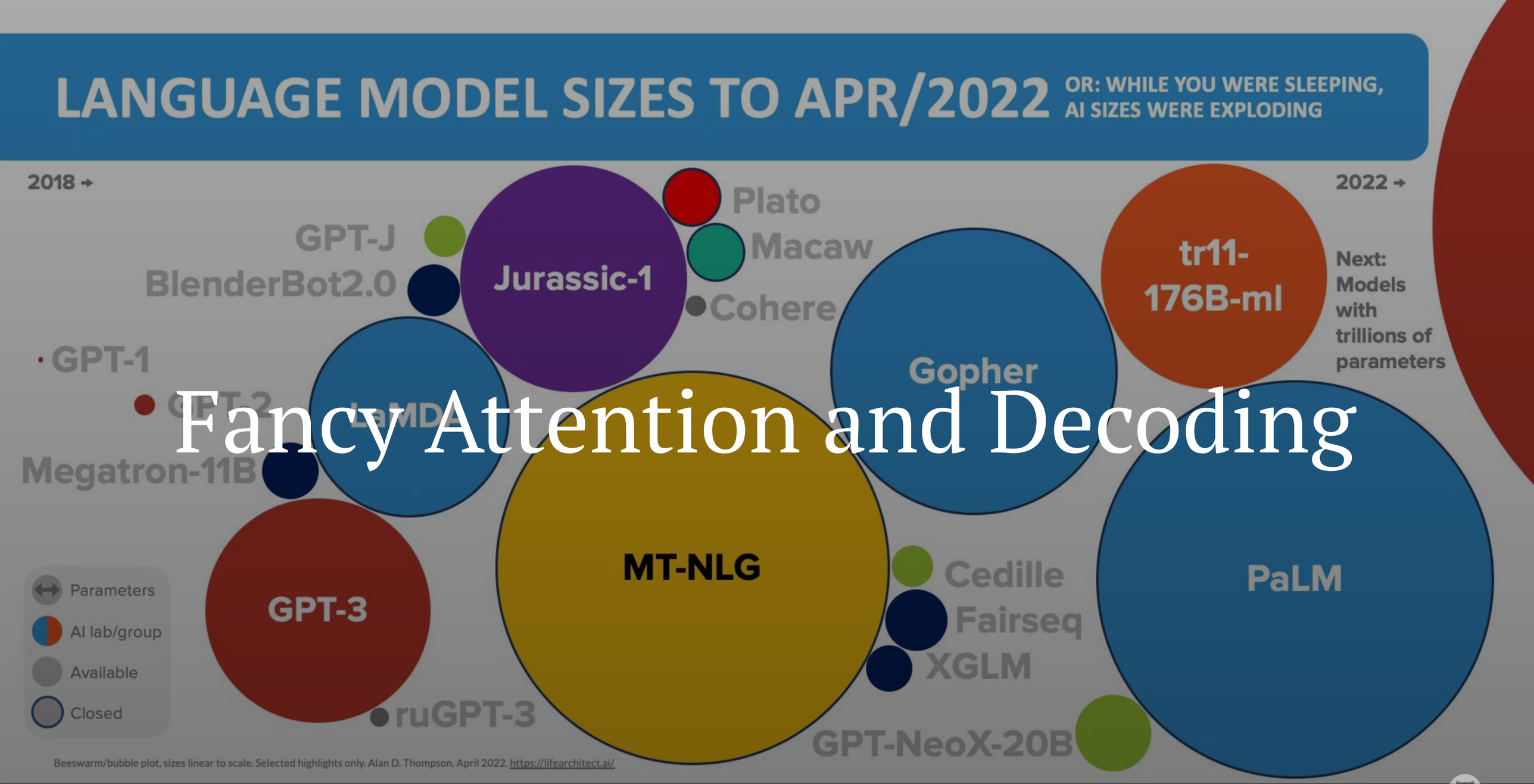
Fancy Attention and Decoding



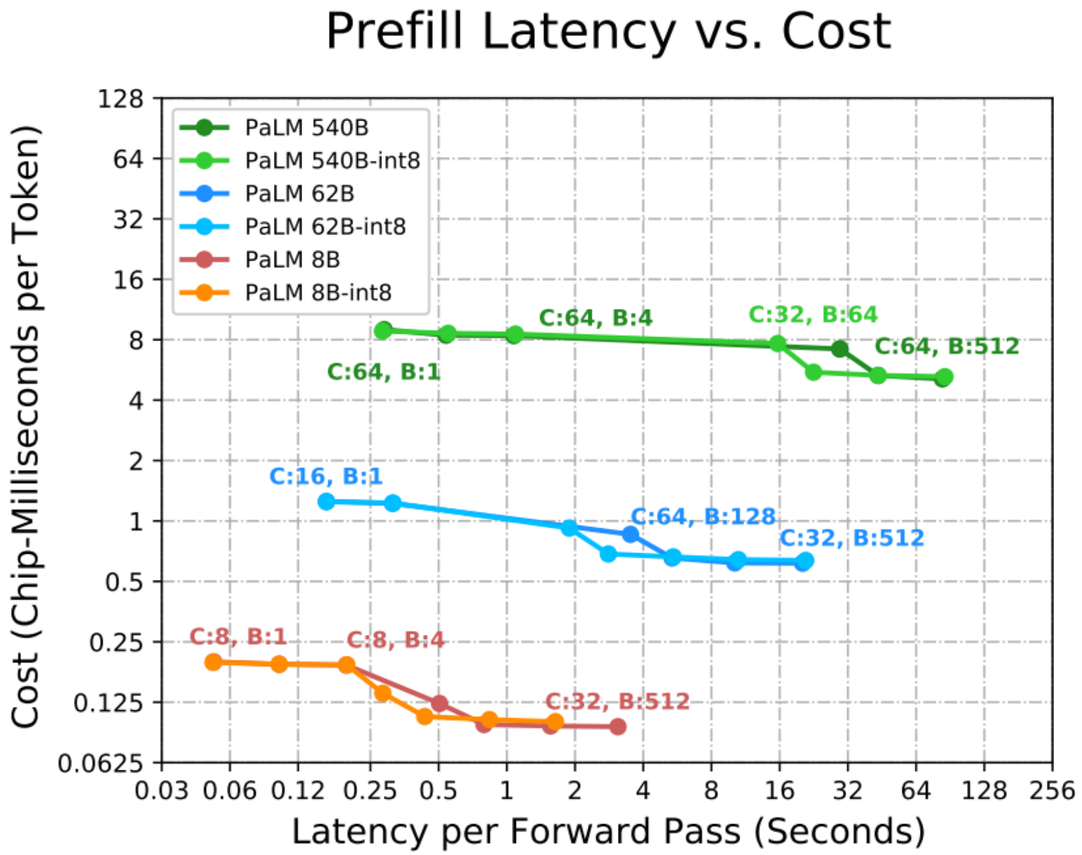
What is the real bottle-neck of LLM?

