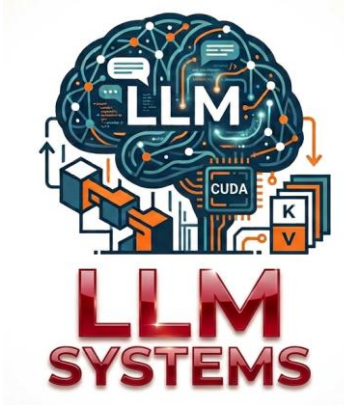




LLM Decoding

Lei Li



Language
Technologies
Institute

Carnegie Mellon University
School of Computer Science

Recap about Tokenization

- Subword tokenization: Byte-Pair-Encoding
 - iteratively merging most frequent pairs of tokens
- Information-theoretic vocabulary (VOLT)
 - solving entropy constrained optimal transport problem
- Pre-tokenization through regex
- Number treatment
- Vocab sharing impact multilingual performance
 - how to solve languages in stagnant quad

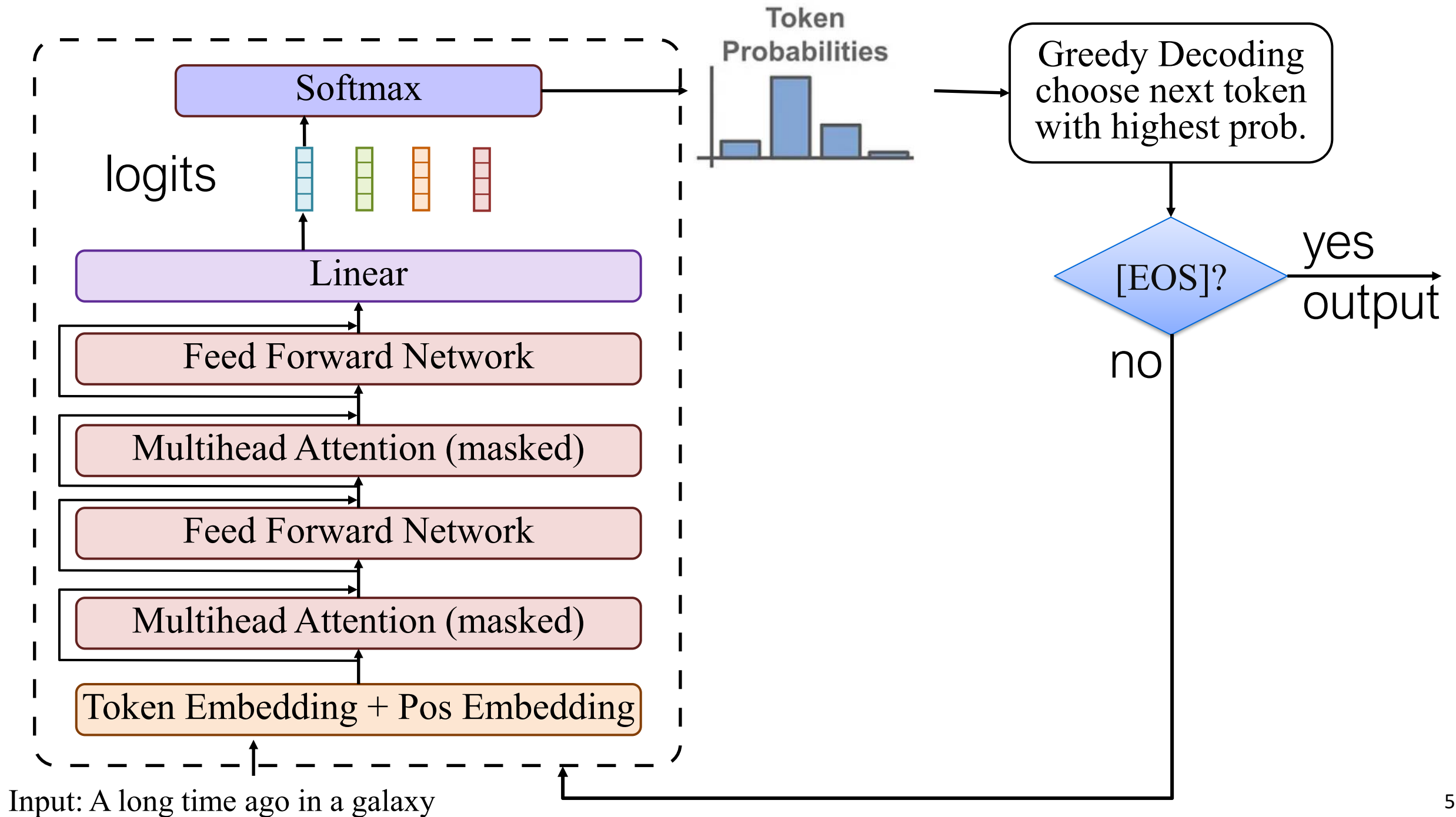
Outline

- Sequence Decoding overview
- Beam search algorithm
- Accelerating Generation: Speculative Decoding
- Further Improvement: EAGLE speculative decoding

Sequence Decoding

$$\operatorname{argmax}_y P(y|x) = f_{\theta}(x, y)$$

- naive solution: exhaustive search over all sequences
 - too expensive $O(V^N)$
- Greedy (max) decoding
- Sampling
- Beam search
 - (approximate) dynamic programming



Max Decoding

- For every next token, pick the one that maximizes the probability

$$\max p(x_t | x_{1 \dots t-1})$$

- equivalent to maximizing logits, no need to normalize

Sampling

- Instead of $\operatorname{argmax}_y P(y|x) = f_\theta(x, y)$
- Generate samples of translation Y from the distribution $P(Y|X)$
 - Each step, just sample from $P(y_t|x, y_{0:t-1}) = \operatorname{Softmax}(h)$
 - Temperature sampling: $P(y_t|x, y_{0:t-1}) = \operatorname{Softmax}(h/T)$
- Q: how to generate samples from a discrete distribution?

Discrete Sampling

- sample n values x 's from k categories, with prob. $p_1, p_2, \dots, p_k$
- Direct sampling: $O(nk)$
- Binary Search: $O(k + n \log k)$
- Alias sampling: $O(k \log k + n)$

```
probs = torch.softmax(logits, dim=-1)
next_token = torch.multinomial(probs, num_samples=1)
```

Fast Sampling with Gumbel Max Trick

- sampling from $\text{Categorical}(\text{Softmax}(h))$ is equivalent to

$$\begin{aligned} z &\sim \text{Uniform}(0,1) \\ x &= h - \log(-\log z) \\ \arg \max x \end{aligned}$$

- Theory: x follows Gumbel distribution, and $\arg \max x$ follows $\text{Categorical}(\frac{\exp h_i}{\sum_{j=1}^k h_j})$

<https://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-and-weightedreservoir-sampling/>

```
class GumbelSampler:
    def __init__(self, batch_size, vocab_size, device):
        self.batch_size = batch_size
        self.vocab_size = vocab_size # Pre-compute noise
        self.noise = self._prepare_gumbel_noise(device)

    def _prepare_gumbel_noise(self, device):
        # Generate noise tensor once
        uniform_noise = torch.rand(self.batch_size, self.vocab_size, device=device)
        return -torch.log(-torch.log(uniform_noise))

    def sample(self, logits):
        # Direct sampling without softmax
        return torch.argmax(logits + self.noise, dim=-1)
```

Improving Quality in Sampling

- Top-k sampling
 - Sample from top-k tokens, discard the remaining
- Top-p sampling
 - selects top tokens whose cumulative probability exceeds a hyperparameter p
- Controlling temperature T

Outline

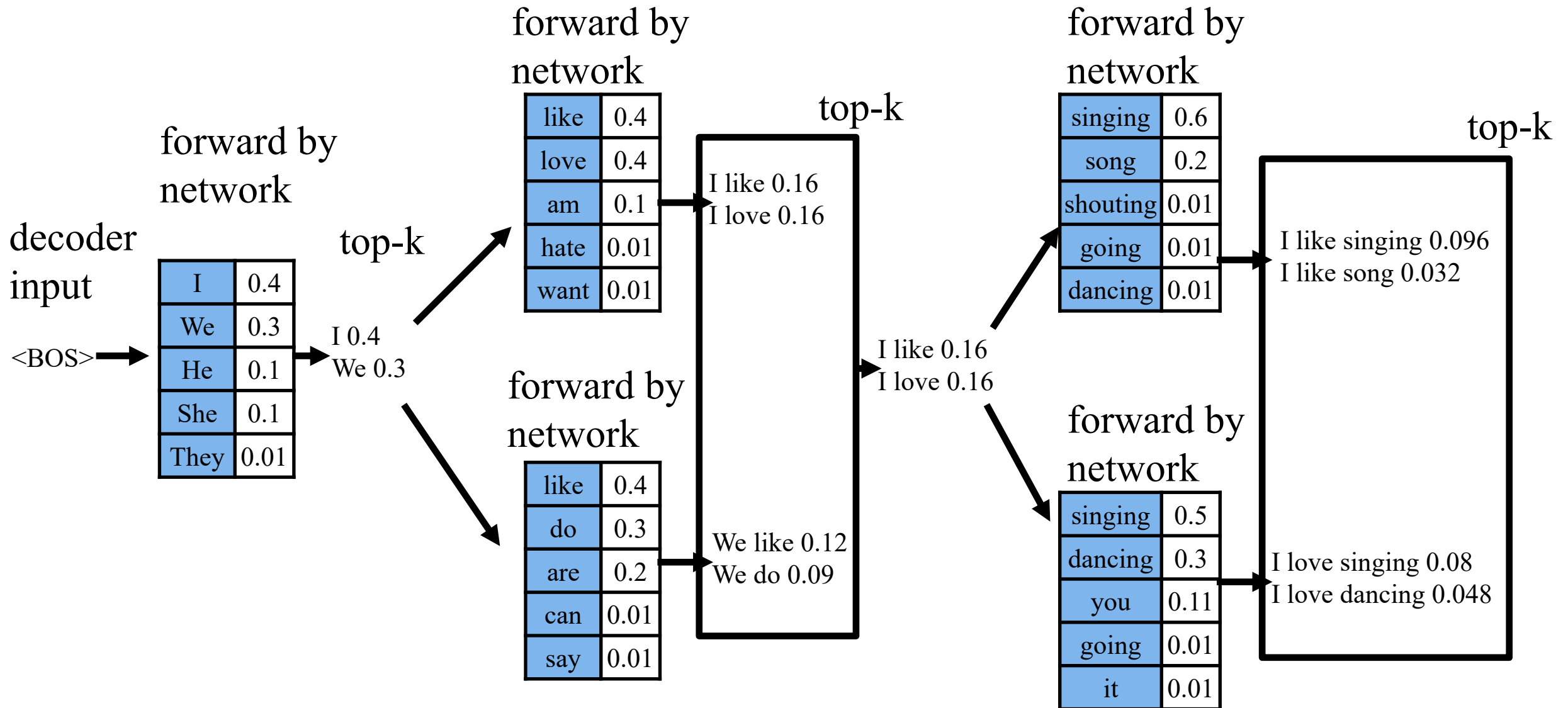
- Sequence Decoding overview
- • Beam search algorithm
- Accelerating Generation: Speculative Decoding
- Further Improvement: EAGLE speculative decoding

