THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RELAXED SYSTEM LAB

Generative Inference Optimization

COMP6211J

Binhang Yuan

RELAXED
SYSTEM LAB

Generative Inference

Recall Generative Inference Workflow

RELAXED SYSTEM LAB

- State-of-the-art implementation splits the computation to two phrases:
  - Prefill phase: the model takes a prompt sequence as input and engages in the generation of a key-value cache (KV cache) for each Transformer layer.
  - Decode phase: for each decode step, the model updates the KV cache and reuses the KV to compute the output.
- Analyze the arithmetic intensity:
  - Prefill phrase: arithmetic bounded;
  - Decode phrase: memory bounded.
- Assume the computation is in fp16, the concrete analysis in next two slides.
  - L is the input sequence length;
  - D is the model dimension;
  - Multi-head attention $D = n_H \times H$; H is the head dimension; n_h is the number of heads.

Prefill Phrase

RELAXED SYSTEM LAB

No.	Computation	Input	Output	Arithmetic Intensity
1	$Q = \mathrm{X}W^Q$	$\mathrm{X} \in \mathbb{R}^{L\times D}, \mathrm{W}^Q \in \mathbb{R}^{D\times D}$	$Q \in \mathbb{R}^{L\times D}$	$\frac{L\times D^2}{L\times D+D^2+L\times D}=\frac{L\times D}{2L+D}$
2	$K = \mathrm{X}W^K$	$\mathrm{X} \in \mathbb{R}^{L\times D}, \mathrm{W}^K \in \mathbb{R}^{D\times D}$	$K \in \mathbb{R}^{L\times D}$	$\frac{L\times D^2}{L\times D+D^2+L\times D}=\frac{L\times D}{2L+D}$
3	$V = \mathrm{X}W^V$	$\mathrm{X} \in \mathbb{R}^{L\times D}, \mathrm{W}^V \in \mathbb{R}^{D\times D}$	$V \in \mathbb{R}^{L\times D}$	$\frac{L\times D^2}{L\times D+D^2+L\times D}=\frac{L\times D}{2L+D}$
4	$[Q_1, Q_2 \ldots, Q_{n_h}] = \mathrm{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L\times D}$	$Q_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$	-
5	$[K_1, K_2 \ldots, K_{n_h}] = \mathrm{Partition}_{-1}(K)$	$K \in \mathbb{R}^{L\times D}$	$K_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$	-
6	$[V_1, V_2 \ldots, V_{n_h}] = \mathrm{Partition}_{-1}(V)$	$V \in \mathbb{R}^{L\times D}$	$V_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$	-
7	$\mathrm{Score_i} = \mathrm{softmax}(\frac{Q_i K_i^T}{\sqrt{\mathrm{H}}}), i = 1, \ldots n_h$	$Q_i, K_i \in \mathbb{R}^{L\times H}$	$\mathrm{score_i} \in \mathbb{R}^{L\times L}$	$\frac{L^2\times H}{L\times H+L^2+L\times H}=\frac{L\times H}{2H+L}$
8	$Z_i = \mathrm{score}_i\, V_i, i = 1, \ldots n_h$	$\mathrm{score_i} \in \mathbb{R}^{L\times L}, V_i \in \mathbb{R}^{L\times H}$	$Z_i \in \mathbb{R}^{L\times H}$	$\frac{L^2\times H}{L\times H+L^2+L\times H}=\frac{L\times H}{2H+L}$
9	$Z = \mathrm{Merge}_{-1}([Z_1, Z_2 \ldots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$	$Z \in \mathbb{R}^{L\times D}$	-
10	$\mathrm{Out} = ZW^O$	$Z \in \mathbb{R}^{L\times D}, \mathrm{W}^O \in \mathbb{R}^{D\times D}$	$\mathrm{Out} \in \mathbb{R}^{L\times D}$	$\frac{L\times D^2}{L\times D+D^2+L\times D}=\frac{L\times D}{2L+D}$
11	$A = \mathrm{Out}\, W^1$	$\mathrm{Out} \in \mathbb{R}^{L\times D}, \mathrm{W}^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{L\times 4D}$	$\frac{4L\times D^2}{L\times D+4D^2+L\times 4D}=\frac{4L\times D}{5L+4D}$
12	$A' = \mathrm{relu}(A)$	$A \in \mathbb{R}^{L\times 4D}$	$A' \in \mathbb{R}^{L\times 4D}$	-
13	$\mathrm{X}' = A'W^2$	$A' \in \mathbb{R}^{L\times 4D}, \mathrm{W}^2 \in \mathbb{R}^{4D\times D}$	$\mathrm{X}' \in \mathbb{R}^{L\times D}$	$\frac{4L\times D^2}{L\times D+4D^2+L\times 4D}=\frac{4L\times D}{5L+4D}$

Decoding Phrase

RELAXED SYSTEM LAB

No	Computationt	Input	Output	Arithmetic Intensity
1	$Q = Q_d = TW^Q$	$T \in \mathbb{R}^{1\times D}$, $W^Q \in \mathbb{R}^{D\times D}$	$Q, Q_d \in \mathbb{R}^{1\times D}$	$\frac{1\times D^2}{1\times D+D^2+1\times D}=\frac{D}{2+D}$
2	$K_d = TW^K$	$T \in \mathbb{R}^{1\times D}$, $W^K \in \mathbb{R}^{D\times D}$	$K_d \in \mathbb{R}^{1\times D}$	$\frac{1\times D^2}{1\times D+D^2+1\times D}=\frac{D}{2+D}$
3	$K = \text{concat}(K_{\text{cache}}, K_d)$	$K_{\text{cache}} \in \mathbb{R}^{L\times D}, K_d \in \mathbb{R}^{1\times D}$	$K \in \mathbb{R}^{(L+1)\times D}$	-
4	$V_d = tW^V$	$t \in \mathbb{R}^{1\times D}$, $W^V \in \mathbb{R}^{D\times D}$	$V_d \in \mathbb{R}^{1\times D}$	$\frac{1\times D^2}{1\times D+D^2+1\times D}=\frac{D}{2+D}$
5	$V = \text{concat}(V_{\text{cache}}, V_d)$	$V_{\text{cache}} \in \mathbb{R}^{L\times D}, V_d \in \mathbb{R}^{1\times D}$	$V \in \mathbb{R}^{(L+1)\times D}$	-
6	$[Q_1, Q_2 \dots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{1\times D}$	$Q_i \in \mathbb{R}^{1\times H}, i = 1, \dots n_h$	-
7	$[K_1, K_2 \dots, K_{n_h}] = \text{Partition}_{-1}(K)$	$K \in \mathbb{R}^{(L+1)\times D}$	$K_i \in \mathbb{R}^{(L+1)\times H}, i = 1, \dots n_h$	-
8	$[V_1, V_2 \dots, V_{n_h}] = \text{Partition}_{-1}(V)$	$V \in \mathbb{R}^{(L+1)\times D}$	$V_i \in \mathbb{R}^{(L+1)\times H}, i = 1, \dots n_h$	-
9	$\text{Score}_i = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{H}}), i = 1, \dots n_h$	$Q_i \in \mathbb{R}^{1\times H}, K_i \in \mathbb{R}^{(L+1)\times H}$	$\text{score}_i \in \mathbb{R}^{1\times(L+1)}$	$\frac{1\times(L+1)\times H}{1\times H+(L+1)\times H+L+1}=\frac{H+LH}{2H+L+LH+1}$
10	$Z_i = \text{score}_i\, V_i, i = 1, \dots n_h$	$\text{score}_i \in \mathbb{R}^{1\times(L+1)}, V_i \in \mathbb{R}^{(L+1)\times H}$	$Z_i \in \mathbb{R}^{1\times H}$	$\frac{(L+1)\times H}{L+1+(L+1)\times H+1\times H}=\frac{H+LH}{2H+L+LH+1}$
11	$Z = \text{Merge}_{-1}([Z_1, Z_2 \dots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{1\times H}, i = 1, \dots n_h$	$Z \in \mathbb{R}^{1\times D}$	-
12	$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{1\times D}$, $W^O \in \mathbb{R}^{D\times D}$	$\text{Out} \in \mathbb{R}^{1\times D}$	$\frac{1\times D^2}{1\times D+D^2+1\times D}=\frac{D}{2+D}$
13	$A = \text{Out}\, W^1$	$\text{Out} \in \mathbb{R}^{1\times D}$, $W^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{1\times 4D}$	$\frac{1\times 4D^2}{1\times D+4D^2+1\times 4D}=\frac{4D}{5+4D}$
14	$A' = \text{relu}(A)$	$A \in \mathbb{R}^{1\times 4D}$	$A' \in \mathbb{R}^{1\times 4D}$	-
15	$T' = A'W^2$	$A' \in \mathbb{R}^{1\times 4D}$, $W^2 \in \mathbb{R}^{4D\times D}$	$T' \in \mathbb{R}^{1\times D}$	$\frac{1\times 4D^2}{1\times 4D+4D^2+1\times D}=\frac{4D}{5+4D}$

Algorithm Optimization

RELAXED
SYSTEM LAB

- Algorithm optimization:
  - "Slightly" change the original computation at its bottleneck to make it run much faster.
- Decrease the I/O volume:
  - Model compression, i.e., quantization;
  - KV cache optimization;
  - Knowledge distillation.

RELAXED
SYSTEM LAB

Model Quantization

Quantization

RELAXED SYSTEM LAB

- Quantization transforms the floating-point values in original LLMs into integers or other discrete forms to reduce memory requirements and computational complexity.
  - int4/int8 takes 4/8 bits;
  - GPU also has a higher throughput for int4/int8 computation compared with fp16.
- How do we get the quantized model?
  - Post-Training Quantization (PTQ).
  - Quantization-Aware Training (QAT).

Preliminary of Quantization

RELAXED SYSTEM LAB

- How to quantize a float point number?
- **Absolute maximum (absmax) quantization**:
  - The original number is divided by the absolute maximum value of the tensor and multiplied by a scaling factor (127) to map inputs into the range [-127, 127].

$$\boldsymbol{X}_{\text{quant}} = \text{round}\left(\frac{127}{\max|\boldsymbol{X}|} \cdot \boldsymbol{X}\right)$$

  - To retrieve the original FP16 values, the INT8 number is divided by the quantization factor.

$$\boldsymbol{X}_{\text{dequant}} = \frac{\max|X|}{127} \cdot \boldsymbol{X}_{\text{quant}}$$

  - For example, suppose we have an absolution maximum value of 3.2. A weight of 0.1 would be quantized to round$(0.1 \times 127/3.2) = 4$. If we want to dequantize it, we would get $4 \times 3.2/127 = 0.1008$, which implies an error of 0.008.

Preliminary of Quantization

RELAXED SYSTEM LAB

- **Zero-point quantization**:
  - The input values are first scaled by the total range of values (255) divided by the difference between the maximum and minimum values. This distribution is then shifted by the zero-point to map it into the range [-128, 127]. First, we calculate the scale factor and the zero-point value:

$$\text{scale} = \frac{255}{\max(\boldsymbol{X}) - \min(\boldsymbol{X})}$$

$$\text{zeropoint} = -\text{round}(\text{scale} \cdot \min(\boldsymbol{X})) - 128$$

  - We can use these variables to quantize:

$$\boldsymbol{X}_{\text{quant}} = \text{round}(\text{scale} \cdot \boldsymbol{X} + \text{zeropoint})$$

  - We can use these variables to dequantize.

$$\boldsymbol{X}_{\text{dequant}} = \frac{\boldsymbol{X}_{\text{quant}} - \text{zeropoint}}{\text{scale}}$$

  - For example, suppose we have a maximum value of 3.2 and a minimum value of -3.0. We can calculate the scale is 255/(3.2 + 3.0) = 41.13 and the zero-point -round(41.13 × -3.0) - 128 = 123 -128 = -5, so our previous weight of 0.1 would be quantized to round(41.13 × 0.1 -5) = -1.

Post-Training Quantization (PTQ)

RELAXED SYSTEM LAB

- Post-training quantization (PTQ) quantizes a pre-trained model using moderate resources, such as a calibration dataset and a few hours of computation.
- PTQ reduces the computational precision of model weights and even activations into either INT8 or INT4 by using custom CUDA kernels for efficiency benefits.
- PTQ is an effective method for enhancing the efficiency of LLMs. It offers a straightforward solution that avoids major modifications or extensive additional training.
- Quantization targets:
  - Weight-only quantization;
  - Weights and activations quantization.
- It is necessary to acknowledge that PTQ can lead to a certain degree of precision loss due to the quantization process.

