Carnegie Mellon University

Speculative Decoding

Richa Gadgil, Yangyong Deng, Haoyang Cai

Outline

1. **Motivation**
2. Related work
3. Foundation
4. Algorithm
5. Experiment
6. Summary and Future direction

Large model inference is Inefficient

Large autoregressive models like Transformers are becoming increasingly more capable but also more computationally expensive during inference.

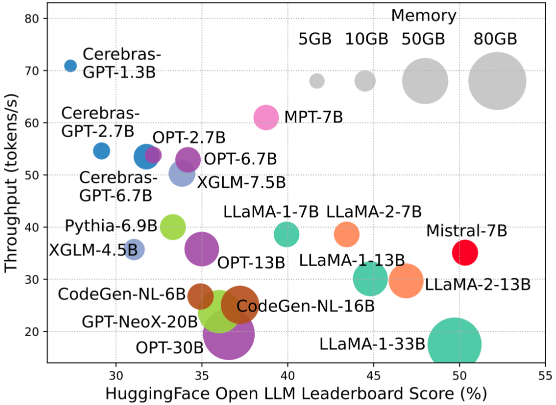
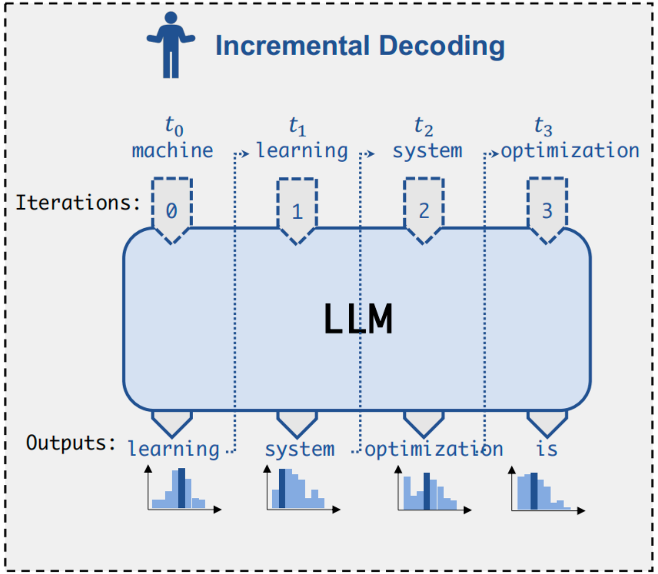


Figure 2: Performance score *vs.* inference throughput for various LLMs. The throughputs are measured on Nvidia A100 80GB GPU with 16-bit floating point quantization. The size of each circle corresponds to the memory footprint (in Gigabytes) of each model when running with batch size of 1, prompt size of 256 and generating 1000 tokens. The original data can be found in Ilyas Moutawwakil (2023).

Autoregressive decoding is slow

Decoding K tokens from large auto-regressive models takes K serial runs, which is slow.



Carnegie Mellon University

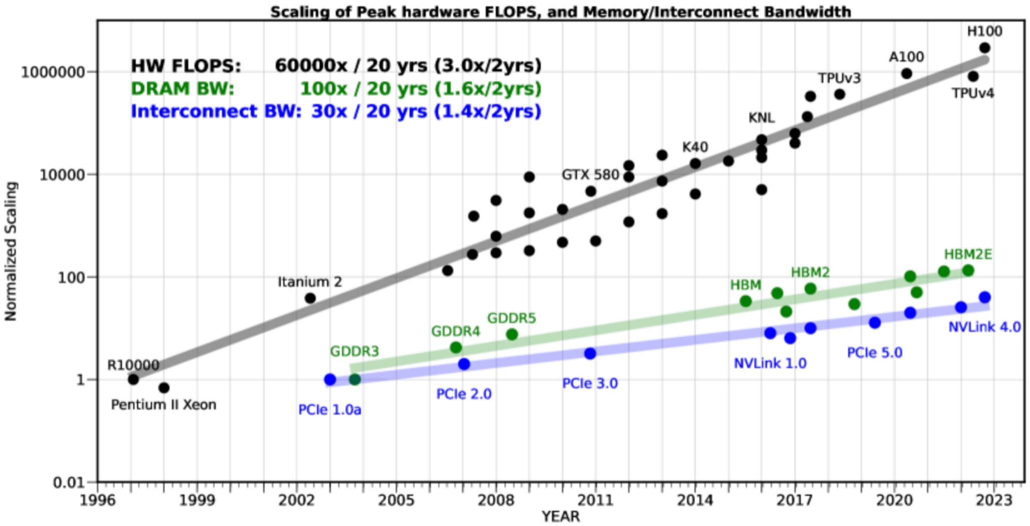
Challenges for Efficient Decoding

Memory bandwidth and communication are often the bottlenecks, not arithmetic operations.

- **Data Transfer**
  - LLMs are often run on multiple GPUs which requires high memory bandwidth to feed all the compute units with large amounts of data.
- **Device Communication**
  - Parallel processing involves multiple GPUs which requires constant communication between processors to synchronize the computation process across different parts of the model.