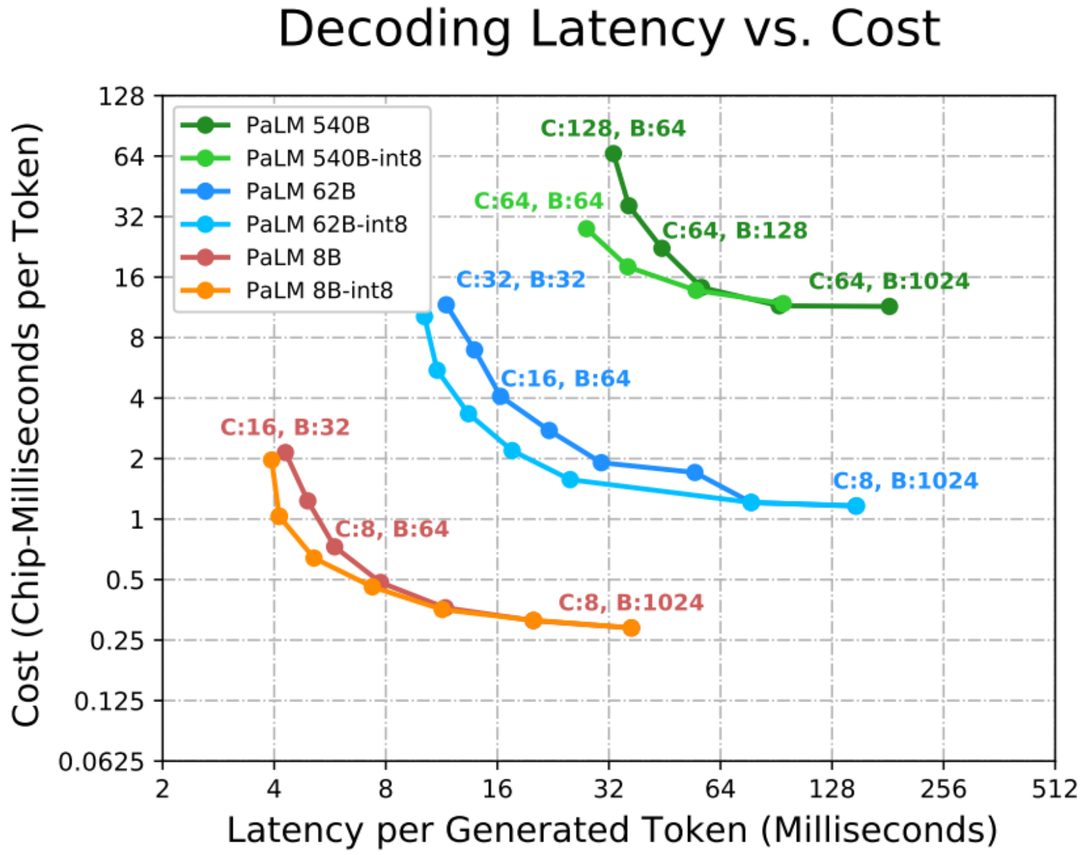
Run Llama-7B inference with llama.cpp

	Nvidia A100	Apple M2
computation	624 TOPS (90X)	7 TOPS
bandwidth	1935 GB/s(20X)	100 GB/s
speed	277 token/s (17X)	16 token/s

Memory bandwidth is the bottle-neck, not computation.

Encoding & Decoding





encode / prefill / prefix

decode / auto-regressive

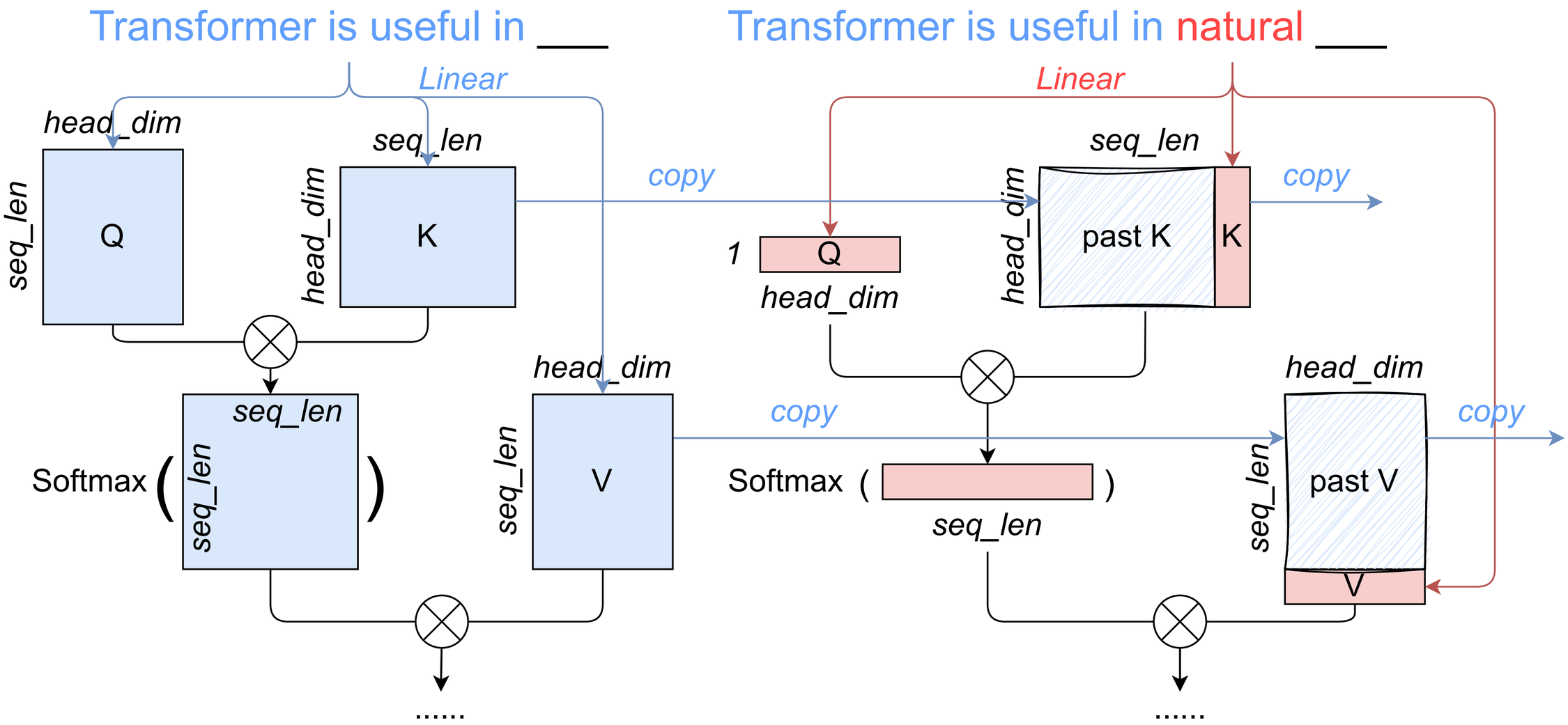
Decoding

> Transformer is useful in

prefix	decode
Transformer is useful in ___	natural
Transformer is useful in natural ___	language
Transformer is useful in natural language ___	processing

Notice something redundant?

