Efficient Inference

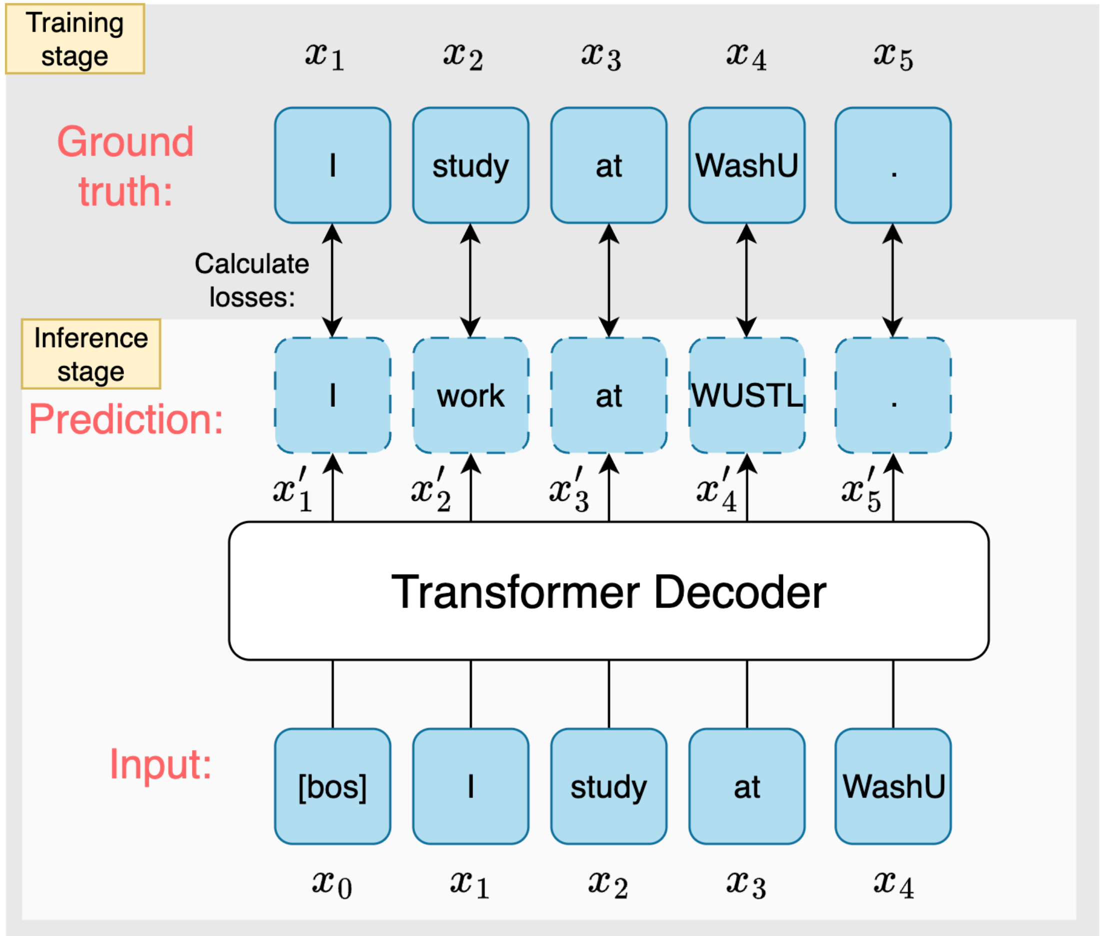
10/3 Langlin Huang, Claire Shi

Background

Why do we need efficient inference?

The Speed of Transformer-based LLMs

Quick review of the data-flow of an LLM:



Inference: $P(x'_t|x'_{<t})$
Training: $L_{CE}(x_t \,||\, P(x'_t|x_{<t}))$

Training stage:

- Parallel
- Fast

Inference stage:

- Auto-regressive
- **Slow**

If x'_{t-1} is unknown, we can't jump to the generation of x'_t.

The Speed of Transformer-based LLMs

Quick review of the complexity of an LLM:

Attention layer:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Time complexity: $O(L^2D) + O(L) + O(L^2D) = \boldsymbol{O(L^2D)}$

Assumptions:

Text length = L

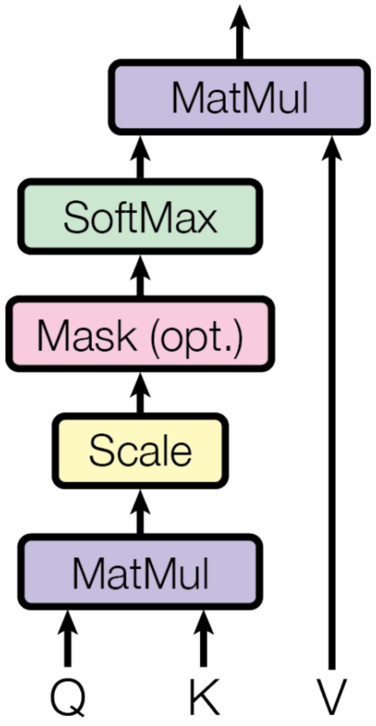
Hidden state dimension = D

$Q, K, V, x \in \mathbb{R}^{L \times D}$

$W_1 \in \mathbb{R}^{D \times D'}$

$W_2 \in \mathbb{R}^{D' \times D}$

Batch size=1



Feed-Forward Network(FFN) layer:

$$\text{FFN}(x) = \sigma_2(W_2(\sigma_1(W_1(x))))$$

Time complexity: $O(LDD') + O(D') + O(LDD') + O(D) = O(LDD') \sim \boldsymbol{O(LD^2)}$

Where D' is usually several times of D.

The Speed of Transformer-based LLMs

Cont.

Overall complexity:

Train: $N \times (O(L^2 D + LD^2))$, where N denotes the number of layers

Inference: $\sum_{l=1}^{L} N \times (O(l^2 D + lD^2)) = N \times O(L^3 D + L^2 D^2)$

Long context significantly slows the inference time!

Model	D	Max(L)
Llama3.2-1B	2048	131072
Llama3.2-3B	3072	131072
Llama3.1-8B	4096	131072
Mistral3-7B	4096	32768

Text	L
Math question	<100
News article	~500
8-page paper	~4000

Typical methods to perform efficient inference

Method	How to accelerate
Speculative Decoding*	Draft with small model (reduce N and D) and verify with LLM in parallel
Prompt Compression*	Shorten the prompt context length (reduce L)
Knowledge Distillation	Use a smaller model (reduce N and D)
Sparse Attention	Use a context-limited attention (reduce L to k, a much smaller number)
……	

WashU McKelvey Engineering

Fast Inference from Transformers via Speculative Decoding

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

Google research

Published at ICML2023

https://openreview.net/pdf?id=C9NEblP8vS

GitHub： https://github.com/feifeibear/LLMSpeculativeSampling

General idea of speculative decoding

Generate with a fast but less accurate model (denoted by $\boldsymbol{M_q}$);

Verify with a slow but more accurate model (denoted by $\boldsymbol{M_p}$).

M_q: Auto-regressive generation, with time complexity $O(N_q(L^3 D_q + L^2 D_q^2))$

M_p: Parallel / Non-auto-regressive verification, with time complexity $O(N_p(L^2 D_p + L D_p^2))$

Where $N_q < N_p$ and $D_q < D_p$

What it achieves:

1. Faster in speed;
2. Exactly the same performance.

General idea of speculative decoding

A case showing the process of speculative decoding

[START] japan ' s benchmark ~~bond~~ n

[START] japan ' s benchmark nikkei 22 ~~,~~5

[START] japan ' s benchmark nikkei 225 index rose 22 ~~.~~6

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 ~~,~~ points

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or ~~0~~ 1

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 98~~5~~9

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 ~~,~~ in

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in ~~tokyo~~ late

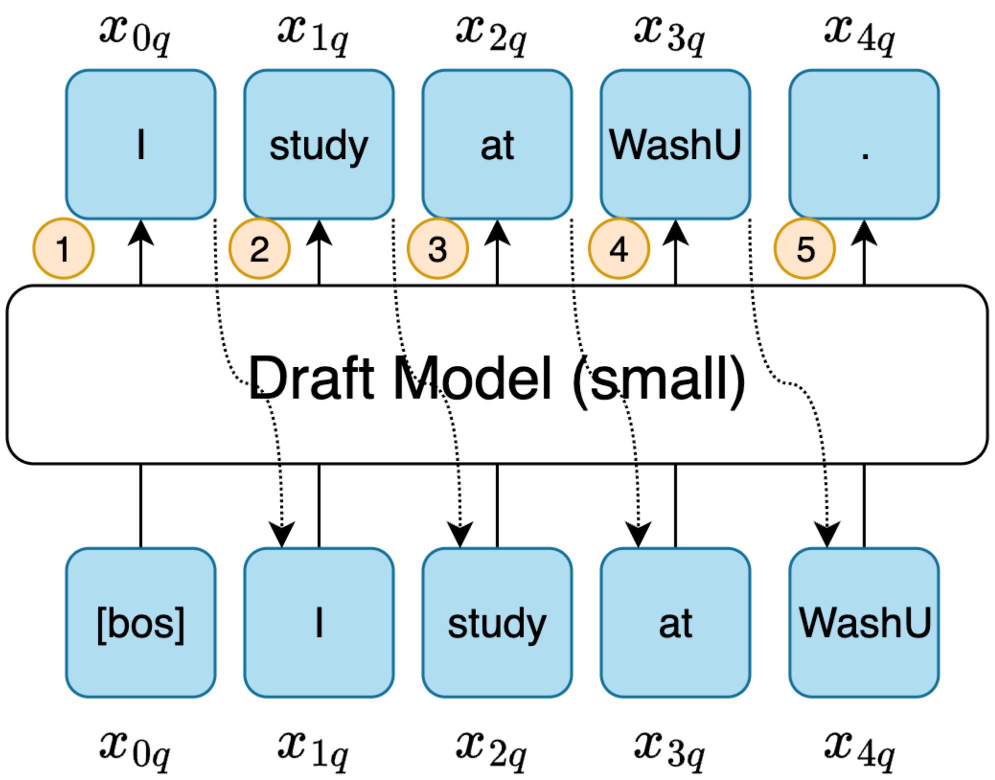
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in late morning trading . [END]

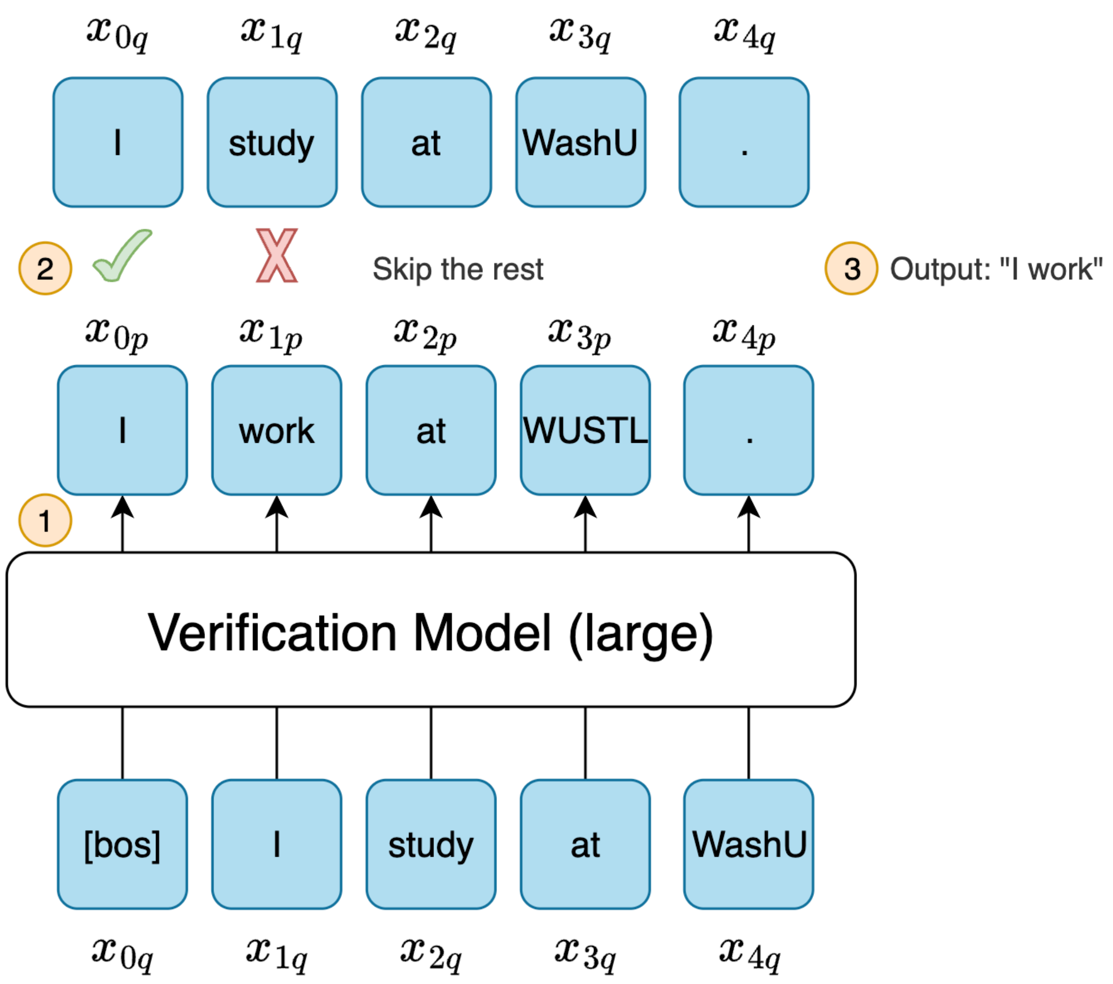
Figure 1. Our technique illustrated in the case of unconditional language modeling. Each line represents one iteration of the algorithm. The **green** tokens are the suggestions made by the approximation model (here, a GPT-like Transformer decoder with 6M parameters trained on lm1b with 8k tokens) that the target model (here, a GPT-like Transformer decoder with 97M parameters in the same setting) accepted, while the **red** and **blue** tokens are the rejected suggestions and their corrections, respectively. For example, in the first line the target model was run only once, and 5 tokens were generated.

