UVA CS 4774: Machine Learning

S3: Lecture 20: LLM Inference: Decoding and Efficient Inference

Dr. Yanjun Qi

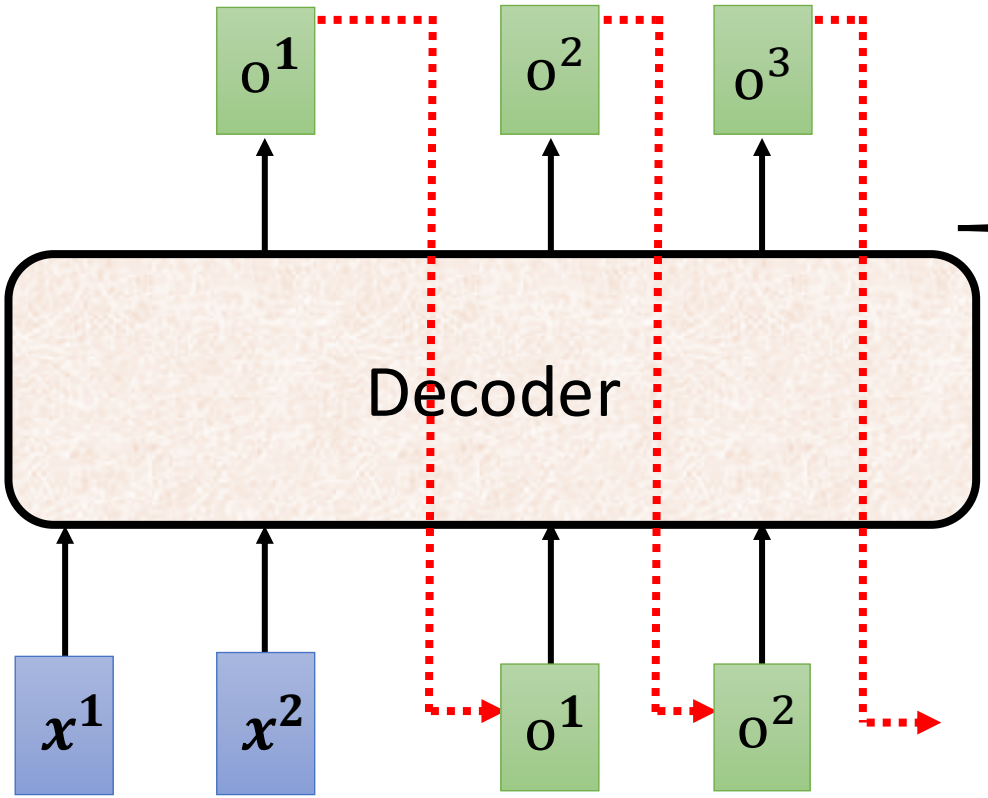
University of Virginia
Department of Computer Science

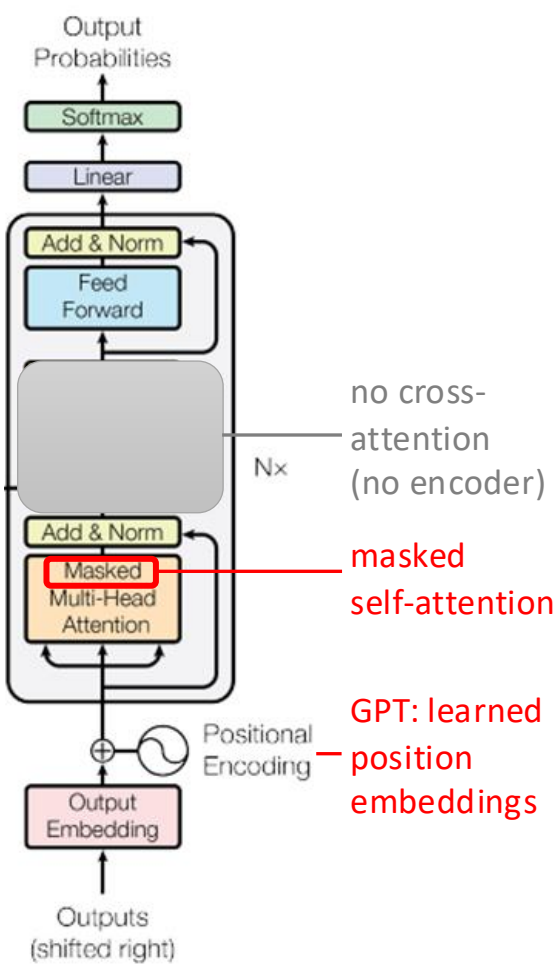
Recap : Last Session

- Training stages (encoder-only, decoder-only)
- Pretraining tactics (tokens, next-token loss, Adam)
- Pretrained GPT models (GPT-1/2/3)
- **Post-training (SFT, aligning a GPT: RLHF, DPO, RLVR)**

Transformer's Decoder

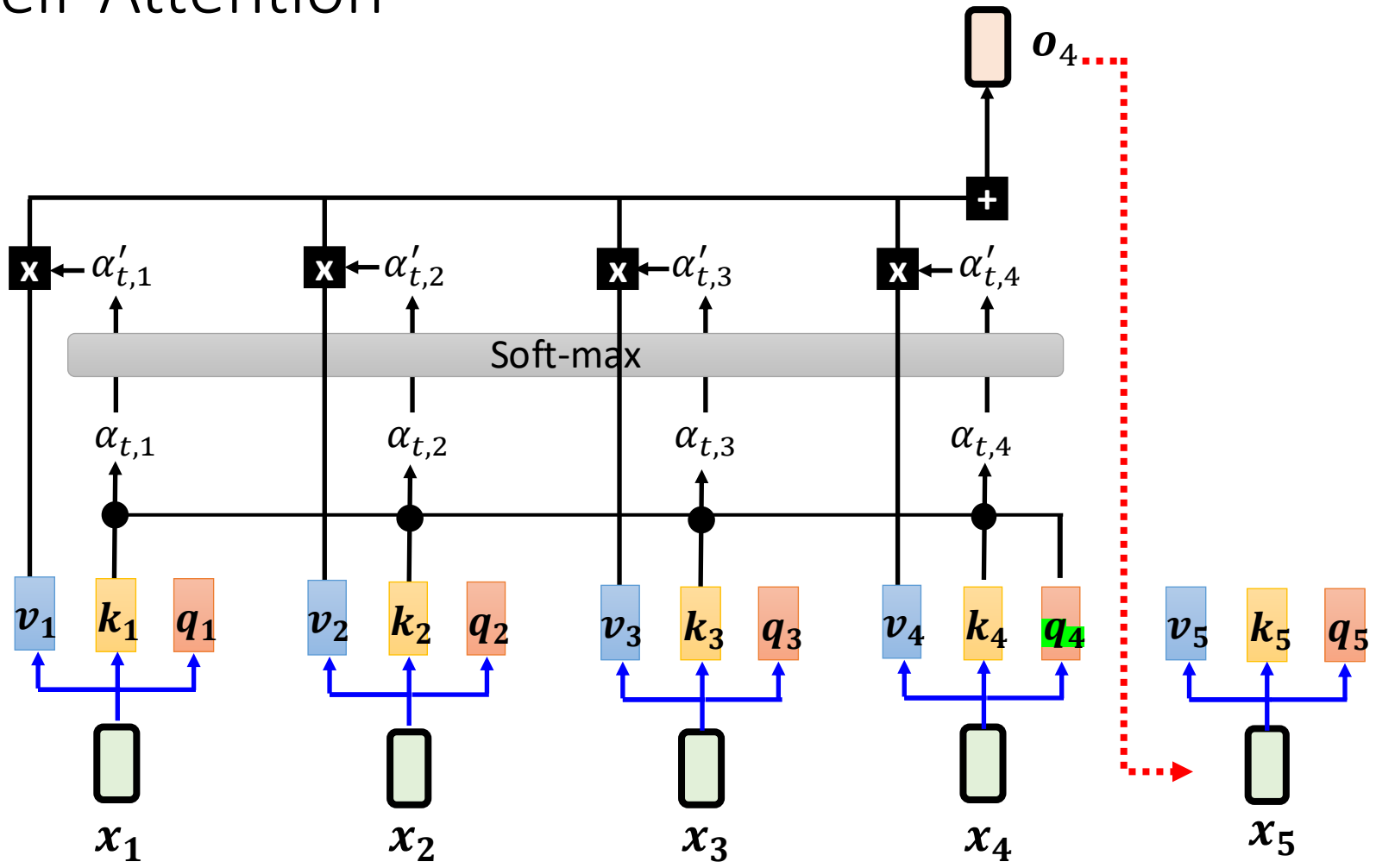
Decoder-only





credit: Prof. Hung-yi Lee,

Self-Attention



credit: Prof. Hung-yi Lee,

Training LLMs – Resources:

Web data



"The cat sat on the mat!"

"The" → "cat"

"The cat" → "sat"

.....

Labeled data



Q: Earth's tallest peak?

A: Everest .....

.....

User feedback



Q: Teach me to make a gun

A1: Sure, ...

