Efficient LLM Decoding

Large Language Models: Introduction and Recent Advances

ELL881 · AIL821

Yatin Nandwani
Research Scientist, IBM Research

Till now…

- **Motivation –** Inference is sequential, memory bound and slow, with high latency
- **KV caching** – avoids re-computation of Keys and Value matrices
- **Paged Attention and vLLM -** efficient memory management
- Can we speed up attention computation?
- Flash Attention?

Flash Attention - Recap

- "*I/O aware*" implementation of Attention

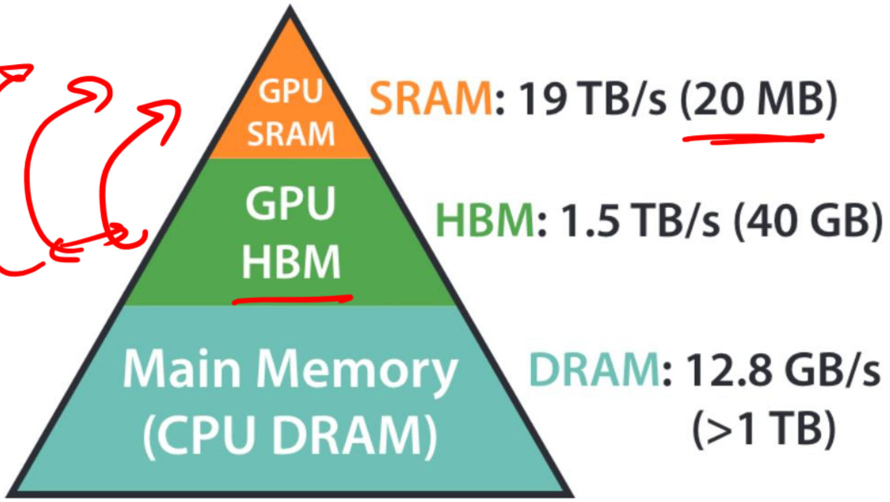
1. Matmul_op (Q,K)
   a. Read Q,K to SRAM (read-op)
   b. Compute matmul A=QxK (compute-op)
   c. Write A to HBM (write-op)
2. Mask_op
   a. Read A to SRAM (read-op)
   b. Mask A into A' (compute-op)
   c. Write A' to HBM (write-op)
3. Softmax_op
   a. Read A' to SRAM (read-op)
   b. Softmax A' into A" (compute-op)
   c. Write A" to HBM (write-op)

Standard Attention Implementation

Flash Attention

1. Read Q,K to SRAM
2. Compute A = QxK
3. Mask A into A'
4. Softmax A' into A"
5. Write A" to HBM

I/O aware attention implementation



Memory Hierarchy with Bandwidth & Memory Size

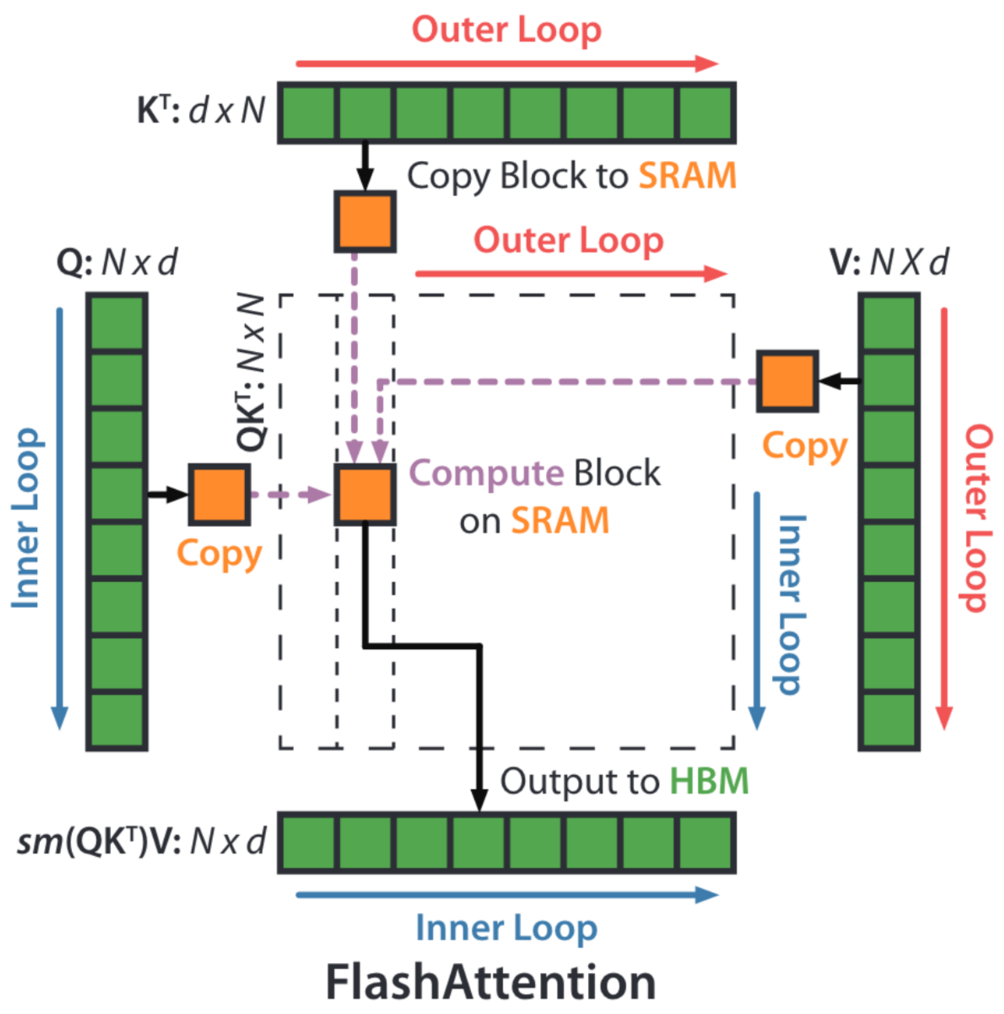
Flash Attention - Recap

- "*I/O aware*" implementation of Attention

  - Write a fused kernel to avoid multiple read / writes b/w HBM and SRAM

  - Tiling – decompose large *softmax* into smaller ones by scaling

A_1 A_2



$$softmax([A_1, A_2]) = [\, \alpha\, softmax(A_1), \beta\, softmax(A_2)\,]$$

$$softmax([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\, softmax(A_1) * V_1 + \beta\, softmax(A_2) * V_2$$

Flash Attention - Recap

- Tiling – decompose large *softmax* into smaller ones by scaling

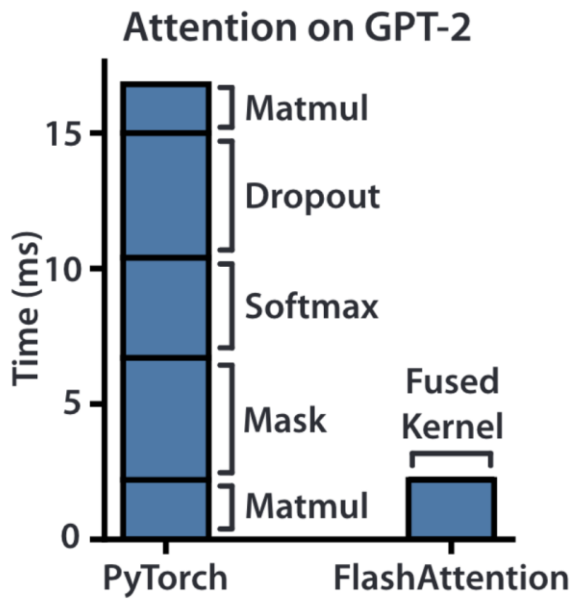
$$softmax([A_1, A_2]) = [\,\alpha\, softmax(A_1), \beta\, softmax(A_2)\,]$$

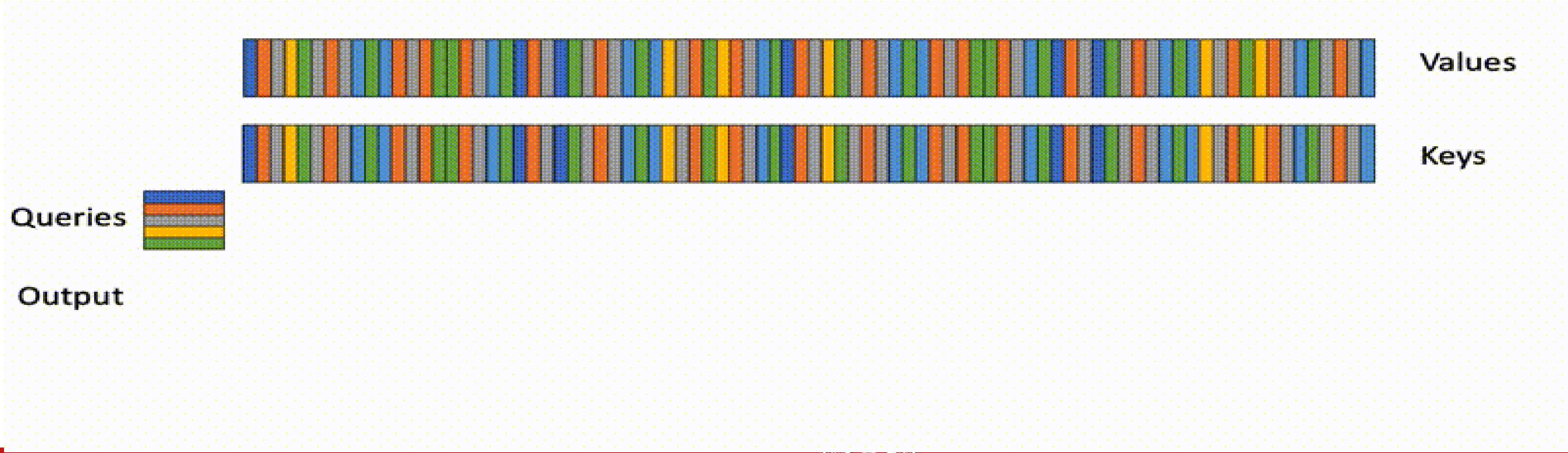
$$softmax([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\, softmax(A_1) * V_1 + \beta\, softmax(A_2) * V_2$$

1. Load inputs by blocks from HBM to SRAM
2. On chip, compute attention output w.r.t that block
3. Update output in HBM by scaling

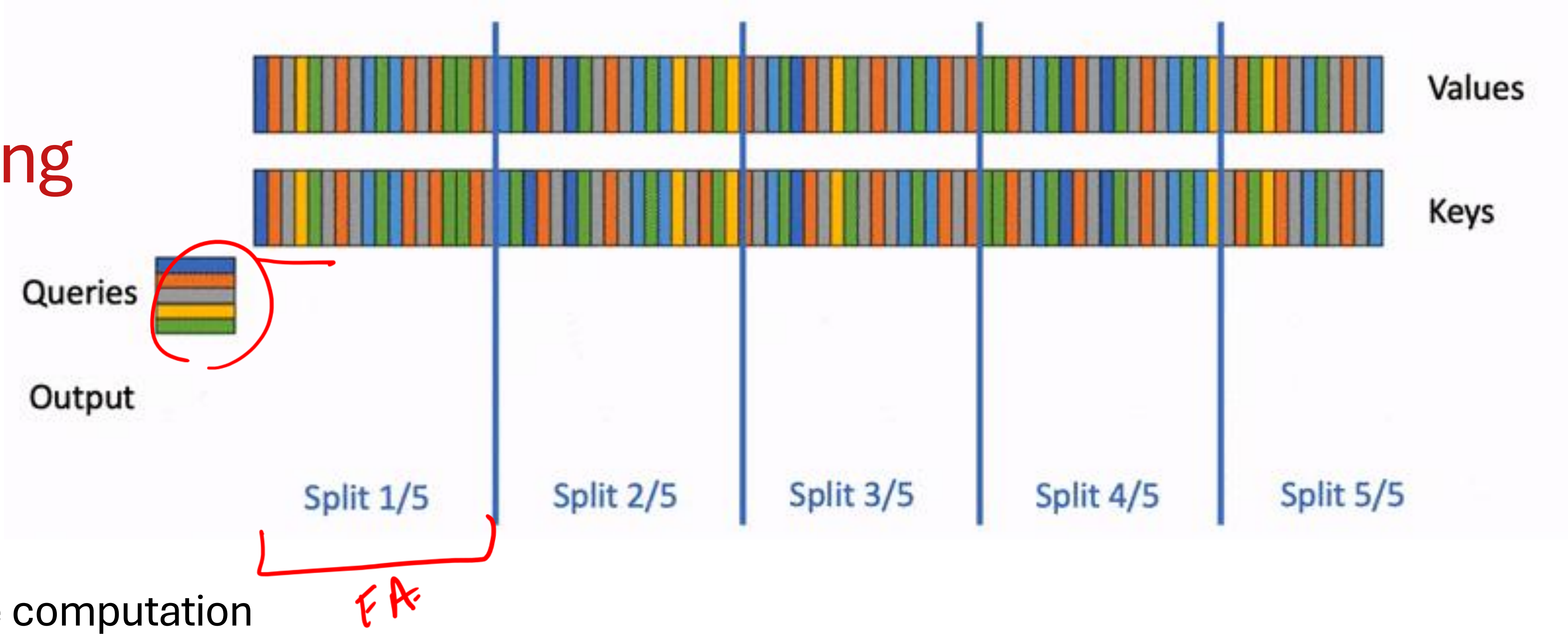
Flash Attention - Recap

- **2-4x Faster, 10-20x memory reduction**
- **Flash Attention** for training – parallelizes across batch size and query length dimension to avoid **memory bandwidth** bottleneck





Flash Decoding

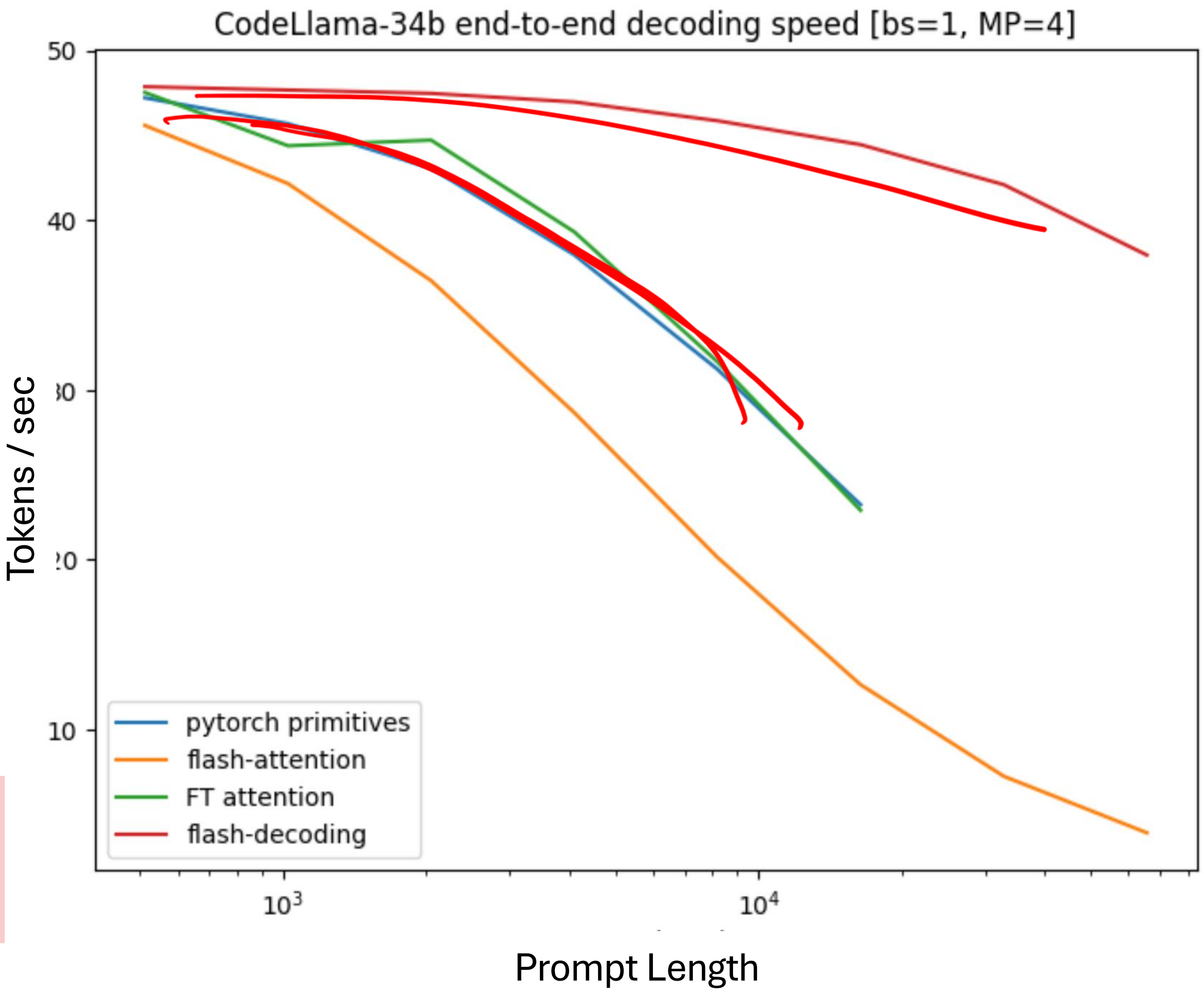


- Parallelize computation
  - split the keys/values in smaller chunks
  - compute the attention of the query with each of these splits in parallel (using Flash Attention)
  - 1 extra scalar per row and per split: the log-sum-exp of the attention values
  - Use the log-sum-exp to scale the contribution of each split

Source: https://princeton-nlp.github.io/flash-decoding/

Benchmarking on CodeLlama-34B

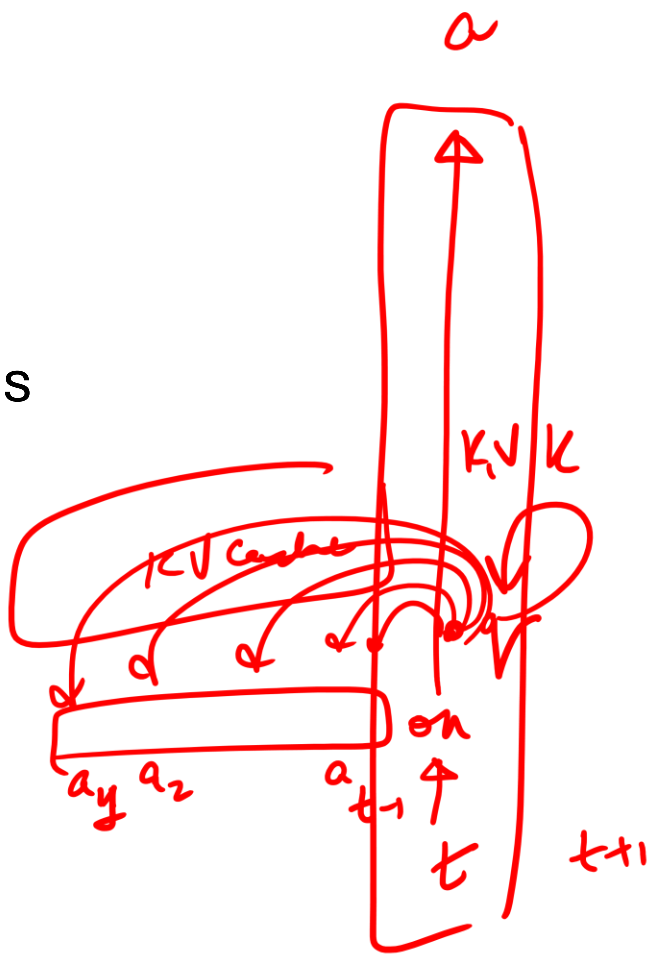
- **Pytorch**: Running the attention using pure PyTorch primitives (without using FlashAttention)
- **FlashAttention v2**
- **FasterTransformer:** Uses the FasterTransformer attention kernel
- Flash-Decoding



Flash-Decoding - 8x speedups in decoding speed for very large sequences

Till now…

- **KV caching** – avoids re-computation of Keys and Value matrices
- **Paged Attention and vLLM -** efficient memory management
- **Flash decoding –** efficient attention for very long sequences
- Generation is still sequential



What if we can generate multiple tokens in one iteration?

Generating multiple tokens in one iteration

Inference through an LLM

- **Input prompt:** “*The cat sat*”

Can we use a guess output to speed up inference?

