LLM Sys

Decoding

Sampling, Beam Search and Speculative 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

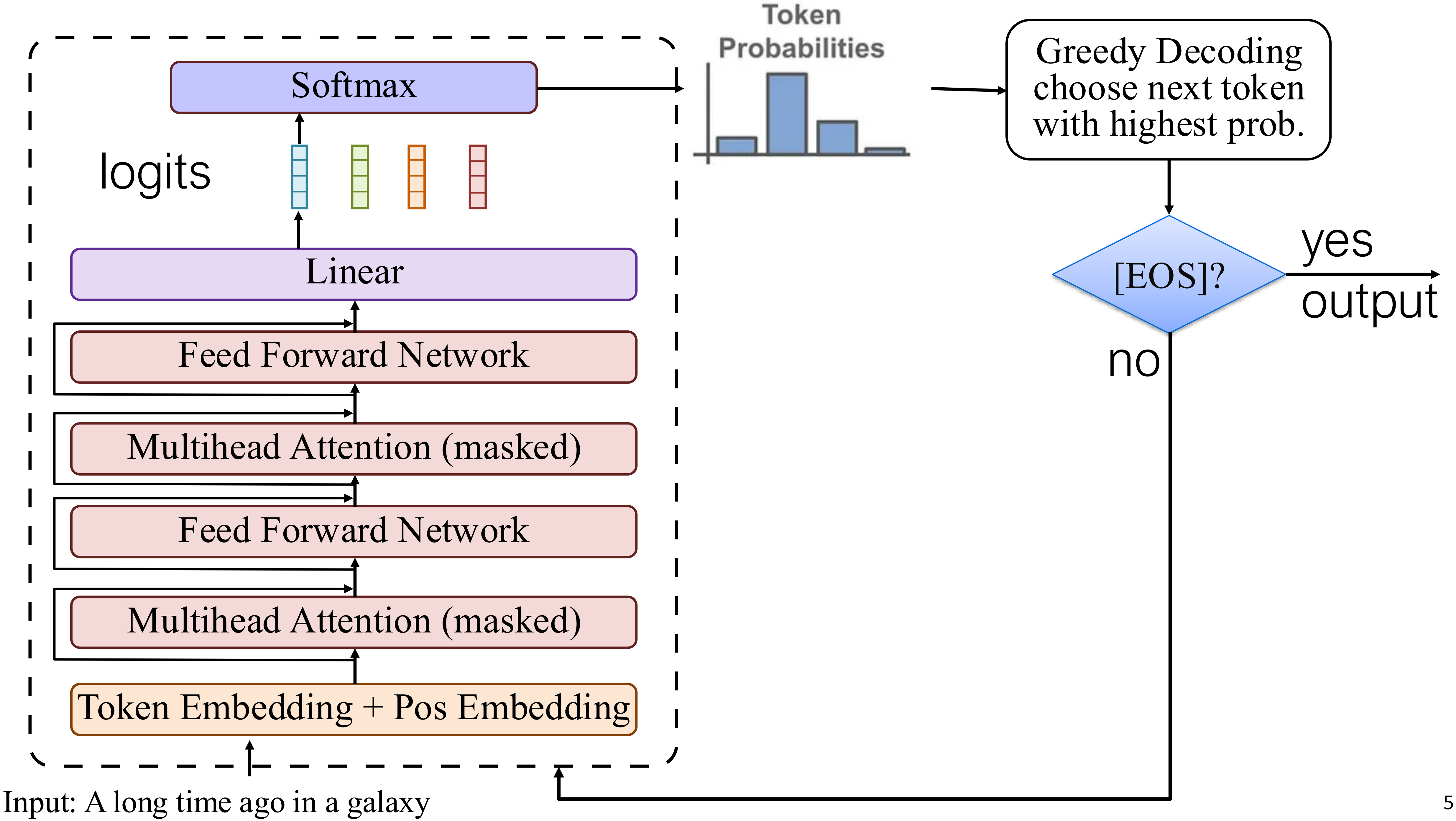
Outline

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

Sequence Decoding

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

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


Token Probabilities
Greedy Decoding choose next token with highest prob.
Softmax
logits
Linear
Feed Forward Network
Multihead Attention (masked)
Feed Forward Network
Multihead Attention (masked)
Token Embedding + Pos Embedding
[EOS]?
yes
output
no
Input: A long time ago in a galaxy

Max Decoding

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

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

- equivalent to maximizing logits, no need to normalize

Sampling

- Instead of $\mathrm{argmax}_{\mathrm{y}} P(y|x) = f_\theta(x, y)$

- Generate samples of translation Y from the distribution P(Y|X)

- Q: how to generate samples from a discrete distribution?

Discrete Sampling

- sample n values x's from k categories, with prob. p1, p2, … pk
- 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 $\mathrm{Categorical}(\mathrm{Softmax}(h))$ is equivalent to

$$\arg\max x$$
$$z \sim \mathrm{Uniform}(0,1)$$
$$x = h - \log(-\log z)$$

- Theory: x follows Gumbel distribution, and argmax x follows 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/

Stochastic Beams and Where to Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement. Kool et al. ICML 2019

```python
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)
```

Outline

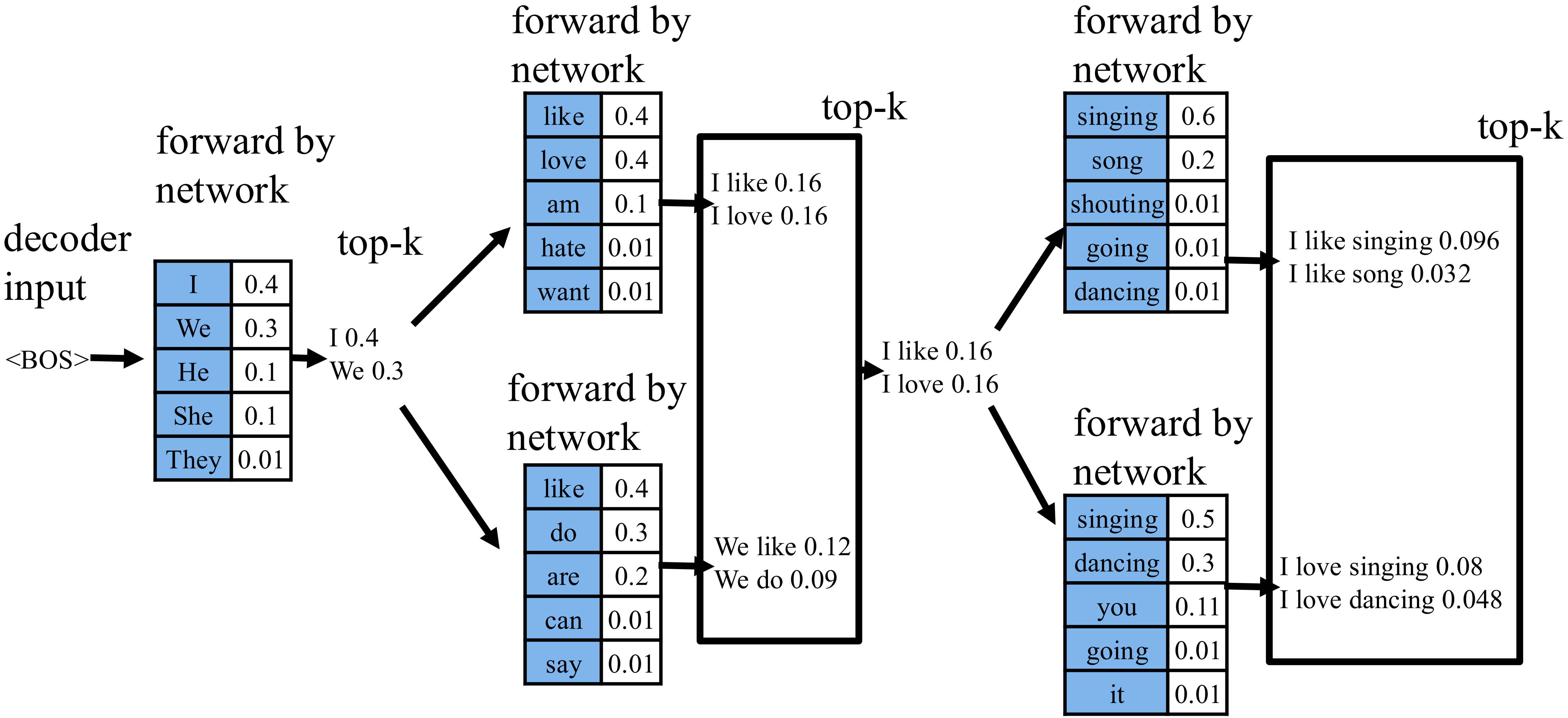
- Sequence Decoding overview
- Beam search algorithm
- Accelerating Generation: Speculative Decoding
- Further Improvement: EAGLE speculative decoding

