Online Speculative Decoding

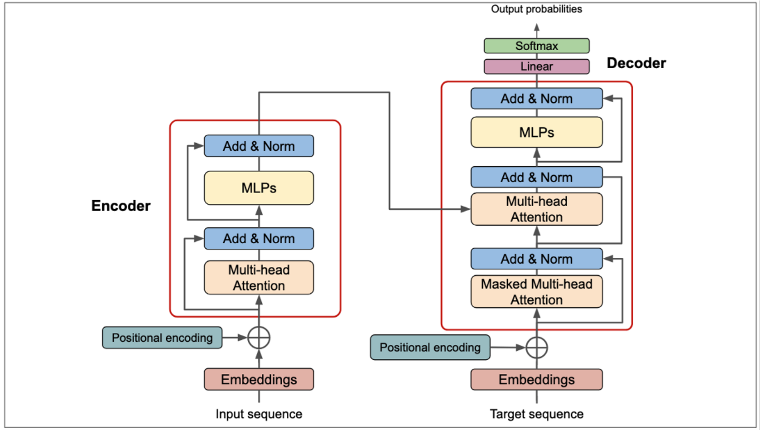
Zhijie Deng

* Joint work with Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Alvin Cheung, and Hao Zhang

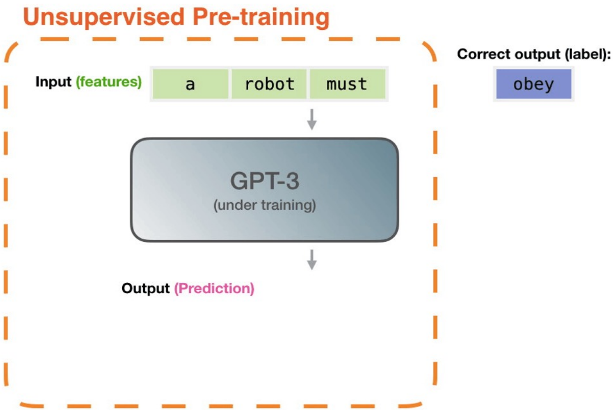
SJTU

https://arxiv.org/abs/2310.07177

Large Language Models (LLMs)



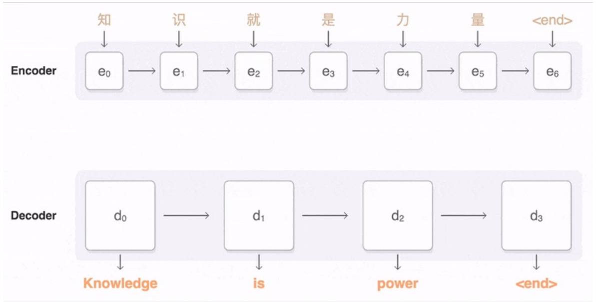
Transformer architecture



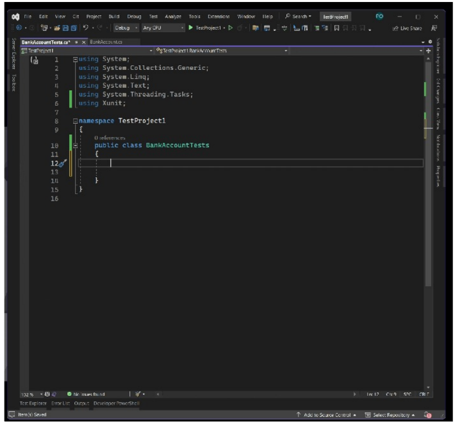
Self-supervised pre-training; 175B parameters; 300B tokens

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. **Attention Is All You Need**. NeurIPS 2017.
Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever. **Improving Language Understanding by Generative Pre-Training**. arXiv 2018.

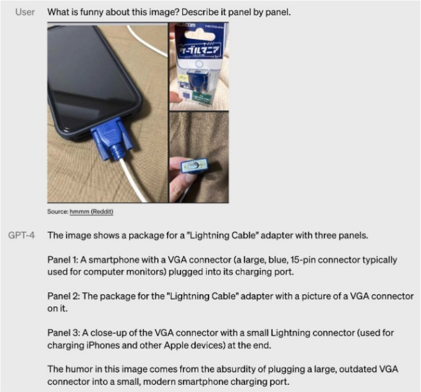
Typical applications of LLMs



Machine translation



Code generation


User What is funny about this image? Describe it panel by panel.

Source: hmmm (Reddit)

GPT-4 The image shows a package for a "Lightning Cable" adapter with three panels.

Panel 1: A smartphone with a VGA connector (a large, blue, 15-pin connector typically used for computer monitors) plugged into its charging port.

Panel 2: The package for the "Lightning Cable" adapter with a picture of a VGA connector on it.

Panel 3: A close-up of the VGA connector with a small Lightning connector (used for charging iPhones and other Apple devices) at the end.