PTQ – Weight-only Quantization

RELAXED SYSTEM LAB

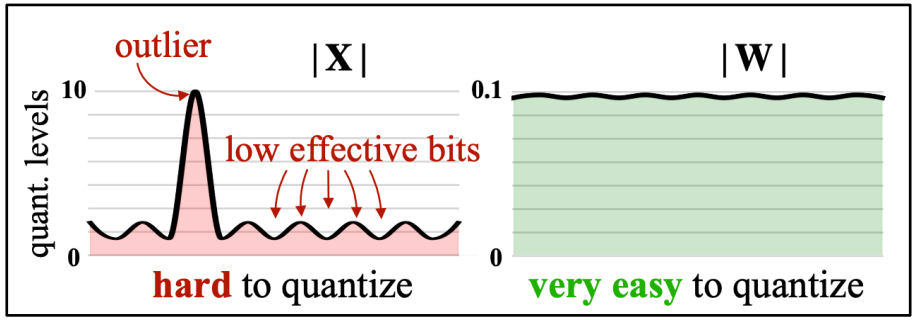
- There are many different methods for weight-only quantization.
- An example approach is **GPTQ** [https://arxiv.org/pdf/2210.17323v1.pdf]
  - GPTQ adopts a mixed int4/fp16 quantization scheme where weights are quantized as int4 while activations remain in float16.
  - During inference, weights are dequantized on the fly and the actual compute is performed in float16.
  - How does GPTQ tackle the quantization problem? (Not required, Details can be found in the paper)
    - **Layerwise Quantization:** aims to find quantized values that minimize the error at the output.

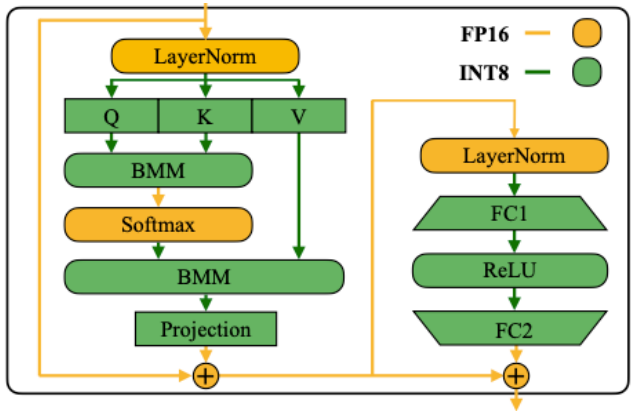
$$\text{argmin}_{\widehat{W}} \|WX - \widehat{W}X\|$$

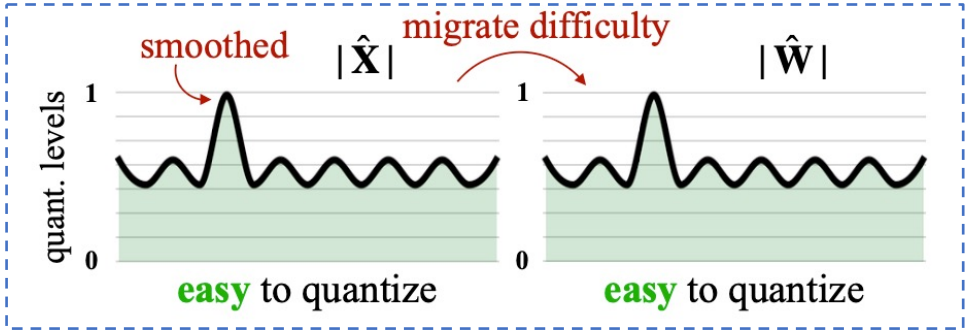
PTQ -Weights & Activations Quantization

RELAXED SYSTEM LAB

- *Why is quantizing activations difficult?*
  - Outliers make activation quantization difficult.
- An example approach is SmoothQuant [https://proceedings.mlr.press/v202/xiao23c/xiao23c.pdf]
  .
  - SmoothQuant enables 8-bit weight, 8-bit activation (W8A8) quantization.
  - All the compute-intensive operators, such as the linear layers, use int8 operations. But still not all computations are in int8.
  - How does SmoothQuant solve the problem?
    - Observation: fixed channels have outliers, and the outlier channels are persistently large.
    - Solution: smooth the outlier channels in activations by migrating their magnitudes into the following weights.



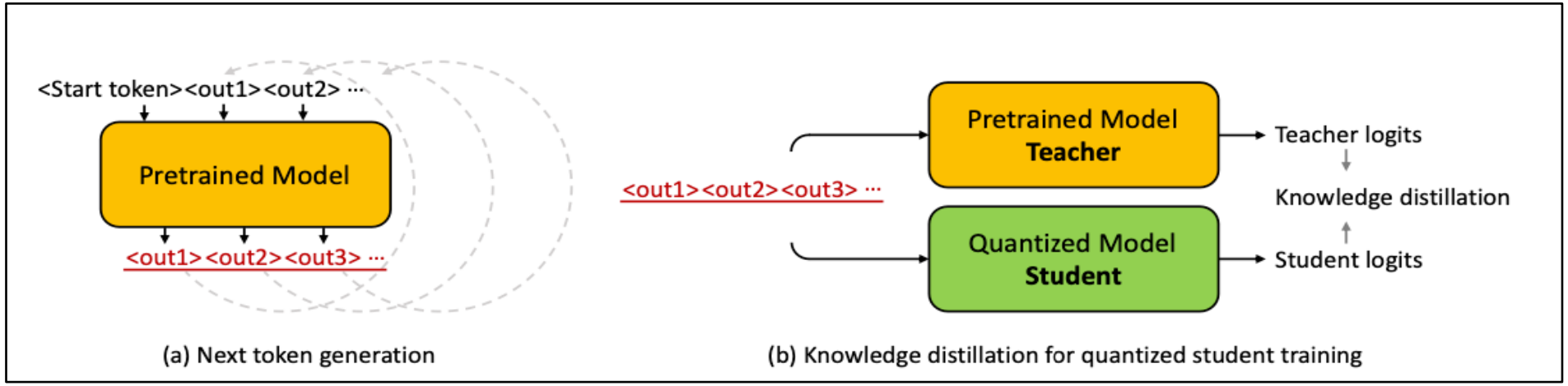




Quantization-Aware Training (QAT)

RELAXED SYSTEM LAB

- Quantization-Aware Training (QAT): the quantization process is seamlessly integrated into the training of LLMs.
- QAT can adapt to low-precision representations and thus mitigating precision loss.
- QAT require more computation than PTQ.
- An example approach is LLM-QAT [https://arxiv.org/abs/2305.17888]
  - A data-free distillation method: LLM-QAT generates data from the pretrained model with next token generation, which is sampled from top-k candidates. Then it uses the generated data as input and the teacher model prediction as label to guide quantized model finetuning.

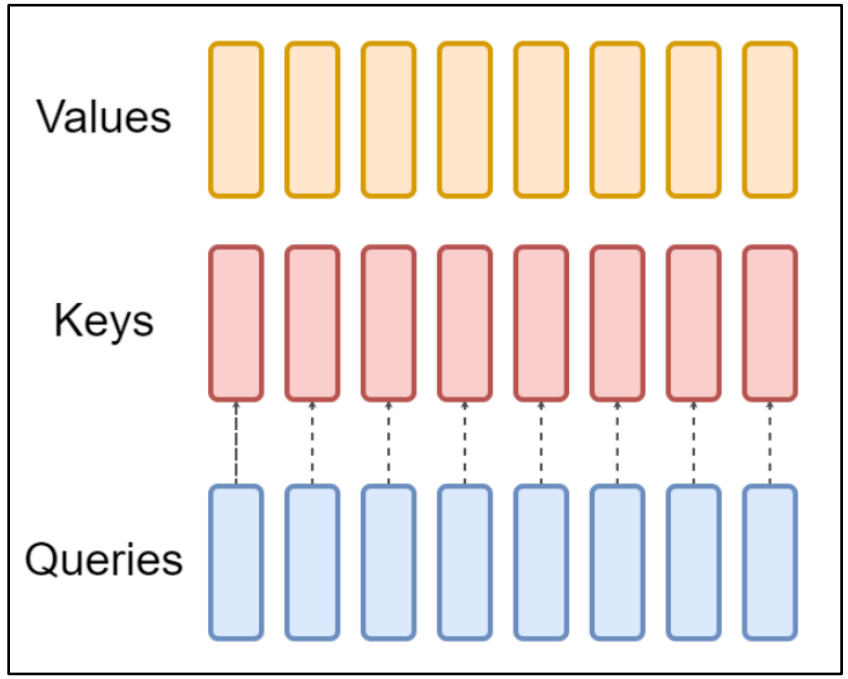


KV Cache Optimization

Memory Footprint of Inference Computation

RELAXED SYSTEM LAB

- Suppose:
  - L is the max context-length;
  - D is the model dimension;
  - Multi-head attention:
    - $D = n_H \times H$
    - H is the head dimension;
    - n_h is the number of heads.
  - Everything computation is in FP16
- For each transformer layer:
  - Model parameters:
    - $2\times(D^2 + D^2 + D^2 + D^2 + 4D^2 + 4D^2) = 24\times D^2$
  - KV cache:
    - $2\times(LD + LD) = 4LD$
  - When $L > 6D$, KV cache can be the new bottleneck.



KV Cache Optimization

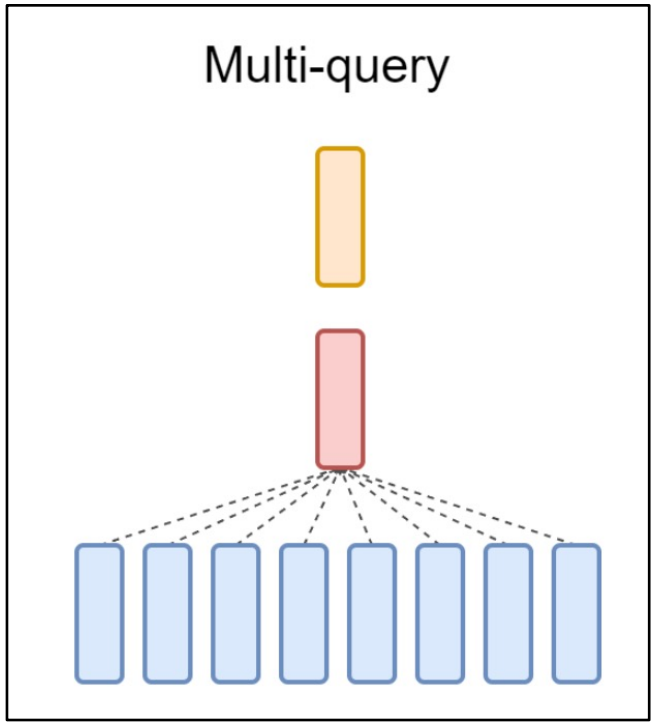
RELAXED
SYSTEM LAB

- Reduce the KV cache by modifying the multi-head attention architecture:
  - Group query attention (GQA)
- KV cache compression:
  - KV cache quantization;
  - Reserve more important tokens;
  - Sparsity at the fine granularity.

Multi-Query Attention (MQA)

RELAXED SYSTEM LAB

- The idea is simple yet effective:
  - Use multiple query heads but only a single key and value head.
- Convert MHA to MQA:
  - Initialize: the weight matrices for key and value heads are mean pooled into single weight matrices;
  - Train the model with the original recipe for a small portion of steps.



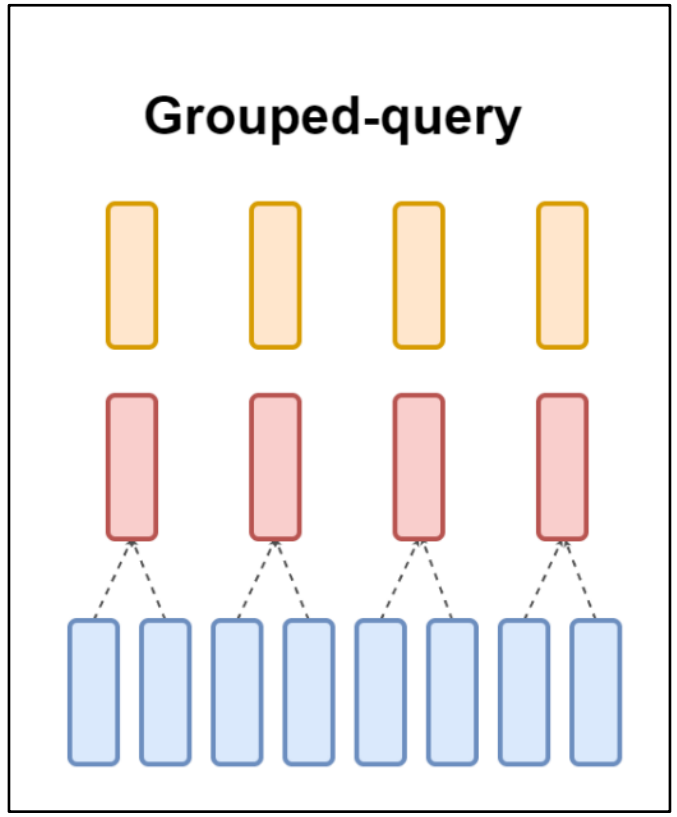
Computation	Input	Output
$Q = XW^Q$	$X \in \mathbb{R}^{L\times D}, W^Q \in \mathbb{R}^{D\times D}$	$Q \in \mathbb{R}^{L\times D}$
$K = XW^K$	$X \in \mathbb{R}^{L\times D}, W^K \in \mathbb{R}^{D\times H}$	$K \in \mathbb{R}^{L\times H}$
$V = XW^V$	$X \in \mathbb{R}^{L\times D}, W^V \in \mathbb{R}^{D\times H}$	$V \in \mathbb{R}^{L\times H}$
$[Q_1, Q_2 \dots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L\times D}$	$Q_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$\text{Score}_i = \text{softmax}(\frac{Q_i K^T}{\sqrt{H}}), i = 1, \dots n_h$	$Q_i, K \in \mathbb{R}^{L\times H}$	$\text{score}_i \in \mathbb{R}^{L\times L}$
$Z_i = \text{score}_i V, i = 1, \dots n_h$	$\text{score}_i \in \mathbb{R}^{L\times L}, V \in \mathbb{R}^{L\times H}$	$Z_i \in \mathbb{R}^{L\times H}$