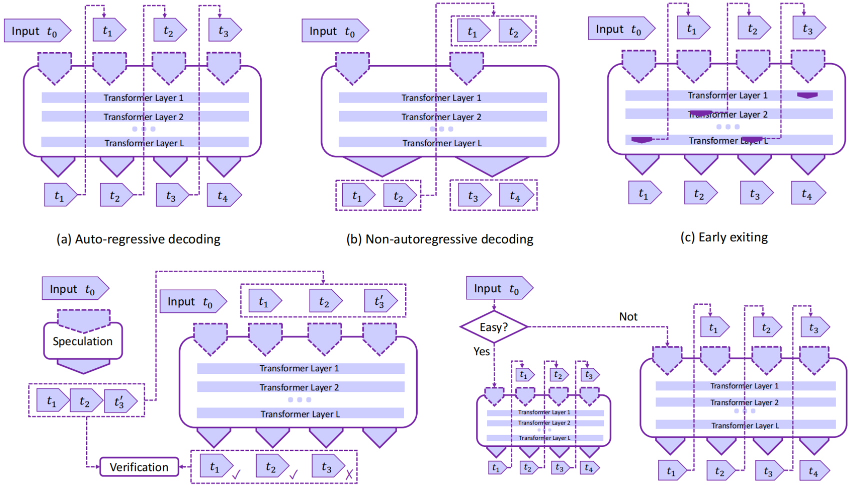
Challenges for Efficient Decoding

The bandwidth of different generations of interconnections and memory is increasing slowly, compared to Peak hardware FLOP.



Challenges for Efficient Decoding

Existing novel decoding algorithms either require architectural changes, retraining, or change the output distribution.



(a) Auto-regressive decoding

(b) Non-autoregressive decoding

(c) Early exiting

(d) Speculative decoding

(f) Cascade inference

Key Observations

- Some inference steps are "harder" and some are "easier" for generation.
- Additional computation resources might be available since arithmetic operations is not the bottleneck.
- Increasing concurrency can complement adaptive computation methods without changing the model architectures, without re-training the models, and maintaining an identical output distribution.

Contributions

- **Speculative Decoding**
  - Introduced speculative decoding which utilizes a draft model to allow several tokens to be decoded in parallel.
  - Do not require any change to the model architectures, training regimes and output distributions.

- **Speculative Sampling**
  - Developed a novel sampling method known as speculative sampling which maximizes the probability that tokens generated speculatively will be used.

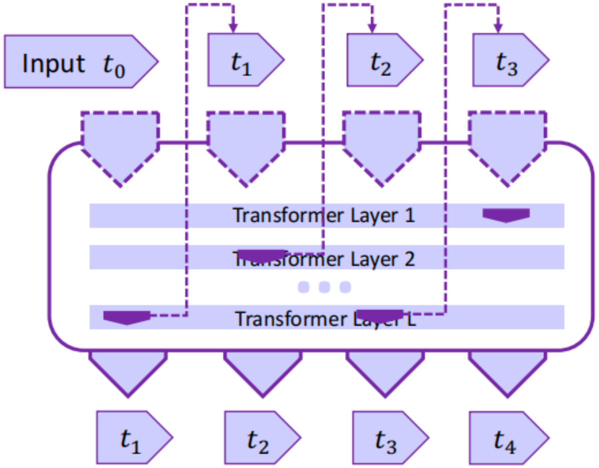
Outline

1. Motivation
2. **Related work**
3. Foundation
4. Algorithm
5. Experiment
6. Summary and Future direction

Adaptive computation methods

- Adapt the amount of computation to problem difficulty
- Learn when computation shortcuts can be taken
- Require architectural changes, retraining, or change the output distribution

Early Exit: the output of early model layers has the potential to infer the target distribution confidently.



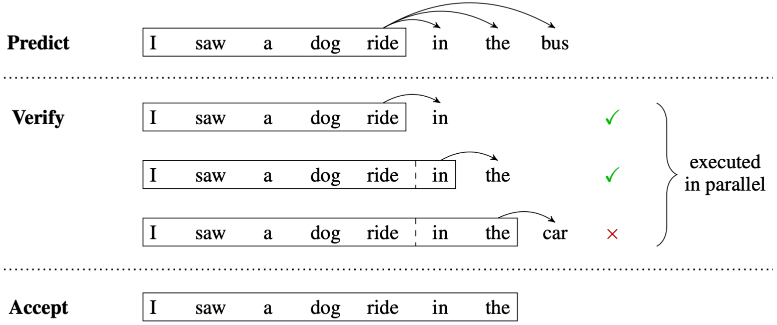
Carnegie Mellon University

Adaptive computation methods

- Inserts a single feedforward layer to the base LLM to make predictions for multiple future positions in parallel
- Only supports greedy decoding
- Requires additional training of a custom model

