THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

RELAXED SYSTEM LAB

# Speculative Decoding

COMP4901Y

Binhang Yuan

# Generative Inference

# Recall Generative Inference Workflow

RELAXED SYSTEM LAB

- State-of-the-art implementation splits the computation to two phrases:
  - Prefill phrase: the model takes a prompt sequence as input and engages in the generation of a key-value cache (KV cache) for each Transformer layer.
  - Decode phrase: for each decode step, the model updates the KV cache and reuses the KV to compute the output.
- Analyze the arithmetic intensity:
  - Prefill phrase: arithmetic bounded;
  - Decode phrase: memory bounded.
- Assume the computation is in fp16, the concrete analysis in next two slides.
  - $L$ is the input sequence length;
  - $D$ is the model dimension;
  - Multi-head attention $D = n_H \times H$; $H$ is the head dimension; $n_h$ is the number of heads.

# Prefill Phrase

| No. | Computation | Input | Output | Arithmetic Intensity |
|---|---|---|---|---|
| 1 | $Q = xW^Q$ | $x \in \mathbb{R}^{L\times D}$, $W^Q \in \mathbb{R}^{D\times D}$ | $Q \in \mathbb{R}^{L\times D}$ | $\frac{L\times D^2}{L\times D+D^2+L\times D}=\frac{L\times D}{2L+D}$ |
| 2 | $K = xW^K$ | $x \in \mathbb{R}^{L\times D}$, $W^K \in \mathbb{R}^{D\times D}$ | $K \in \mathbb{R}^{L\times D}$ | $\frac{L\times D^2}{L\times D+D^2+L\times D}=\frac{L\times D}{2L+D}$ |
| 3 | $V = xW^V$ | $x \in \mathbb{R}^{L\times D}$, $W^V \in \mathbb{R}^{D\times D}$ | $V \in \mathbb{R}^{L\times D}$ | $\frac{L\times D^2}{L\times D+D^2+L\times D}=\frac{L\times D}{2L+D}$ |
| 4 | $[Q_1, Q_2 \ldots, Q_{n_h}] = \text{Partition}_{-1}(Q)$ | $Q \in \mathbb{R}^{L\times D}$ | $Q_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$ | - |
| 5 | $[K_1, K_2 \ldots, K_{n_h}] = \text{Partition}_{-1}(K)$ | $K \in \mathbb{R}^{L\times D}$ | $K_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$ | - |
| 6 | $[V_1, V_2 \ldots, V_{n_h}] = \text{Partition}_{-1}(V)$ | $V \in \mathbb{R}^{L\times D}$ | $V_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$ | - |
| 7 | $\text{Score}_\text{i} = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \ldots n_h$ | $Q_i, K_i \in \mathbb{R}^{L\times H}$ | $\text{score}_\text{i} \in \mathbb{R}^{L\times L}$ | $\frac{L^2\times H}{L\times H+L^2+L\times H}=\frac{L\times H}{2H+L}$ |
| 8 | $Z_i = \text{score}_i V_i, i = 1, \ldots n_h$ | $\text{score}_\text{i} \in \mathbb{R}^{L\times L}$, $V_i \in \mathbb{R}^{L\times H}$ | $Z_i \in \mathbb{R}^{L\times H}$ | $\frac{L^2\times H}{L\times H+L^2+L\times H}=\frac{L\times H}{2H+L}$ |
| 9 | $Z = \text{Merge}_{-1}([Z_1, Z_2 \ldots, Z_{n_h}])$ | $Z_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$ | $Z \in \mathbb{R}^{L\times D}$ | - |
| 10 | $\text{Out} = ZW^O$ | $Z \in \mathbb{R}^{L\times D}$, $W^O \in \mathbb{R}^{D\times D}$ | $\text{Out} \in \mathbb{R}^{L\times D}$ | $\frac{L\times D^2}{L\times D+D^2+L\times D}=\frac{L\times D}{2L+D}$ |
| 11 | $A = \text{Out}\, W^1$ | $\text{Out} \in \mathbb{R}^{L\times D}$, $W^1 \in \mathbb{R}^{D\times 4D}$ | $A \in \mathbb{R}^{L\times 4D}$ | $\frac{4L\times D^2}{L\times D+4D^2+L\times 4D}=\frac{4L\times D}{5L+4D}$ |
| 12 | $A' = \text{relu}(A)$ | $A \in \mathbb{R}^{L\times 4D}$ | $A' \in \mathbb{R}^{L\times 4D}$ | - |
| 13 | $x' = A'W^2$ | $A' \in \mathbb{R}^{L\times 4D}$, $W^2 \in \mathbb{R}^{4D\times D}$ | $x' \in \mathbb{R}^{L\times D}$ | $\frac{4L\times D^2}{L\times D+4D^2+L\times 4D}=\frac{4L\times D}{5L+4D}$ |