KV cache


Transformer is useful in ___
Transformer is useful in natural ___
Linear
Linear
head_dim
seq_len
seq_len
head_dim
Q
K
copy
copy
1
Q
head_dim
seq_len
head_dim
past K
K
Softmax ( )
seq_len
seq_len
head_dim
V
copy
copy
Softmax ( )
seq_len
seq_len
past V
V
.....
.....

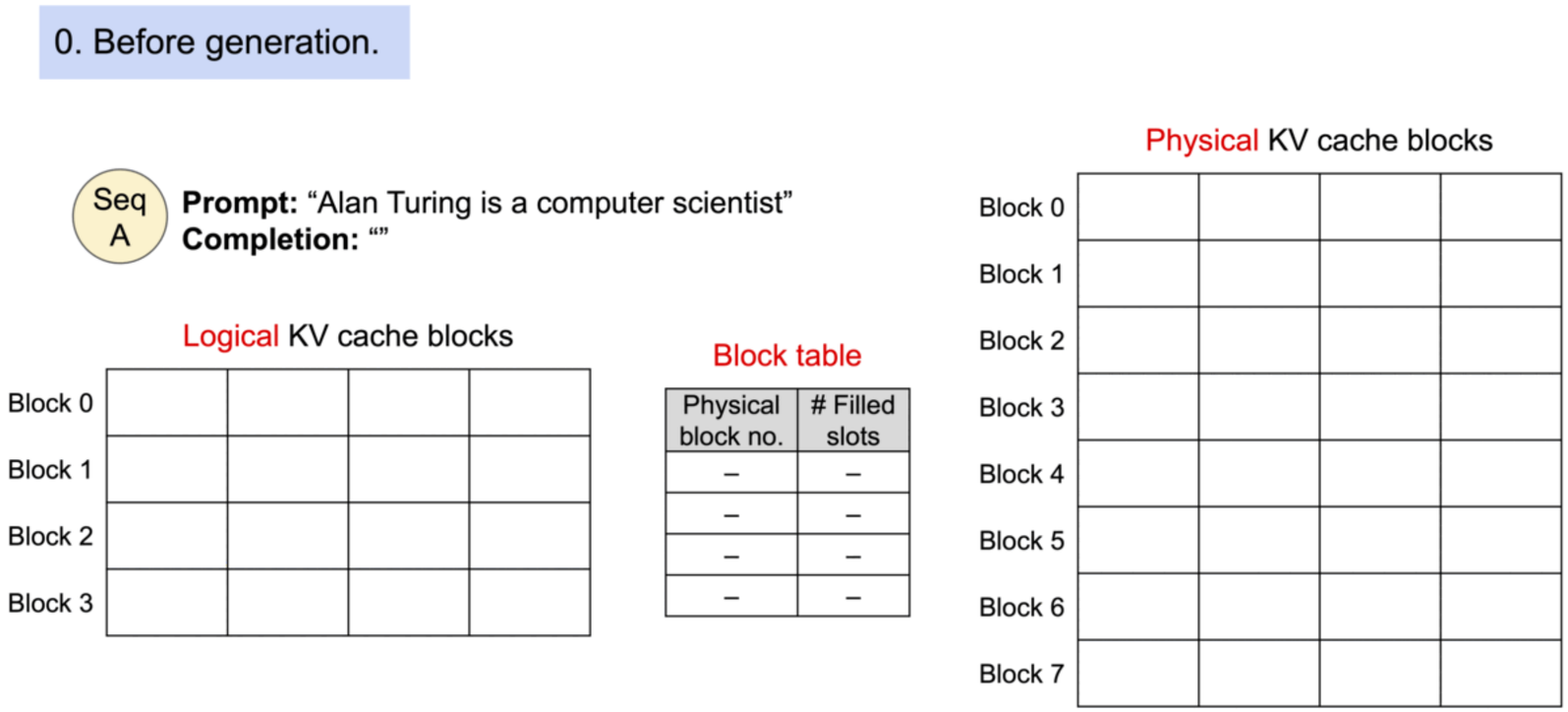
KV cache

1. KV cache is not *the cache*
2. Q (query) doesn’t need cache (always 1 tensor)

vLLM & PagedAttention

KV cache length is unpredictable, fragmentation and over-reservation cause 60% – 80% memory redundency

Adopt page table from OS to manage KV cache



vLLM throughput

FlashAttention

Attention computation can be divide-and-conquer

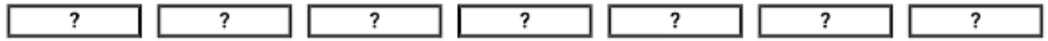
Divide Q K V, compute each block on SRAM

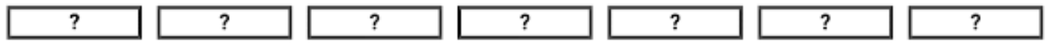
FlashAttention speed

Table 18: Forward pass runtime (ms) of various exact/approximate/sparse attention mechanisms by sequence length. Best in **bold**, second best underlined.

Attention Method	128	256	512	1024	2048	4096	8192	16384	32768	65536
PyTorch Attention	0.21	0.22	0.43	1.27	4.32	16.47	67.77	-	-	-
Megatron	0.24	0.26	0.42	1.33	4.28	-	-	-	-	-
Reformer	1.77	2.82	5.01	9.74	20.03	41.11	87.39	192.40	-	-
Local Attention	0.48	0.57	0.80	1.90	5.76	11.56	23.13	46.65	94.74	-
Linformer	0.46	0.36	0.45	**0.50**	1.09	2.09	4.01	7.90	15.70	35.40
Smyrf	1.94	1.96	3.01	5.69	11.26	22.23	44.21	88.22	-	-
LSformer	1.21	1.34	1.34	3.31	11.01	21.71	43.27	86.32	172.85	-
Block Sparse	0.96	1.04	1.66	2.16	5.41	16.15	-	-	-	-
Longformer	0.99	0.98	0.99	1.56	4.79	11.07	32.98	-	-	-
BigBird	0.96	1.02	1.02	1.48	5.05	11.59	34.16	-	-	-
FlashAttention	**0.08**	**0.09**	**0.18**	0.68	2.40	8.42	33.54	134.03	535.95	2147.05
Block-Sparse FlashAttention	0.56	0.52	0.63	0.65	**0.61**	**0.96**	**1.69**	**3.02**	**5.69**	**11.77**

Early Exiting (Again)





CALM

> Confident Adaptive Language Modeling. NeruIPS 2022

Most tokens can early exit with less than 4 layers!