A2: I can't teach you that



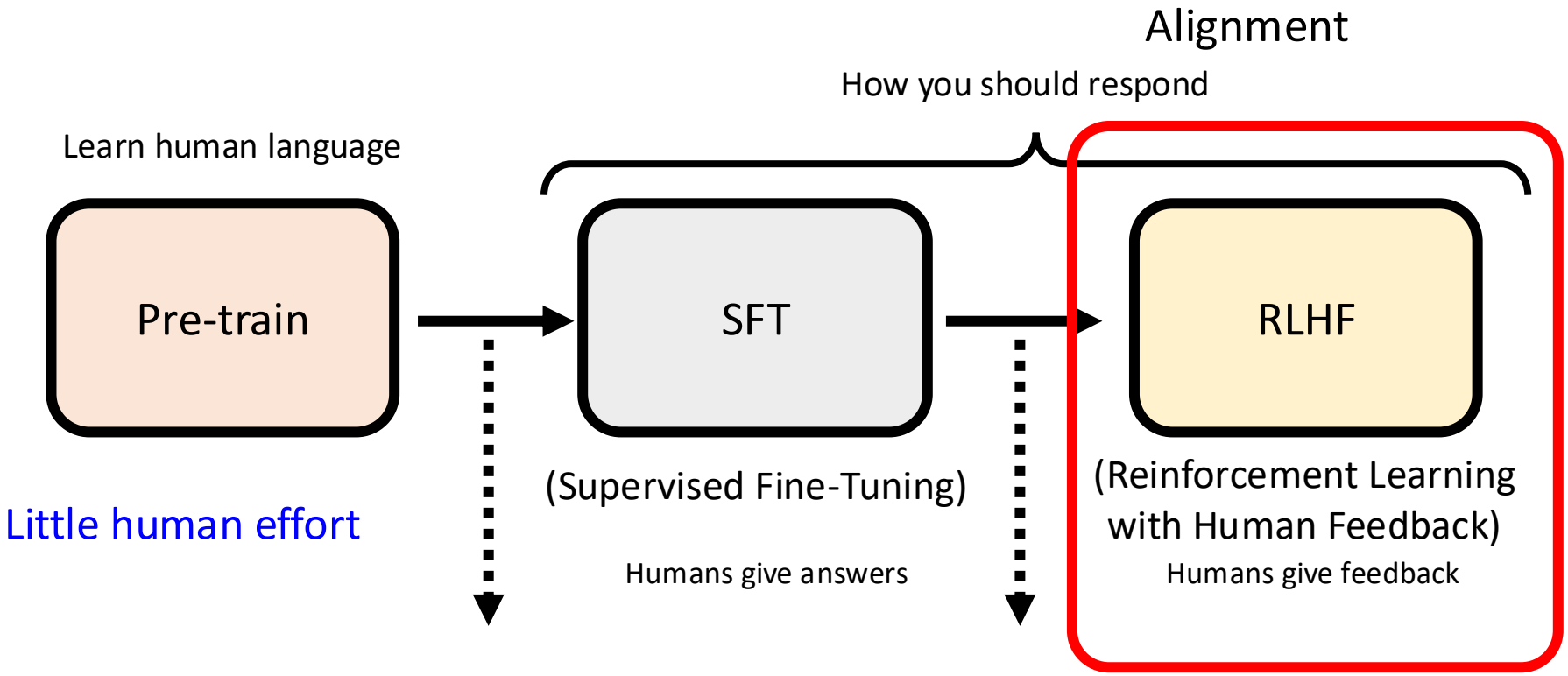
When asked "Teach me to make a gun"

lower the prob. of "A1"

raise the prob. of "A2"

credit: Prof. Hung-yi Lee,

How LLMs Learn (into ChatGPT)

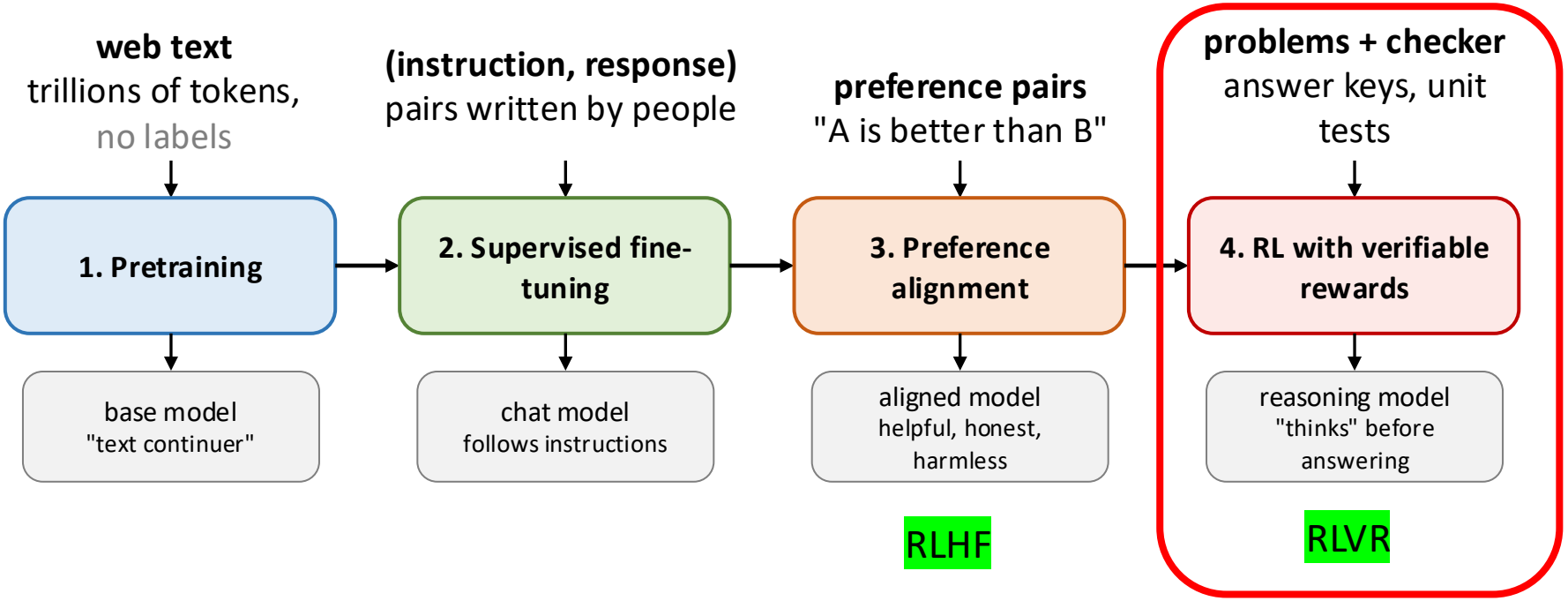


Each stage initializes from the previous stage's params (Initialization)

credit: Prof. Hung-yi Lee,

How a base decoder model becomes a chat/reason agent's model

the training pipeline



- For each stage we ask the course's four questions: **Data? Loss? Optimization? What is learned?**

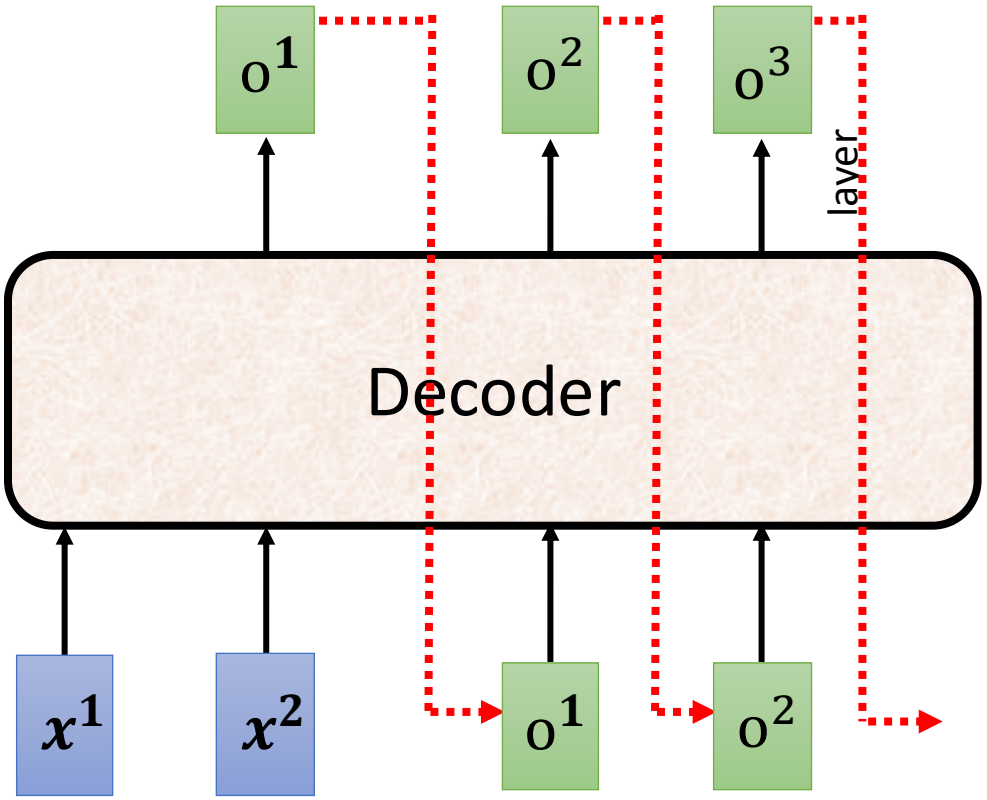
credit: Peng Wang, UVA CS 4774 L15

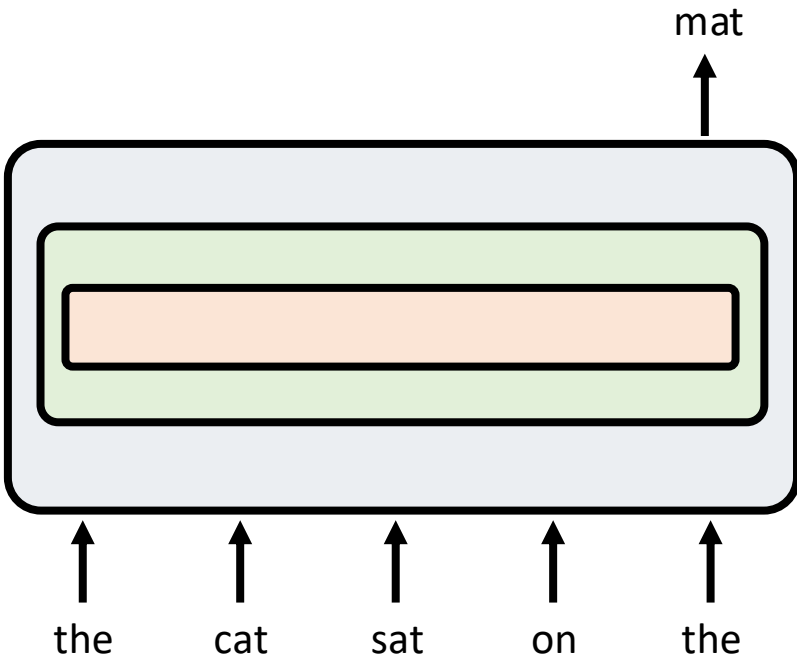
Roadmap : Today's Lecture

- **Generating text**
- Where the cost goes
- Cheaper attention
- Faster decoding and serving
- Choosing what to do

Generating: one token per forward pass

(Inference)

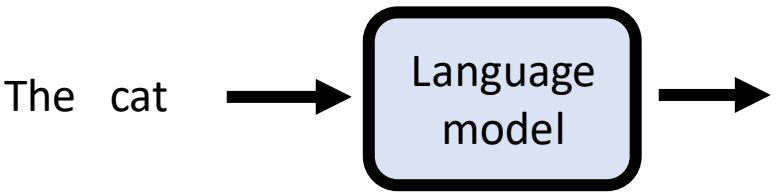




autoregressive loop

credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

How an LM Works Inside



Token prob. distribution

Token	Score
ran	0.2
sat	0.5
how	0.0000…
자	0.0000…
ち	0.0000…
@	0.0000…
……	

Vocabulary

Many valid continuations; how does the model pick one?