Beam Search

Find approximate solutions to $\operatorname{argmax}_y P(y|x) = f_{\theta}(x, y)$

1. start with empty candidate set S
2. at each step, keep k best partial sequences
3. expand them with one more forward generation
4. collect new partial results and keep top- $k \rightarrow S$

Beam Search



```
best_scores = []
add {[0], 0.0} to best_scores # 0 is for beginning of sentence token
for i in 1 to max_length:
    new_seqs = PriorityQueue()
    for (candidate, s) in best_scores:
        if candidate[-1] is EOS:
            prob = all -inf
            prob[EOS] = 0
        else:
            prob = using model to take candidate and compute next token probabilities (logp)
        pick top k scores from prob, and their index
        for each score, index in the top-k of prob:
            new_candidate = candidate.append(index)
            new_score = s + score
            if not new_seqs.full():
                add (new_candidate, new_score) to new_seqs
            else:
                if new_seqs.queue[0][1] < new_score:
                    new_seqs.get() # pop the one with lowest score
                    add (new_candidate, new_score) to new_seqs
```

Pruning for Beam Search

- Relative threshold pruning
 - prune candidates with too low score from the top one
 - Given a pruning threshold rp and an active candidate list C , a candidate $cand \in C$ is discarded if: $score(cand) \leq rp * \max\{score(c)\}$
- Absolute threshold pruning:
 - $score(cand) \leq \max\{score(c)\} - ap$
- Relative local threshold pruning

Combine Sample and Beam Search

- Sample the first tokens
- continue beam search for the later
- why?
 - to improve sequence diversity

Beam Search Code example

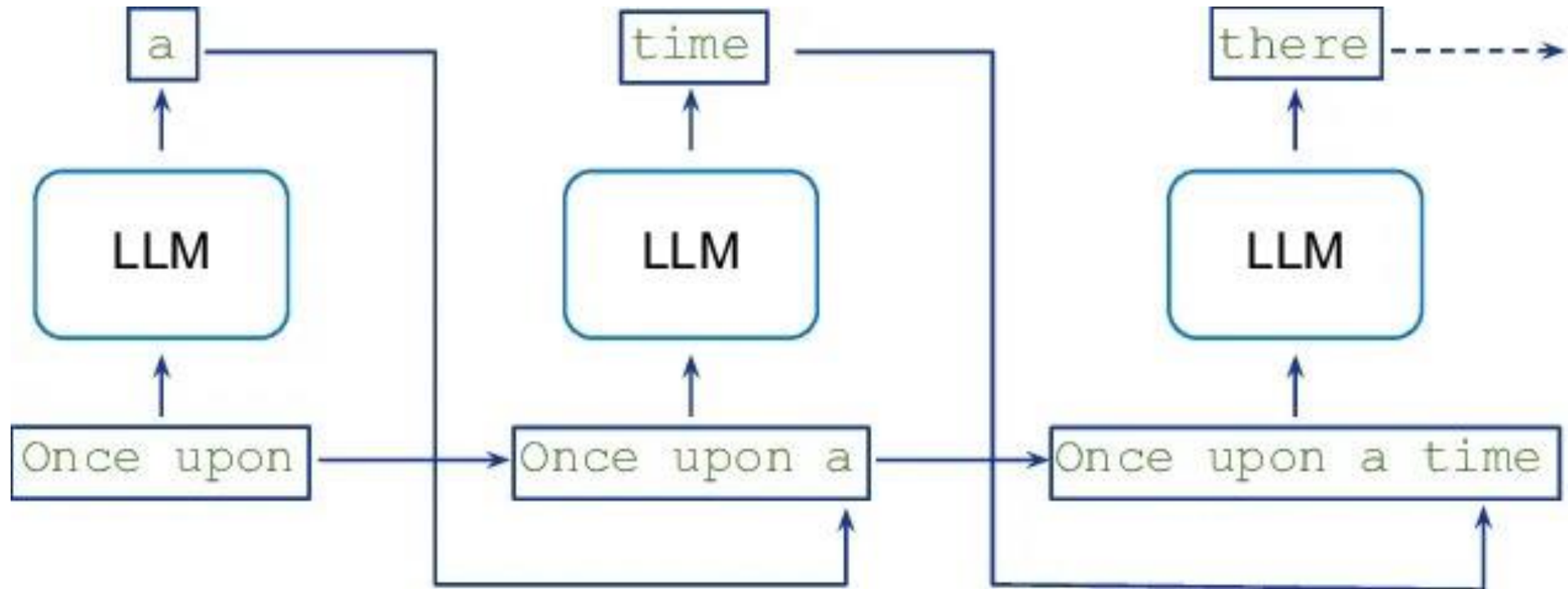
- https://github.com/lmsystem/lmsys_code_examples/blob/main/decoding/decoding.ipynb

Outline

- Sequence Decoding overview
- Beam search algorithm
-  • Accelerating Generation: Speculative Decoding
- Further Improvement: EAGLE speculative decoding

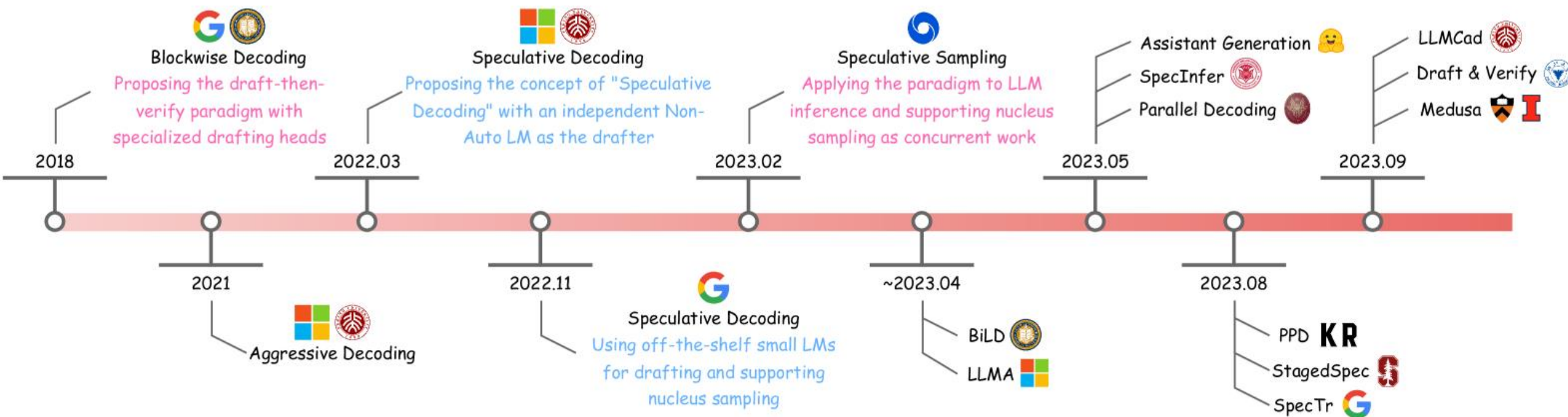
LLM Autoregressive Decoding is slow

- need to generate one token at the time in a **sequential** manner and each token can take 100s of milliseconds



Accelerating with Speculative Decoding

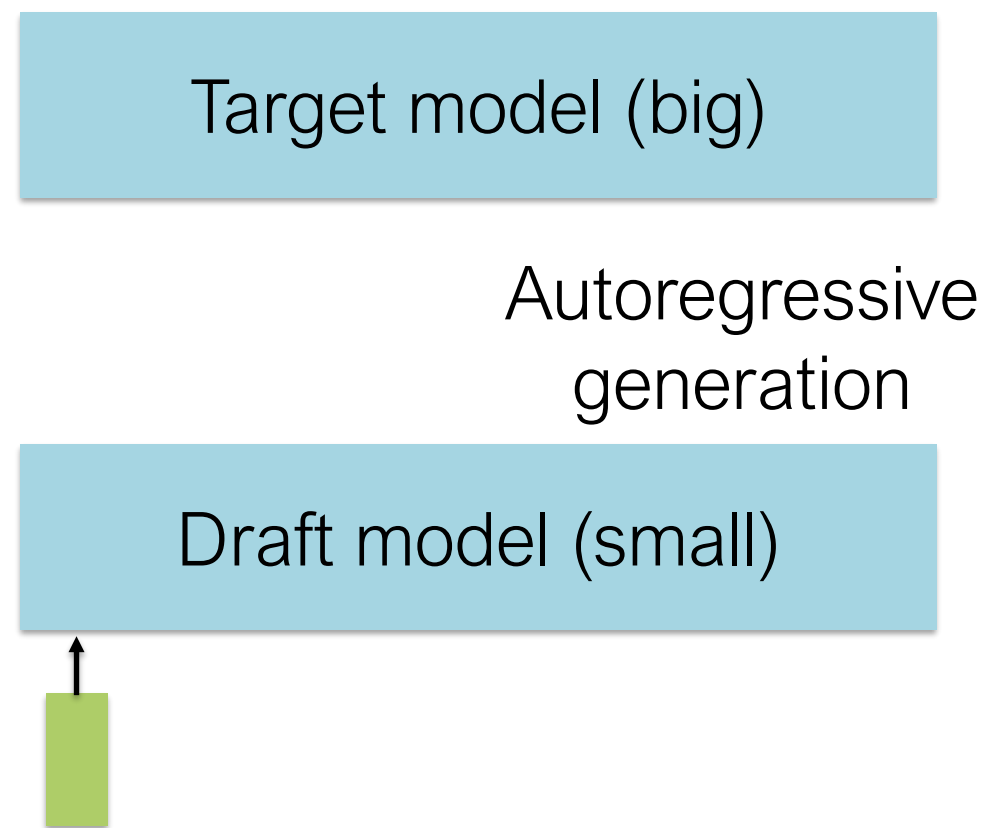
- Commonly used to reduce latency in LLMs inference applications