Group-Query Attention (GQA)

RELAXED SYSTEM LAB

- The trade-off between MHA and MQA:
  - Divide query heads into g groups, each sharing a single key head and value head;
  - MHA: $g = n_H$; MQG: $g = 1$.
- Convert MHA to GQA:
  - Similar pooling as MQA

Computation	Input	Output
$Q = XW^Q$	$X \in \mathbb{R}^{L\times D}, \mathrm{W}^Q \in \mathbb{R}^{D\times D}$	$Q \in \mathbb{R}^{L\times D}$
$K = XW^K$	$X \in \mathbb{R}^{L\times D}, \mathrm{W}^K \in \mathbb{R}^{D\times gH}$	$K \in \mathbb{R}^{L\times gH}$
$V = XW^V$	$X \in \mathbb{R}^{L\times D}, \mathrm{W}^V \in \mathbb{R}^{D\times gH}$	$V \in \mathbb{R}^{L\times gH}$
$[Q_1, Q_2 \dots, Q_{n_h}] = \mathrm{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L\times D}$	$Q_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$[K_1, K_2 \dots, K_g] = \mathrm{Partition}_{-1}(K)$	$K \in \mathbb{R}^{L\times gH}$	$K_i \in \mathbb{R}^{L\times H}, i = 1, \dots g$
$[V_1, V_2 \dots, V_g] = \mathrm{Partition}_{-1}(V)$	$V \in \mathbb{R}^{L\times gH}$	$V_i \in \mathbb{R}^{L\times H}, i = 1, \dots g$
$\mathrm{Score_i} = \mathrm{softmax}(\frac{Q_i \mathrm{K}^T_{\lfloor i/g \rfloor}}{\sqrt{H}}), i = 1, \dots n_h$	$Q_i, K_{\lfloor i/g \rfloor} \in \mathbb{R}^{L\times H}$	$\mathrm{score_i} \in \mathbb{R}^{L\times L}$
$Z_i = \mathrm{score}_i V_{\lfloor i/g \rfloor}, i = 1, \dots n_h$	$\mathrm{score_i} \in \mathbb{R}^{L\times L}, V \in \mathbb{R}^{L\times H}$	$Z_i \in \mathbb{R}^{L\times H}$



Memory Footprint with MQA and GQA

RELAXED SYSTEM LAB

- Suppose:
  - L is the max context-length;
  - D is the model dimension;
  - Multi-head attention:
    - $D = n_H \times H$
    - H is the head dimension;
    - n_h is the number of heads.
    - g is the number of groups in GQA
  - Everything computation is in FP16

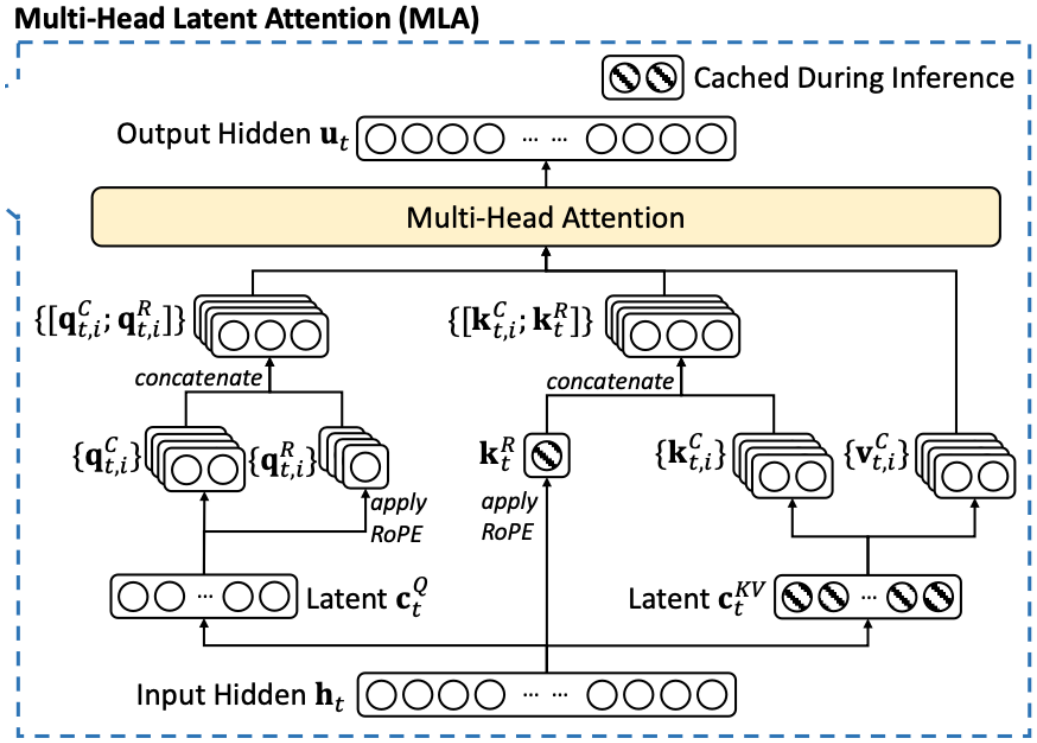
- For each transformer layer:
  - Model parameters:
    - MHA: $24D^2$
    - GQA: $2\times(D^2 + DgH + DgH + D^2 + 4D^2 + 4D^2) = 20D^2 + 4DgH$
    - MQA: $2\times(D^2 + DH + DH + D^2 + 4D^2 + 4D^2) = 20D^2 + 4DH$
  - KV cache:
    - MHA: $4LD$
    - GAQ: $2\times(LgH + LgH) = 4LgH$
    - MQA: $2\times(LH + LH) = 4LH$

DeepSeek MLA

RELAXED SYSTEM LAB

- Multi-head latent attention high level idea:
  - The basic idea of MLA is to compress the attention input $\boldsymbol{h_t} \in \mathbb{R}^d$ into a low-dimensional latent vector $\boldsymbol{c_t^{KV}} \in \mathbb{R}^{d_c}$ where d_c is much lower than the original dimension $d = d_h n_h$ that $d_c \ll d_h n_h$.
  - when we need to calculate attention, we can map this latent vector back to the high-dimensional space to recover the keys and values. As a result, only the latent vector needs to be stored, leading to significant memory reduction.
- Some advanced tricky to accommodate ROPE (extended material for this https://medium.com/data-science/deepseek-v3-explained-1-multi-head-latent-attention-ed6bee2a67c4)

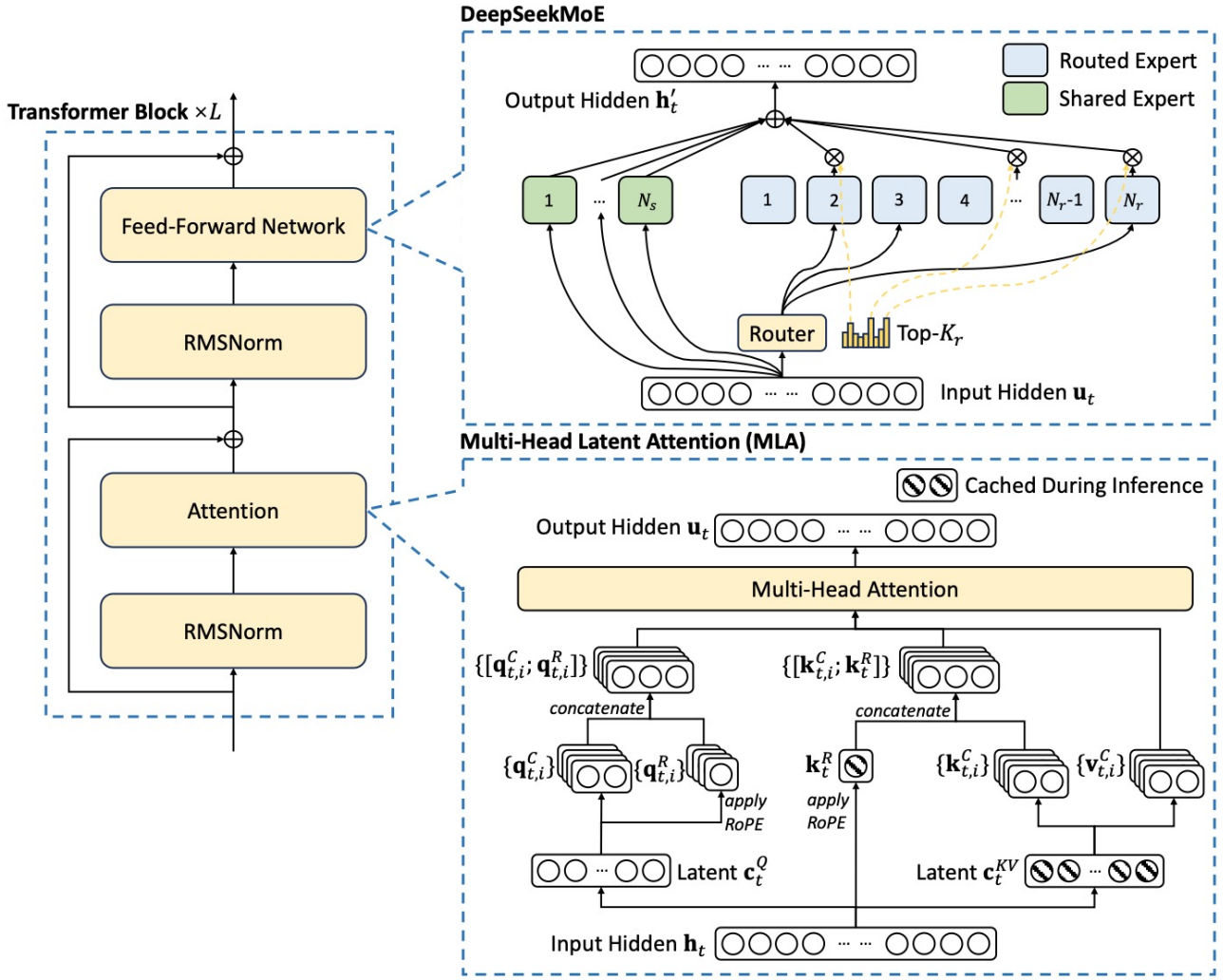
deepseek



Deepseek MLA + MoE

RELAXED SYSTEM LAB

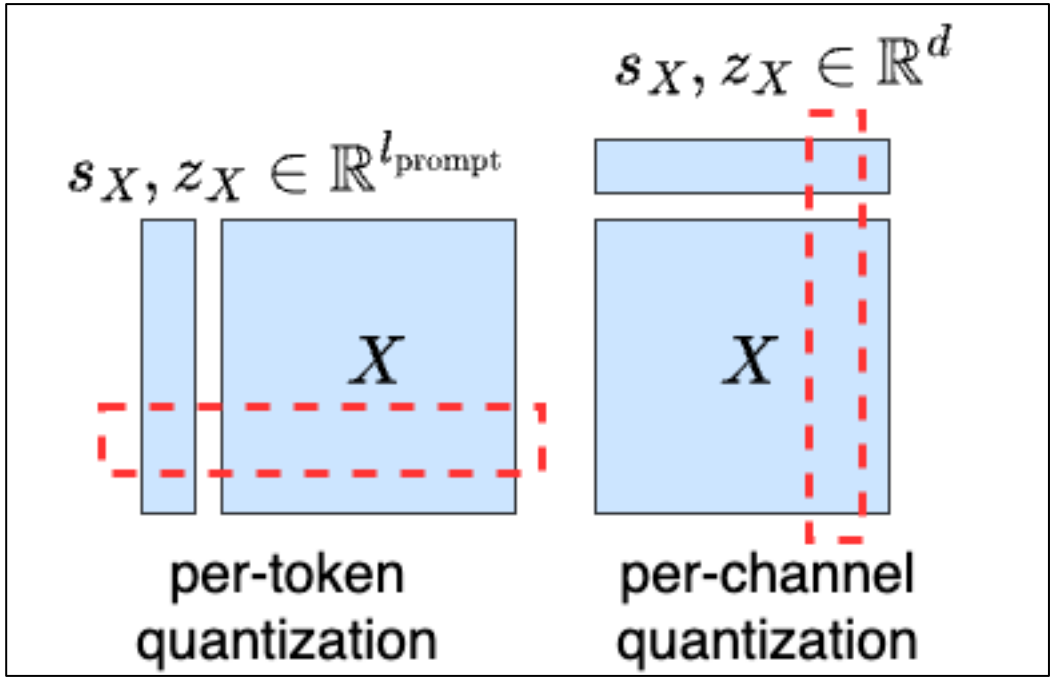
- Deepseek model advances:
  - MoE in MLP;
  - ML in attention.