Beam Search

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

1. start with empty 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

Beam Search



decoder input
<BOS>
forward by network
I 0.4
We 0.3
He 0.1
She 0.1
They 0.01
top-k
I 0.4
We 0.3
forward by network
like 0.4
love 0.4
am 0.1
hate 0.01
want 0.01
forward by network
like 0.4
do 0.3
are 0.2
can 0.01
say 0.01
top-k
I like 0.16
I love 0.16
We like 0.12
We do 0.09
I like 0.16
I love 0.16
forward by network
singing 0.6
song 0.2
shouting 0.01
going 0.01
dancing 0.01
forward by network
singing 0.5
dancing 0.3
you 0.11
going 0.01
it 0.01
top-k
I like singing 0.096
I like song 0.032
I love singing 0.08
I love dancing 0.048

```
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: $\text{score(cand)} \leq rp * \max\{\text{score}(c)\}$

- Absolute threshold pruning:
  - $\text{score(cand)} \leq \max\{\text{score}(c)\} - ap$

- Relative local threshold pruning

Freitag & Al-Onaizan. Beam Search Strategies for Neural Machine Translation. 2017.

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/llmsystem/llmsys_code_examples/blob/main/decoding/decoding.ipynb

Quiz 5.2

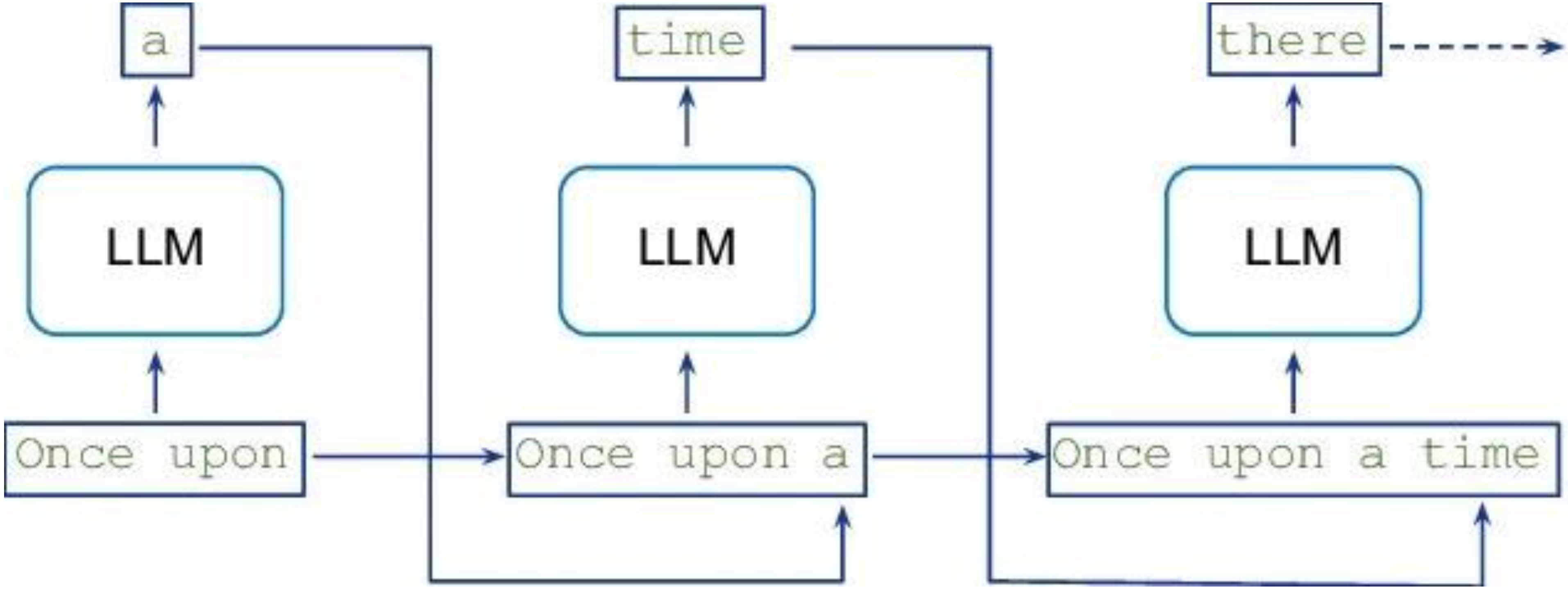
- on canvas

Outline

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

LLM Autoregressive Decoding is slow

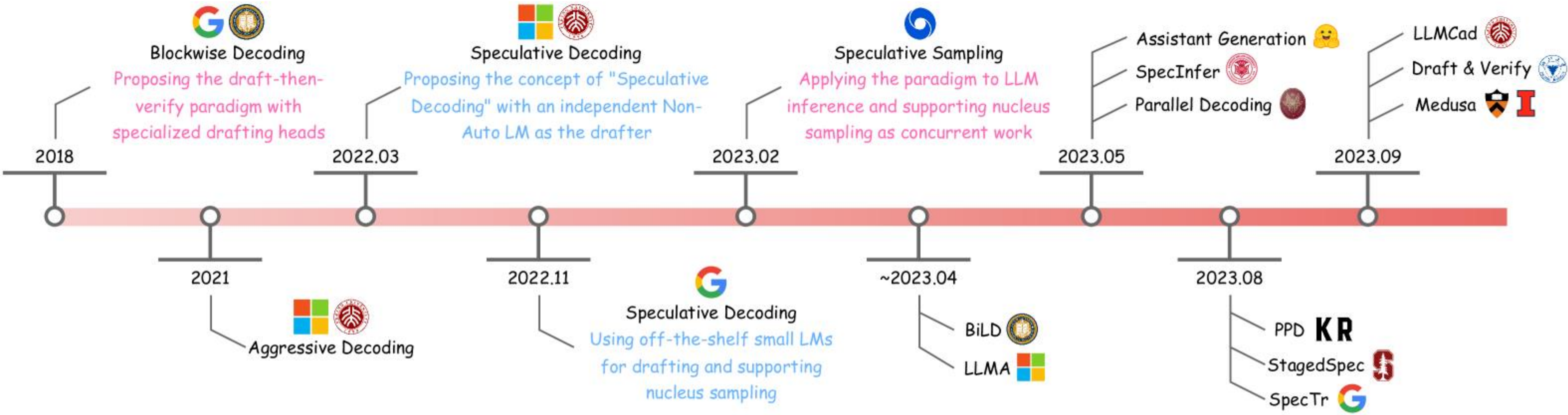
- need to generate one token at the time in a **sequential** manner and each token can take 100s of milliseconds



https://medium.com/ai-science/speculative-decoding-make-llm-inference-faster-c004501af120

Accelerating with Speculative Decoding

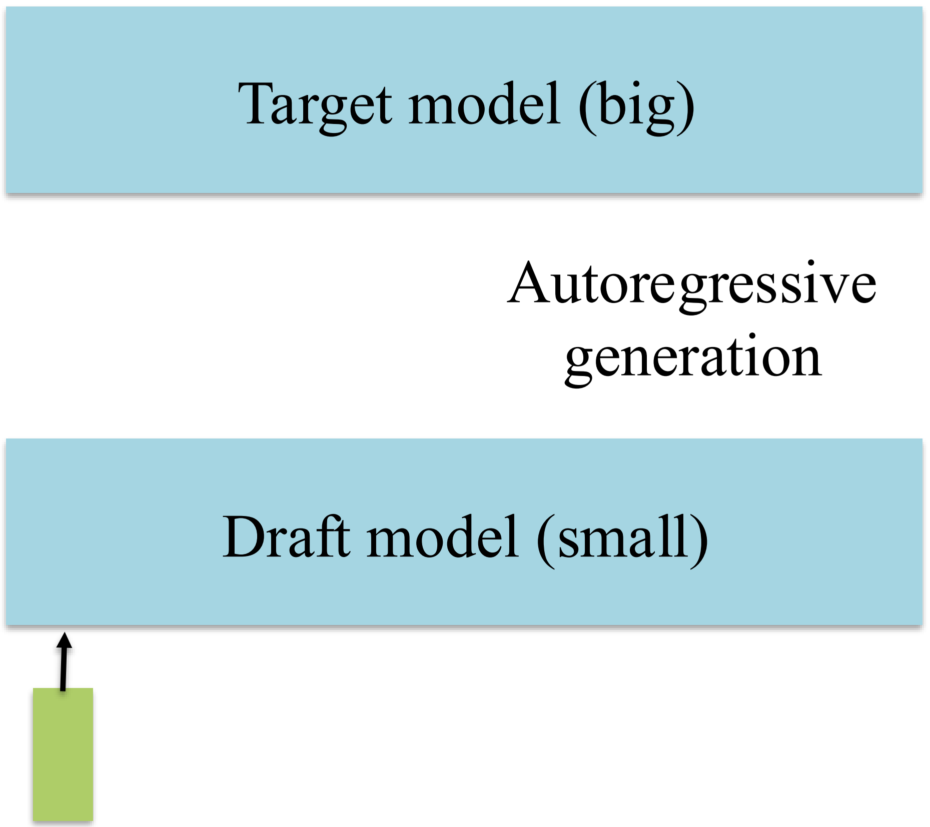
- Commonly used to reduce latency in LLMs inference applications



https://aclanthology.org/2024.findings-acl.456.pdf

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