Transformer based LLM (θ)

<s>	**The**	**cat**	**sat**				
0	1	2	3	4	5	6	7

Inference through an LLM

- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair*”

Transformer based LLM (ϑ)

<s>	**The**	**cat**	**sat**				
0	1	2	3	4	5	6	7

Inference through an LLM

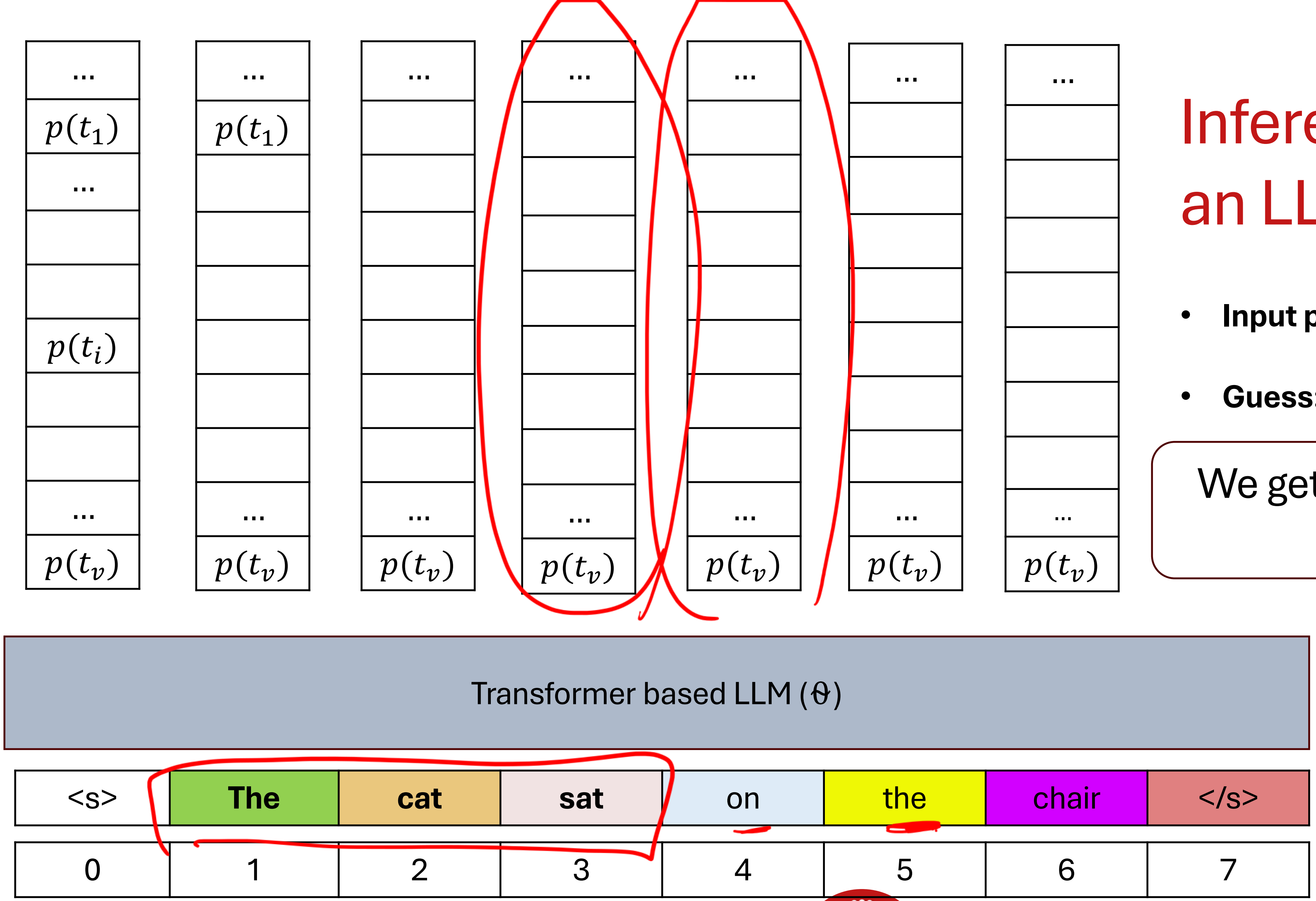
- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”

Run a forward pass with the guess completion

Transformer based LLM (θ)

<s>	**The**	**cat**	**sat**	on	the	chair	</s>
0	1	2	3	4	5	6	7

Inference through an LLM



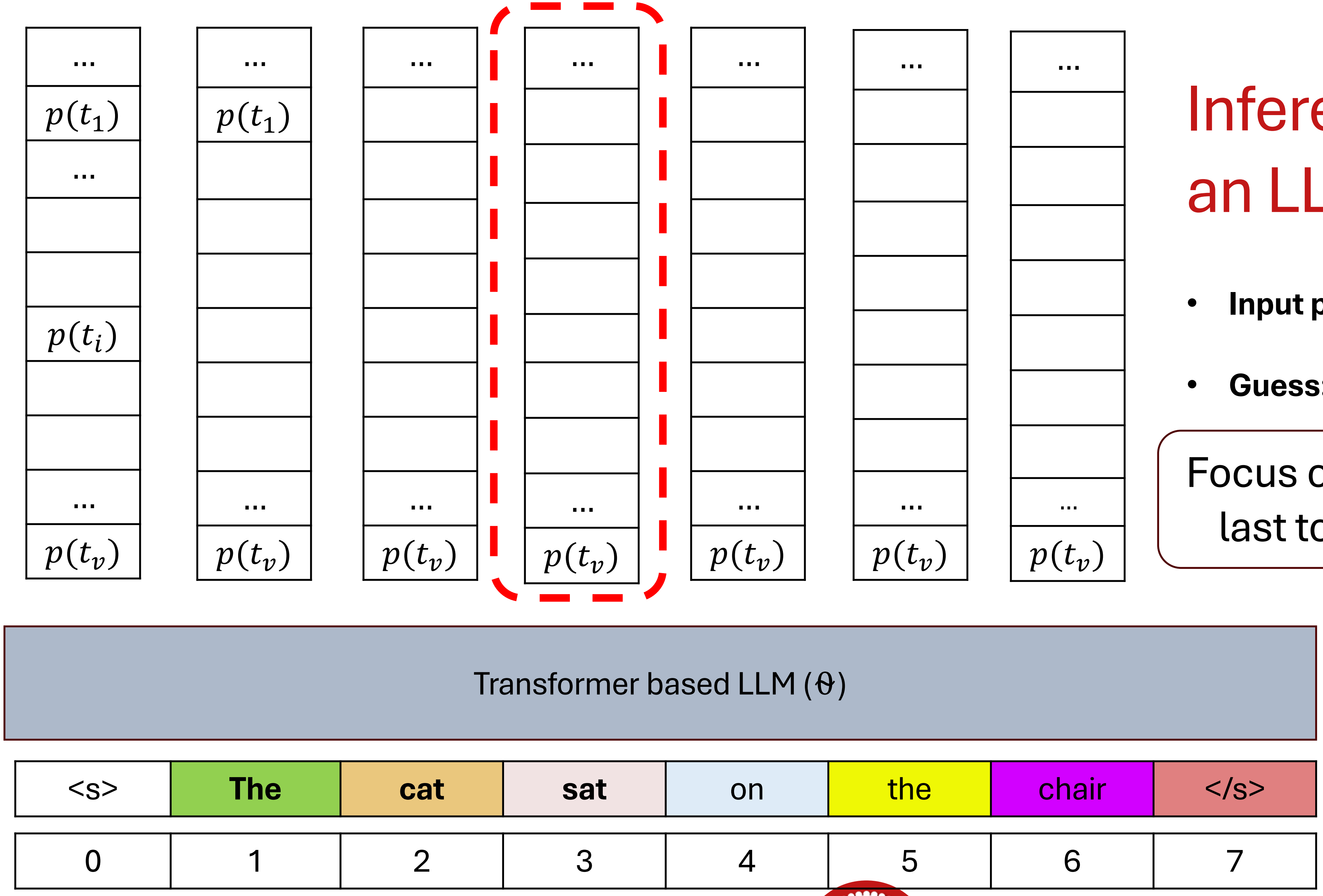
- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"

We get prob. dist. at each step

Inference through an LLM

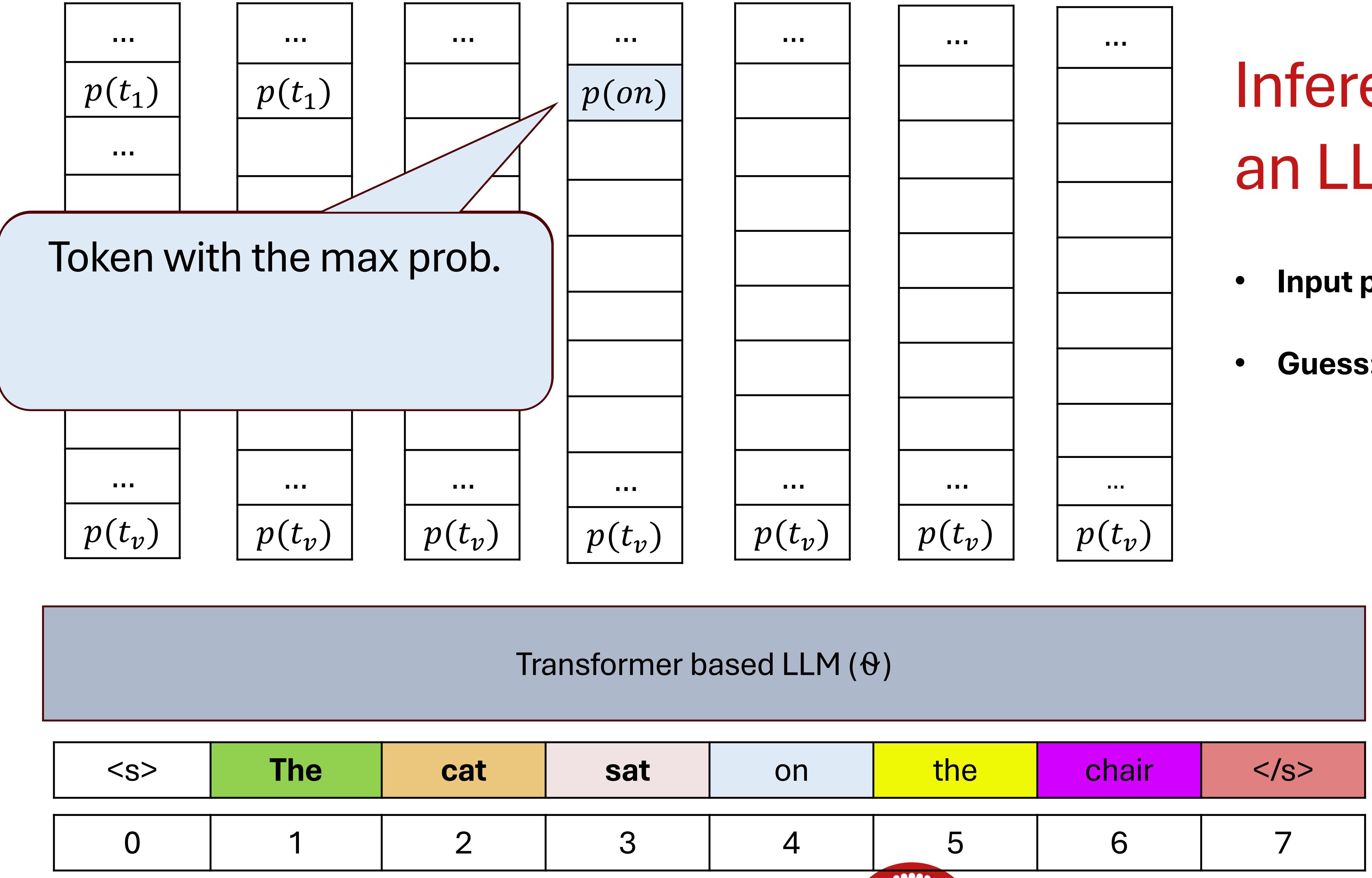
- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”

Focus on distribution at the last token in the prompt



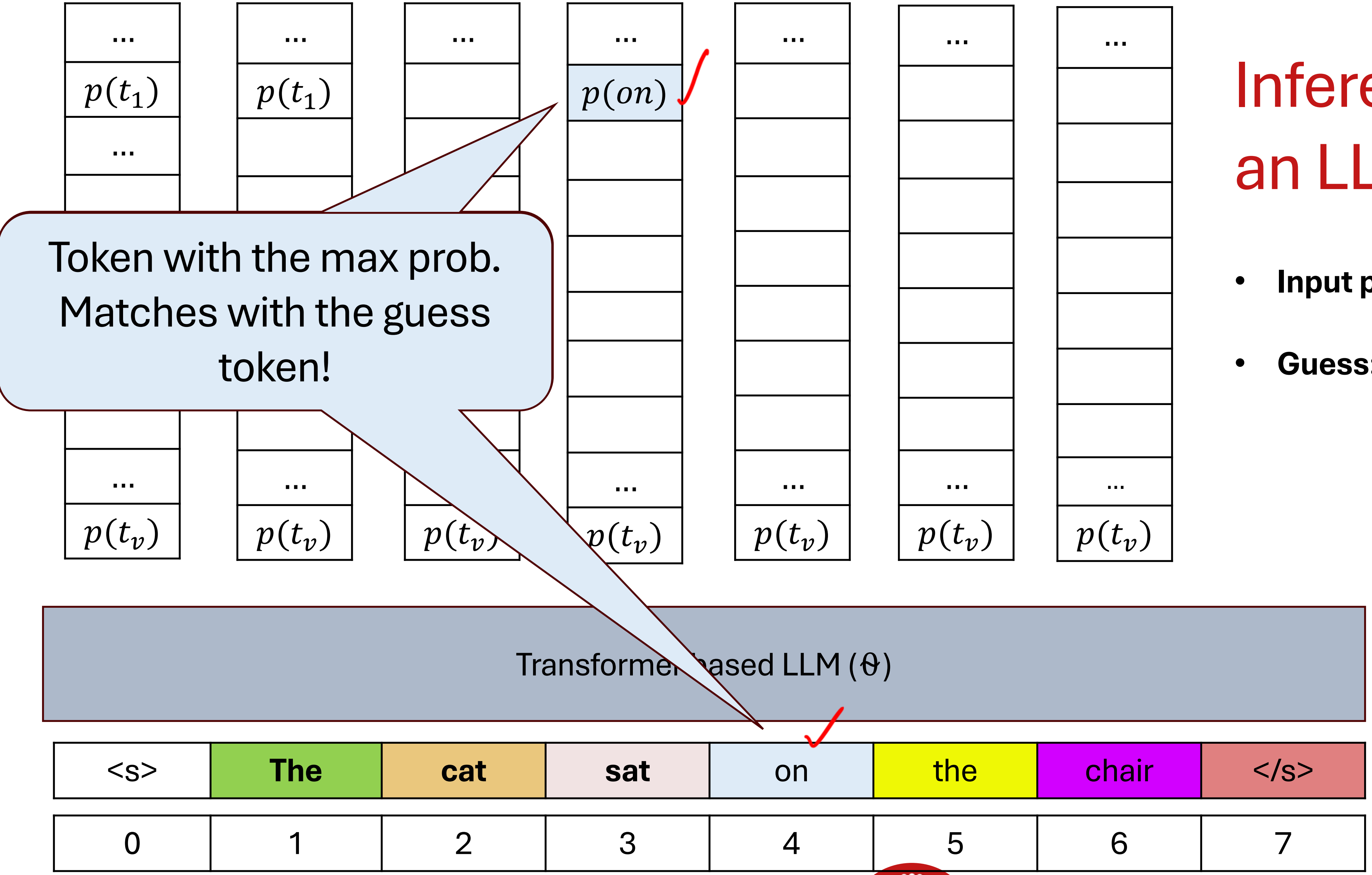
Inference through an LLM

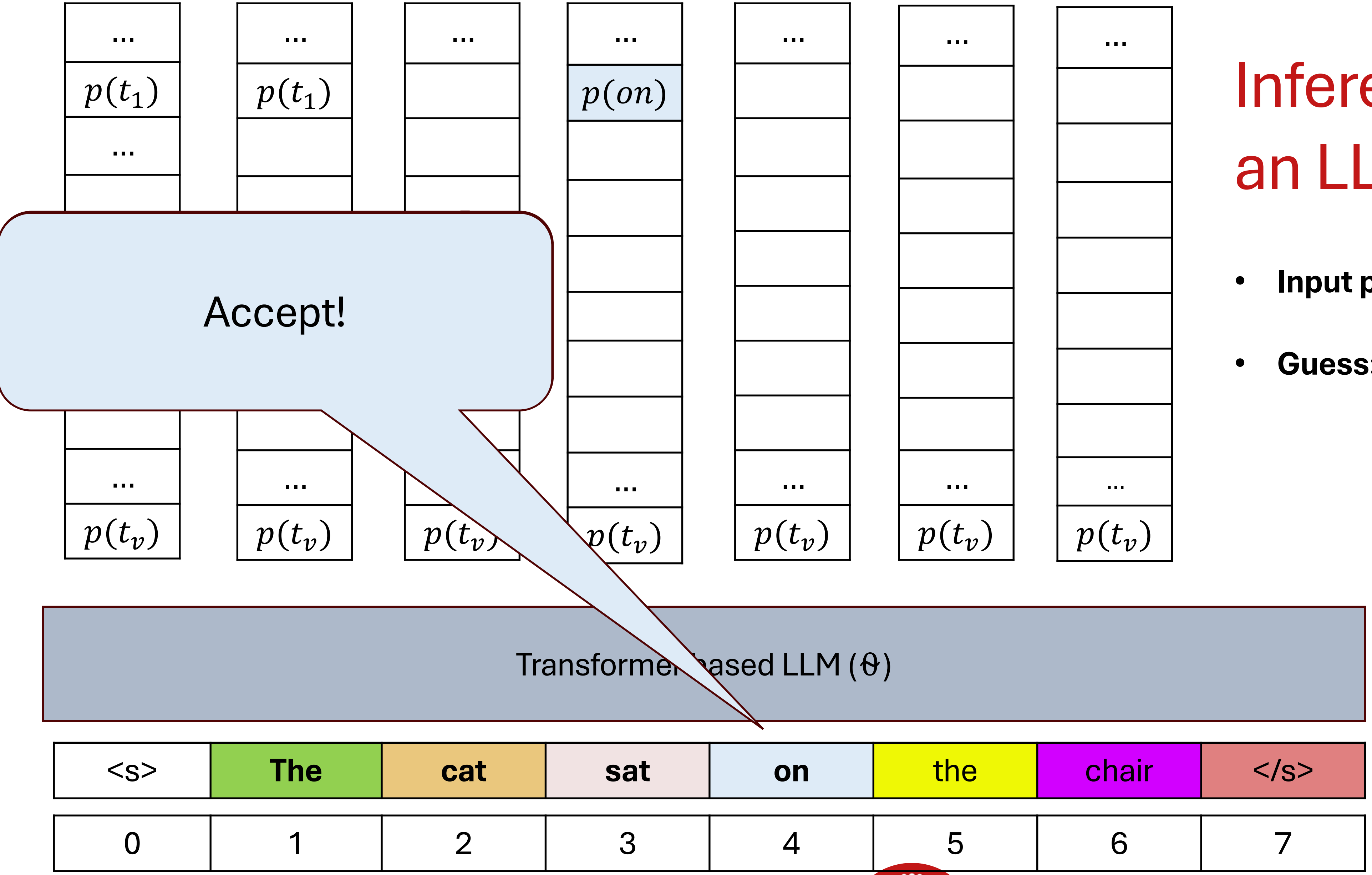
- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”



Inference through an LLM

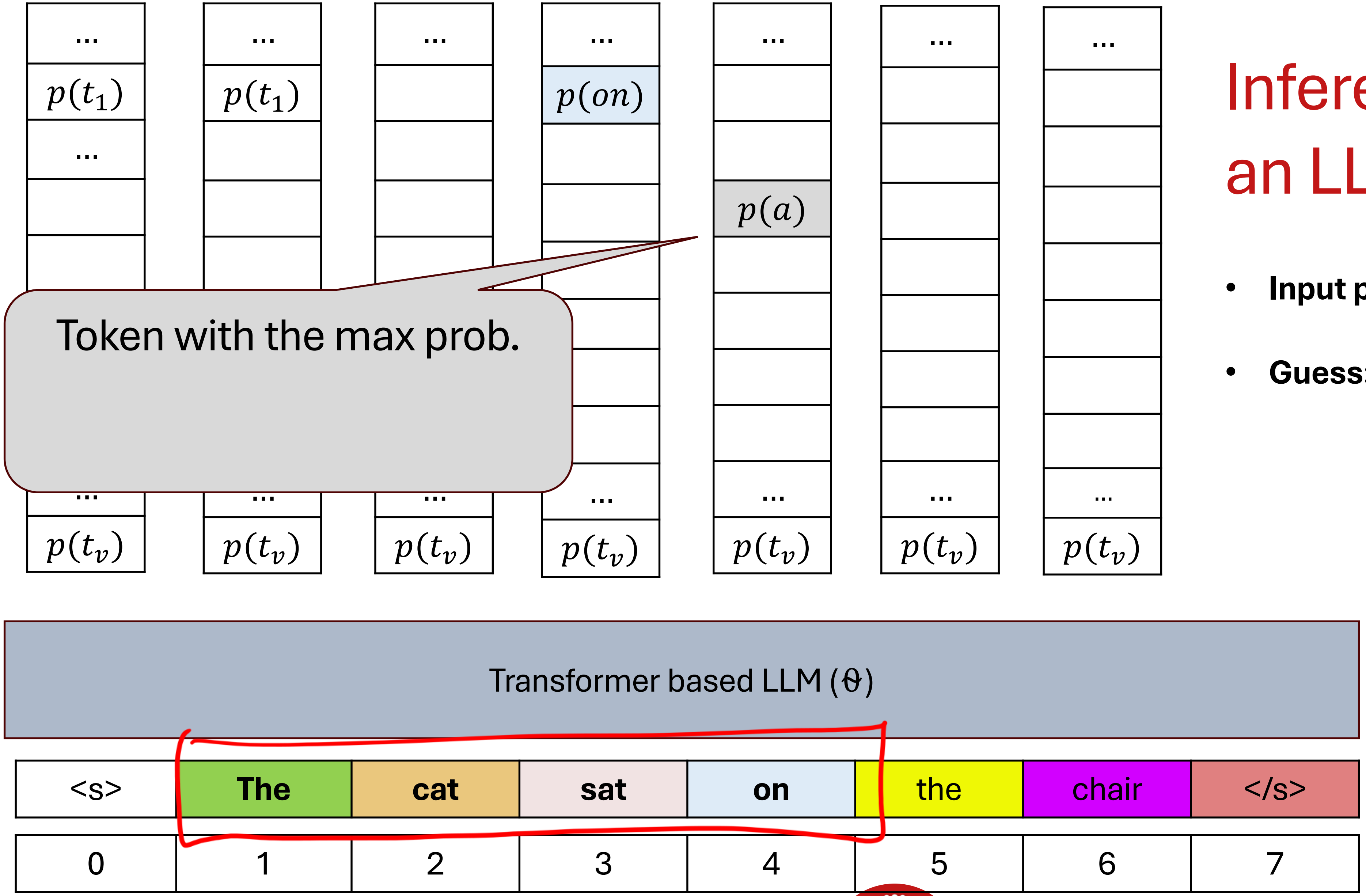
- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”





