MultiDecode

May 29, 2025

Ted Kyi and Roger Stager

MultiDecode overview

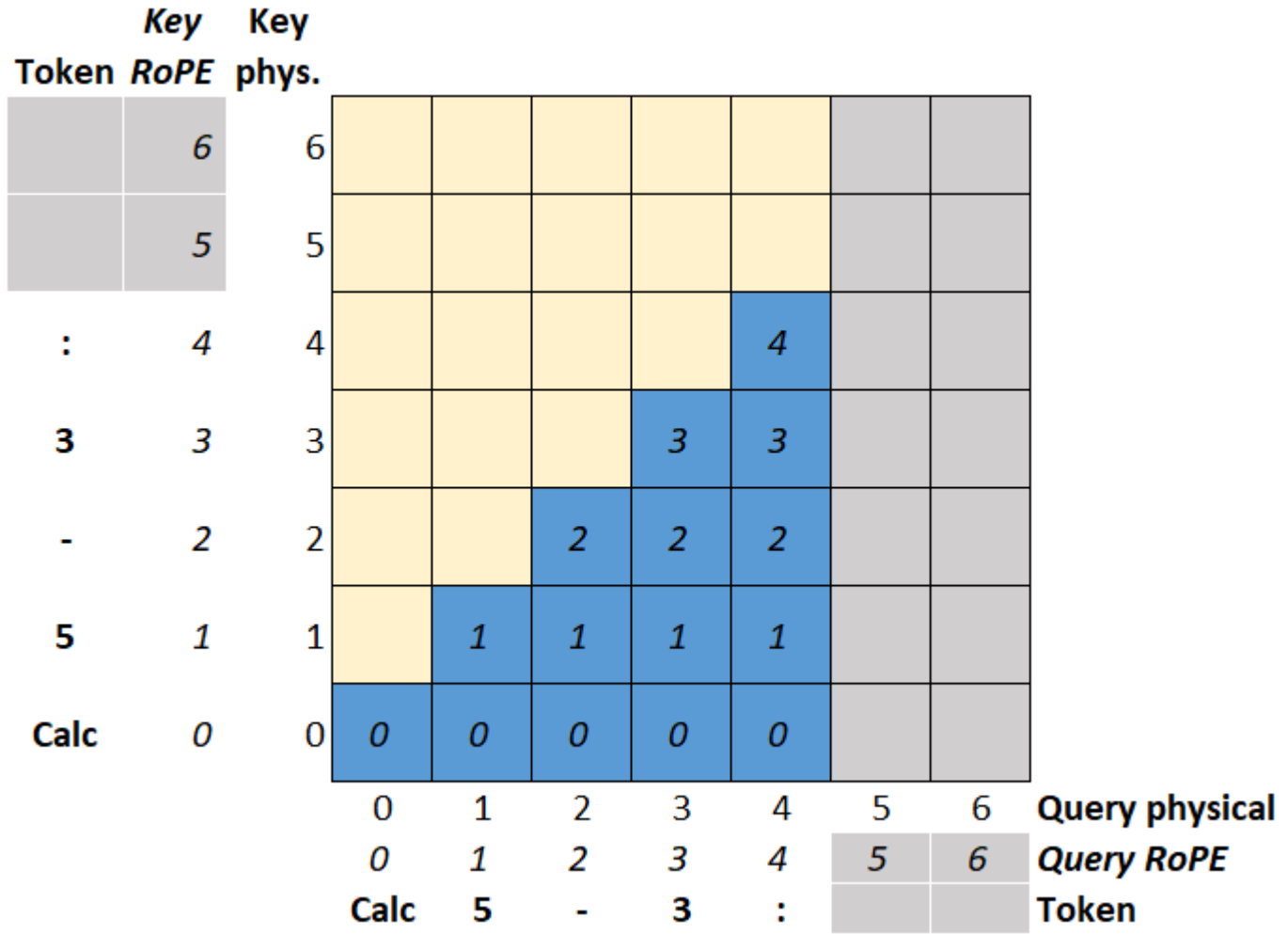
- When training GPT-style, autoregressive decoder-only LLMs, it is common practice to use teacher forcing and learn in parallel from all token positions
  - The predictions are used from every token position for backpropagation
  - The triangular autoregressive mask is required to prevent tokens from attending to other tokens that are later in the sequence
- During decoding, the common practice is to decode one token at a time, using only the prediction from the last token position
- However, nothing stops us from using the predictions from multiple token positions, as long as we ensure every token attends to an appropriate subset of the other tokens