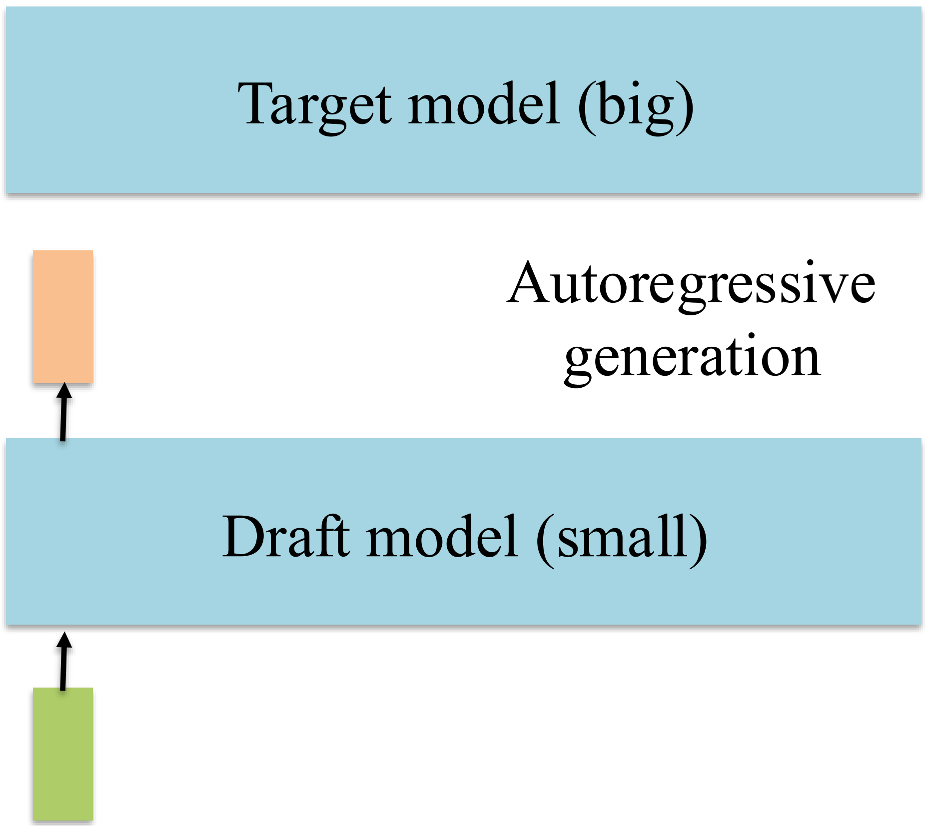


https://aclanthology.org/2024.findings-acl.456.pdf

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

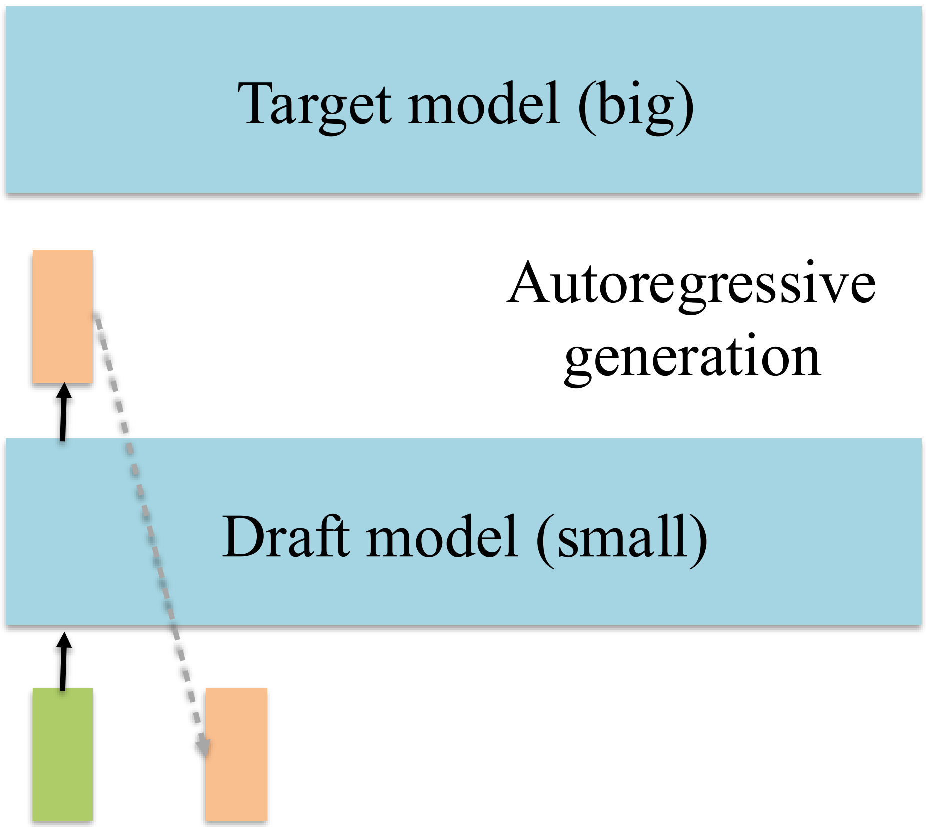
- $y_{1:k} = (y_1, y_2, \ldots, y_k) \sim f_{\text{draft}}(\cdot \mid x)$



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

- $y_{1:k} = (y_1, y_2, \ldots, y_k) \sim f_{\mathrm{draft}}(\cdot \mid x)$

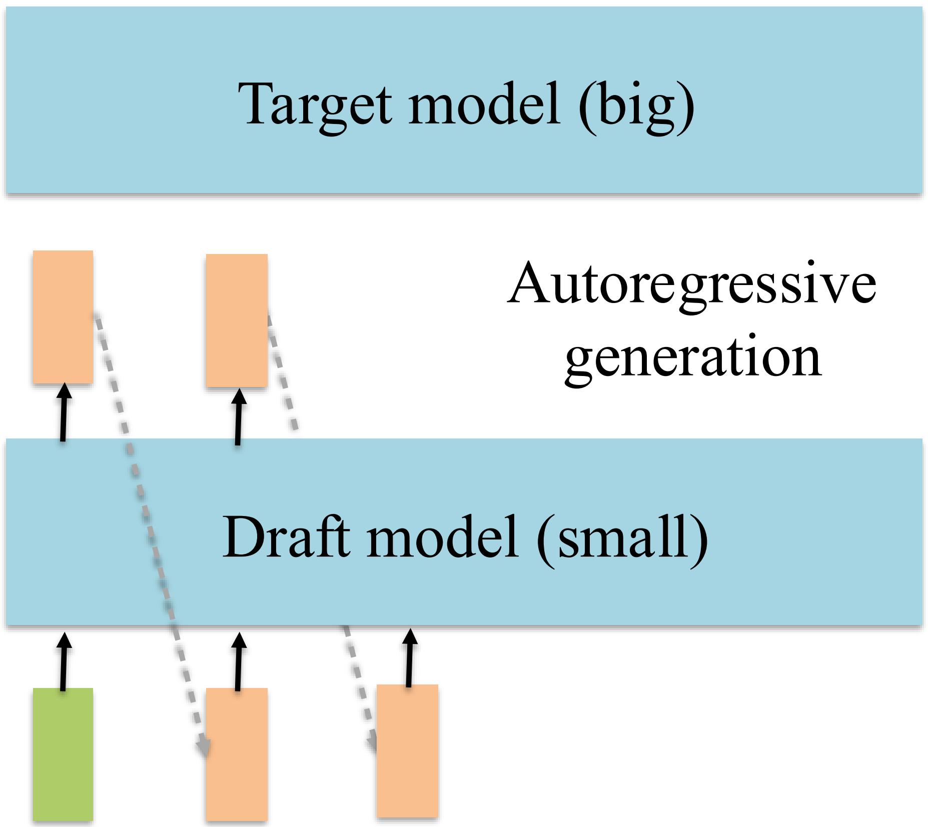


https://aclanthology.org/2024.findings-acl.456.pdf

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

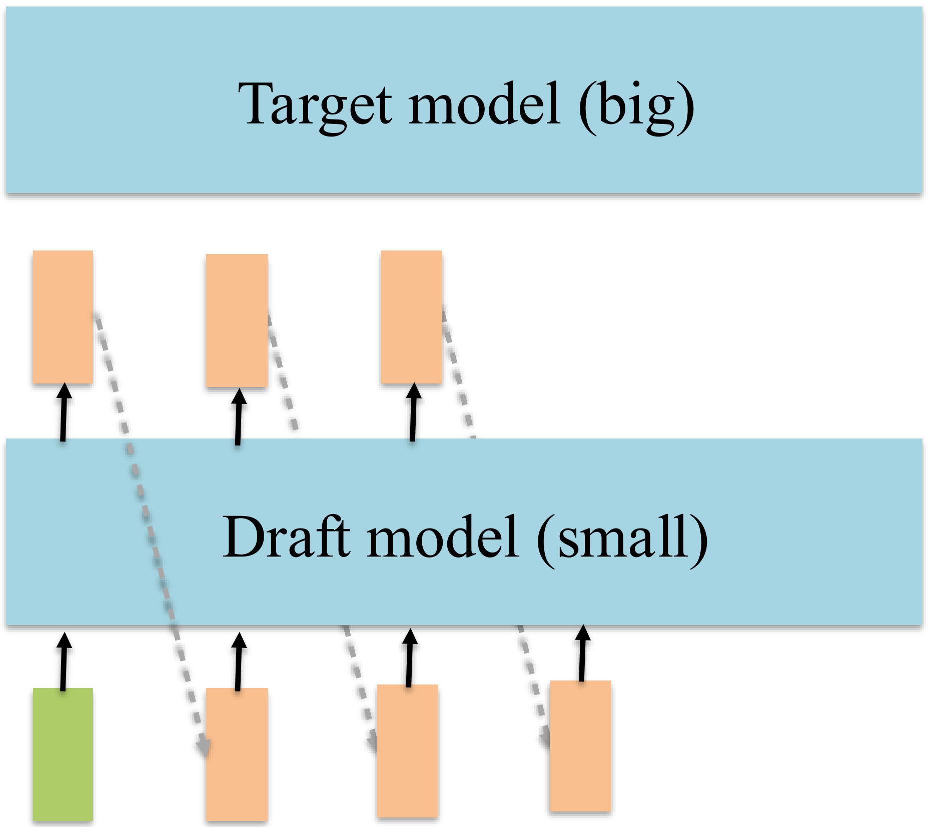
- $y_{1:k} = (y_1, y_2, \ldots, y_k) \sim f_{\text{draft}}(\cdot\,|x)$



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

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

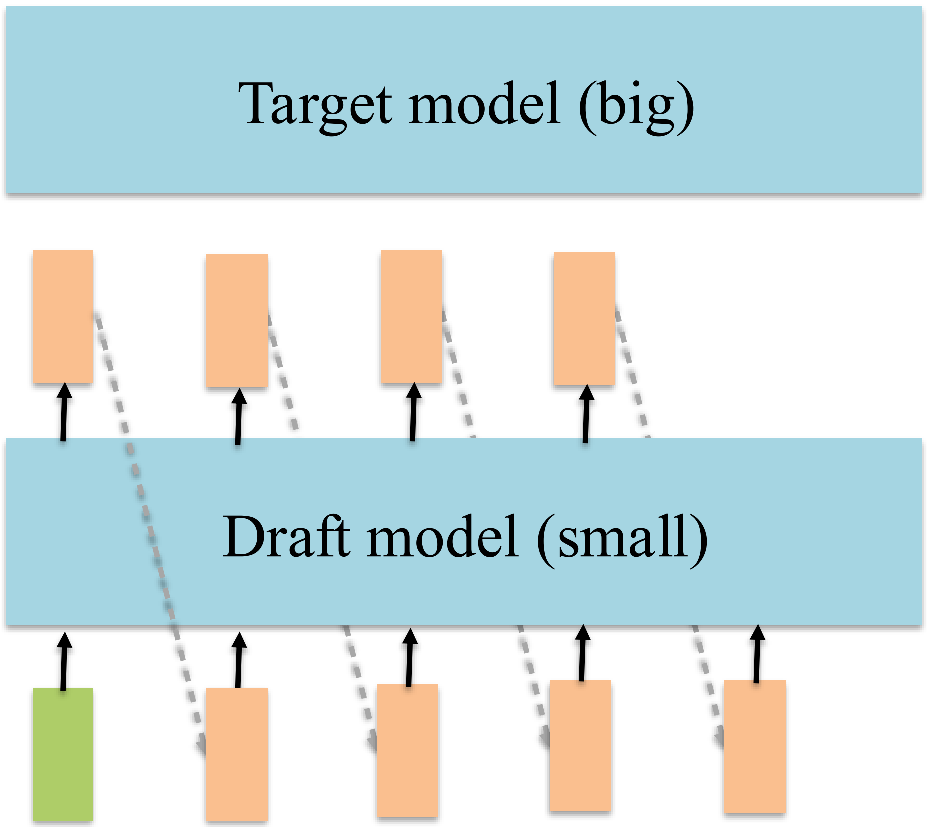


https://aclanthology.org/2024.findings-acl.456.pdf