Inference through an LLM

- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"

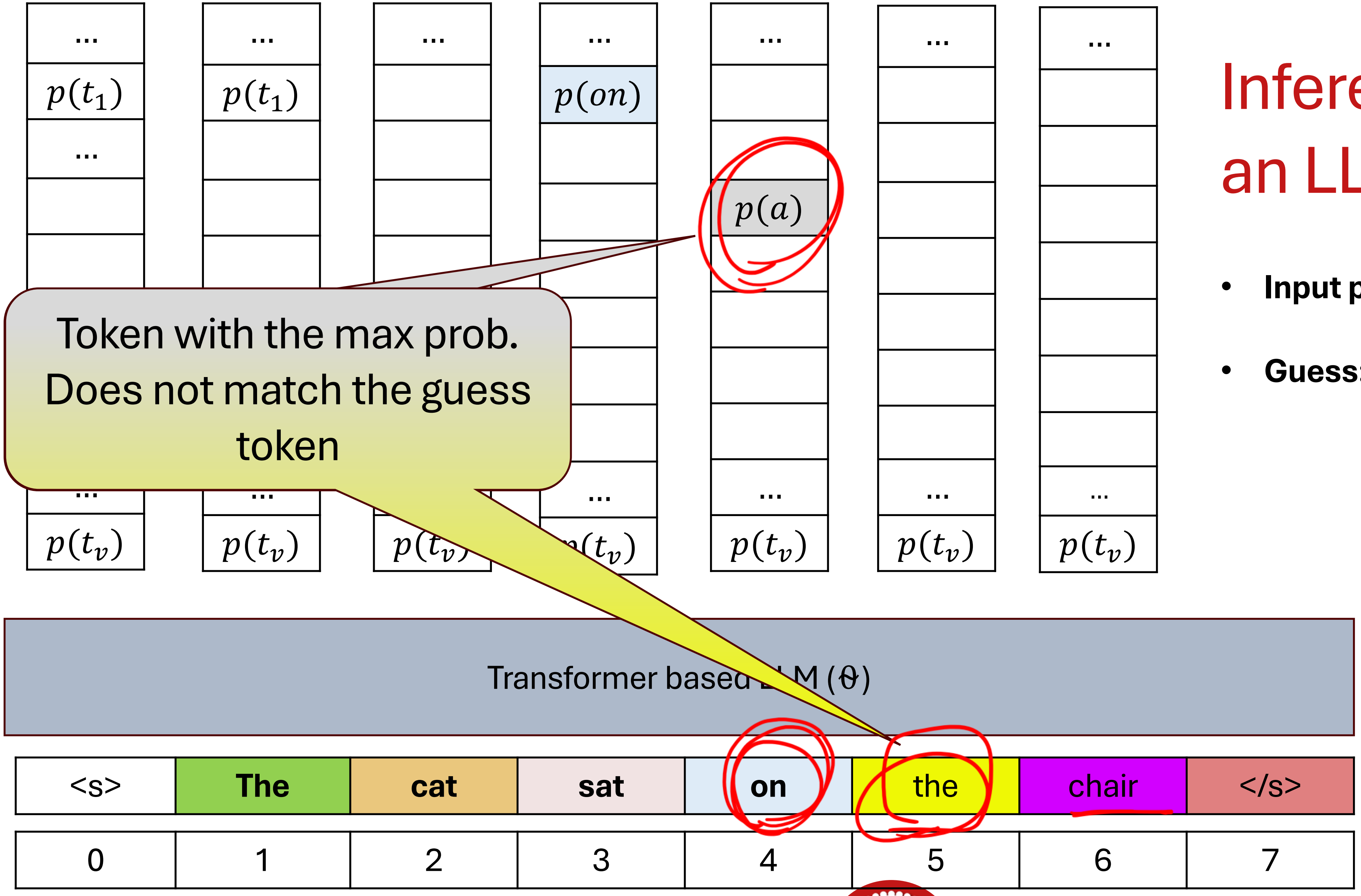


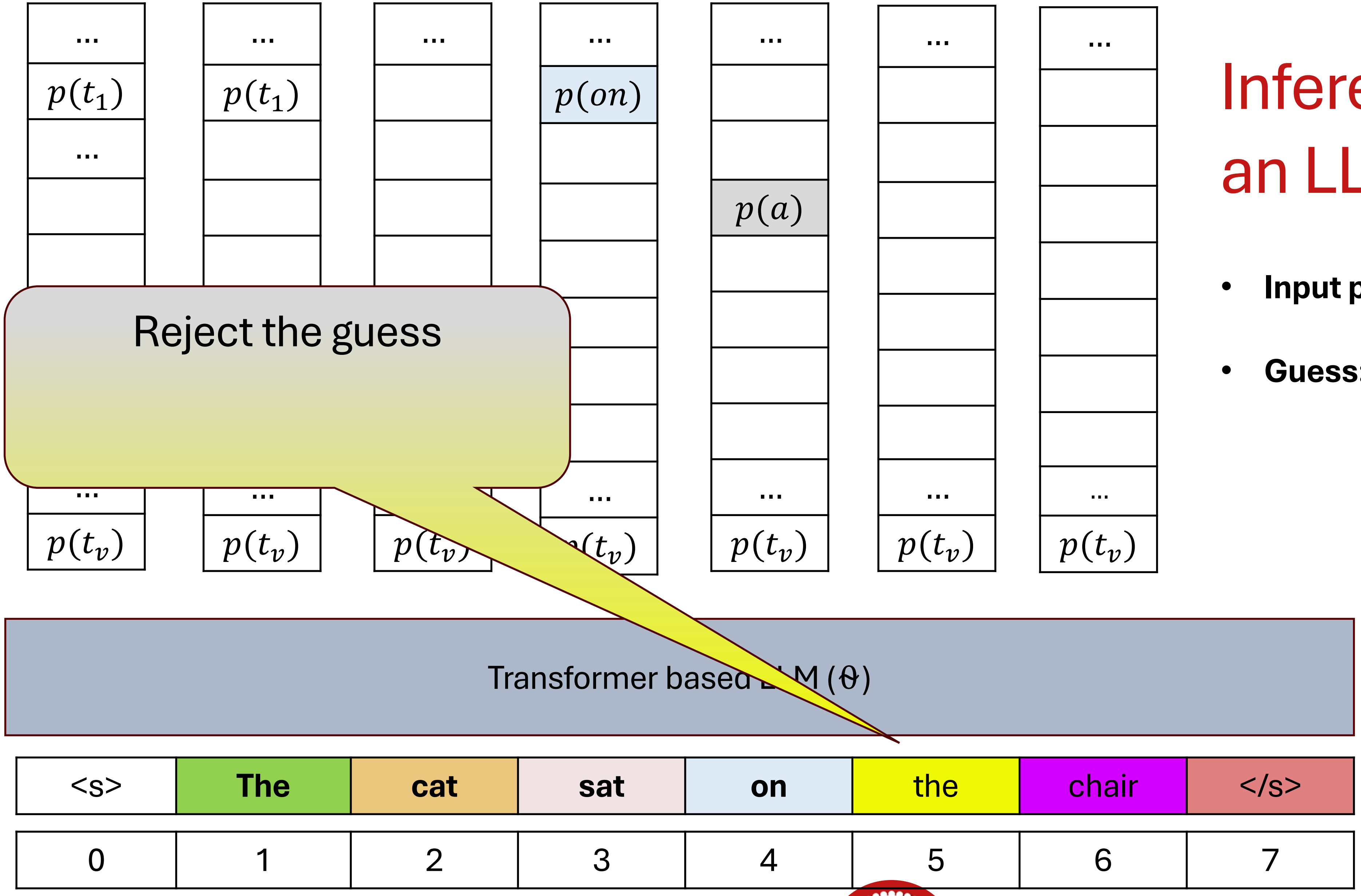
Inference through an LLM

- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"

Inference through an LLM

- **Input prompt:** "*The cat sat*"
- **Guess:** "*on the chair </s>*"



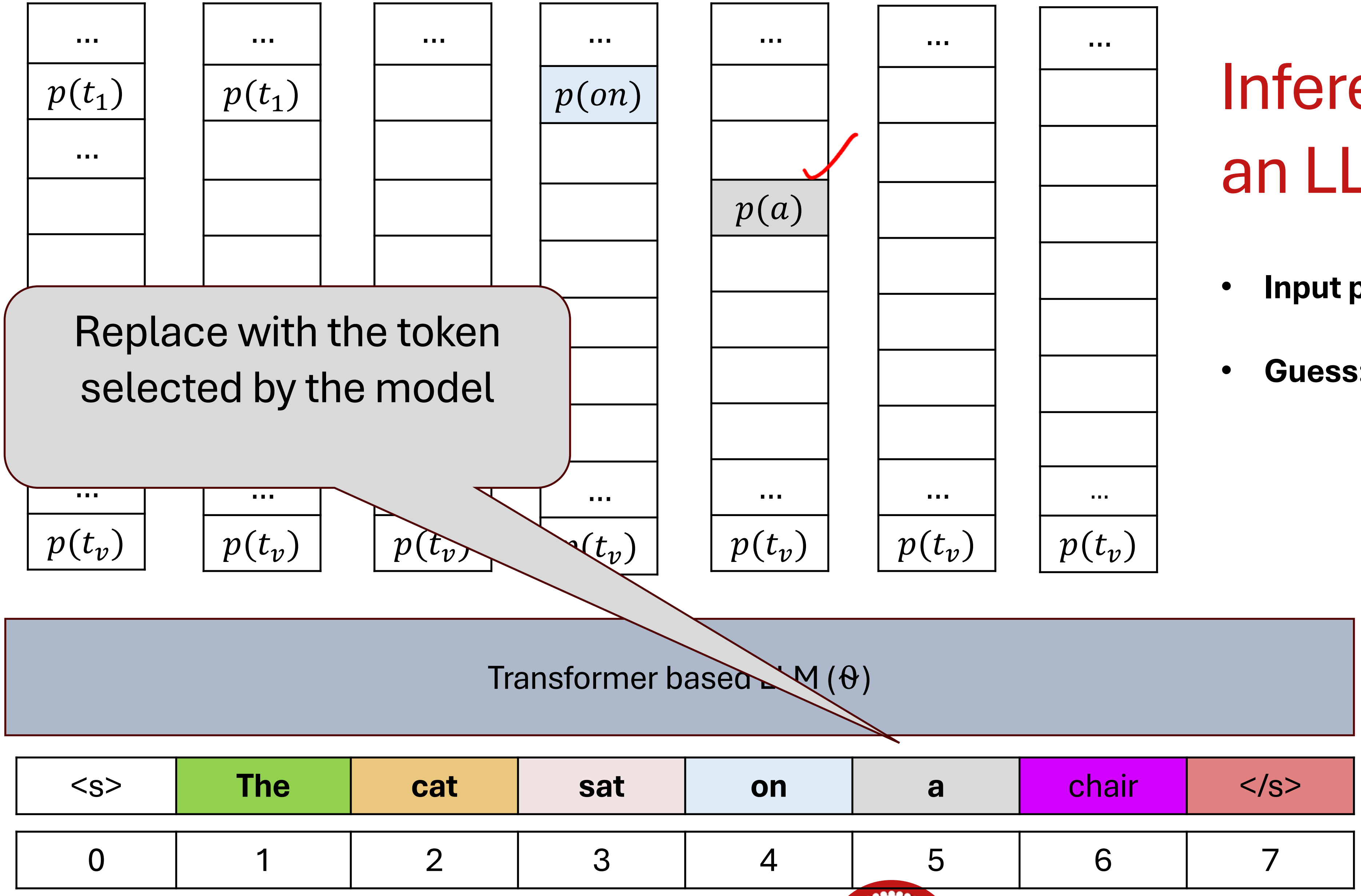


Inference through an LLM

- **Input prompt:** *"The cat sat"*
- **Guess:** *"on the chair </s>"*

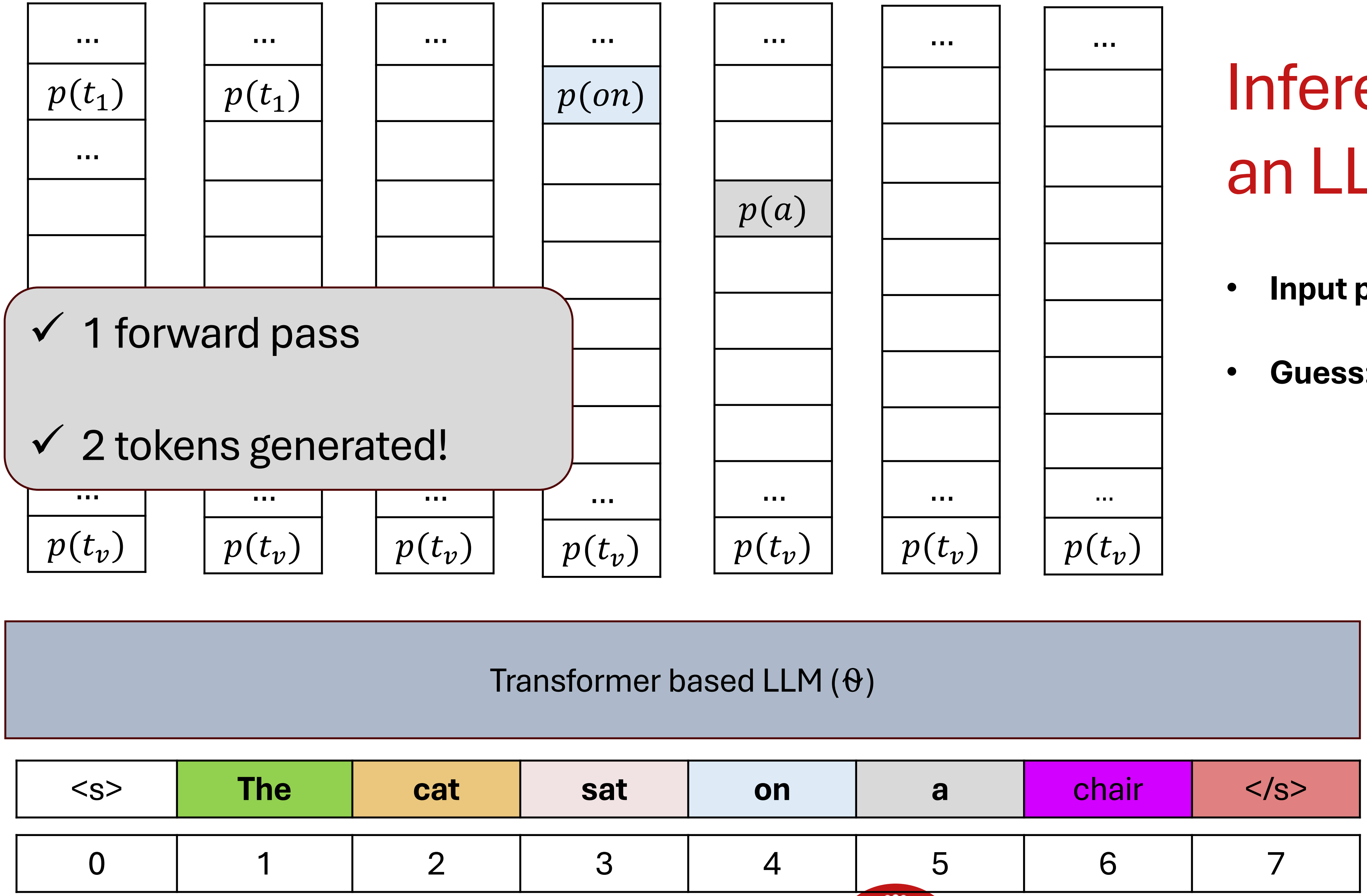
Inference through an LLM

- **Input prompt:** “*The cat sat*”
- **Guess:** “*on the chair </s>*”



Inference through an LLM

- **Input prompt:** *"The cat sat"*
- **Guess:** *"on the chair </s>"*



Can’t use rest of the completion as it was dependent on token “the” that has been rejected

Guess completion

<s>	**The**	**cat**	**sat**	**on**	**the**	chair	</s>

Verification by the LLM

				✓	✗		

New Input Prompt

<s>	**The**	**cat**	**sat**	**on**	**a**		

New Guess

<s>	**The**	**cat**	**sat**	**on**	**a**	table	</s>

Inference through an LLM

- **Input prompt:** “*The cat sat on a* ”
- **Guess:** “*table </s>*”

✓ Make a new guess with the updated input prompt

❑ Updated input: “*The cat sat on a*”

❑ Guess: “*table </s>*”

…

$p(a)$

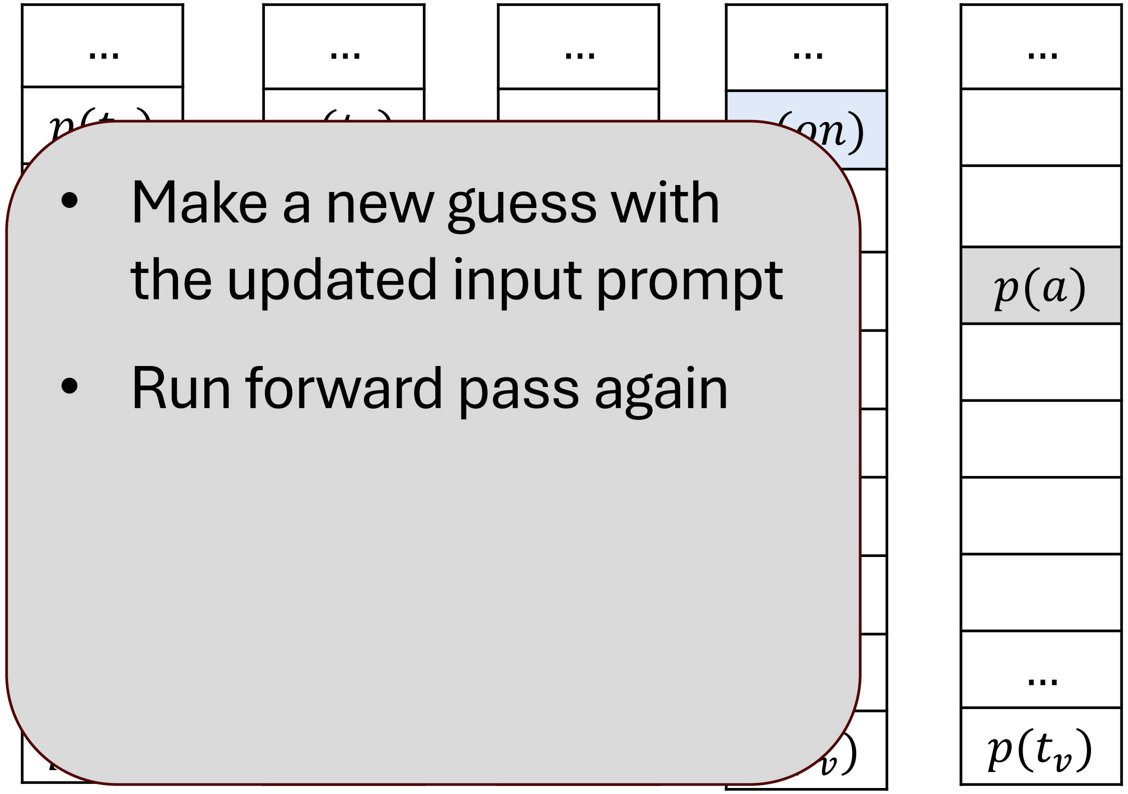
…

$p(t_v)$

Transformer based LLM (ϑ)

<s>	**The**	**cat**	**sat**	**on**	**a**	chair	</s>
0	1	2	3	4	5	6	7

Inference through an LLM



- Make a new guess with the updated input prompt
- Run forward pass again

- **Input prompt:** "*The cat sat on a* "
- **Guess:** "*table </s>*"