Method of speculative decoding

1. Greedy decoding

$$x_t = argmax_{w \in V} P(x_w | x_{<t})$$





Method of speculative decoding

2. Sampling

x_t is sampled from $P_{w \in V}(x_w | x_{<t})$

$$P(w_i) = \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})}$$

Standard: $P_{w \in V}(x_w | x_{<t}) = \text{softmax}(\boldsymbol{z} | w_{<t})$, $\boldsymbol{z}$: logit of $\boldsymbol{x}$

Sampling with temperature T, $P_{w \in V}(x_w | x_{<t}) = \text{softmax}(\frac{\boldsymbol{z}}{T} | w_{<t})$

For example:

$T = 1$

$$P(w_1) = \frac{e^5}{e^5 + e^2 + e^{-1}} \approx 0.947$$

$$P(w_2) = \frac{e^2}{e^5 + e^2 + e^{-1}} \approx 0.047$$

$$P(w_3) = \frac{e^{-1}}{e^5 + e^2 + e^{-1}} \approx 0.006$$

$T = 0.5$

$$P(w_1) = \frac{e^{5/0.5}}{e^{5/0.5} + e^{2/0.5} + e^{-1/0.5}} \approx 0.999$$

$$P(w_2) = \frac{e^{2/0.5}}{e^{5/0.5} + e^{2/0.5} + e^{-1/0.5}} \approx 0.001$$

$$P(w_3) = \frac{e^{-1/0.5}}{e^{5/0.5} + e^{2/0.5} + e^{-1/0.5}} \approx 10^{-7}$$

$T = 2$

$$P(w_1) = \frac{e^{5/2}}{e^{5/2} + e^{2/2} + e^{-1/2}} \approx 0.718$$

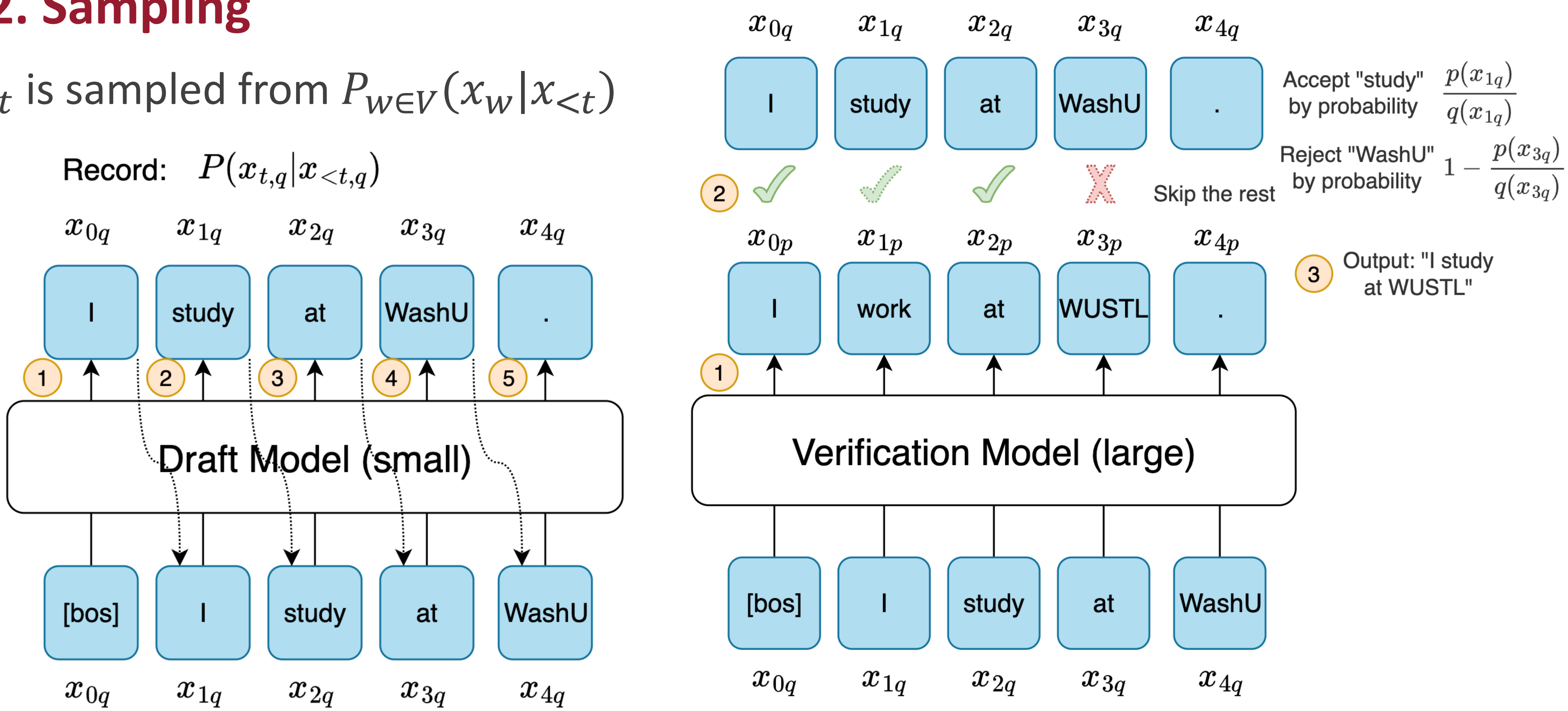
$$P(w_2) = \frac{e^{2/2}}{e^{5/2} + e^{2/2} + e^{-1/2}} \approx 0.237$$

$$P(w_3) = \frac{e^{-1/2}}{e^{5/2} + e^{2/2} + e^{-1/2}} \approx 0.045$$

Method of speculative decoding

2. Sampling

x_t is sampled from $P_{w \in V}(x_w | x_{<t})$



Method of speculative decoding

2. Sampling

x_t is sampled from $P_{w \in V}(x_w | x_{<t})$

M_p: 97M
M_q: 6M
M_q predicts the next γ tokens

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.
▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.
for $i = 1$ **to** γ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$
  $x_i \sim q_i(x)$
end for
▷ Run M_p in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$
▷ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
▷ Adjust the distribution from M_p if needed.
$p'(x) \leftarrow p_{n+1}(x)$
if $n < \gamma$ **then**
  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if
▷ Return one token from M_p, and n tokens from M_q.
$t \sim p'(x)$
return $prefix + [x_1, \ldots, x_n, t]$

Method of speculative decoding

2. Sampling

Step 1: $\boldsymbol{M_q}$ autoregressively generate $\boldsymbol{\gamma}$ guessed tokens.

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.

▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$

  $x_i \sim q_i(x)$

end for

▷ Run M_p in parallel.

$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$

  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$

▷ Determine the number of accepted guesses n.

$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$

▷ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

if $n < \gamma$ **then**

  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$

end if

▷ Return one token from M_p, and n tokens from M_q.

$t \sim p'(x)$

return $prefix + [x_1, \ldots, x_n, t]$

Method of speculative decoding

2. Sampling

Step 1: $\boldsymbol{M_q}$ autoregressively generate $\boldsymbol{\gamma}$ guessed tokens.
Step 2: $\boldsymbol{M_p}$ examine these $\boldsymbol{\gamma}$ tokens in parallel.

Algorithm 1 SpeculativeDecodingStep

```
Inputs: M_p, M_q, prefix.
▷ Sample γ guesses x_{1,...,γ} from M_q autoregressively.
for i = 1 to γ do
    q_i(x) ← M_q(prefix + [x_1, ..., x_{i-1}])
    x_i ~ q_i(x)
end for
▷ Run M_p in parallel.
p_1(x), ..., p_{γ+1}(x) ←
      M_p(prefix), ..., M_p(prefix + [x_1, ..., x_γ])
▷ Determine the number of accepted guesses n.
r_1 ~ U(0,1), ..., r_γ ~ U(0,1)
n ← min({i - 1 | 1 ≤ i ≤ γ, r_i > p_i(x)/q_i(x)} ∪ {γ})
▷ Adjust the distribution from M_p if needed.
p'(x) ← p_{n+1}(x)
if n < γ then
    p'(x) ← norm(max(0, p_{n+1}(x) - q_{n+1}(x)))
end if
▷ Return one token from M_p, and n tokens from M_q.
t ~ p'(x)
return prefix + [x_1, ..., x_n, t]
```

Method of speculative decoding

2. Sampling

Step 1: $\boldsymbol{M_q}$ autoregressively generate $\boldsymbol{\gamma}$ guessed tokens.

Step 2: $\boldsymbol{M_p}$ examine these $\boldsymbol{\gamma}$ guesses in parallel.

Step 3: determine the number $\boldsymbol{n}$, accept guessed tokens from 1 to $\boldsymbol{n}$.

In greedy search, examine if $q_i(x) = argmax\big(p(x_i|x_{<i})\big)$

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.

▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$

  $x_i \sim q_i(x)$

end for

▷ Run M_p in parallel.

$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$

  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$

▷ Determine the number of accepted guesses n.

$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$

▷ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

if $n < \gamma$ **then**

  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$

end if

▷ Return one token from M_p, and n tokens from M_q.