- Decoding: rule turning those probabilities into text

A distribution on token vocabulary

Decoding (A): Greedy - Every position argmax!

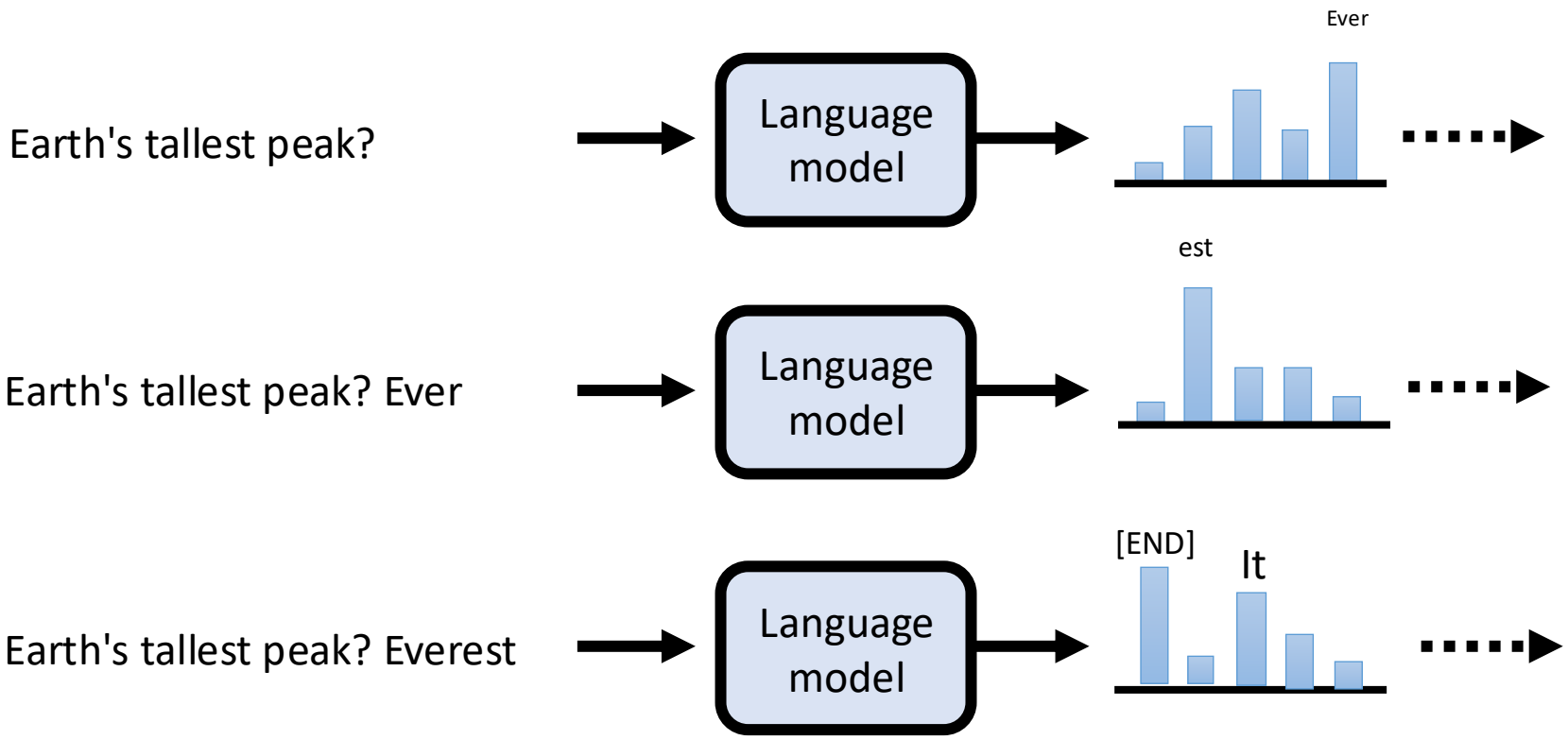
1. Run the model on the token (character like HW-Q1) so far
2. Take the logits at the LAST position → argmax
3. Predict the next character
4. Append it; stop at "." (or your special end token!) or at a length limit

repeat until a stop condition

```
prompt  : "the cat then "
step 1  : "the cat then n"
step 2  : "the cat then na"
  ...
final   : "the cat then napped
           beside the fence."
```

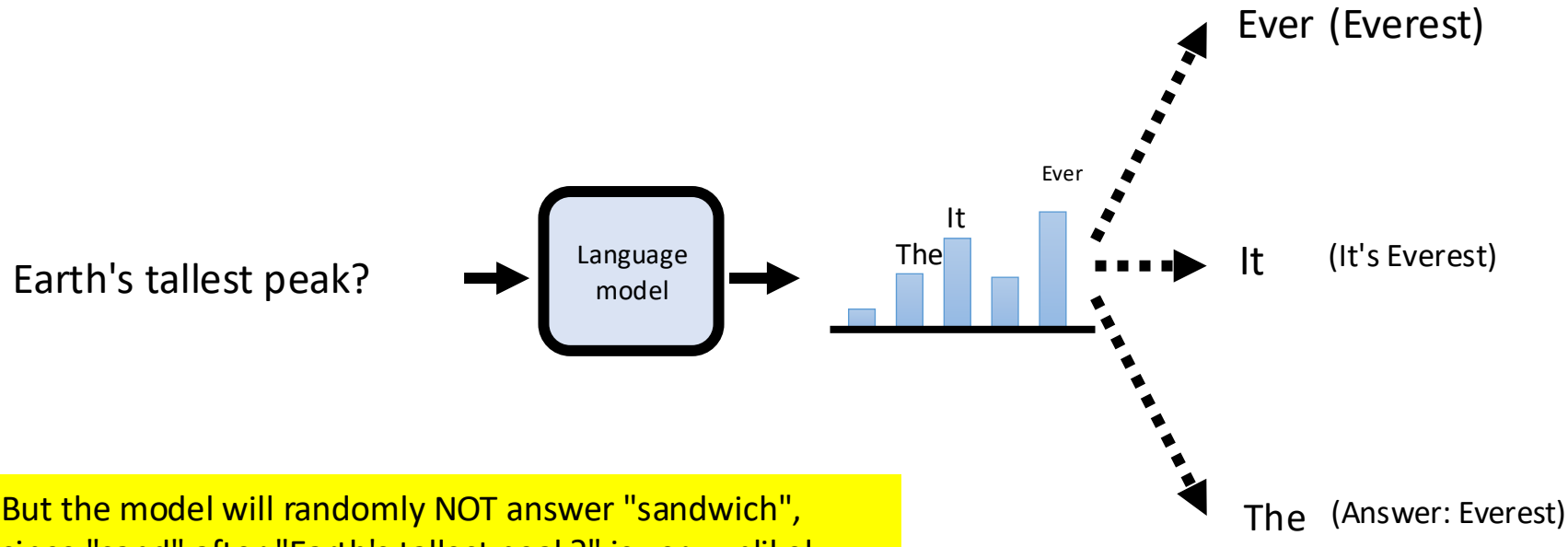
Same causal LLM model as training. Only the input changes:

Decoding (B): Sampling!



torch.multinomial(probs, 1)

Sampling make: Answers Differ Each Time



But the model will randomly NOT answer "sandwich", since "sand" after "Earth's tallest peak?" is very unlikely

torch.multinomial(probs, 1)