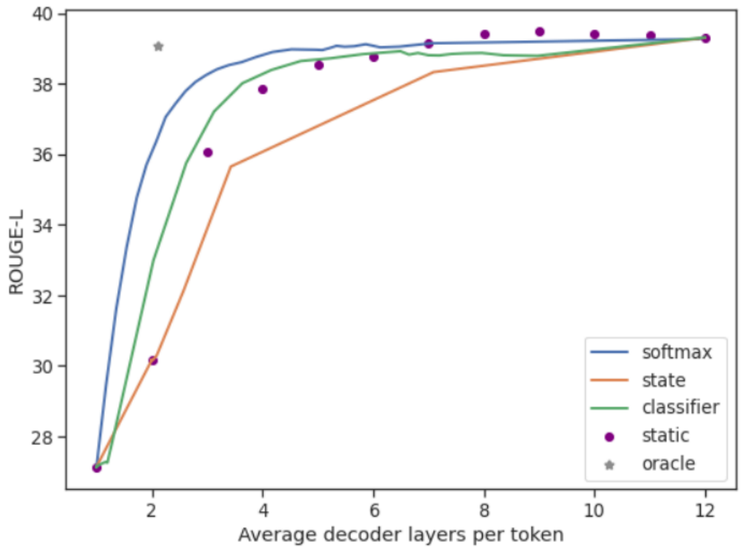
Z_{test} :	South Africa-born Grant Elliott hit match-winning 84 not out in semi-final . Black Caps reached first World Cup final with Elliott's penultimate ball six . Elliott, 36, had not played international cricket for 14 months when picked . Win is vindication for the attacking brand played under Brendon McCullum . New Zealand play the winner of the semi-final between Australia or India .
Y_{full} :	Grant Elliott hit a six to put New Zealand through to the World Cup final . The 36-year-old was born in South Africa but a naturalised Kiwi . Elliott will surely never play another innings like his 84 . New Zealand will take on either Australia or India in the final on Sunday .
$Y^{(1)}_{\text{early}}$:	__Grant__Elliott__hit__a__six__to__put__New__Zealand__through__to__the__World__Cup__final__.__Elliott__was__born__in__South__Africa__but__a__naturalised__Kiwi__.__The__36-year-old__will__surely__never__play__another__innings__like__his__unbeaten__84__. __New__Zealand__will__now__take__on__either__Australia__ or__India__in__the__final__on__Sunday__.<EOS>
$Y^{(2)}_{\text{early}}$:	__Grant__Elliott__hit__84__in__the__Black__Caps__chase__.__New__Zealand__reached__the__World__Cup__final__.__Elliott__was__born__in__South__Africa__but__a__naturalised__Kiwi__.__Elliott__will__surely__never__play__another__innings__like__his__unbeaten__84__.<EOS>

	$D(Y_{\text{early}}, Y_{\text{full}})$	$R_{\text{early}} - R_{\text{full}}$	Average layers	Speedup
$Y^{(1)}_{\text{early}}$	**0.02**	**0.01**	**2.1**	**X 2.9**
$Y^{(2)}_{\text{early}}$	**0.33**	**-0.3**	**1.9**	**X 3.6**

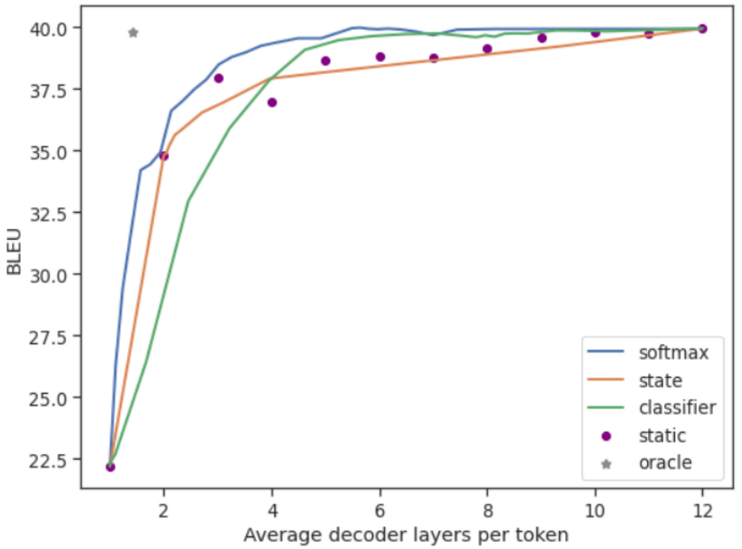
Exit layer — color mapping: 12345678

D and R are computed with ROUGE-L

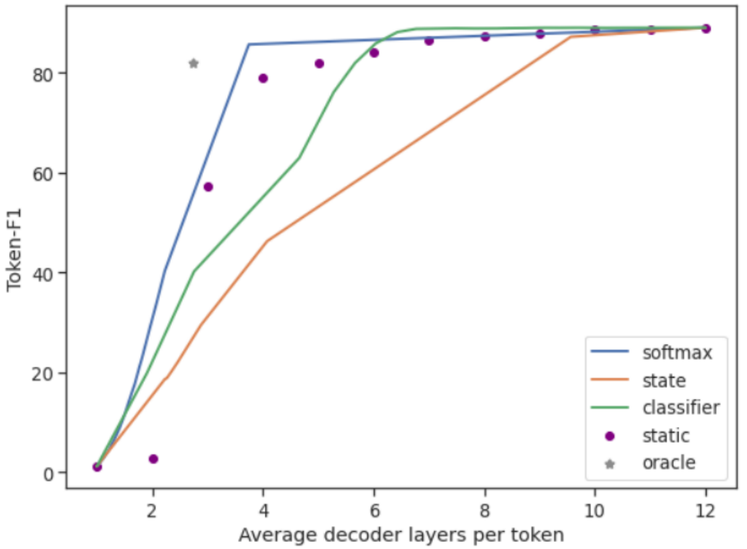
Exit point – Softmax still reigns



(a) CNN/DM



(b) WMT



(c) SQuAD

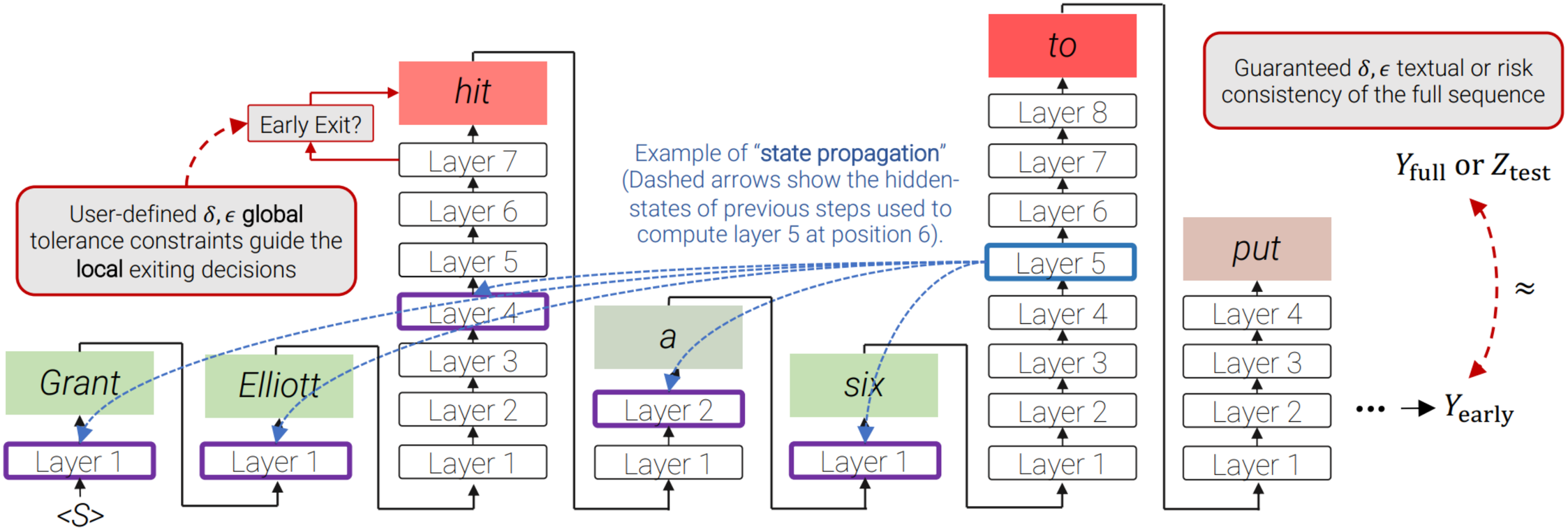
Early exit confidence measure:

Softmax > classifier > hidden states

But 🤔

Softmax is **expensive**! And,

NLG requires previous **KV cache**!