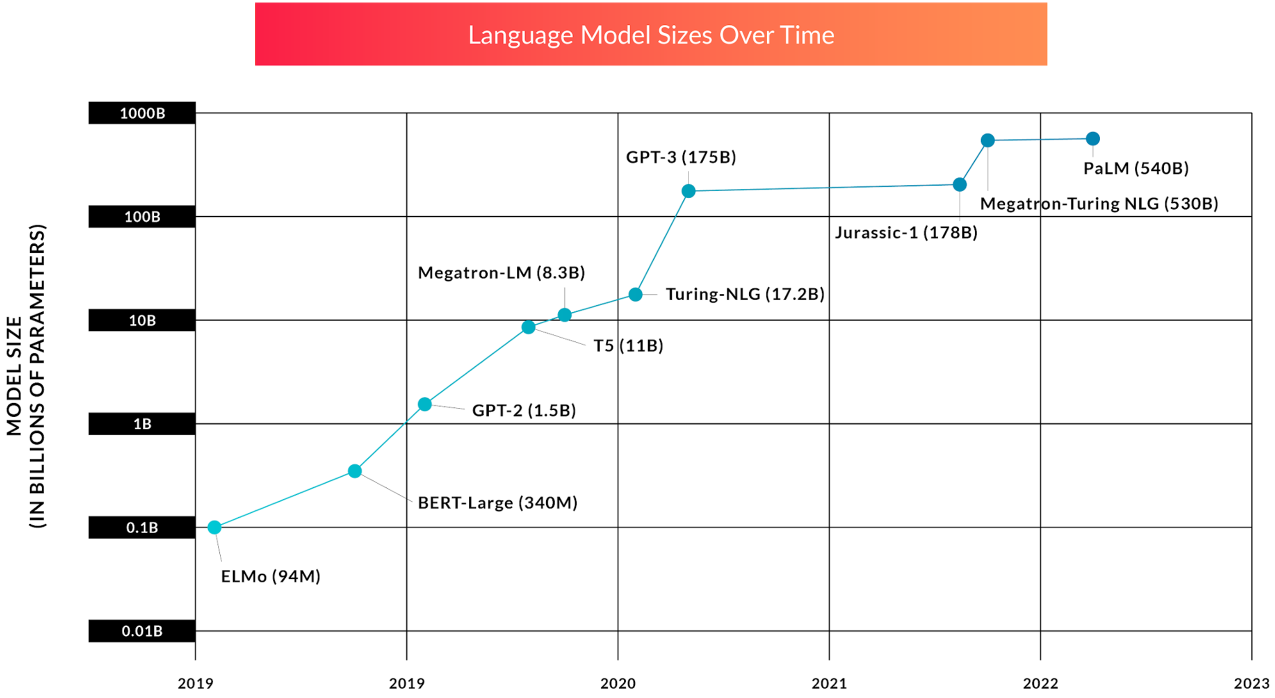
The humor in this image comes from the absurdity of plugging a large, outdated VGA connector into a small, modern smartphone charging port.

(Multimodal) Chatbot

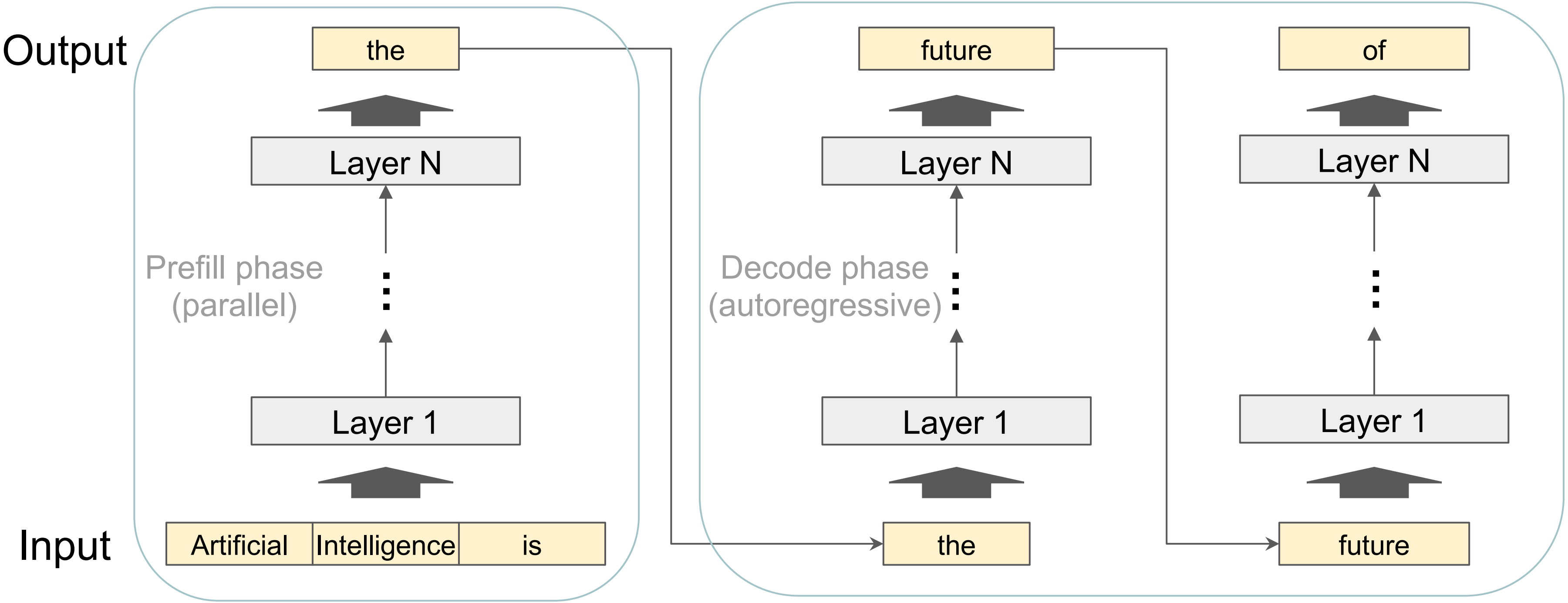
Agents

etc.