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

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

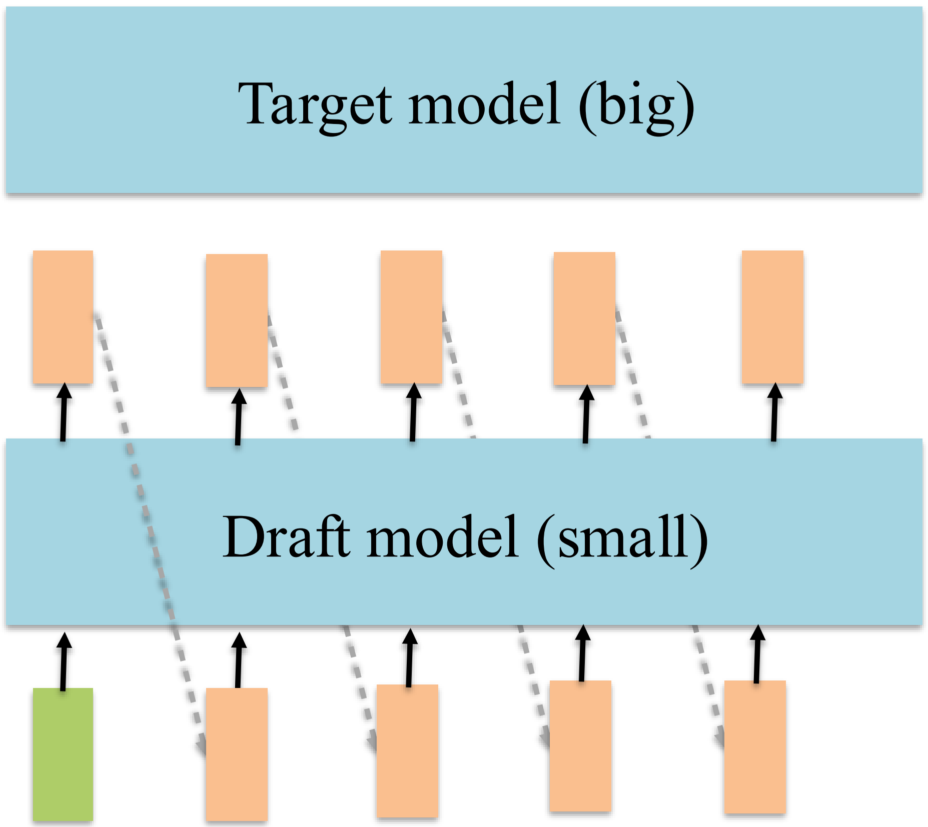


https://aclanthology.org/2024.findings-acl.456.pdf

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

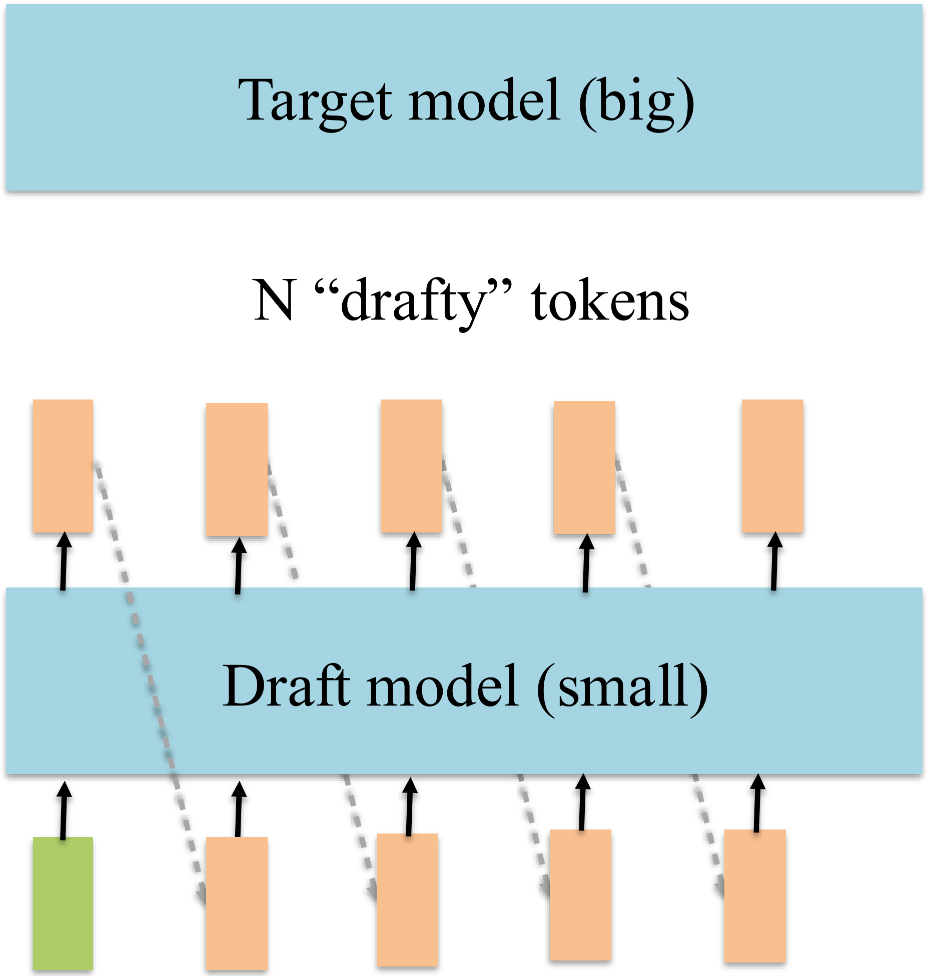
- $y_{1:k} = (y_1, y_2, \ldots, y_k) \sim f_{\text{draft}}(\cdot \mid x)$



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

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

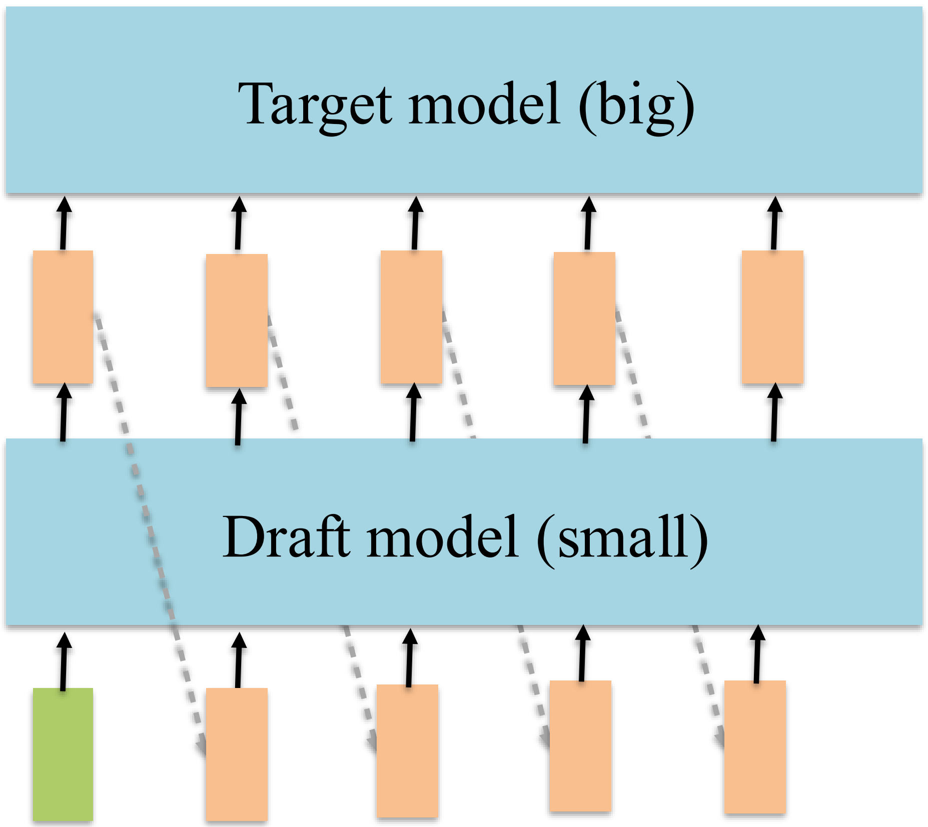


Speculative Decoding

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

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

Parallel validation of tokens



Speculative Decoding

- Each drafty token is considered valid if it is among the top-k predictions of the target LLM

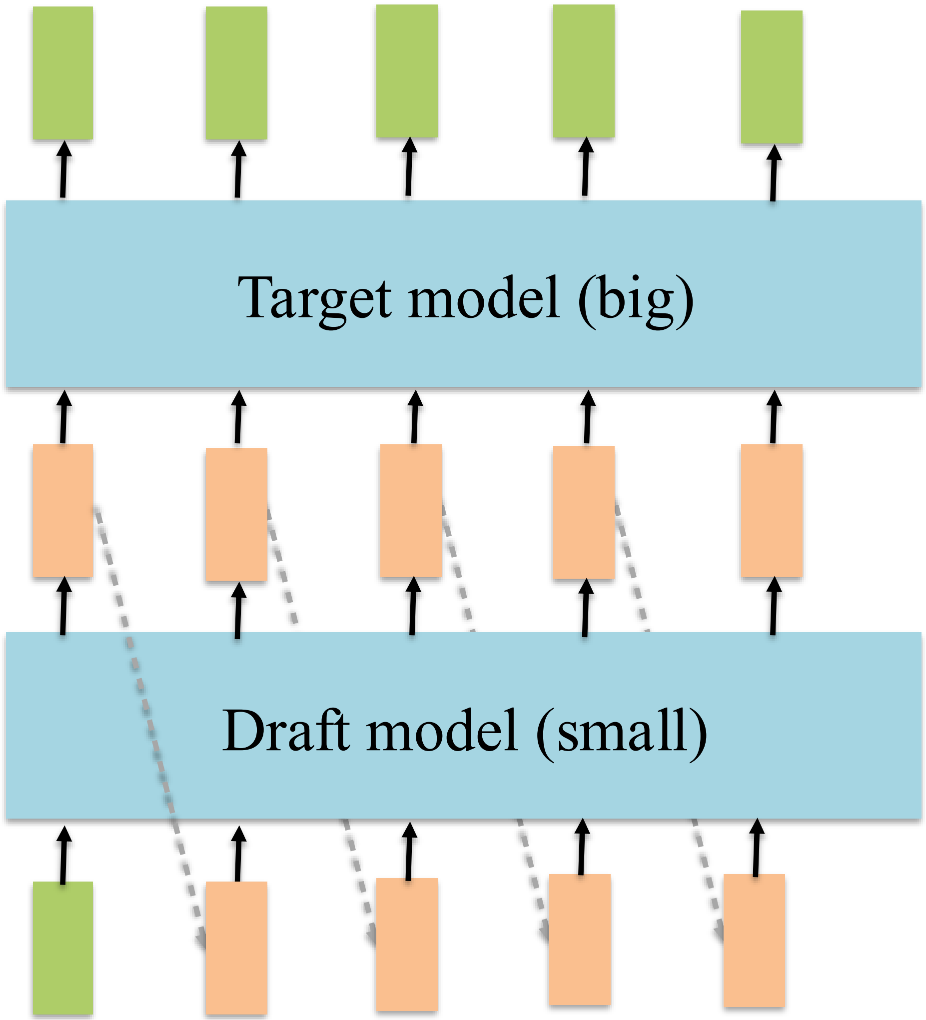