CALM's drawback

Method	Model Type	Generation	Token Level	Batching	KV-Caching	Full Attention	Controlled Comp. Cost
CALM	Enc-Dec	✓	✓	×	×	×	×
`SkipDecode`	Dec Only	✓	✓	✓	✓	✓	✓

Table 1: Comparison of CALM and `SkipDecode`. `SkipDecode` supports batching and KV caching for increasing inference efficiency with controlled computational budget.

SkipDecode

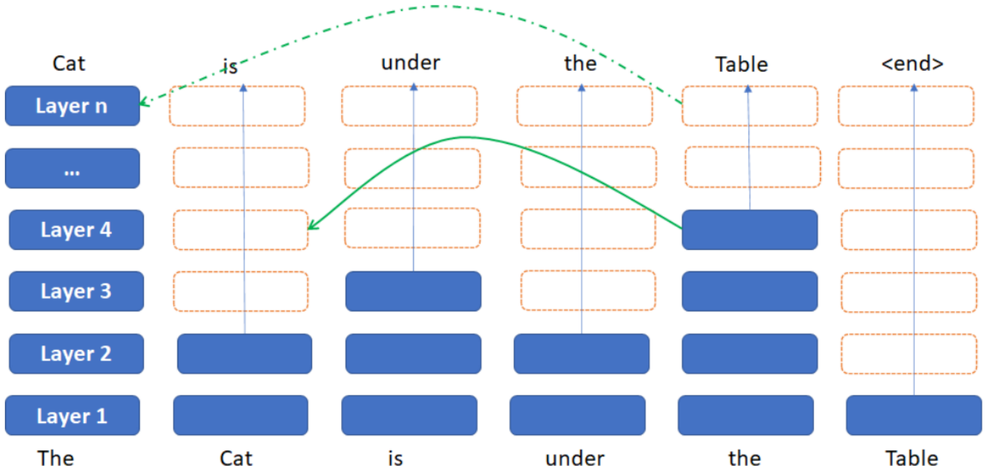
Confident Adaptive Language Modeling. NeruIPS 2022

Practical Blockers (Existing):

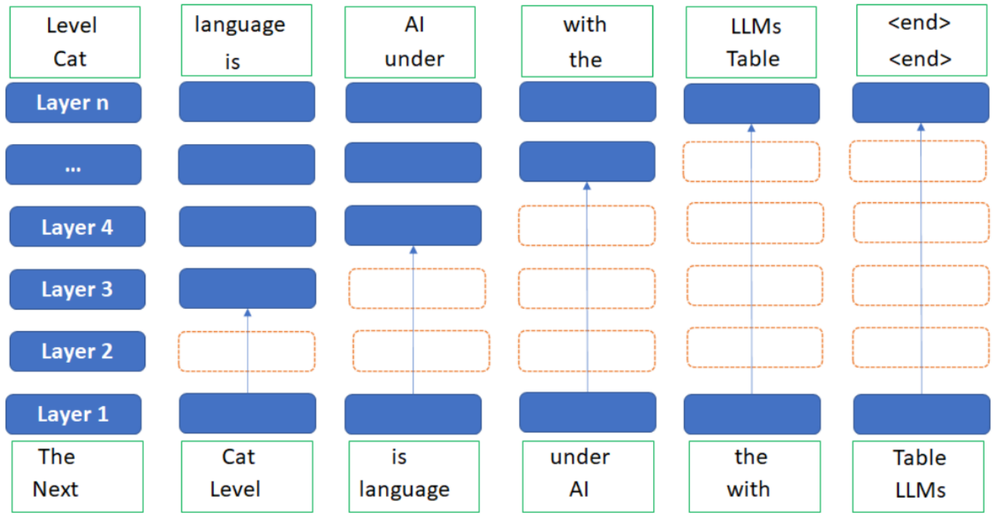
- **Batching:** Computational cost defined by the last exit token
- **KV Caching:** If next token exits later than previous one, we need to recompute KV values for previous tokens.
- **Computational Efficiency:** If next token exits earlier, it does not attend full computation of previous token.
- **Cost Uncertainty:** Worst case scenario (for a bad token exit, e.g., from last layer) equivalent to processing the whole network.

Solutions (Ours):

- **Batching:** Exit per position per batch (column-wise).
- **KV Caching:** Next column has to exit earlier than previous column. Leftwards tokens are more difficult to generate.
- **Computational Efficiency:** Spend most of computational budget on top layers. Implicitly attends the full computation of previous tokens.
- **Cost Uncertainty:** Static policy (no surprises), computation costs are predefined.