LLM inference latency affects the quality of service and user experience



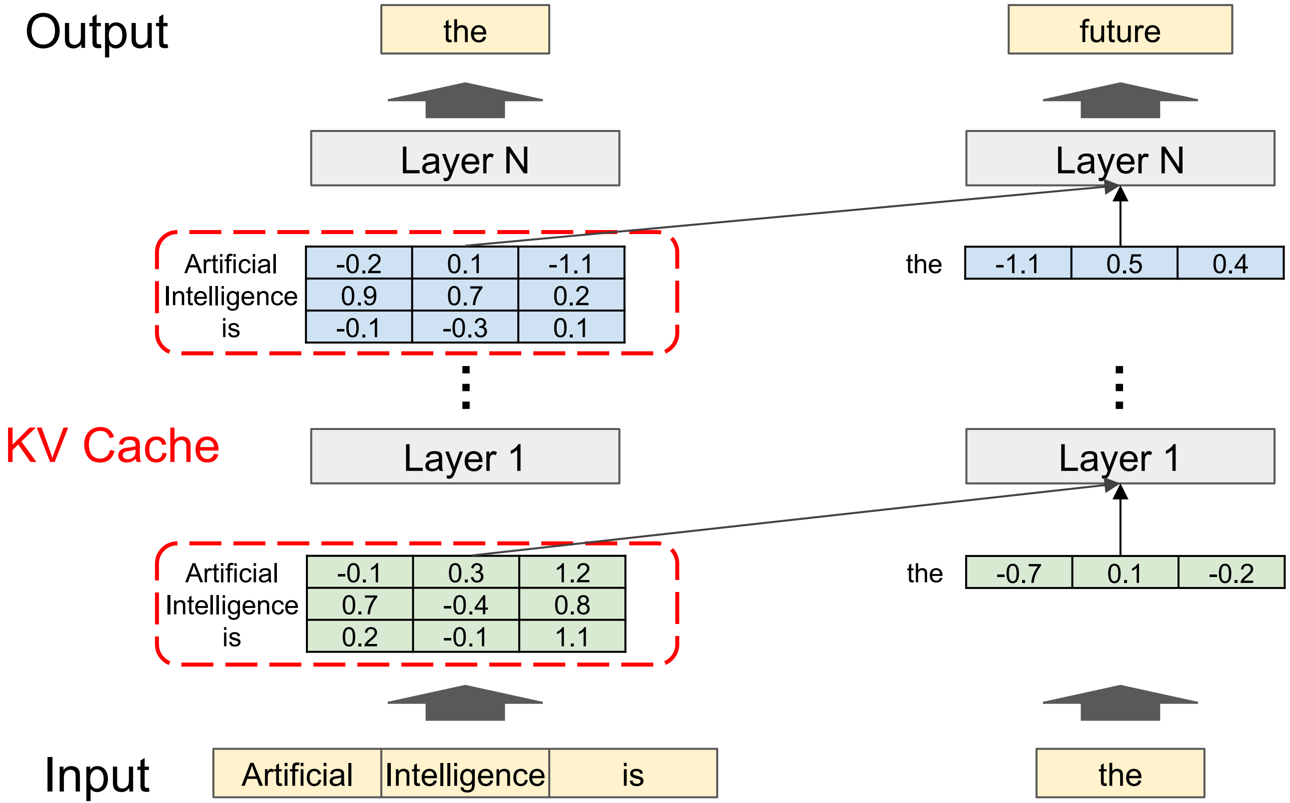
How do LLMs infer?



Repeat until the sequence

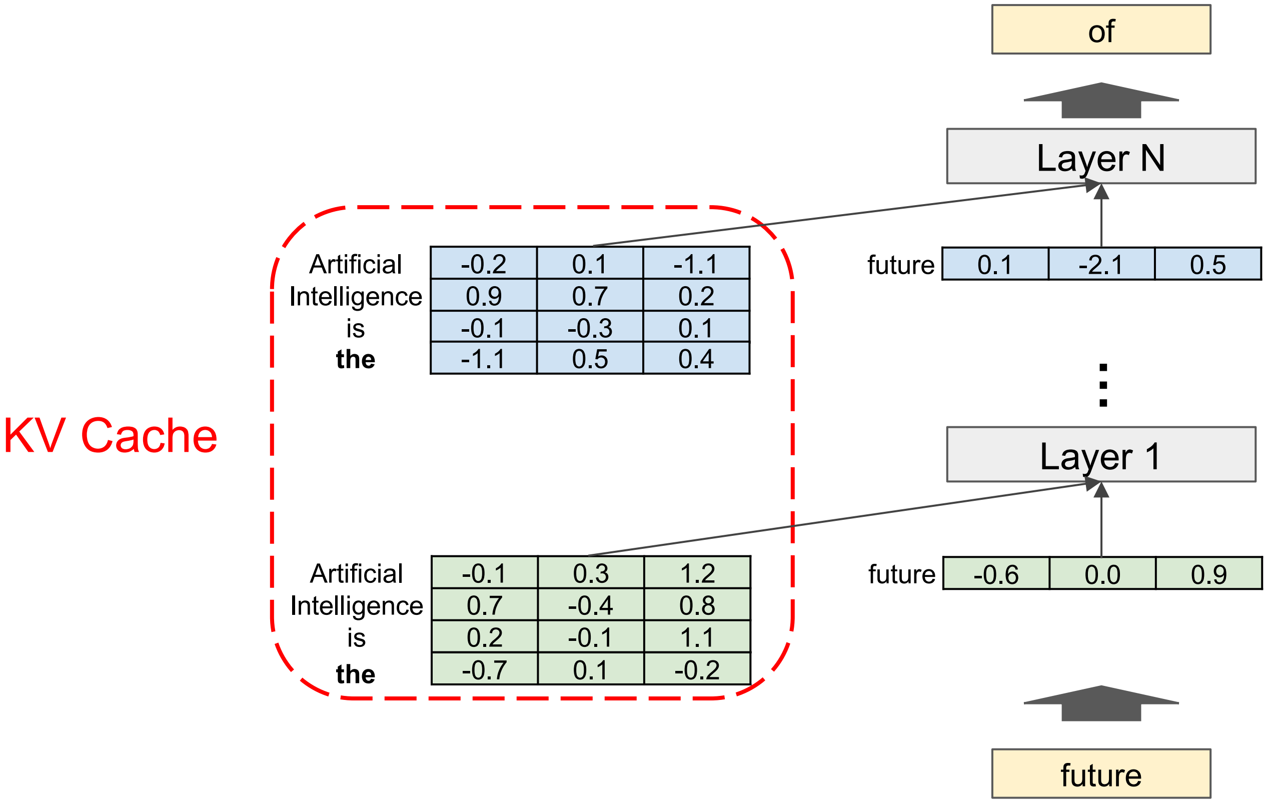
- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|*end of sequence*|>”)

KV Cache


Output
the
future
Layer N
Layer N
Artificial
Intelligence
is
-0.2 0.1 -1.1
0.9 0.7 0.2
-0.1 -0.3 0.1
the
-1.1 0.5 0.4
KV Cache
Layer 1
Layer 1
Artificial
Intelligence
is
-0.1 0.3 1.2
0.7 -0.4 0.8
0.2 -0.1 1.1
the
-0.7 0.1 -0.2
Input
Artificial
Intelligence
is
the