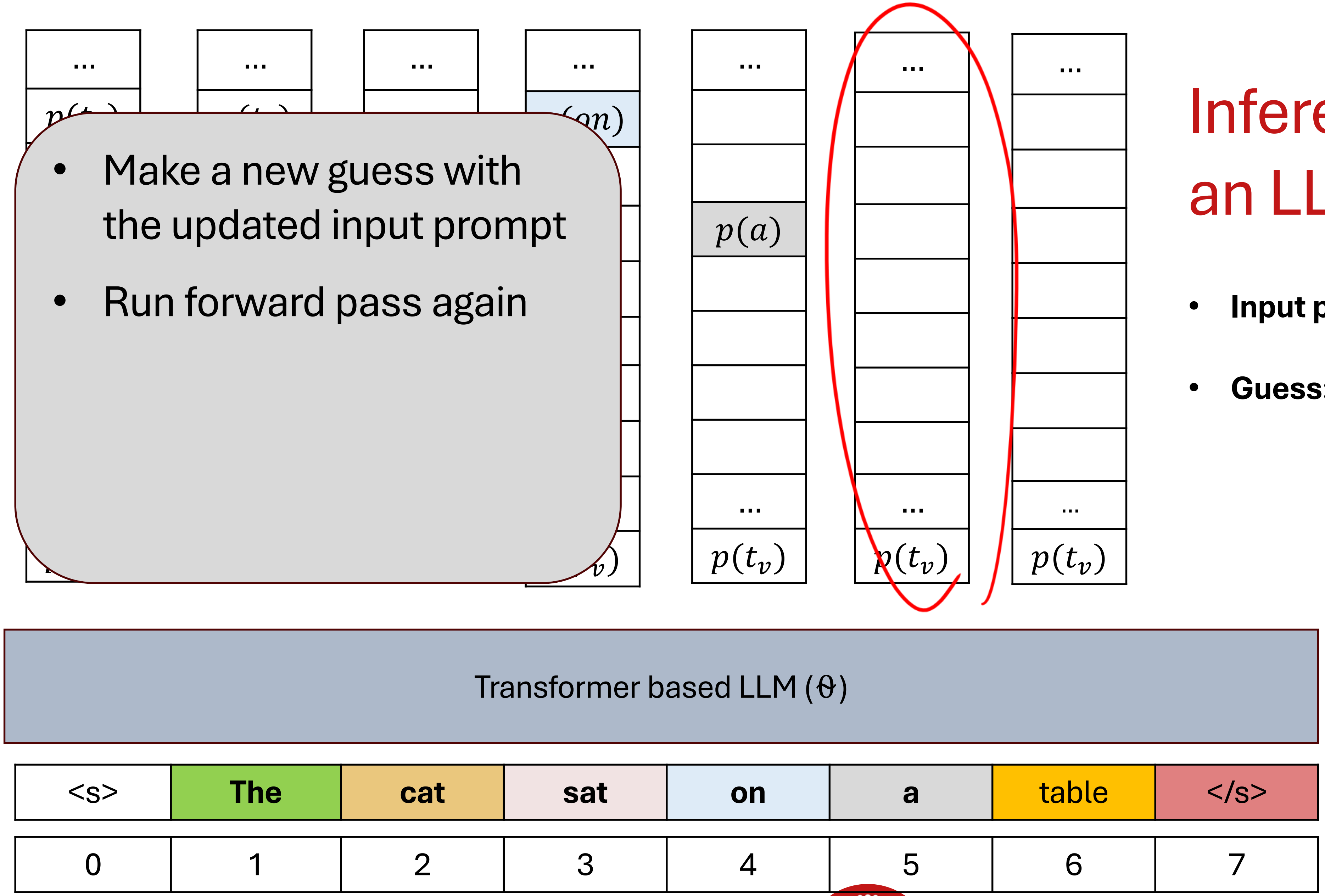
Transformer based LLM (ϑ)

<s>	**The**	**cat**	**sat**	**on**	**a**	table	</s>
0	1	2	3	4	5	6	7

Inference through an LLM

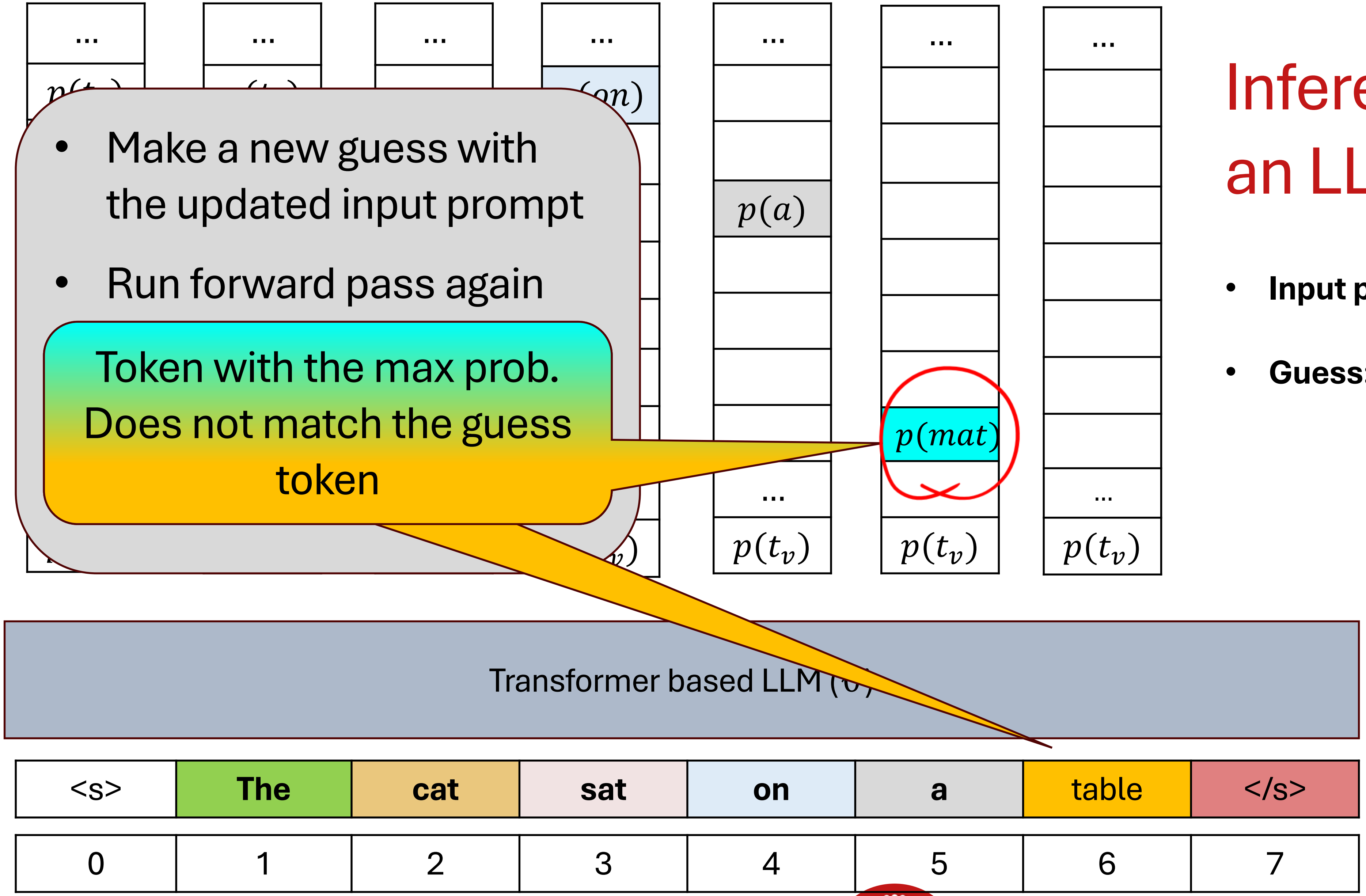
- **Input prompt:** “*The cat sat on a* ”
- **Guess:** “*table </s>*”

- Make a new guess with the updated input prompt
- Run forward pass again



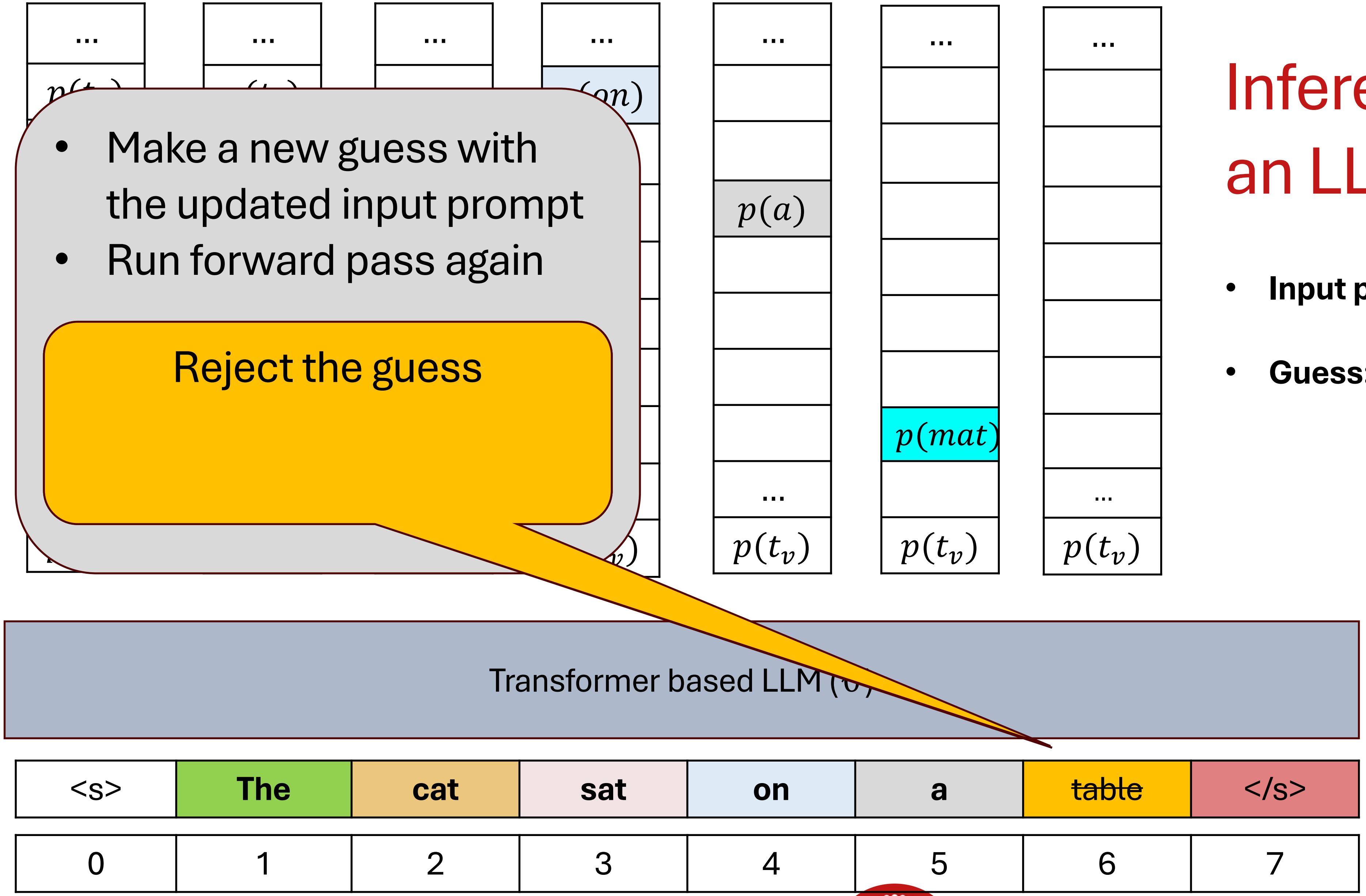
Inference through an LLM

- **Input prompt:** “*The cat sat on a* ”
- **Guess:** “*table </s>*”

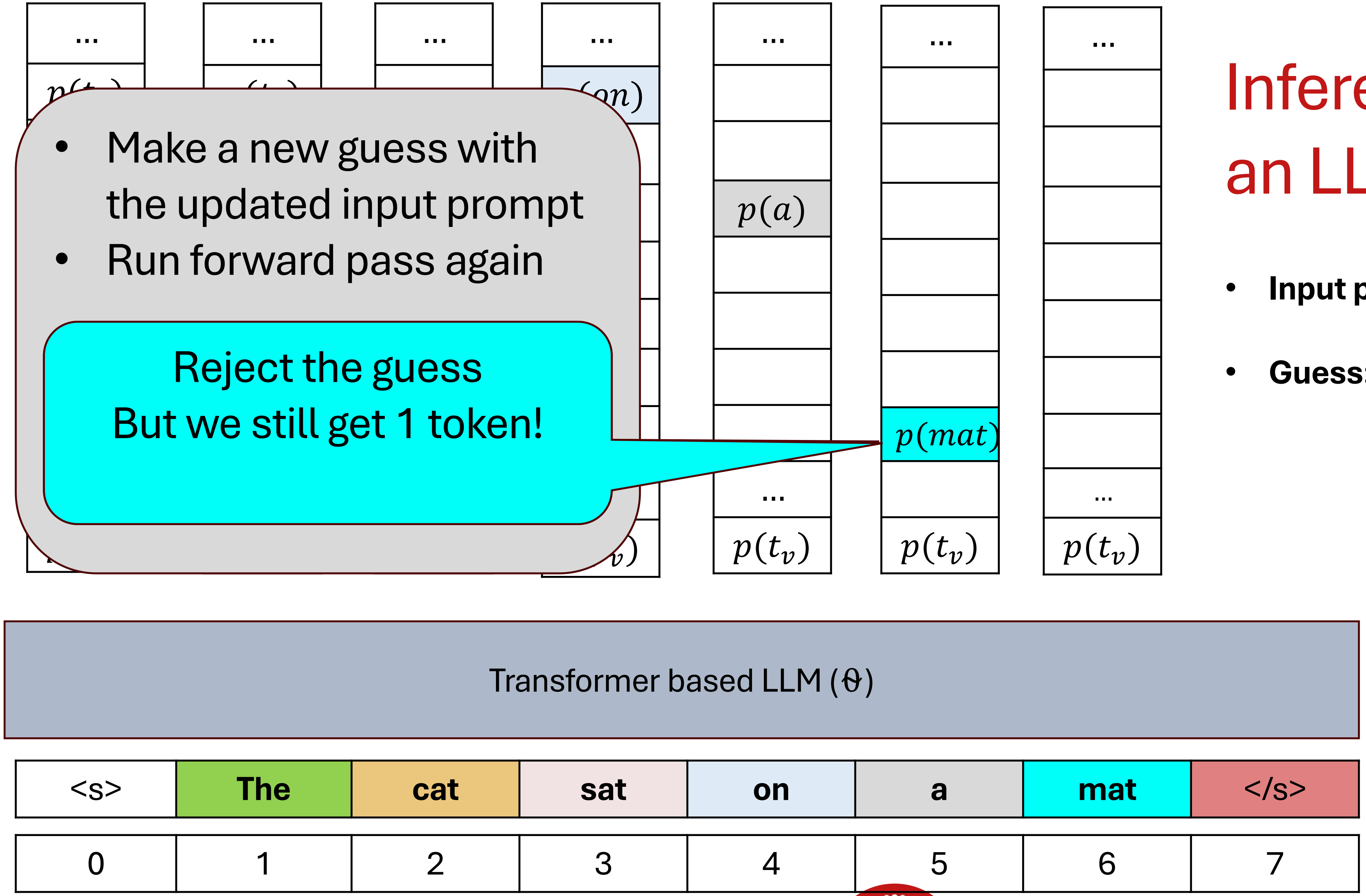


Inference through an LLM

- **Input prompt:** "*The cat sat on a* "
- **Guess:** "*table </s>*"



Inference through an LLM



- **Input prompt:** *"The cat sat on a "*
- **Guess:** *"table </s>"*

Speculative decoding

- ✓ Guess – “*on the chair</s>*”
- ✓ Verify
  - ✓ Accept: “*on*”
  - ✓ Reject: “*~~the chair </s>~~*”
- ✓ Repeat with the updated prompt:

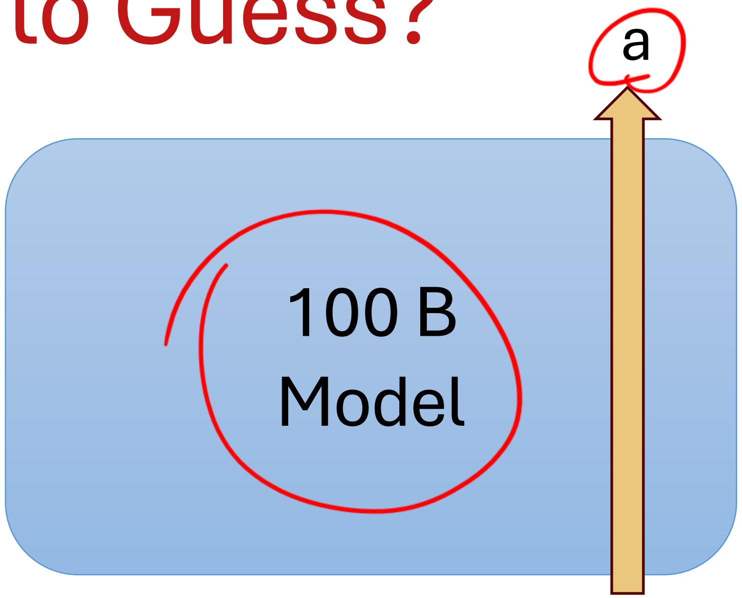
Input prompt: “*The cat sat* ”

Token selected by the model in place of the 1st rejected token

“*The cat sat on a*”

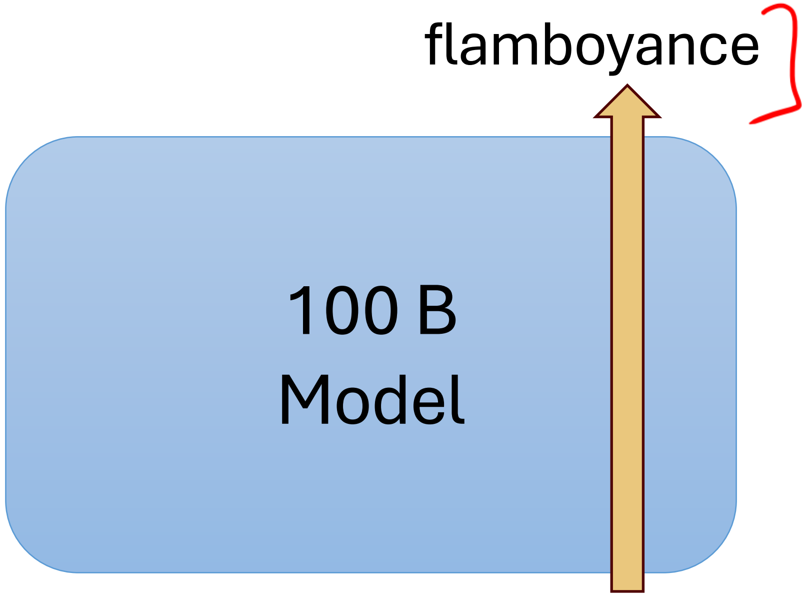
How to guess?

How to Guess?



A group of flamingos
is called …

Very easy



A group of flamingos
is called a …

Difficult

Content credit: https://youtu.be/S-8yr_RibJ4?si=-u2dh3PRBwTnXBOZ

How to Guess?

a ~~flock~~

100 B
Model

A group of flamingos
is called …

Very easy

Can use a small "*draft*" model to guess!

flamboyance

100 B
Model

A group of flamingos
is called a …

Difficult

Verify & correct it using the "***target*** model"

Content credit: https://youtu.be/S-8yr_RibJ4?si=-u2dh3PRBwTnXBOZ

```
def below_threshold(l: list, t: int):
    """Return True if all numbers in the list l are below threshold t.
    >>> below_threshold([1, 2, 4, 10], 100)
    True
    >>> below_threshold([1, 20, 4, 10], 5)
    False
    """
    if isinstance(l, list):
        return True
    else:
        if t <= l < below_threshold(l, t):
            return True
    else:
        # If the first l element of l is an integer, then it is
        # the whole range of integers.
        if not isinstance(l[0], list):
            return True
    else:
        # If the first l element of l is a str, then it is
        # the whole string.
        if hasattr(l, 'findlen'):
            return findlen(l)
        return False


def thresh(t: int, max: int) -> int:
    """Return
```

Example from HumanEval dataset

- Only red tokens are generated by the bigger target model!

Content credits:: Leviathan et al. 2023, Fast Inference from Transformers via Speculative Decoding

Speculative Sampling

- Greedy decoding
  - Target model selection: Token with max. probability
  - Easy to verify with the “proposal” generated by the “draft model”

- But what about sampling by varying – top-p, top-k, or temperature?

Speculative Sampling

DeepMind

2023-2-3

Accelerating Large Language Model Decoding with Speculative Sampling

Charlie Chen[1], Sebastian Borgeaud[1], Geoffrey Irving[1], Jean-Baptiste Lespiau[1], Laurent Sifre[1] and John Jumper[1]
[1]All authors from DeepMind

We present speculative sampling, an algorithm for accelerating transformer decoding by enabling the generation of multiple tokens from each transformer call. Our algorithm relies on the observation that the latency of parallel scoring of short continuations, generated by a faster but less powerful draft model, is comparable to that of sampling a single token from the larger target model. This is combined with a novel modified rejection sampling scheme which preserves the distribution of the target model within hardware numerics. We benchmark speculative sampling with Chinchilla, a 70 billion parameter language model, achieving a 2–2.5× decoding speedup in a distributed setup, without compromising the sample quality or making modifications to the model itself.

Fast Inference from Transformers via Speculative Decoding

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

Abstract

large autoregressive models like
slow - decoding K tokens takes
the model. In this work we in-
tive decoding - an algorithm to
oregressive models faster *without*
he outputs, by computing several
l. At the heart of our approach lie
that (1) hard language-modeling
de easier subtasks that can be ap-
l by more efficient models, and

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 re-

Google Research

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

M_p = draft model
M_q = target model

meta-llama/Llama-2-7b-chat-hf
meta-llama/Llama-2-70b-chat-hf

Algorithm

pf = prefix, K = 5 tokens

$$\tilde{x} = \boxed{p(x \mid x_1 \ldots x_{t-1})}$$
$$\boxed{q(x \mid x_1 \ldots x_{t-1})} \; ?$$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Algorithm

M_p = draft model

M_q = target model

meta-llama/Llama-2-7b-chat-hf

meta-llama/Llama-2-70b-chat-hf

pf = prefix, $K = 5$ tokens

$$p_1(x) = M_p(pf) \xrightarrow{\text{Sample}} x_1$$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Algorithm

M_p = draft model

M_q = target model

meta-llama / Llama-2-7b-chat-hf

meta-llama / Llama-2-70b-chat-hf

pf = prefix, K = 5 tokens

$p_1(x) = M_p(pf) \longrightarrow x_1$

$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$

...