Speculative Decoding

- Key idea: use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them

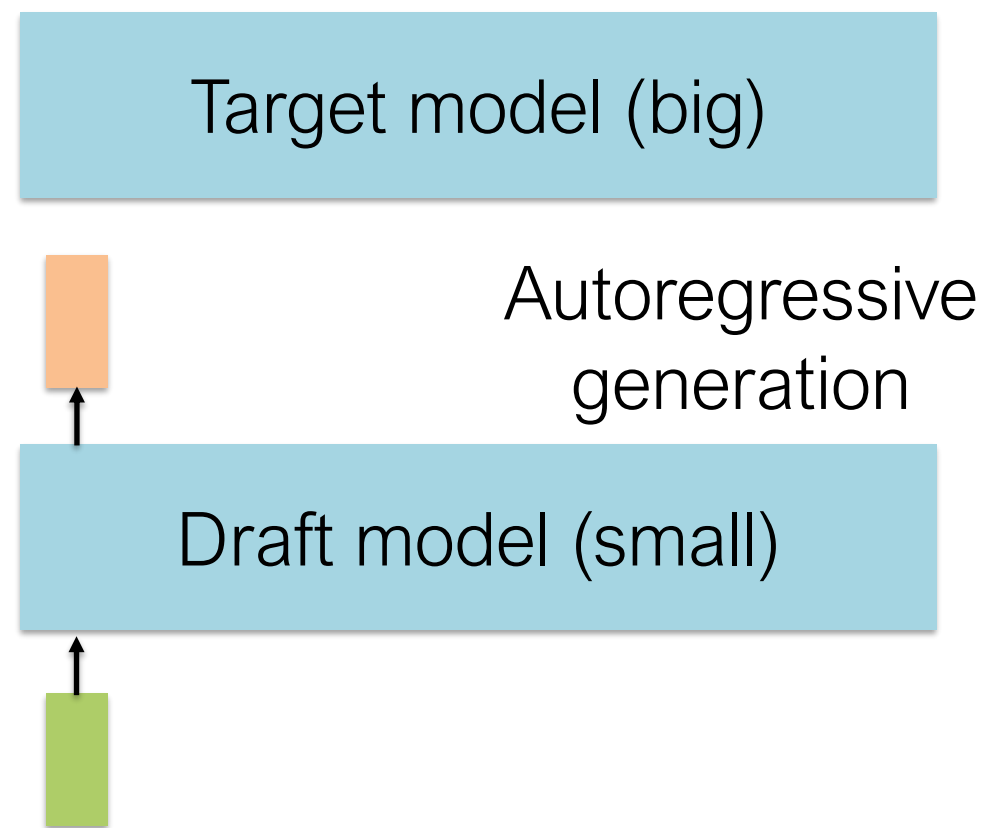
- $y_{1:k} = (y_1, y_2, \dots, y_k) \sim f_{\text{draft}}(\cdot | x)$



Speculative Decoding

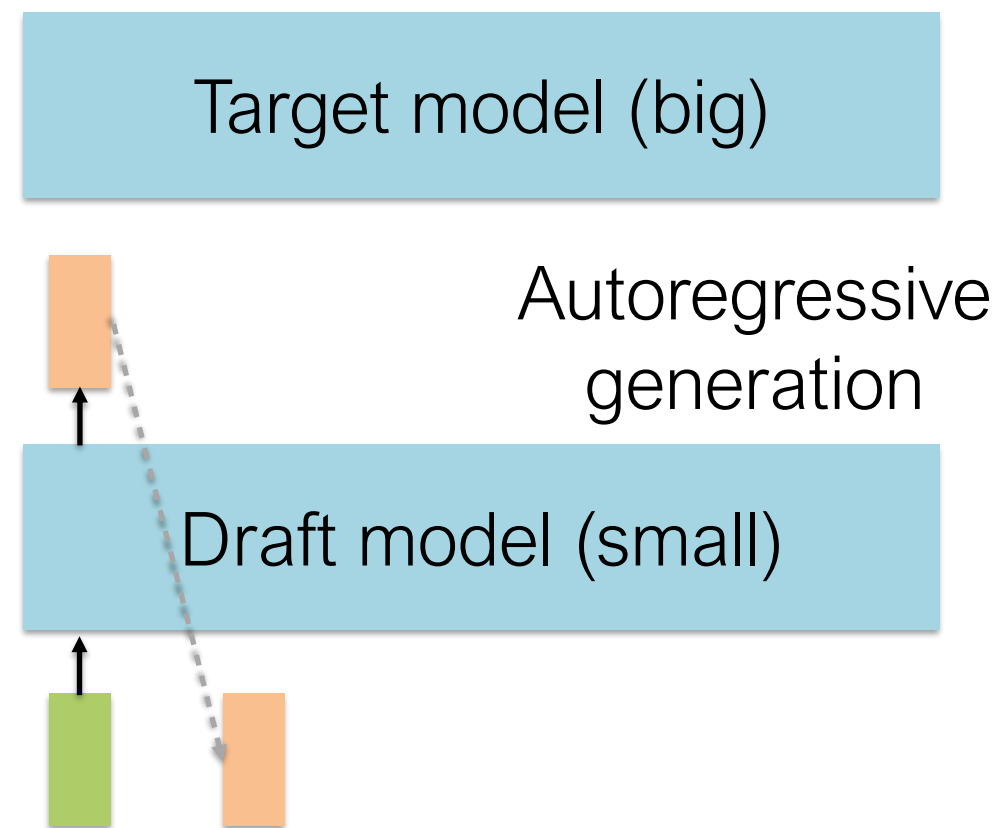
- Key idea: use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them

- $y_{1:k} = (y_1, y_2, \dots, y_k)$
 $\sim f_{\text{draft}}(\cdot | x)$



Speculative Decoding

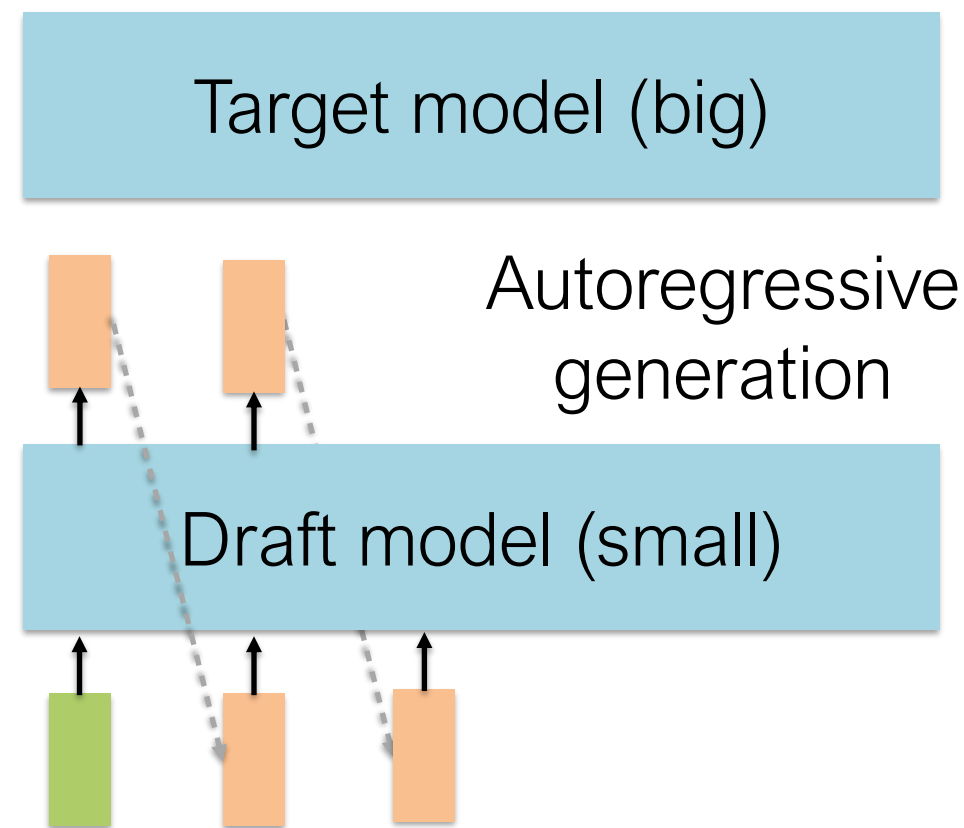
- Key idea: use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them
- $y_{1:k} = (y_1, y_2, \dots, y_k)$
 $\sim f_{\text{draft}}(\cdot | x)$