KV Cache

Output



Input

Two features

- Prefill phase takes about as much time as the generation of each subsequent token due to the use of GPU’s parallel compute
- LLM inference is **memory-IO bound** (between High Bandwidth Memory (HBM) and Static Random Access Memory (SRAM)), not compute bound
  - Abundant “spare FLOPs” exists in the serving of LLMs

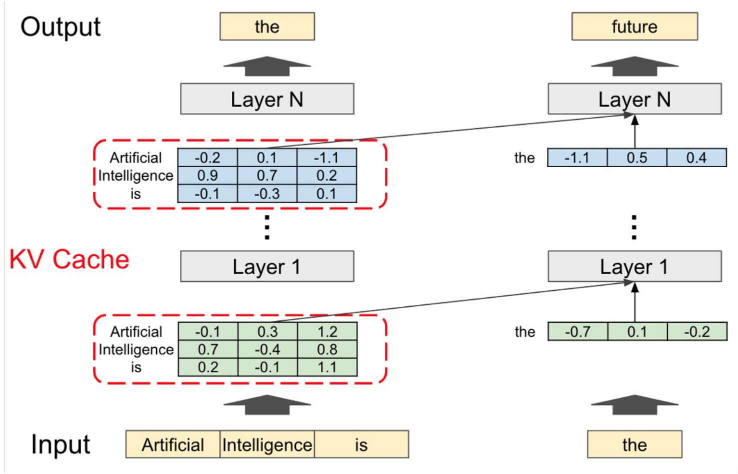
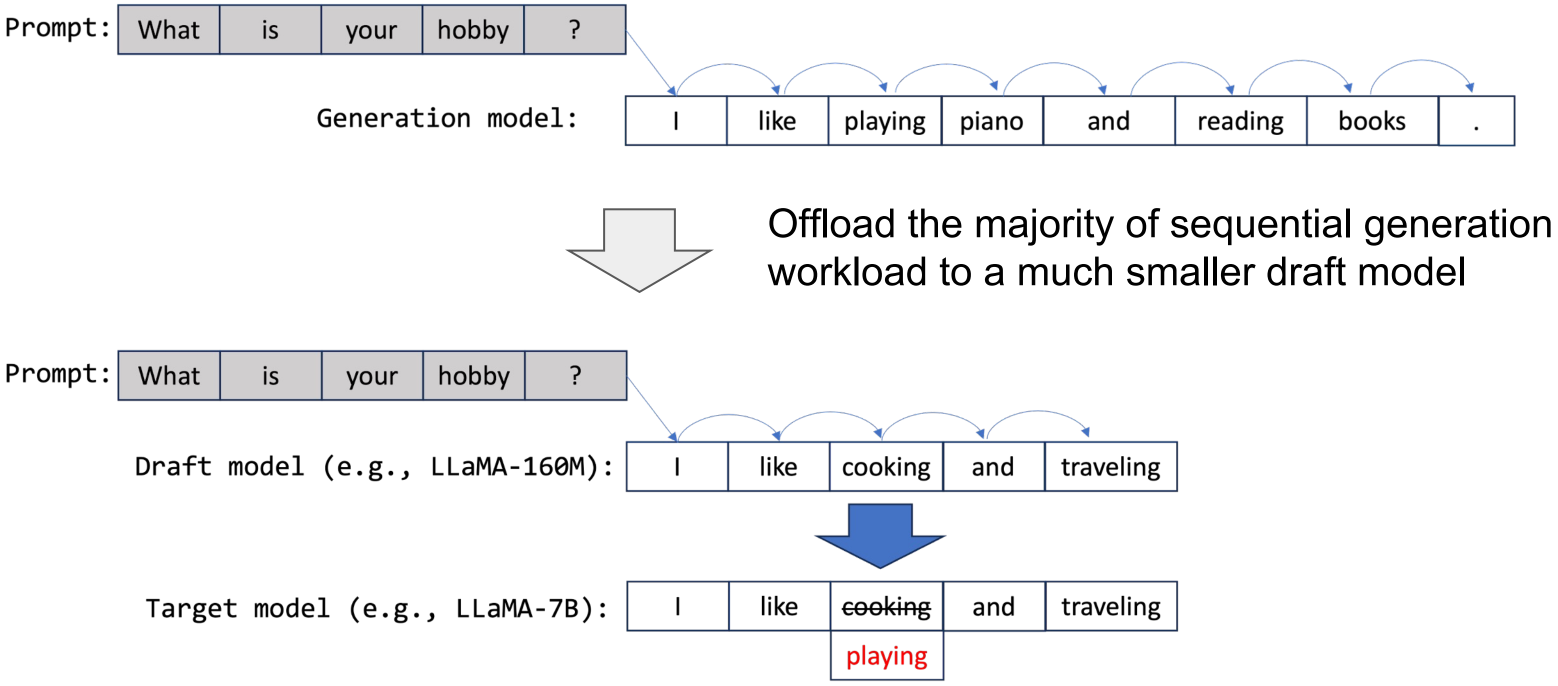


Table 1. Examples of neural network operations with their arithmetic intensities. Limiters assume FP16 data and an NVIDIA V100 GPU.

Operation	Arithmetic Intensity	Usually limited by...
Linear layer (4096 outputs, 1024 inputs, batch size 512)	315 FLOPS/B	arithmetic
Linear layer (4096 outputs, 1024 inputs, batch size 1)	1 FLOPS/B	memory
Max pooling with 3x3 window and unit stride	2.25 FLOPS/B	memory
ReLU activation	0.25 FLOPS/B	memory
Layer normalization	< 10 FLOPS/B	memory