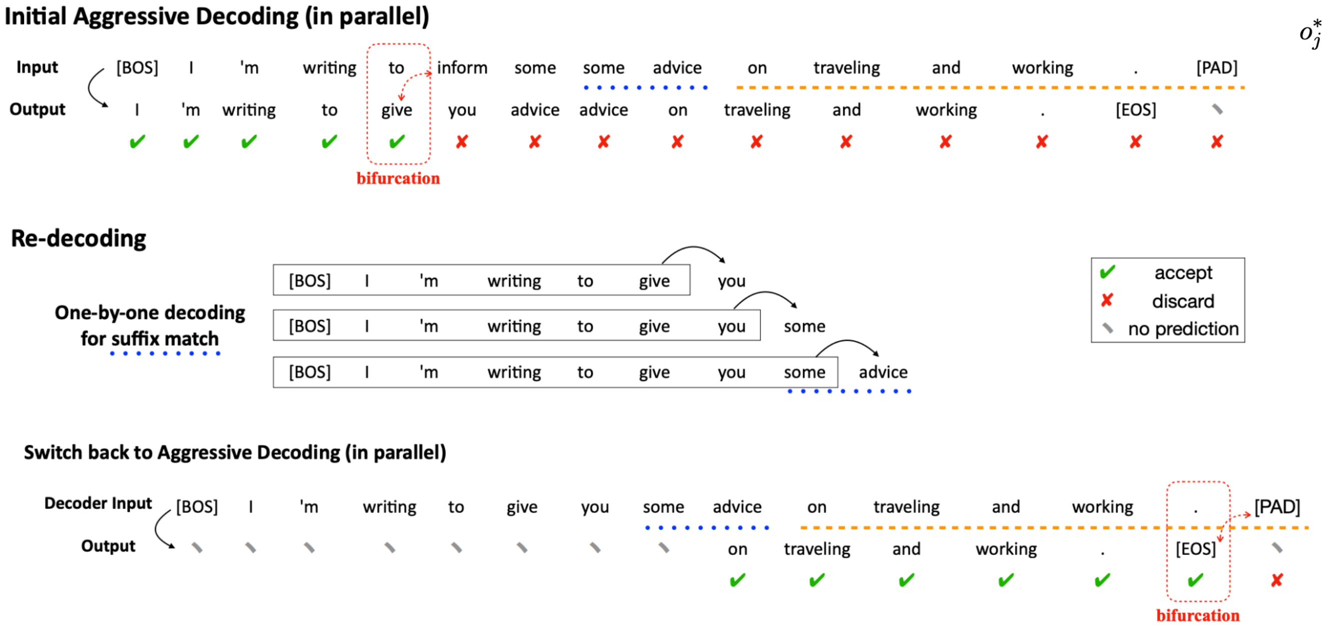
Three Step of BDP:

- Predict: base model p1, auxiliary models p2,..pk
- Verify: base model process a batch in parallel
- Accept: only accept the same prediction



Shallow Aggressive Decoding

- Propose aggressive decoding with a shallow decoder to execute in parallel
- Only supports copying the input to the output



Initial Aggressive Decoding (in parallel)

Input	[BOS]	I	'm	writing	to	inform	some	some	advice	on	traveling	and	working	.	[PAD]
Output	I	'm	writing	to	give	you	advice	advice	on	traveling	and	working	.	[EOS]	\
	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

bifurcation

Re-decoding

One-by-one decoding for suffix match

[BOS] I 'm writing to give → you

[BOS] I 'm writing to give you → some

[BOS] I 'm writing to give you some → advice

✓	accept
✗	discard
\	no prediction

Switch back to Aggressive Decoding (in parallel)

Decoder Input	[BOS]	I	'm	writing	to	give	you	some	advice	on	traveling	and	working	.	[PAD]
Output	\	\	\	\	\	\	\	\	on	traveling	and	working	.	[EOS]	\
									✓	✓	✓	✓	✓	✓	✗

bifurcation

$$
\begin{aligned}
o^*_{j+1} &= \arg\max_{o_{j+1}} \log P(o_{j+1} \mid \boldsymbol{o}_{\le j}, \boldsymbol{x}; \boldsymbol{\Phi}) \\
&= \arg\max_{o_{j+1}} \log P(o_{j+1} \mid \hat{\boldsymbol{o}}_{\le j}, \boldsymbol{x}; \boldsymbol{\Phi}) \\
&= \arg\max_{o_{j+1}} \log P(o_{j+1} \mid \boldsymbol{x}_{\le j}, \boldsymbol{x}; \boldsymbol{\Phi})
\end{aligned}
$$

Outline

1. Motivation
2. Related work
3. **Foundation**
4. Algorithm
5. Experiment
6. Summary and Future direction

Generation Is Sampling

Methods to generate new tokens, such as:

- Argmax
- Top-K
- Beam search

Techniques such as setting a temperature, can be viewed as sampling from the probabilistic distribution of the next token.