Speculative Decoding

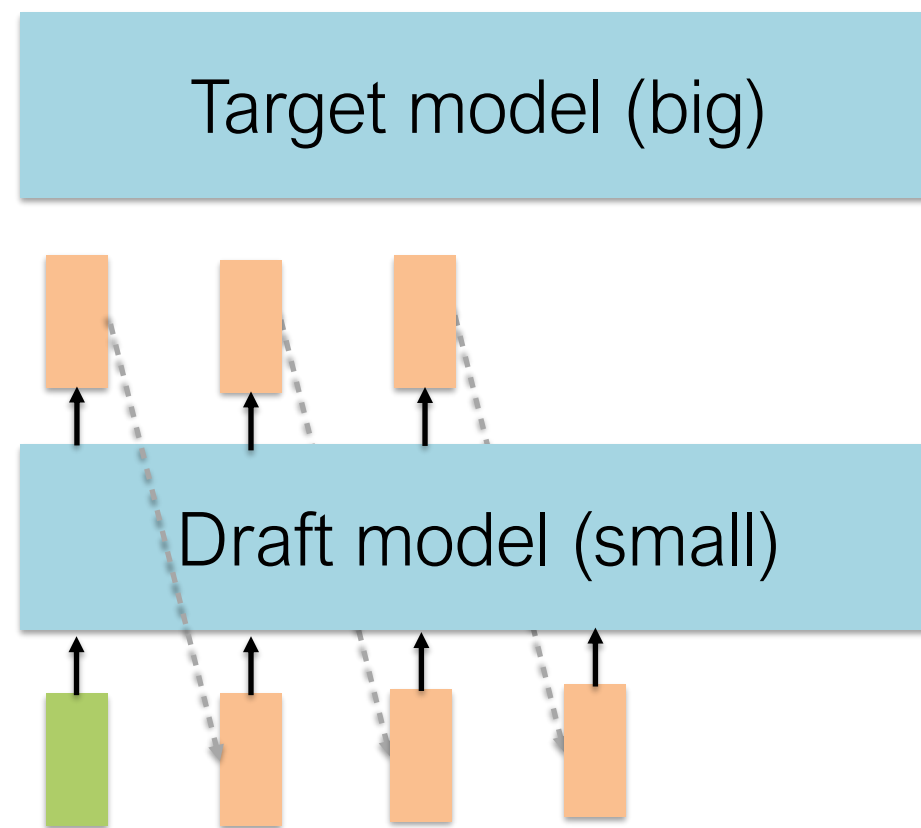
- Key idea: use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them

- $y_{1:k} = (y_1, y_2, \dots, y_k)$
 $\sim f_{\text{draft}}(\cdot | x)$



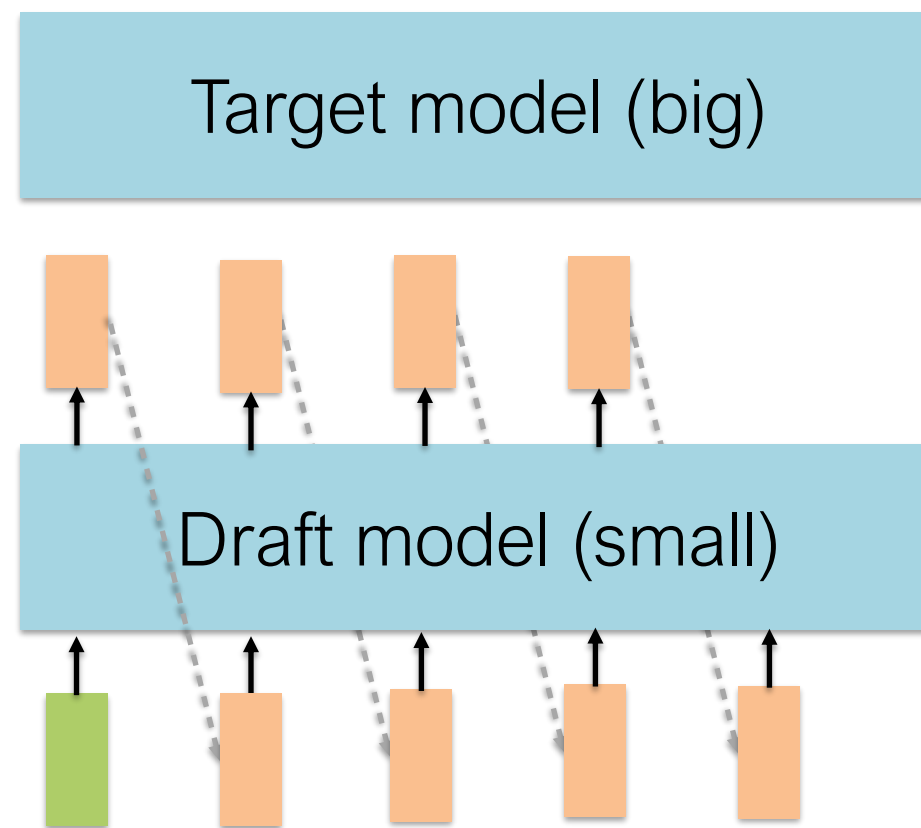
Speculative Decoding

- Key idea: use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them
- $y_{1:k} = (y_1, y_2, \dots, y_k)$
 $\sim f_{\text{draft}}(\cdot | x)$



Speculative Decoding

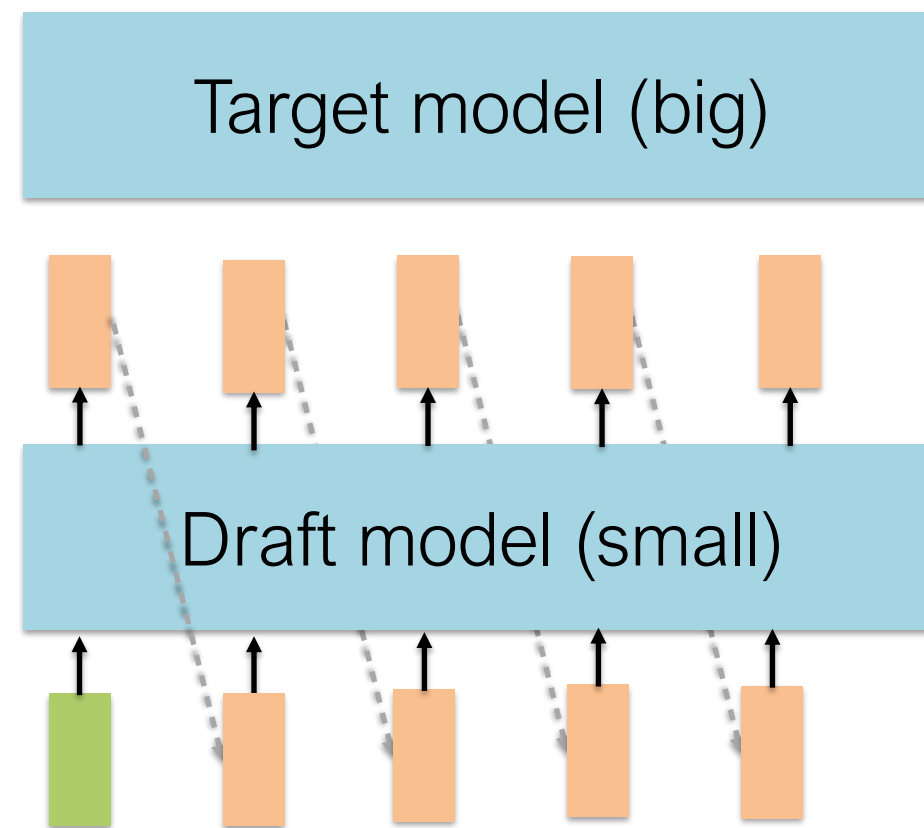
- You use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them
- $y_{1:k} = (y_1, y_2, \dots, y_k)$
 $\sim f_{draft}(\cdot | x)$



Speculative Decoding

- Key idea: use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them

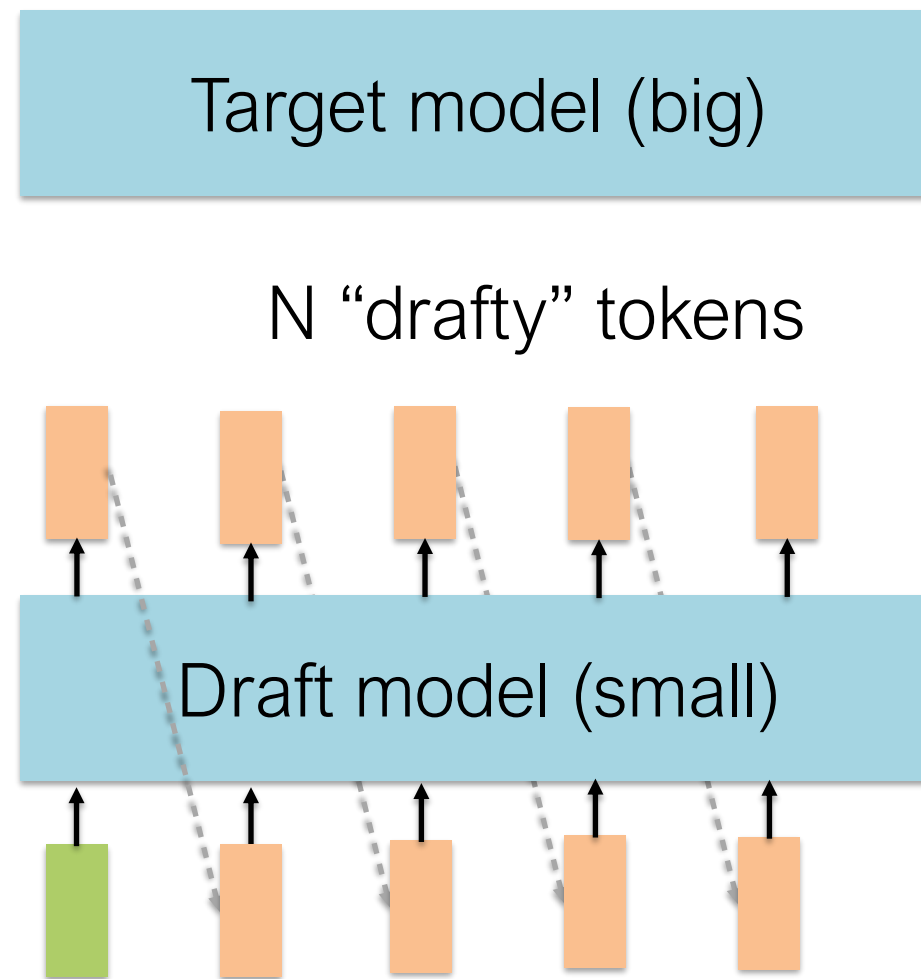
- $y_{1:k} = (y_1, y_2, \dots, y_k)$
 $\sim f_{\text{draft}}(\cdot | x)$