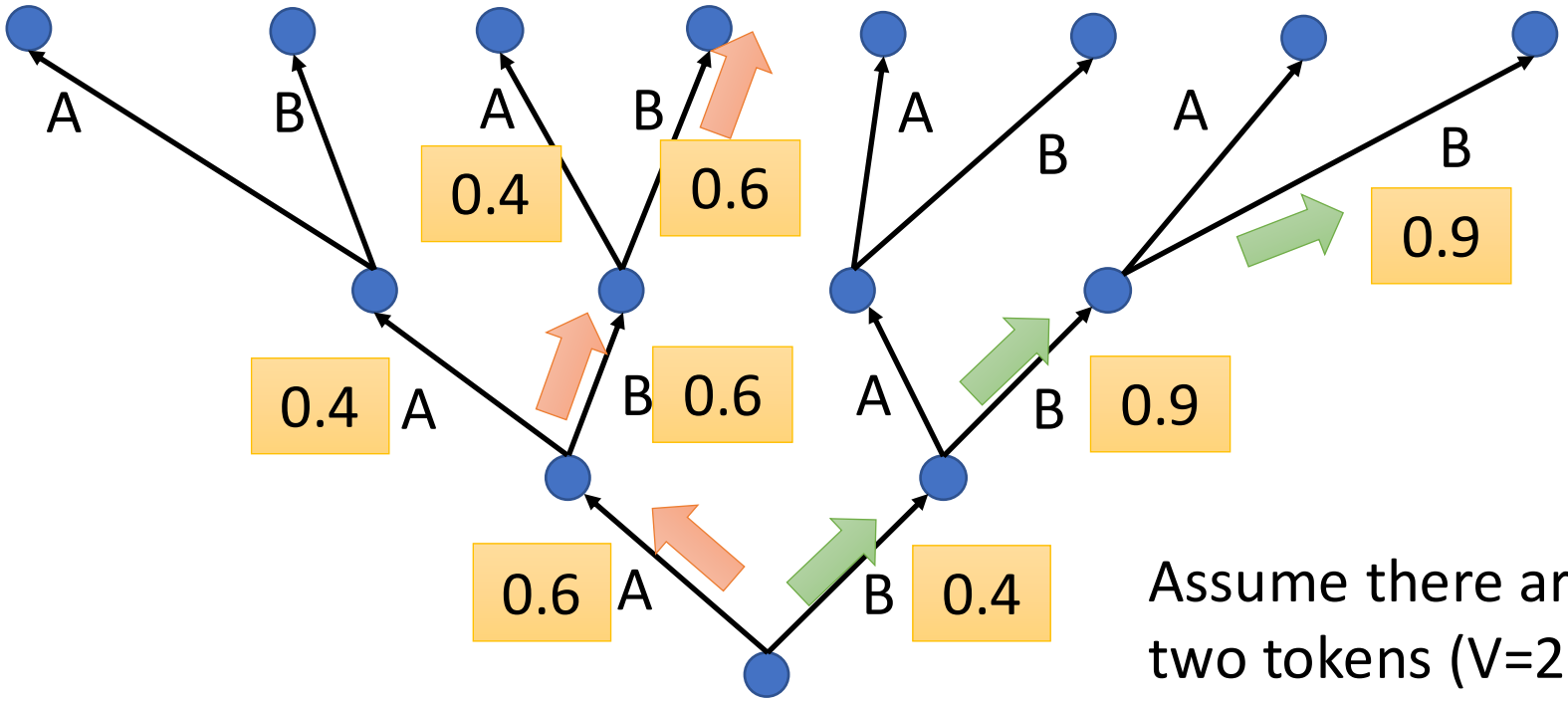
Decoding (C): Beam Search

The red path is ***Greedy Decoding***.

The green path is the best one.

Not possible to check all the paths … → Beam Search

e.g. B=2



Assume there are only two tokens (V=2).

credit: Prof. Hung-yi Lee, NTU ML 2021

Beam Search, Step by Step (B = 2)

Score of a path = product of its token probabilities (sum of log-probs)

Step	Candidates (path prob.)	Keep top 2
1	A 0.6, B 0.4	A, B
2	AA 0.24, AB 0.36, BA 0.04, BB 0.36	AB, BB
3	ABA 0.144, ABB 0.216, BBA 0.036, BBB 0.324	BBB, ABB

Output: BBB (0.324), the green path; greedy gives **ABB (0.216)**

Finished paths of different lengths? Divide by length

- Short answer p = 0.5 vs. BBB p = 0.324
- Raw score: 0.5 wins (fewer factors below 1)
- Per token: ln 0.5 / 1 = −0.69 vs. ln 0.324 / 3 = −0.38 → BBB wins

this compares the geometric mean per token:
- BBB: 0.324^(1/3) ≈ 0.69 per step
- short answer: 0.5 per step

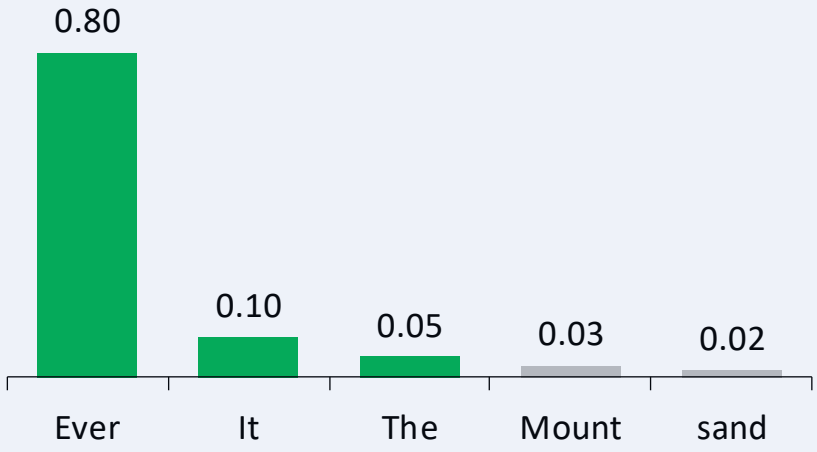
tree: Prof. Hung-yi Lee, NTU ML 2021

Decoding (D): Top-k Sampling

Rule: keep the k = 3 most likely tokens

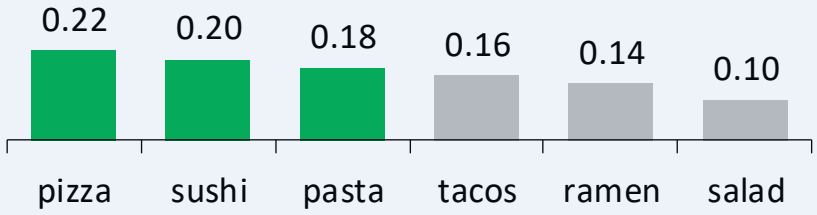
green = kept, then renormalized and sampled; gray = cut

Peaked: "Earth's tallest peak?"



Keeps 3 tokens (mass 0.95)

Flat: "My favorite food is"



Keeps 3 (mass 0.60); cuts tacos, ramen, salad

keep only the k most likely next tokens, renormalize their probabilities so they sum to one

Decoding (D): Top-k in Real Inference Code

```
z = logits[:, -1, :] / tau  # (batch, |V|), |V| ≈ 128k
kth = torch.topk(z, k).values[:, -1:]  # k-th largest
z = z.masked_fill(z < kth, float("-inf"))  # cut the rest
tok = torch.multinomial(z.softmax(-1), 1)  # draw one token
```

Is selecting top-k from V quick? Yes

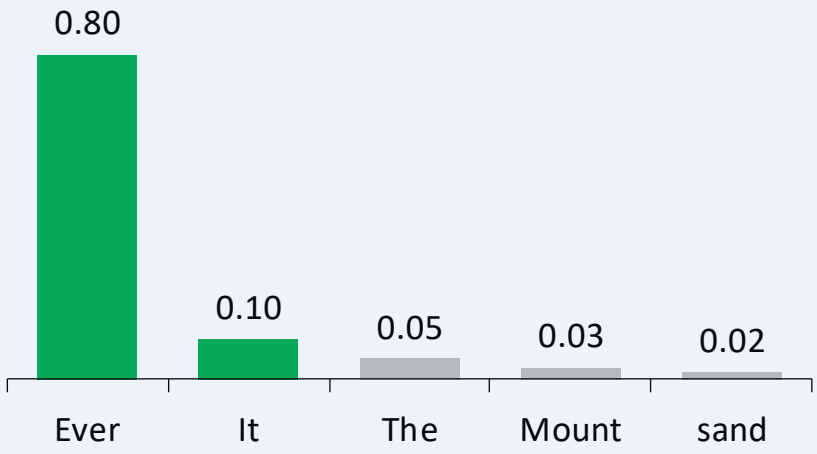
- Finding the top k is one quick pass, no sorting needed
- Computing the scores costs far more than picking from them
- Servers fuse these steps into one GPU kernel
- The real cost: reading weights and the KV cache

Decoding (E): Top-p (also called Nucleus) Sampling

Rule: keep the fewest top tokens whose total ≥ p = 0.9

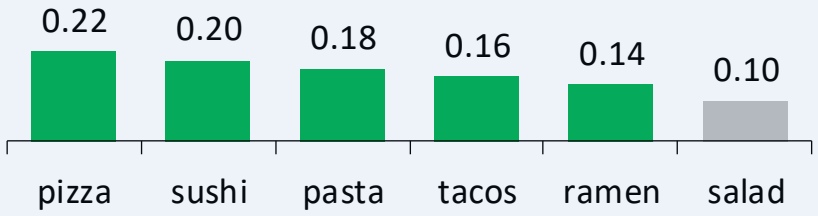
green = kept, then renormalized and sampled; gray = cut

Peaked: "Earth's tallest peak?"



Keeps 2 tokens (0.80 + 0.10 = 0.90)

Flat: "My favorite food is"



Keeps 5 (mass 0.90); cuts only salad

Why Top-p Sampling?
Paper: The Curious Case of Neural Text Degeneration

https://arxiv.org/abs/1904.09751

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Beam Search, *b*=32:
"The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de ..."