$t \sim p'(x)$

return $prefix + [x_1, \ldots, x_n, t]$

Method of speculative decoding

2. Sampling

Step 1: $\boldsymbol{M_q}$ autoregressively generate $\boldsymbol{\gamma}$ guessed tokens.
Step 2: $\boldsymbol{M_p}$ examine these $\boldsymbol{\gamma}$ guesses in parallel.
Step 3: determine the number $\boldsymbol{n}$, accept guessed tokens from 1 to $\boldsymbol{n}$.
In greedy search, examine if $q_i(x) = argmax(p(x_i|x_{<i}))$
Step 4: $\boldsymbol{M_p}$ generate $p_{n+1}(x)$
In greedy search, $x_{n+1} = argmax(p_{n+1}(x))$

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.
▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.
for $i = 1$ **to** γ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$
  $x_i \sim q_i(x)$
end for
▷ Run M_p in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$
▷ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \le i \le \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
▷ Adjust the distribution from M_p if needed.
$p'(x) \leftarrow p_{n+1}(x)$
if $n < \gamma$ **then**
  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if
▷ Return one token from M_p, and n tokens from M_q.
$t \sim p'(x)$
return $prefix + [x_1, \ldots, x_n, t]$

Analysis on the efficiency

- Let $\boldsymbol{\alpha}$ be the expectation of acceptance rate.
- $E(\#generated\_tokens) = 1 \times (1-\alpha) + 2 \times (\alpha - \alpha^2) + 3 \times (\alpha^2 - \alpha^3) + \cdots + \gamma \times (\alpha^{(\gamma-1)} - \alpha^\gamma) + (\gamma+1) \times \alpha^\gamma$

$$= (1-\alpha)\left(1 + 2\alpha + 3\alpha^2 + \cdots + \gamma\alpha^{(\gamma-1)}\right) + (\gamma+1)\alpha^\gamma$$

$$= 1 + \alpha + \alpha^2 + \cdots + \alpha^\gamma$$

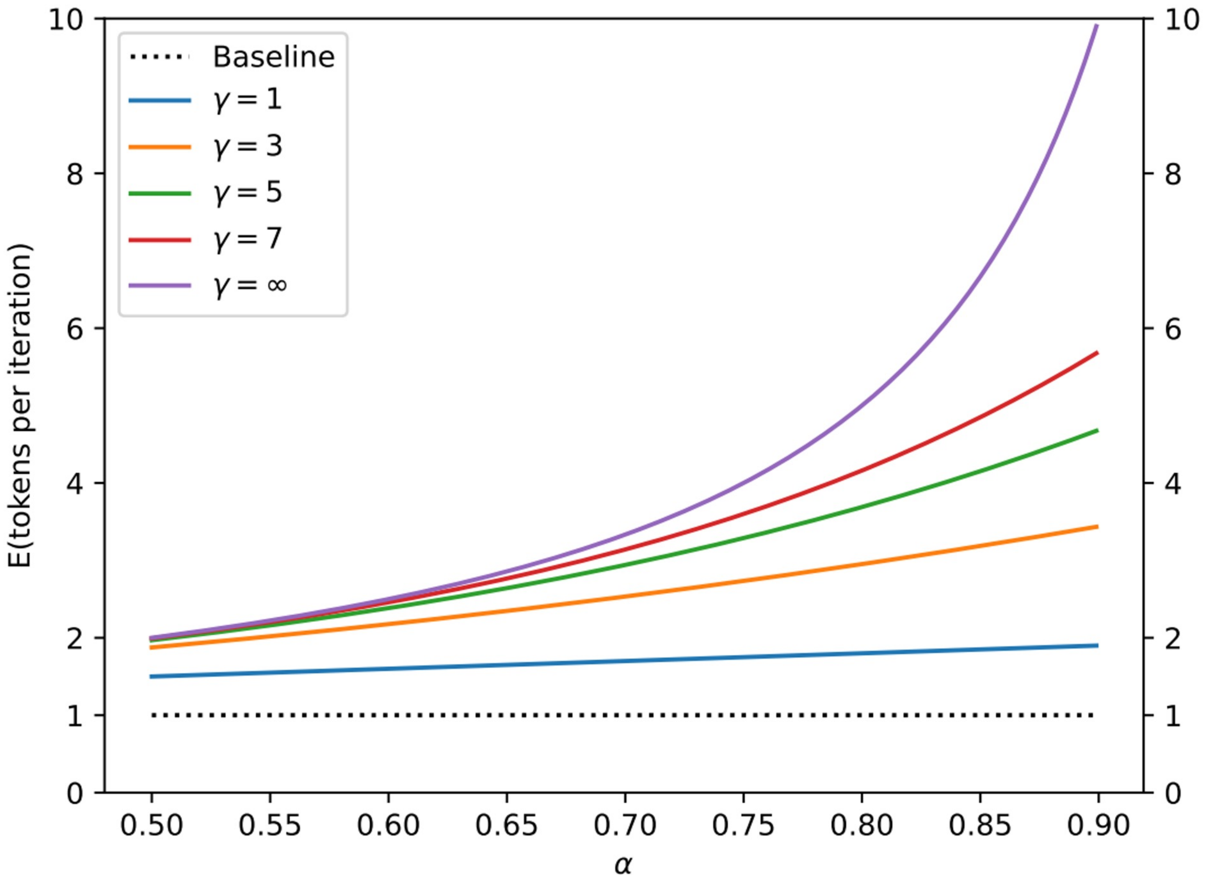
$$= \frac{1-\alpha^{\gamma+1}}{1-\alpha}$$

Analysis on the efficiency

$$E(\#generated\_tokens) = \frac{1-\alpha^{\gamma+1}}{1-\alpha}$$

We need bigger γ and α!

γ: number of tokens small model generates
α: the divergence between two models



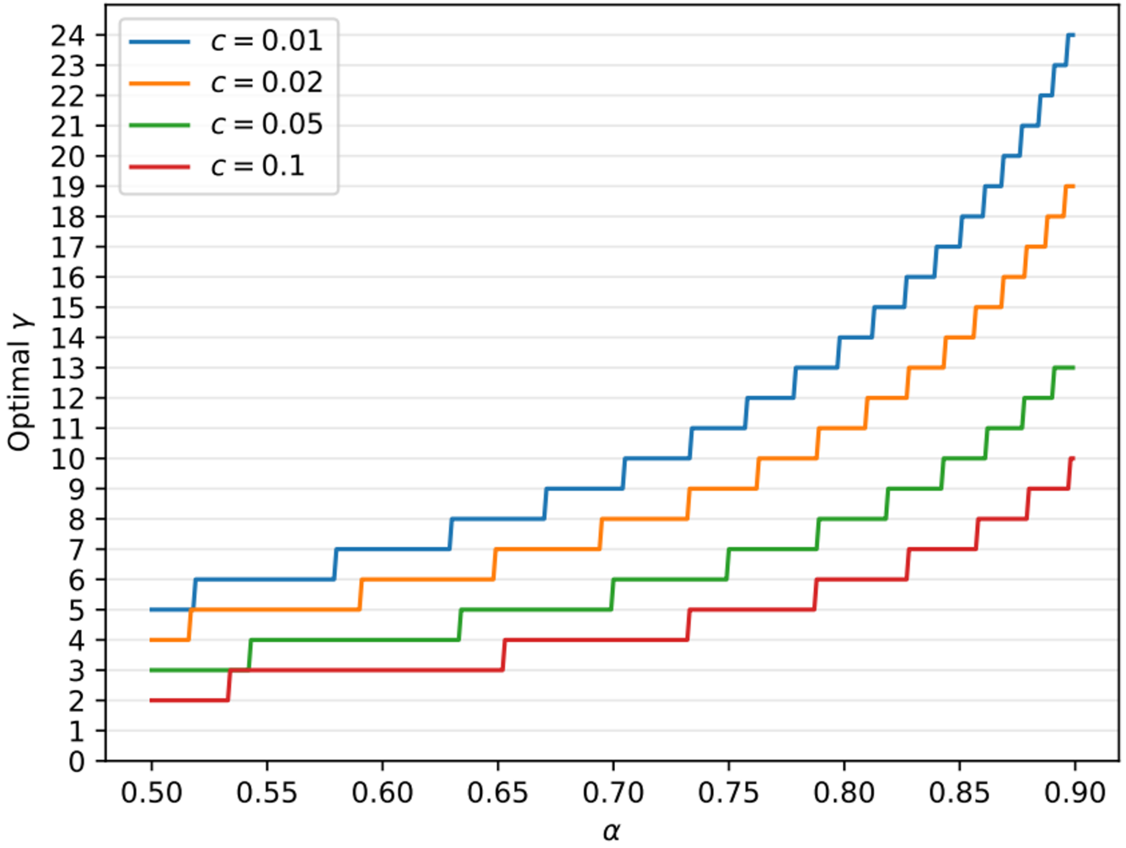
The wall time cost

The overall time cost, including time spent on both models

- Let's assume the ratio between running a small model and the main model is $\boldsymbol{c}$
- For a single round, the time cost is $T + Tc\gamma$, the tokens generated is $\frac{1-\alpha^{\gamma+1}}{1-\alpha}$, so the average time cost to generate a token is $\frac{(c\gamma+1)(1-\alpha)}{1-\alpha^{\gamma+1}}T$. The theoretic accelerate ratio is $\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$.

Optimize the wall time cost

- Assume the compute resources are infinite, then we can simply optimize this number $\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$.



Visualization of the time cost

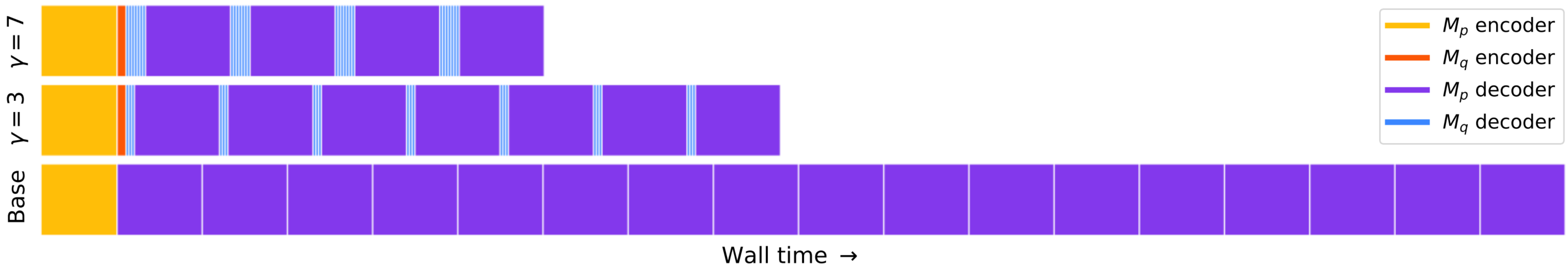


Figure 5. A simplified trace diagram for a full encoder-decoder Transformer stack. The top row shows speculative decoding with $\gamma = 7$ so each of the calls to M_p (the purple blocks) is preceded by 7 calls to M_q (the blue blocks). The yellow block on the left is the call to the encoder for M_p and the orange block is the call to the encoder for M_q. Likewise the middle row shows speculative decoding with $\gamma = 3$, and the bottom row shows standard decoding.

Empirical experiments

Table 2. Empirical results for speeding up inference from a T5-XXL 11B model.

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	**3.4X**
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	**2.6X**
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNDM	T5-SMALL ★	0	5	0.65	**3.1X**
CNNDM	T5-BASE	0	5	0.73	3.0X
CNNDM	T5-LARGE	0	3	0.74	2.2X
CNNDM	T5-SMALL ★	1	5	0.53	**2.3X**
CNNDM	T5-BASE	1	3	0.55	2.2X
CNNDM	T5-LARGE	1	3	0.56	1.7X

The importance of having two models "conjugate"

M_p	M_q	SMPL	α
GPT-LIKE (97M)	UNIGRAM	T=0	0.03
GPT-LIKE (97M)	BIGRAM	T=0	0.05
GPT-LIKE (97M)	GPT-LIKE (6M)	T=0	0.88
GPT-LIKE (97M)	UNIGRAM	T=1	0.03
GPT-LIKE (97M)	BIGRAM	T=1	0.05
GPT-LIKE (97M)	GPT-LIKE (6M)	T=1	0.89
T5-XXL (ENDE)	UNIGRAM	T=0	0.08
T5-XXL (ENDE)	BIGRAM	T=0	0.20
T5-XXL (ENDE)	T5-SMALL	T=0	0.75
T5-XXL (ENDE)	T5-BASE	T=0	0.80
T5-XXL (ENDE)	T5-LARGE	T=0	0.82
T5-XXL (ENDE)	UNIGRAM	T=1	0.07
T5-XXL (ENDE)	BIGRAM	T=1	0.19
T5-XXL (ENDE)	T5-SMALL	T=1	0.62
T5-XXL (ENDE)	T5-BASE	T=1	0.68
T5-XXL (ENDE)	T5-LARGE	T=1	0.71

T5-XXL (CNNDM)	UNIGRAM	T=0	0.13
T5-XXL (CNNDM)	BIGRAM	T=0	0.23
T5-XXL (CNNDM)	T5-SMALL	T=0	0.65
T5-XXL (CNNDM)	T5-BASE	T=0	0.73
T5-XXL (CNNDM)	T5-LARGE	T=0	0.74
T5-XXL (CNNDM)	UNIGRAM	T=1	0.08
T5-XXL (CNNDM)	BIGRAM	T=1	0.16
T5-XXL (CNNDM)	T5-SMALL	T=1	0.53
T5-XXL (CNNDM)	T5-BASE	T=1	0.55
T5-XXL (CNNDM)	T5-LARGE	T=1	0.56
LAMDA (137B)	LAMDA (100M)	T=0	0.61
LAMDA (137B)	LAMDA (2B)	T=0	0.71
LAMDA (137B)	LAMDA (8B)	T=0	0.75
LAMDA (137B)	LAMDA (100M)	T=1	0.57
LAMDA (137B)	LAMDA (2B)	T=1	0.71
LAMDA (137B)	LAMDA (8B)	T=1	0.74

WashU McKelvey Engineering

MEDUSA: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads

Tianle Cai[*1,2] Yuhong Li[*3] Zhengyang Geng[4] Hongwu Peng[5] Jason D. Lee[1] Deming Chen[3] Tri Dao[1,2]