# Decoding Phrase

| No | Computationt | Input | Output | Arithmetic Intensity |
|---|---|---|---|---|
| 1 | $Q = Q_d = tW^Q$ | $t \in \mathbb{R}^{1\times D}$, $W^Q \in \mathbb{R}^{D\times D}$ | $Q, Q_d \in \mathbb{R}^{1\times D}$ | $\frac{1\times D^2}{1\times D+D^2+1\times D} = \frac{D}{2+D}$ |
| 2 | $K_d = tW^K$ | $t \in \mathbb{R}^{1\times D}$, $W^K \in \mathbb{R}^{D\times D}$ | $K_d \in \mathbb{R}^{1\times D}$ | $\frac{1\times D^2}{1\times D+D^2+1\times D} = \frac{D}{2+D}$ |
| 3 | $K = \text{concat}(K_{\text{cache}}, K_d)$ | $K_{\text{cache}} \in \mathbb{R}^{L\times D}, K_d \in \mathbb{R}^{1\times D}$ | $K \in \mathbb{R}^{(L+1)\times D}$ | - |
| 4 | $V_d = tW^V$ | $t \in \mathbb{R}^{1\times D}$, $W^V \in \mathbb{R}^{D\times D}$ | $V_d \in \mathbb{R}^{1\times D}$ | $\frac{1\times D^2}{1\times D+D^2+1\times D} = \frac{D}{2+D}$ |
| 5 | $V = \text{concat}(V_{\text{cache}}, V_d)$ | $V_{\text{cache}} \in \mathbb{R}^{L\times D}, V_d \in \mathbb{R}^{1\times D}$ | $V \in \mathbb{R}^{(L+1)\times D}$ | - |
| 6 | $[Q_1, Q_2 \ldots, Q_{n_h}] = \text{Partition}_{-1}(Q)$ | $Q \in \mathbb{R}^{1\times D}$ | $Q_i \in \mathbb{R}^{1\times H}, i = 1, \ldots n_h$ | - |
| 7 | $[K_1, K_2 \ldots, K_{n_h}] = \text{Partition}_{-1}(K)$ | $K \in \mathbb{R}^{(L+1)\times D}$ | $K_i \in \mathbb{R}^{(L+1)\times H}, i = 1, \ldots n_h$ | - |
| 8 | $[V_1, V_2 \ldots, V_{n_h}] = \text{Partition}_{-1}(V)$ | $V \in \mathbb{R}^{(L+1)\times D}$ | $V_i \in \mathbb{R}^{(L+1)\times H}, i = 1, \ldots n_h$ | - |
| 9 | $\text{Score}_\text{i} = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \ldots n_h$ | $Q_i \in \mathbb{R}^{1\times H}, K_i \in \mathbb{R}^{(L+1)\times H}$ | $\text{score}_\text{i} \in \mathbb{R}^{1\times(L+1)}$ | $\frac{1\times(L+1)\times H}{1\times H+(L+1)\times H+L+1} = \frac{H+LH}{2H+L+LH+1}$ |
| 10 | $Z_i = \text{score}_i V_i, i = 1, \ldots n_h$ | $\text{score}_\text{i} \in \mathbb{R}^{1\times(L+1)}$, $V_i \in \mathbb{R}^{(L+1)\times H}$ | $Z_i \in \mathbb{R}^{1\times H}$ | $\frac{(L+1)\times H}{L+1+(L+1)\times H+1\times H} = \frac{H+LH}{2H+L+LH+1}$ |
| 11 | $Z = \text{Merge}_{-1}([Z_1, Z_2 \ldots, Z_{n_h}])$ | $Z_i \in \mathbb{R}^{1\times H}, i = 1, \ldots n_h$ | $Z \in \mathbb{R}^{1\times D}$ | - |
| 12 | $\text{Out} = ZW^O$ | $Z \in \mathbb{R}^{1\times D}$, $W^O \in \mathbb{R}^{D\times D}$ | $\text{Out} \in \mathbb{R}^{1\times D}$ | $\frac{1\times D^2}{1\times D+D^2+1\times D} = \frac{D}{2+D}$ |
| 13 | $A = \text{Out}\ W^1$ | $\text{Out} \in \mathbb{R}^{1\times D}$, $W^1 \in \mathbb{R}^{D\times 4D}$ | $A \in \mathbb{R}^{1\times 4D}$ | $\frac{1\times 4D^2}{1\times D+4D^2+1\times 4D} = \frac{4D}{5+4D}$ |
| 14 | $A' = \text{relu}(A)$ | $A \in \mathbb{R}^{1\times 4D}$ | $A' \in \mathbb{R}^{1\times 4D}$ | - |
| 15 | $t' = A'W^2$ | $A' \in \mathbb{R}^{1\times 4D}$, $W^2 \in \mathbb{R}^{4D\times D}$ | $t' \in \mathbb{R}^{1\times D}$ | $\frac{1\times 4D^2}{1\times 4D+4D^2+1\times D} = \frac{4D}{5+4D}$ |