Outline

1. Motivation
2. Related work
3. Foundation
4. **Algorithm**
5. Experiment
6. Summary and Future direction

Algorithm Overview

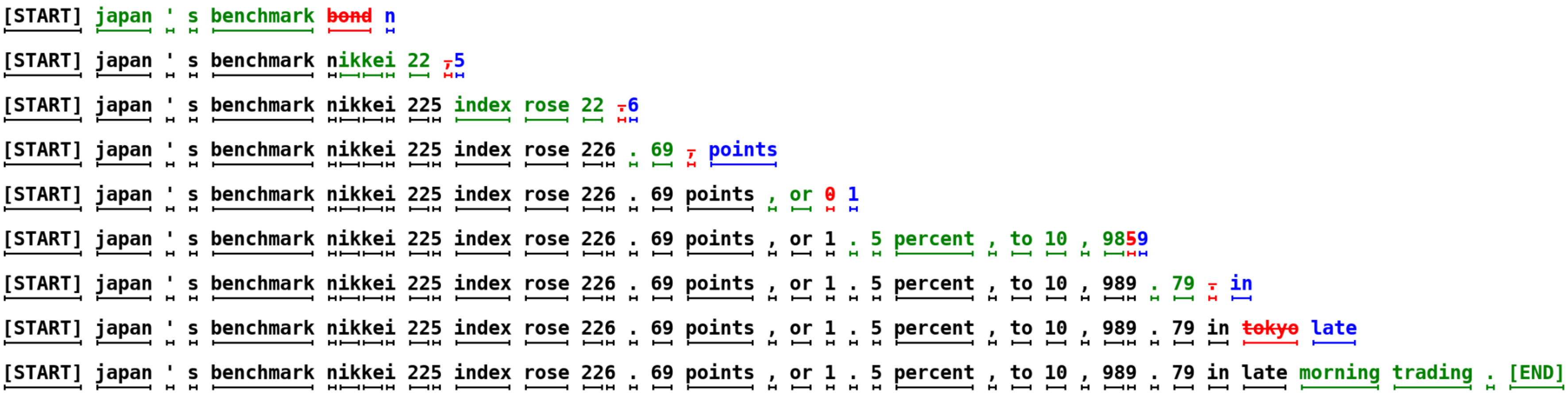
Hard Constraint: must sample from the same distribution

Core Ideas:

1. Use a more efficient model to make guesses about future tokens.
2. Use the target model to evaluate all guesses in parallel.
3. Accept guesses until the first guess rejected.
4. Fix the rejected guess using the result from 2.

Speculative Decoding Illustrated

- **green tokens** are the suggestions made by the approximation model
- **red** and **blue** tokens are the rejected suggestions and their corrections



[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 , 9859

[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]

Algorithm Details: Notations

Let M_p be the target model, M_q be the efficient model.

$p(x_t|x_{<t})$ is the distribution we get from the target model for prefix $x_{<t}$

$q(x_t|x_{<t})$ is the distribution from the efficient model for prefix $x_{<t}$

γ is the number of guesses we make about future tokens.

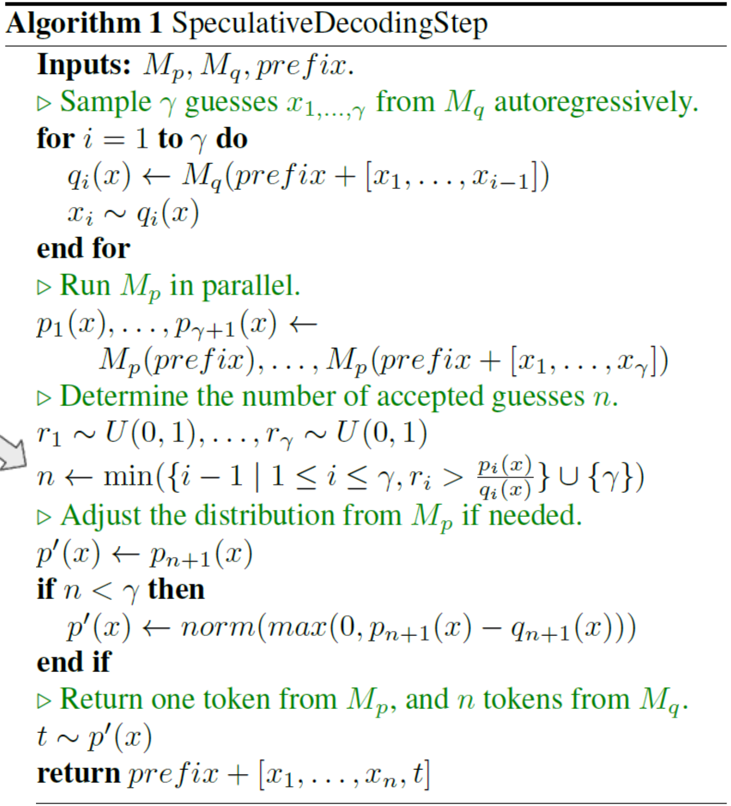
Algorithm Steps

It guarantees that x_t sampled this way follows $p(x_t|x_{<t})$

If $q(x) \leq p(x)$, accept;

otherwise, reject with prob $1 - \frac{p(x)}{q(x)}$

Note: the speedup comes from parallelism when calling M_p, as we assume the generation is memory-bounded.



