MEDUSA: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads

Zhongsheng Wang

School of Computer Science
The University of Auckland

zhongsheng.wang@auckland.ac.nz

June 4, 2025

Waipapa Taumata Rau
University of Auckland

Why this topic?

- Improving LLMs design from an engineering perspective, a “solid” paper
- LLMs inference acceleration cannot rely solely on the acceleration framework (vLLM, Deepseed), but should focus on the efficient use of data
- I am interested and wanna explore some potential ideas :) (Some entry difficulty)

Preliminary

Transformer-based LLMs Architecture, when generating the t_{th} token:

$$P(x_t \mid x_{<t}) = softmax(W_o \cdot h_t)$$

where:

- $h_t = Transformer(x_1, x_2, x_3 \ldots, x_{t-1})$: Contextual representation of the previous token (obtained through self-attention and position encoding)
- W_o: Output projection matrix (maps hidden state to vocabulary space)
- softmax: Output probability of each word

Problem

LLMs are slow, and the bigger they are the slower they get

The bottleneck is not in the math or the GPU not being able to multiply faster but in the data transfer

For each forward pass:

- Model file (.bin/.pth) & Input Token : Disk $\rightarrow$ CPU RAM
- Weight Parameters & Token Representation: CPU RAM $\rightarrow$ GPU RAM
- Forward Calculation & Output Logits: GPU RAM

So a lot of time is spent doing data transfers (underutilizing the GPU's computational potential)

What can we do?

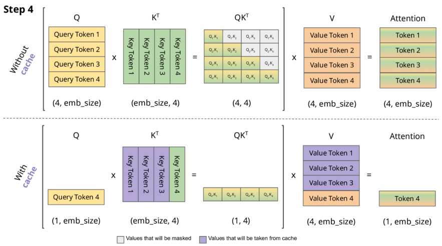
Batch inference

- Process multiple inputs at once (batch), parameters only need to be loaded once, but calculations can be performed in parallel
- KV-Cache limitation (When generating Attention for token x_t):

$$\text{Attention}(Q_t, K_{1:t}, V_{1:t}) = \text{softmax}\left(\frac{Q_t K_{1:t}^T}{\sqrt{d_k}}\right) V_{1:t}$$

- Q_t: Token from the current generated location x_t; $K_{1:t}, V_{1:t}$: Vectors from all past tokens $x_{<t}$

What can we do?



Comparison of scaled dot-product attention with and without KV caching. emb_size means embedding size. Image created by the author.

KV cache size per layer: $Size_{perlayer} = 2 \times B \times H \times T \times d_k \times$ 2 btyes (FP16)

Example: For a transformer-based model with Batch size $= 1$, Attention head $= 32$, Max token sequence length $= 2048$, $d_k = 128$, bytes-per-element $= 2$

$Size_{perlayer} = 2 \times 1 \times 32 \times 2048 \times 128 \times 2$ btyes $= 33554432$ bytes ≈ 34 MB

GPT-3 has 32 layers, so KV Cache $= 32 \times 32 = 1$ GB,

What can we do?

Quantization

Precision	**Value Range**	**Size**	**Relative Error**	**Usage**
FP32	32-bit floating point	4 bytes	Very low	Training, high-precision inference
FP16	16-bit floating point	2 bytes	Low	Faster inference with good precision
INT8	$[-128, 127]$	1 byte	Moderate	Common inference quantization
INT4	$[-8, 7]$	0.5 byte	High	Ultra-low precision (e.g., GPTQ)
INT1	$\{0, 1\}$	0.125 byte	Very high	Experimental on specialized hardware

Table: Comparison of numerical precisions in quantized neural networks

Speculative decoding

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

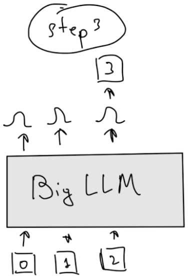
Intuition:

- Auto-regressive generation means we can only generate them one by one $\cdots$ so can we?
- What if we can generate multiple future tokens in parallel (a single forward pass)?