# Optimization Goal

- Decrease the I/O volume:
  - Model compression (last time):
    - Model quantization;
    - Knowledge distillation.
- Increase the computation load for each generation step:
  - speculative decoding (today).

# Speculative Decoding

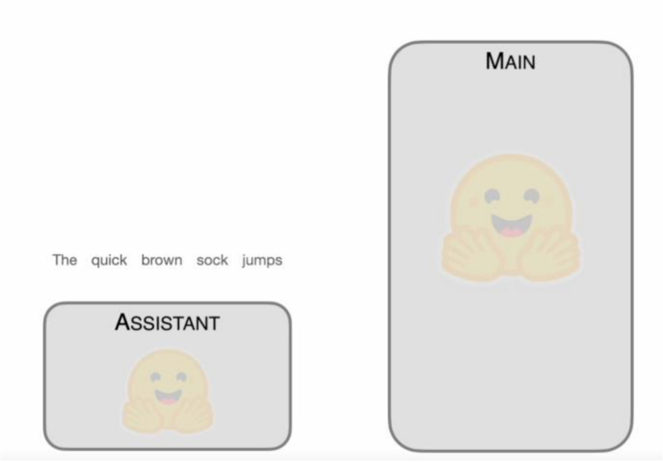
# Speculative Decoding Overview

RELAXED SYSTEM LAB

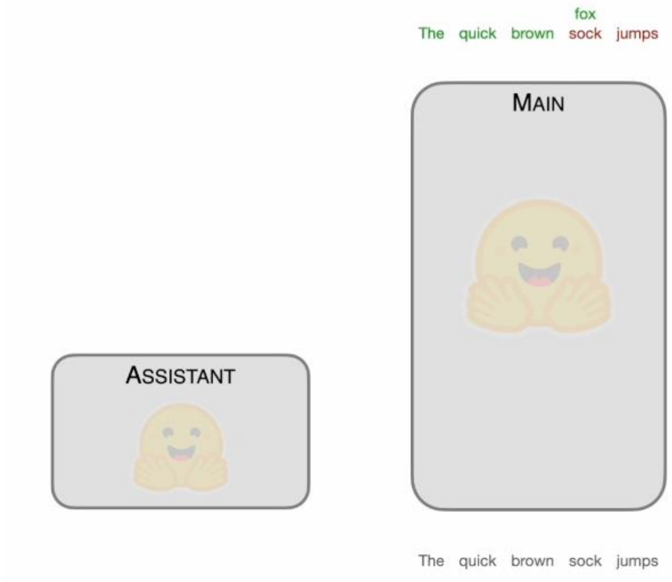
- **Observation**:
  - A small *assistant/speculative model* very often generates the same tokens as the large original LLM (sometime s referred to as the *main/target model*).
- Speculative decoding overview:
  - The assistant model auto-regressively generates a sequence of $N$ candidate tokens;
  - The candidate tokens are passed to the original LLM to be verified. The original model takes the candidate tokens as input and performs ***a single forward pass***:
    - All candidate tokens up to the first mismatch are correct;
    - After the first mismatch, replace the first incorrect candidate token with the correct token from the main model (fox), and discard all predicted tokens that come after this mismatched token.
  - Repeat this process until the end condition is reached.

# Speculative Decoding Example

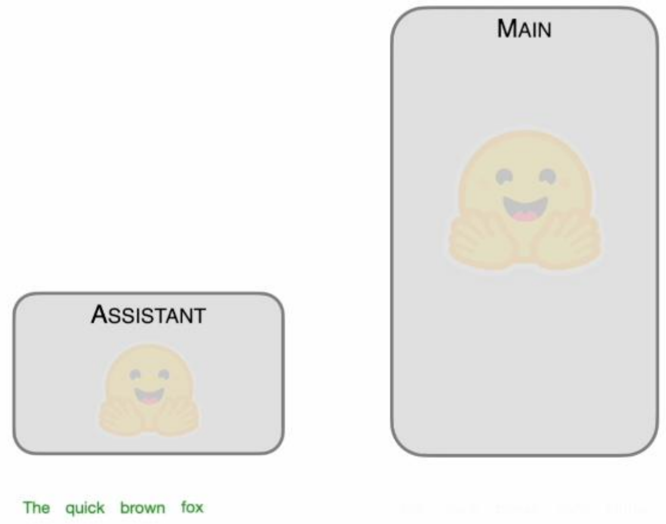





1. The assistant model auto-regressively generates sequence: [The quick brown sock jumps].




2. The first three tokens predicted by the original LLM agree with those from the assistant model: [The quick brown]. However, the fourth candidate token from the assistant model (sock), mismatches with the correct token from the main model (fox).