KV Cache Compression – Quantization

RELAXED SYSTEM LAB

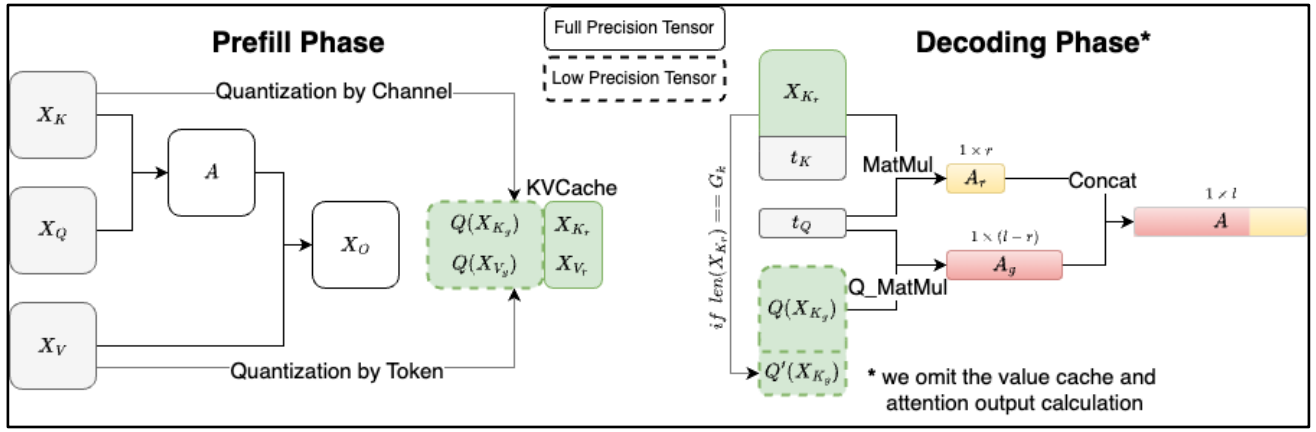
- KIVI: https://arxiv.org/abs/2402.02750
- Key observation:
  - *key cache should be quantized per-channel and value cache should be quantized per-token.*
- However, per-channel quantization, the quantization process spans across different tokens;
- Thus, this cannot be directly implemented in the streaming setting (prefill-decoding paradigm).



KV Cache Compression – Quantization

RELAXED SYSTEM LAB

- KIVI: https://arxiv.org/abs/2402.02750
- Key observation:
  - *key cache should be quantized per-channel and value cache should be quantized per-token.*
- Solution:
  - Group key cache every G tokens and quantize them separately.



Algorithm 1: The `KIVI` Prefill & Decoding Algorithm

parameter: group size G, residual length R

procedure `Prefill`:
- **Input:** $\boldsymbol{X} \in \mathbb{R}^{l_{\text{prompt}} \times d}$
- $\boldsymbol{X}_K = \boldsymbol{X}\boldsymbol{W}_K, \boldsymbol{X}_V = \boldsymbol{X}\boldsymbol{W}_V$
- $\boldsymbol{X}_{V_g} = \boldsymbol{X}_V[: l_{\text{prompt}} - R], \boldsymbol{X}_{V_r} = \boldsymbol{X}_V[l_{\text{prompt}} - R :]$
- $Q(\boldsymbol{X}_{V_g}) \leftarrow \text{GroupQuant}(\boldsymbol{X}_{V_g}, \text{dim=token}, \text{numGroup=}d//G)$
- $Q(\boldsymbol{X}_{K_g}), \boldsymbol{X}_{K_r} \leftarrow \texttt{KeyQuant}(\boldsymbol{X}_K)$
- `KV cache` $\leftarrow Q(\boldsymbol{X}_{K_g}), \boldsymbol{X}_{K_r}, Q(\boldsymbol{X}_{V_g}), \boldsymbol{X}_{V_r}$
- **return** $\boldsymbol{X}_K, \boldsymbol{X}_V$

end

procedure `Decoding`:
- **Input:** `KV cache`, $\boldsymbol{t} \in \mathbb{R}^{1 \times d}$
- $\boldsymbol{t}_Q = \boldsymbol{t}\boldsymbol{W}_Q, \boldsymbol{t}_K = \boldsymbol{t}\boldsymbol{W}_K, \boldsymbol{t}_V = \boldsymbol{t}\boldsymbol{W}_V$
- $Q(\boldsymbol{X}_{K_g}), \boldsymbol{X}_{K_r}, Q(\boldsymbol{X}_{V_g}), \boldsymbol{X}_{V_r} \leftarrow$ `KV cache`
- $\boldsymbol{X}_{K_r} \leftarrow \text{Concat}([\boldsymbol{X}_{K_r}, \boldsymbol{t}_K], \text{dim=token})$
- $\boldsymbol{X}_{V_r} \leftarrow \text{Concat}([\boldsymbol{X}_{V_r}, \boldsymbol{t}_V], \text{dim=token})$
- **if** $len(\boldsymbol{X}_{K_r}) = R$ **then**
  - $Q(\boldsymbol{X}_{K_r}), \_ \leftarrow \texttt{KeyQuant}(\boldsymbol{X}_{K_r})$
  - $Q(\boldsymbol{X}_{K_g}) \leftarrow \text{Concat}([Q(\boldsymbol{X}_{K_g}), Q(\boldsymbol{X}_{K_r})], \text{dim=token})$
  - $\boldsymbol{X}_{K_r} \leftarrow$ empty tensor.
- **end**
- **if** $len(\boldsymbol{X}_{V_r}) > R$ **then**
  - $Q(\boldsymbol{X}_{V_r'}) \leftarrow \text{GroupQuant}(\boldsymbol{X}_{V_r}[: -R], \text{dim=token}, \text{numGroup} = d//G)$
  - $Q(\boldsymbol{X}_{V_g}) \leftarrow \text{Concat}([Q(\boldsymbol{X}_{V_g}), Q(\boldsymbol{X}_{V_r'})], \text{dim=token})$
  - $\boldsymbol{X}_{V_r} \leftarrow \boldsymbol{X}_{V_r}[-R :]$
- **end**
- $\boldsymbol{A} \leftarrow \text{Concat}([\boldsymbol{t}_Q Q(\boldsymbol{X}_{K_g})^\top, \boldsymbol{t}_Q \boldsymbol{X}_{K_r}^\top], \text{dim=token})$
- $\boldsymbol{A}_g = \text{Softmax}(\boldsymbol{A})[: -R], \boldsymbol{A}_r = \text{Softmax}(\boldsymbol{A})[-R :]$
- $\boldsymbol{t}_O \leftarrow \boldsymbol{A}_g Q(\boldsymbol{X}_{V_g}) + \boldsymbol{A}_r \boldsymbol{X}_{V_r}$
- `KV cache` $\leftarrow Q(\boldsymbol{X}_{K_g}), \boldsymbol{X}_{K_r}, Q(\boldsymbol{X}_{V_g}), \boldsymbol{X}_{V_r}$
- **return** $\boldsymbol{t}_O$

end

function `KeyQuant`$(\boldsymbol{X}_K \in \mathbb{R}^{l \times d})$:
- $r = l\%R,$
- $\boldsymbol{X}_{K_g} = \boldsymbol{X}_K[: l - r], \boldsymbol{X}_{K_r} = \boldsymbol{X}_K[l - r :]$
- $Q(\boldsymbol{X}_{K_g}) \leftarrow \text{GroupQuant}(\boldsymbol{X}_{K_g}, \text{dim=channel}, \text{numGroup=}l//G)$
- **return** $Q(\boldsymbol{X}_{K_g}), \boldsymbol{X}_{K_r}$

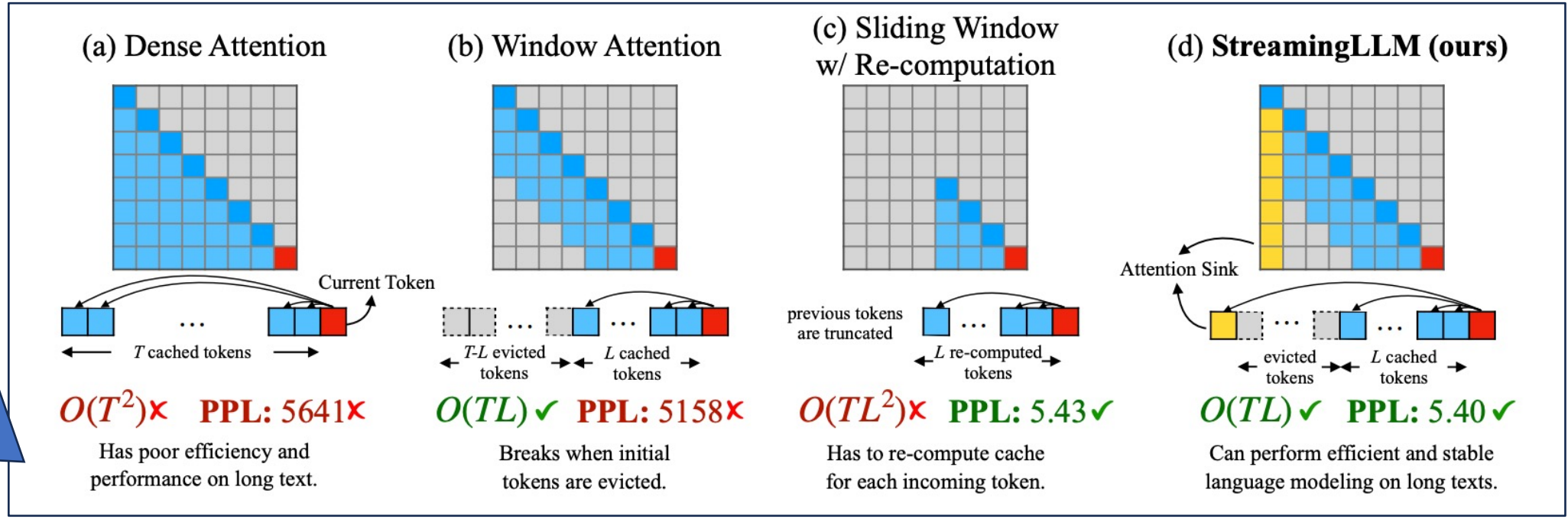
end

KV Cache Compression – Reserving More Important Tokens

RELAXED SYSTEM LAB

- StreamingLLM: https://arxiv.org/pdf/2309.17453
- Streaming based methods:

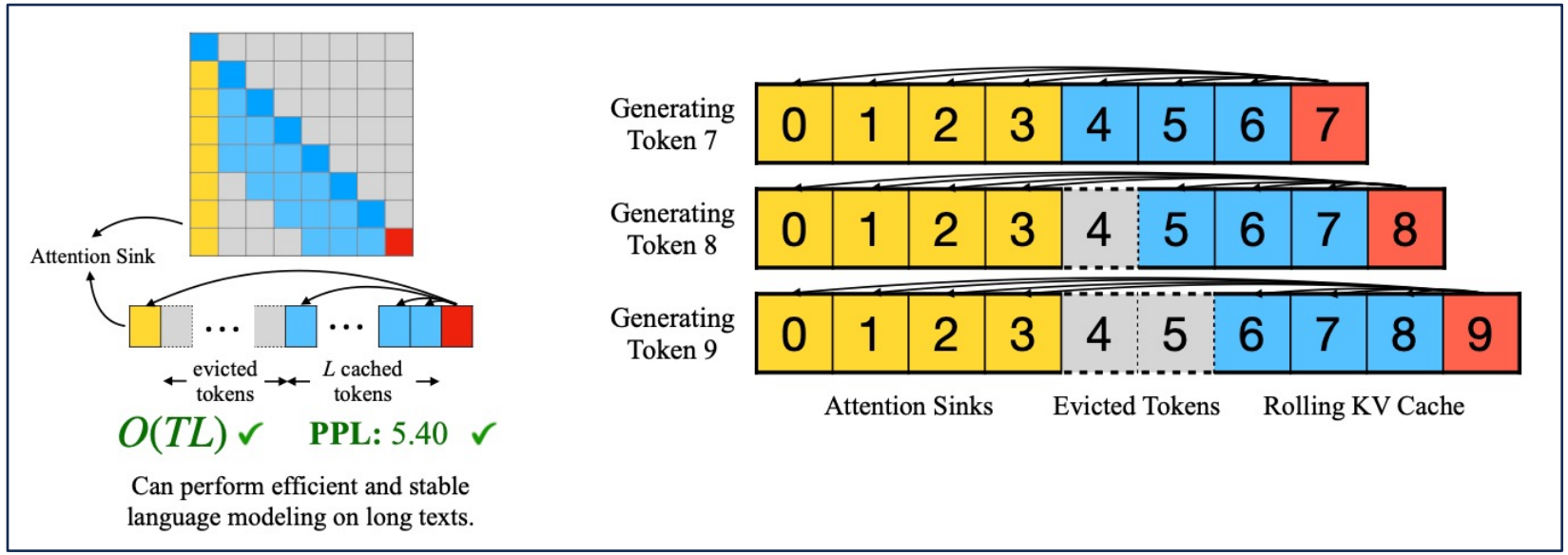
The setting here is a little different, the paper assumes the LLM is pre-trained on texts of length L, and the LLM can do generative inference with longer context T, where $T \gg L$.



KV Cache Compression – Reserving More Important Tokens

RELAXED SYSTEM LAB

- StreamingLLM: https://arxiv.org/pdf/2309.17453
- Key idea:
  - ***Attention sink***: tokens that disproportionately attract attention irrespective of their relevance.
  - Initial tokens have large attention scores, even if they're not semantically significant.
  - Initial tokens' advantage in becoming sinks due to their visibility to subsequent tokens, rooted in autoregressive language modeling.