Attention calculation

- The attention calculation is position-agnostic and happens in parallel across all token positions
- Position embeddings, such as RoPE, are required in order for the LLM to know the order of the input tokens
- Given we can specify both the position of each token and the mask of which other tokens each token can attend to, we have total control
  - The input tokens don’t have to appear in sequential order, nor do the RoPE positions need to match the physical token positions
  - Further there can be multiple tokens with the same RoPE position number
  - If each token with RoPE position *n* sees *n*-1 tokens with each of the previous numbers, the calculation will be identical to standard decoding

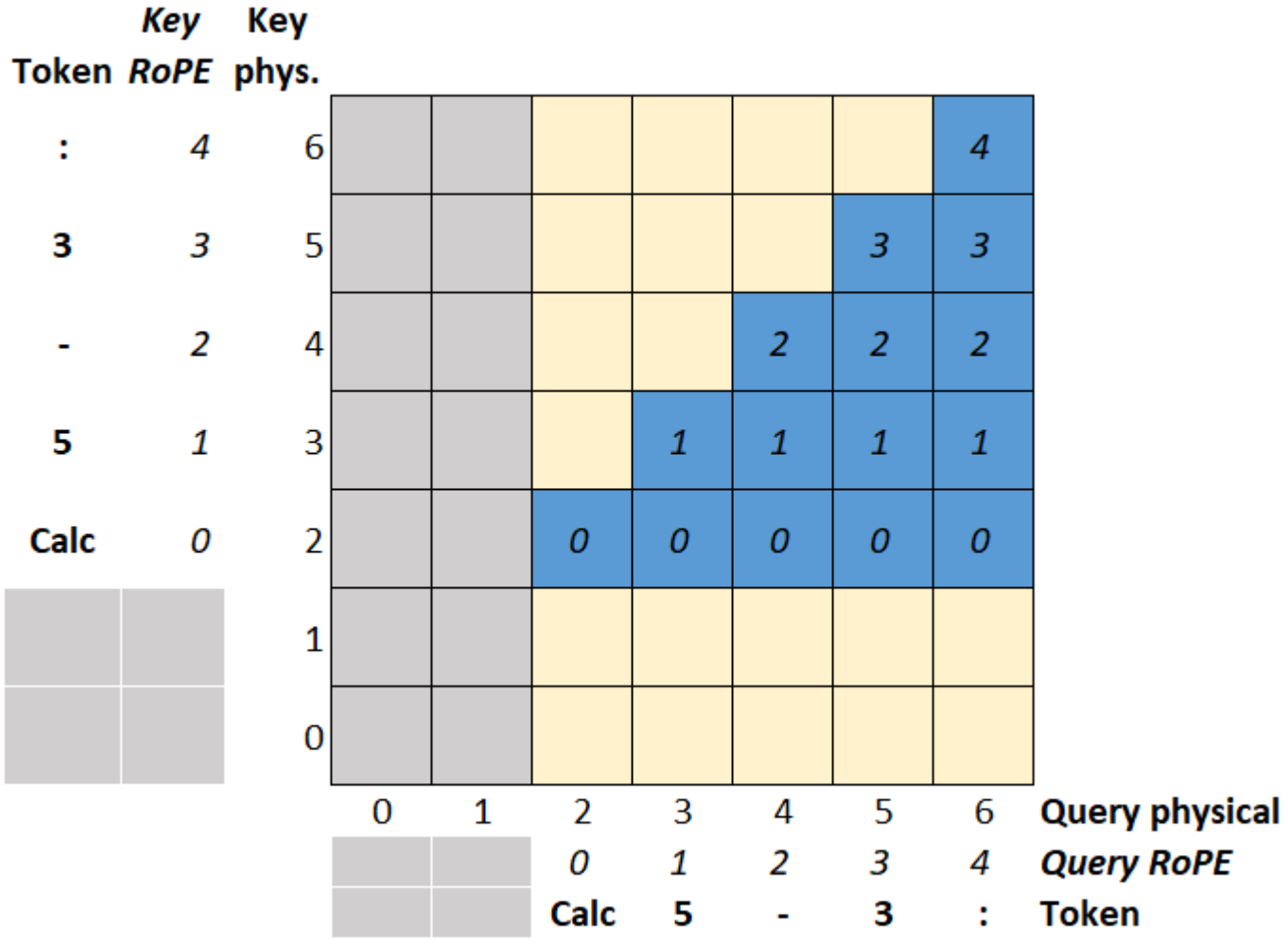
Attention masks and position encoding

- Prompt: “Calc 5 - 3:”
- This diagram shows standard decoding
- The RoPE values always match the physical token position
- The masks grow by one each token position



