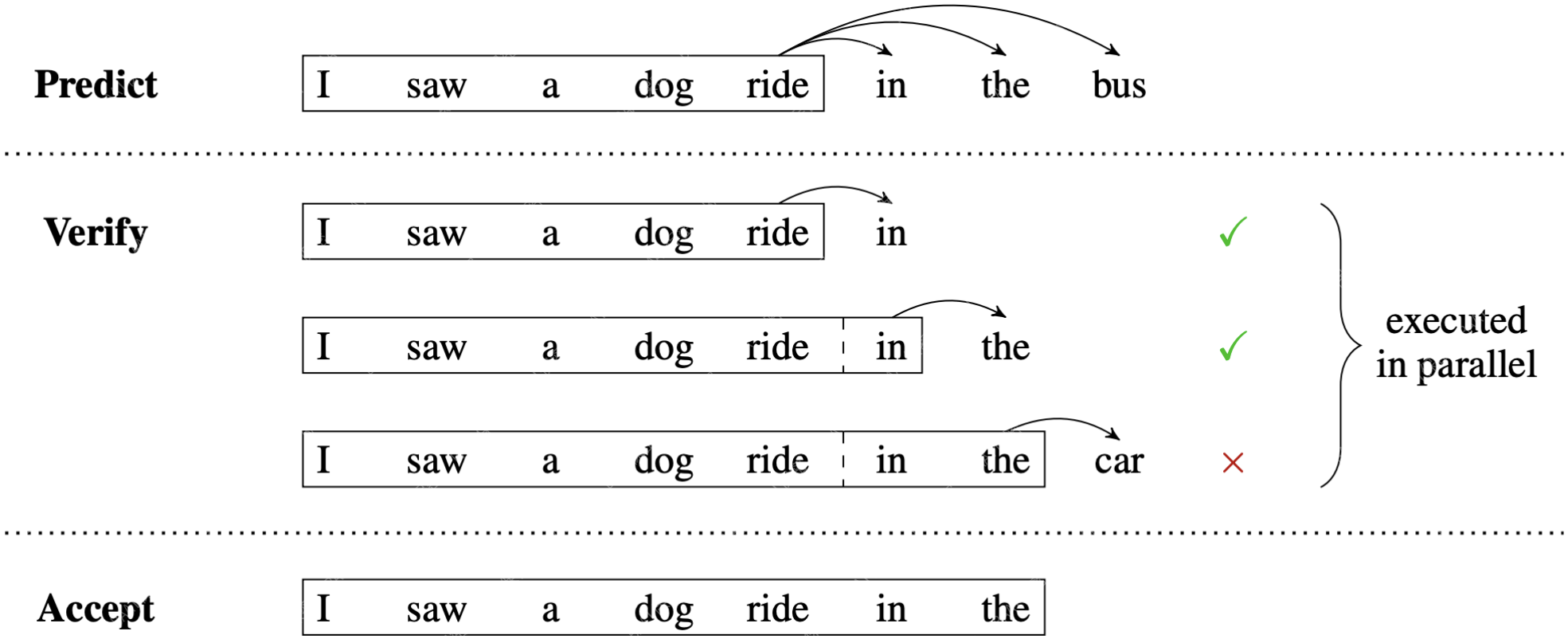
大模型压缩推理优化 — 蒸馏



source: https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

蒸馏：把教师模型中的知识迁移到学生模型中。

大模型压缩推理优化 — 投机采样



source: https://arxiv.org/pdf/1811.03115

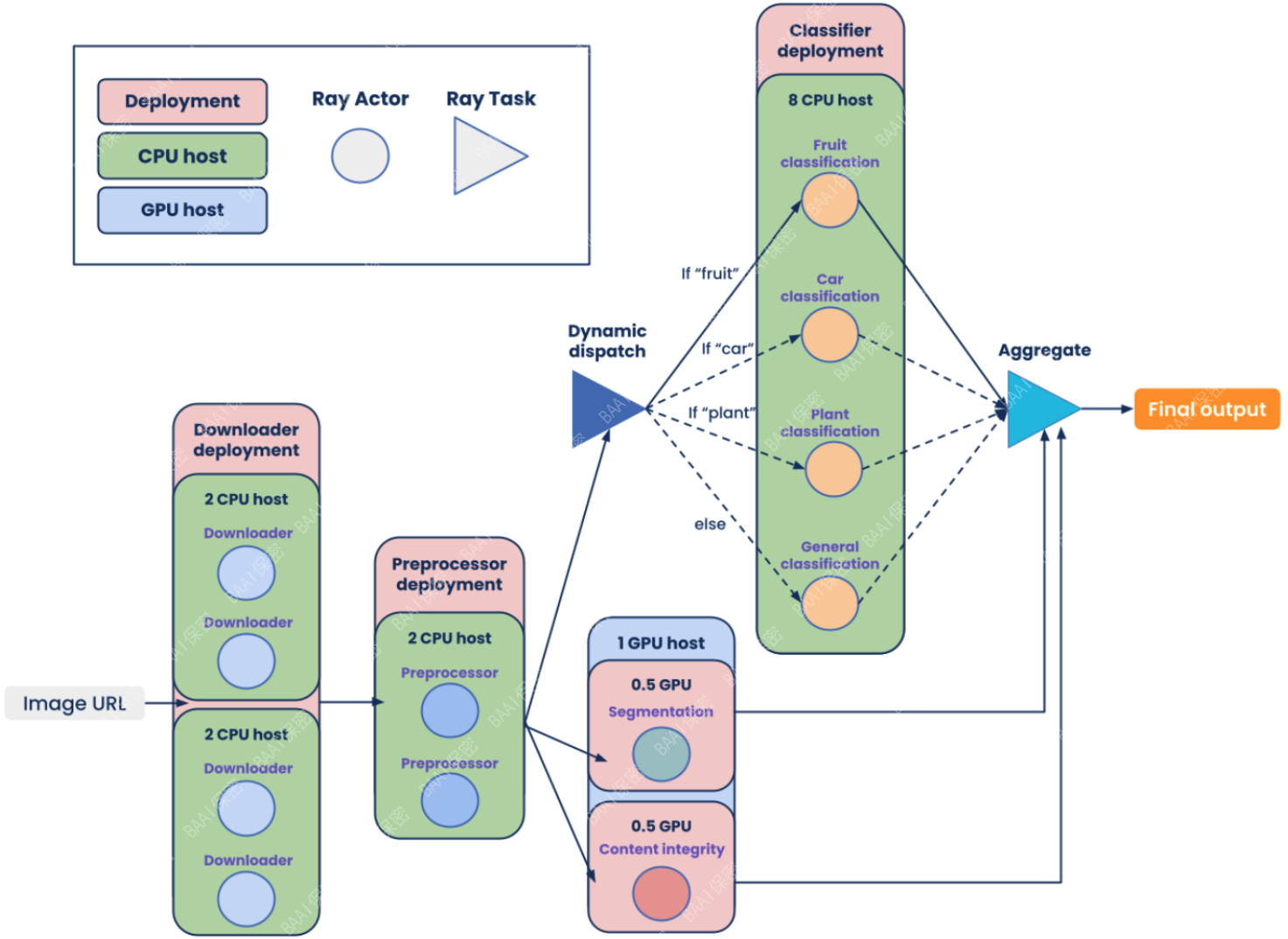
投机采样：使用一个小的草稿模型预测了多个token，大模型只需要验证这些token是被接受还是拒绝即可，可以并行对多个token进行验证。

PART 4： FlagScale部署框架

FlagScale部署框架

部署阶段可以将模型编排更细粒度的子模块，有助于更精细化，可复用的，高效的资源管理和调度。

每个模块内部可以执行不同的优化策略。



source: https://www.anyscale.com/blog/multi-model-composition-with-ray-serve-deployment-graphs#example:-scalable-image-processing-pipeline-for-content-understanding