Knowledge Distillation

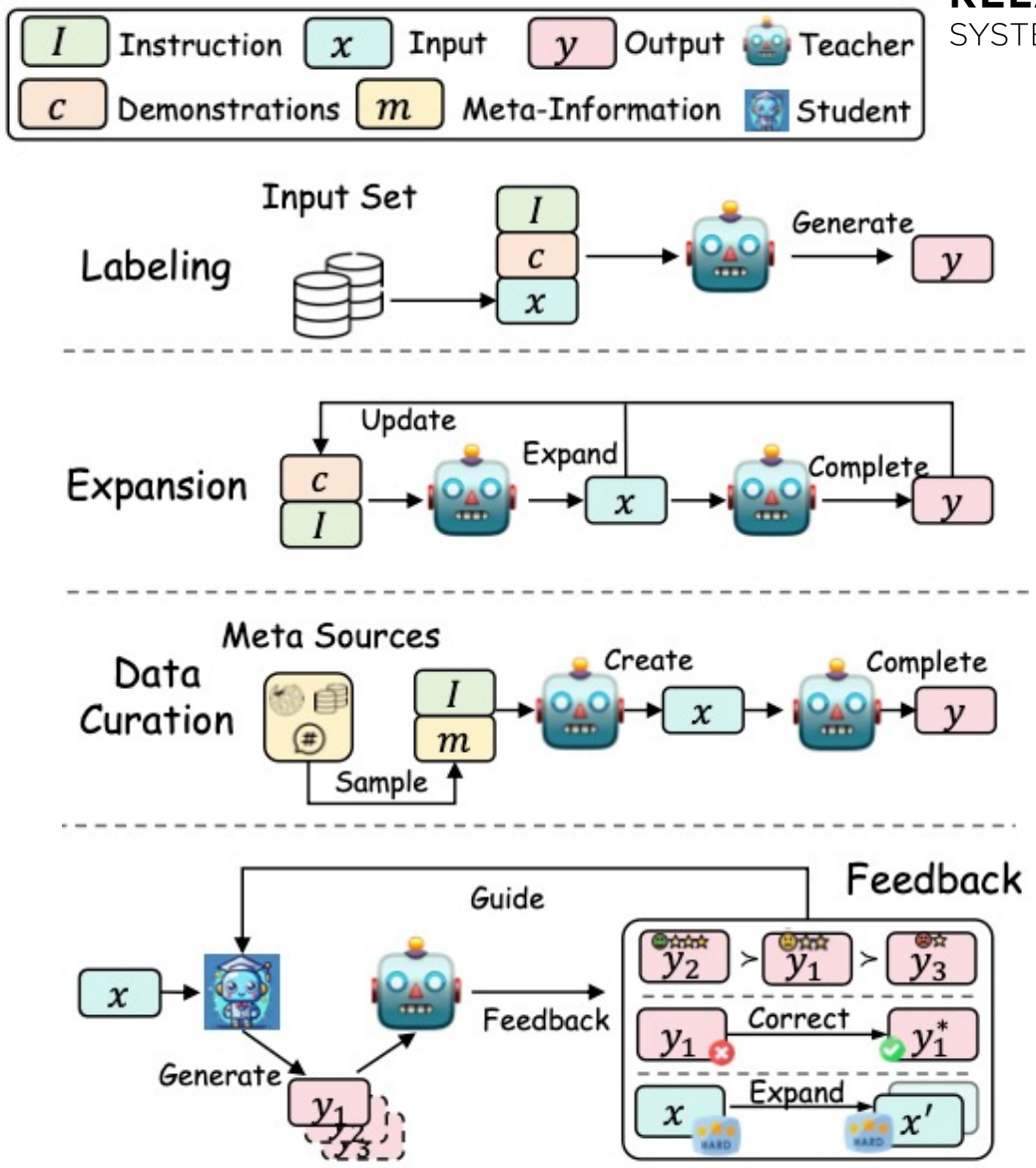
Knowledge Distillation

RELAXED
SYSTEM LAB

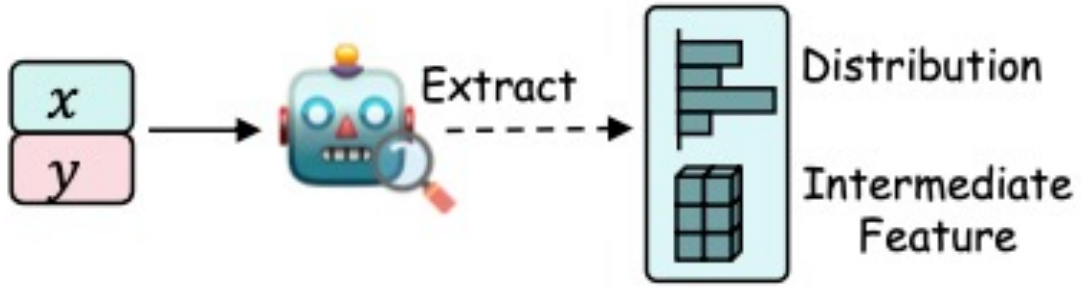
- Knowledge distillation is a technique that facilitates the transfer of capabilities from a larger model (referred to as the "*teacher model*") to a smaller model (referred to as the "*student model*").
- Knowledge distillation allows the smaller model to perform tasks with proficiency similar to the larger model but with reduced computational resources.
- Two main categories:
  - **Black-box distillation**: focuses on replicating the output behavior of the teacher model. The student model learns solely from the input-output pairings produced by the teacher without any insight into its internal operations.
  - **White-box distillation:** the internal workings and architecture of the teacher model are fully accessible and utilized during the distillation process.

Black-box Distillation

- In black-box distillation, there are different knowledge elicitation methods for teacher LLMs:
  - *Labeling*: The teacher generates the output from the input;
  - *Expansion*: The teacher generates samples similar to the given demonstrations through in-context learning;
  - *Data Curation*: The teacher synthesizes data according to meta-information, such as a topic or an entity;
  - *Feedback*: The teacher provides feedback on the student's generations, such as preferences, corrections, and expansions of challenging samples.



White-box Distillation



- White-box distillation involves leveraging the output distributions or intermediate features from teacher LLMs.
  - *Output distributions*: the teacher model labels a fixed dataset of sequences with token-level probability distributions. The KL divergence can be used as the loss function.
  - *Intermediate features*: To leverage the rich semantic and syntactic knowledge in intermediate layers of the teacher model, layer-wise distillation can align the student's hidden representations with those of the teacher at each layer.

System Optimization

RELAXED SYSTEM LAB

- System optimization:
  - Improve the system efficiency without changing the original computation;
- What is the measurement for generative inference?
  - From the user's perspective:
    - First token generation latency;
    - Generation throughput;
  - From the system's perspective:
    - Serving throughput for each model replica;
    - SLO (service-level objective):
      - For example, 90% of the requests can be accomplished within 1.5X of the time when they are running alone on the same hardware.

System Optimization

RELAXED SYSTEM LAB

- System optimization:
  - Improve the system efficiency without changing the original computation;
- General ideas
  - For each request, generate multiple tokens simultaneously:
    - Speculative decoding;
    - Parallel decoding.
  - For multiple requests, effectively batching them:
    - Continuous batching;
    - Disaggregated inference.

Speculative Decoding

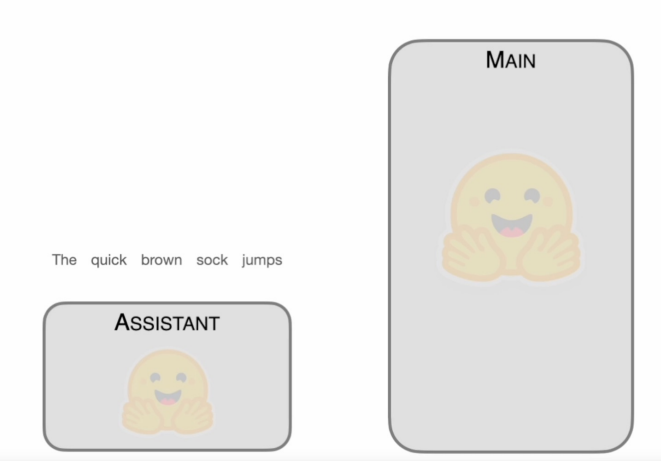
Speculative Decoding Overview

RELAXED SYSTEM LAB

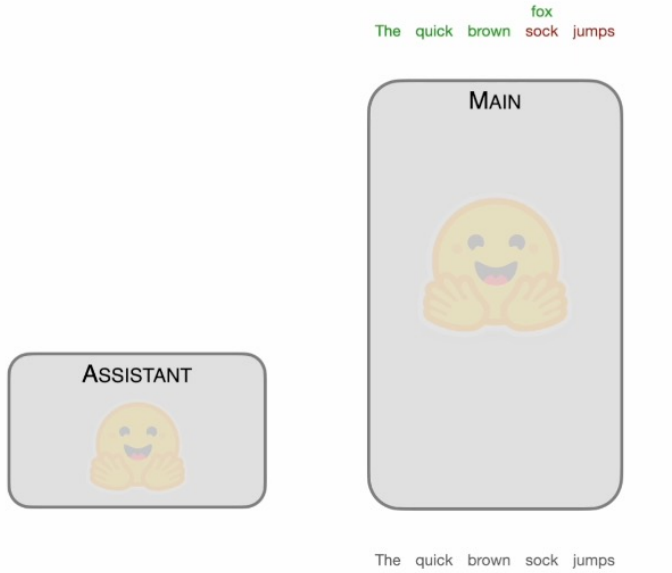
- **Observation**:
  - A small *assistant/speculative model* very often generates the same tokens as the large original LLM (sometime s referred to as the *main/target model*).
- Speculative decoding overview:
  - The assistant model auto-regressively generates a sequence of N candidate tokens;
  - The candidate tokens are passed to the original LLM to be verified. The original model takes the candidate tokens as input and performs ***a single forward pass***:
    - All candidate tokens up to the first mismatch are correct;
    - After the first mismatch, replace the first incorrect candidate token with the correct token from the main model, and discard all predicted tokens that come after this mismatched token.
  - Repeat this process until the end condition is reached.

Speculative Decoding Example

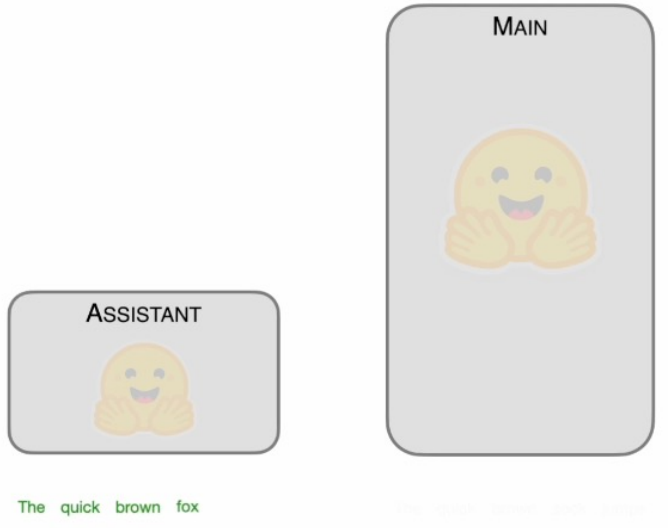
RELAXED SYSTEM LAB



1. The assistant model auto-regressively generates sequence: [The quick brown sock jumps].



2. The first three tokens predicted by the original LLM agree with those from the assistant model: [The quick brown]. However, the fourth candidate token from the assistant model (sock), mismatches with the correct token from the main model (fox).



3. We replace the first incorrect candidate token (sock) with the correct token from the main model (fox) and discard all predicted tokens that come after this. The corrected sequence, [The quick brown fox] now forms the new input to the assistant model for the next step.

Speculative Decoding Notes

RELAXED
SYSTEM LAB

- We auto-regressively generate using the fast, assistant model, and only perform verification forward passes with the slow, main model, the decoding process is sped-up substantially.
- The verification forward passes performed by the main model ensures that *exactly the same outputs are achieved* as if we were using the main model standalone.
- Trade-off:
  - "The assistant model should be significantly faster." V.S. "The assistant model should predict the same token distribution as often as possible."
  - Since 70-80% of all predicted tokens tend to be "easier" tokens, this trade-off is heavily biased towards selecting a faster model, rather than a more accurate one.