3. We replace the first incorrect candidate token (sock) with the correct token from the main model (fox) and discard all predicted tokens that come after this. The corrected sequence, [The quick brown fox] now forms the new input to the assistant model for the next step.

# Speculative Decoding Notes

RELAXED SYSTEM LAB

- We auto-regressively generate using the fast, assistant model, and only perform verification forward passes with the slow, main model, the decoding process is sped-up substantially.
- The verification forward passes performed by the main model ensures that *exactly the same outputs are achieved* as if we were using the main model standalone.
- Trade-off:
  - "The assistant model should be significantly faster." V.S. "The assistant model should predict the same token distribution as often as possible."
  - Since 70-80% of all predicted tokens tend to be "easier" tokens, this trade-off is heavily biased towards selecting a faster model, rather than a more accurate one.

# Speculative Decoding in HF inference API

RELAXED SYSTEM LAB

**Example Code**

```
from transformers import AutoModelForCausalLM, AutoTokenizer

prompt = "What is Apple?"

model_id = "EleutherAI/pythia-160m"
assistant_model_id = "EleutherAI/pythia-14m"

tokenizer = AutoTokenizer.from_pretrained(model_id)
inputs = tokenizer(prompt, return_tensors="pt")

model = AutoModelForCausalLM.from_pretrained(model_id)
assistant_model = AutoModelForCausalLM.from_pretrained(assistant_model_id)

outputs = model.generate(**inputs, max_new_tokens=20, assistant_model=assistant_model,
return_dict_in_generate=True)

input_length = inputs.input_ids.shape[1]
token = outputs.sequences[0, input_length+1:]
print(f"[INFO] raw token: {token}")
output = tokenizer.decode(token)
print(f"[Context]: {prompt} \n[Output]:{output}\n")
```

# How to do sampling in Speculative Decoding? *

RELAXED SYSTEM LAB

- We have two distribution now:
  - $x \sim p(x)$ sampling from the distribution of original model;
  - $x \sim q(x)$ sampling from the distribution of the assistant model;
- Speculative sampling:
  - To sample $x \sim p(x)$,
  - We instead sample from $x \sim q(x)$:
    - If $q(x) \leq p(x)$, keep $x$;
    - Otherwise ($q(x) > p(x)$), reject the sample with probability $1 - \frac{p(x)}{q(x)}$ and sample $x$ again from an adjusted distribution $p'(x)$.

$$p'(x) = \frac{p(x) - \min(q(x), p(x))}{\sum_{x'}(p(x') - \min(q(x'), p(x')))}$$

arXiv:2211.17192v2 [cs.LG] 18 May 2023

**Fast Inference from Transformers via Speculative Decoding**

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

**Abstract**

Inference from large autoregressive models like Transformers is slow - decoding $K$ tokens takes $K$ serial runs of the model. In this work we introduce *speculative decoding* - an algorithm to sample from autoregressive models faster *without any changes to the outputs*, by computing several tokens in parallel. At the heart of our approach lie the observations that (1) hard language-modeling tasks often include easier subtasks that can be approximated well by more efficient models, and (2) using speculative execution and a novel sampling method, we can make exact decoding from the large models faster, by running them in parallel on the outputs of the approximation models, potentially generating several tokens concurrently, and without changing the distribution. Our method can accelerate existing off-the-shelf models without retraining or architecture changes. We demonstrate it on T5-XXL and show a **2X-3X** acceleration compared to the standard T5X implementation, with identical outputs.

**1. Introduction**

Large autoregressive models, notably large Transformers (Vaswani et al., 2017), are much more capable than smaller models, as is evidenced countless times in recent years e.g., in the text or image domains, like GPT-3 (Brown et al., 2020), LaMDA (Thoppilan et al., 2022), Parti (Yu et al., 2022), and PaLM (Chowdhery et al., 2022). Unfortunately, a single decode step from these larger models is significantly slower than a step from their smaller counterparts, and making things worse, these steps are done serially - decoding $K$ tokens takes $K$ serial runs of the model.