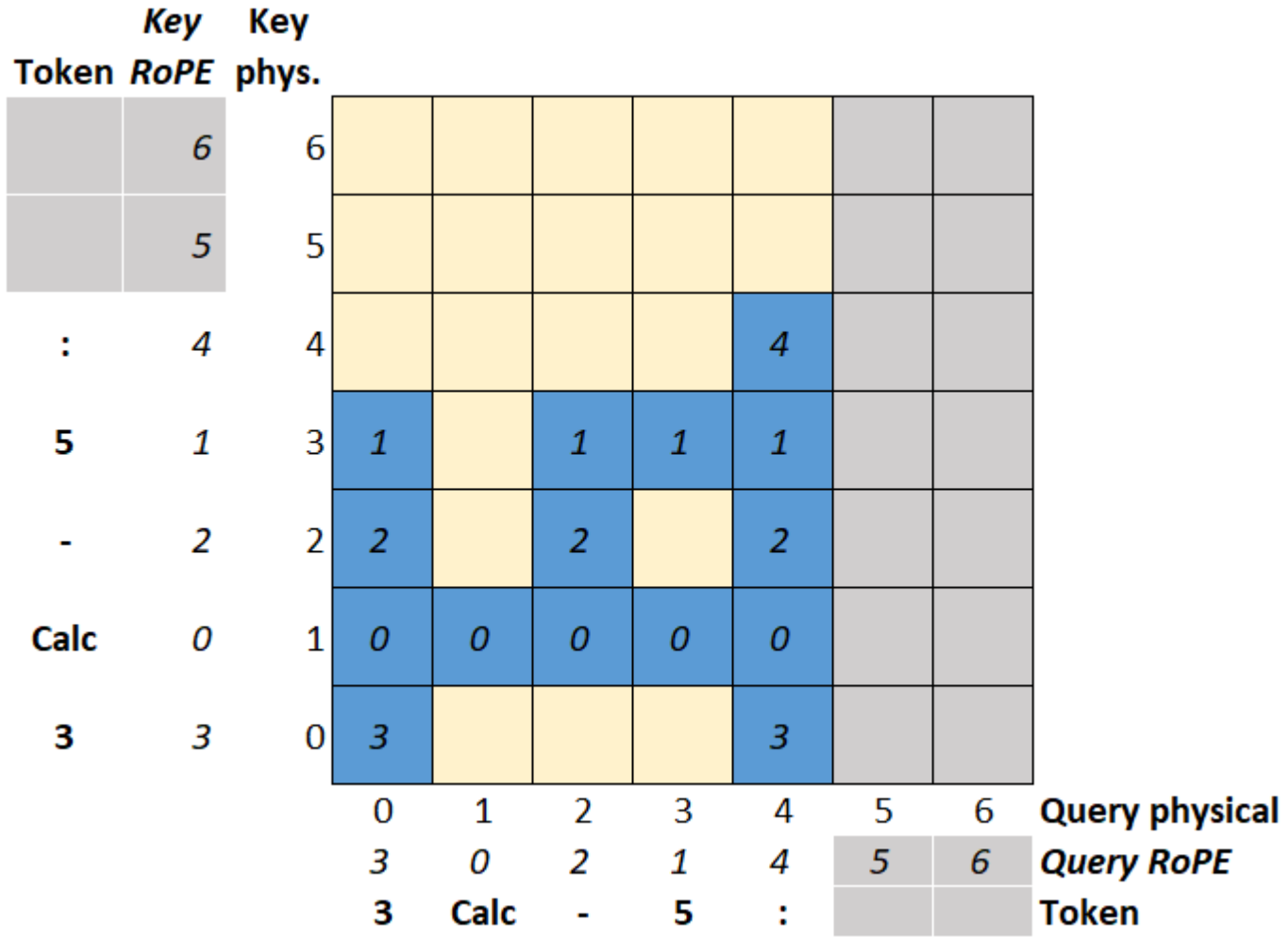
Shifting input tokens

- Prompt: “Calc 5 - 3:”
- Here, the RoPE values no longer match the physical token position
- The first two physical token positions are masked out for all of the real tokens
- The predictions for next token after the “:” will be the same



Shuffling input tokens

- Prompt: “Calc 5 - 3:”
- Here we have a hypothetical input that will also produce the exact same next-token predictions
- This further demonstrates that masks need not be contiguous



MultiDecode method

- We have seen that tokens can move around, and as long as we do the proper bookkeeping, the decoding calculation is identical
- For the next-token prediction at a particular position to be correct, we only need to ensure that every token attends to an appropriate subset of the other tokens
- The subset of tokens attended to need not be contiguous, triangular, or any limitation other than tokens cannot attend to later tokens
- Further, we can have multiple positions where we make next-token predictions, as long as we follow the above, minimal limitations
  - This is an exact decoding method (not an approximation)
  - MultiDecode is fully compatible with KV caching