Speculative Decoding in HF inference API

RELAXED SYSTEM LAB

Example Code

```
from transformers import AutoModelForCausalLM, AutoTokenizer

prompt = "What is Apple?"

model_id = "EleutherAI/pythia-160m"
assistant_model_id = "EleutherAI/pythia-14m"

tokenizer = AutoTokenizer.from_pretrained(model_id)
inputs = tokenizer(prompt, return_tensors="pt")

model = AutoModelForCausalLM.from_pretrained(model_id)
assistant_model = AutoModelForCausalLM.from_pretrained(assistant_model_id)

outputs = model.generate(**inputs, max_new_tokens=20, assistant_model=assistant_model,
return_dict_in_generate=True)

input_length = inputs.input_ids.shape[1]
token = outputs.sequences[0, input_length+1:]
print(f"[INFO] raw token: {token}")
output = tokenizer.decode(token)
print(f"[Context]: {prompt} \n[Output]:{output}\n")
```

How to do sampling in Speculative Decoding? *

RELAXED SYSTEM LAB

- We have two distribution now:
  - $x \sim p(x)$ sampling from the distribution of original model;
  - $x \sim q(x)$ sampling from the distribution of the assistant model;
- Speculative sampling:
  - To sample $x \sim p(x)$,
  - We instead sample from $x \sim q(x)$:
    - If $q(x) \leq p(x)$, keep x;
    - Otherwise ($q(x) > p(x)$), reject the sample with probability $1 - \frac{p(x)}{q(x)}$ and sample x again from an adjusted distribution $p'(x)$.

$$p'(x) = \frac{p(x) - \min(q(x), p(x))}{\sum_{x'}(p(x') - \min(q(x'), p(x')))}$$

arXiv:2211.17192v2 [cs.LG] 18 May 2023

Fast Inference from Transformers via Speculative Decoding

Yaniv Leviathan [*1] Matan Kalman [*1] Yossi Matias [1]

Abstract

Inference from large autoregressive models like Transformers is slow - decoding K tokens takes K serial runs of the model. In this work we introduce *speculative decoding* - an algorithm to sample from autoregressive models faster *without any changes to the outputs*, by computing several tokens in parallel. At the heart of our approach lie the observations that (1) hard language-modeling tasks often include easier subtasks that can be approximated well by more efficient models, and (2) using speculative execution and a novel sampling method, we can make exact decoding from the large models faster, by running them in parallel on the outputs of the approximation models, potentially generating several tokens concurrently, and without changing the distribution. Our method can accelerate existing off-the-shelf models without retraining or architecture changes. We demonstrate it on T5-XXL and show a **2X-3X** acceleration compared to the standard T5X implementation, with identical outputs.

1. Introduction

Large autoregressive models, notably large Transformers (Vaswani et al., 2017), are much more capable than smaller models, as is evidenced countless times in recent years e.g., in the text or image domains, like GPT-3 (Brown et al., 2020), LaMDA (Thoppilan et al., 2022), Parti (Yu et al., 2022), and PaLM (Chowdhery et al., 2022). Unfortunately, a single decode step from these larger models is significantly slower than a step from their smaller counterparts, and making things worse, these steps are done serially - decoding K tokens takes K serial runs of the model.

Given the importance of large autoregressive models and specifically large Transformers, several approaches were developed to make inference from them faster. Some approaches aim to reduce the inference cost for *all* inputs equally (e.g. Hinton et al., 2015; Jaszczur et al., 2021; Hubara et al., 2016; So et al., 2021; Shazeer, 2019). Other approaches stem from the observation that not all inference steps are born alike - some require a very large model, while others can be approximated well by more efficient models. These *adaptive computation* methods (e.g. Han et al., 2021; Sukhbaatar et al., 2019; Schuster et al., 2021; Scardapane et al., 2020; Bapna et al., 2020; Elbayad et al., 2019; Schwartz et al., 2020) aim to use less compute resources for easier inference steps. While many of these solutions have proven extremely effective in practice, they usually require changing the model architecture, changing the training-procedure and re-training the models, and don't maintain identical outputs.

The key observation above, that some inference steps are "harder" and some are "easier", is also a key motivator for our work. We additionally observe that inference from large models is often not bottlenecked on arithmetic operations, but rather on memory bandwidth and communication, so additional computation resources might be available. Therefore we suggest increasing concurrency as a complementary approach to using an adaptive amount of computation. Specifically, we are able to accelerate inference without changing the model architectures, without changing the training-procedures or needing to re-train the models, and without changing the model output distribution. This is accomplished via *speculative execution*.

Speculative execution (Burton, 1985; Hennessy & Patterson, 2012) is an optimization technique, common in processors, where a task is performed in parallel to verifying if it's actually needed - the payoff being increased concurrency. A well-known example of speculative execution is branch prediction. For speculative execution to be effective, we need an efficient mechanism to suggest tasks to execute that are likely to be needed. In this work, we generalize speculative execution to the stochastic setting - where a task *might be* needed with some probability. Applying this to decoding from autoregressive models like Transformers, we sample generations from more efficient *approximation models* as speculative prefixes for the slower *target models*. With a novel sampling method, *speculative sampling*, we maximize the probability of these speculative tasks to

[*]Equal contribution [1]Google Research, Mountain View, CA, USA. Correspondence to: Yaniv Leviathan <leviathan@google.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

https://arxiv.org/pdf/2211.17192.pdf

Speculative Decoding Optimization

RELAXED SYSTEM LAB

- If the assistant model is asked to output the length m draft sequence, and the original LLM accepts n, $n < m$, the $(m - n)$ tokens are automatically discarded.
- If $n \ll m$, every LLM forward leads to only a limited number of tokens being decoded.
- Optimization, e.g., [SpecInfer https://arxiv.org/pdf/2305.09781.pdf]:
  - Let the assistant model(s) sample multiple plausible draft sequences for the original LLM to evaluate in parallel.
  - Sampling from multiple small assistant model drafts.
  - Organize the draft sequences as a tree to reuse the KV-cache. [Tree Attention]

Parallel Decoding

Parallel Decoding Overview

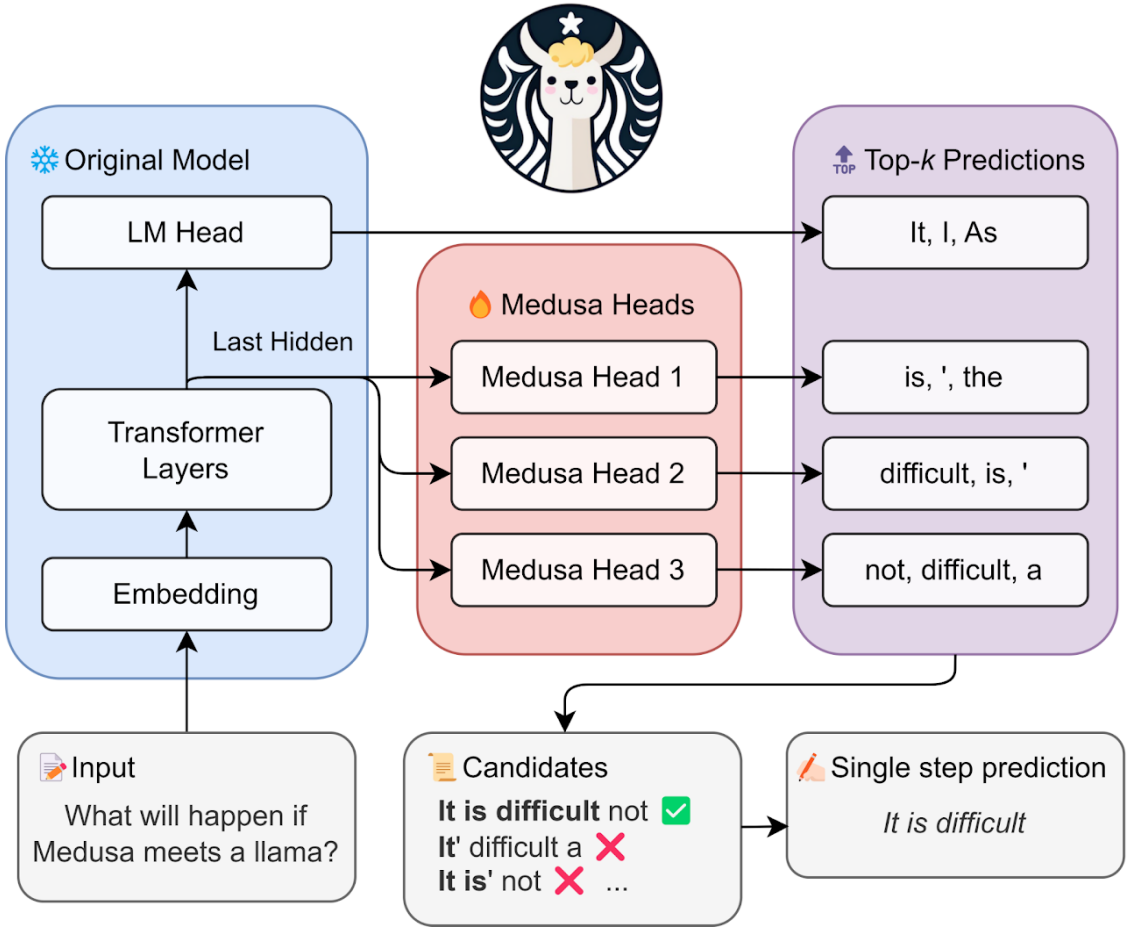
RELAXED SYSTEM LAB

- Speculative decoding generates multiple token sequences using the assistant model for verification by the original LLM.
- Parallel decoding enables multiple token predictions directly from one forward pass of the original LLM.
- How to enable this?
  - [Medusa https://arxiv.org/pdf/2401.10774.pdf, https://github.com/FasterDecoding/Medusa]
    - Medusa heads (last layer projection);
    - Tree Attention.

Medusa Heads

RELAXED SYSTEM LAB

- Rather than pulling in an entirely new assistant model to predict subsequent tokens, Medusa simply extends the original LLM itself.
- Medusa heads are the additional decoding heads built on top of the last hidden states of the LLM, enabling the prediction of several subsequent tokens in parallel.
- Each Medusa head is a single layer of feed-forward network, augmented with a residual connection.
- Training these heads is straightforward: for a relatively small dataset, the original model remains static; only the Medusa heads are updated.

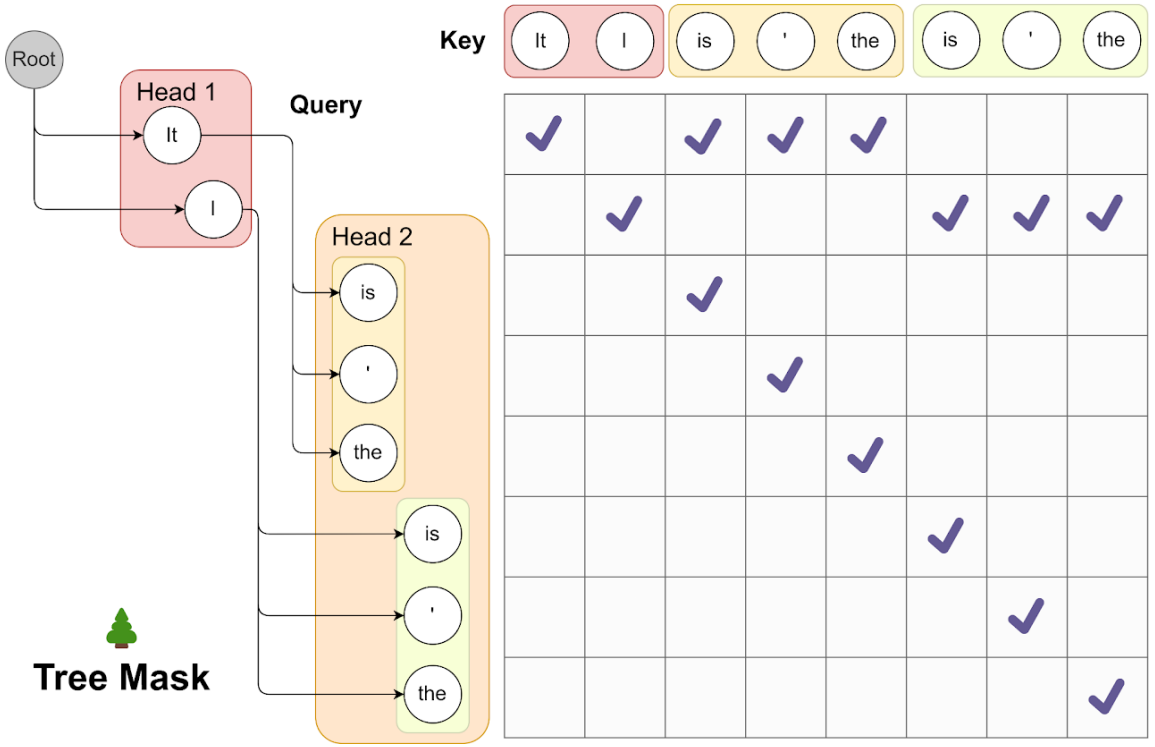


https://www.together.ai/blog/medusa

Tree Attention

RELAXED SYSTEM LAB

- Tree Attention in Medusa:
  - The top-1 accuracy for predicting the 'next-next' token hovers around 60%;
  - The top-5 accuracy soars to over 80%.
  - This substantial increase indicates that leveraging the multiple top-ranked predictions made by the Medusa heads can significantly amplify the number of tokens generated per decoding step.
  - Construct the Cartesian product of the top predictions from each Medusa head and encode the dependency graph into the attention.