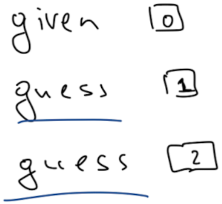
Guess & Verification:

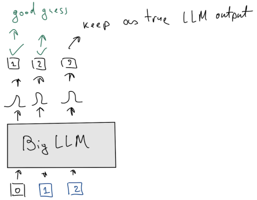
- Guess (speculate) n tokens in the future and then verify
- If the guess is bad, at least we can get 1 correct token
- Make LLM “guess” as well as possible

Speculative decoding



normal way to generate tokens

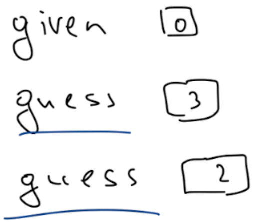


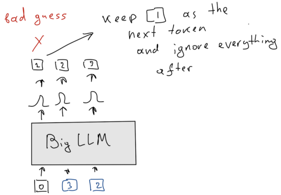


sort of "good guess"

- 3 Forward passes → 3 Tokens (left)
- In one pass we verified the 2 guesses (green) and got the last token
- 2 guesses + 1 forward pass → 3 Tokens

Bad speculative





- 2 guesses + 1 forward pass → 1 Token (like we would get normally)
- A bit worse because we have some extra overhead
- the better the guessing strategy the faster we can go

Different Kinds of Speculating:

- Random guessing / more informed guessing
- Speculative Decoding With Smaller (faster) draft model verified by target (big) model
- Medusa (this paper), like a guesser built into the model itself

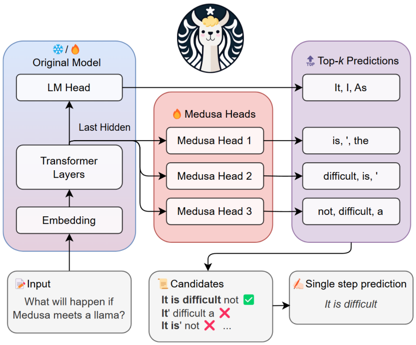
Why abandon the draft model?

- Need to manage 2 models, which can be awkward in inference settings
- How do you choose the draft (helper) model
- Fine-tuning is a pain because still have to fine-tune a possibly not-so-small draft LLM

What Medusa did?

- Predict multiple tokens (k or more) at a time and keep accepting this batch of predictions for as long as possible

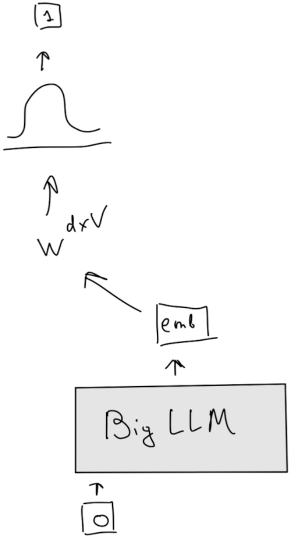
Madusa framework

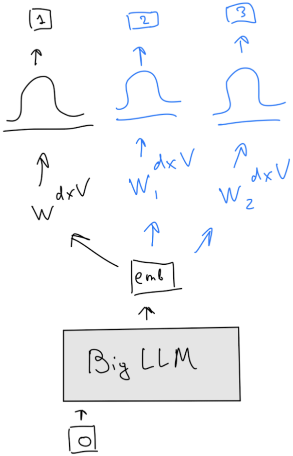


Medusa Framework

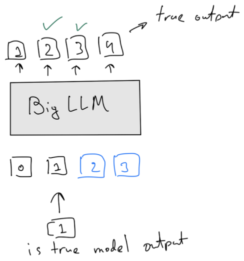
- Expand three Medusa heads behind the last hidden layer (Transformer-based model)
- Each head generates top-k candidates
- Permutations and combinations of candidates for each head
- Select the best candidate through some acceptance mechanism

Generating tokens with Medusa heads





Verifying tokens with Medusa



Example:

- Generated a sequence [0, 1, {2}, {3}] with 2 Medusa Heads based on the input {0}
- Feed [0, 1, 2, 3] back to your model and model generates something like [1, 2, 3, 4]
- LLM confirmed candidates 2 and 3 and gave us 4
- 2 forward passes $\rightarrow$ 4 tokens

What if we mess up:

- Feed [0, 1, 2, 3] and get [1, 2, 6, 7]
- 2 is a good guess but 3 was wrong
- Finally we got [1, 2, 6]
- 2 forward passes $\rightarrow$ 3 tokens, and start with 6 next time