  - Different completions can share subsets of KV cache entries

Example [1]

- As a simple RAG example of reading comprehension, we can build proper masks to ask two questions in parallel:

  `<system_prompt> <retrieved_story> Who is the main character? Where is the story located?`

- We can correctly read next-token predictions for both question marks (shown in red and blue) in parallel because they are independent
- We will be merging two separate predictions into one:
  - `<system_prompt> <retrieved_story> Who is the main character?`
  - `<system_prompt> <retrieved_story> Where is the story located?`

Example [2]

- The masks for the question marks will have shared (green underline) tokens and private tokens (red and blue underline, respectively): `<system_prompt> <retrieved_story> Who is the main character? Where is the story located?`

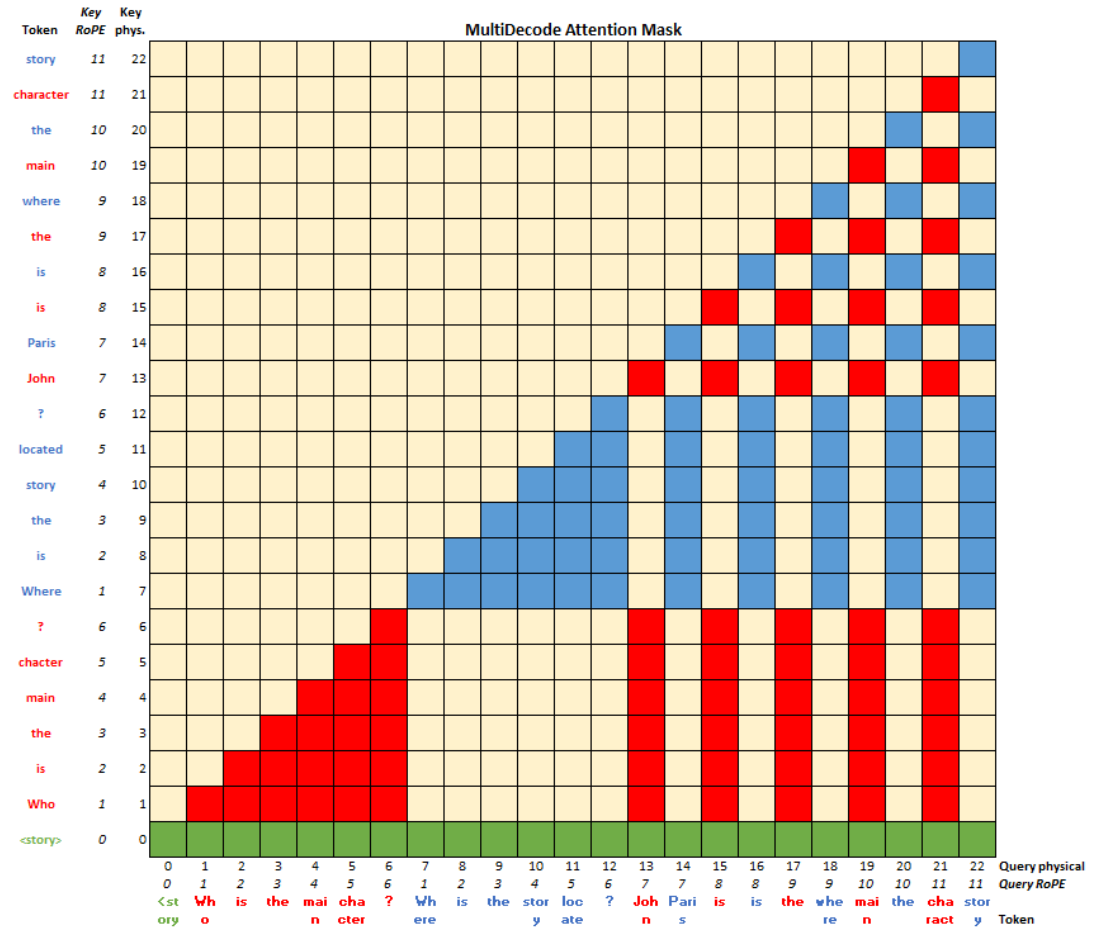
- We can show an inference step with new tokens appended: `<system_prompt> <retrieved_story> Who is the main character? Where is the story located? John Paris`
  - There will be two (red and blue) new tokens each inference step
  - As we append the new tokens, we will see red and blue tokens interleaved