Algorithm 1 SpeculativeDecodingStep

```
Inputs: $M_p, M_q, prefix$.
▷ Sample $\gamma$ guesses $x_{1,\ldots,\gamma}$ from $M_q$ autoregressively.
for $i = 1$ to $\gamma$ 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 $prefix + [x_1, \ldots, x_n, t]$
```

Carnegie Mellon University

More Notations

$\beta_{x<t}$ is the acceptance rate given $x_{<t}$

(we will be omitting the prefix notation)

$\alpha = E(\beta)$

$p'(x')$ is the adjusted distribution on the first rejected token

Why this process samples from the same distribution?

$$P(x = x') = P(guess\ accepted, x = x') + P(guess\ rejected, x = x')$$

where

$$P(guess\ accepted, x = x') = q(x') \min(1, \frac{p(x')}{q(x')}) = \min(q(x'), p(x'))$$

$$P(guess\ rejected, x = x') = (1 - \beta)p'(x') = p(x') - \min(q(x'), p(x'))$$

Also we have

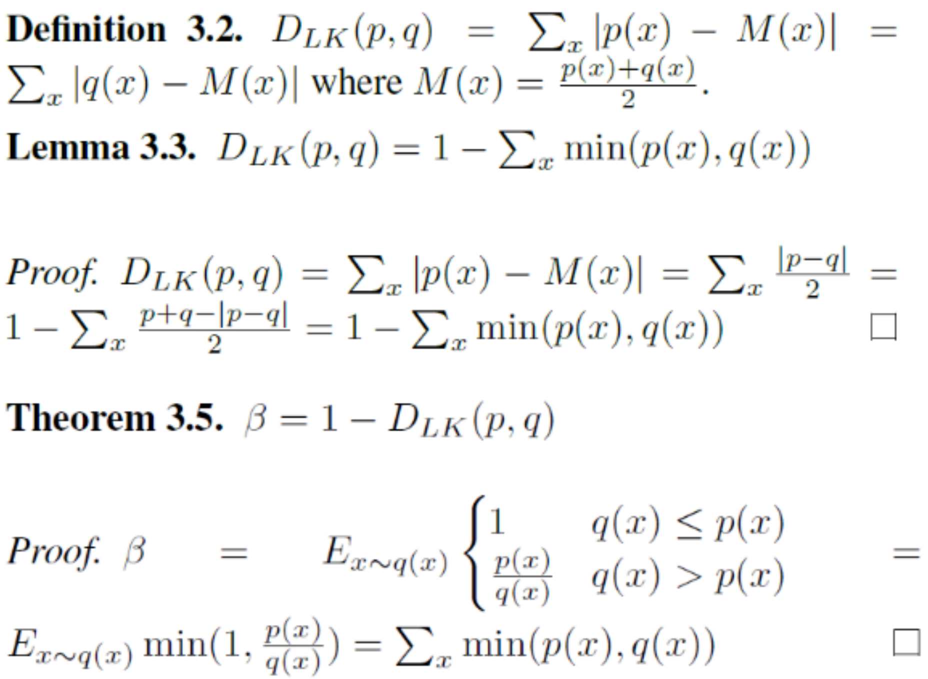
$$p'(x) = norm(max(0, p(x) - q(x))) = \frac{p(x)-min(q(x),p(x))}{\sum_{x'}(p(x')-min(q(x'),p(x')))} = \frac{p(x)-min(q(x),p(x))}{1-\beta}$$

Next Slide

Therefore,

$$P(x = x') = \min(p(x'), q(x')) + p(x') - \min(p(x'), q(x')) = p(x').$$

How to compute $\beta_{x<t}$



Definition 3.2. $D_{LK}(p,q) = \sum_x |p(x) - M(x)| = \sum_x |q(x) - M(x)|$ where $M(x) = \frac{p(x)+q(x)}{2}$.