Princeton & UIUC

Published at ICML2024

https://openreview.net/pdf?id=PEpbUobfJv

blog: https://sites.google.com/view/medusa-llm

video： https://icml.cc/virtual/2024/poster/34133

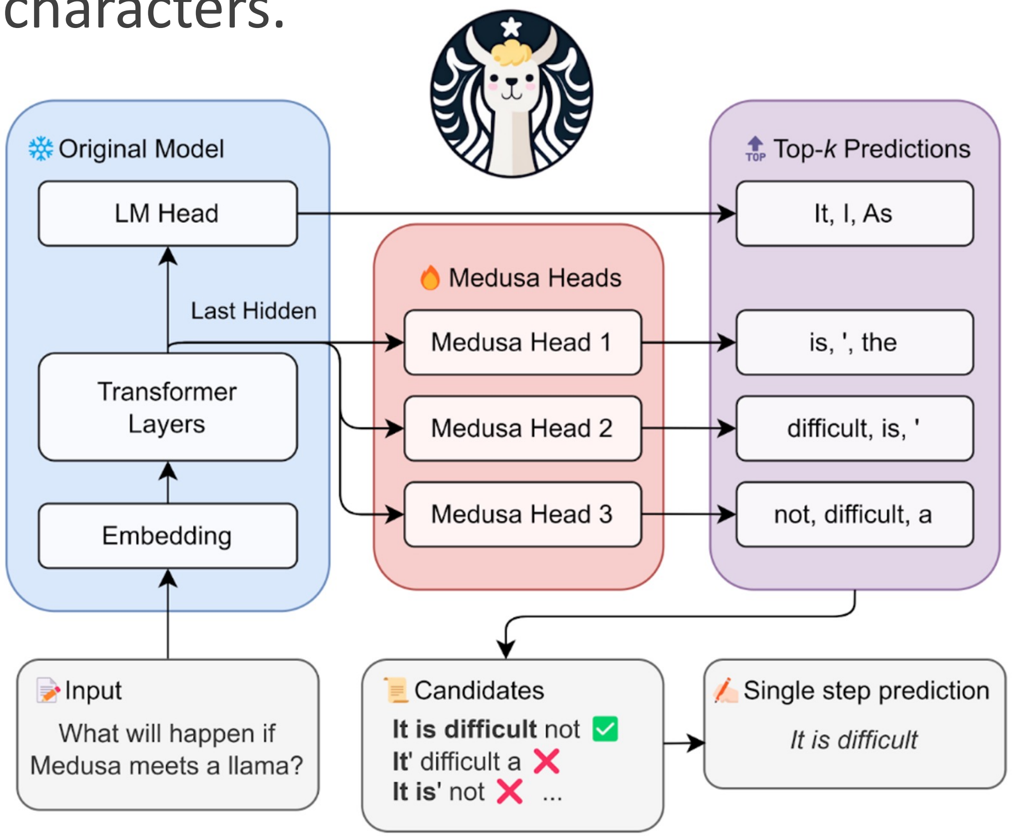
GitHub： https://github.com/FasterDecoding/Medusa/tree/main

General idea of Medusa

Speculative decoding suffers from the discrepancy between two models

Using one primary model structure to act as both characters.

Specifically, medusa runs 1 time of main body and predict the next n tokens, with corresponding decoding heads.



Multiple heads from Medusa

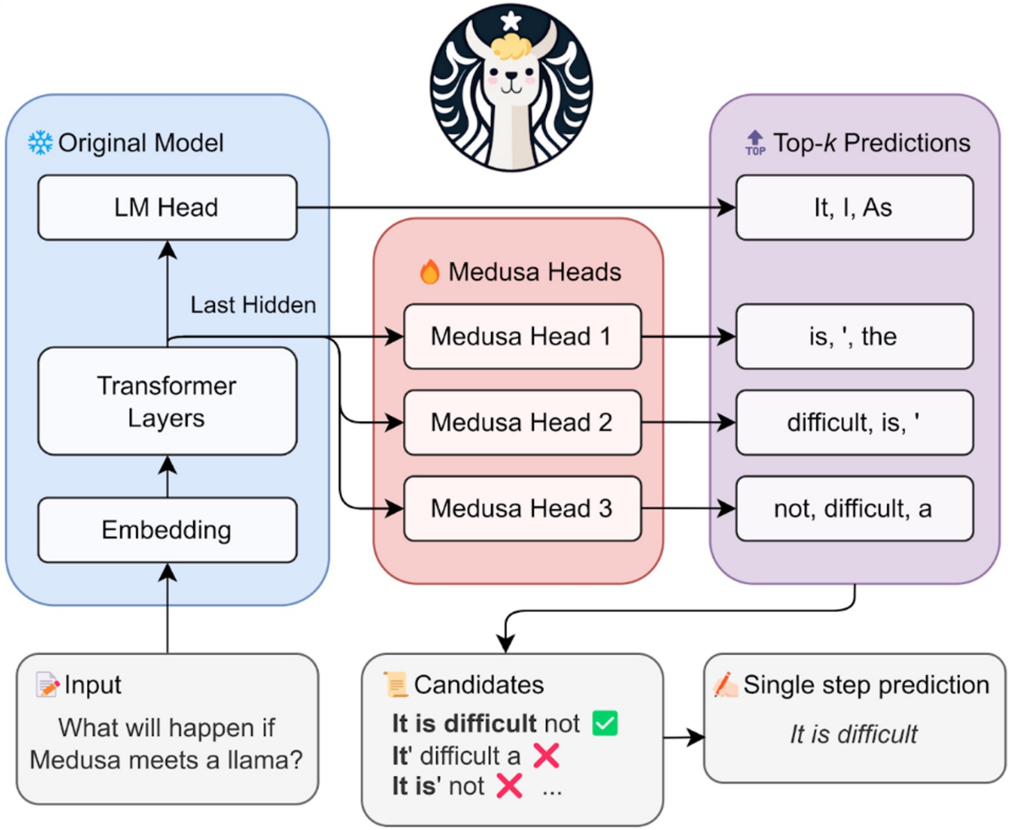
An LM head projects a hidden state to a distribution over the vocabulary

The traditional head predicts the next 1 token.

Medusa has multiple heads for the next few tokens each.

Each head is a simple FFN:

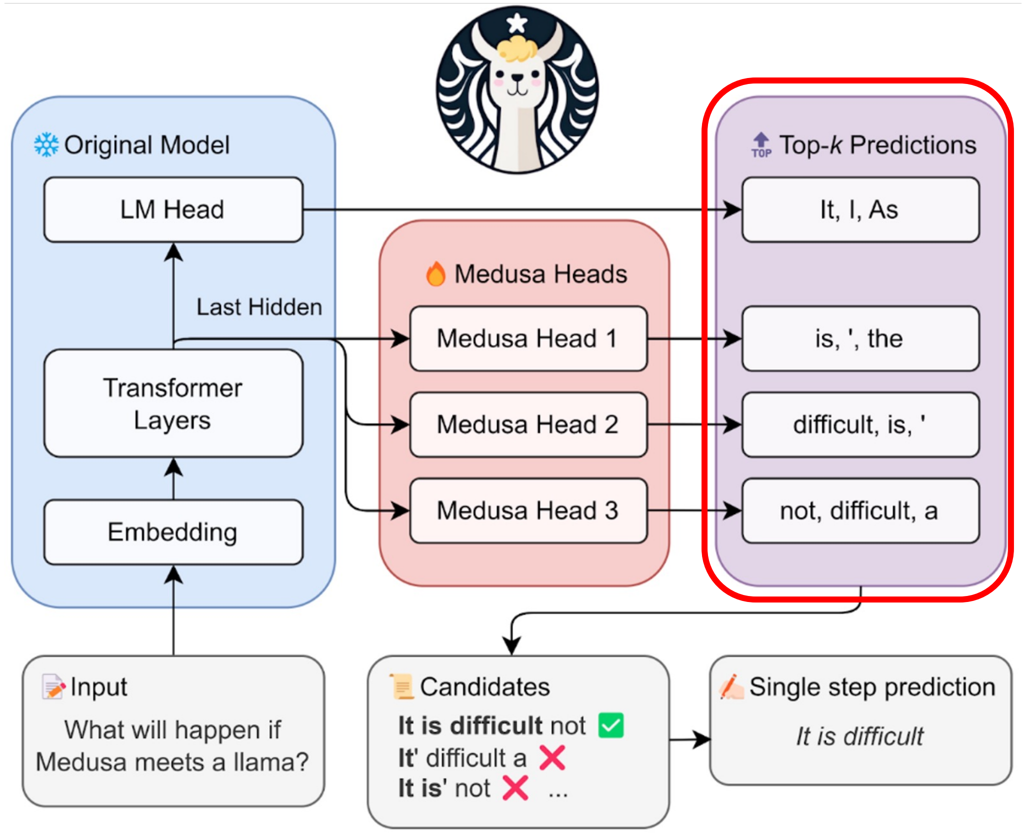
$$p_t^{(k)} = \text{softmax}\left(W_2^{(k)} \cdot \left(\text{SiLU}(W_1^{(k)} \cdot h_t) + h_t\right)\right), \text{ where } W_2^{(k)} \in \mathbb{R}^{d\times V}, W_1^{(k)} \in \mathbb{R}^{d\times d}.$$



Decoding strategy

How does Medusa verify the drafted tokens?

Instead of greedy search or sampling, Medusa predicts Top-k tokens for each position, and verify their combinations.

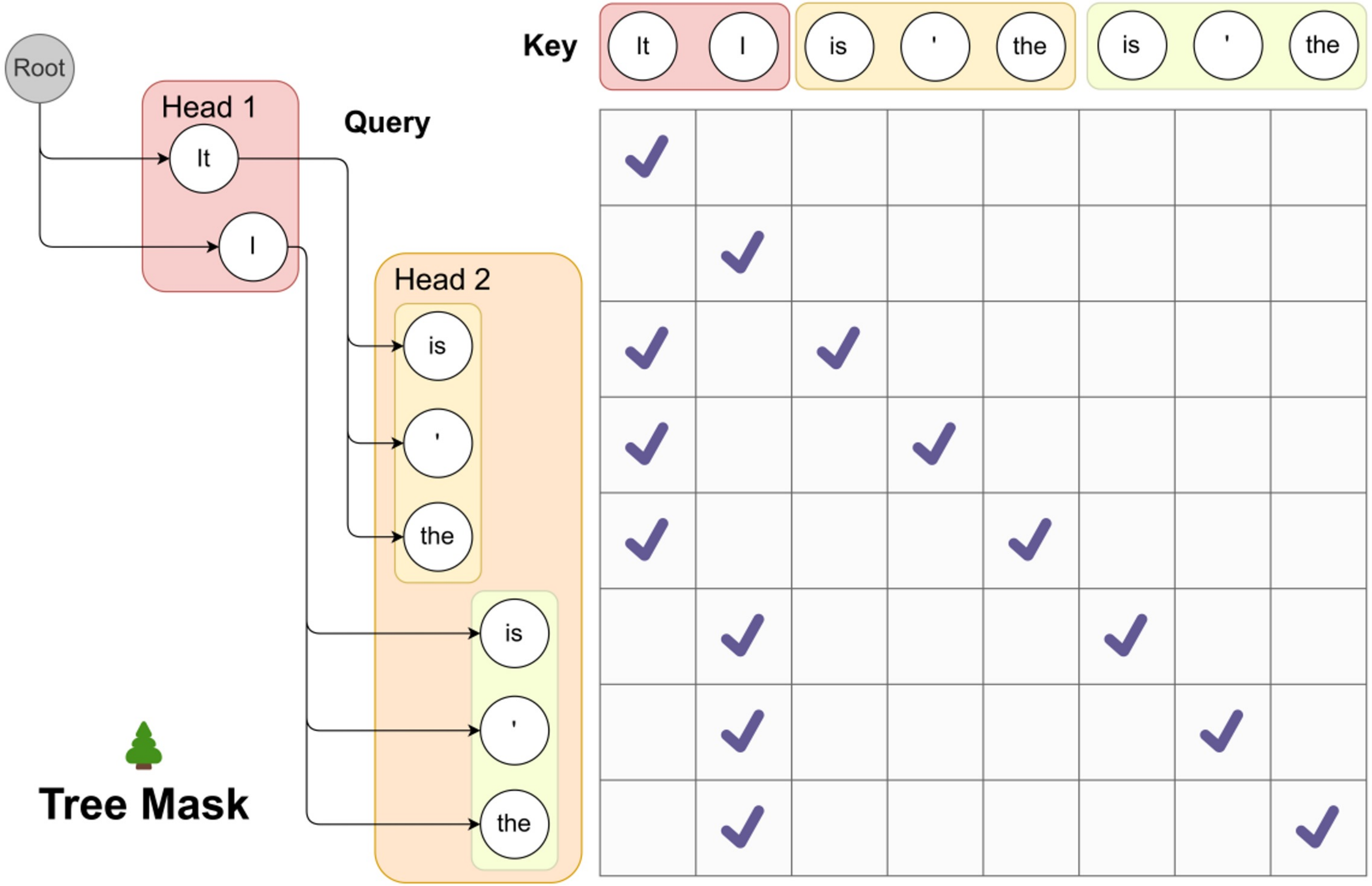


Decoding strategy

Verification step

Examine the token combinations in parallel with one run.