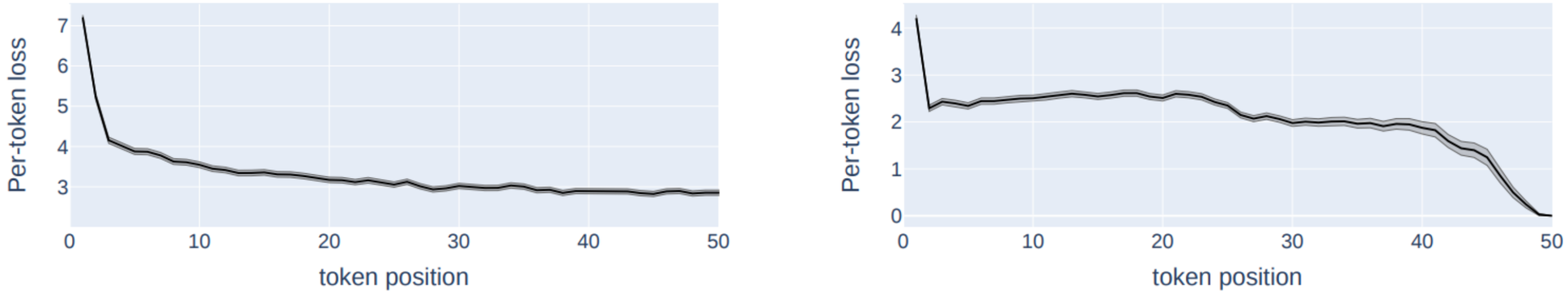


(a) Early Termination



(b) Skipping

Skipped tokens



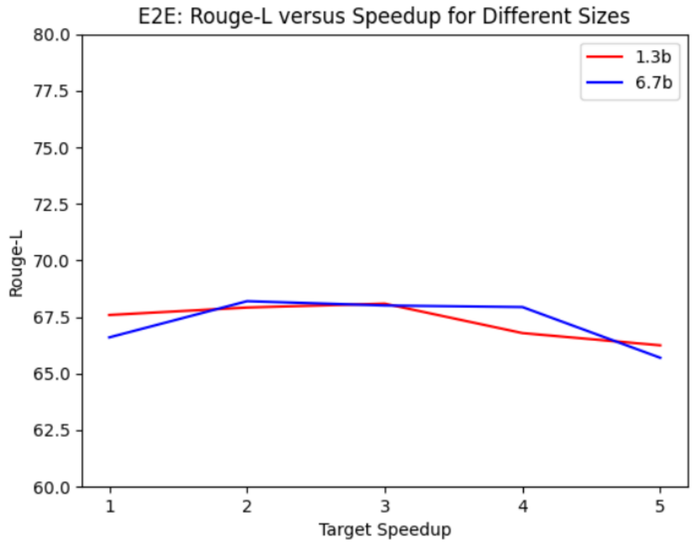
(a) **OPT-350m on OpenWebText**. Average loss per token position shows a strong monotonically-decreasing trend for general text.

(b) **OPT-350m (finetuned) on Reddit-TLDR**. Average loss per token position. Decreasing trend but with a different function.

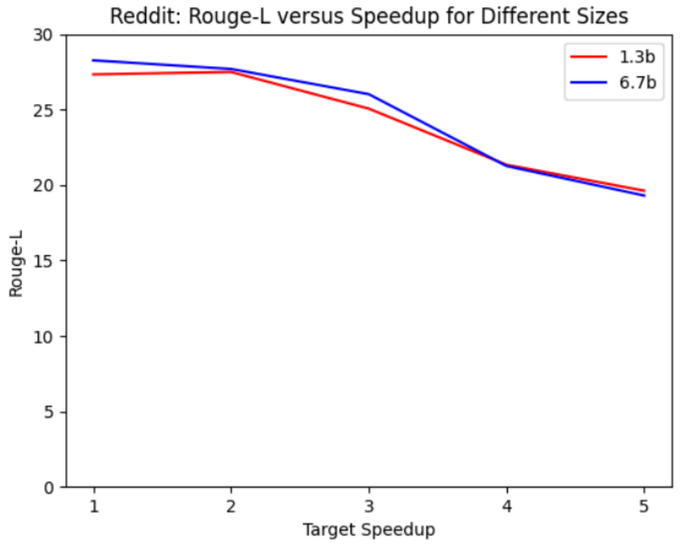
Figure 1: Average loss per token position (black) during the forward pass of OPT-350m model on a general and a task-specific dataset. Grey represents the 95% confidence interval on the mean.

Front tokens have large losses, which need more computation

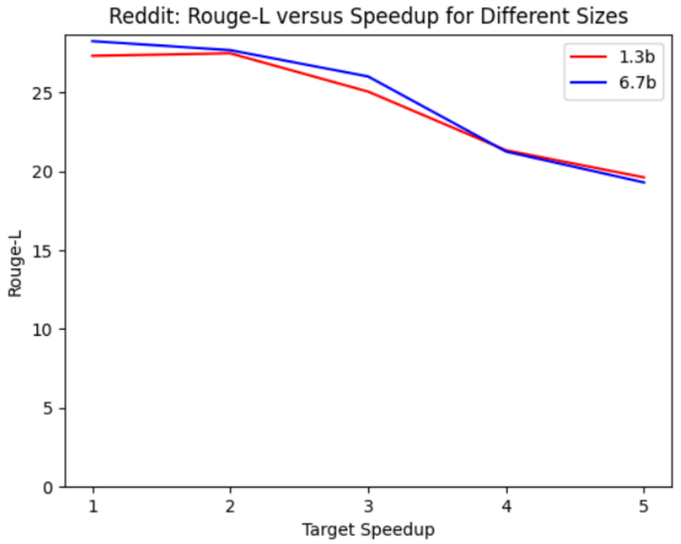
SkipDecode results



(a) E2E



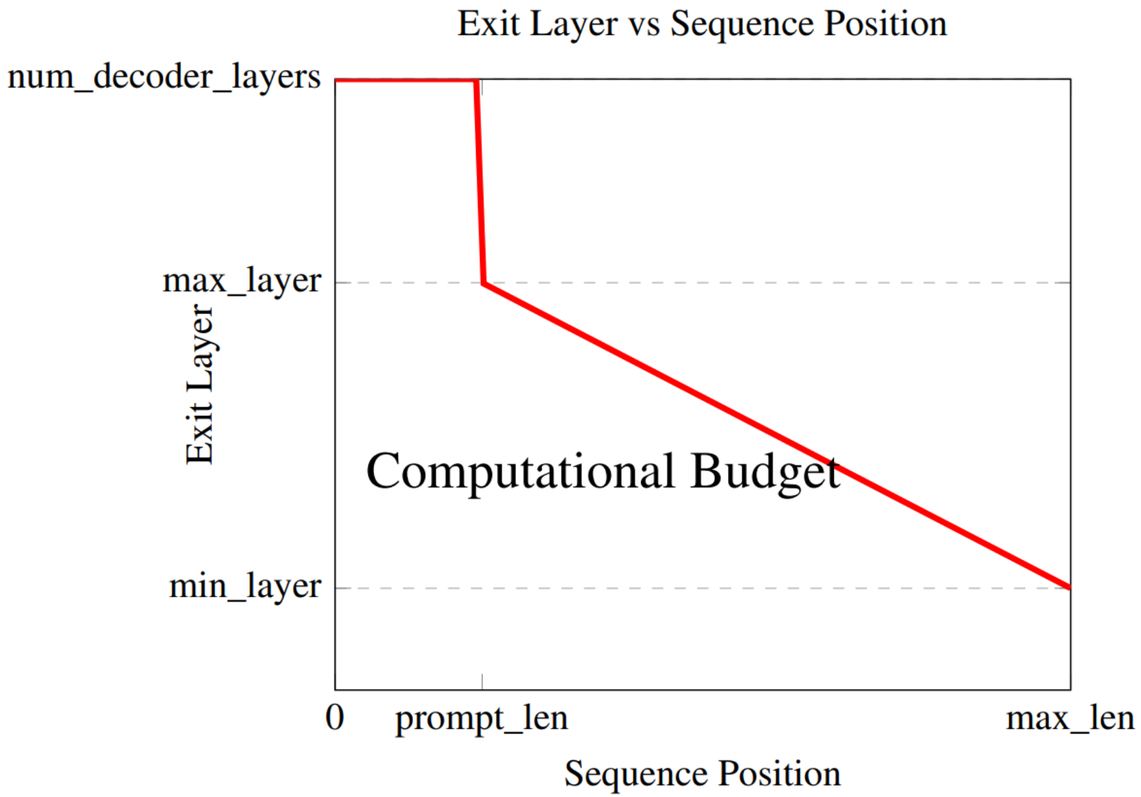
(b) Reddit



(c) CNN_DM

Works on LM, but fails on summarization, which is a common delima in efficient inference. 🤔

Fixed exit point



> SkipDecode pre-define fixed exit point of each token 🤔

Strength: **No Softmax**, batching √

Rethinking fixed exit point 🤔

SkipDecode:

> The token at front of each **SEQUENCE** is hard. 🤔

Recall CALM's result:

> The token at front of each **SENTENCE** is hard.