Lemma 3.3. $D_{LK}(p,q) = 1 - \sum_x \min(p(x), q(x))$

Proof. $D_{LK}(p,q) = \sum_x |p(x) - M(x)| = \sum_x \frac{|p-q|}{2} = 1 - \sum_x \frac{p+q-|p-q|}{2} = 1 - \sum_x \min(p(x), q(x))$ □

Theorem 3.5. $\beta = 1 - D_{LK}(p,q)$

Proof. $\beta = E_{x\sim q(x)} \begin{cases} 1 & q(x) \leq p(x) \\ \frac{p(x)}{q(x)} & q(x) > p(x) \end{cases} = E_{x\sim q(x)} \min(1, \frac{p(x)}{q(x)}) = \sum_x \min(p(x), q(x))$ □

Theoretical Analysis

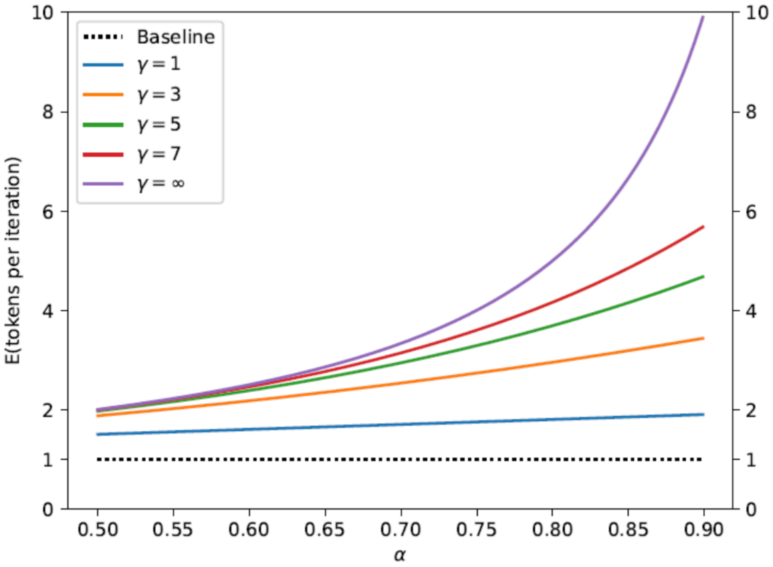
$\alpha = E(\beta)$ is a natural measure of how well M_q approximates M_p

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

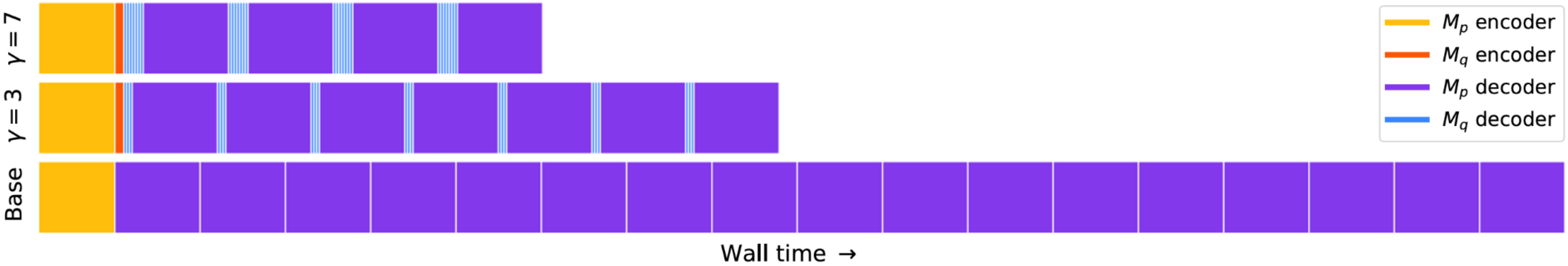
If the runtime of M_p is 1, each run of our algorithm takes $1 + c\gamma$

Total Walltime Improvement:

$$\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$$

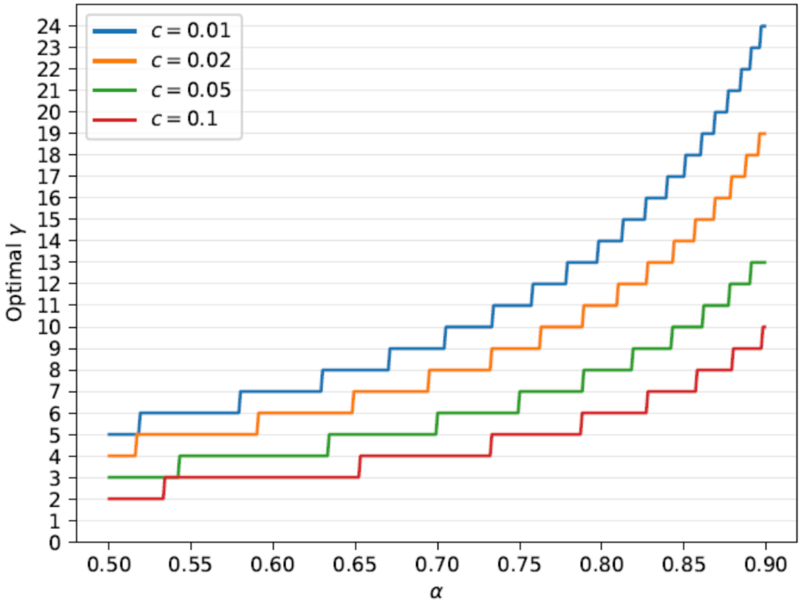


Walltime Visualized



Choosing γ

How many steps should we make guesses ahead of time?