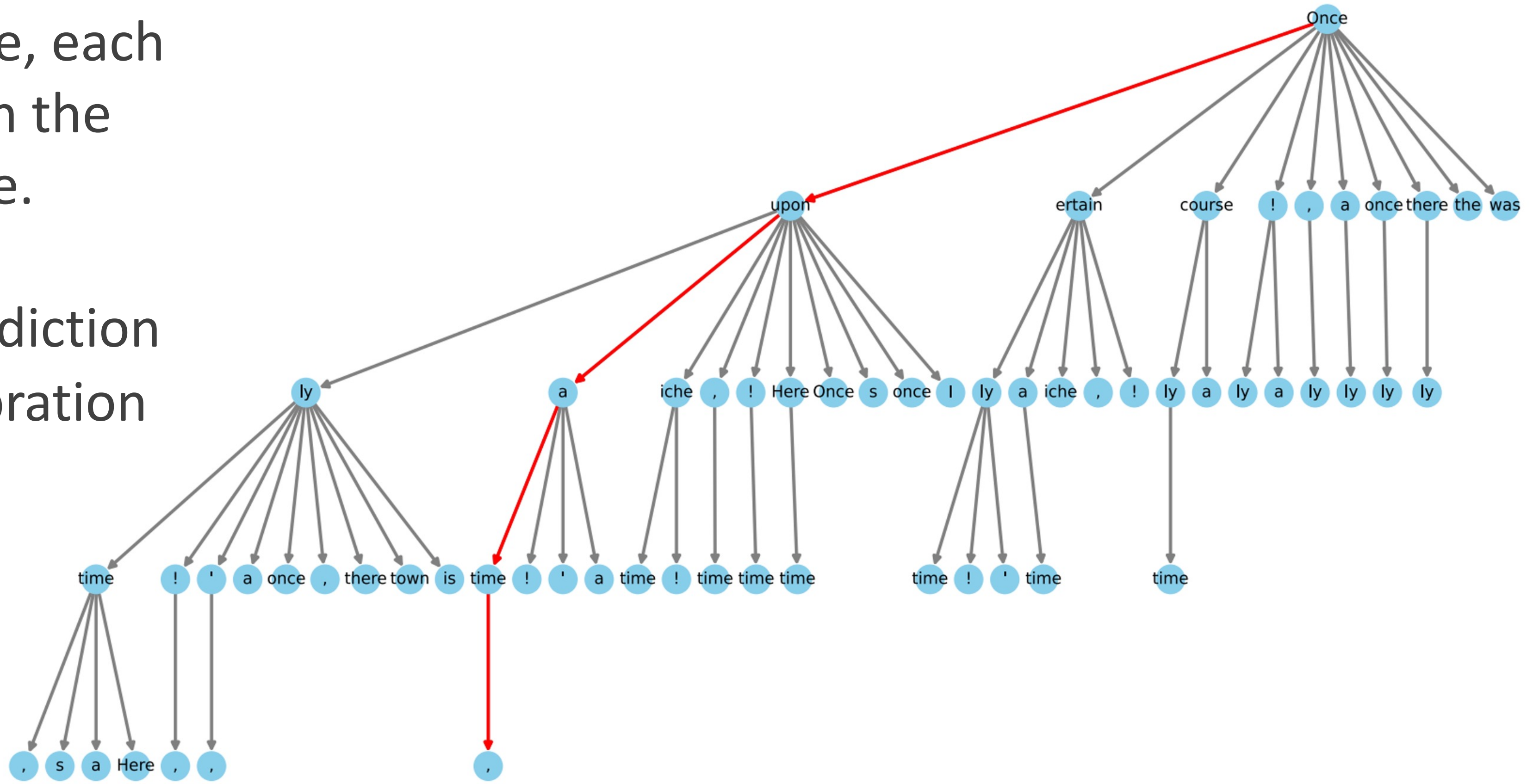
Candidates from different positions form a Cartesian set.



Extension to tree attention

Build the tree node by node, each time connect the node with the highest accuracy to the tree.

Accuracy of the i^{th} top prediction of the k^{th} head: use a calibration dataset to calculate.



The training process

Training few Medusa heads suffices, but training with main body proves better

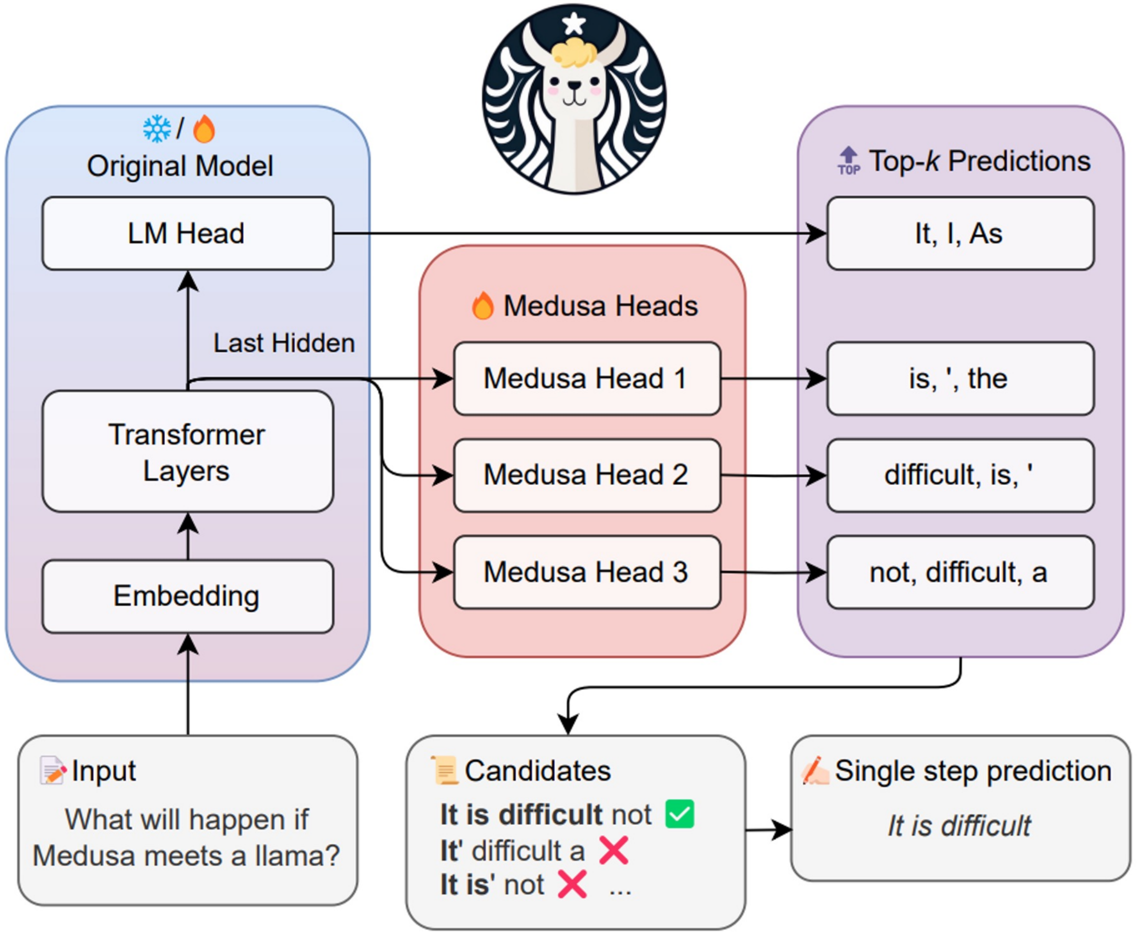
Cross-Entropy loss:

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

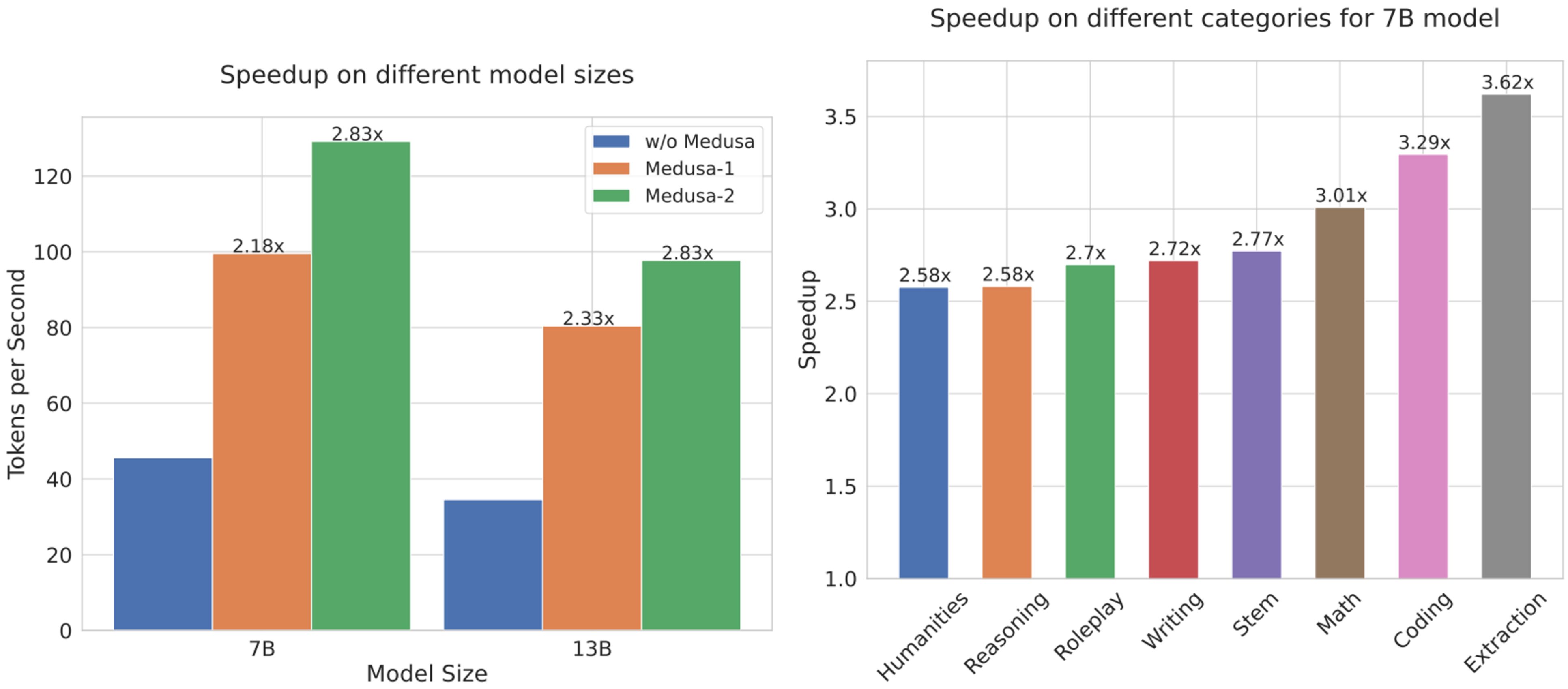
As the position k goes up, CE loss becomes larger, so $\lambda_k = 0.8^k$ is applied.