Given the importance of large autoregressive models and specifically large Transformers, several approaches were 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 resources for easier inference steps. While many of these solutions have proven extremely effective in practice, they usually require changing the model architecture, changing the training-procedure and re-training the models, and don't maintain identical outputs.

The key observation above, that some inference steps are "harder" and some are "easier", is also a key motivator for our work. We additionally observe that inference from large models is often not bottlenecked on arithmetic operations, but rather on memory bandwidth and communication, so additional computation resources might be available. Therefore we suggest increasing concurrency as a complementary approach to using an adaptive amount of computation. Specifically, we are able to accelerate inference without changing the model architectures, without changing the training-procedures or needing to re-train the models, and without changing the model output distribution. This is accomplished via *speculative execution*.

Speculative execution (Burton, 1985; Hennessy & Patterson, 2012) is an optimization technique, common in processors, where a task is performed in parallel to verifying if it's actually needed - the payoff being increased concurrency. A well-known example of speculative execution is branch prediction. For speculative execution to be effective, we need an efficient mechanism to suggest tasks to execute that are likely to be needed. In this work, we generalize speculative execution to the stochastic setting - where a task *might be* needed with some probability. Applying this to decoding from autoregressive models like Transformers, we sample generations from more efficient *approximation models* as speculative prefixes for the slower *target models*. With a novel sampling method, *speculative sampling*, we maximize the probability of these speculative tasks to

*Equal contribution [1]Google Research, Mountain View, CA, USA. Correspondence to: Yaniv Leviathan <leviathan@google.com>.

*Proceedings of the 40th International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. 

https://arxiv.org/pdf/2211.17192.pdf

# Speculative Decoding Optimization

- If the assistant model is asked to output the length $m$ draft sequence, and the original LLM accepts $n, n < m$, the $(m - n)$ tokens are automatically discarded.
- If $n \ll m$, every LLM forward leads to only a limited number of tokens being decoded.
- Optimization, e.g., [SpecInfer https://arxiv.org/pdf/2305.09781.pdf]:
  - Let the assistant model(s) sample multiple plausible draft sequences for the original LLM to evaluate in parallel.
  - Sampling from multiple small assistant model drafts.
  - Organize the draft sequences as a tree to reuse the KV-cache. [Tree Attention]