FlagScale部署框架

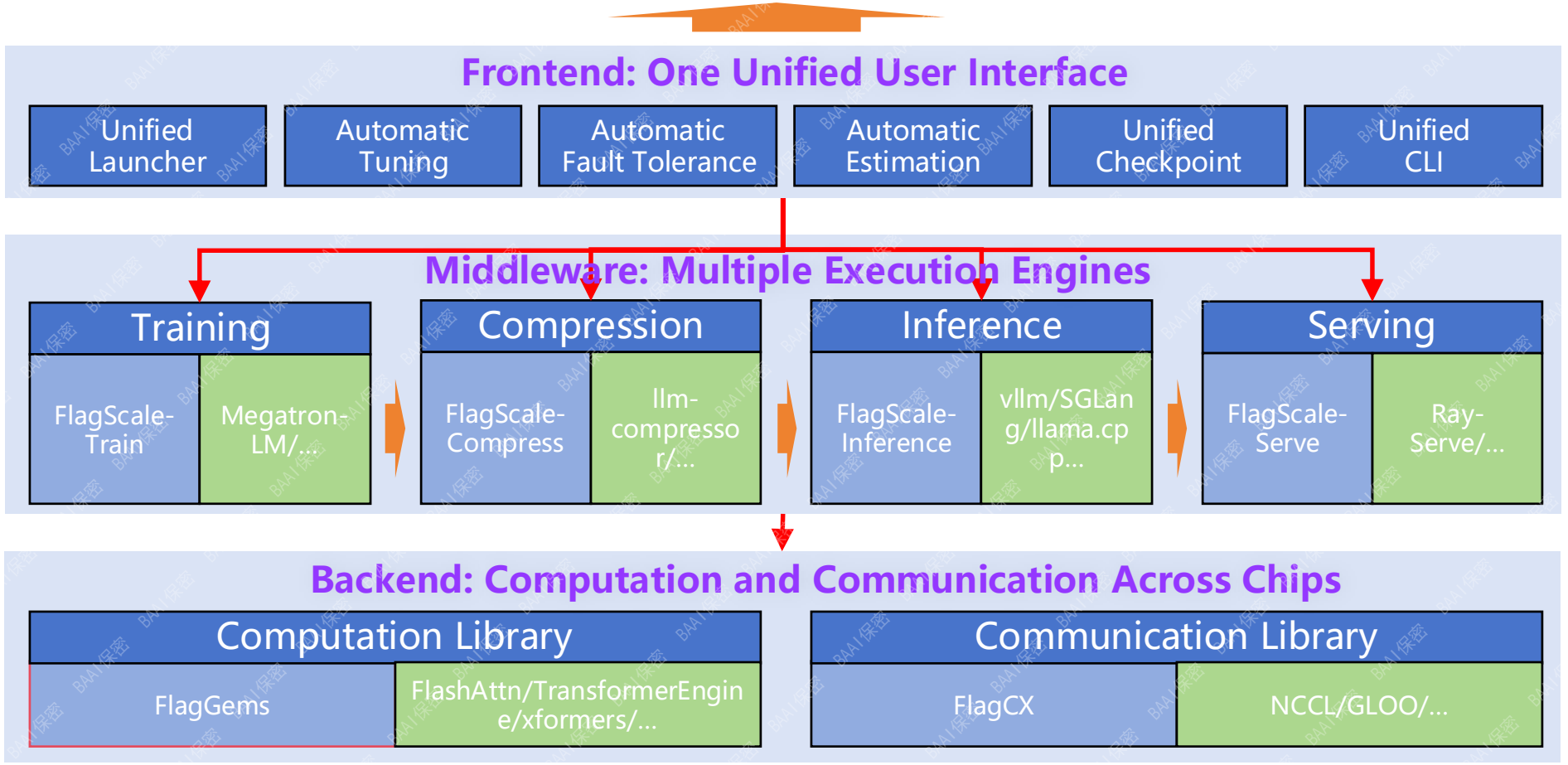

```yaml
defaults:
- _self_
- serve: 7b

experiment:
  exp_name: qwen2.5_7b
  exp_dir: outputs/${experiment.exp_name}
  task:
    type: serve
  runner:
    hostfile: null
    deploy:
      use_fs_serve: false
  envs:
    CUDA_VISIBLE_DEVICES: 0
    CUDA_DEVICE_MAX_CONNECTIONS: 1

action: run

hydra:
  run:
    dir: ${experiment.exp_dir}/hydra
```

FlagScale serving多芯片一键部署流程

```yaml
- serve_id: vllm_model
  engine: vllm
  engine_args:
    model: /models/Qwen2.5-7B-Instruct
    host: 0.0.0.0
    tensor_parallel_size: 1
    pipeline_parallel_size: 1
    gpu_memory_utilization: 0.9
    max_model_len: 32768
    max_num_seqs: 256
    enforce_eager: true
    trust_remote_code: true
    enable_chunked_prefill: true
```

```
python run.py --config-path ./examples/qwen/conf --config-name serve action=run
```

FlagScale框架



Accelerated by a variety of AI chips from multiple vendors

FlagScale统一训推框架

谢谢聆听，
欢迎合作与建议。

https://github.com/FlagOpen/FlagScale