Rejection sampling

Algorithm 2 Speculative Sampling (SpS) with Auto-Regressive Target and Draft Models

```
Given lookahead $K$ and minimum target sequence length $T$.
Given auto-regressive target model $q(.|.)$, and auto-regressive draft model $p(.|.)$, initial prompt sequence $x_0, \dots, x_t$.
Initialise $n \leftarrow t$.
while $n < T$ do
  for $t = 1 : K$ do
    Sample draft auto-regressively $\tilde{x}_t \sim p(x|, x_1, \dots, x_n, \tilde{x}_1, \dots, \tilde{x}_{t-1})$
  end for
  In parallel, compute $K+1$ sets of logits from drafts $\tilde{x}_1, \dots, \tilde{x}_K$ :
      $$q(x|, x_1, \dots, x_n),\ q(x|, x_1, \dots, x_n, \tilde{x}_1),\ \dots,\ q(x|, x_1, \dots, x_n, \tilde{x}_1, \dots, \tilde{x}_K)$$
  for $t = 1 : K$ do
    Sample $r \sim U[0,1]$ from a uniform distribution.
    if $r < \min\left(1, \frac{q(x|x_1,\dots,x_{n+t-1})}{p(x|x_1,\dots,x_{n+t-1})}\right)$, then
      Set $x_{n+t} \leftarrow \tilde{x}_t$ and $n \leftarrow n+1$.
    else
      sample $x_{n+t} \sim (q(x|x_1, \dots, x_{n+t-1}) - p(x|x_1, \dots, x_{n+t-1}))_+$ and exit for loop.
    end if
  end for
  If all tokens $x_{n+1}, \dots, x_{n+K}$ are accepted, sample extra token $x_{n+K+1} \sim q(x|, x_1, \dots, x_n, x_{n+K})$ and
  set $n \leftarrow n+1$.
end while
```

Typical Acceptance

Given the context, $x_1, x_2, x_3, \cdots, x_n$ we have a candidate sequence:

$$(x_{n+1}, x_{n+2}, x_{n+3}, \cdots, x_{n+K})$$

For each token x_{n+K}, check whether it satisfies:

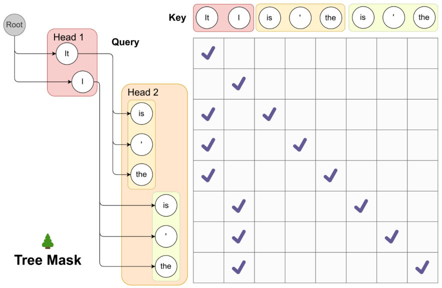
$$P_{\text{orig}}(x_{n+k} \mid x_{1:n+k-1}) > \min\left(\epsilon, \delta \cdot e^{-H(p)}\right)$$

- $P_{\text{orig}}(x_{n+k} \mid x_{1:n+k-1})$: The probability assigned by the original language model to the candidate token x_{n+k}, given the context tokens $x_1, x_2, \ldots, x_{n+k-1}$.
- ϵ: A hard threshold.
- δ: A scaling coefficient that adjusts the entropy-dependent soft threshold.
- $H(p)$: The entropy of the predicted probability distribution p at position $n+k$, defined as:

$$H(p) = -\sum_i p_i \log p_i$$

- $\delta \cdot e^{-H(p)}$: The soft threshold.

Tree attention

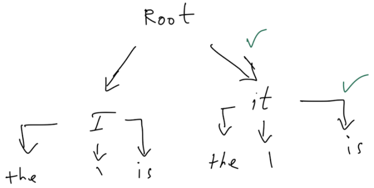


- Calculate attention for all candidate paths in one go
- Let one token only pay attention to its path ancestors and does not attend unrelated paths
- 2 important components:
  - Attention Mask
  - Position Embedding

Tree attention

- The top-k results of multiple head predictions can be constructed into a tree network
- Instead of validating a single candidate, a complete branch in the tree structure is validated
- Sequence length: not $O(n^2)$ as a traditional attention mechanism, just depending on the number of heads and k
- **BUT**: Do not improve the calculation complexity, just overlay path visibility controls on top of it

Tree attention demo



More likely to be a model trimming process (especially for attention calculation)

Model predictions:

- Root → It ⇒ discard left sub-tree (since tokens don't match)
- Root → It → is ⇒ Discard paths [Root, it, the] and [Root, it, I] since they do not match
- Path [Root, it, is] is verified by the model

Integrate Medusa to LLM

Medusa 1:

- Freeze the backbone LLM and only train the Medusa heads
- Train with cross-entropy loss (suitable for scenarios with insufficient resources)

Medusa 2:

- Train Medusa heads with the rest of the LLM
- Can lead to better speed-ups (but it may affect the performance of the native model and requires special processing)

Medusa 1

$$\mathcal{L}_{\text{Medusa-1}} = \sum_{k=1}^{K} -\lambda_k \cdot log p_t^{(k)}(y_{t+k+1})$$

- K: Number of Heads
- λ_k: k-th power of a constant like 0.8
- The training only takes a few hours (e.g., 5 hours for MEDUSA-1 on Vicuna 7B model with a single NVIDIA A100 PCIE GPU to train on 60k ShareGPT samples).