- It computes the probability of each draft token under its own distribution $f_{target}(\cdot \mid x, y_{1:i-1})$

- Accept token if $y_i \in TopK(f_{target}(\cdot \mid x, y_{1:i-1}))$

Speculative Decoding

- Target model accepts each token only if it lies in the top-k predictions

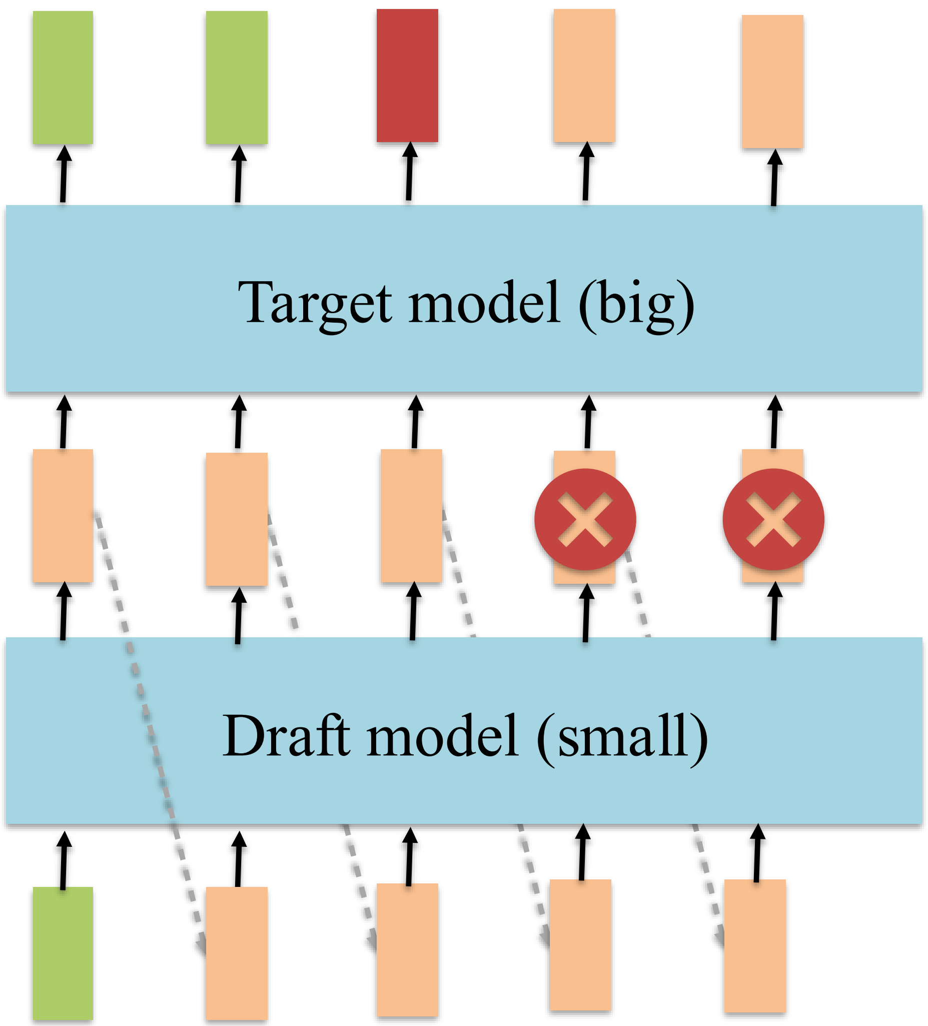
- Accept if $y_i \in TopK(f_{target}(\cdot\,|x, y_{1:i-1}))$



https://aclanthology.org/2024.findings-acl.456.pdf

Speculative Decoding

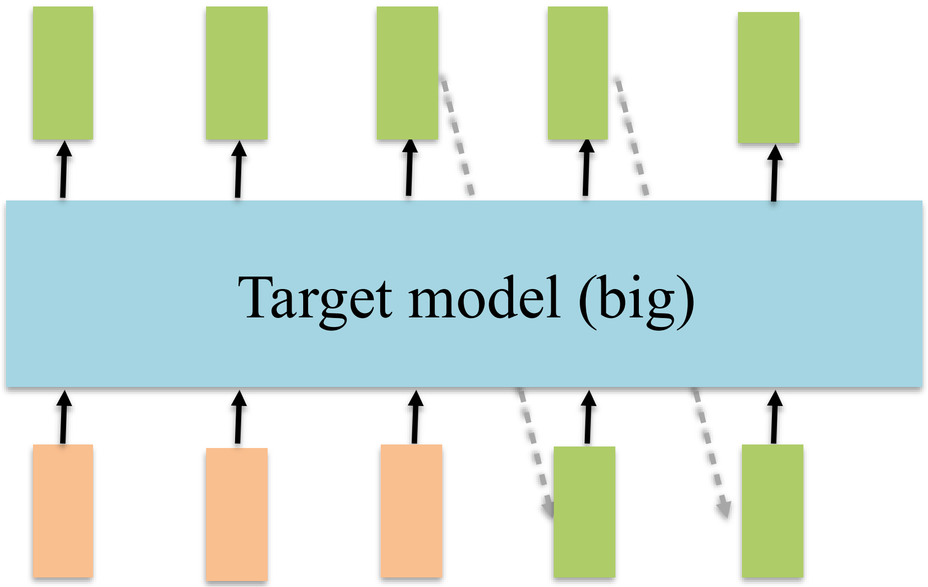
- If a drafty token is rejected, the target model will start generating on its own starting from the last accepted token



https://aclanthology.org/2024.findings-acl.456.pdf

Speculative Decoding

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

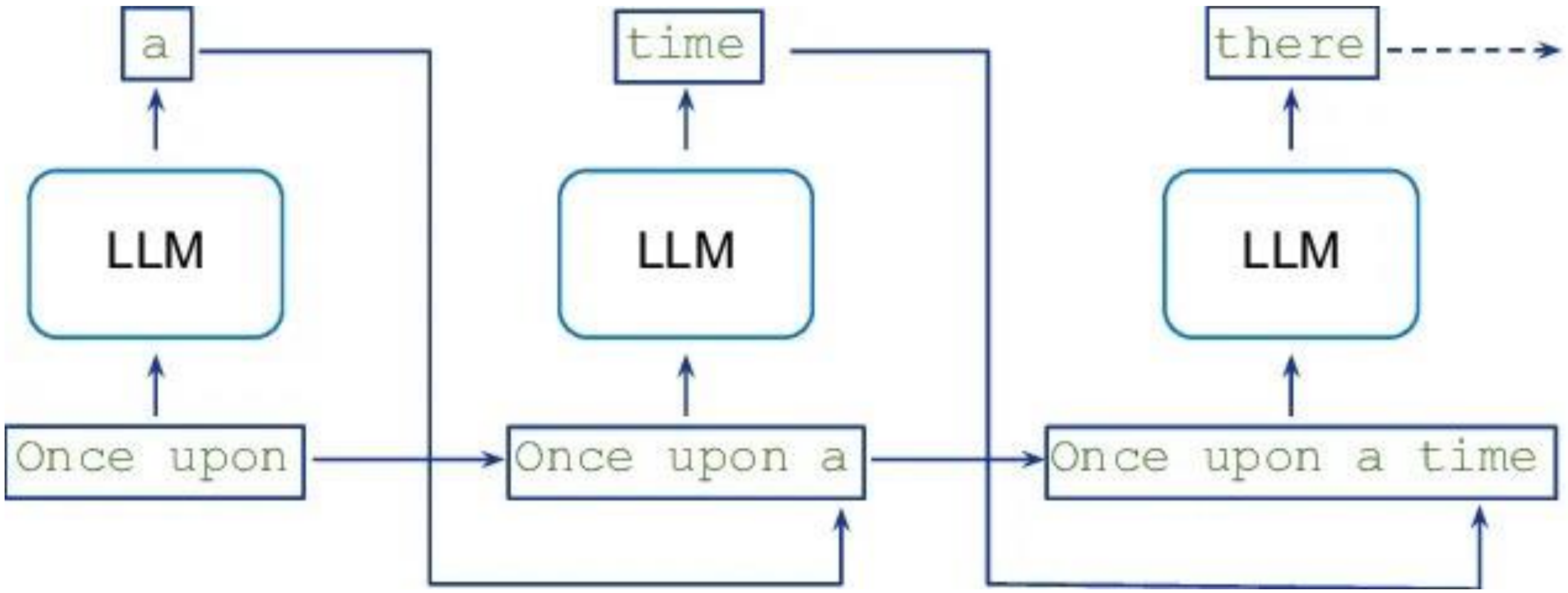


Why is Speculative Decoding faster?

- 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

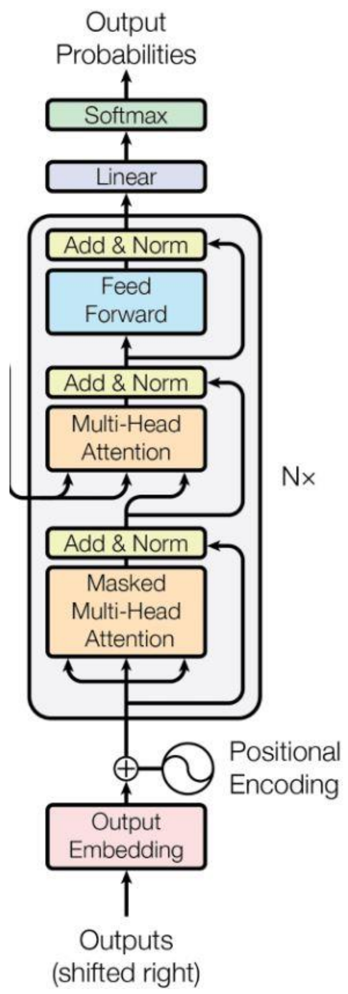


