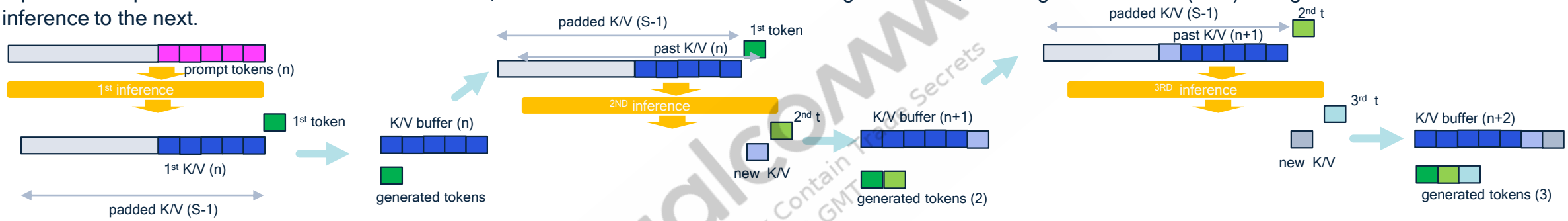
Qualcomm

LLM Technical Notes KV Cache Implementation for inference on HTP

QC AI Research and AISW Teams

Design Principles

The autoregression nature of LLM (such as GPT-x or Llama) leads to dynamic tensor size in some input, output and hidden states. In order to eliminate repetitive computation of K and V in attention heads, KV cache is used in 2nd and following inferences, and length of KV cache(KV$) also grows from one inference to the next.



The current generation of model inference engine for NSP, does not support dynamic input, output and hidden states, and there is no built-in caching scheme inside the inference engine.

The KV cache (KV$) implementation is as follows:

1. The cache (KV$) is hosted outside inference as an application-level component, along with its own buffer management (KV manager).
2. The first bunch of computed KV tensors, from the first inference, are mapped as model output and saved in the KV manager
3. For the following inferences, each inference takes the cached K or V tensors (one or more) as input, and gives the newly computed one K or V tensor as output
4. To overcome the limitation of no dynamic size tensor support, each cached K/V tensor is left padded to the length of S-1. S is the max sequence length
5. Since NSP inference engine does quantized computing, the K/V tensor quantization has special handling to avoid unnecessary hidden quantization/re-quantization inside each inference and between inferences

* The number of tokens generated from 2nd inference and on could be greater than 1, for illustration purpose, we use 1

Quantized model optimization

For quantized model, the cached KV tensors shall avoid unnecessary re-quantization in inferences. The repetitive re-quantization adds execute cost with potential of accumulating quantization loss.

With HW aware graph transformation of multi-head-attention to single-head-attention. For each single head attention, the following optimizations are placed:

a) Each padded past (cached) K or V sequence is a separate input tensor, dimensions [1,1,S-1, D]

b) Each newly computed K or V is a separate output tensor, dimensions [1,1,1,D]

c) To form a complete padded sequence, Each past K or V sequence is concatenated with newly computed value, dimensions become [1,1,S,D]

d) The past K or V sequence, the newly computed K or V, have the same quantization parameters. By this, there is no requantization when passing K or V inside inference or between inferences

KV cache buffer management optimizations

The KV cache could grow so big that it could be quite an overhead to inference cost, in terms of both memory consumption and latency increase.

The input K or V sequence is left padded to S-1 so that it could be consumed by a static graph. The input tensor is required to placed in contiguous memory.

After each inference, the KV manager needs to append the newly computed value to grow the cache size. Conceptually, this is a left shift and append operation. In contiguous buffer layout, it requires a memory copy operation.