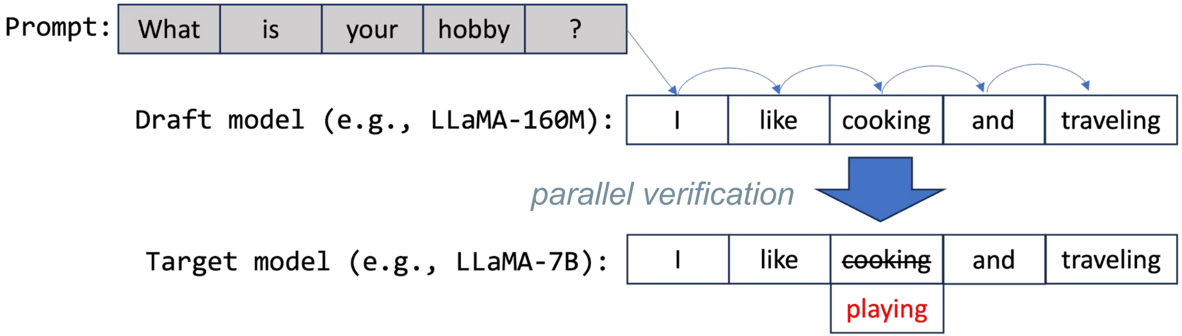
https://docs.nvidia.com/deeplearning/performance/dl-performance-gpu-background/index.html#understand-perf

Speculative decoding reduces LLM inference latency



Fast Inference from Transformers via Speculative Decoding. Yaniv Leviathan, Matan Kalman, Yossi Matias

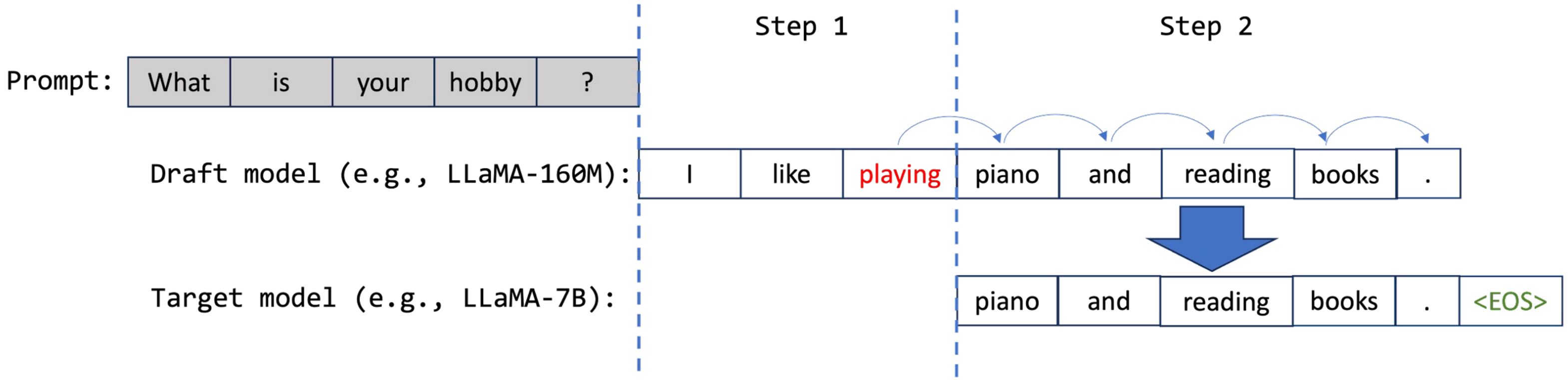
Speculative decoding



Key insights:

- Some tokens are straightforward to generate, while others are more challenging
- We can utilize a streamlined 'draft' model for the easier tokens
- To ensure identical output to the original generation method, the tokens proposed by draft model are then validated by the target model in parallel by the principle of rejection sampling

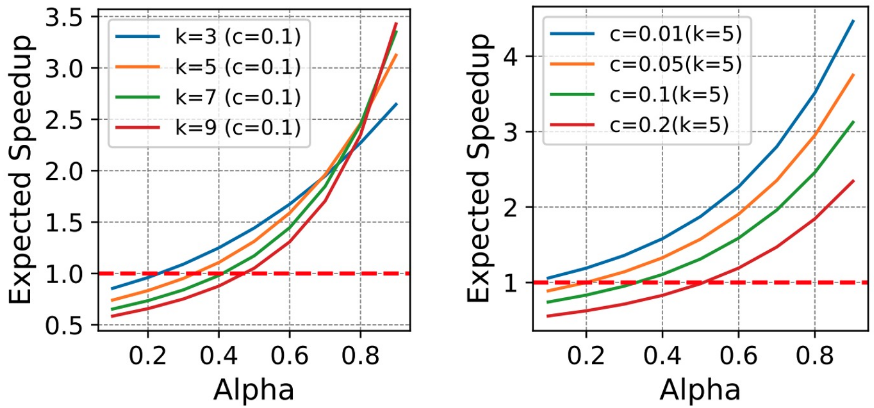
Speculative decoding (cont.)



Why can speculative decoding reduce latency?

- The draft model retains the autoregressive nature, generating tokens one at a time but with a significantly **faster** speed
- The target model can validate multiple generated tokens from the draft model in a **single** forward pass
- Consequently, speculative decoding helps **amortize** the overhead of loading model weights and key-value caches
  - *Originally, each token required accessing the weights and key-value cache, but now it’s reduced to just one access per x tokens, where x represents the number of accepted tokens in each generation step*

When will speculative decoding work?

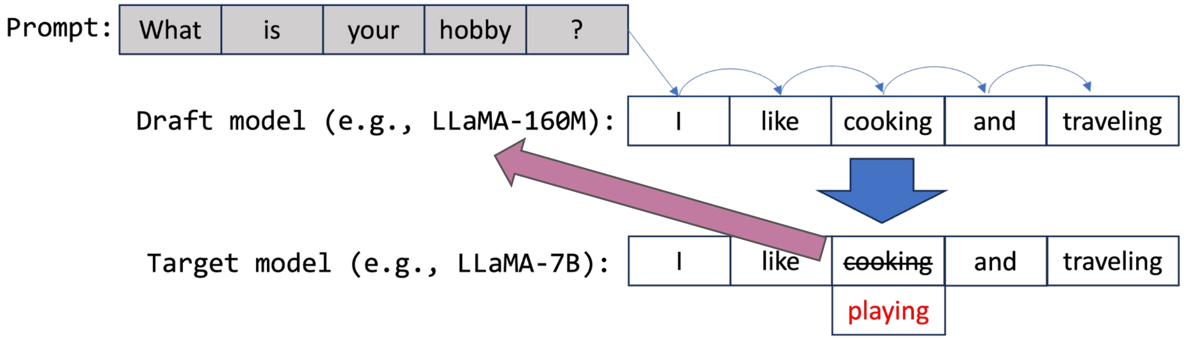


c: the time ratio for a single run between the draft and target model. k: number of proposed tokens each step. Alpha: token acceptance rate.