https://medium.com/ai-science/speculative-decoding-make-llm-inference-faster-c004501af120

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 Speed	EN→DE BLEU	DE→EN Speed	DE→EN BLEU	EN→RO Speed	EN→RO BLEU	RO→EN Speed	RO→EN 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**

https://aclanthology.org/2023.findings-emnlp.257.pdf

Speculative Decoding is faster

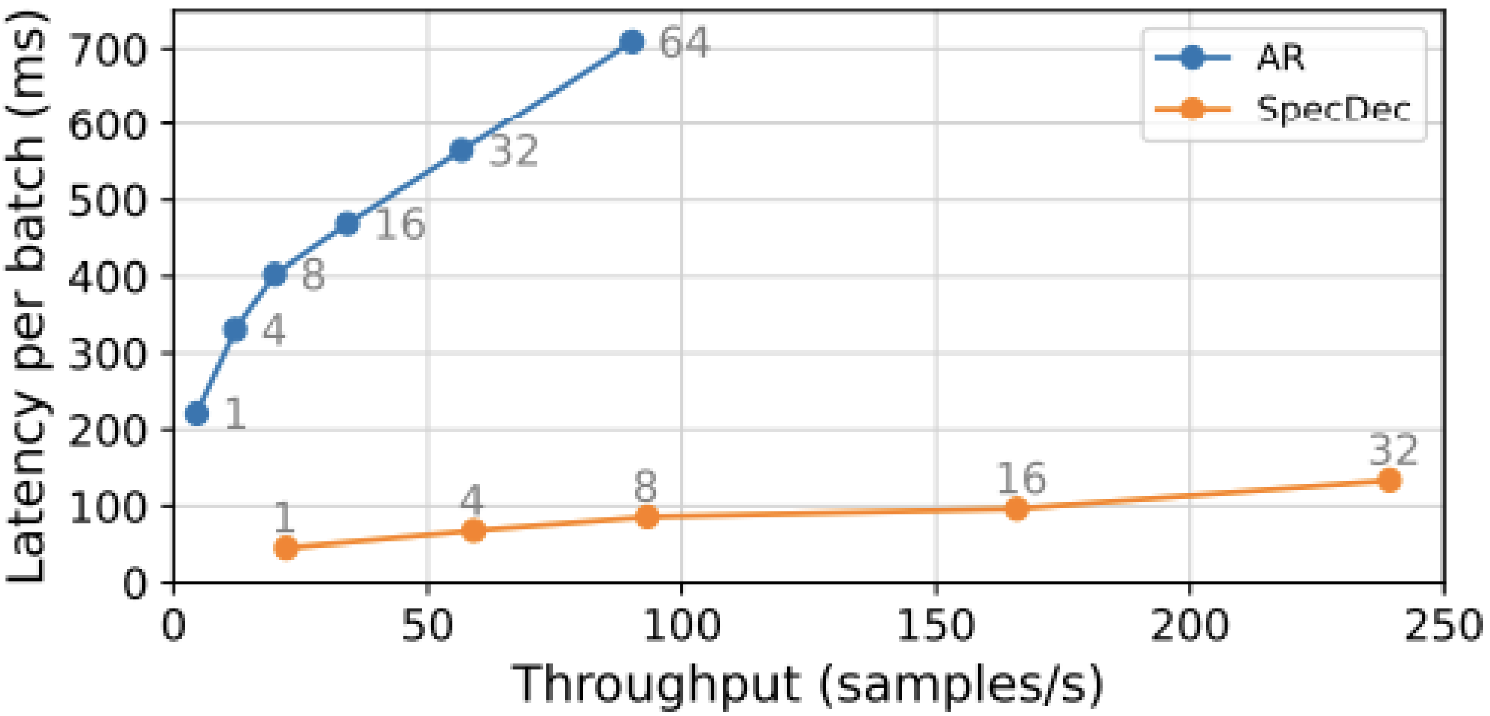


Figure 5: The latency-throughput curve with various batch sizes on WMT14 EN→DE.

https://aclanthology.org/2023.findings-emnlp.257.pdf

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

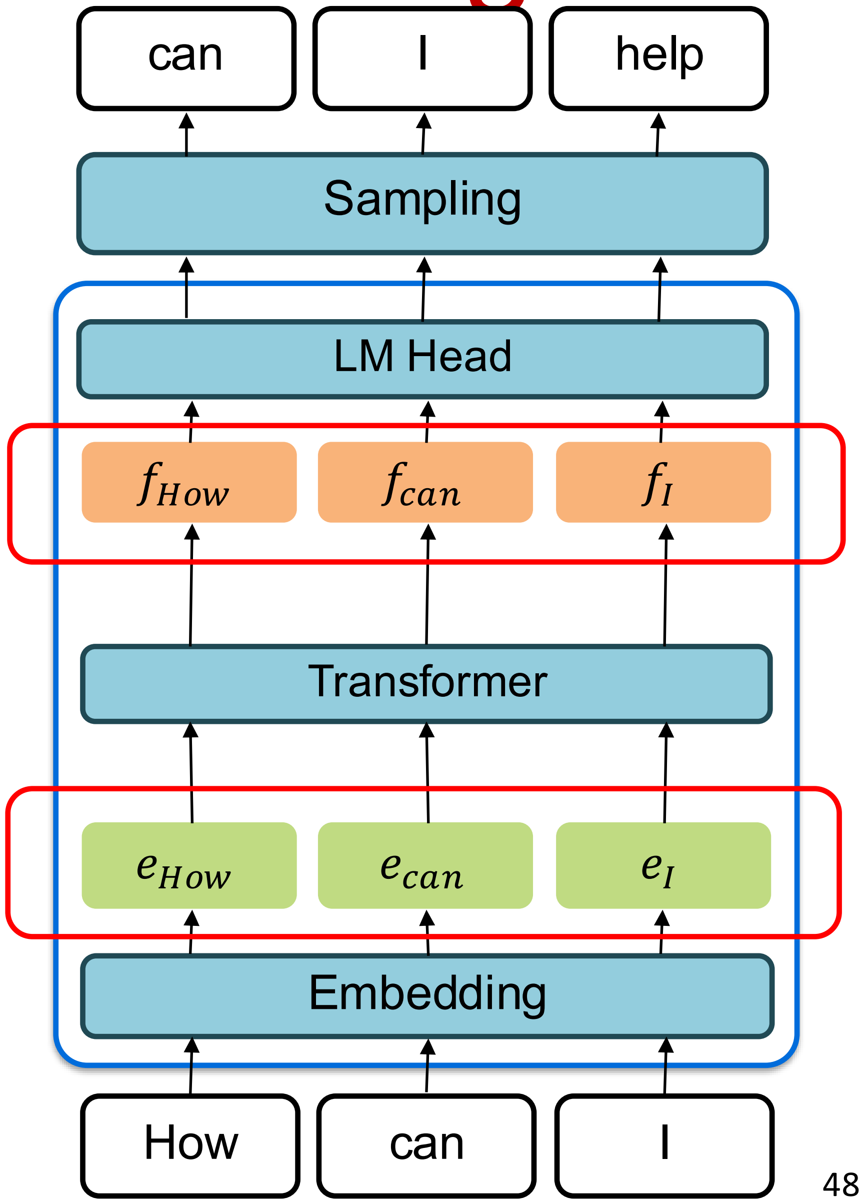
- Sequence Decoding overview
- Beam search algorithm
- Accelerating Generation: Speculative Decoding
- Further Improvement: EAGLE speculative decoding

Improving Speculative Decoding

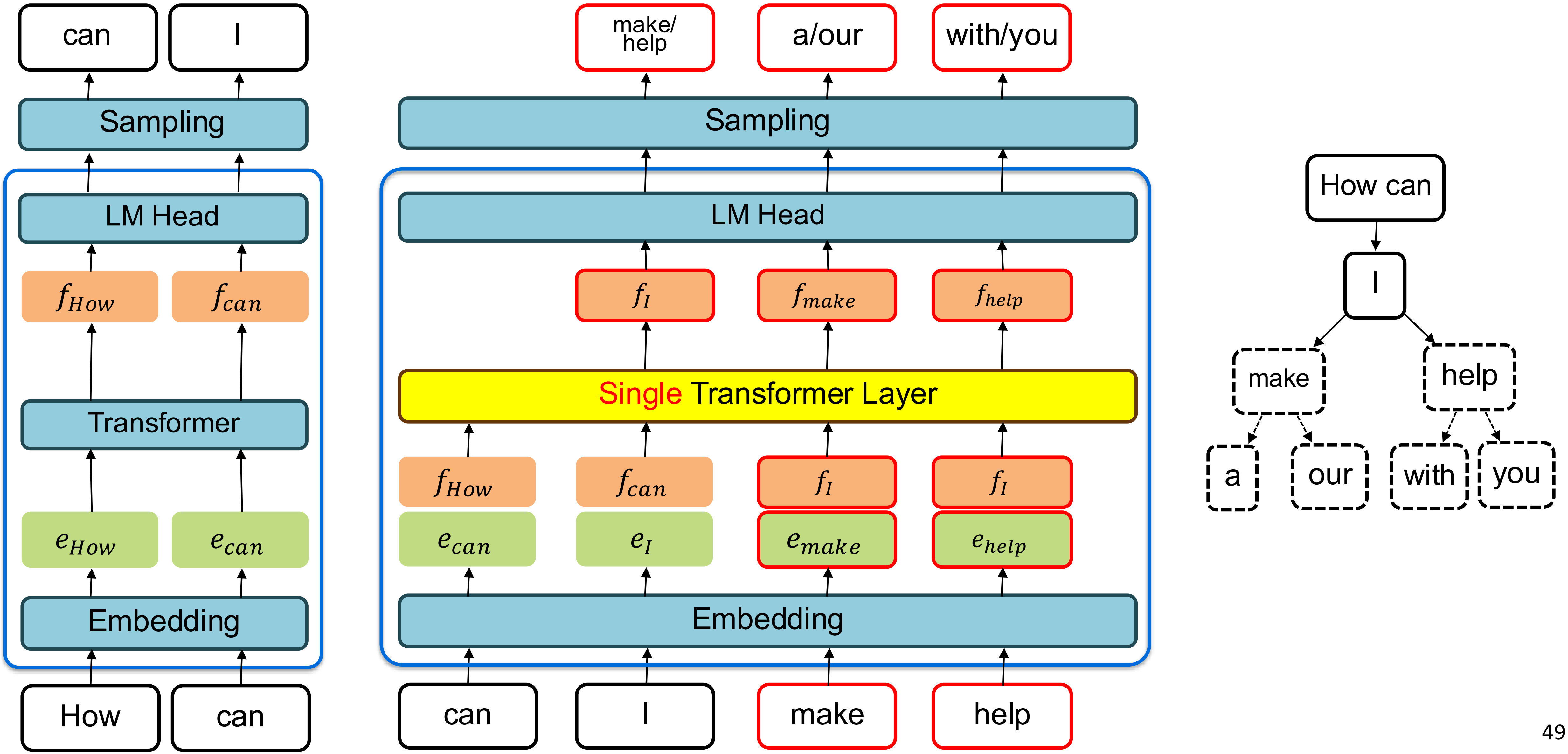