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

Gradually increase λ_0



Experiment results



(a) (b)

Demonstration

Prompt Compression and Contrastive Conditioning for Controllability and Toxicity Reduction in Language Models

David Wingate
Brigham Young University*
wingated@cs.byu.edu

Mohammad Shoeybi
Nvidia, Inc.
mshoeybi@nvidia.com

Taylor Sorensen
University of Washington†
tsor13@cs.washington.edu

https://arxiv.org/abs/2210.03162

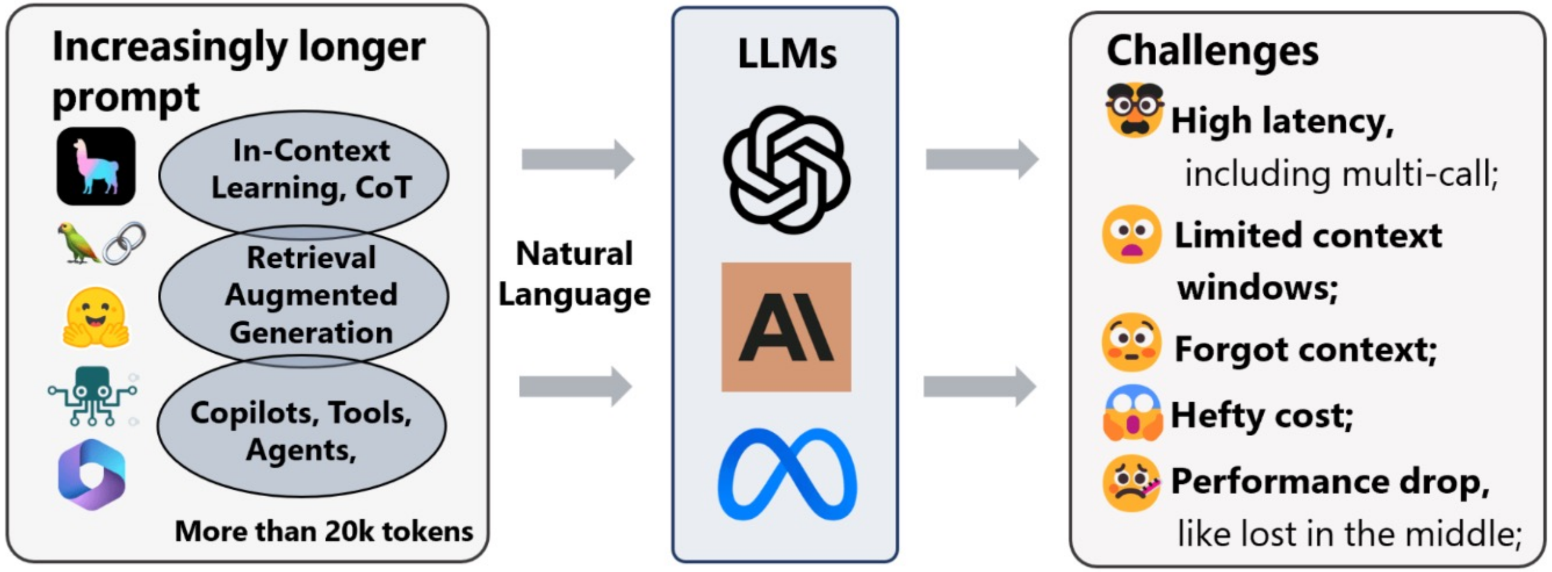
Introduction

The Role of Prompt Compression in LLMs

- Reducing Input Size
- Decreasing Attention Mechanism Complexity
- Reducing Latency
- Efficient Memory Usage
- Cost Reduction
- Prevents Model Overload

Background

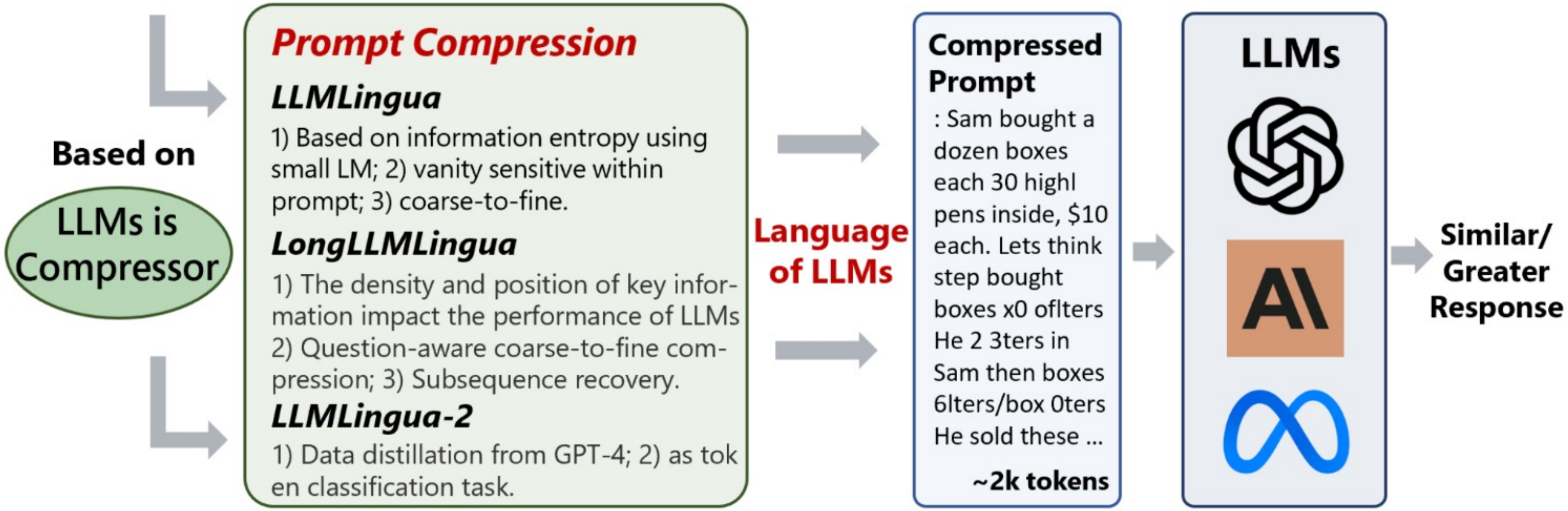
Why we need compress input?



How to efficiently utilize limited tokens while retaining and enhancing the information contained in the prompt?

Prompt Compression

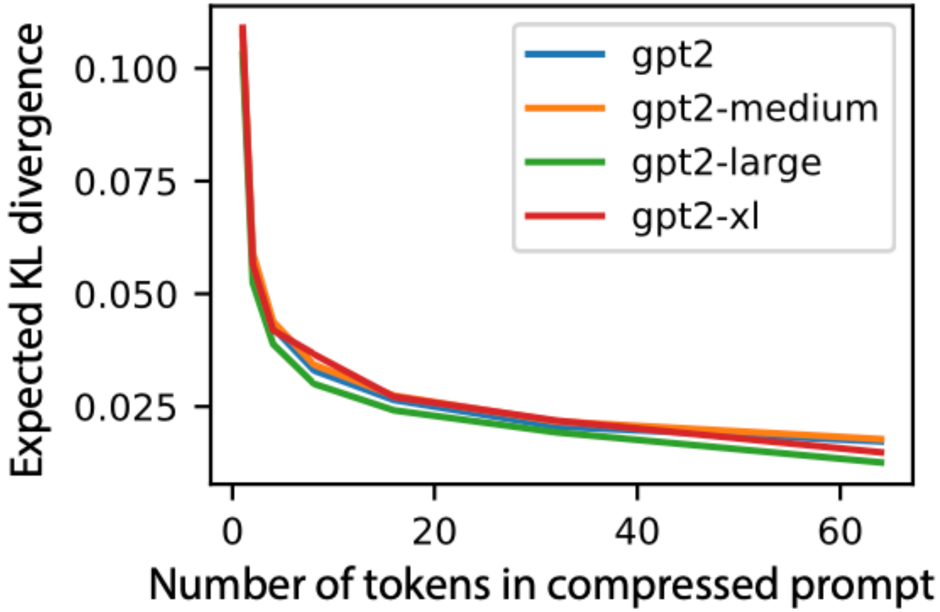
How to efficiently utilize limited tokens while retaining and enhancing the information contained in the prompt?



Background and Related Work

Compressed Prompts for various sizes of GPT-2 models – The influence of the length of the prompt

- Smaller KL divergence means the compressed prompt is closer to the original prompt in terms of information content
- The longer the compressed prompt (i.e., more tokens), the more information remains



Main Methods

- Hard prompt as a Baseline;
- Compressed (soft) prompt is trained to approximate the behavior of the hard prompt;

$$\min_{\theta_n} \mathbb{E}_{x_{t:k}} \left[\mathrm{KL}(p(x_{t:k}|x_h) || q(x_{t:k}|\theta_n)) \right]$$

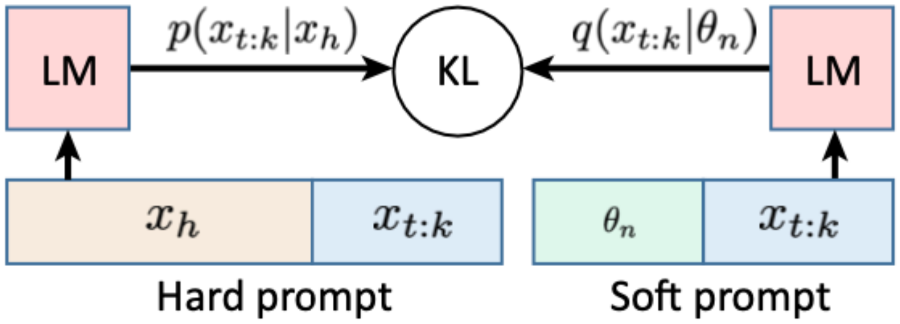
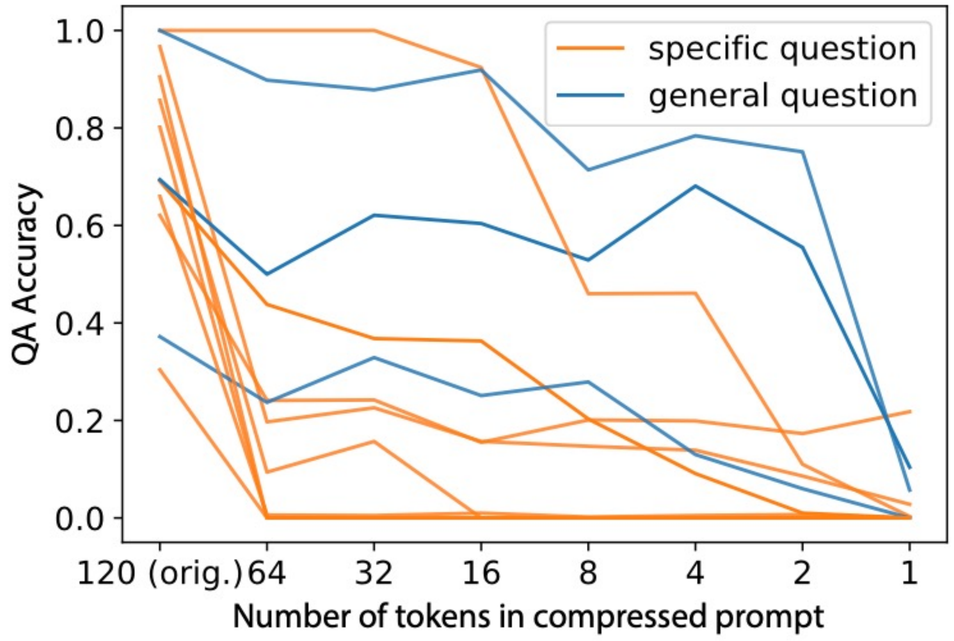


Figure 1: Schematic of prompt compression. Weights of the soft prompt are tuned to minimize the KL divergence between hard and soft prompts, for all $x_{t:k}$.

Reading Comprehension Task

- As the prompt is compressed, accuracy for specific question degrades more rapidly
- (GPT-2 xl for this experiment.)



Reconstruction Task

Figure 4: Assessing the information retained as a prompt is compressed more and more severely. The model is tasked with recovering the passage given a hard prompt (the passage), compressed prompts, or no prompt. For each token, likelihood is calculated and scaled so that the probability according to the hard context is 1 and the probability with no context is 0. It is visualized with a heatmap, where yellow corresponds to 1 (hard context) and pink corresponds to 0 (no context).