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

GPT-2 Large; blue = repetition, red = incoherence. Source: Holtzman et al., Fig. 1

Randomness is needed for decoder when generating sequence in some tasks, like open-ended writing

➔ Chatbots LLM sampling usually with a temperature and top-p

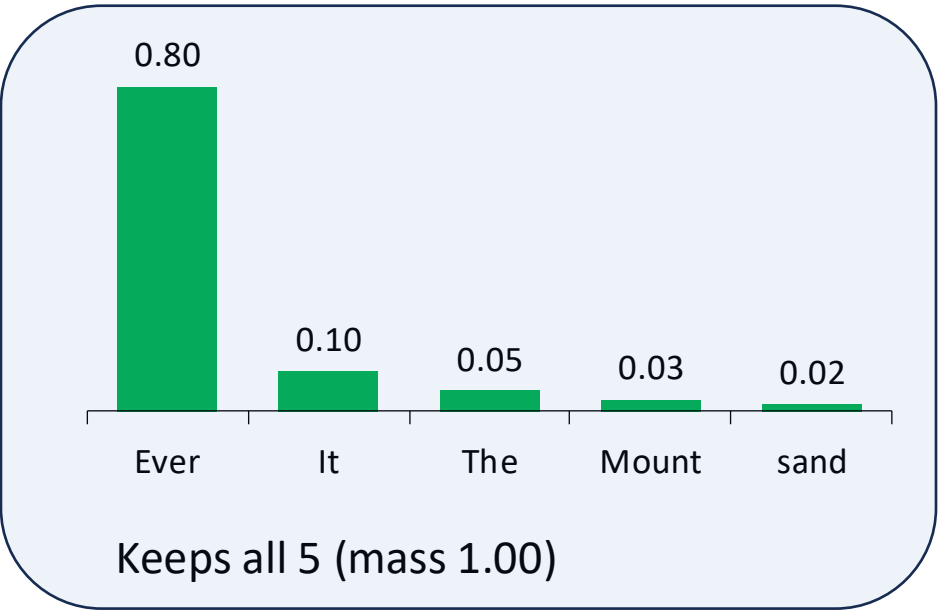
credit: Prof. Hung-yi Lee, NTU ML 2021

Pure Sampling: Top-k with k = |V|, Top-p with p = 1

Rule: keep every token; sample from the full distribution

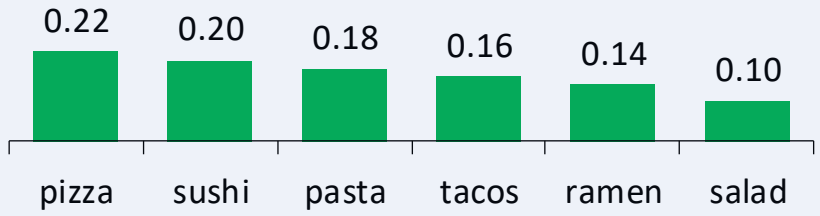
green = kept: nothing is cut or renormalized

Peaked: “Earth's tallest peak?”



Keeps all 5 (mass 1.00)

Flat: “My favorite food is”



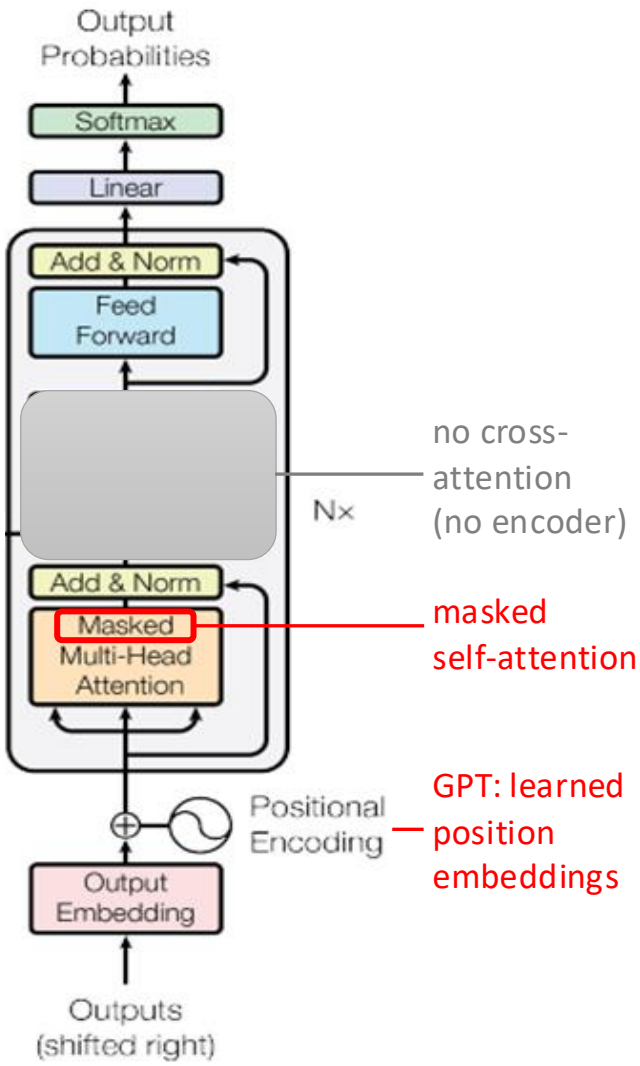
Keeps all 6 (mass 1.00)

k = |V| (whole vocabulary) = p = 1 = pure sampling

When sampling: Softmax Temperature τ

p = softmax(z / τ) divide logits by τ, then softmax

z = [3.0, 1.5, 0.5, −1.0] (training uses τ = 1)



When sampling: Softmax Temperature τ

p = softmax(z / τ) divide logits by τ, then softmax

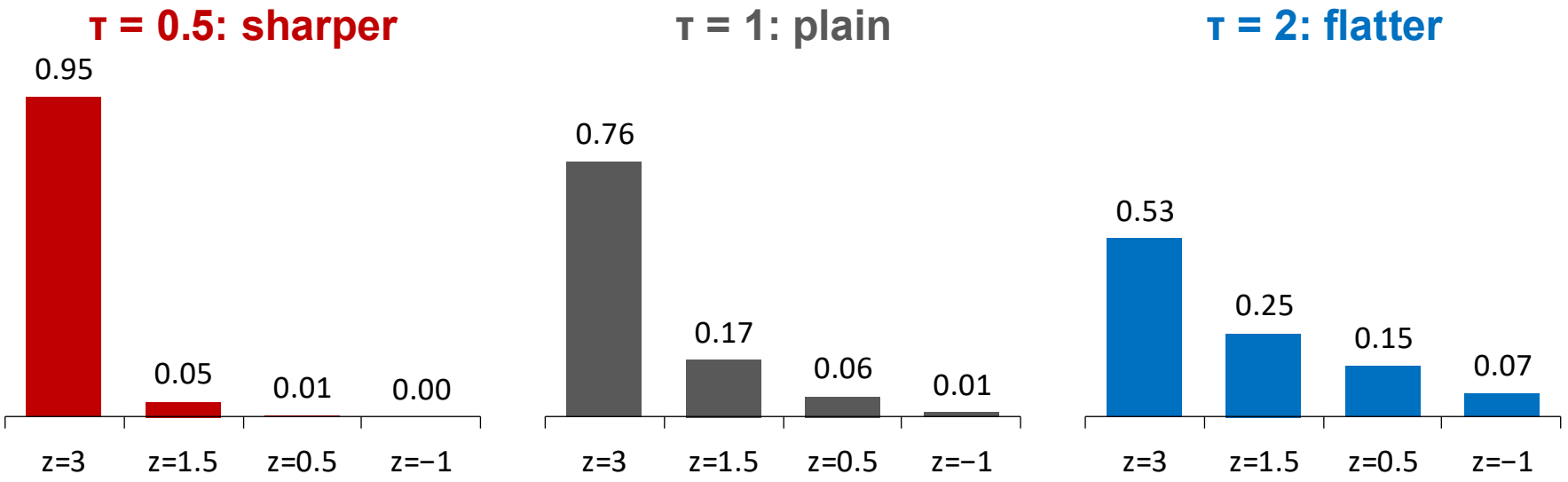
z = [3.0, 1.5, 0.5, −1.0] (training uses τ = 1)

τ	z / τ	p = softmax(z / τ)
0.5 (sharper)	[6, 3, 1, −2]	[0.946, 0.047, 0.006, 0.0003]
1 (plain)	[3, 1.5, 0.5, −1]	[0.756, 0.169, 0.062, 0.014]
2 (flatter)	[1.5, 0.75, 0.25, −0.5]	[0.528, 0.249, 0.151, 0.071]

When sampling: Three τ, Three Distributions

p = softmax(z / τ) divide logits by τ, then softmax

z = [3.0, 1.5, 0.5, −1.0] (training uses τ = 1)



- **Same order in all three; τ only changes the spread**

Summy: sampling: Softmax Temperature τ

p = softmax(z / τ) divide logits by τ, then softmax

z = [3.0, 1.5, 0.5, −1.0] (training uses τ = 1)

τ	z / τ	p = softmax(z / τ)
0.5 (sharper)	[6, 3, 1, −2]	[0.946, 0.047, 0.006, 0.0003]
1 (plain)	[3, 1.5, 0.5, −1]	[0.756, 0.169, 0.062, 0.014]
2 (flatter)	[1.5, 0.75, 0.25, −0.5]	[0.528, 0.249, 0.151, 0.071]

- **τ < 1: sharper** → mass on the argmax;
  - τ→0 gives one-hot
- **τ > 1: flatter** → tail gains weight;
  - τ→∞ gives uniform 0.25
- **Ranking never changes:** $p_i / p_j = \exp((z_i - z_j) / \tau)$

Homework: both - greedy vs sampling

- **Greedy**: argmax, always the same output
- **Sampling**: may generate long tail
- **Temperature τ**:
  - τ < 1 sharper,
  - τ > 1 flatter
- **Top-k / top-p**:
  - sample only from the most likely tokens

temperature τ	a	o	e	_
τ = 0.5	0.95	0.05	0.01	0.00
τ = 1.0	0.76	0.17	0.06	0.01
τ = 2.0	0.53	0.25	0.15	0.07

$p = \text{softmax}(z / \tau)$,
$z = [3.0, 1.5, 0.5, -1.0]$

Please try to implement both:

- argmax when you want one best guess (as when scoring next-token accuracy for like translation tasks),
- sampling for text generation (so outputs differ).