- **Better token acceptance rate leads to more speedup**: the draft model must approximate the target model sufficiently while being small to achieve latency reduction

Opportunity for improving token acceptance rate online

- Speculative decoding detects inaccuracies within the smaller draft model and provides corrections
  - Such information can be harnessed to refine the draft model, thereby enhancing the draft model's token acceptance rate, all *without incurring any additional labeling costs*



  - The *spare FLOPs*

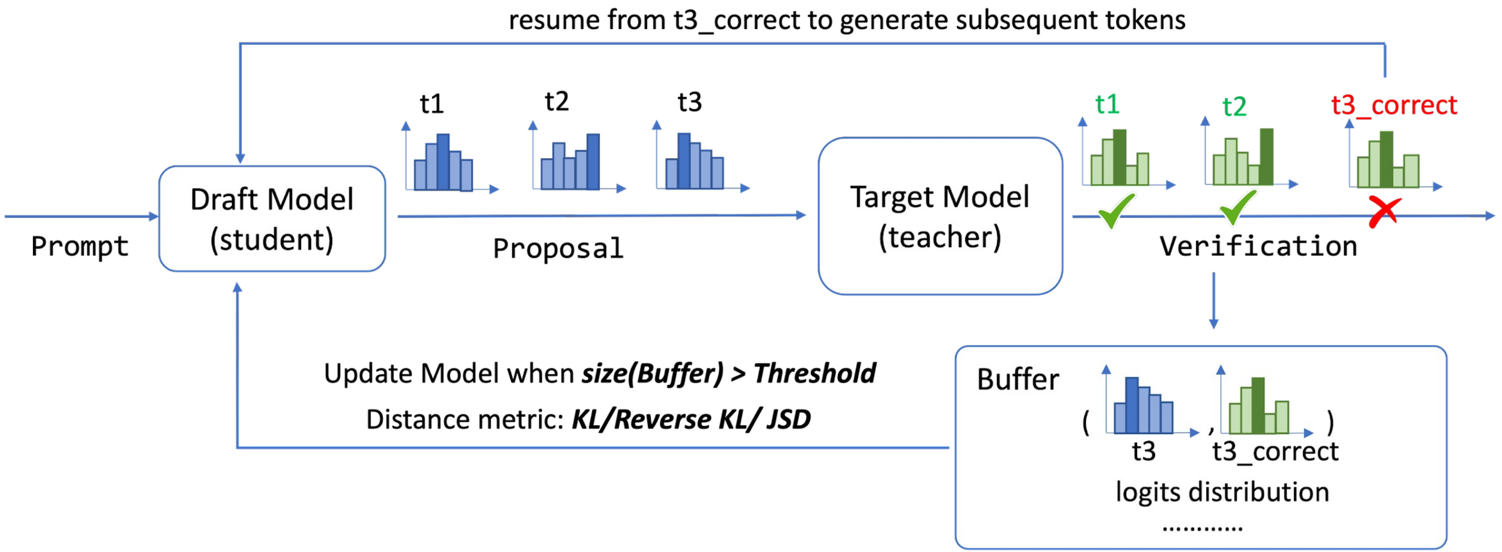
Necessity for improving token acceptance rate online

- Performance of speculative decoding algorithm depends heavily on one or a set of reliable draft models

x Open-domain draft models has poor speculation accuracy (due to size)

x It’s hard to predict query distributions and prepare specialized draft models offline to ensure speculation accuracy

Online speculative decoding (OSD): online distillation+SD



If the student proposes incorrect tokens, both the draft and target distributions are stored in a **buffer**. Once the buffer exceeds a specified threshold, the draft model is **updated** by calculating the loss between the draft and target distributions using various distance metrics.

Distillation loss

$$\ell(\boldsymbol{\theta}) = \frac{1}{n_B} \sum_{\boldsymbol{x}^{(i)} \in \mathcal{B}} \ell(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), \quad \ell(\boldsymbol{x}, \boldsymbol{\theta}) = D(p(\cdot|\boldsymbol{x}) \| q_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}))$$

$$\begin{aligned}
\ell_{KL}(\boldsymbol{x}, \boldsymbol{\theta}) &= D_{\mathrm{KL}}(p(\cdot|\boldsymbol{x}) \| q_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})), \\
\ell_{RKL}(\boldsymbol{x}, \boldsymbol{\theta}) &= D_{\mathrm{KL}}(q_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}) \| p(\cdot|\boldsymbol{x})), \\
\ell_{JSD[\beta]}(\boldsymbol{x}, \boldsymbol{\theta}) &= \beta D_{\mathrm{KL}} \left(p(\cdot|\boldsymbol{x}) \Big\| p^{\beta}_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}) \right) + (1-\beta) D_{\mathrm{KL}} \left(q_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}) \Big\| p^{\beta}_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}) \right)
\end{aligned}$$

* Estimating the above objectives involves the expectation over qθ(·|x) or p(·|x), which should be expanded **recursively**
* When sampling from qθ(·|x), we should **differentiate through the sampling process** for unbiased gradient estimation