Example [3]

- Inference steps continue:
  - <system_prompt> <retrieved_story> Who is the main character? Where is the story located?
  - <system_prompt> <retrieved_story> Who is the main character? Where is the story located? John Paris
  - <system_prompt> <retrieved_story> Who is the main character? Where is the story located? John Paris is is
  - <system_prompt> <retrieved_story> Who is the main character? Where is the story located? John Paris is is the where
  - <system_prompt> <retrieved_story> Who is the main character? Where is the story located? John Paris is is the where main the
  - <system_prompt> <retrieved_story> Who is the main character? Where is the story located? John Paris is is the where main the character story

Example Mask

- Subset of the attention mask for the questions and decoded tokens

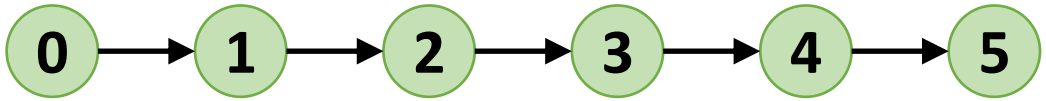


Why MultiDecode is faster

- Let’s consider the previous example with two questions being asked about a block of text loaded into the context
- MultiDecode is faster than answering each question sequentially because it reduces the number of inference steps by approximately half
- An alternative would be to save the KV cache for the tokens prior to the first question. Multiple questions could use copies of the same KV cache in separate batch entries.
  - Using copies of the KV cache avoids the cost of redundant prefill calculations
  - However, using multiple copies of the KV cache creates additional IO pressure on the GPU for the loading of the KV cache
  - MultiDecode shares one copy of the KV cache. Despite having about the same number of inference steps, since decoding is IO bound, MultiDecode is much faster

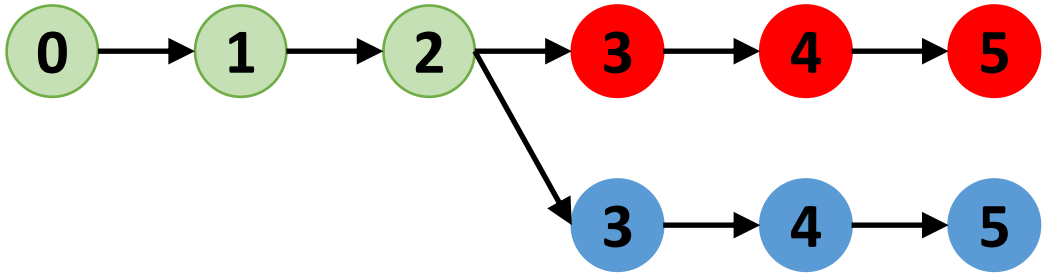
Valid decoding patterns [1]