$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

Run draft model for K steps

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

$$\underline{q_1(x)}, \underline{q_2(x)}, q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Run target model once

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

$p_1(x) = M_p(pf) \longrightarrow x_1$

$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$

Run draft model for K steps

...

$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$

A distribution at each step over entire vocabulary

$$\overbrace{q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)}$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Run target model once

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x) ?	0.9	0.8	0.8	0.3	0.8

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Rejection Sampling

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

$\frac{.9}{.8} > 1$

$\text{Accept}(x) = \text{Prob}\left(\frac{q(x)}{p(x)}\right)$

$\frac{.8}{.7} > 1$ ✓

$\frac{.8}{.9} \approx .9$

Rejection Sampling

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Rejection Sampling

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

Rejection Sampling

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Similar to Importance Sampling

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$

$p_1(x) = M_p(pf) \longrightarrow x_1$

$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$

...

$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$

	Token	x1	x2	x3	x4	x5
		dogs	love	chasing	after	cars
Draft Model	p(x)	0.8	0.7	0.9	0.8	0.7
Target Model	q(x)	0.9	0.8	0.8	0.3	0.8
		✓	✓	✓	✗	

$q_1(x), q_2(x), q_3(x), \boxed{q_4(x)}, q_5(x), q_6(x)$

$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$

Content credits: https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV

Rejection Sampling

Actually, don't sample $q(x)$ $\times$ $q_4(x)$

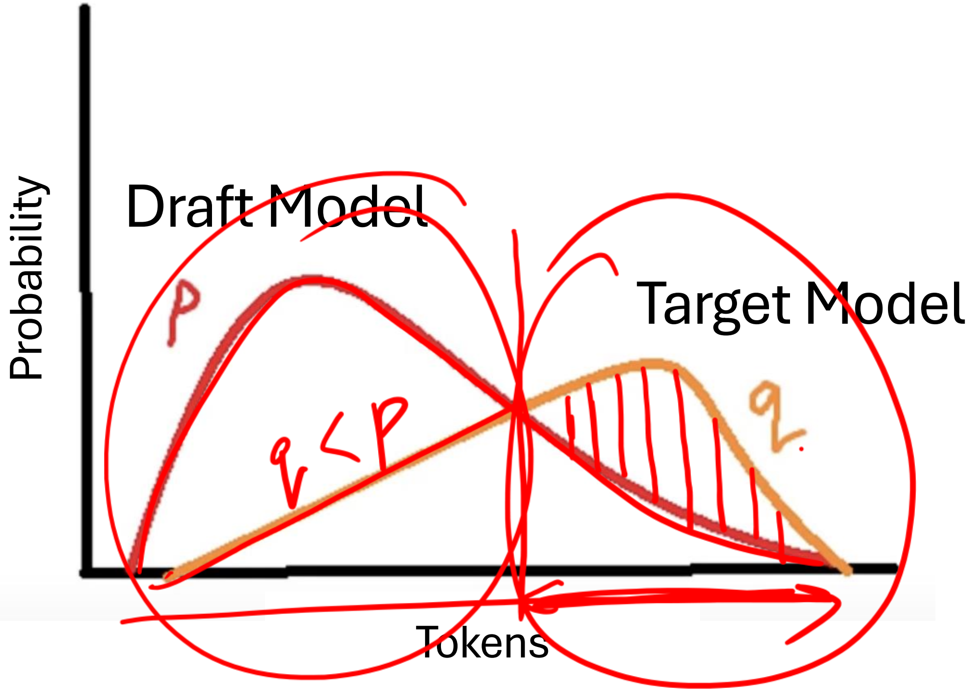
Adjusted distribution: $\left(q(x) - p(x)\right)_+$

(Target Model -- Draft Model)+

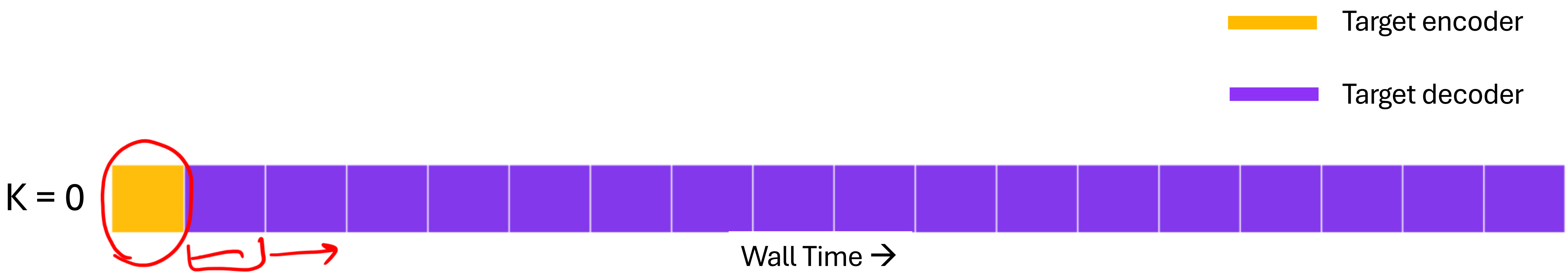
Rejection Sampling

Actually, don't sample $q(x)$

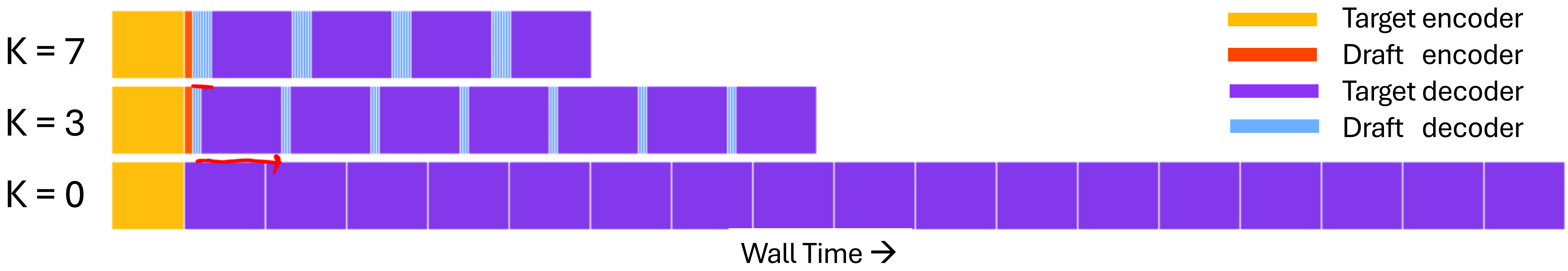
Adjusted distribution: $\left(q(x) - p(x)\right)_+$



Wall time speedup:
Illustration on an encoder-decoder model



Wall time speedup:
Illustration on an encoder-decoder model



Results

Model	d_{model}	Heads	Layers	Params
Target (Chinchilla)	8192	64	80	70B
Draft	6144	48	8	4B

Table 1 | **Chinchilla performance and speed on XSum and HumanEval with naive and speculative sampling at batch size 1 and** $K = 4$. XSum was executed with nucleus parameter $p = 0.8$, and HumanEval with $p = 0.95$ and temperature 0.8.

Sampling Method	Benchmark	Result	Mean Token Time	Speed Up
ArS (Nucleus)	XSum (ROUGE-2)	0.112	14.1ms/Token	1×
SpS (Nucleus)		0.114	7.52ms/Token	1.92×
ArS (Greedy)	XSum (ROUGE-2)	0.157	14.1ms/Token	1×
SpS (Greedy)		0.156	7.00ms/Token	2.01×
ArS (Nucleus)	HumanEval (100 Shot)	45.1%	14.1ms/Token	1×
SpS (Nucleus)		47.0%	5.73ms/Token	2.46×

2x

How to guess?

- **Speculative decoding:**
  - Smaller model from the same family – Draft model: Llama-7B, for target model: Llama-70B
  - Is 7B small enough?

How to guess?

- **Speculative decoding:**
  - Smaller model from the sa
  - Is 7B small enough?

How to guess?

- **Speculative decoding:**
  - Smaller model from the same family – Draft model: Llama-7B, for target model: Llama-70B
  - Is 7B small enough?
  - Is it easy to host two models?

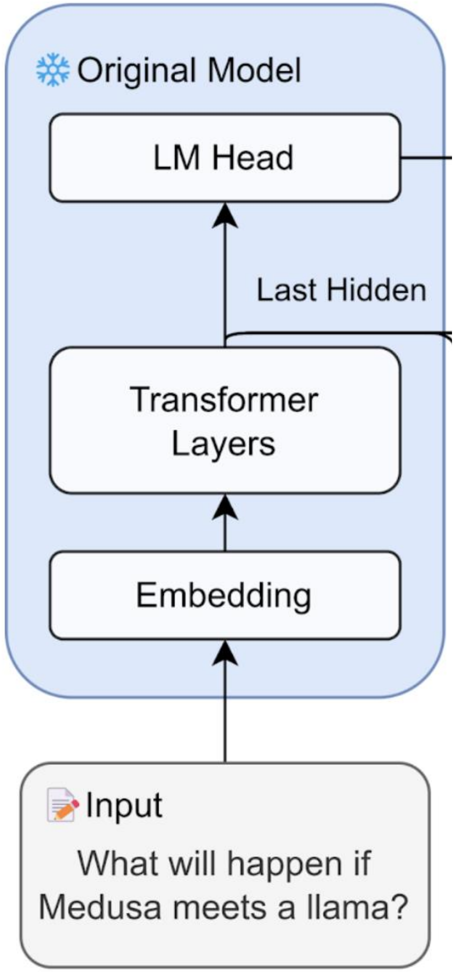
- Can we somehow generate multiple candidates from the target model itself?

- What if you are allowed to further fine-tune using PEFT?

Medusa

- Multiple LM heads to predict *next-next* tokens

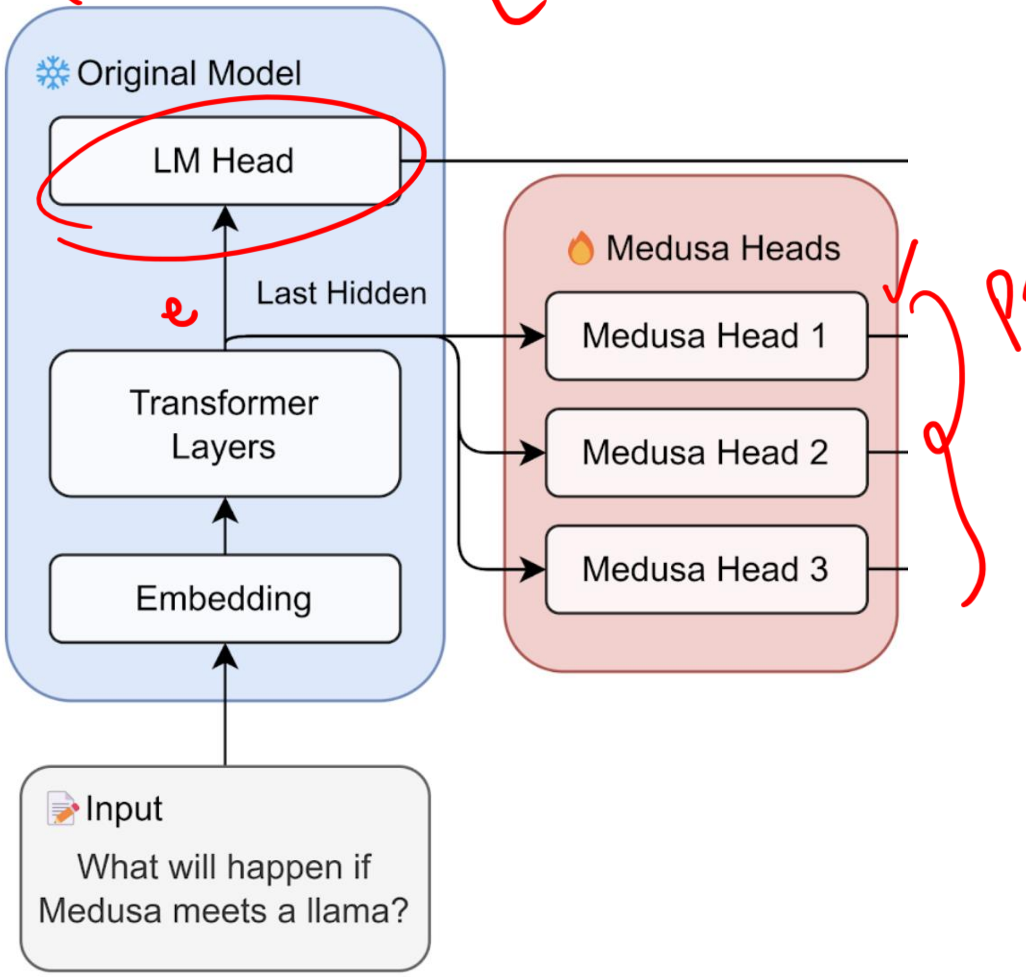
Medusa



- Multiple LM heads to predict *next-next* tokens

Frozen

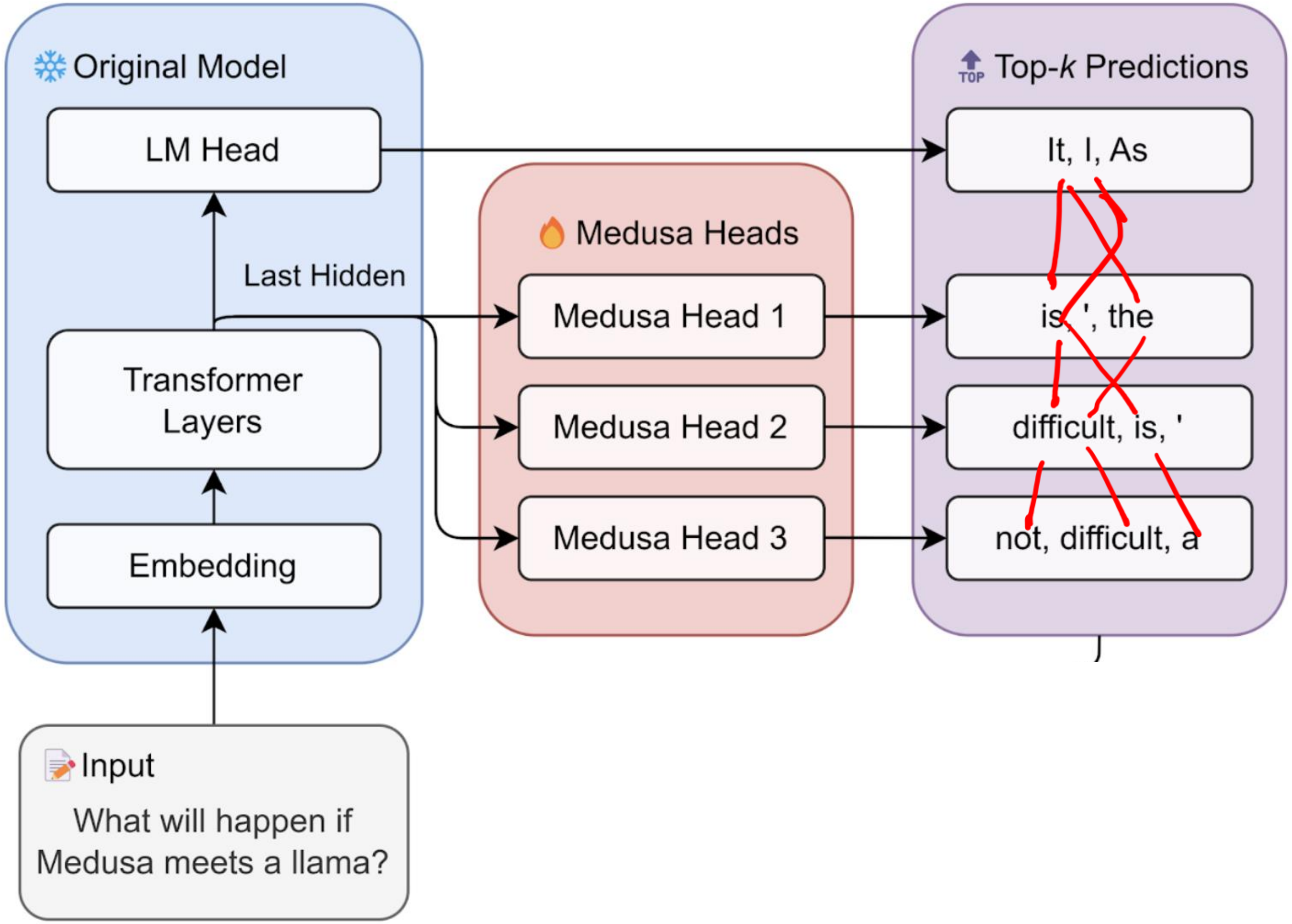
$Me = \begin{bmatrix} \\ \\ \end{bmatrix}_{T_V}^{T_1}$



Medusa

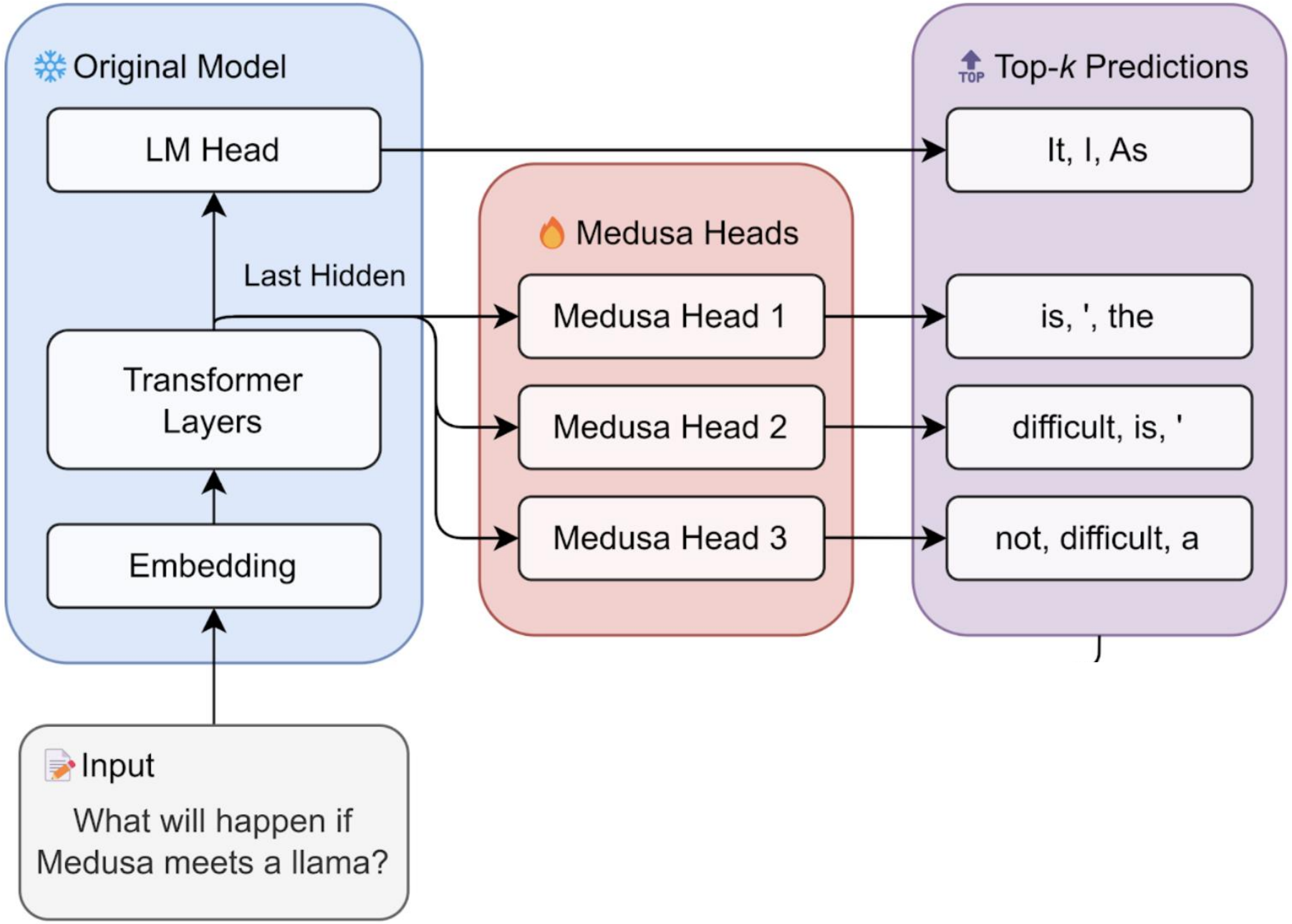
- Multiple LM heads to predict *next-next* tokens