The algorithm

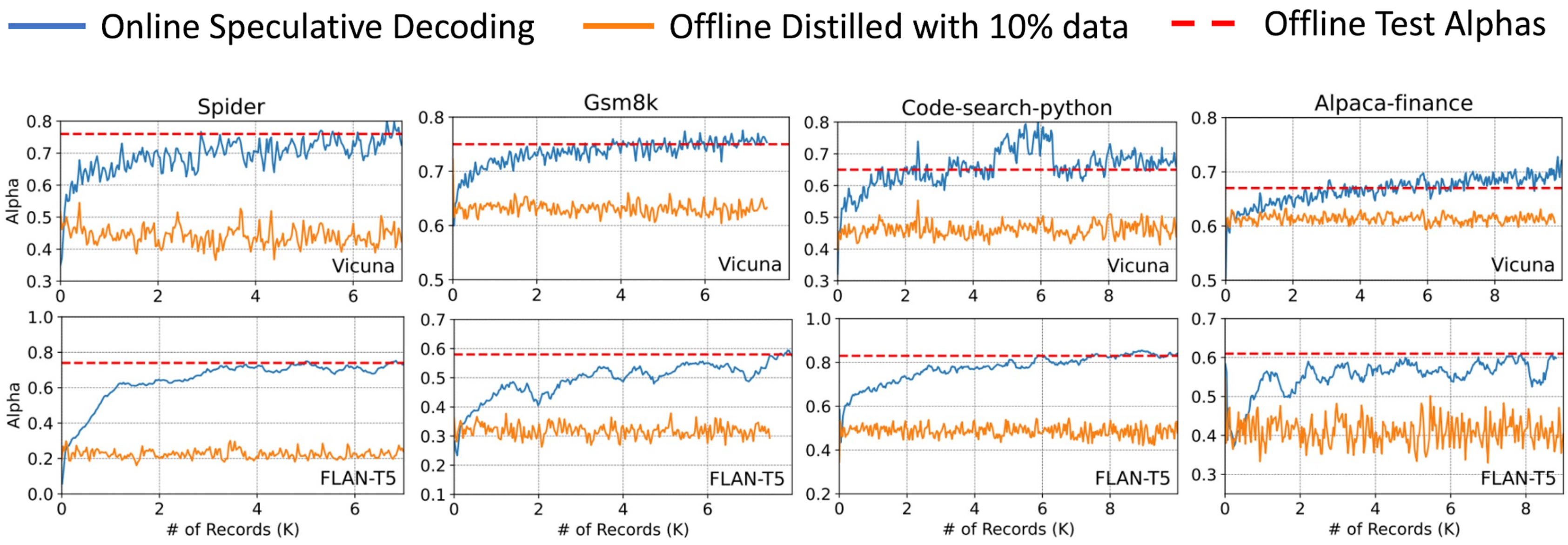
Algorithm 1 Online Speculative Decoding.

Input: Target LLM $p(\cdot|\boldsymbol{x})$, draft LLM $q_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})$, warmup dataset $\mathcal{D}$, online data stream $\mathcal{S}$, guess number k, temporary buffer $\mathcal{R}$, replay buffer $\mathcal{Q}$, update interval for the draft model I.
Pre-train $q_{\boldsymbol{\theta}}$ to approximate p with data from $\mathcal{D}$ by minimizing $\ell(\boldsymbol{x}, \boldsymbol{\theta})$ using Equation (5);
$i \leftarrow 0$;
$\mathcal{Q} \leftarrow []$;
$cur\_len = |\boldsymbol{x}|$ // Total sequence length, including prompt length and tokens generated so far.
while True **do**
  $\mathcal{R} \leftarrow []$ // List of ($error\_index$, target logits at $error\_index$) pairs for a single request.
  $\boldsymbol{x} \sim \mathcal{S}, i \leftarrow i + 1$;
  while $\langle$EOS$\rangle$ not in $\boldsymbol{x}$ **do**
    $\boldsymbol{y} = \{y_1, ..., y_k\} \sim q_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})$;
    Estimate $\{p(y|\boldsymbol{x}, \boldsymbol{y}_{<i})\}_{i=1}^{k+1}$ in parallel;
    Determine number of accepted tokens a and sample one more token, yielding $\boldsymbol{y} = \{y_1, \ldots, y_{a+1}\}$;
    $cur\_len \leftarrow cur\_len + a + 1$;
    $error\_index \leftarrow cur\_len$;
    Append $(error\_index, p(y|\boldsymbol{x}, \boldsymbol{y}_{<a+1}))$ to $\mathcal{R}$;
    $\boldsymbol{x} \leftarrow [\boldsymbol{x}, \boldsymbol{y}_{<a+2}]$;
  end while
  Append $(\boldsymbol{x}, \mathcal{R})$ to $\mathcal{Q}$;
  if $i \bmod I = 0$ **then**
    Update $q_{\boldsymbol{\theta}}$ on $\mathcal{Q}$ to minimize $\ell(\boldsymbol{x}, \boldsymbol{\theta})$ analytically;
    $\mathcal{Q} \leftarrow []$;
  end if
end while

Experimental setup

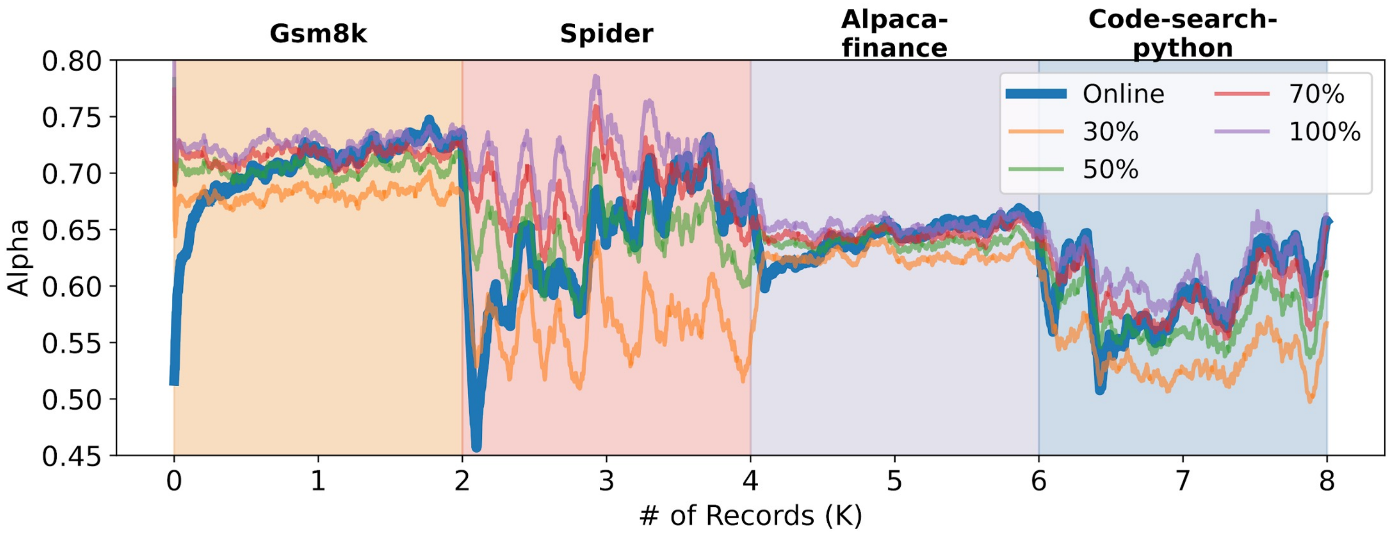
- Metric: token acceptance rate and wall-clock time
- Target model: Vicuna7B and FLAN-T5-XL (3B)
- Draft model: LLaMA-160m and T5-Small
- GPU: A100-80GB
- Datasets: Text-to-SQL (Spider), graduate school math (Gsm8k), Python code generation (Code-search-Python), and financial question answering (Alpaca-finance)
- Number of proposed tokens: 5