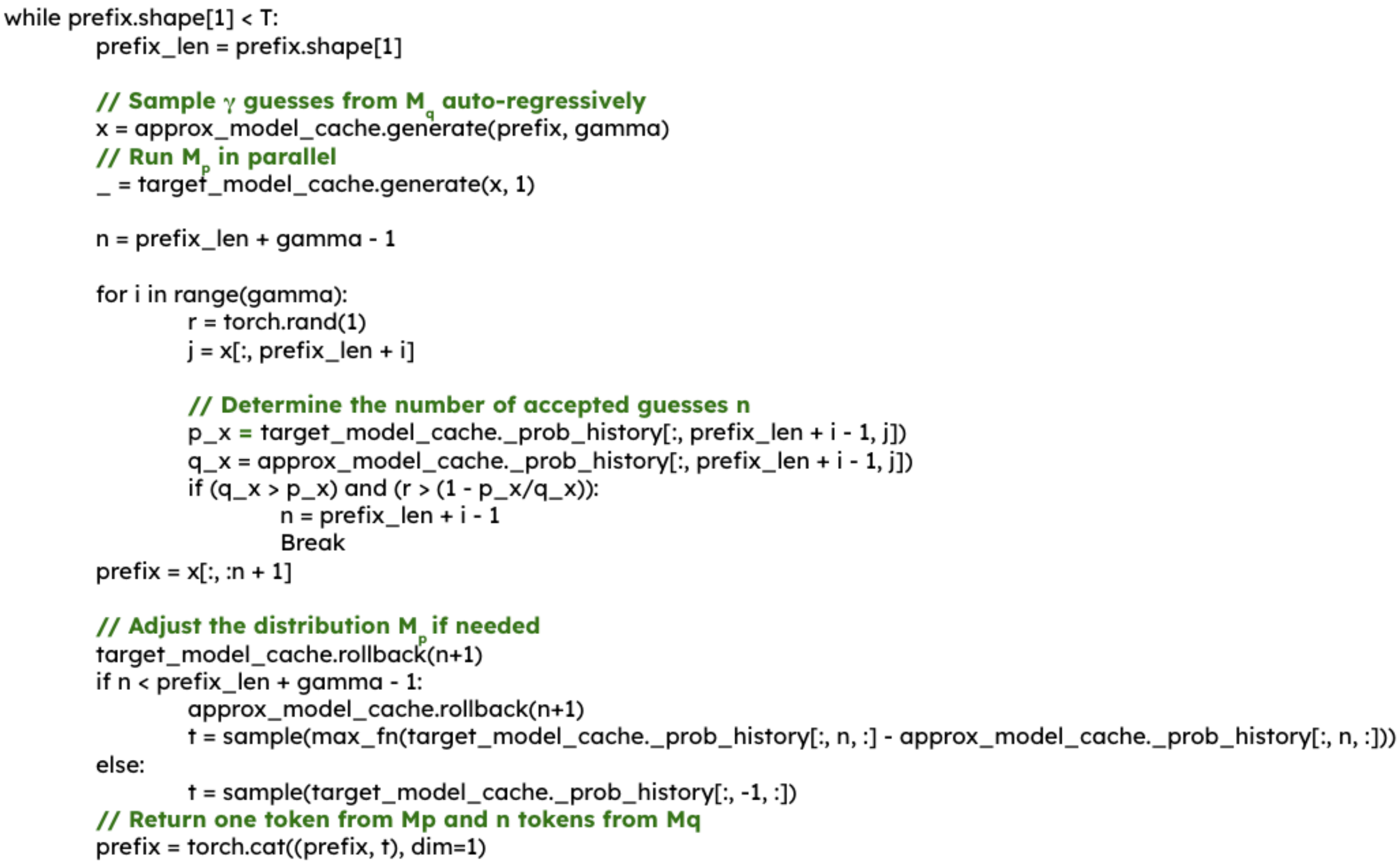
Walltime Improvement is in a closed form!

$$\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$$

Outline

1. Motivation
2. Related work
3. Foundation
4. Algorithm
5. **Experiment**
6. Summary and Future direction

Code



```
while prefix.shape[1] < T:
        prefix_len = prefix.shape[1]

        // Sample $\gamma$ guesses from $M_q$ auto-regressively
        x = approx_model_cache.generate(prefix, gamma)
        // Run $M_p$ in parallel
        _ = target_model_cache.generate(x, 1)

        n = prefix_len + gamma - 1

        for i in range(gamma):
                r = torch.rand(1)
                j = x[:, prefix_len + i]

                // Determine the number of accepted guesses n
                p_x = target_model_cache._prob_history[:, prefix_len + i - 1, j])
                q_x = approx_model_cache._prob_history[:, prefix_len + i - 1, j])
                if (q_x > p_x) and (r > (1 - p_x/q_x)):
                        n = prefix_len + i - 1
                        Break
        prefix = x[:, :n + 1]

        // Adjust the distribution $M_p$ if needed
        target_model_cache.rollback(n+1)
        if n < prefix_len + gamma - 1:
                approx_model_cache.rollback(n+1)
                t = sample(max_fn(target_model_cache._prob_history[:, n, :] - approx_model_cache._prob_history[:, n, :]))
        else:
                t = sample(target_model_cache._prob_history[:, -1, :])
        // Return one token from Mp and n tokens from Mq
        prefix = torch.cat((prefix, t), dim=1)
```