Quiz examples: decoding

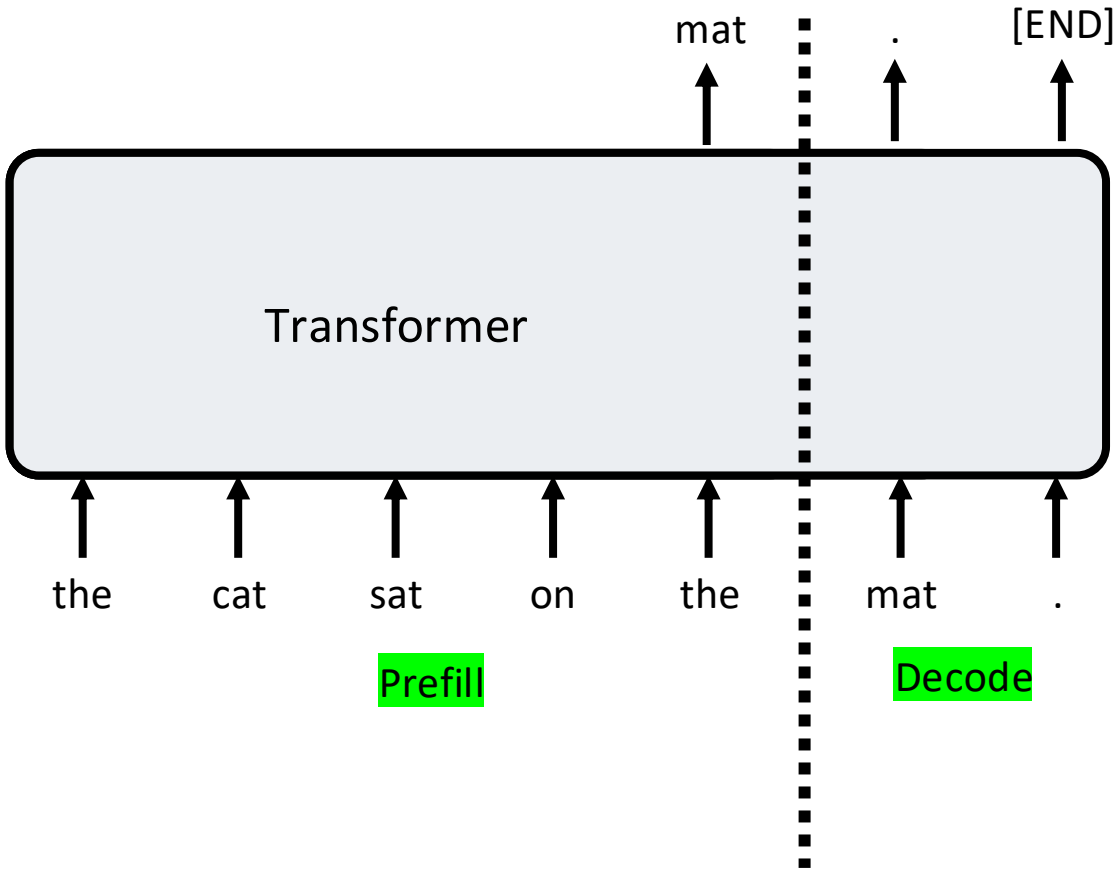
- 1. You run greedy decoding twice on the same prompt. Same output? With sampling at $\tau = 1$?
  - Greedy: yes, always the same. Sampling: no, runs differ
- 2. Logits $z = [2, 1, 0]$. What does $\text{softmax}(z / \tau)$ approach as $\tau \to 0$? As $\tau \to \infty$?
  - $\tau \to 0$: one-hot [1, 0, 0], i.e. greedy; $\tau \to \infty$: uniform, ⅓ each
- 3. Why does beam search suit translation but give bland stories?
  - It maximizes probability: fine for one right answer; stories turn generic
- 4. Why can a decoder reuse old keys and values, but BERT cannot?
  - Causal mask: old tokens never see new ones; BERT looks both ways

Roadmap : Today's Lecture

- Generating text
- **Where the cost goes**
- Cheaper attention
- Faster decoding and serving
- Choosing what to do

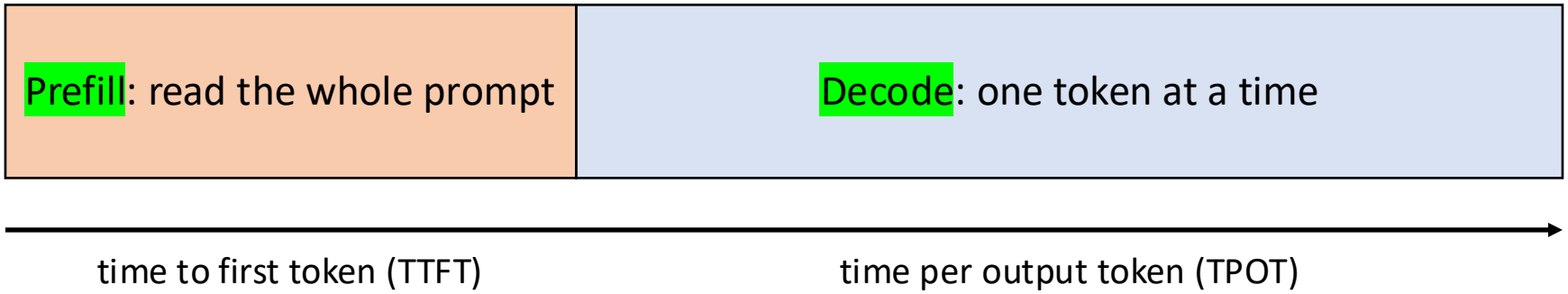
Prefill and decode

Two different stages in (Inference)



credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

One inference request, two phases

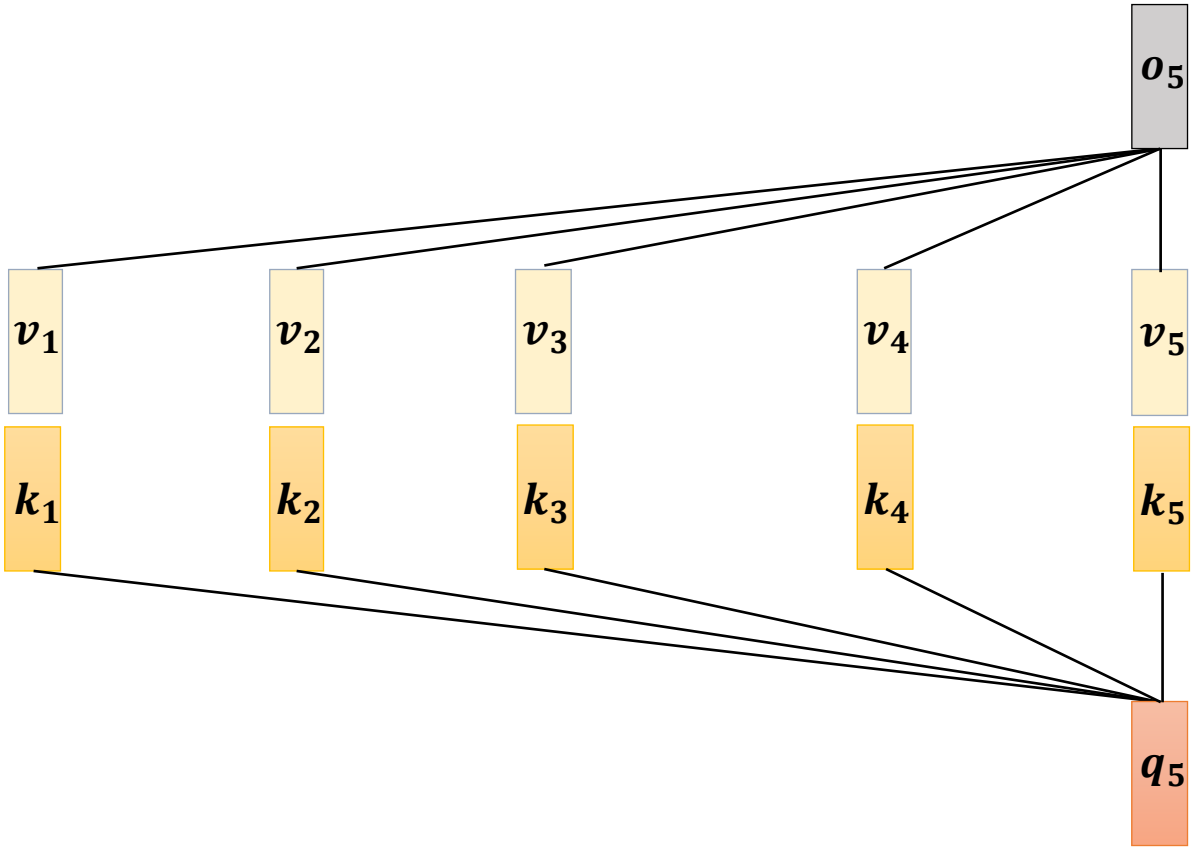


$$\text{latency} = \text{TTFT} + \text{TPOT} \times (\text{output tokens} - 1)$$

- Throughput: tokens per second, across all users
- Target: for example, 99% of users under a latency limit