Does the online algorithm increase the token acceptance rate?



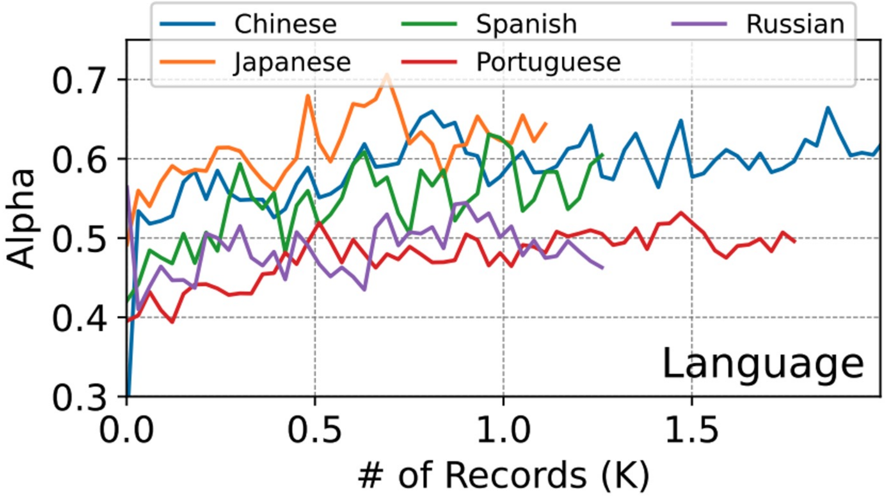
The acceptance rate rises swiftly as the draft model is exposed to more data

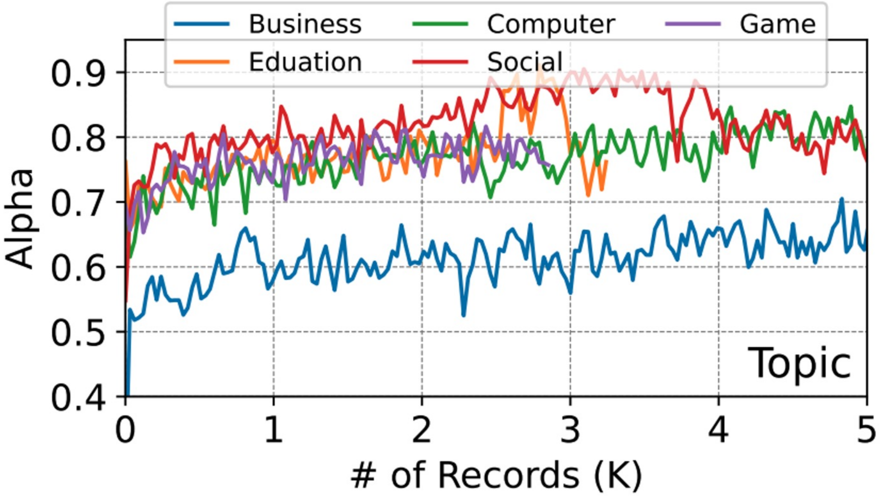
How quickly can OSD adapt to distribution shift



*OSD's alpha value dips notably at distribution boundaries but **rebounds** quickly as OSD processes more data, quickly matching or even surpassing performances seen with 70% to 100% data access*

OSD+routing on Arena (real LMSYS-chat conversations that span 4 months)





- One individual draft model for each language/topic
- When routing by language, OSD enhances rates by 0.1 to **0.2**
- When routing by topic, acceptance rates are above **0.6** across topics, with Social and Computer discussions peaking near **0.8**

Measured execution time/speedup and theoretical execution time/speedup

	Original	OSD, $\alpha = 0.5$	OSD, $\alpha = 0.6$	OSD, $\alpha = 0.7$	OSD, $\alpha = 0.8$	OSD, $\alpha = 0.9$
Measured time in ms/token (speedup)	51.09	39.90 ($1.28\times$)	35.48 ($1.44\times$)	30.96 ($1.65\times$)	25.42 ($2.01\times$)	19.43 ($2.63\times$)
Theoretical time in ms/token (speedup)	51.09	39.00 ($1.31\times$)	32.12 ($1.59\times$)	26.07 ($1.96\times$)	20.77 ($2.46\times$)	16.38 ($3.12\times$)

- Up to **2.63x speedup** over inference without speculative decoding

Measured speedup on four evaluated datasets on a single A100-80G

Dataset	Spider	Gsm8k	Alpaca-Finance	Code-Python
Measured time in ms/token (Speedup)	23.53 (2.17 $\times$)	27.40 (1.89 $\times$)	26.53 (1.92 $\times$)	30.12 (1.69 $\times$)

- TinyLLaMA-1.1B, Vicuna-33B
- Inference without speculative decoding has a token latency of 51.09 ms/token
- Up to **2.17x speedup**

Thanks