Why is it not enough?

- Relying on the contextual representation given by the main model, a lack of collaborative training
- Token predicted by Medusa heads is far different from the real distribution of the main model

Medusa 2

Backbone model loss:

$$\mathcal{L}_{LM} = -\lambda_k \cdot logp_t^{(0)}(y_{t+1})$$

Combine loss:

$$\mathcal{L}_{\text{Medusa-2}} = \mathcal{L}_{\text{LM}} + \lambda_0 \mathcal{L}_{\text{Medusa-1}}$$

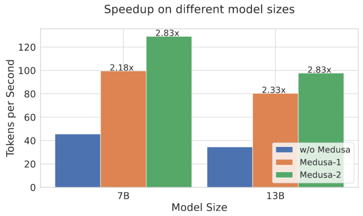
- Learning rate of the backbone model is small, and the Medusa heads are large
  - Avoid shifting the LLMs learned distribution with small learning rate
- Warm-up strategy: first use Medusa 1 to train Medusa heads, then use Medusa 2 for overall fine-tuning

Self-distillation

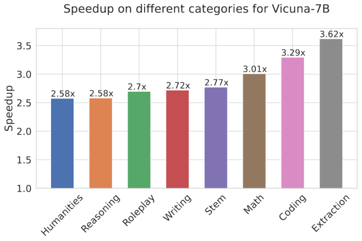
$$\mathcal{L}_{\text{SD}} = \underbrace{\text{KL}(p_{\text{orig}} \,||\, p_t^{(0)})}_{\text{Main LM distillation}} + \lambda_0 \cdot \sum_{k=1}^{K} \lambda_k \cdot \underbrace{\text{CE}(p_t^{(k)}, y_{t+k+1})}_{\text{Medusa Heads cross-entropy}}$$

- Experiments show that Medusa 2 will affect the native capabilities of the model, even if only the backbone model is trained using data
- Self-distillation maintains the probability distribution of the backbone model before fine-tuning (ShareGPT)

Some results



(a)



(b)

Model Name	Vicuna-7B	Zephyr-7B	Vicuna-13B	Vicuna-33B
Acc. rate	3.47	3.14	3.51	3.01
Overhead	1.22	1.18	1.23	1.27
Quality	6.18 (+0.01)	7.25 (-0.07)	6.43 (-0.14)	7.18 (+0.05)
$S_{\text{SpecDecoding}}$	1.47	-	1.56	1.60
S_{MEDUSA}	2.83	2.66	2.83	2.35

Table 1. Comparison of various MEDUSA-2 models. The first section reports the details of MEDUSA-2, including accelerate rate, overhead, and quality that denoted the average scores on the MT-Bench compared to the original models. The second section lists the speedup (S) of SpecDecoding and MEDUSA, respectively.

Summary

- No separate draft model needed
- Multiple lightweight decoding heads are attached to the backbone LLM to achieve multi-token parallel prediction
- Tree attention mask, multiple candidate paths can be verified in parallel in a single forward pass while maintaining the auto-regressive dependency structure

Potential idea

- Improve the acceptance rate (confidence) of tokens generated by draft models
- Calculate the expectation of the max k tokens that verification models can accept once (or the probability that the verification model will reject the k_{th} token)
  - Serial output model (single head) & Parallel output model (multi-head) may have different probabilities
  - Once we get the expectation above according to the specific architecture of the LLMs, we can set a proper number of heads to best speed up the inference part

References

- Paper: Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads
- Paper: Accelerating Large Language Model Decoding with Speculative Sampling
- Note: https://www.notion.so/Live-Medusa-Simple-LLM-Inference-Acceleration-Framework-with-Multiple-Decoding-Heads-fdb415a5ec524efbb346890793baac62#c0085050da29457b85281ffe74c7e652
- Video: https://www.youtube.com/watch?v=Jjjn-J9SJ1s&t=2885s&ab_channel=Oxen
- Video: https://www.youtube.com/watch?v=JmYFunlTeVI