It is hard to find **SENTENCE** begining (dynamic) with batching

Review early exiting

`input_tensor` of shape `[bsz, len, hsize]`

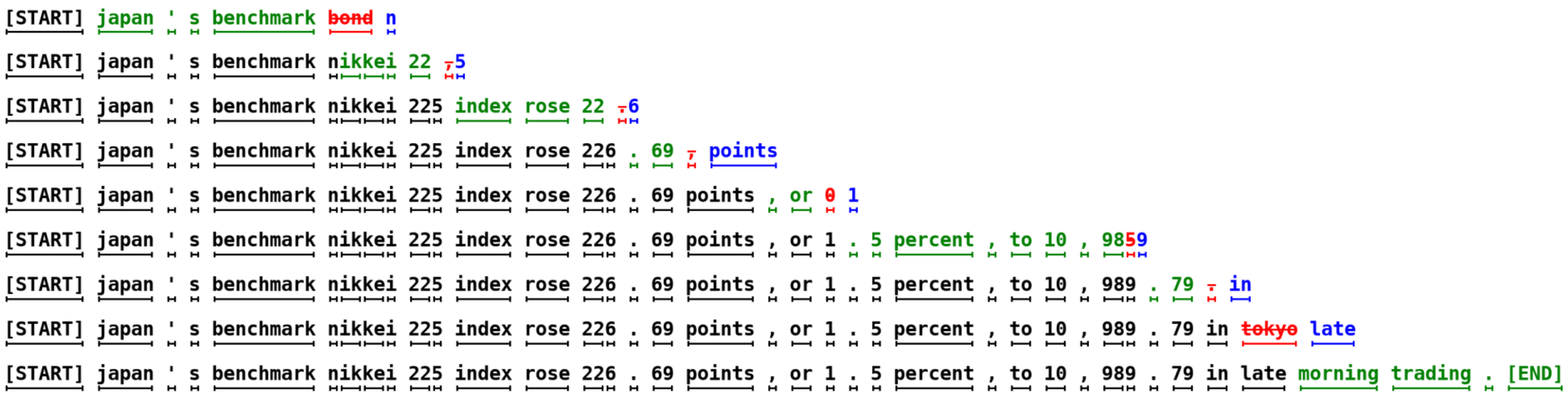
`model` with L layer and H hsize

- Early exiting reduces decode L only
- Pruning reduces H
- We try to use batching with larger `bsz`

What about `len` 🤔

Speculative Decoding

> Fast Inference from Transformers via Speculative Decoding. ICML 2023



- Using small model to generate multiple tokens
- Using LLM to verify the tokens, accept / reject. Then, and re-generate rejected token or generate the last token

Algorithm

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

Small model

Generate γ tokens **autoregressively**

▷ 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

LLM

Generate $\gamma + 1$ tokens **in parallel** (same as training)

▷ Run M_p in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
$M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$

Validate γ tokens, reject **once** LLM don't think the guess is good $p_i(x) < q_i(x) \rightarrow r_i > p_i(x)/q_i(x)$

▷ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$

Regenerate last token or rejected token

▷ Adjust the distribution from M_p if needed

Output

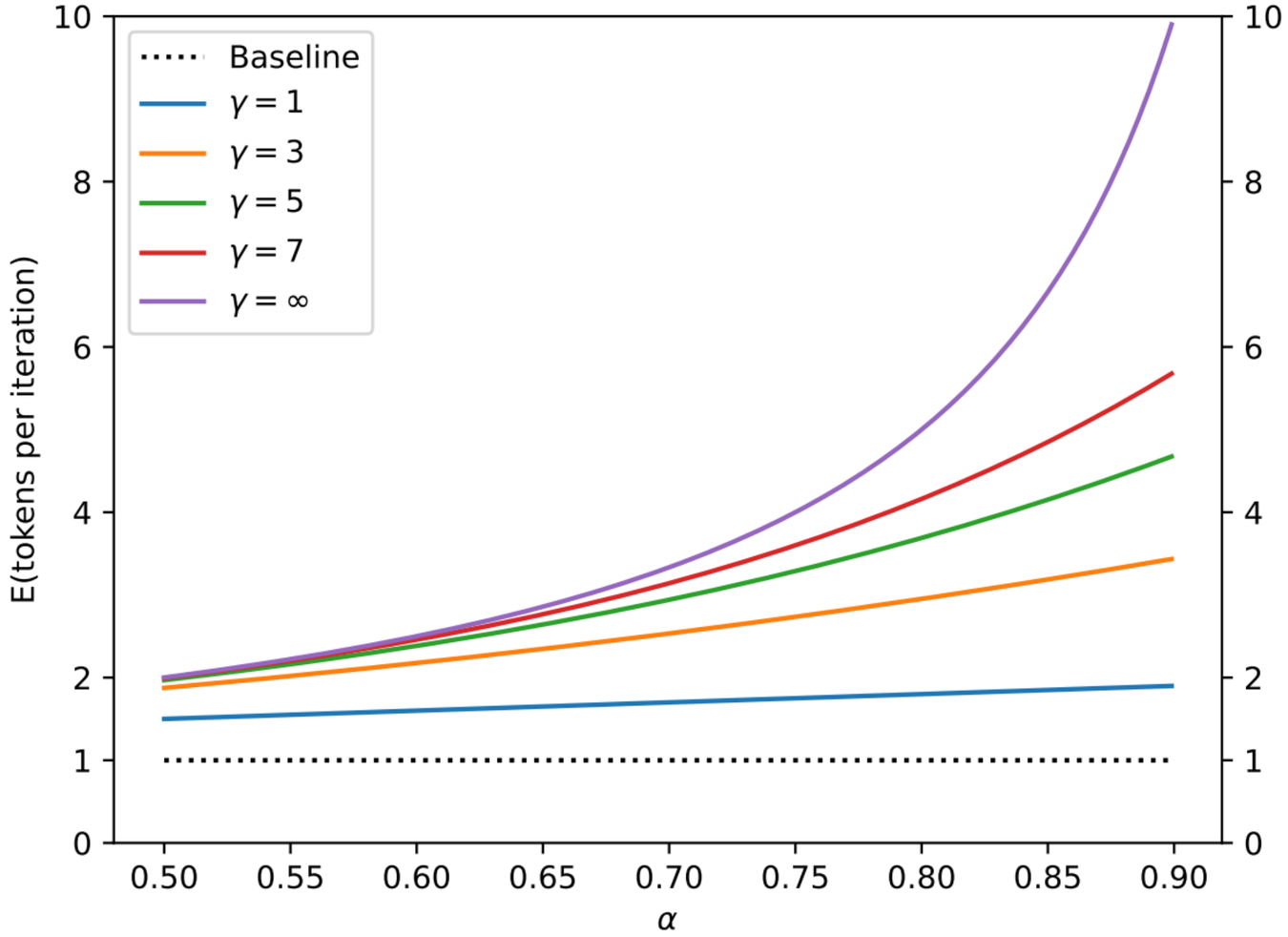
▷ Return one token from M_p, and n tokens from M_q.