RELAXED
SYSTEM LAB

# Parallel Decoding

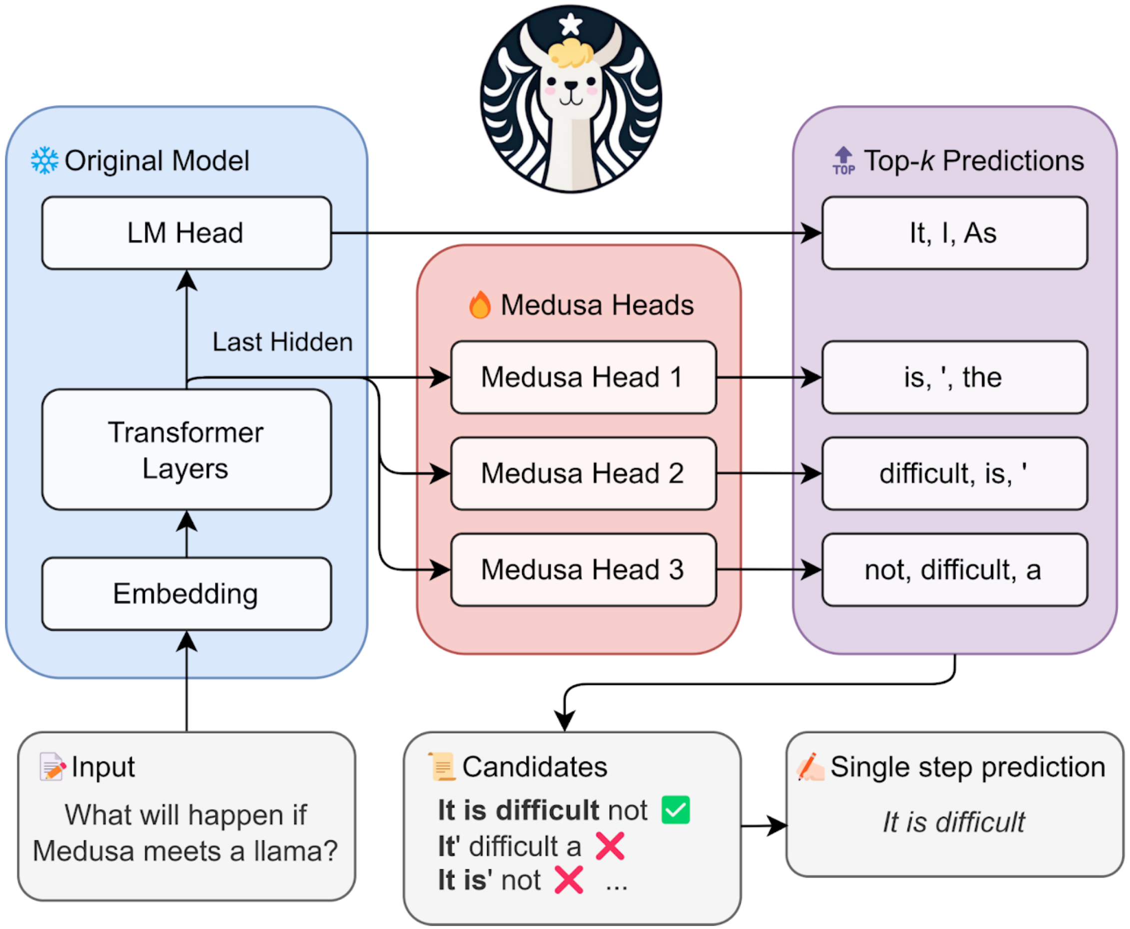
# Parallel Decoding Overview

RELAXED SYSTEM LAB

- Speculative decoding generates multiple token sequences using the assistant model for verification by the original LLM.
- Parallel decoding enables multiple token predictions directly from one forward pass of the original LLM.
- How to enable this?
  - [Medusa https://arxiv.org/pdf/2401.10774.pdf, https://github.com/FasterDecoding/Medusa]
    - Medusa heads (last layer projection);
    - Tree Attention.

# Medusa Heads

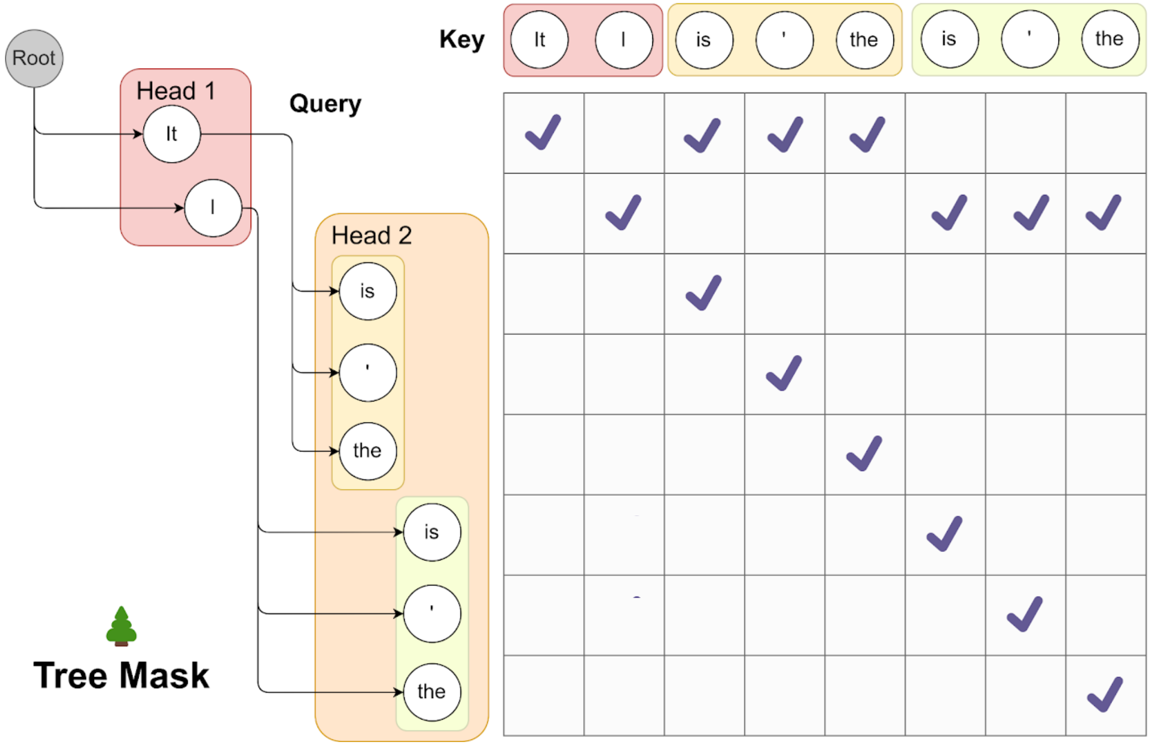
- Rather than pulling in an entirely new assistant model to predict subsequent tokens, Medusa simply extends the original LLM itself.
- Medusa heads are the additional decoding heads built on top of the last hidden states of the LLM, enabling the prediction of several subsequent tokens in parallel.
- Each Medusa head is a single layer of feed-forward network, augmented with a residual connection.
- Training these heads is straightforward: for a relatively small dataset, the original model remains static; only the Medusa heads are updated.




https://www.together.ai/blog/medusa

# Tree Attention

- Tree Attention in Medusa:
  - The top-1 accuracy for predicting the 'next-next' token hovers around 60%;
  - The top-5 accuracy soars to over 80%.
  - This substantial increase indicates that leveraging the multiple top-ranked predictions made by the Medusa heads can significantly amplify the number of tokens generated per decoding step.
  - Construct the Cartesian product of the top predictions from each Medusa head and encode the dependency graph into the attention.