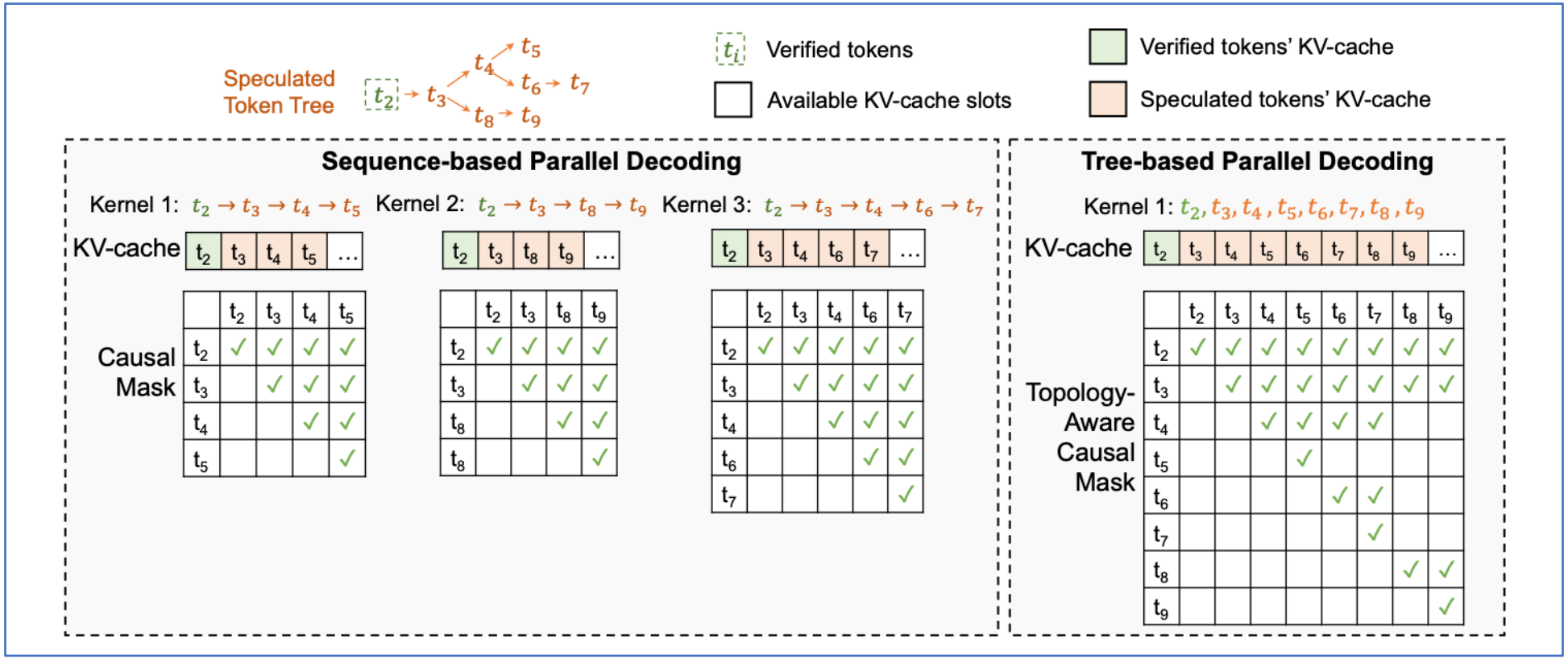


Modified from https://www.together.ai/blog/medusa to avoid confusion when compared with SpecInfer

Tree Attention

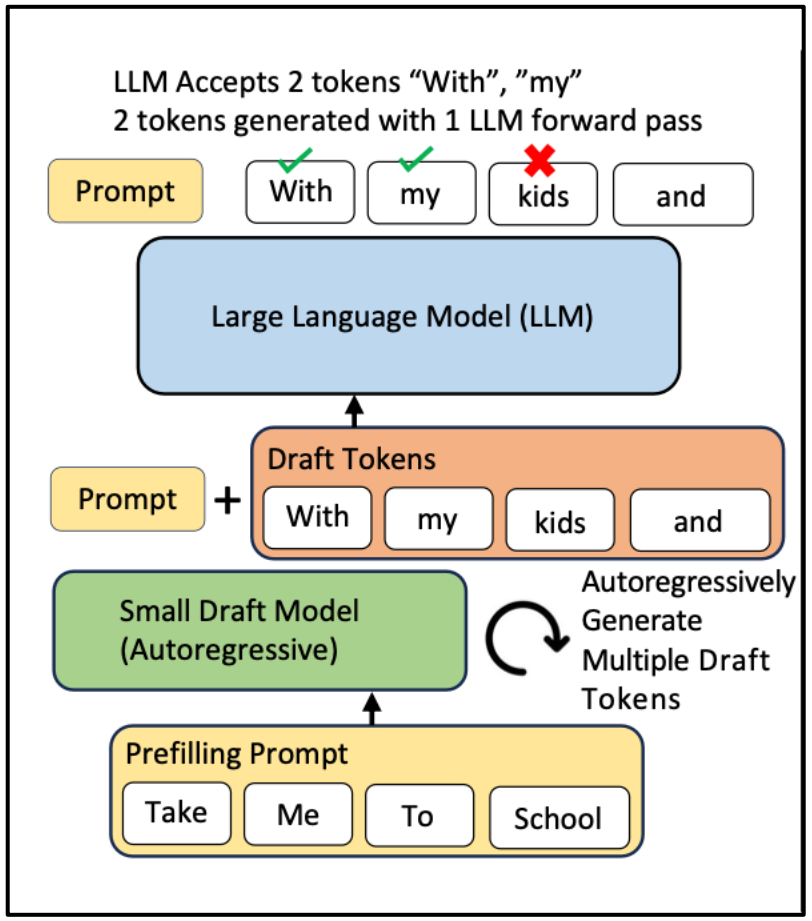
RELAXED SYSTEM LAB

- A similar idea can also be leveraged in speculative decoding when you organize multiple sequences as a prefix tree.

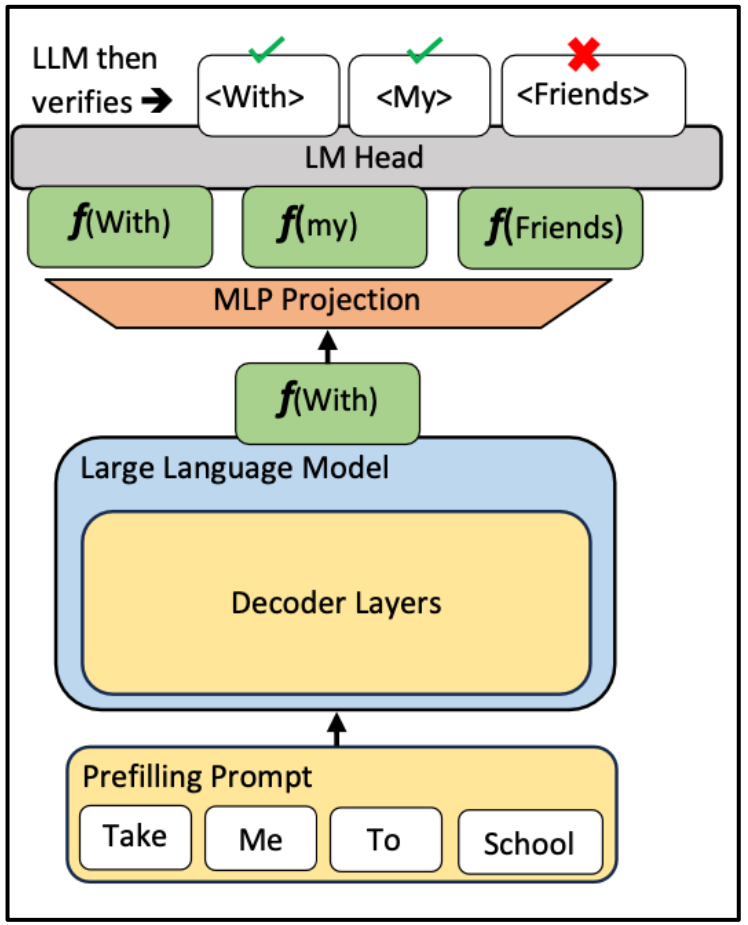


[Figure from SpecInfer.]

Speculative Decoding v.s. Parallel Decoding



Speculative Decoding



Parallel Decoding

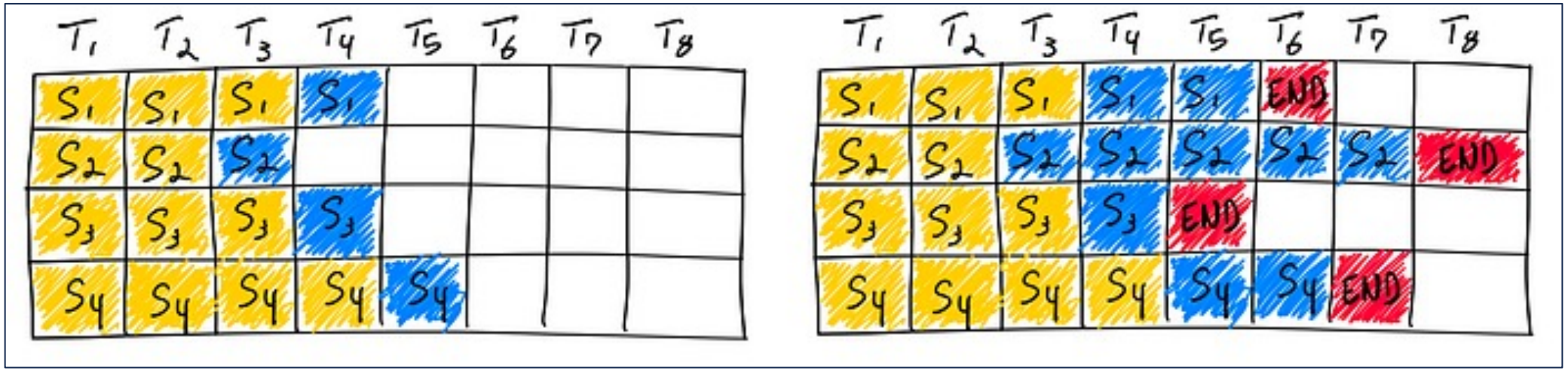
https://arxiv.org/abs/2402.16363

Continuous Batching

Navie Batching

RELAXED SYSTEM LAB

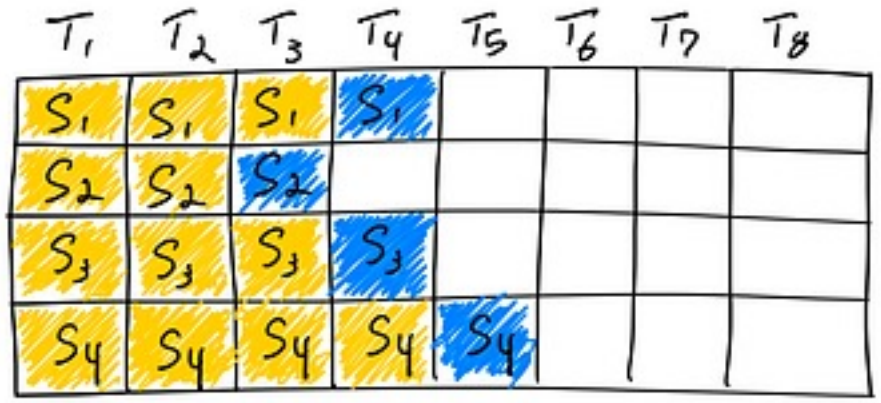
- The naïve approach is static batching:
  - The size of the batch remains constant until the inference is complete.
- Issue:
  - The inference process of an LLM is iterative.
  - Some requests may ‘end’ before the completion of the batch.
  - This means that the GPU may be underutilized.

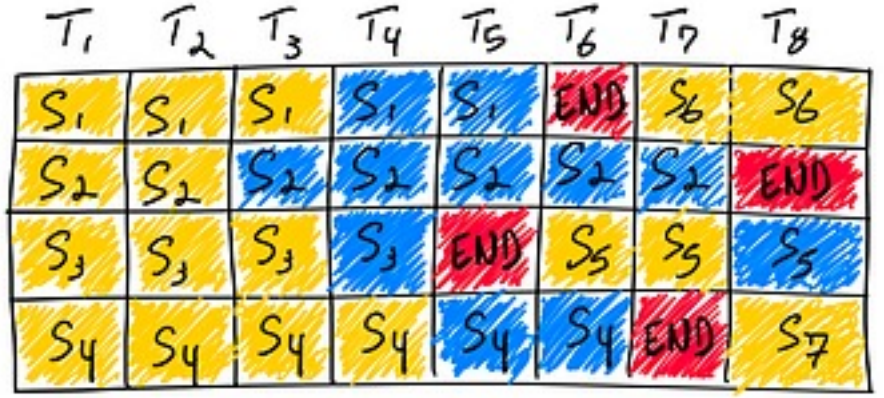


Continuous Batching

RELAXED SYSTEM LAB

- Orca: https://www.usenix.org/conference/osdi22/presentation/yu
- Key idea:
  - Instead of waiting until every sequence in a batch has completed generation, Orca implements iteration-level scheduling;
  - Once a sequence in a batch has completed generation, a new sequence can be inserted in its place;
  - Higher GPU utilization than static batching.