$t \sim p'(x)$

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

Model Alignment and Speedup

Expected generated tokens in one run $\frac{1-\alpha^{\gamma+1}}{1-\alpha}$



Model Alignment Evaluation

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

Even **N-gram** can accelerate LLM generation 

Speculative Decoding Speedup

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

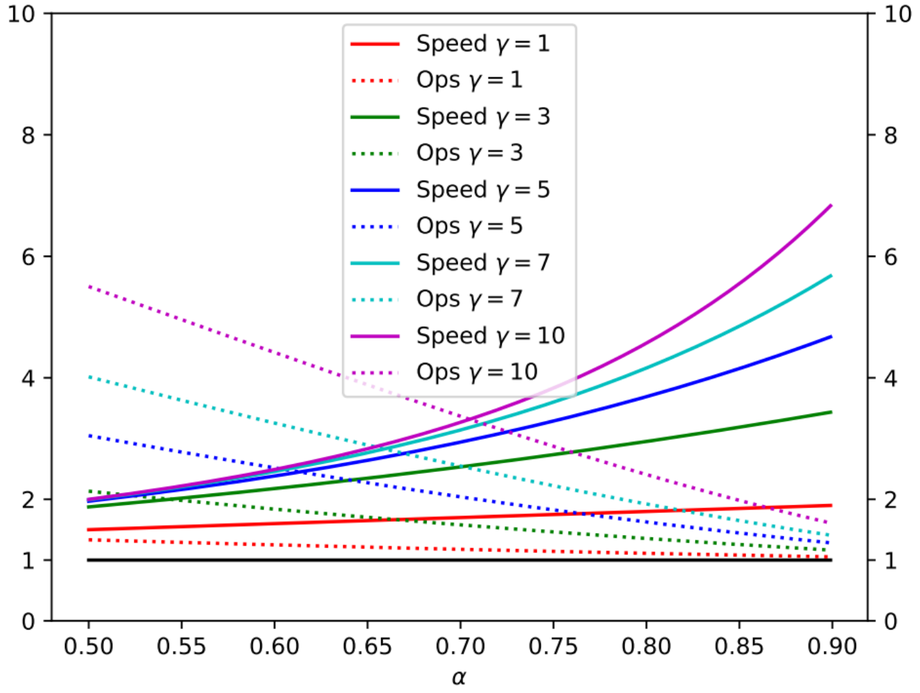


Figure 4. The speedup factor and the increase in number of arithmetic operations as a function of α for various values of γ.

2-3X speedup with a small model 😲

Strength and weakness

Strength

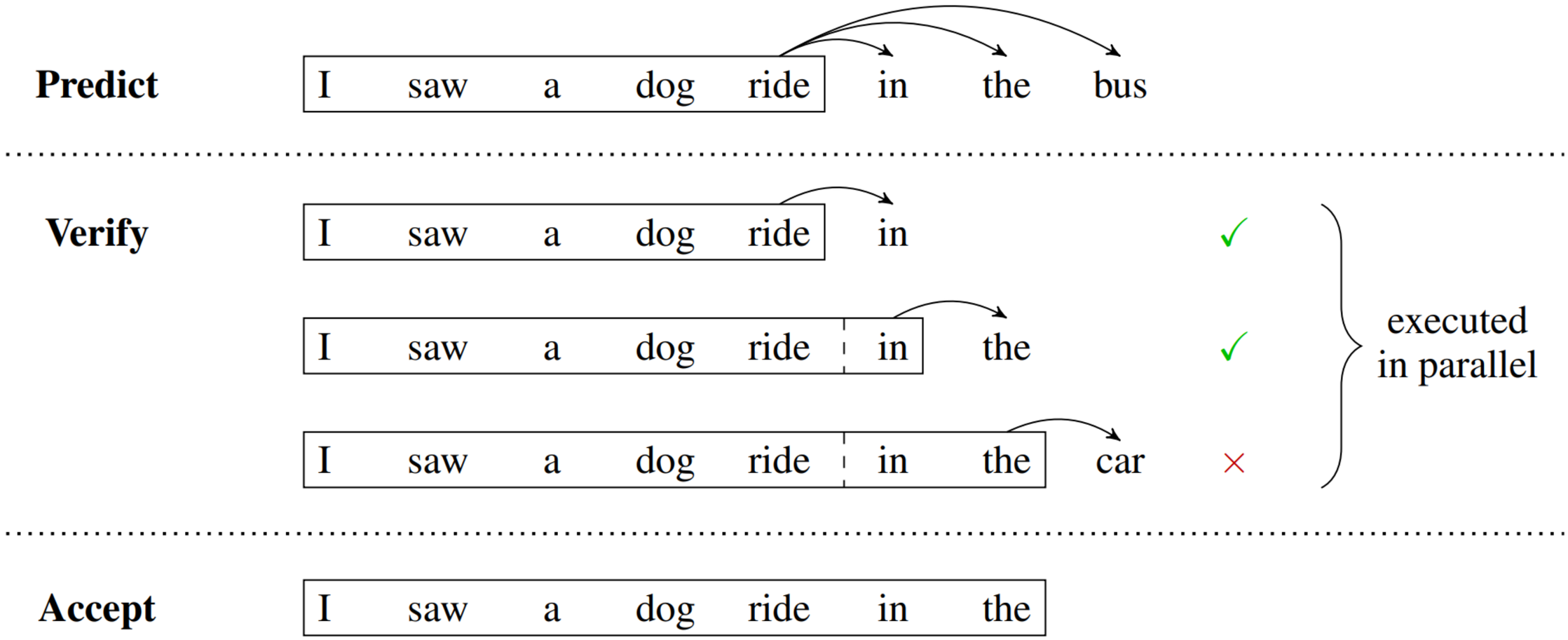
- Don’t need to change **any** of the model!
- **Always** has a fallback accuracy to LLM!

Weakness

- Need two (or more) models, much more **memory usage**
- Small model need to be **aligned** with LLM or the efficiency falls back

Block-wise generation

> Blockwise parallel decoding for deep autoregressive models. NeruIPS 2018



Resources

- 大模型推理性能优化之KV Cache解读
- NLP（十七）：从 FlashAttention 到 PagedAttention, 如何进一步优化 Attention 性能
- FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness
- vLLM. Easy, fast, and cheap LLM serving for everyone
- Accelerating text generation with Confident Adaptive Language Modeling (CALM)
- SkipDecode: Autoregressive Skip Decoding with Batching and Caching for Efficient LLM Inference
- Fast Inference from Transformers via Speculative Decoding
- Efficient Systems for Foundation Models ES-FoMo@ICML2023 talks