Contrastive Contexts

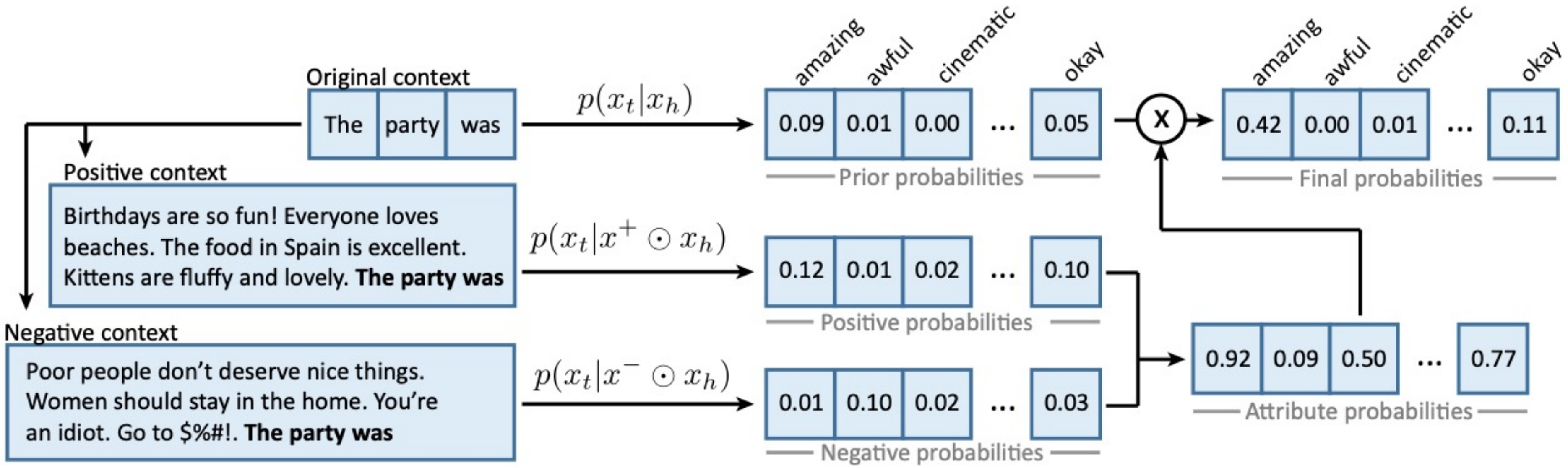


Figure 5: Contrastive conditioning. Content warning: The example text is offensive. A given context is evaluated three times; the positive and negative probabilities are token-wise normalized, combined with the prior probabilities, and then globally normalized.

Results in hard contexts

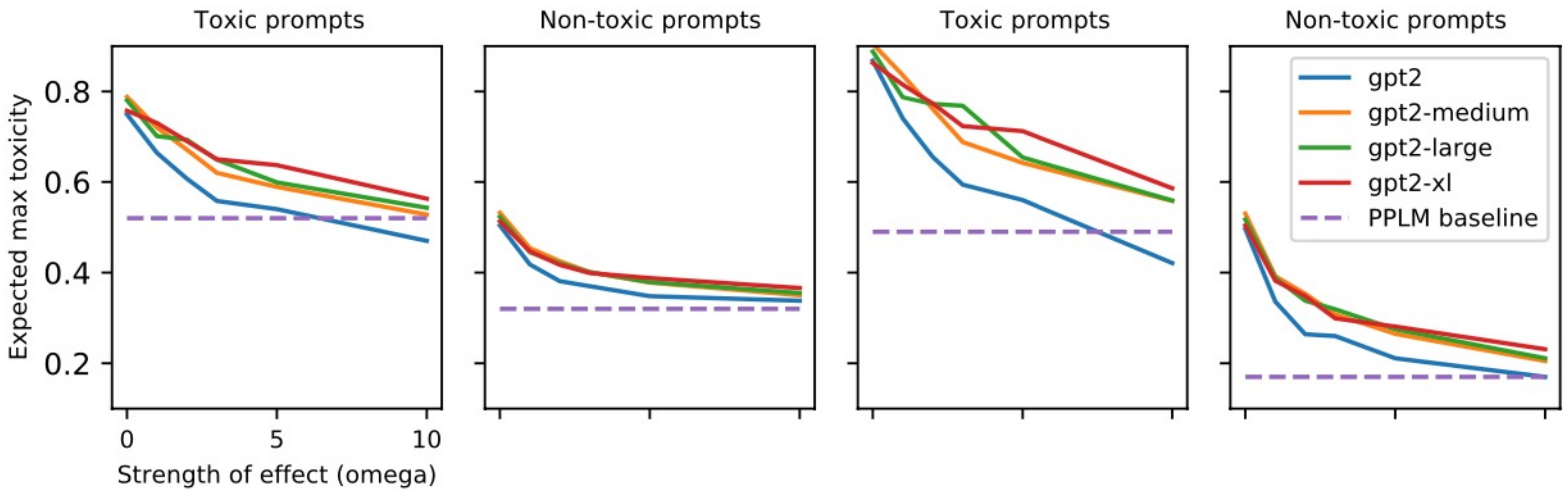


Figure 6: Toxicity reduction using hard contexts, for various settings of the ω parameter and various model sizes. Smaller models experience a stronger effect.

Results in soft contexts

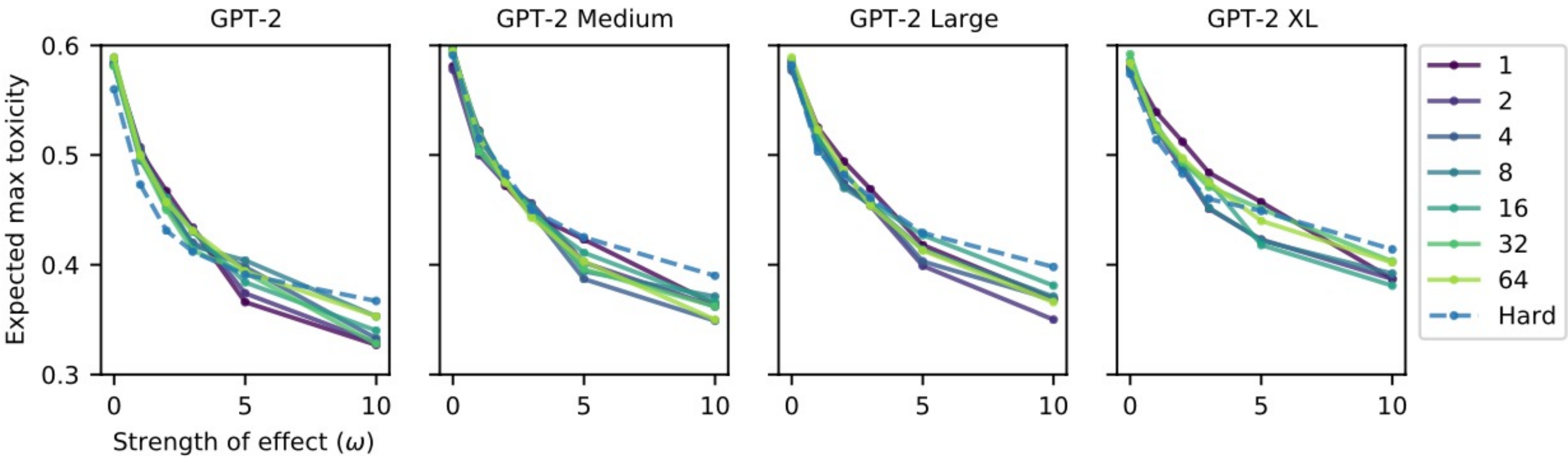


Figure 7: Toxicity reduction using compressed prompts, for various settings of the ω parameter, various model sizes, and various amounts of compression. Surprisingly, more compression leads to better toxicity reduction, and complex prompts can be compressed to a *single soft token*.

Questions?

WashU McKelvey Engineering

Adapting Language Models to Compress Contexts

Alexis Chevalier* **Alexander Wettig*** **Anirudh Ajith** **Danqi Chen**
Department of Computer Science & Princeton Language and Intelligence
Princeton University
{achevalier, anirudh.ajith}@princeton.edu
{awettig, danqic}@cs.princeton.edu

https://arxiv.org/abs/2305.14788

Introduction

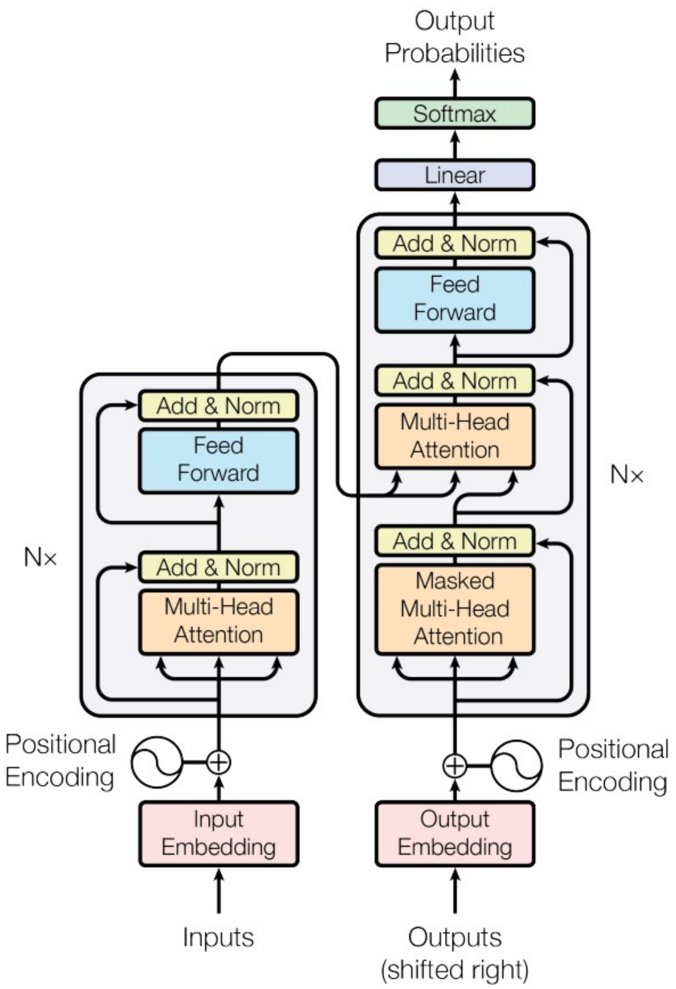
The paper builds on several established concepts in machine learning and NLP

- **Soft Prompt Tuning:**
  Tunable prompts that adjust to tasks without changing the model
- **Long-range Transformers:**
  Reducing context while keeping key information

Introduction

Transformer-based models

- Rely on fixed-size input sequences
- Computationally expensive
- Inefficient for long document processing



https://arxiv.org/pdf/1706.03762

AutoCompressor: How Summary Tokens and Vectors Work

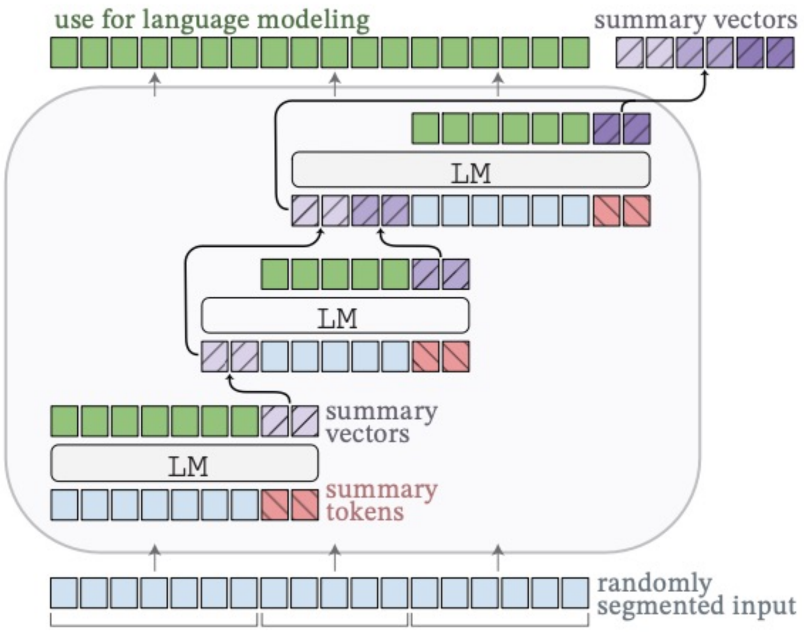


Figure 1: *AutoCompressors* process long documents by recursively generating summary vectors which are passed as soft prompts to all subsequent segments.