Experiments: Empirical Wall-time Improvement

- **Model Type:** Standard encoder-decoder T5 version
- **Tasks:**
    - (1) English to German translation fine tuned on WMT EnDe
    - (2) Text summarization fine tuned on CCN/DM
- **Mp Model:** T5-XXL (11B)
- **Mq Model:**
    - T5-large (800M)
    - T5-base (250M)
    - T5-small (77M)
- **Hardware:** Single TPU-v4 with batch_size=1

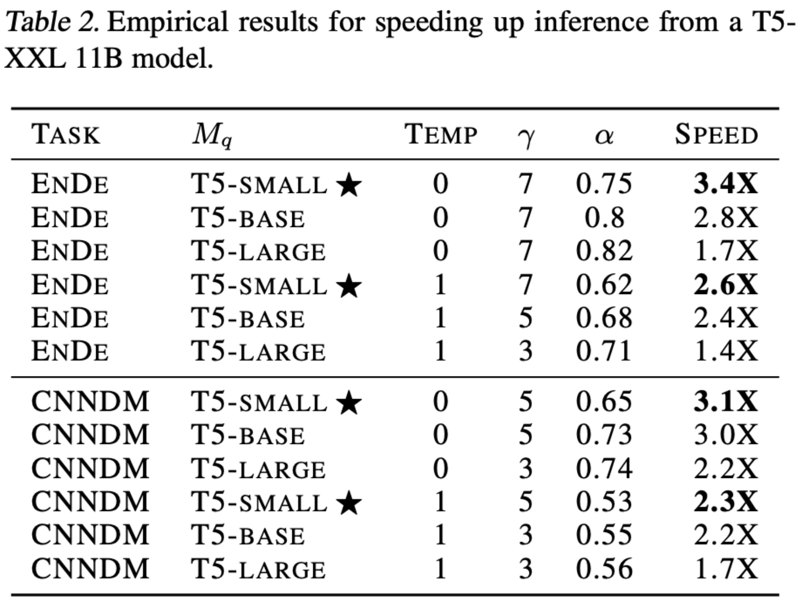
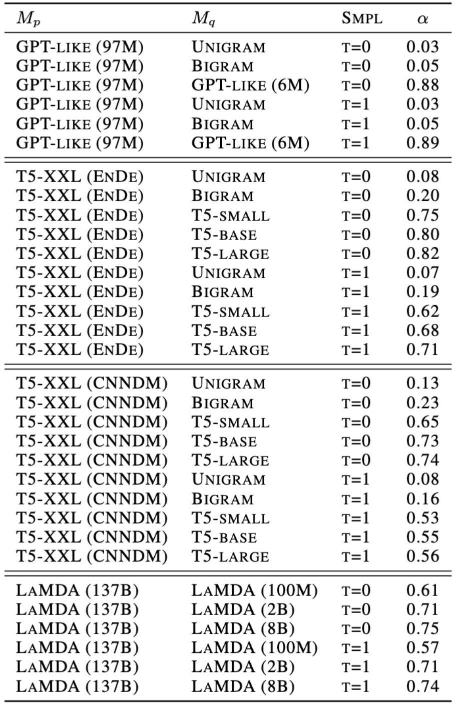


Table 2. Empirical results for speeding up inference from a T5-XXL 11B model.

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

Experiments: Empirical α Values

- **Introduce Model Types and Tasks:**
  - (1) Decoder-only LaMDA model on a dialogue task
  - (2) Decoder-only GPT-Like Transformer model on unconditional language generation, trained on lm1b
- **Mp Model:**
  - GPT-Like (97M)
  - T5-XXL (11B)
  - LaMDA (137B)
- **Compare with**
  - Unigram
  - Bigram
  - Same Model with Fewer Parameters



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

Summary

- Speculative decoding accelerates inference from autoregressive models without changing architectures, retraining, or output distributions
- Using speculative execution and a novel sampling method, we can make exact decoding from the large models faster
- Achieves 2X-3X speedups on T5-XXL

Future Directions

- **Application to Beam Search**
  - Given original beam width (w), perform beam search with approximation model and beam-width u > w for chosen steps
  - Use target model to check candidates in parallel
  - Accept guesses of approximate model as long as:

$$top_w(M_p) \subseteq top_u(M_q)$$

Future Directions

- **Oracle Factor**
  - Instead of picking single gamma, vary the value of gamma based on predicting the value of beta.
  - Assuming unbounded compute, enhanced wall-time improvement factor can be up to 60% higher than fixed gamma