Medusa

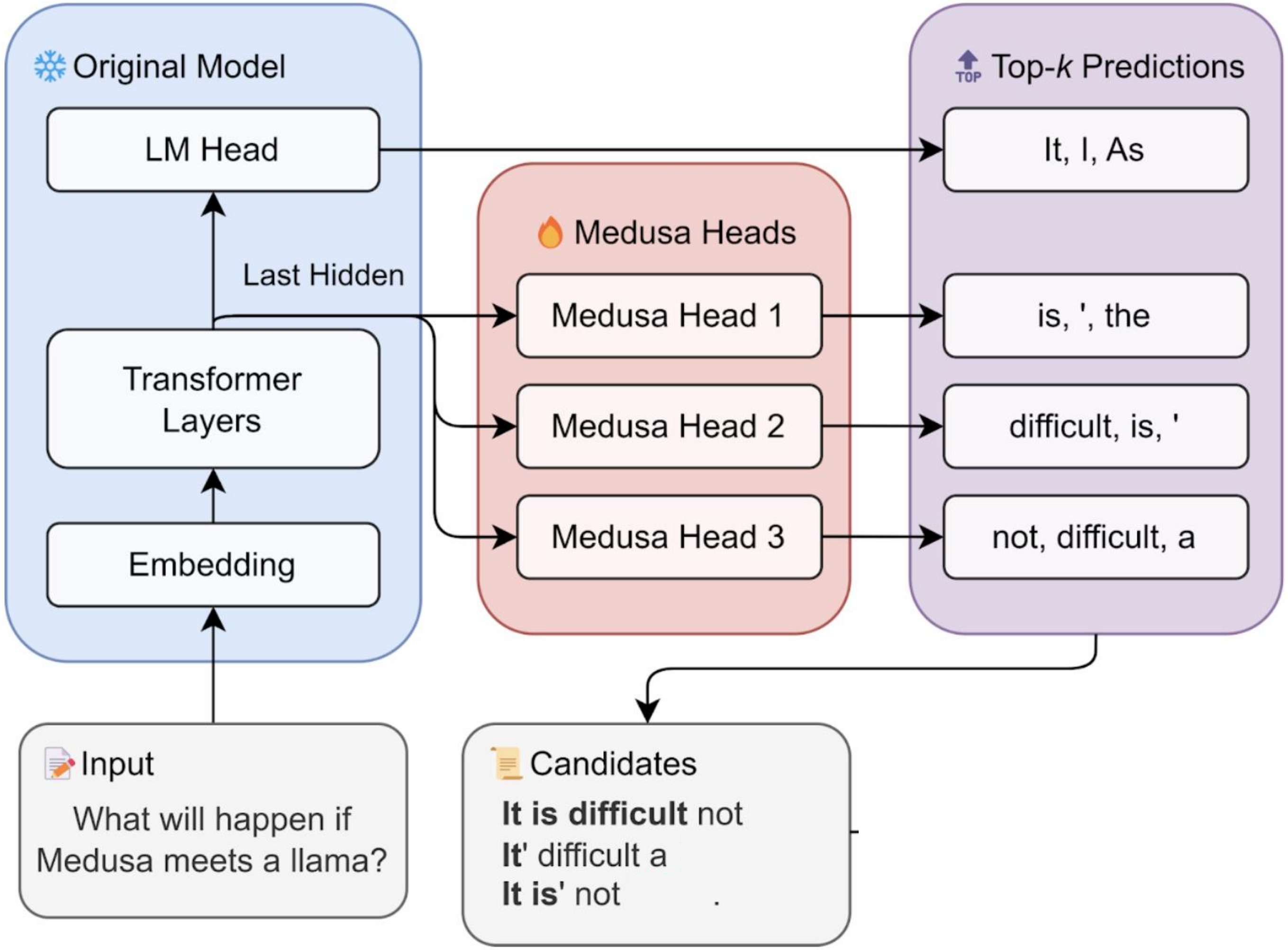


- Multiple LM heads to predict *next-next* tokens

Medusa



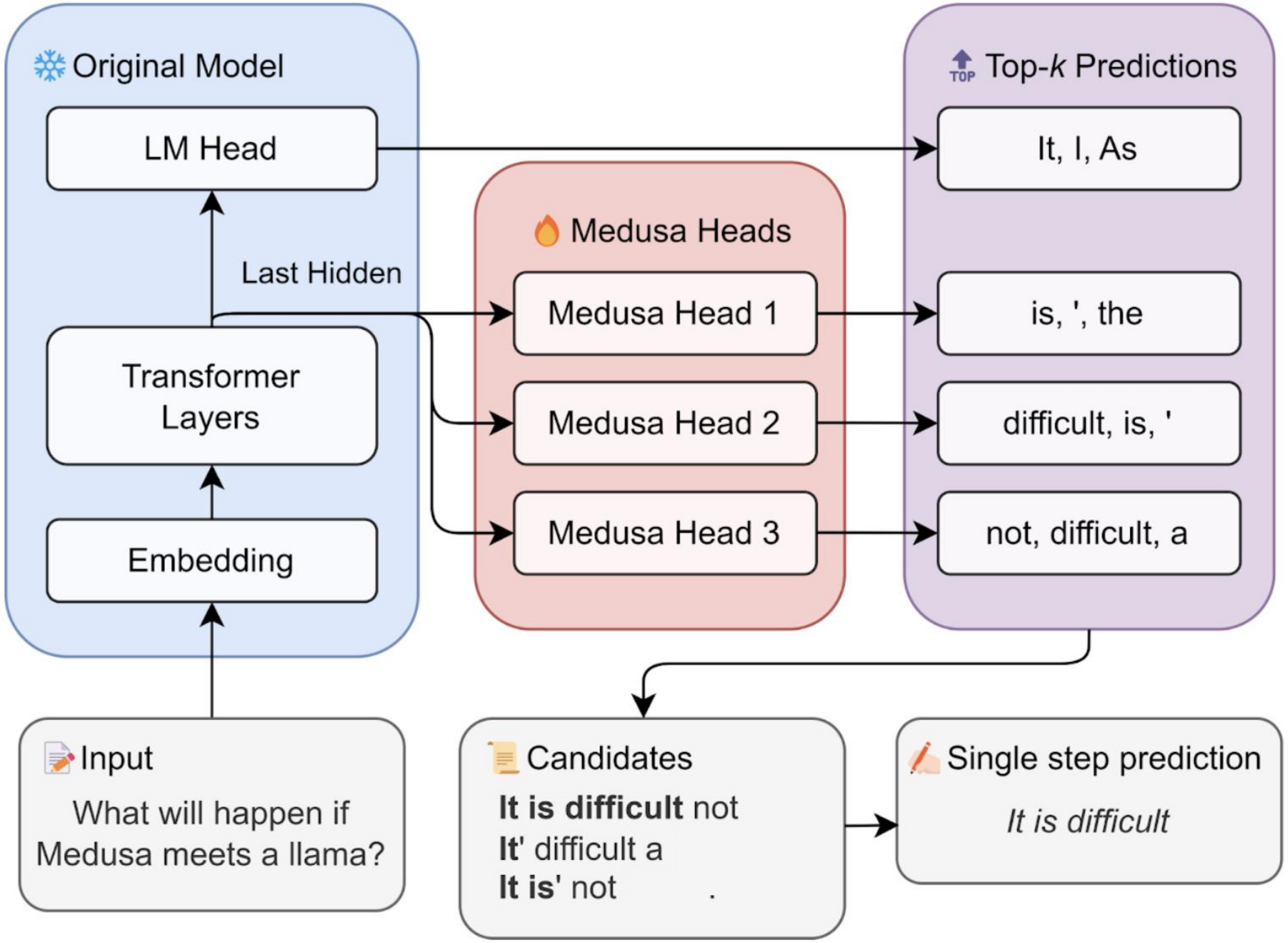
- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates



Medusa

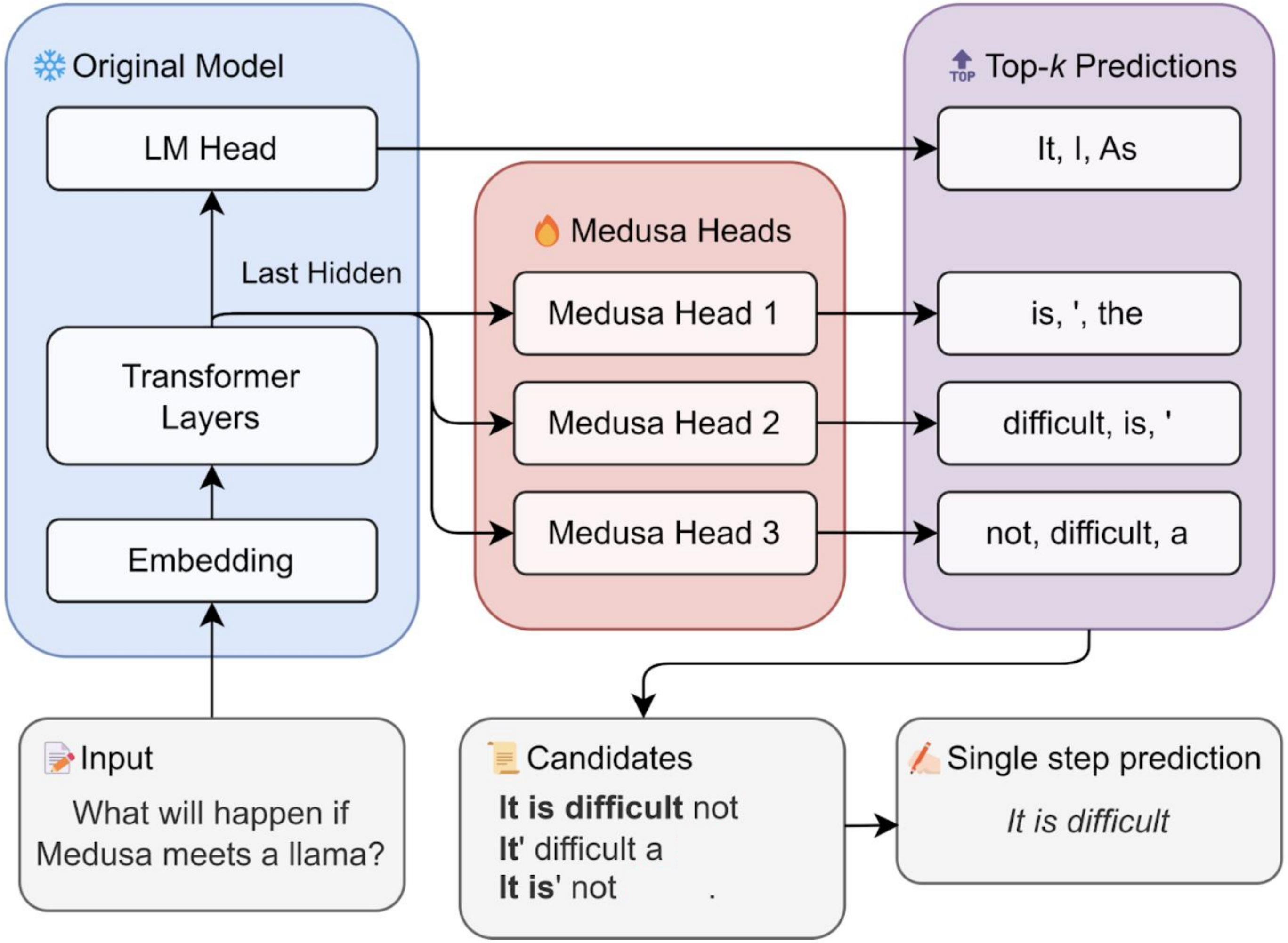
- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates

Medusa



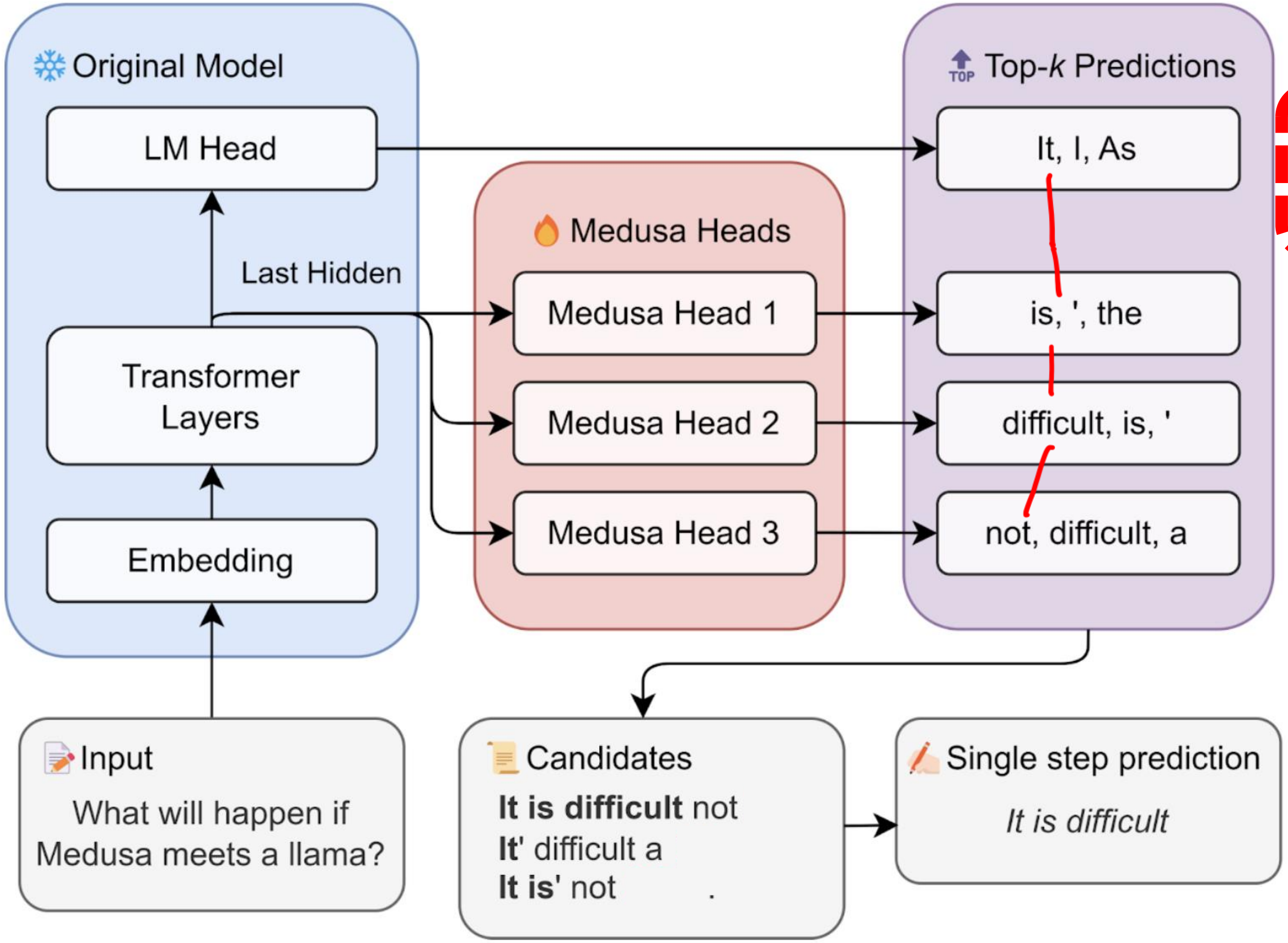
- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention

Medusa



- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)} = 256$ candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

Medusa

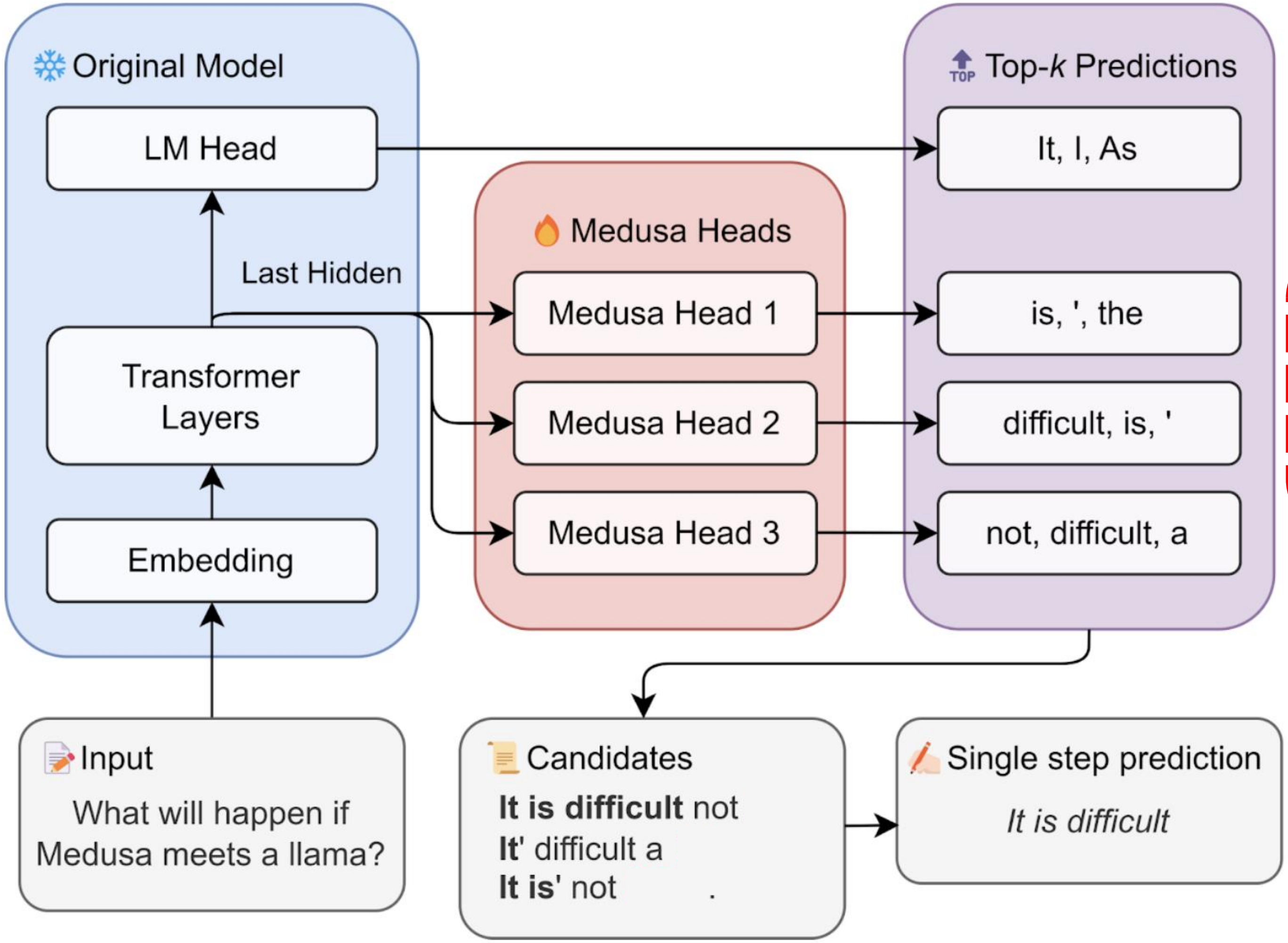


- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)} = 256$ candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

How to train multiple LM heads?