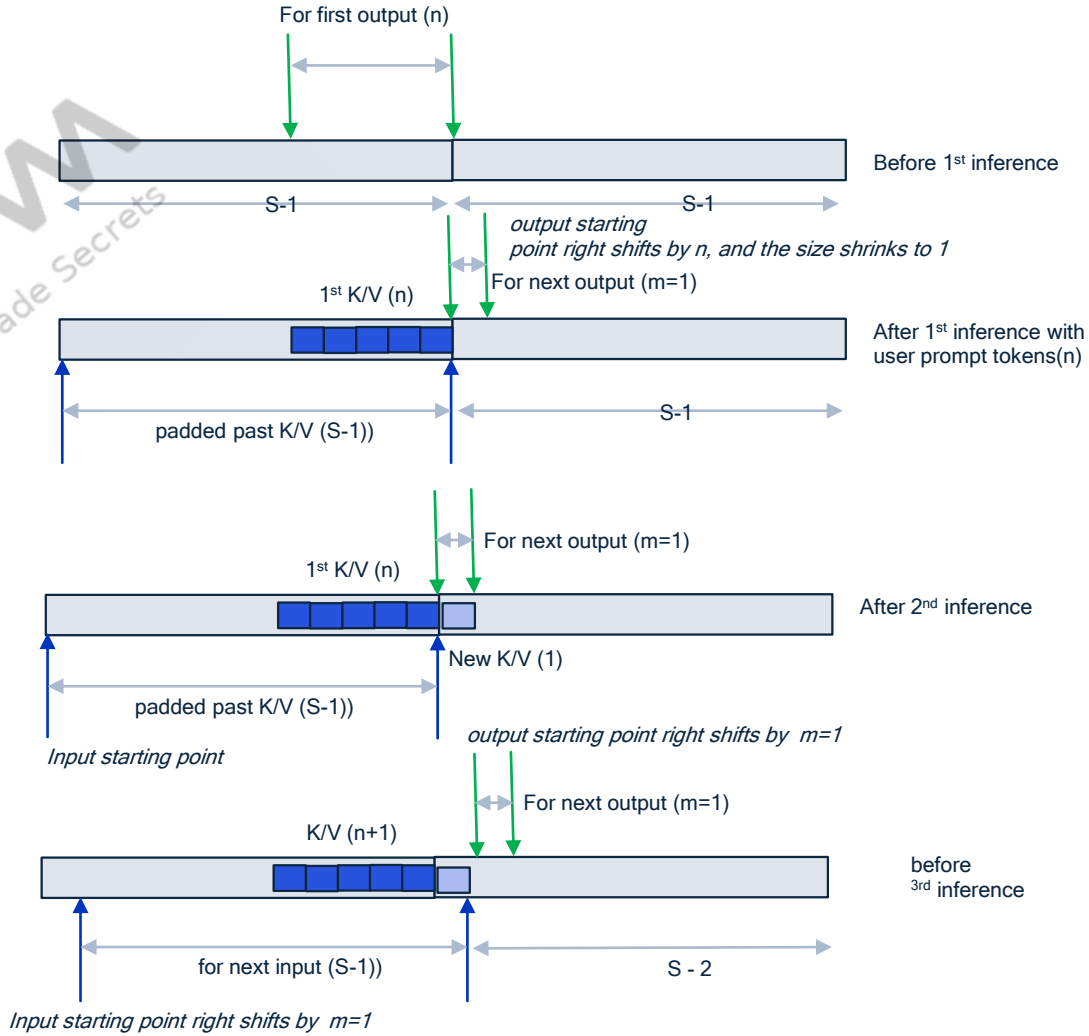
The following optimization are (will be) implemented to avoid expensive large buffer copy, but at cost of reserving more memory.

a) The application KV manager and model inference input/output share a common buffer

b) The application KV manager stores padded K or V tensor in the shared buffer for model input, ready to use by next inference without any more costly left shifting in contiguous buffer

c) The KV manager reserves 2(S-1) x D instead of S x D amount of memory for each K or V to avoid copy. With the extra memory, the append is a no-op and left-shifting becomes much cheaper pointer arithmetic.

d) The KV cache grows up to S-1 without any copy. If token generation is allowed to be byond max sequence length S, the KV manager could do one expensive copy to shift entire K/V sequence to left by S-1



KV cache buffer size

If the application KV manager reserves 2x S for each cached K or V sequence, the total amount of shared memory required:

2 x 2 (for both K and V) x n_decoder_blocks x n_heads (number of heads in MHA) x (S (max seq. length) - 1) x D(single head hidden state dimension) x quant-bit-width/8

For Llama 7B, n_heads = 32, D=128, n_decoder_blocks = 32, quant-bit-width = 16, S=2048

Total size = 2,146,435,072

When the model size is large, the amount of memory required to avoid copy entirely could be impractical.

As a compromise, less memory has to be reserved, for example S-1 + (S-1)/2,

Total size = 1,609,826,304

and the buffer manager algorithm needs to be adjusted to do copy more often