- Vanilla speculative decoding uses a small LM as drafting model which predicts the next token

- 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

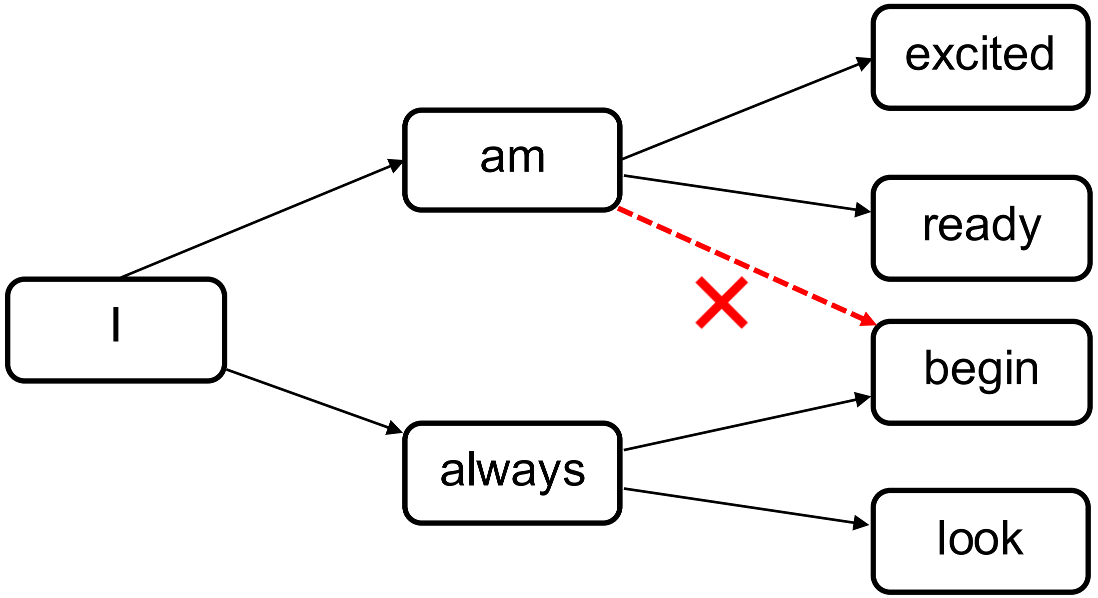


EAGLE Overview



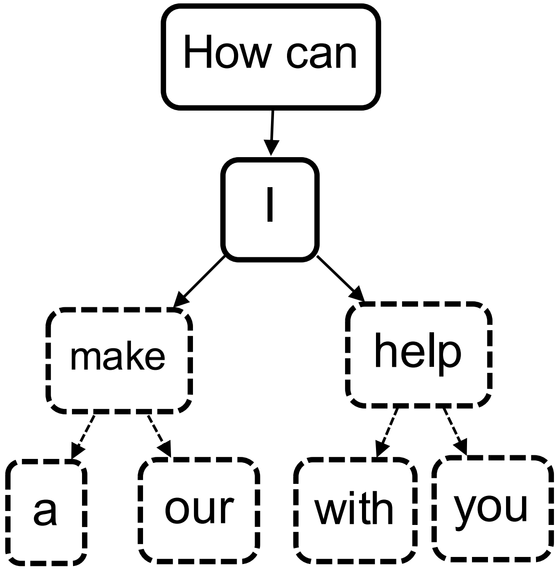
EAGLE: Word Embedding + Final Layer Feature

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

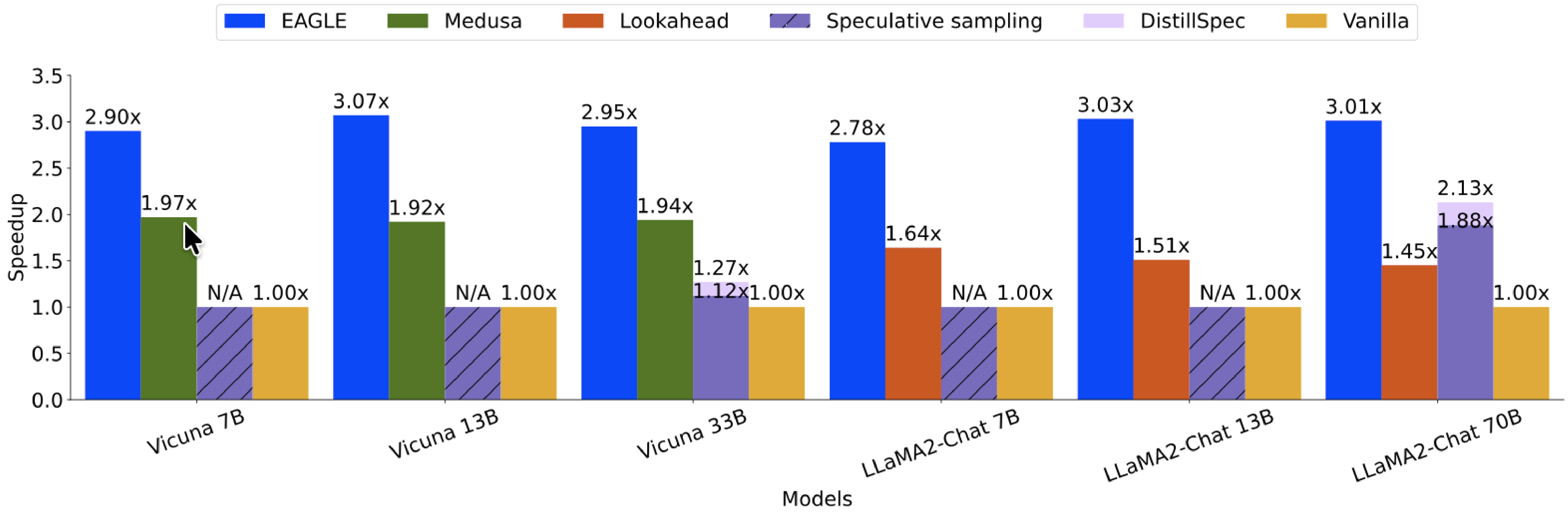
EAGLE Training

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

EAGLE Training

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

Much Faster Decoding on MT-Bench



Futher 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!

Demo

- https://github.com/llmsystem/llmsys_code_examples/blob/main/speculative_decoding/Speculative_decoding_demo.ipynb

- https://github.com/llmsystem/llmsys_code_examples/blob/main/speculative_decoding/EAGLE/demo.py

Quiz 5.3

- on canvas

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