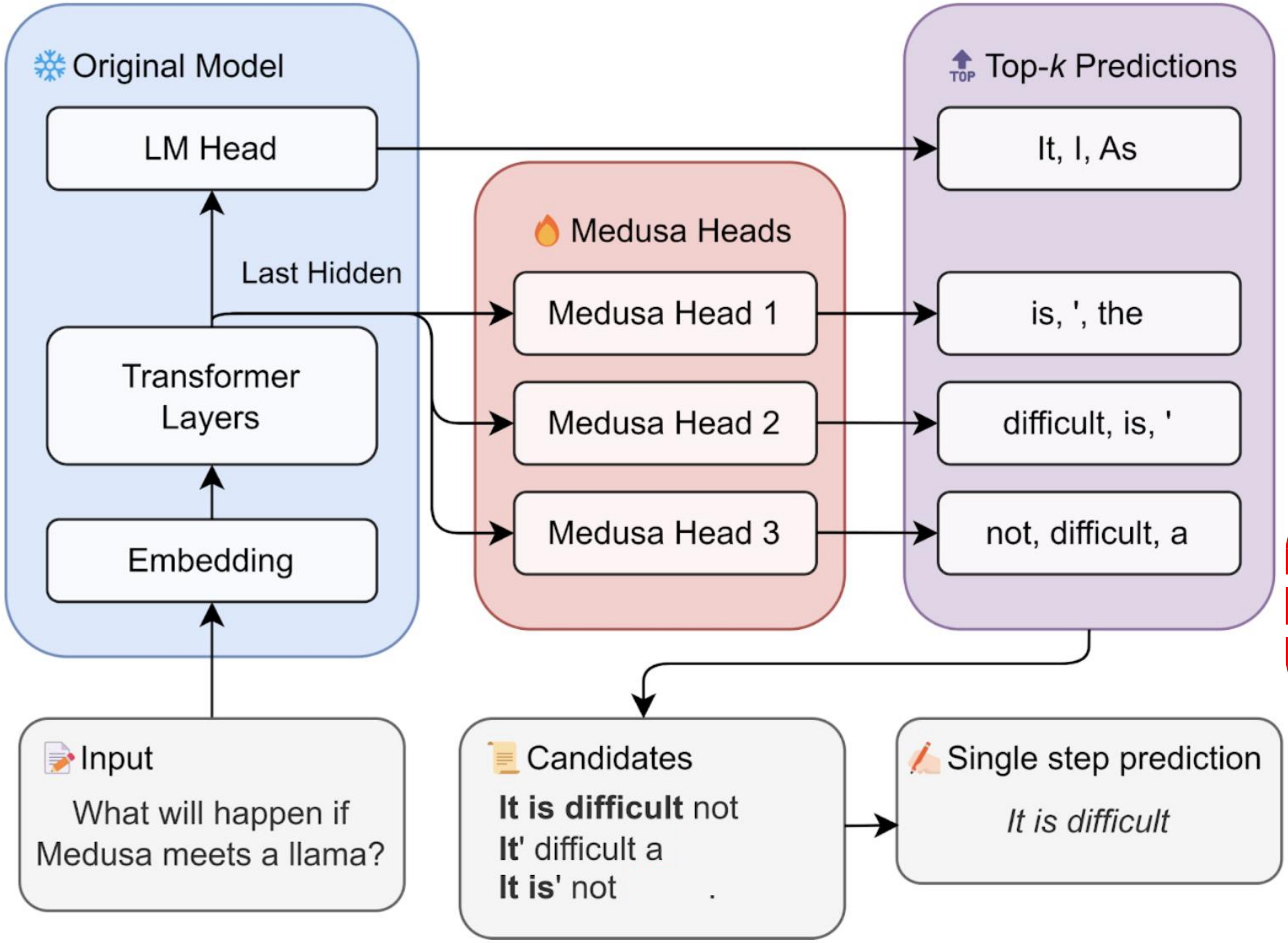
- Each Medusa head is as a single layer of feed-forward network, augmented with a residual connection.
- Keep the backbone architecture frozen and train the heads using PEFT.
- Can use the same corpus that trained the original model.
- On Vicuna-7B, Medusa Head 1 get
  - top-1 accuracy rate of approximately 60%
  - Top-5 accuracy rate of ~ 80% (hence we use top-k approach)

Medusa

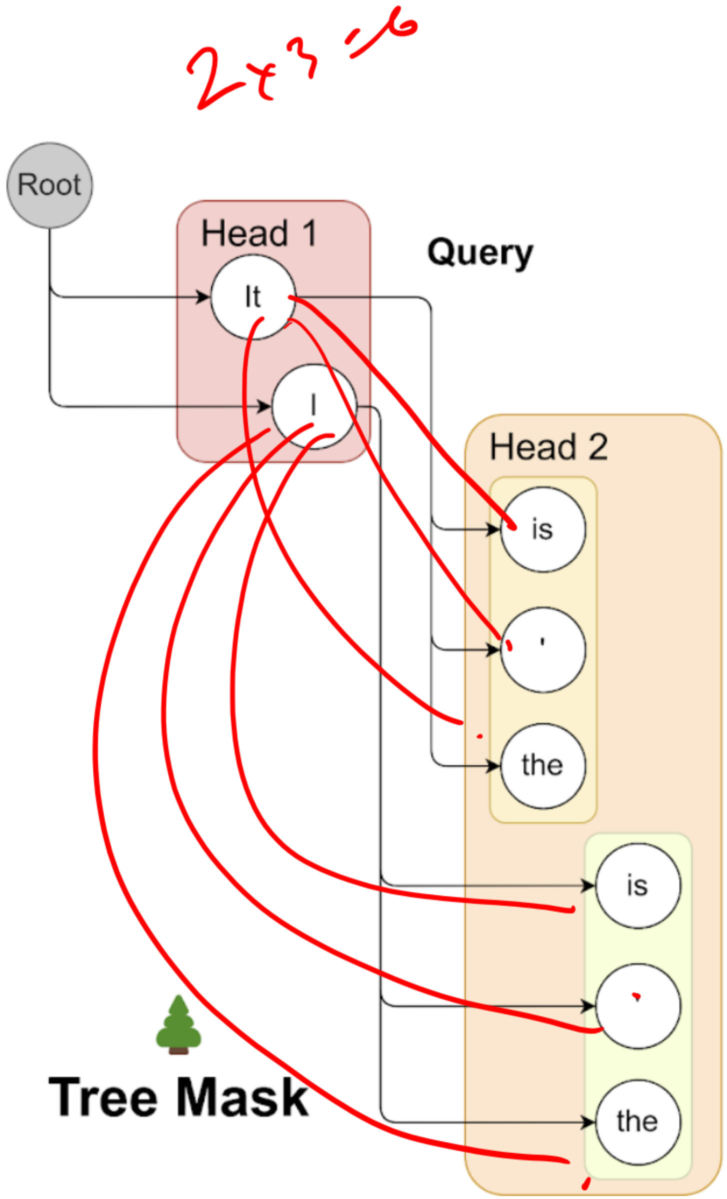


- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

Medusa



- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.



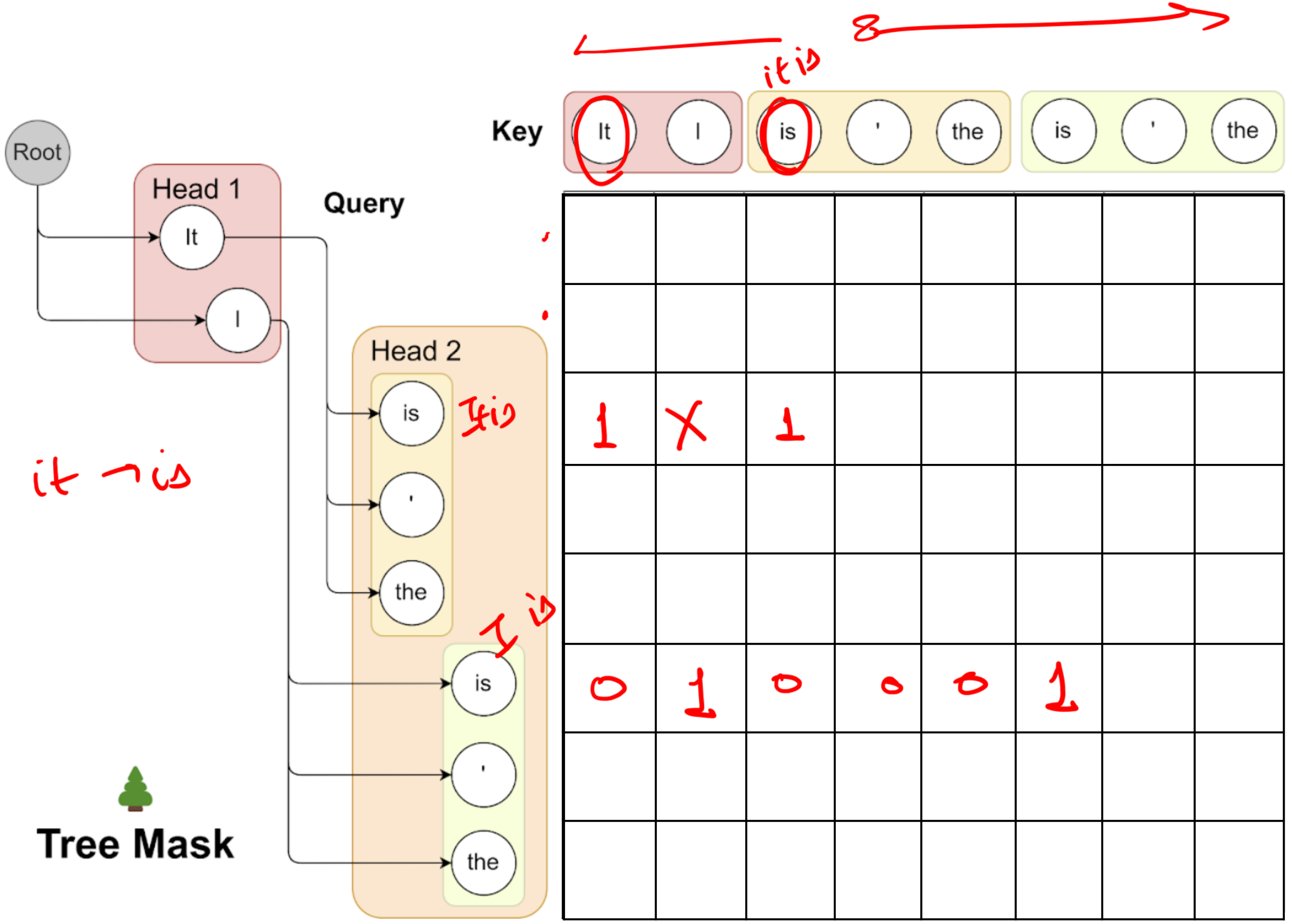
Tree Attention

Head 1: “ It ” “ I ”

Head 2: “ is ” “ ’ ” “ the ”

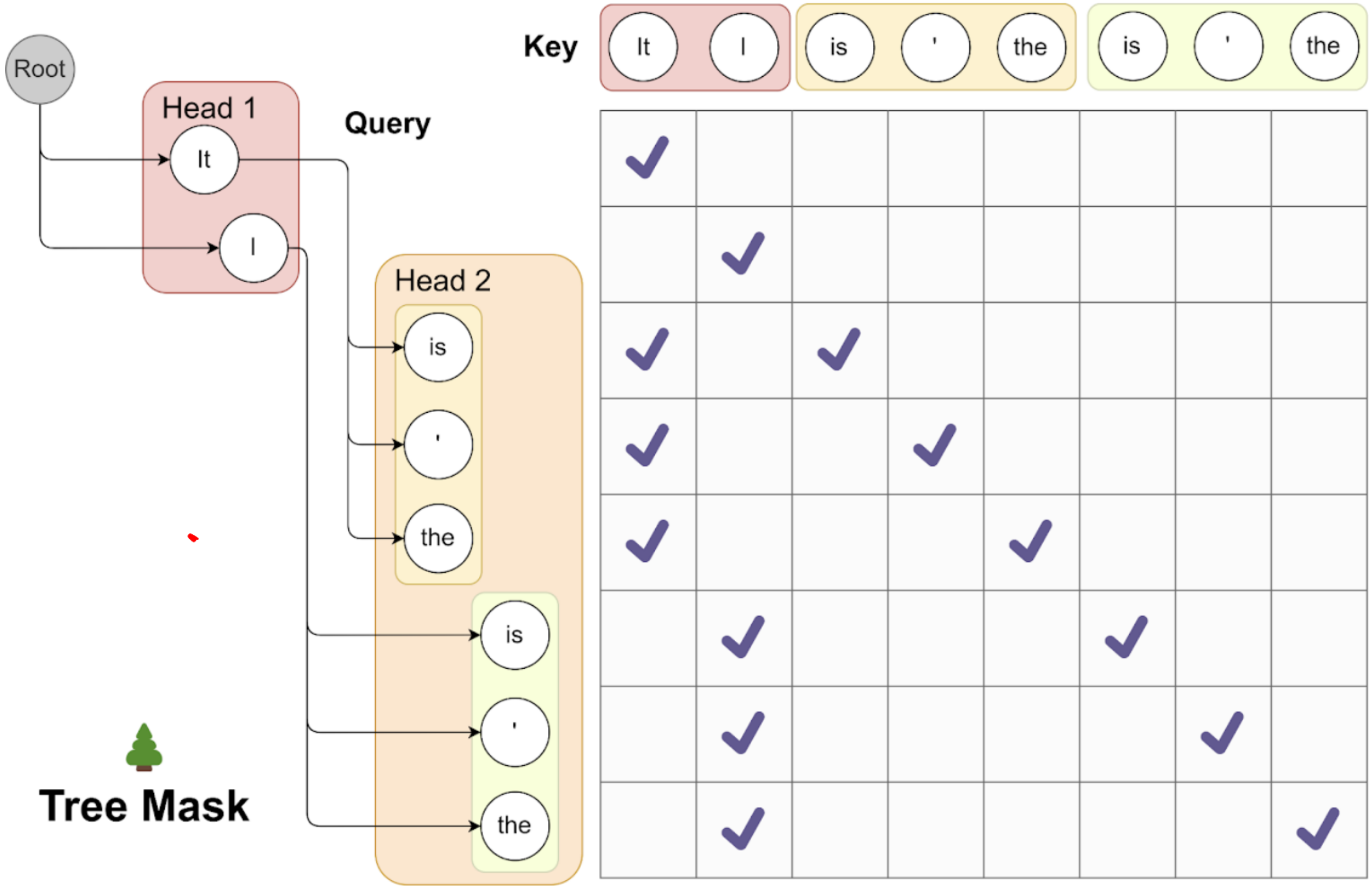
Tree Attention

- Head 1: “ It ” “ I ”
- Head 2: “ is ” “ ’ ” “ the ”



It
I
It is
It '
It the
I

8



Tree Attention

- Head 1: “ It ” “ I ”
- Head 2: “ is ” “ ’ ” “ the ”
- Attention mask exclusively permits attention flow from the current token back to its antecedent tokens.
- The positional indices for positional encoding are adjusted in line with this structure.

It is

Prune the tree!

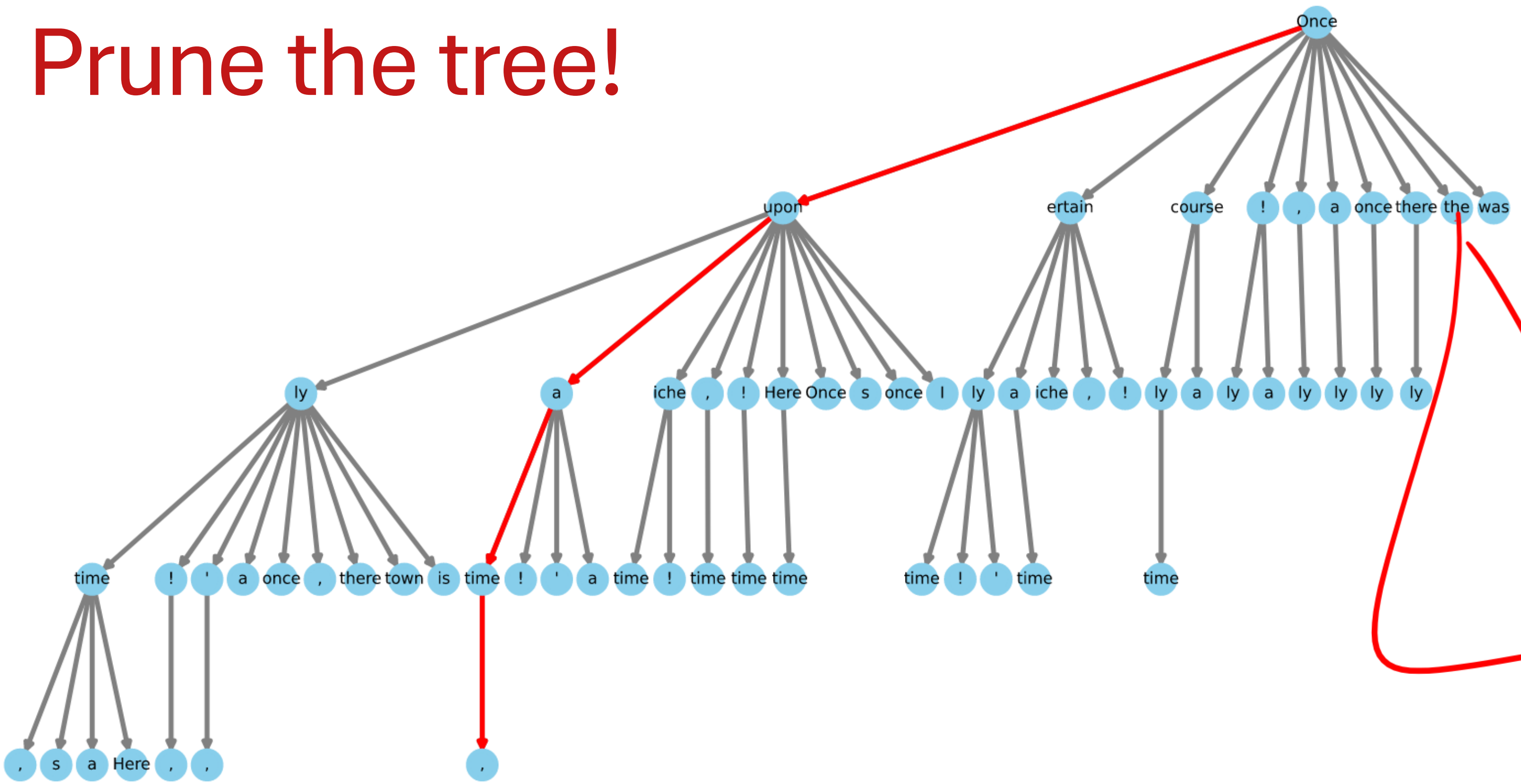
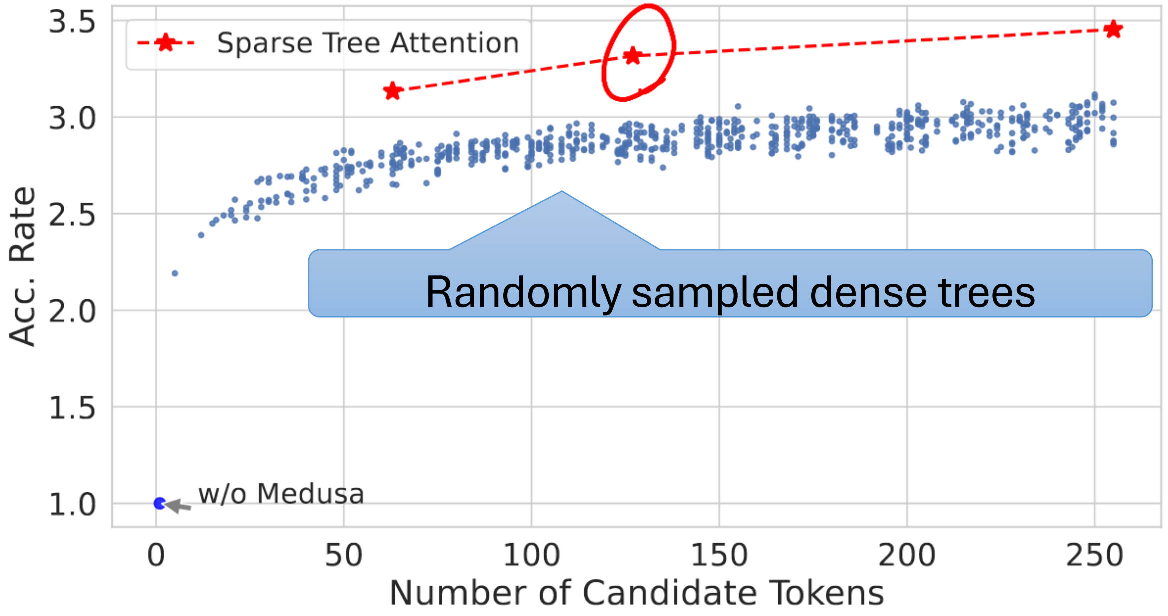


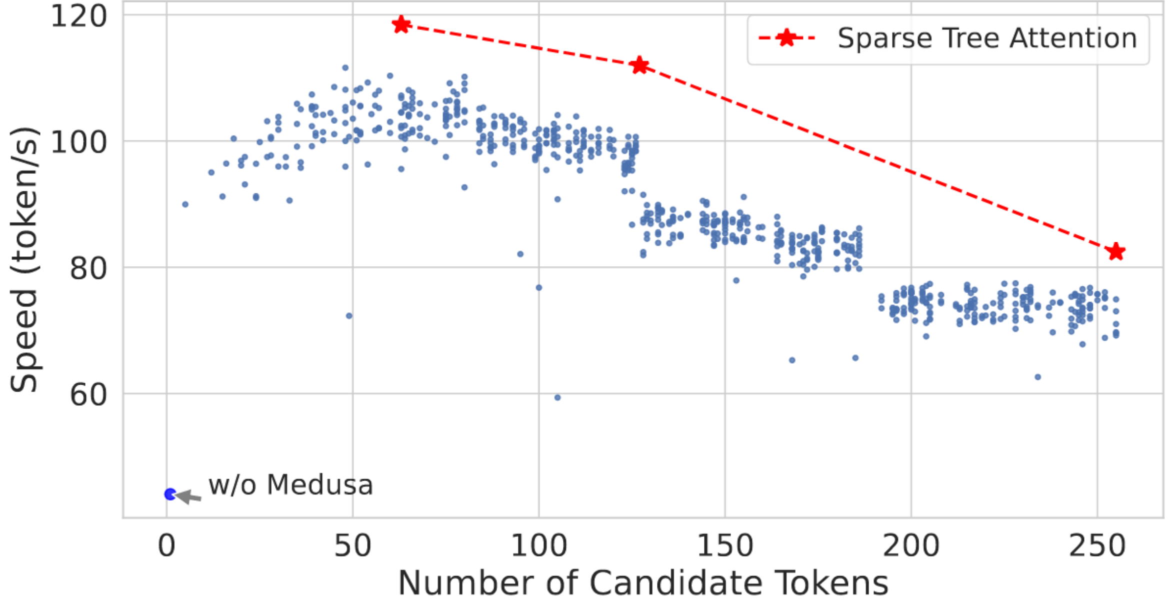
Figure 6. Visualization of a sparse tree setting for MEDUSA-2 Vicuna-7B. The tree has 64 nodes representing candidate tokens and a depth of 4 which indicates 4 MEDUSA heads involved in calculation. Each node indicates a token from a top-k prediction of a MEDUSA head, and the edges show the connections between them. The red lines highlight the path that correctly predicts the future tokens.

- Cartesian product is expensive.
- Based on expected top-k accuracy for each head, create a static tree

Prune the tree!