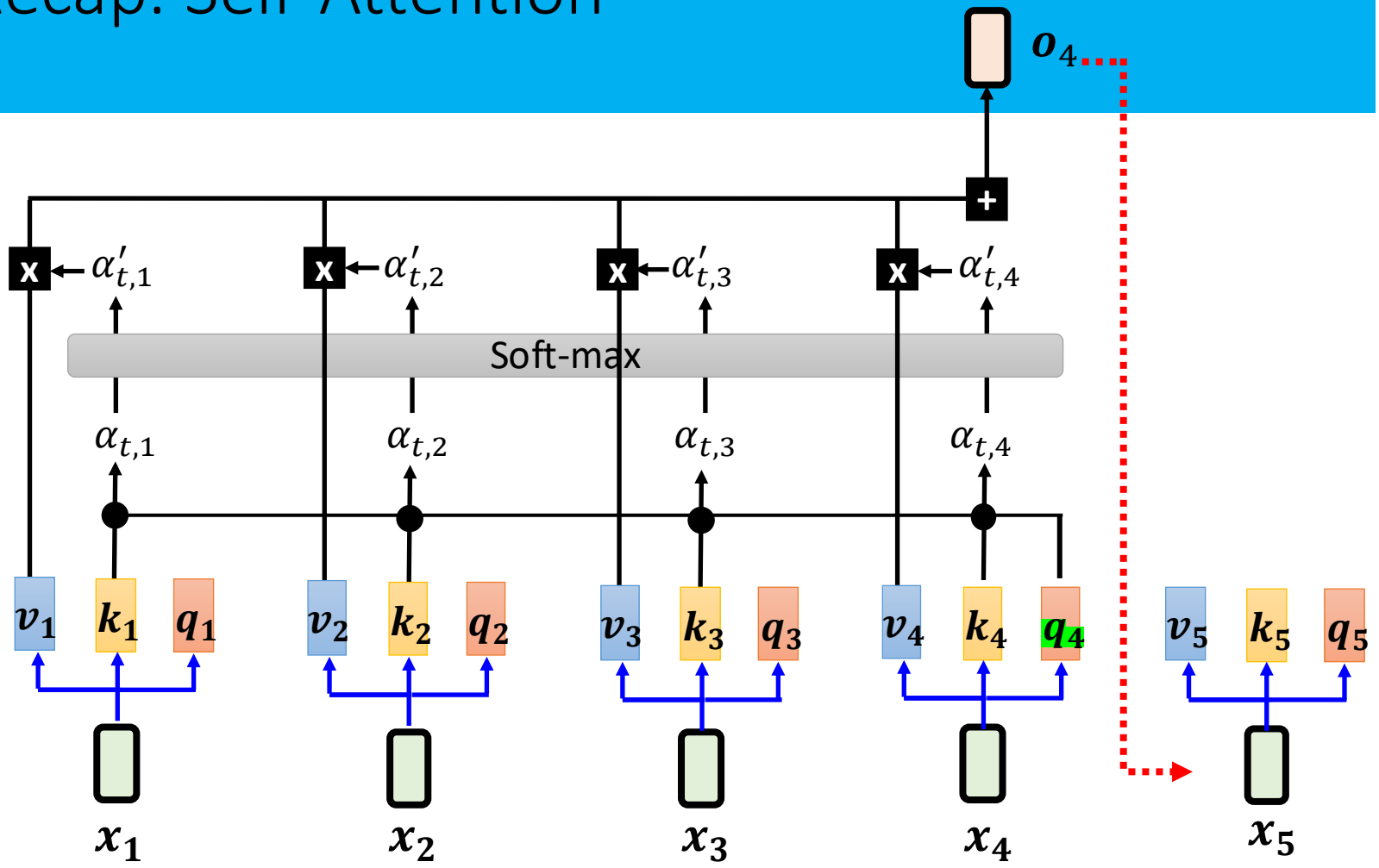
As an example, assume the prompt is 1,000 tokens, TTFT is half a second, and TPOT is 20 milliseconds. A 200-token answer takes about four and a half seconds, and 50 tokens per second is already faster than people read

KV cache: only the new query is new



credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

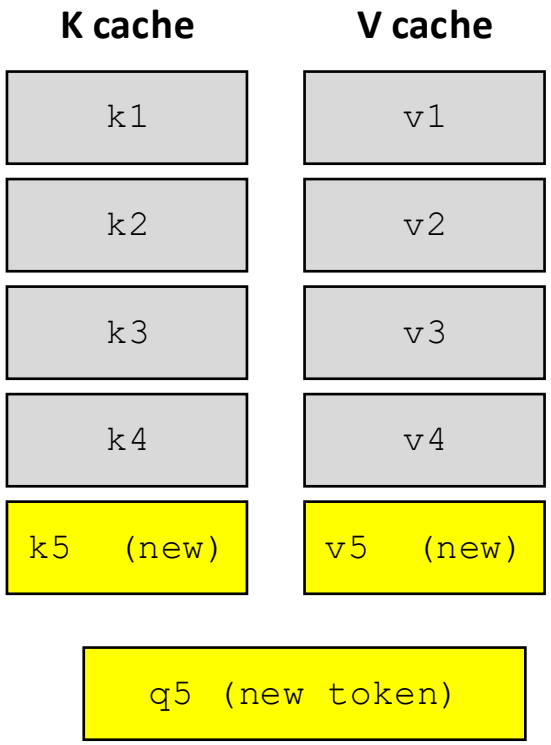
Recap: Self-Attention



credit: Prof. Hung-yi Lee,

KV cache: don't recompute the past

- Naive loop: at step t, re-run the model on all t tokens
  - recomputes K and V for the whole prefix, every step
- Causal mask ⇒ old tokens never attend to new ones, so their **keys and values never change**
- So cache them, per layer:
  - compute q, k, v for the NEW token only
  - append k, v to the cache
  - output = softmax(q K_{cache}^{T} / √D) V_{cache}



Gray rows: reused from the cache.
Yellow: computed this step.

Only valid for causal (decoder) models:
In BERT, a new token changes every old representation.

EXTRA

KV cache: trading memory for speed

per new token	without cache	with cache
K, V projections	all t tokens (to its left)	1 token
attention scores	t × t (only last row used)	1 × t
extra memory	none	2 × layers × t × D values

- Example: 32 layers, D = 4096, 16-bit numbers, a 4,096-token context
  - 2 × 32 × 4096 × 4096 values × 2 bytes ≈ 2.1 GB per sequence
  - (real models shrink this by sharing K/V across heads and by quantizing the cache)
- This memory cost, not compute, often limits the batch size and context length when serving LLMs

For short sequences, re-running the whole prefix is fine; no cache needed.

KV cache: a real model's numbers

EXTRA

Gemma 2

Parameters	2B	9B	27B
d_model	2304	3584	4608
Layers	26	42	46
Pre-norm	yes	yes	yes
Post-norm	yes	yes	yes
Non-linearity	GeGLU	GeGLU	GeGLU
Feedforward dim	18432	28672	73728
Head type	GQA	GQA	GQA
Num heads	8	16	32
Num KV heads	4	8	16
Head size	256	256	128
Global att. span	8192	8192	8192
Sliding window	4096	4096	4096
Vocab size	256128	256128	256128
Tied embedding	yes	yes	yes

Gemma 2 27B, per token:
46 layers × 16 KV heads × 128
× 2 (K and V) × 2 bytes = 376,832 B
≈ 0.37 MB (0.75 MB without sharing)

A100, 80 GB: about 228k tokens
(about 114k without sharing)

credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

EXTRA

Memory: weights plus KV cache

Item	Size
Weights: 7B parameters at 16 bits (shared)	14 GB
KV cache per token: 2 × 32 × 4096 × 2 bytes	524 KB
KV cache, one user, 4096 tokens	2.1 GB
KV cache, 8 users	17 GB

Weights are shared. Each user pays for a cache.
The cache often limits the batch

source: same example as the GPT lecture; numbers computed

Summary Latency: prefill, decode, and local vs API

- Each model call = **prefill** (read the whole context, in parallel: fast per token) + **decode** (one token at a time, using the KV cache)
- Agent latency ≈ Σ over steps of (prefill + decode + tool time); e.g. 20 × (0.5 s + 300 tok ÷ 50 tok/s + 1 s) ≈ 150 s
- Speed-ups: fewer steps (code actions), shorter outputs, parallel tool calls, cached prefixes, a smaller model for easy steps

	Local model (e.g. 8B via Ollama)	**API model (frontier)**
Quality on hard agent tasks	lower	higher
Cost	no per-token fee; your hardware	pay per token
Privacy	data stays on your machine	data sent to the provider
Latency	no network; slow prefill on long contexts	network + queue; fast hardware

credit: UVA 2026 SP student team, "Model serving" survey; numbers illustrative

Summary of Today: LLM inference

- **Generating text:** the loop; greedy, sampling, temperature, beam search
- **Where the cost goes:** prefill vs decode; the KV cache; memory and speed limits
- **Cheaper attention:** FlashAttention; sharing or compressing K and V; windows
- **Faster decoding and serving:** speculative decoding; prompt caching; batching; paged memory
- **Choosing what to do:** which trick first, and what each one costs

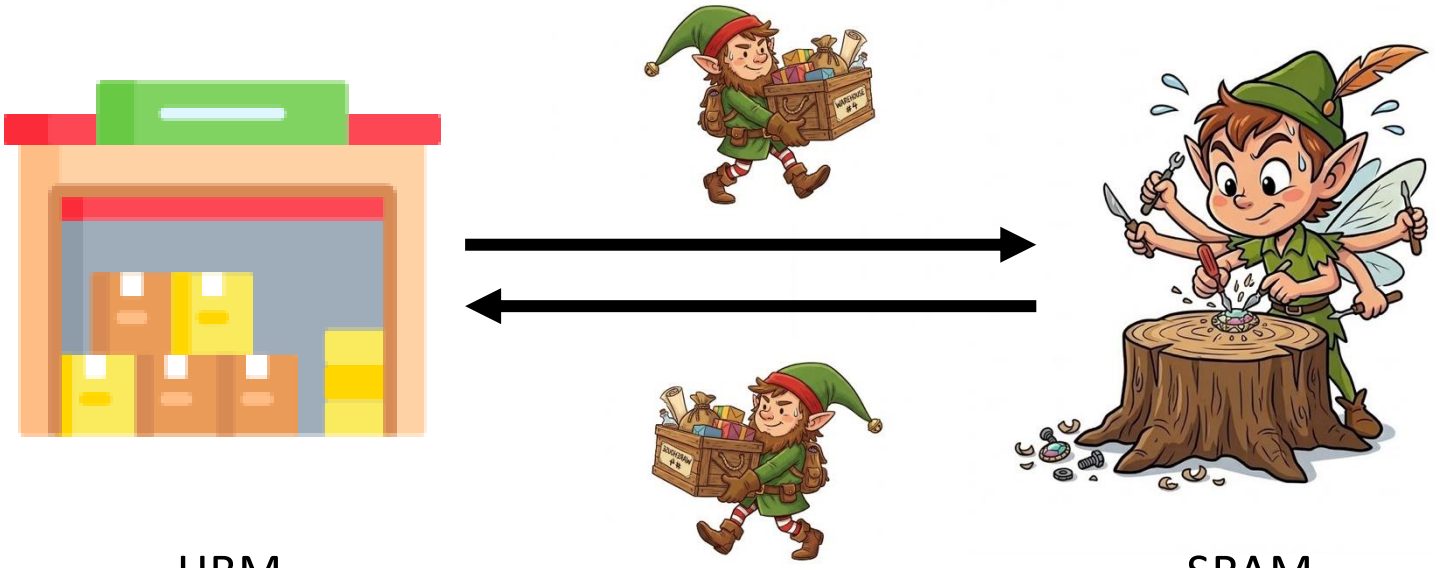
Same trained network throughout: no gradients today, only forward passes, done fast and cheaply.