Speculative Decoding

- Key idea: use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them

- $y_{1:k} = (y_1, y_2, \dots, y_k)$
 $\sim f_{\text{draft}}(\cdot | x)$

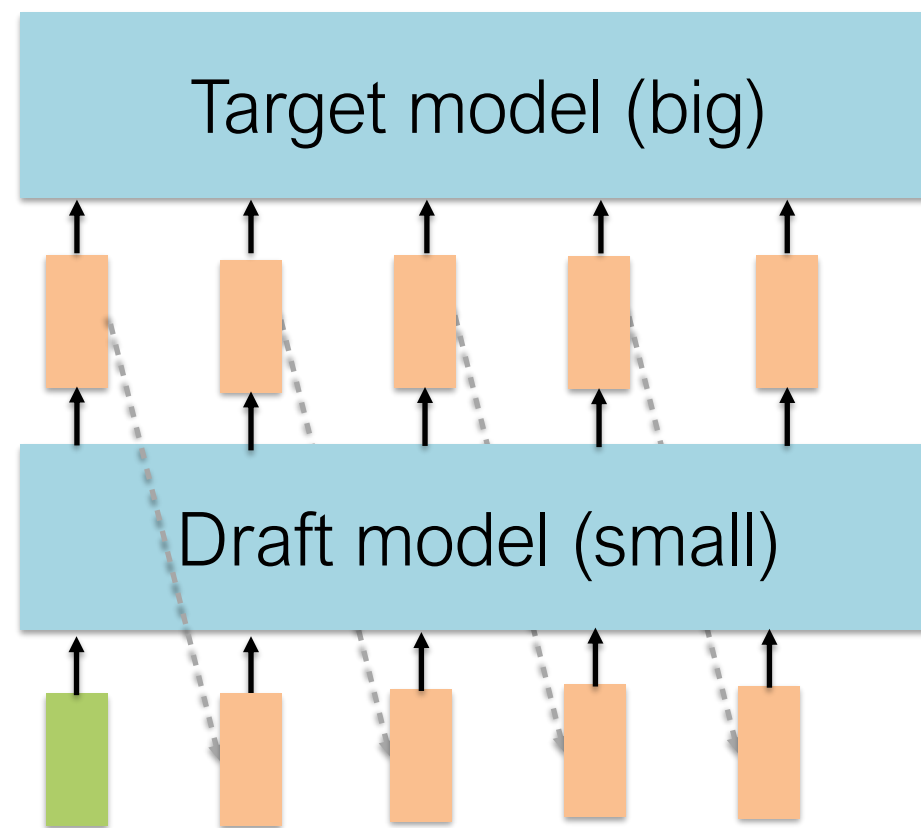


Speculative Decoding

- Key idea: use a small model (draft model) to generate N “drafty” tokens and then leverage the large model (target model) to validate them

- $y_{1:k} = (y_1, y_2, \dots, y_k)$
 $\sim f_{\text{draft}}(\cdot | x)$

Parallel validation of tokens (fast)

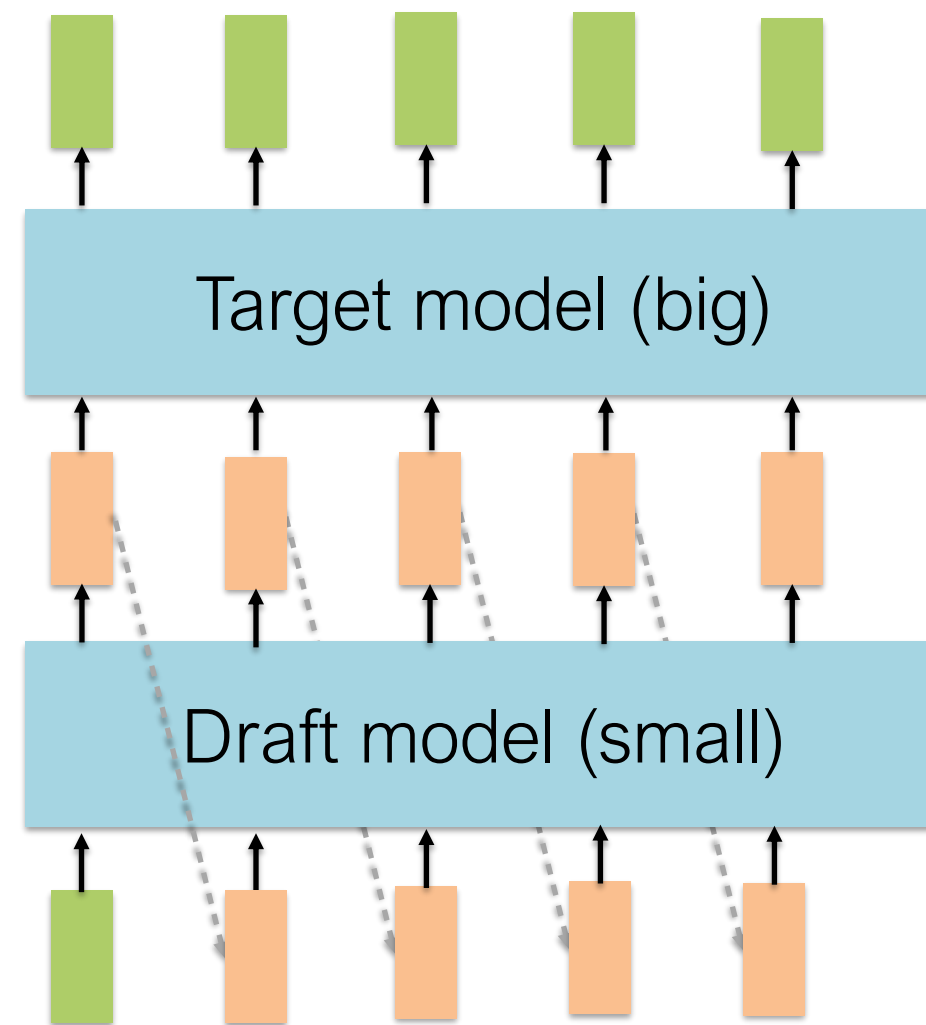


Speculative Decoding

- Each draft token is considered valid if it is among the top-k predictions of the target LLM
- It computes the probability of each draft token under its own distribution $f_{\text{target}}(\cdot | x, y_{1:i-1})$
- Accept token if $y_i \in \text{TopK}(f_{\text{target}}(\cdot | x, y_{1:i-1}))$

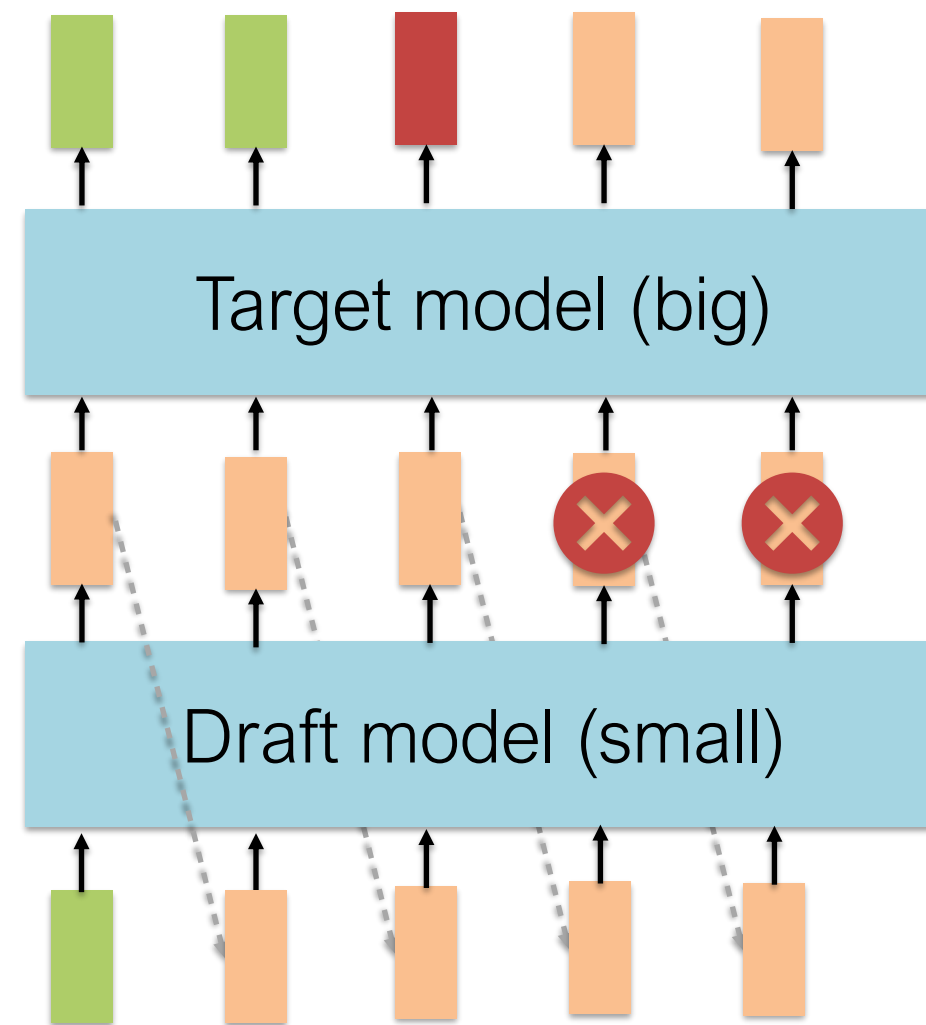
Speculative Decoding

- Target model accepts each token only if it lies in the top-k predictions
- Accept if $y_i \in \text{TopK}(f_{\text{target}}(\cdot | x, y_{1:i-1}))$