- The standard decoding pattern has tokens in a simple linear sequence



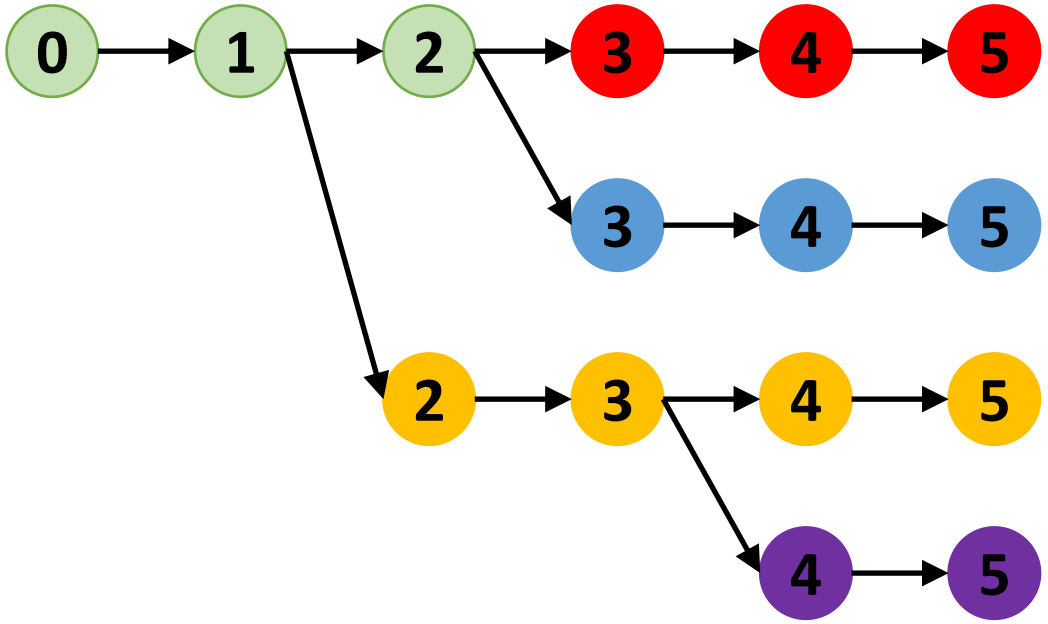
Valid decoding patterns [2]

- Branching on later tokens preserves the causal pattern for each token



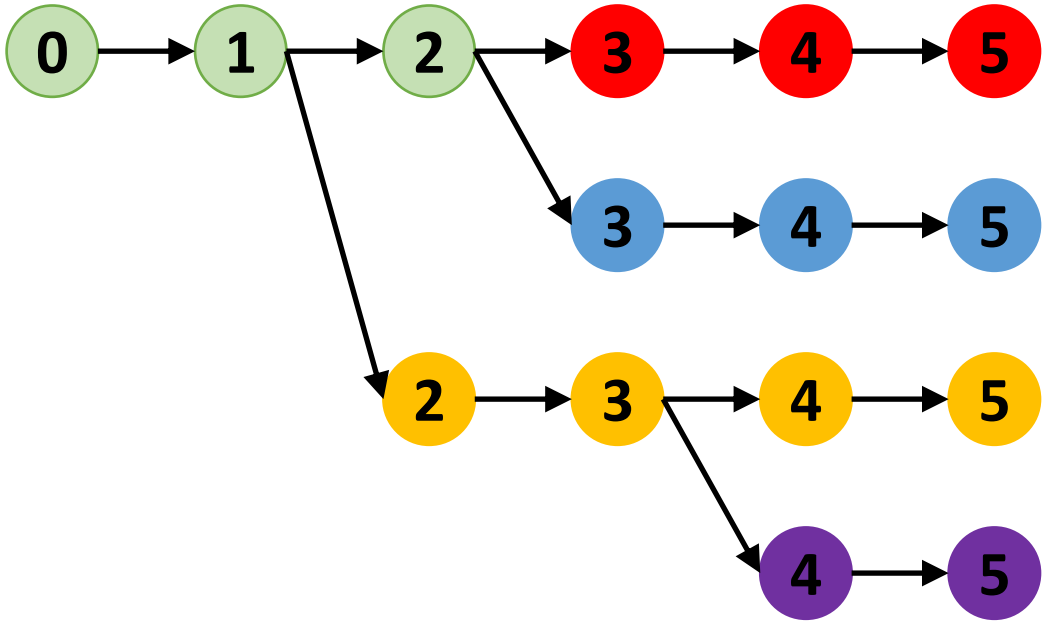
Valid decoding patterns [3]