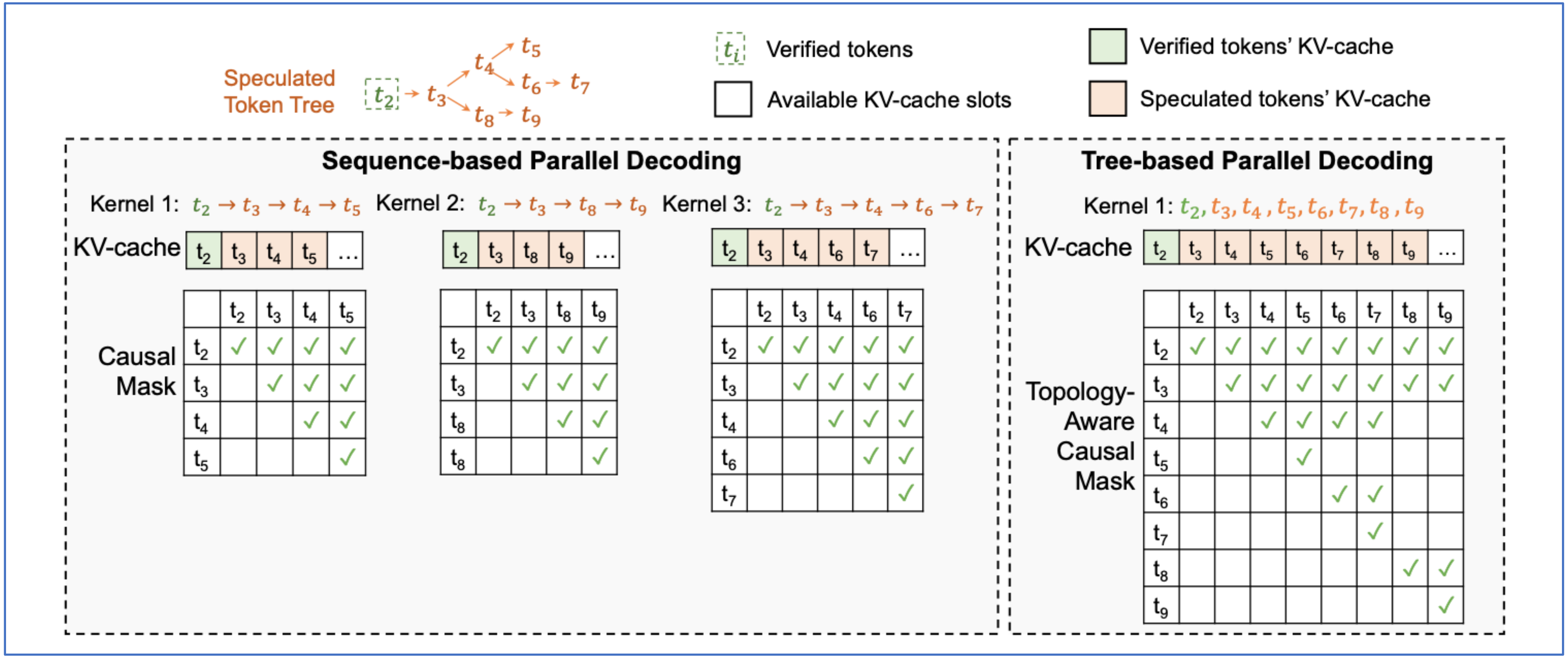



Modified from https://www.together.ai/blog/medusa to avoid confusion when compared with SpecInfer