Disaggregated Inference

Disaggregated Inference

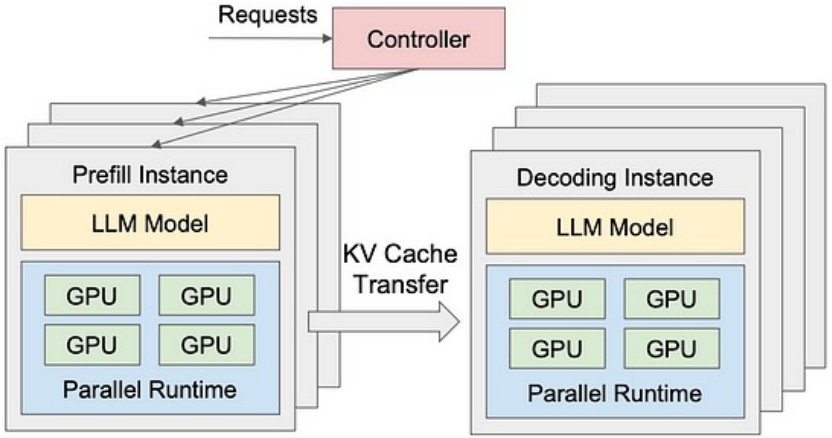
RELAXED SYSTEM LAB

- Issues with batching based approach:
  - Prefill:
    - Computation-bounded => Batching's benefit is marginal;
  - Decoding:
    - Memory-bounded => Batching's benefit is significant.
  - Prefill-decoding interference:
    - You feel some sudden stall for some output token when you are using the streaming inference API.
    - Perhaps this is when a new request's prefill step is injected to your batch for decoding.

Disaggregated Inference

RELAXED SYSTEM LAB

- DistServe: https://www.usenix.org/system/files/osdi24-zhong-yinmin.pdf
- Key ideas:
  - Prefill computation on some GPUs;
  - Decoding computation on some other GPUs;
  - Prefill and decoding instances can have different parallel configurations;
  - Dynamic configuration of the prefill / decoding ratio;
  - Overhead: KV-cache communication.



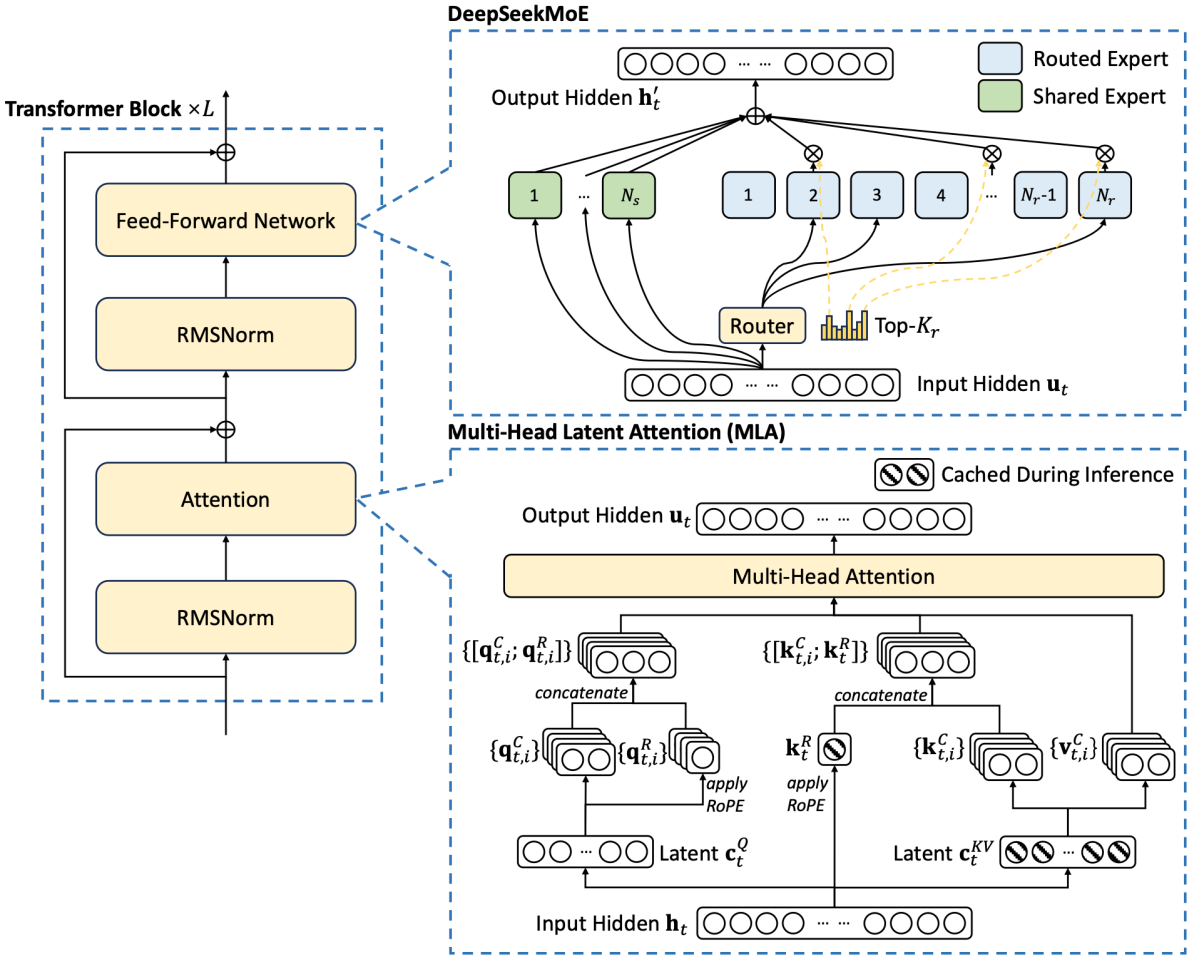
RELAXED
SYSTEM LAB

Deepseek Disaggregated Inference Service

Deepseek V3/R1 Model

RELAXED SYSTEM LAB

- Multi-Head Latent Attention (MLA):
  - Similar memory footprint with MQA;
  - Similar performance with MHA.
- Deepseek-MoE:
  - Finely segmentation;
  - Shared expert



Deepseek V3/R1 Inference Service

RELAXED SYSTEM LAB

- Minimum deployment unit of the prefilling stage configuration (4 node - 32 GPU):
  - MLA: 1-way TP & 32-way DP;
  - MoE: 32-way EP.
- Optimization:
  - Redundant MoE experts:
    - Duplicates high-load experts and deploys them redundantly;
    - The high-load experts are detected based on statistics collected during the online deployment and are adjusted periodically (e.g., every 10 minutes);
    - Set 32 redundant experts for the prefilling stage.
  - Rearrange MoE experts:
    - Rearrange experts among GPUs within a node based on the observed loads, striving to balance the load across GPUs as much as possible without increasing the cross-node all-to-all communication overhead.

Deepseek V3/R1 Inference Service

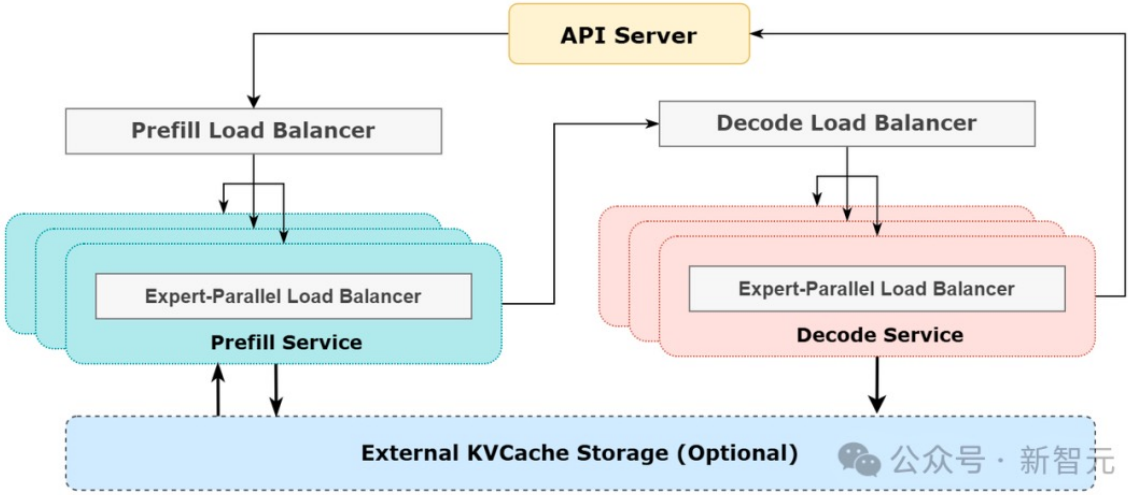
RELAXED SYSTEM LAB

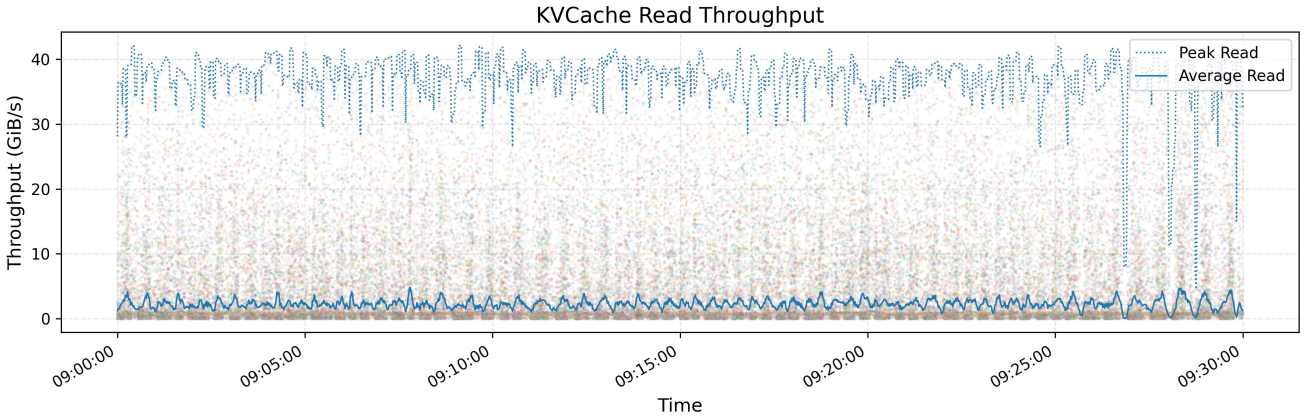
- Minimum deployment unit of the decoding stage configuration (**18 node - 144 GPU**):
  - MLA: 1-way TP & 144-way DP;
  - MoE: 144-way EP (each GPU hosts only two routed experts).
- Try to optimize the batch size per expert (usually 256 tokens), but the bottleneck is memory access rather than computation.
- Optimization:
  - NVSHMEM are leveraged for All-to-all communication of the dispatch and combine part
  - 64 GPUs are responsible for hosting redundant experts and shared experts.

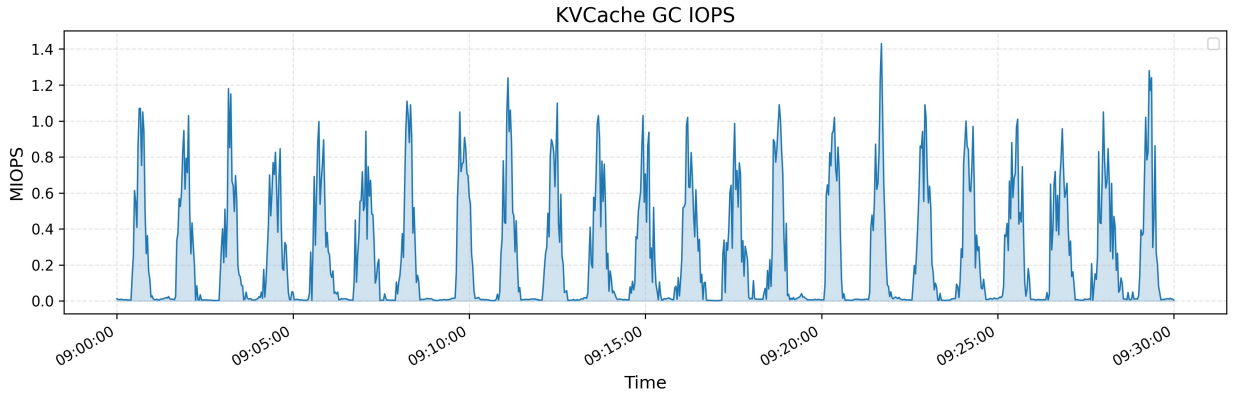
Deepseek V3/R1 Inference Service

RELAXED SYSTEM LAB

- Efficient file system for KV-cache sharing.
- Read throughput of all KVCache clients (1×400Gbps NIC/node), highlighting both peak and average values, with peak throughput reaching up to 40 GiB/s.







https://github.com/deepseek-ai/3FS

References

- https://towardsdatascience.com/introduction-to-weight-quantization-2494701b9c0c
- https://medium.com/data-science/deepseek-v3-explained-1-multi-head-latent-attention-ed6bee2a67c4
- https://arxiv.org/abs/2210.17323
- https://huggingface.co/blog/gptq-integration
- https://arxiv.org/abs/2402.13116
- https://arxiv.org/abs/1911.02150
- https://arxiv.org/abs/2305.13245
- https://arxiv.org/abs/2402.02750
- https://arxiv.org/abs/2309.17453
- https://arxiv.org/abs/2306.14048
- https://arxiv.org/pdf/2407.02490

References

- https://huggingface.co/blog/whisper-speculative-decoding
- https://arxiv.org/pdf/2211.17192.pdf
- https://www.together.ai/blog/medusa
- https://arxiv.org/abs/2402.16363
- https://medium.com/@yohoso/llm-inference-optimisation-continuous-batching-2d66844c19e9
- https://www.usenix.org/conference/osdi22/presentation/yu
- https://www.usenix.org/system/files/osdi24-zhong-yinmin.pdf
- https://zhuanlan.zhihu.com/p/27181462601