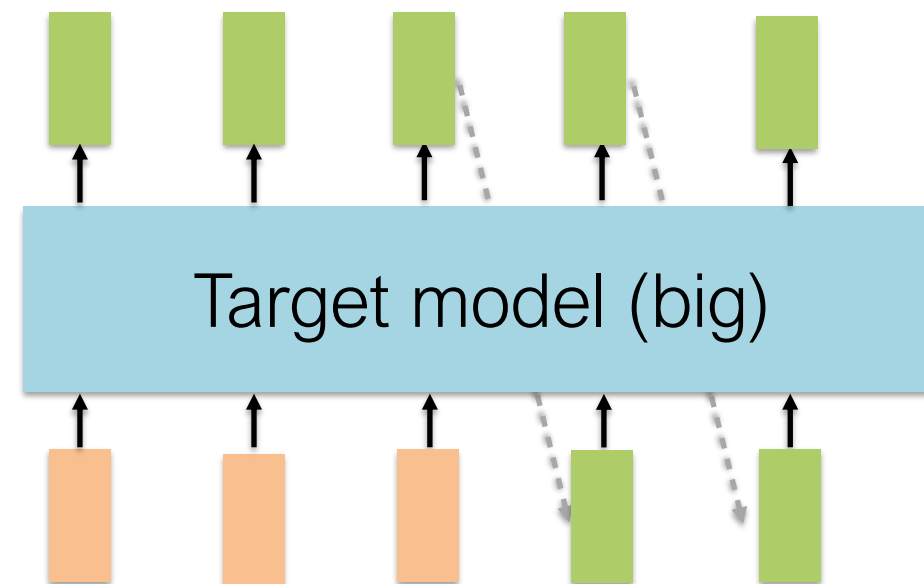
Speculative Decoding

- If a draft token is rejected, the target model will start generating on its own starting from the last accepted token



Speculative Decoding

- If a draft token is rejected, the target model will start generating on its own starting from the last good token

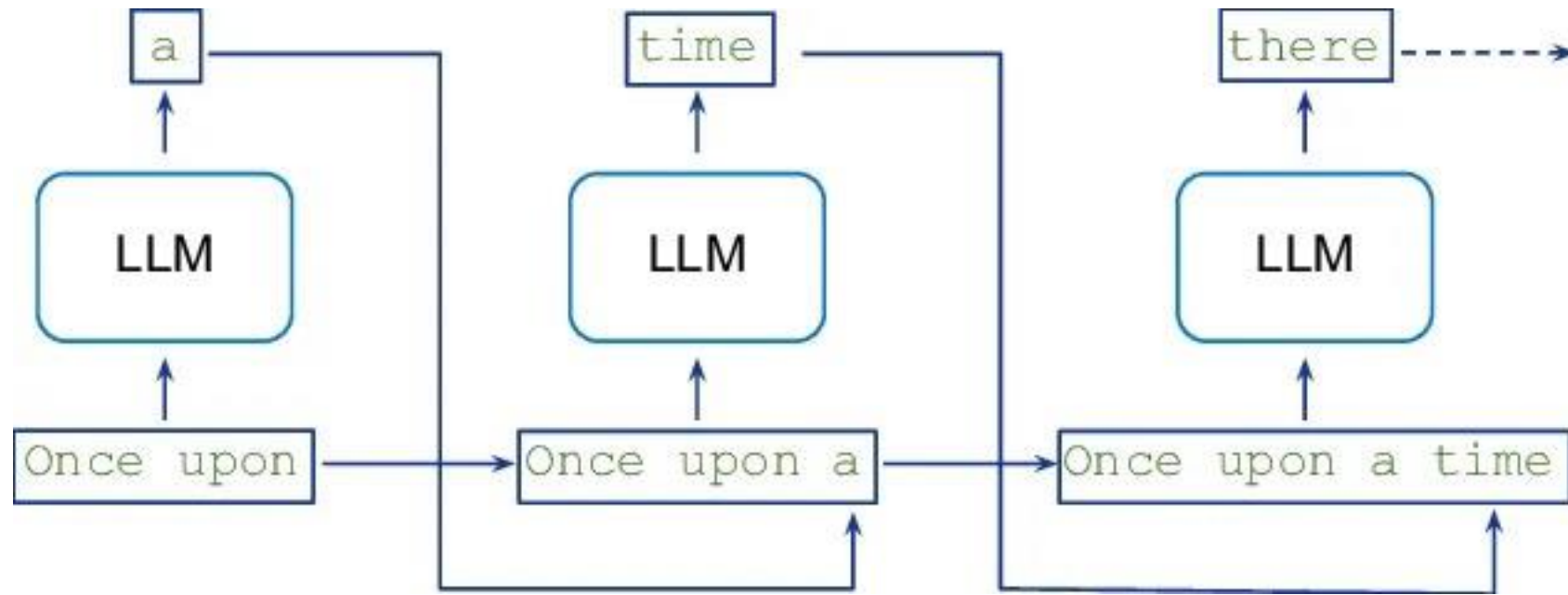


Why is Speculative Decoding faster?

- Speculative decoding achieves a speed up compared to sequential decoding because **it is faster to validate tokens than to generate them from scratch**

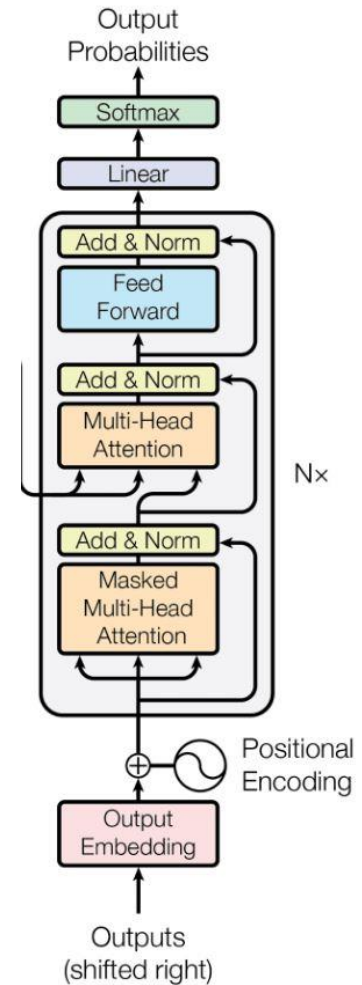
Why is it faster?

- To generate N tokens, you need to run N forward passes in an autoregressive way



Why is it faster?

- In the speculative decoding setting, you are looking at the likelihood of a token given the previous ones
- With causal attention you are able to compute that likelihood for N tokens in a single forward pass



Speculative Decoding Performance

- By using TopK validation of each draft token we make sure the generated sequence **cannot deviate too much from the target model's result**
- Any mismatch in token distribution between target and draft models is corrected by having the target model generating the rest of the N tokens

Speculative Decoding generates good text

Models	EN→DE		DE→EN		EN→RO		RO→EN	
	Speed	BLEU	Speed	BLEU	Speed	BLEU	Speed	BLEU
Transformer-base ($b = 5$)	1.0×	28.89	1.0×	32.53	1.0×	34.96	1.0×	34.86
Transformer-base ($b = 1$)	1.1×	28.73	1.1×	32.18	1.1×	34.83	1.1×	34.65
Blockwise Decoding ($k = 10$)	1.9×	28.73	2.0×	32.18	1.4×	34.83	1.4×	34.65
Blockwise Decoding ($k = 25$)	1.6×	28.73	1.7×	32.18	1.2×	34.83	1.2×	34.65
SpecDec ($k = 10$)	4.2×	28.90	4.6×	32.61	3.9×	35.29	4.1×	34.88
SpecDec ($k = 25$)	5.1×	28.93	5.5×	32.55	4.6×	35.45	4.8×	35.03
12+2 Transformer-base ($b = 5$)	1.0×	29.13	1.0×	32.45	1.0×	34.93	1.0×	34.80
12+2 Transformer-base ($b = 1$)	1.1×	28.99	1.1×	32.08	1.1×	34.79	1.1×	34.55
Blockwise Decoding ($k = 10$)	1.6×	28.99	1.7×	32.08	1.2×	34.79	1.2×	34.55
Blockwise Decoding ($k = 25$)	1.4×	28.99	1.5×	32.08	1.1×	34.79	1.1×	34.55
SpecDec ($k = 10$)	2.7×	29.08	3.0×	32.40	2.3×	35.12	2.4×	34.85
SpecDec ($k = 25$)	3.0×	29.13	3.3×	32.48	2.5×	35.07	2.6×	34.91