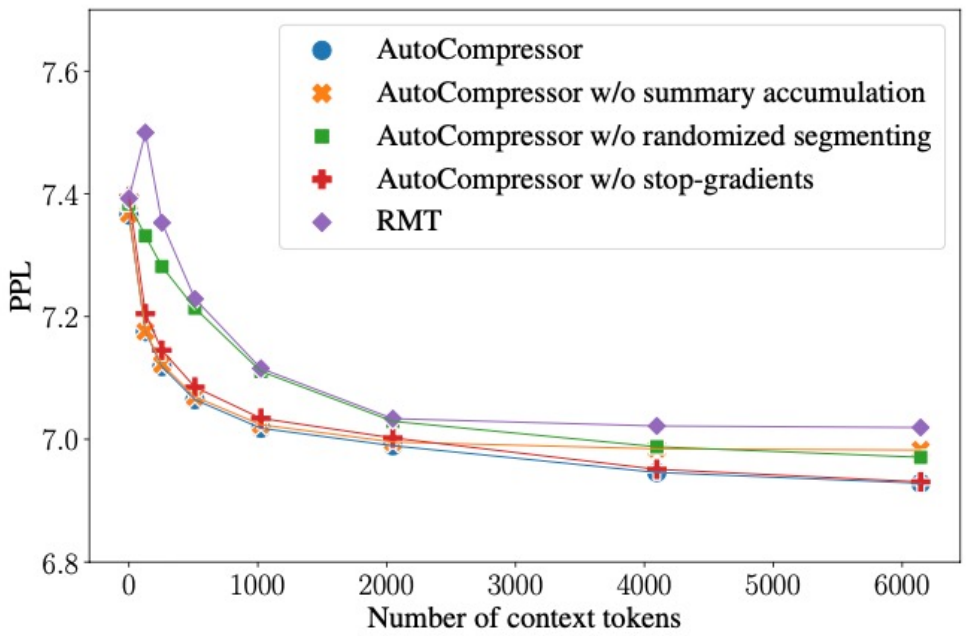
- Summary tokens direct the model to produce Summary Vectors
- Summary Vectors allow the model to retain and access long-range context efficiently

Training Summary Vectors with Cross-Entropy Loss

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{n}\sum_{t=1}^{m_i}\log p(x_t^i \mid x_1^i, \ldots, x_{t-1}^i, \sigma_{<i}).$$

- σ<i: The **summary vectors** generated from all previous segments.

Efficient Training: Randomized Segments & BPTT



- **Randomized Segmenting**
  handle text segments of various lengths

- **Stopping Gradients**
  reduces memory use without affecting performance

Methods

Improved Long-Sequence Processing with AutoCompressors

	In-domain					Out-of-domain				
Segments	*1*	*1*	*1*	*– 2 –*	*– 3 –*	*1*	*1*	*1*	*– 2 –*	*– 3 –*
Context tokens	128	512	2048	4096	6144	128	512	2048	4096	6144
Extended FA†	6.33† ↑1.0%	6.15† ↓2.1%	5.94† ↓5.4%	-	-	8.57† ↑0.5%	8.28† ↓2.9%	7.93† ↓7.0%	-	-
RMT	6.42 ↑2.2%	6.19 ↓1.4%	6.02 ↓4.1%	6.02 ↓4.1%	6.01 ↓4.3%	8.76 ↑2.7%	8.44 ↓1.1%	8.21 ↓3.8%	8.20 ↓3.9%	8.20 ↓3.9%
AutoCompressor	6.14 ↓2.2%	6.04 ↓3.8%	5.98 ↓4.8%	5.94 ↓5.4%	**5.93** ↓5.6%	8.39 ↓1.6%	8.26 ↓3.2%	8.17 ↓4.2%	8.12 ↓4.8%	**8.10** ↓5.0%

Table 1: Held-out perplexity on 2,048 tokens, while varying the length of the preceding context (all the experiments are based on OPT-2.7B models). For RMT and AutoCompressor, we condition on summary vectors. We also report the perplexity gains compared to the fine-tuned OPT baseline without extra context, which achieves 6.28 in-domain and 8.53 out-of-domain (gains shown in colored numbers). †: Although the extended full attention (Extended FA) achieves similar or slightly better perplexity, it uses up to 2,048 additional tokens and cannot extend further. However, the AutoCompressor uses only $50 \times 3 = 150$ summary vectors to process 6,144 context tokens.

Methods

Few-Shot Learning Improvements with AutoCompressors

	AG News	SST-2	BoolQ	WIC	WSC	RTE	CB	COPA	MultiRC	MR	Subj
Zero-shot	$63.3_{(0.0)}$	$67.7_{(0.0)}$	$67.4_{(0.0)}$	$50.8_{(0.0)}$	$43.3_{(0.0)}$	$58.8_{(0.0)}$	$42.9_{(0.0)}$	$52.5_{(0.0)}$	$52.5_{(0.0)}$	$57.4_{(0.0)}$	$49.3_{(0.0)}$
50 summary vecs.	$79.6_{(4.9)}$	$\mathbf{94.2}_{(1.6)}$	$\mathbf{70.1}_{(3.3)}$	$51.6_{(2.1)}$	$47.7_{(8.7)}$	$66.3_{(7.0)}$	$46.4_{(18.7)}$	$84.5_{(1.0)}$	$52.6_{(2.8)}$	$91.5_{(1.0)}$	$53.5_{(3.6)}$
100 summary vecs.	$\mathbf{87.6}_{(1.2)}$	$92.6_{(3.3)}$	$66.3_{(2.8)}$	$52.5_{(2.2)}$	$42.9_{(2.5)}$	$63.5_{(6.6)}$	$\mathbf{64.5}_{(5.9)}$	$85.9_{(0.4)}$	$\mathbf{56.1}_{(1.2)}$	$90.7_{(2.6)}$	$57.0_{(5.6)}$
150 summary vecs.	$85.4_{(3.4)}$	$92.3_{(2.9)}$	$68.0_{(1.8)}$	$\mathbf{52.8}_{(1.5)}$	$49.9_{(7.6)}$	$65.3_{(6.6)}$	$54.8_{(5.8)}$	$\mathbf{86.1}_{(0.6)}$	$54.8_{(2.2)}$	$91.1_{(2.2)}$	$56.6_{(7.9)}$
ICL (150 tokens)	$74.5_{(2.2)}$	$92.4_{(3.1)}$	$67.4_{(0.0)}$	$52.4_{(2.7)}$	$\mathbf{51.8}_{(6.9)}$	$69.1_{(2.1)}$	$46.4_{(23.0)}$	$80.0_{(1.9)}$	$52.5_{(0.0)}$	$79.7_{(15.7)}$	$57.9_{(10.7)}$
ICL (750 tokens)	$81.2_{(4.1)}$	$93.8_{(1.2)}$	$67.7_{(2.7)}$	$52.4_{(2.0)}$	$40.0_{(5.7)}$	$\mathbf{73.1}_{(3.5)}$	$50.3_{(2.8)}$	$82.6_{(1.6)}$	$47.0_{(3.2)}$	$\mathbf{91.6}_{(0.8)}$	$\mathbf{60.7}_{(14.8)}$

Table 4: Evaluation of the ICL performance of the Llama-2 7B model. Each summary is 50 tokens-long and corresponds to a segment of 750 tokens' worth of demonstrations. We also report accuracies when prompting the AutoCompressor with 150 and 750 tokens' worth of plaintext demonstrations as baselines. Note that for BoolQ and MultiRC, demonstrations are too long to fit into 150 tokens.

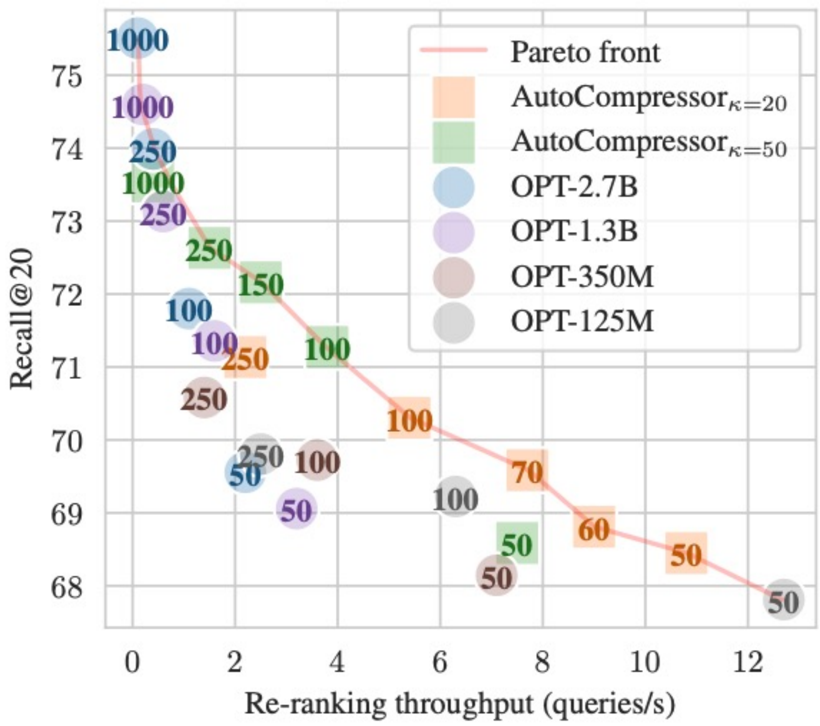
Results

Fused Summaries achieves a good trade-off between storage costs and throughput.

Passages		Perplexity Gain (%)				Throughput (examples/s)			
		top-1	top-2	top-5	top-10	top-1	top-2	top-5	top-10
50 tokens	REPLUG	-0.64	0.58	1.68	2.35	51	38	16	9
50 tokens	Fused Passages	0.71	1.01	1.70	2.60	28	27	23	17
512 tokens $\rightarrow$ 50 sum. vecs.	Fused Summaries	**1.04**	**1.67**	**2.63**	**3.74**	28	27	23	17
512 tokens	REPLUG	-1.47	2.24	5.25	8.30	18	10	6	3

Table 5: PPL gains (%) from different retrieval-augmented language modeling settings, over the no-retrieval baseline. We evaluate the OPT-2.7B AutoCompressor and we report throughput on a single NVIDIA A100 GPU for each method without batching examples. Fused Summaries outperforms Fused Passages and REPLUG with 50-token passages. Moreover, Fused Summaries top-10 outperforms REPLUG top-2 with 512-token passages while also gaining a 1.7× throughput increase.

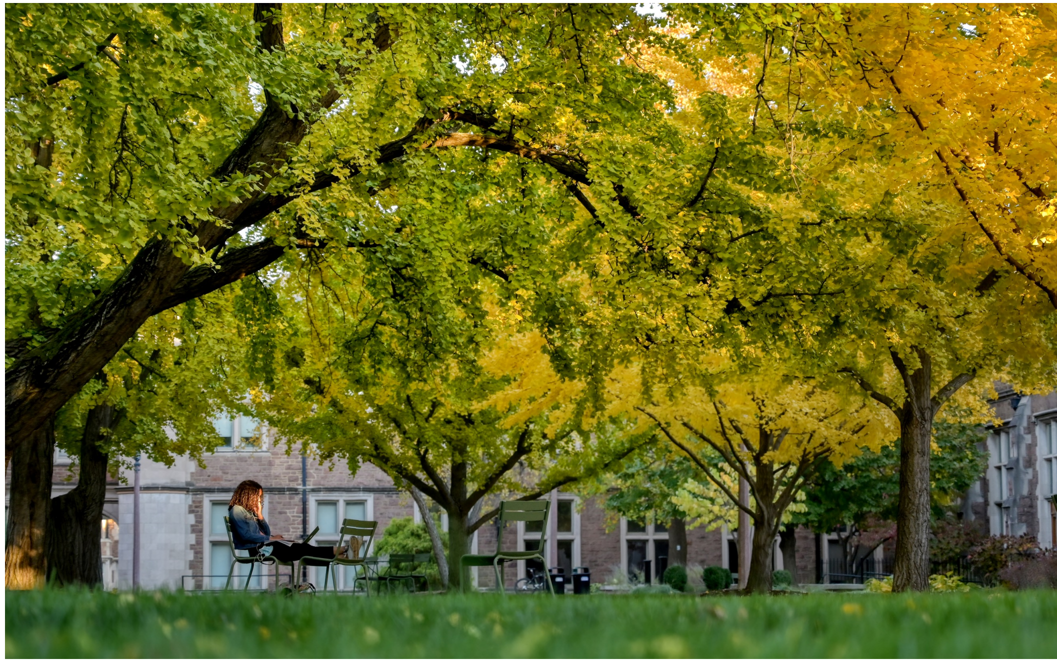
Performance vs. Throughput in Passage Re-ranking



- AutoCompressors achieve a strong balance of high recall and efficient throughput, outperforming traditional models in passage re-ranking.

Applications and Future Work

- Retrieval tasks:
  Summary vectors enable efficient retrieval and ranking of relevant documents
- Document summarization and text generation:
  Compressing long contexts improves performance and reduces computational costs
- Scalability to Larger Models
- Improving Summary Vector Quality
- Efficient Multimodal Inference



WashU McKelvey Engineering

Thanks for your attention!