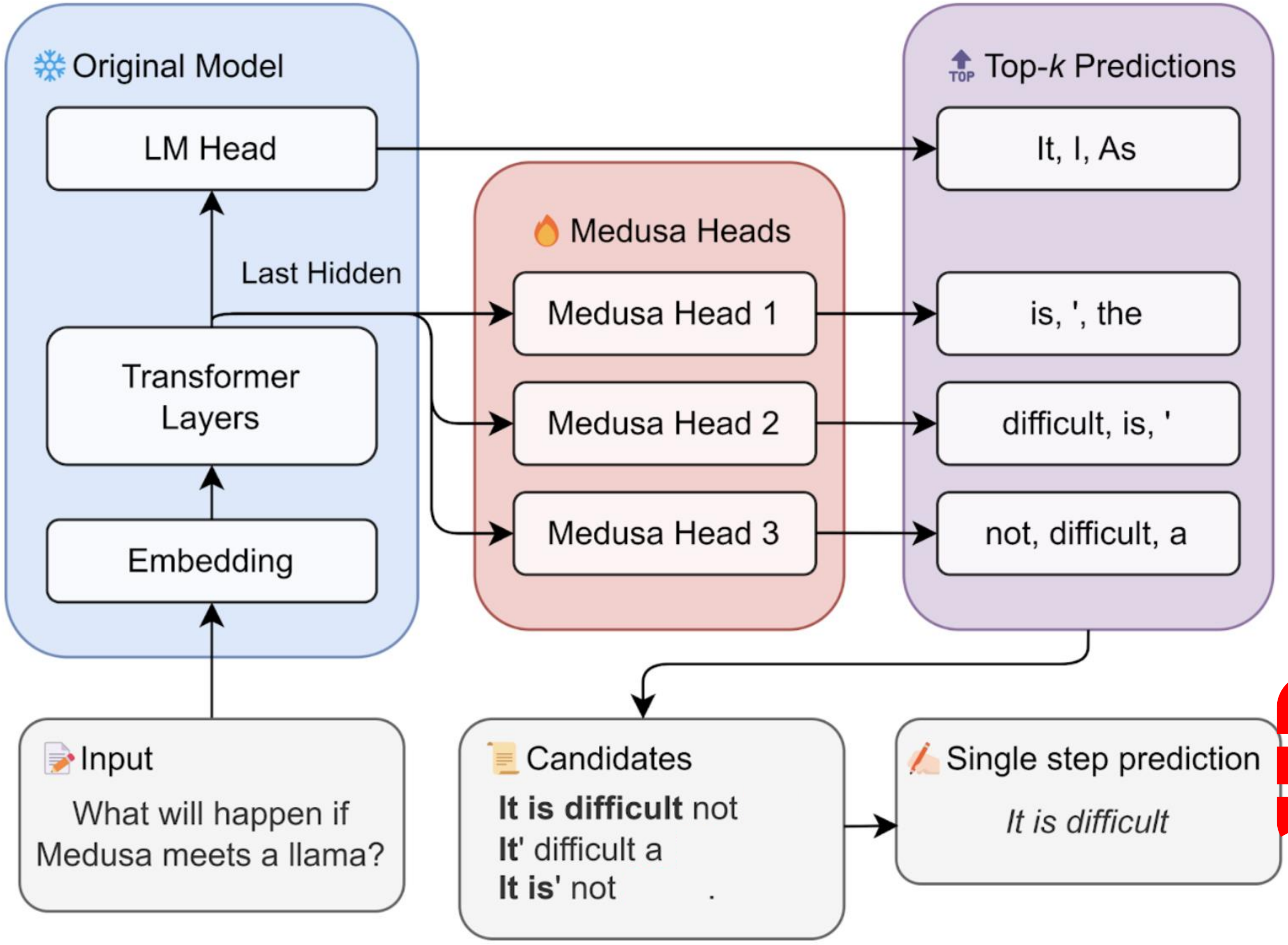
acceleration rate



speed (tokens/s)

Randomly sampled dense trees

Medusa



- Multiple LM heads to predict *next-next* tokens
- Take the Cartesian product to create multiple potential candidate sequences
  - With top-k=4, and 3 heads, we get $4^{(3+1)}$= 256 candidates
- Process all the candidates in parallel
  - Enabled by Tree attention
- Accept the "*largest*" sub-sequence above a threshold prob.

Acceptance criteria

- Device their own sampling method, instead of supporting standard nucleus sampling
- Aim to pick candidates that are likely enough according to the original model
- Always select the 1st token greedily
- For the rest of the tokens:

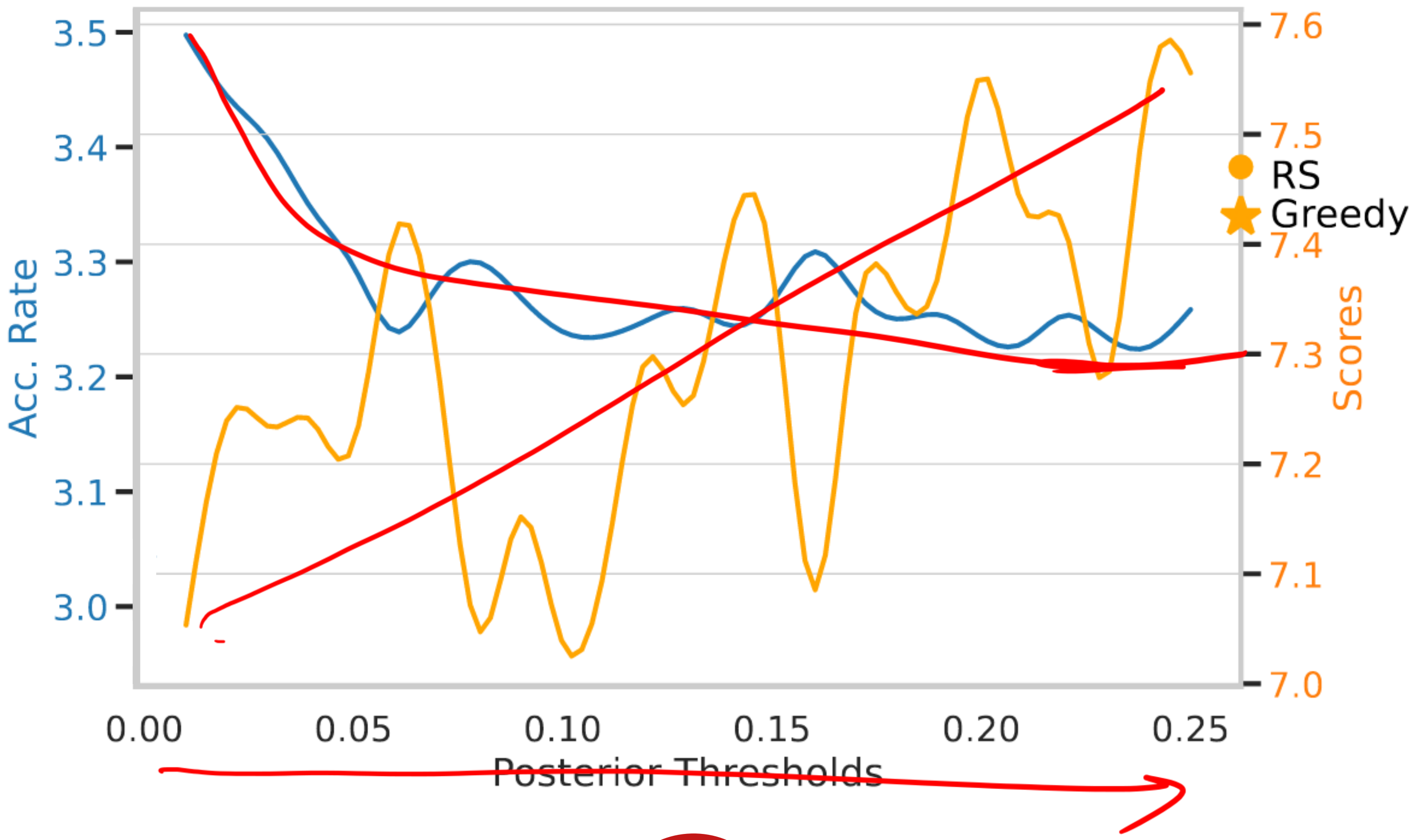
$$p_{\text{original}}(x_{n+k} | x_1, x_2, \cdots, x_{n+k-1}) > \min\left(\epsilon, \delta \exp\left(-H(p_{\text{original}}(\cdot | x_1, x_2, \cdots, x_{n+k-1}))\right)\right),$$

Minimum of a hard threshold and an entropy-dependent threshold

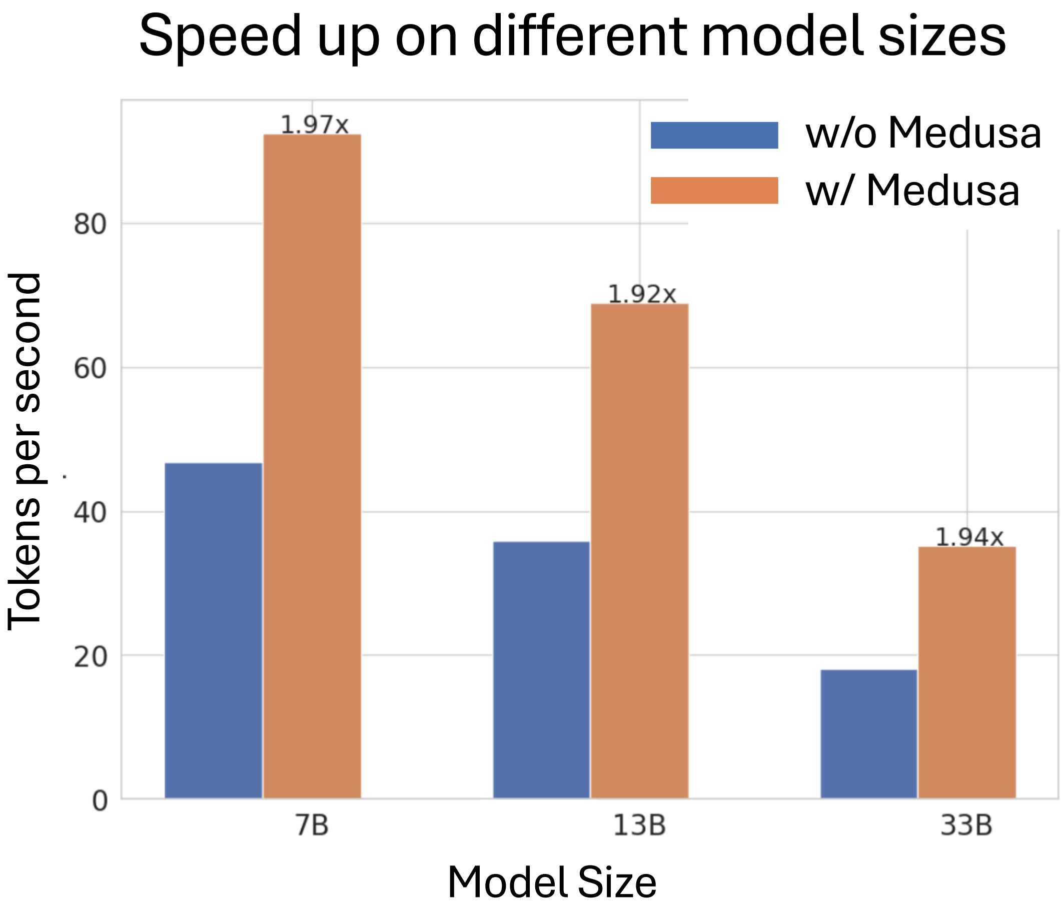
Entropy

- Select the longest sub-sequence in which all tokens satisfy the above criteria

Impact of the threshold



Results


Speed up on different model sizes
w/o Medusa
w/ Medusa
1.97x
1.92x
1.94x
80
60
40
20
0
Tokens per second
7B
13B
33B
Model Size

How to guess?

- **Speculative decoding** -- uses a small draft model with same tokenizer
- **Medusa –** trains multiple LM heads to predict next-next tokens

Think about tasks like

- Content grounded QA,
- RAG,
- Summarization…

Where should you look for potential candidate completions?

Prompt Lookup Decoding

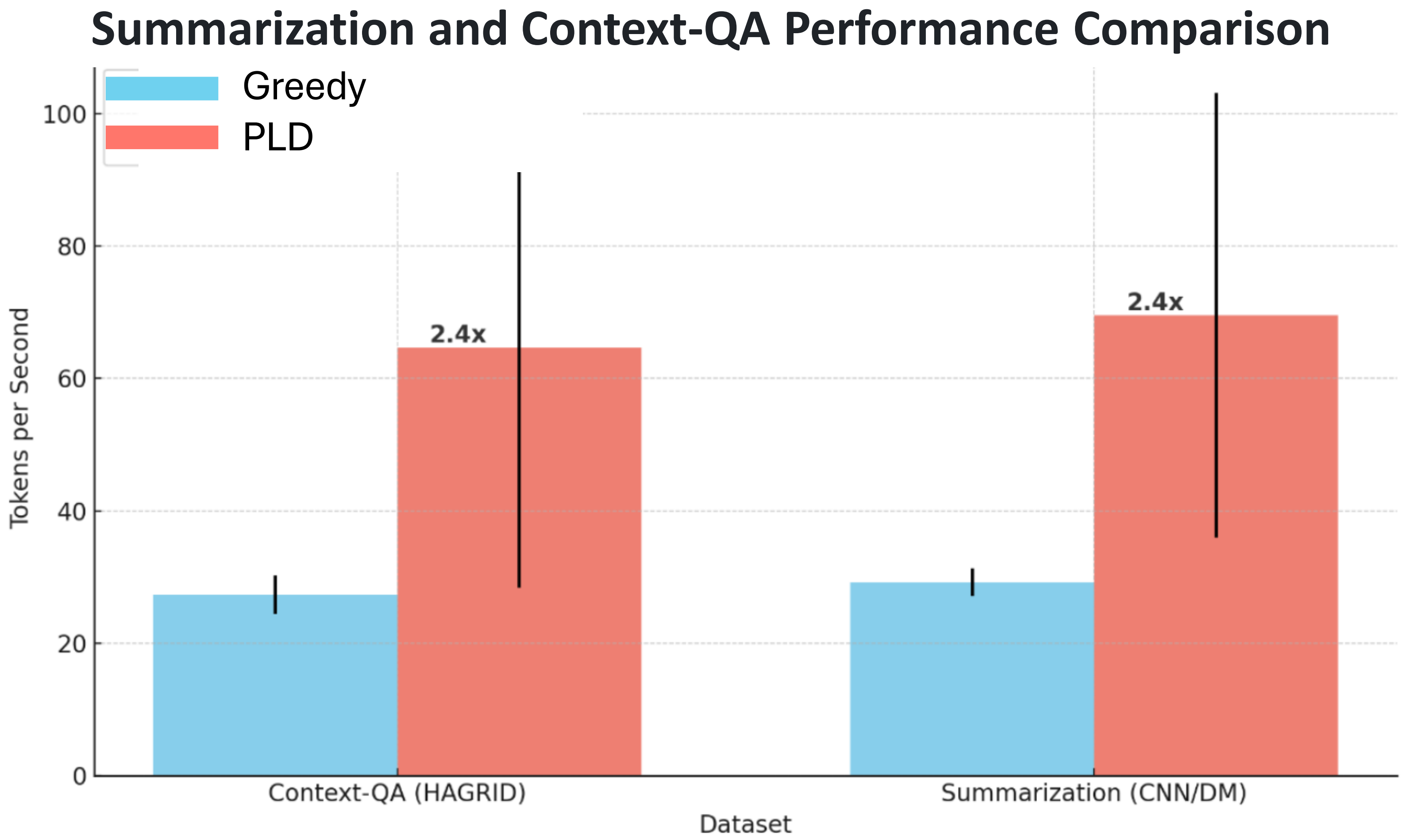
Prompt-lookup decoding

```
print(f"Tokens per second: {tokens_per_sec} tokens/sec")
print(f"Total tokens generated: {num_tokens_generated}")
```

Greedy decoding

Content credits: https://g

Results



Summarization and Context-QA Performance Comparison
Greedy
PLD
Tokens per Second
100
80
60
40
20
0
2.4x
2.4x
Context-QA (HAGRID)
Summarization (CNN/DM)
Dataset

How to guess?

- **Speculative decoding** -- uses a small draft model with same tokenizer
- **Medusa –** trains multiple LM heads to predict next-next tokens
- **Prompt-lookup decoding:** Search for n-grams in the prompt as potential completions

Can we create potential candidates (n-grams)

- Without relying on the input prompt, and
- Without additional finetuning ?

a

x_1 x_2 x_3 x_4 x_5

on $\boxed{\cdot}$ table

Lookahead Decoding

Another way of generating n-gram candidates and verifying them

Prefix "on a"

- No need to train "additional" LM heads for next-next token predictions
- Doesn't rely on input prompt to search for n-grams
- Inspired by Jacobi iteration method
- Starts with a random guess completion and maintains a pool of n-grams generated by the model.
- Heavily relies on tree-attention to verify as well as generate multiple n-gram candidates in parallel, starting from the random guess
- Checkout the blog - https://lmsys.org/blog/2023-11-21-lookahead-decoding

Summary

- **Motivation –** Inference is sequential, memory bound and slow, with high latency
- **KV caching** – avoids re-computation of Keys and Value matrices
- **Paged Attention and vLLM -** efficient memory management
- **Flash decoding –** efficient attention for very long sequences
- **Breaking sequential generation**
  - Speculative decoding – guess and verify paradigm
  - How to guess?
    - Smaller draft model with same tokenizer
    - Medusa

Look ahead decode]

Addresses memory issues

Makes it fast!

Addresses sequential generation

Continuous batching

- Continuous batching
  - ORCA - https://www.usenix.org/conference/osdi22/presentation/yu

Continuous batching

https://www.usenix.org/conference/osdi22/presentation/yu (09/2022)

Available in Hugging Face TGI

- Decoder-only inference requests are harder to batch than for traditional Transformers
- Input and output lengths can greatly vary, leading to very different generation times

Traditional batching waits for all requests to complete

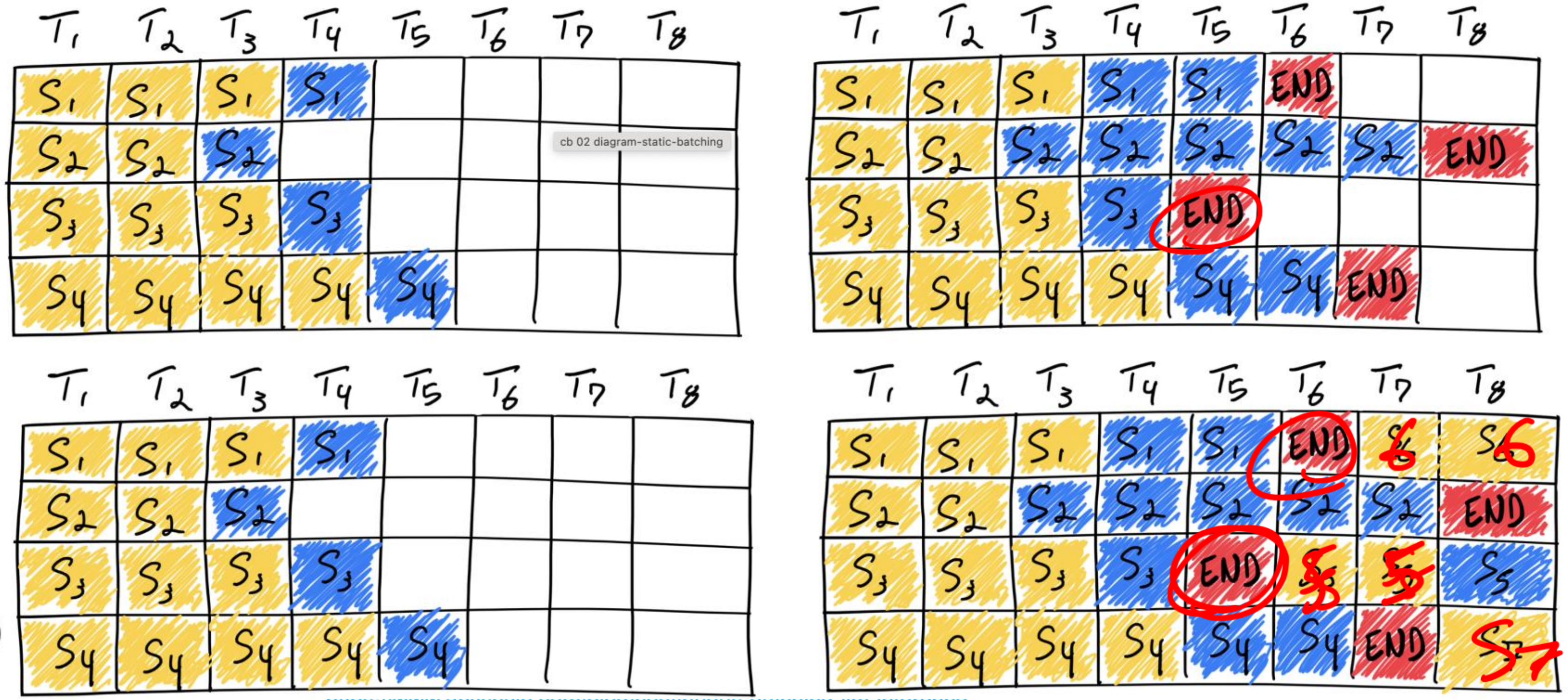
➡️ low hardware usage

S3

Continuous batching evicts completed requests and runs new requests

➡️ high hardware usage

Token generation must pause regularly to run prefill for new requests (`waiting_served_ratio` parameter in TGI)



https://www.anyscale.com/blog/continuous-batching-llm-inference

Content Credit: https://www.slideshare.net/slideshow/julien-simon-deep-dive-optimizing-llm-inference-69d3/270921961

Slides Credit

- For all topics
  - Papers and official blogs
- Paged attention
  - https://www.youtube.com/watch?v=5ZlavKF_98U&t=1646s&ab_channel=Anyscale [Ray Summit 23 Talk]
  - https://youtu.be/yVXtLTcdO1Q?si=XO2Dk-VYOShUMH1u [Waterloo lecture]
- Speculative Decoding
  - https://www.slideshare.net/slideshow/julien-simon-deep-dive-optimizing-llm-inference-69d3/270921961
  - https://youtu.be/S-8yr_RibJ4?si=Kv8xyyTsJvu8oKLV [Efficient NLP]