Speculative Decoding is faster

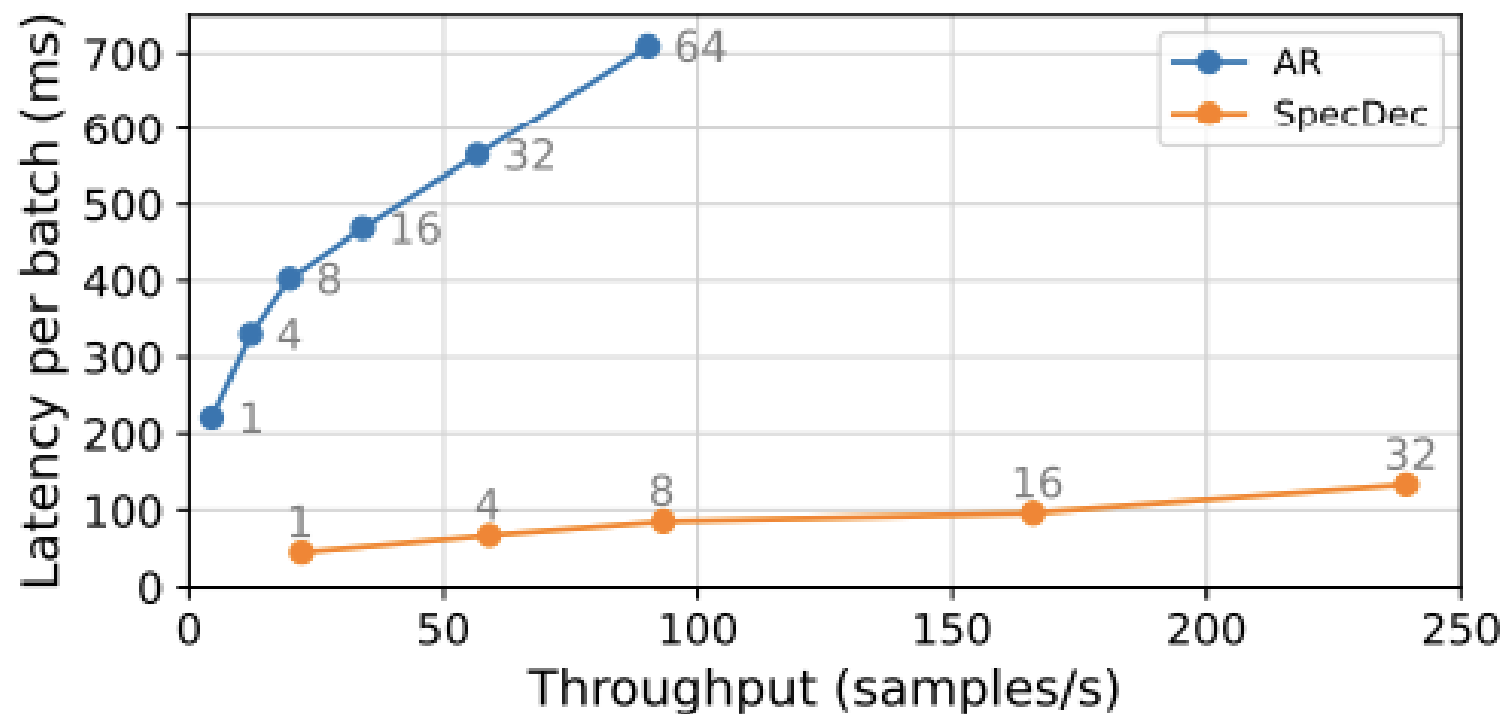


Figure 5: The latency-throughput curve with various batch sizes on WMT14 EN→DE.

How to choose N?

- This comes as a tradeoff
- Pros of a big N: can theoretically achieve more speed up
- Cons of a big N: higher chances of getting a rejected token, higher cost of a rejection, need to compute more softmax over the vocab (potential memory bottlenecks), longer stall times for real-time applications like chat bots
- Popular choices are $N = 4, 8$

Alignment considerations

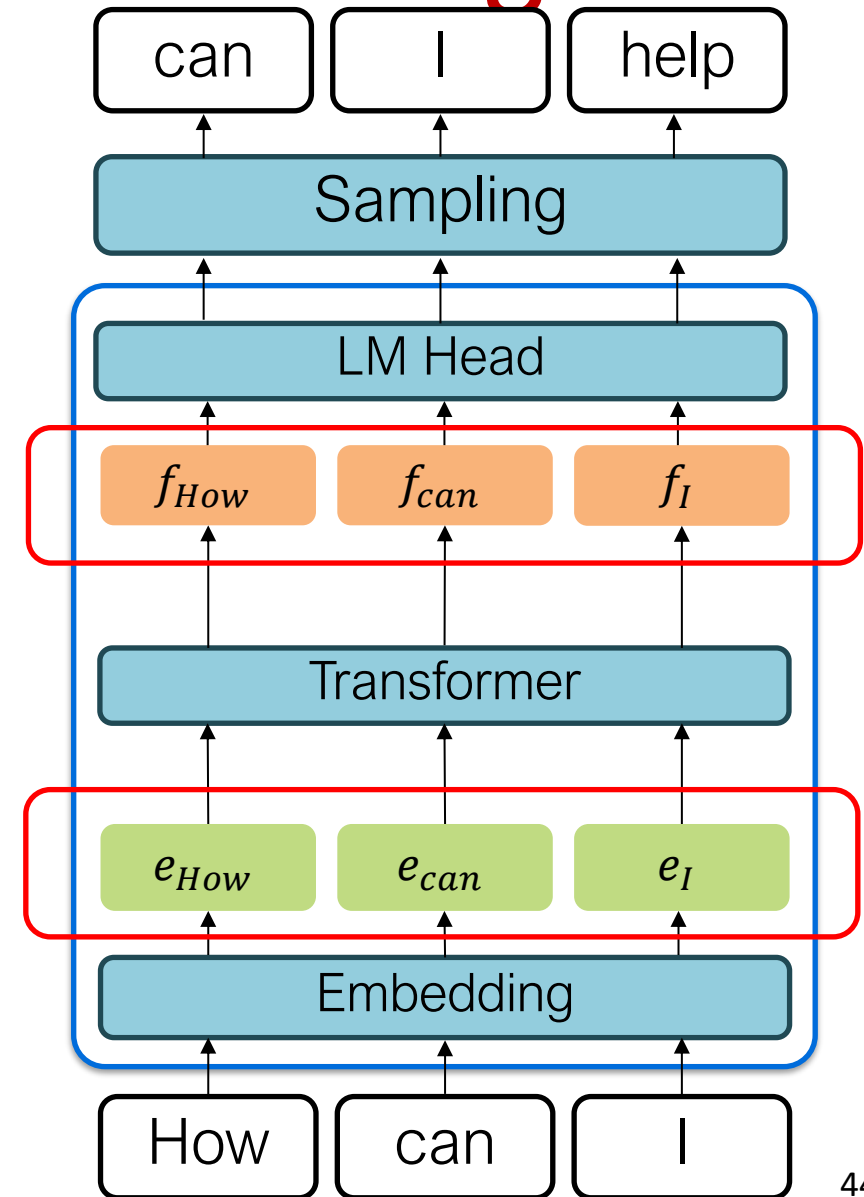
- The draft model is said to be well aligned with the target model when **the rejection rate of the drafty tokens is low**
- Good alignment is key for performance as the speedup is canceled when a rejection happens
- Choosing models from the same family as draft-target pairs is common and usually shows good results

Outline

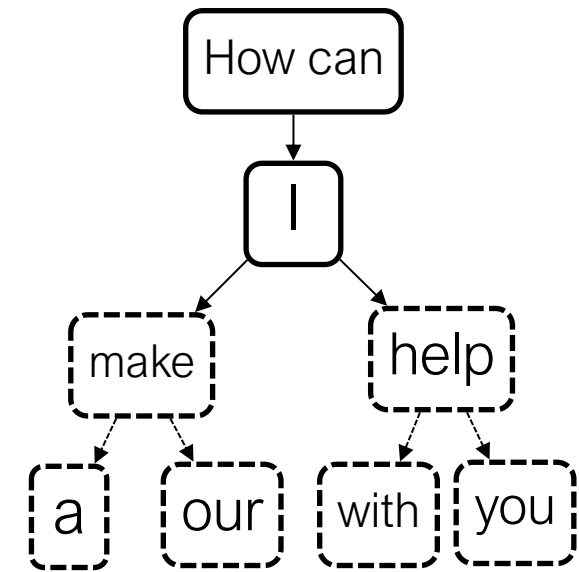
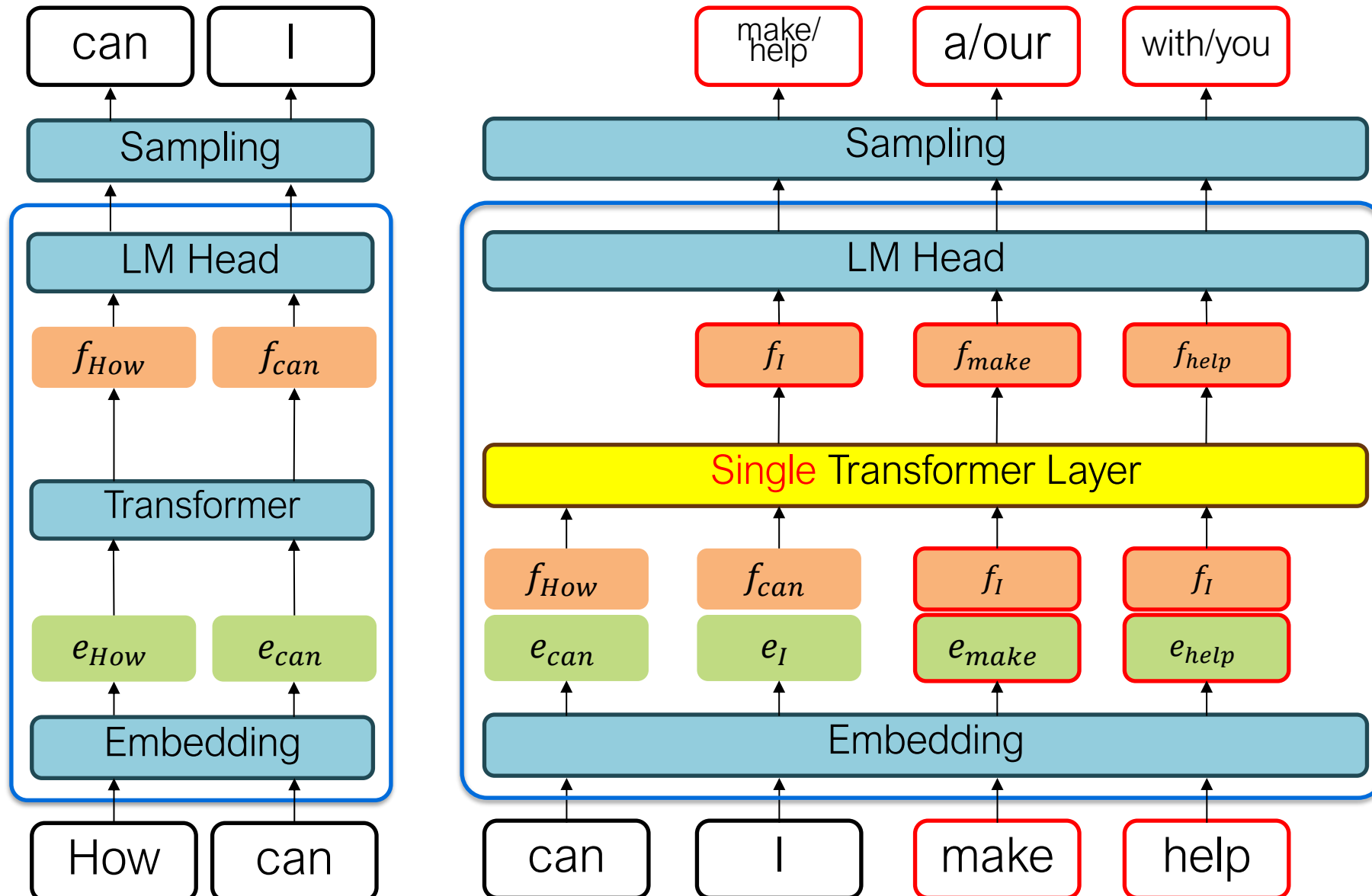
- Sequence Decoding overview
- Beam search algorithm
- Accelerating Generation: Speculative Decoding
- ➔ • Further Improvement: EAGLE speculative decoding