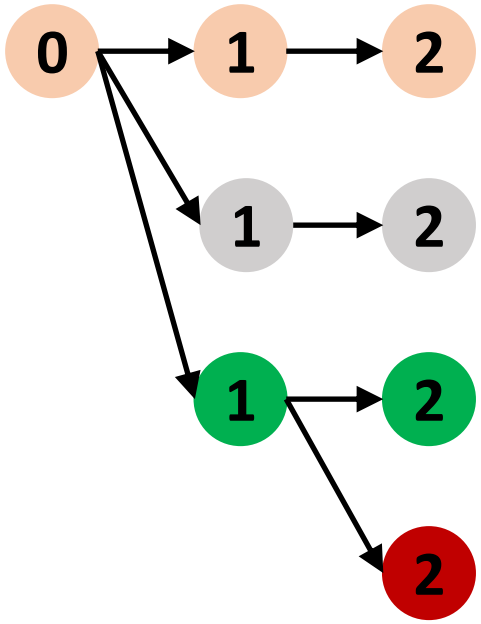
- Any tree is valid



Valid decoding patterns [4]

- Any forest is valid





Linear sequence

- A tree/forest of multiple completions must be fitted into the standard one-dimensional token sequence for MultiDecoding
- An intuitive requirement is that tokens earlier in the causal chain for one or more other tokens should be placed physically earlier than the tokens with causal dependence on them
- Either a depth-first search or a breadth-first search (or a mix of them) of the tree/forest is sufficient to meet this causal requirement

Use cases [1]

- There are many use cases where parallel decoding can improve the speed and/or quality of LLM inference
  - Fast beam search or other algorithms which consider multiple completions before committing to a next token
    - The Lookahead Decoding paper uses custom masks and position values. MultiDecoding would appear to be a generalization of potential strategies for exact multiple decoding
  - Parallel implementation of margin generation for the Writing in the Margins technique for improving RAG accuracy
  - Multiple, parallel chains of thought
  - Faster, cheaper answering of multiple independent questions
  - Batching multiple short prompts with one or more long prompts
- Inference can be forward-only, but could also delete/rewind tokens from the KV cache like beam search and Writing in the Margins

Use cases [2]

- Further discussion of interesting use cases:
  - RAG – the documents retrieved form a naturally parallel use case. Whether writing in the margins or some other algorithm, processing of documents can be done in parallel. MultiDecode also allows the context for document prompts to be customized to the exact subset of tokens desired.
  - Search algorithms for better decoding for reasoning. Greedy decoding can lead to myopic token choices. A simple alternative would be to generate *k* parallel thoughts (such as complete sentences), then choose between them. Beam search, entropy-based sampling, and many other algorithms can be performed with greatly reduced overhead using MultiDecode.
- Parallel completions can be merged with an extra prefill step, incorporating these completions into a single context for future decoding
  - Allowing gaps in RoPE values may allow completions to be spaced apart and then combined without a prefill step to renumber them. This is an area of future research.

MultiDecode conclusion

- MultiDecode takes advantage of the parallel position-agnostic nature of the attention calculation to obtain correct next-token predictions from multiple locations during each inference step
- MultiDecode is an exact method
- Does *not* require any of:
  - Special model architecture or heads
  - Special training
  - Special hardware
- Does require the LLM decoding software to support custom masks and RoPE embeddings
- There are many use cases where parallel decoding can improve the speed and/or quality of LLM inference

References

- Writing in the Margins: Better Inference Pattern for Long Context Retrieval
  Melisa Russak et al. (2024)
  https://arxiv.org/abs/2408.14906
- Break the Sequential Dependency of LLM Inference Using Lookahead Decoding
  Yichao Fu et al. (2024)
  https://arxiv.org/abs/2402.02057