# Tree Attention



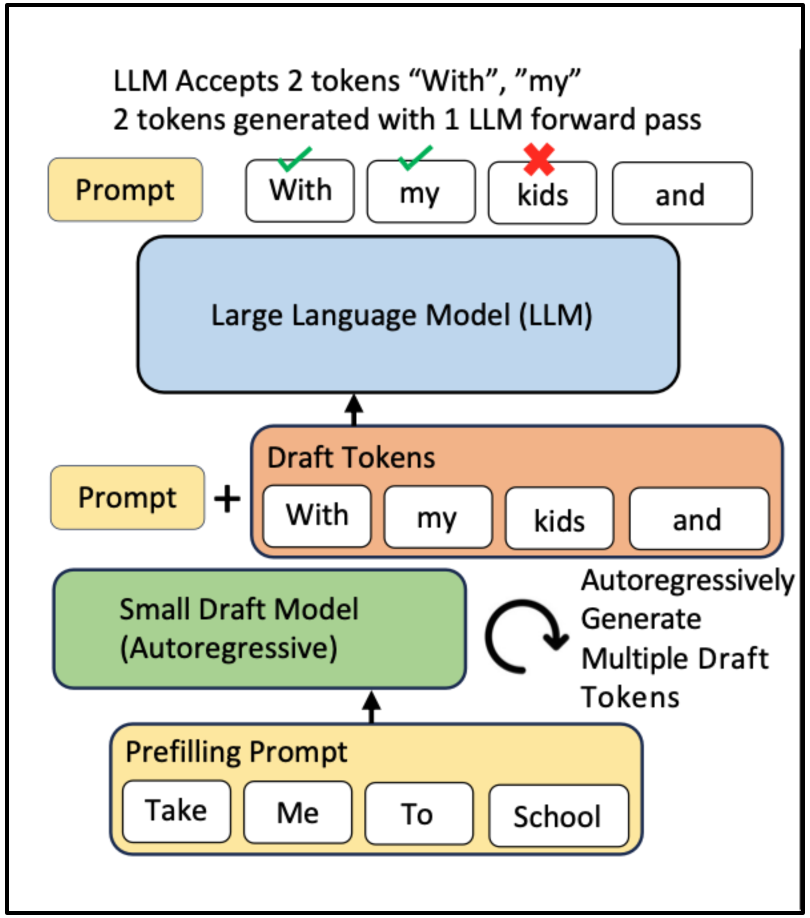
- A similar idea can also be leveraged in speculative decoding when you organize multiple sequences as a prefix tree.



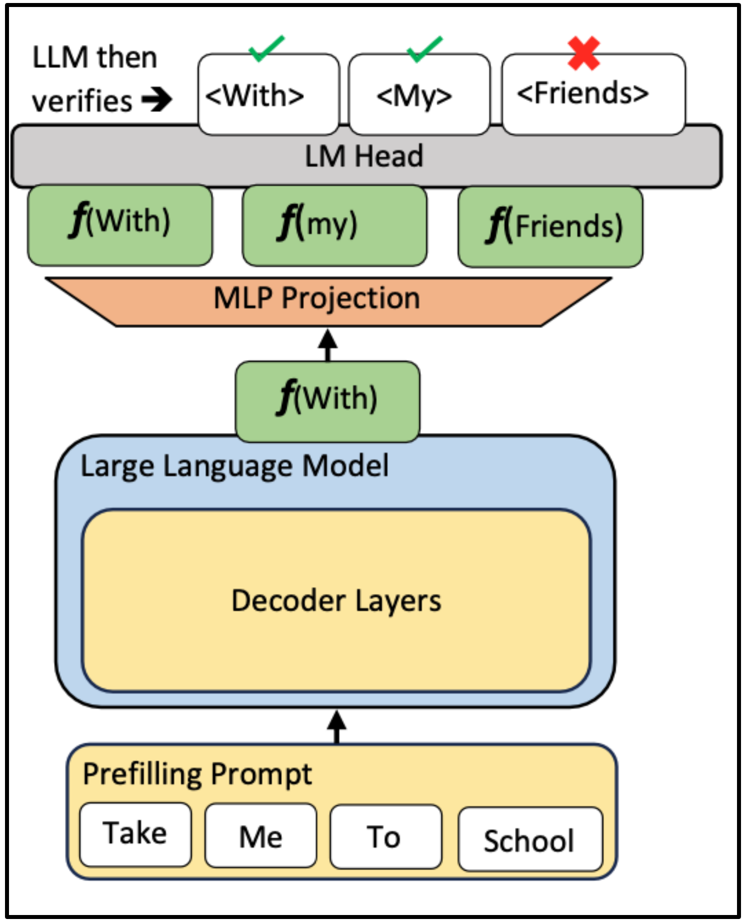

[Figure from SpecInfer.]

# Speculative Decoding v.s. Parallel Decoding

RELAXED SYSTEM LAB




Speculative Decoding




Parallel Decoding

https://arxiv.org/abs/2402.16363

# References


- https://huggingface.co/blog/whisper-speculative-decoding
- https://arxiv.org/pdf/2211.17192.pdf
- https://www.together.ai/blog/medusa
- https://arxiv.org/abs/2402.16363