Improving Speculative Decoding

- Vanilla speculative decoding uses a small LM as drafting model which predicts the next token
- Observation: the next LLM final layer feature is simpler to predict than next token
- EAGLE directly predicts the next LLM final layer feature with a small model

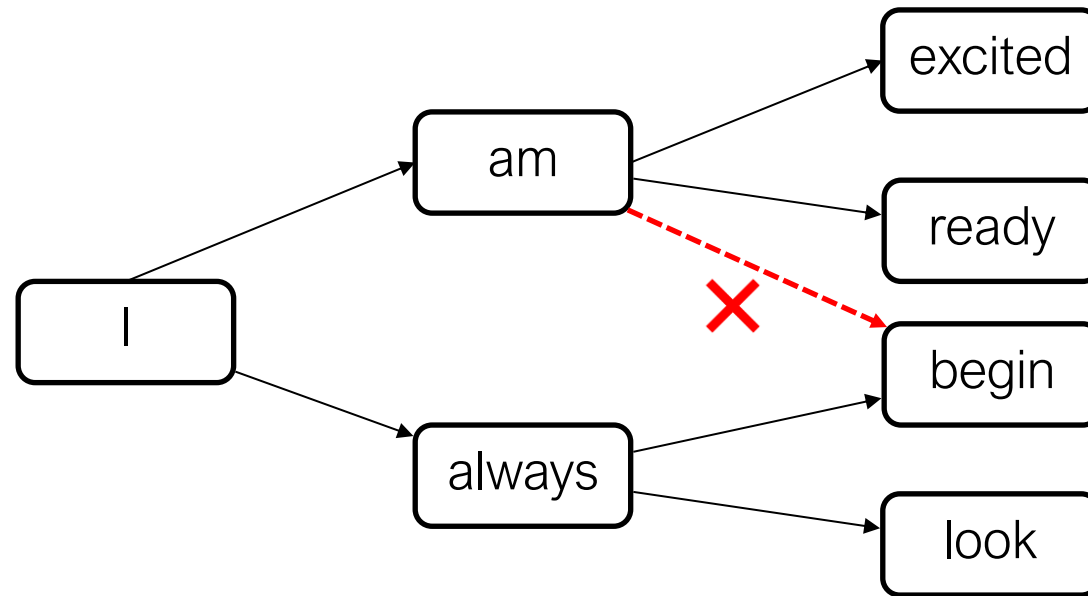


EAGLE Overview



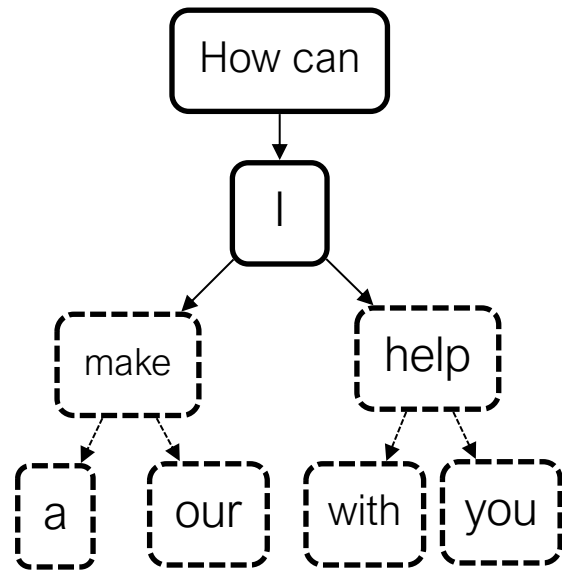
EAGLE: Word Embedding + Final Layer Feature

- The sampled token affects the final layer feature a lot
- “begin” cannot appear after “I am”



Efficient Implementation with Tree Attention

- Drafted tokens are flattened on input with tree shaped attention mask



How
can
I
make
help
a
our
with
you

How	can	I	make	help	a	our	with	you
1								
1	1							
1	1	1						
1	1	1	1					
1	1	1		1				
1	1	1	1		1			
1	1	1	1			1		
1	1	1		1			1	
1	1	1		1				1

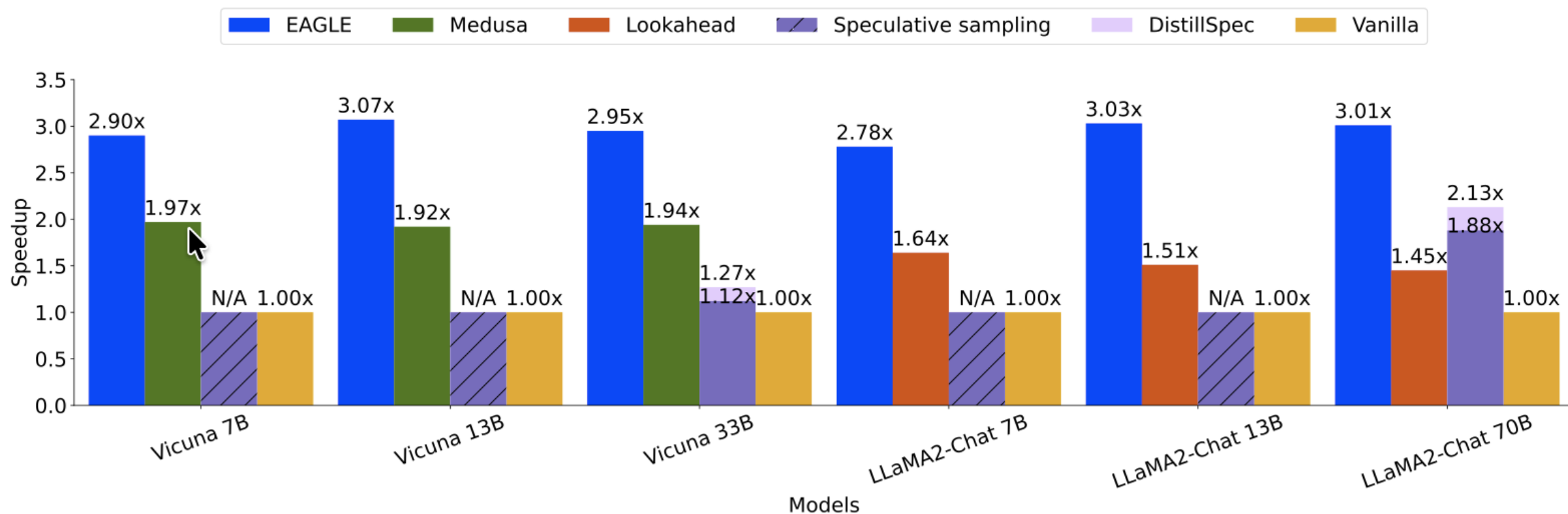
EAGLE Training

- Autoregressive training with both smooth L1 loss on final layer feature and cross entropy loss on token distribution
- $L = L_{reg} + w_{cls}L_{cls}$
- $L_{reg} = \text{Smooth L1}(f_{i+1}, \text{Draft_Model}(T_{2:i+1}, F_{1:i}))$
- $T_{2:i+1}$ are token embeddings and $F_{1:i} = (f_1, \dots, f_i)$

EAGLE Training

- Autoregressive training with both smooth L1 loss on final layer feature and cross entropy loss on token distribution
- $L = L_{reg} + w_{cls}L_{cls}$
- $p_{i+2} = \text{Softmax}(\text{LM_head}(f_{i+1}))$
- $\hat{p}_{i+2} = \text{Softmax}\left(\text{LM_head}\left(\hat{f}_{i+1}\right)\right)$
- $L_{cls} = \text{Cross_Entropy}\left(p_{i+2}, \hat{p}_{i+2}\right)$

Much Faster Decoding on MT-Bench



Further Improvement on EAGLE

- [EAGLE-2](#) prunes the tokens in tree with low confidence
- [EAGLE-3](#) scales the method to larger training data
- Feel free to check the papers!

Summary

- Beam Search decoding: approximate dynamic programming to keep a beam of partial sequences.
- Speculative decoding makes LLMs decoding phase faster
 - key idea: It uses a draft model to generate tokens and uses the target model to validate them
 - The speed up comes because validating tokens is faster than generating them
- EAGLE: use the original LM's embedding and LM head, and a small Transformer layer to predict the final layer features for prediction
 - implementing with tree attention mask

Demo

- https://github.com/lmsys/lmsys_code_examples/blob/main/speculative_decoding/Speculative_decoding_demo.ipynb
- https://github.com/lmsys/lmsys_code_examples/blob/main/speculative_decoding/EAGLE/demo.py

Quiz