EXTRA Content

Roadmap : Today's Lecture

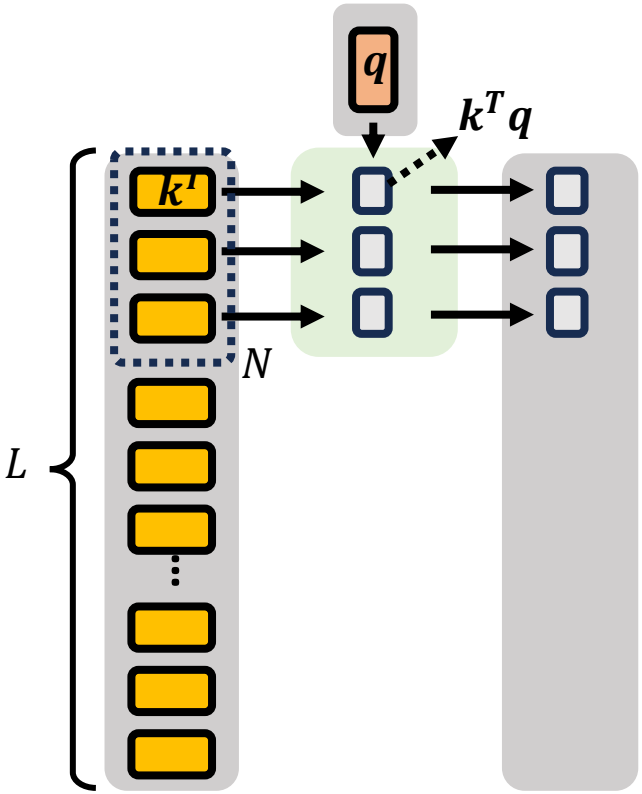
- Generating text
- Where the cost goes
- **Cheaper attention**
- Faster decoding and serving
- Choosing what to do

FlashAttention: memory, not math, is slow



credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

FlashAttention: scores one block at a time

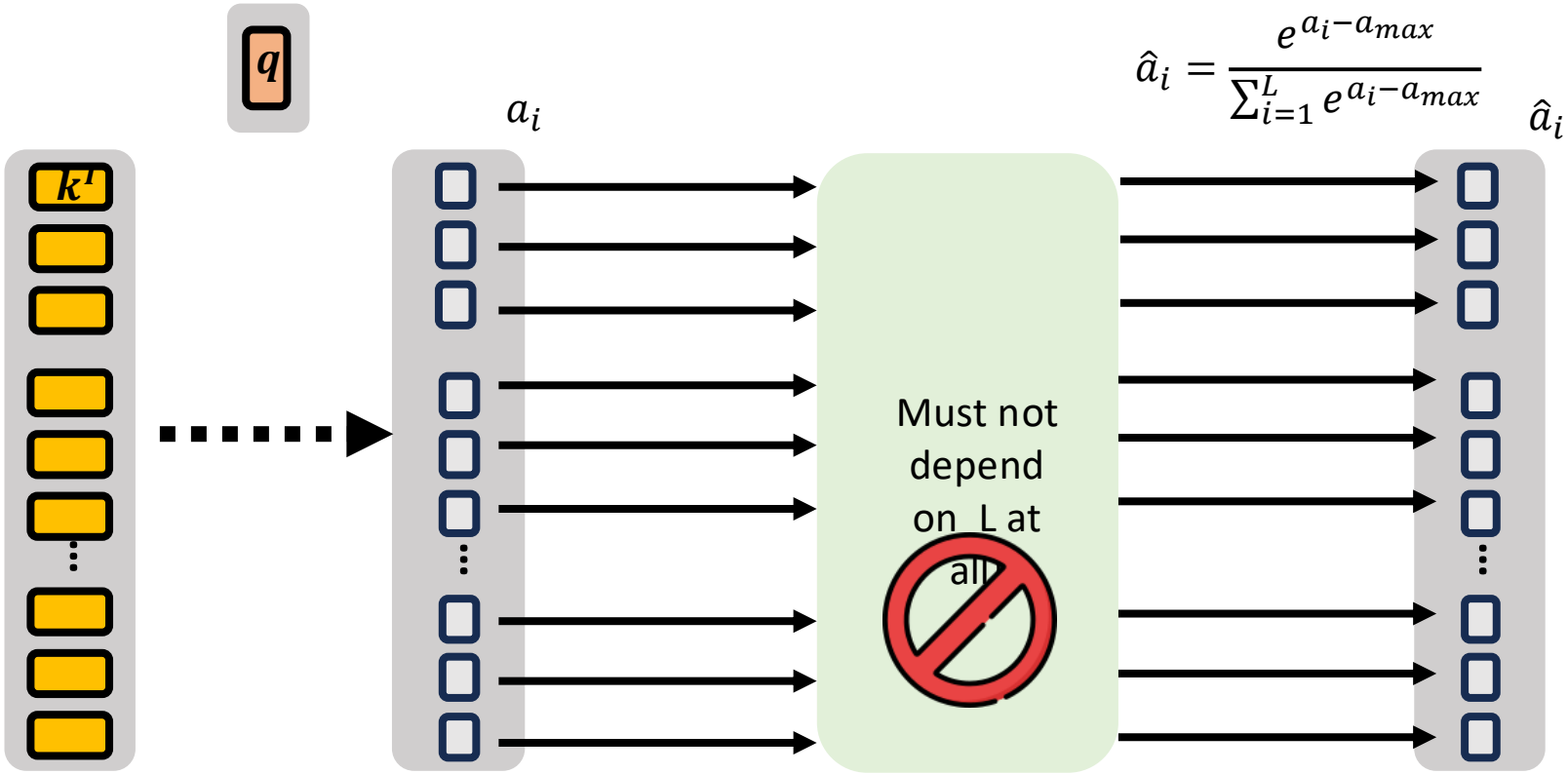




$k^T q$ here is one dot product q·k: the score of one key for this query (our code keeps one token per row)

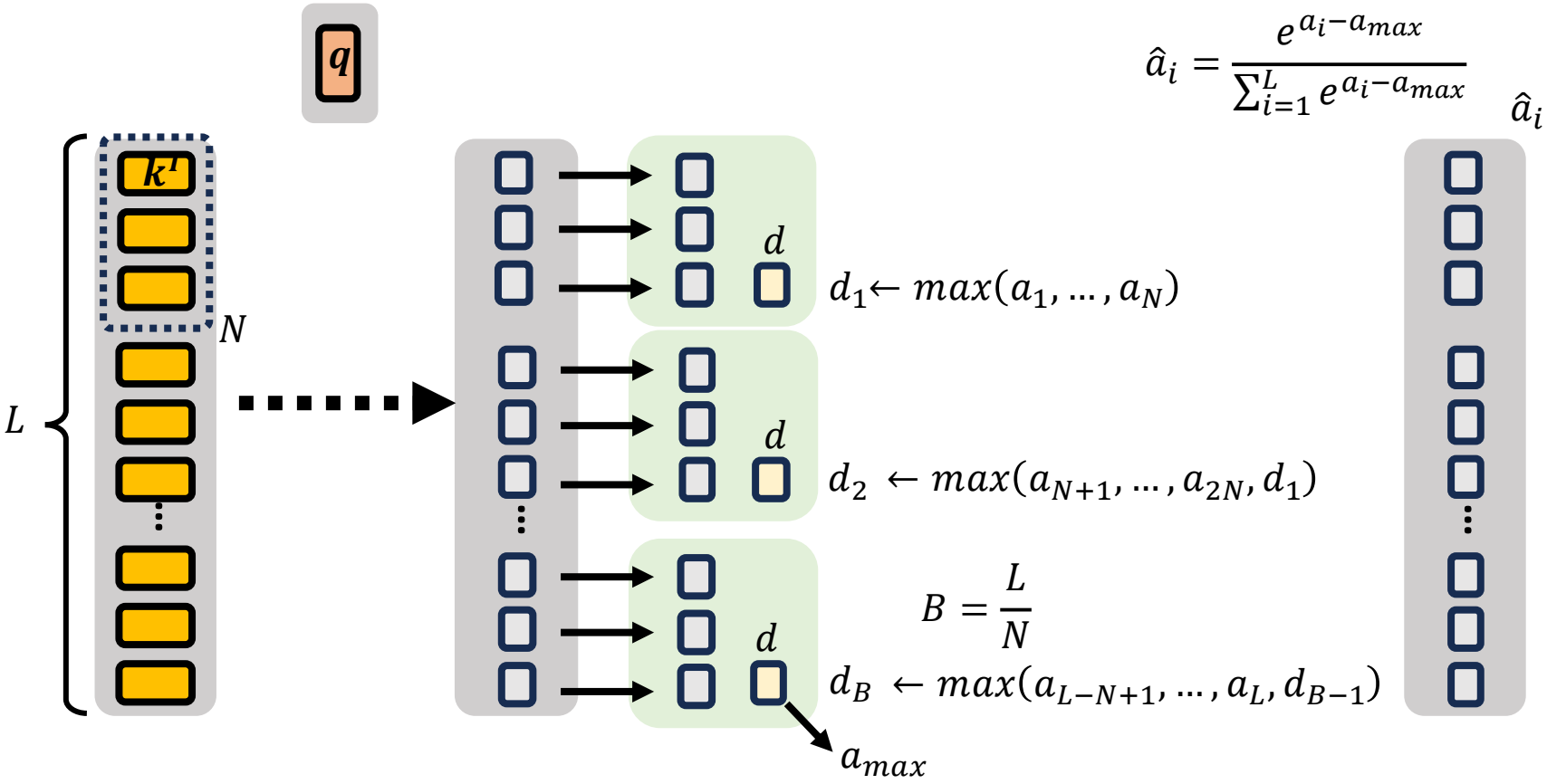
credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

FlashAttention: softmax needs the whole row



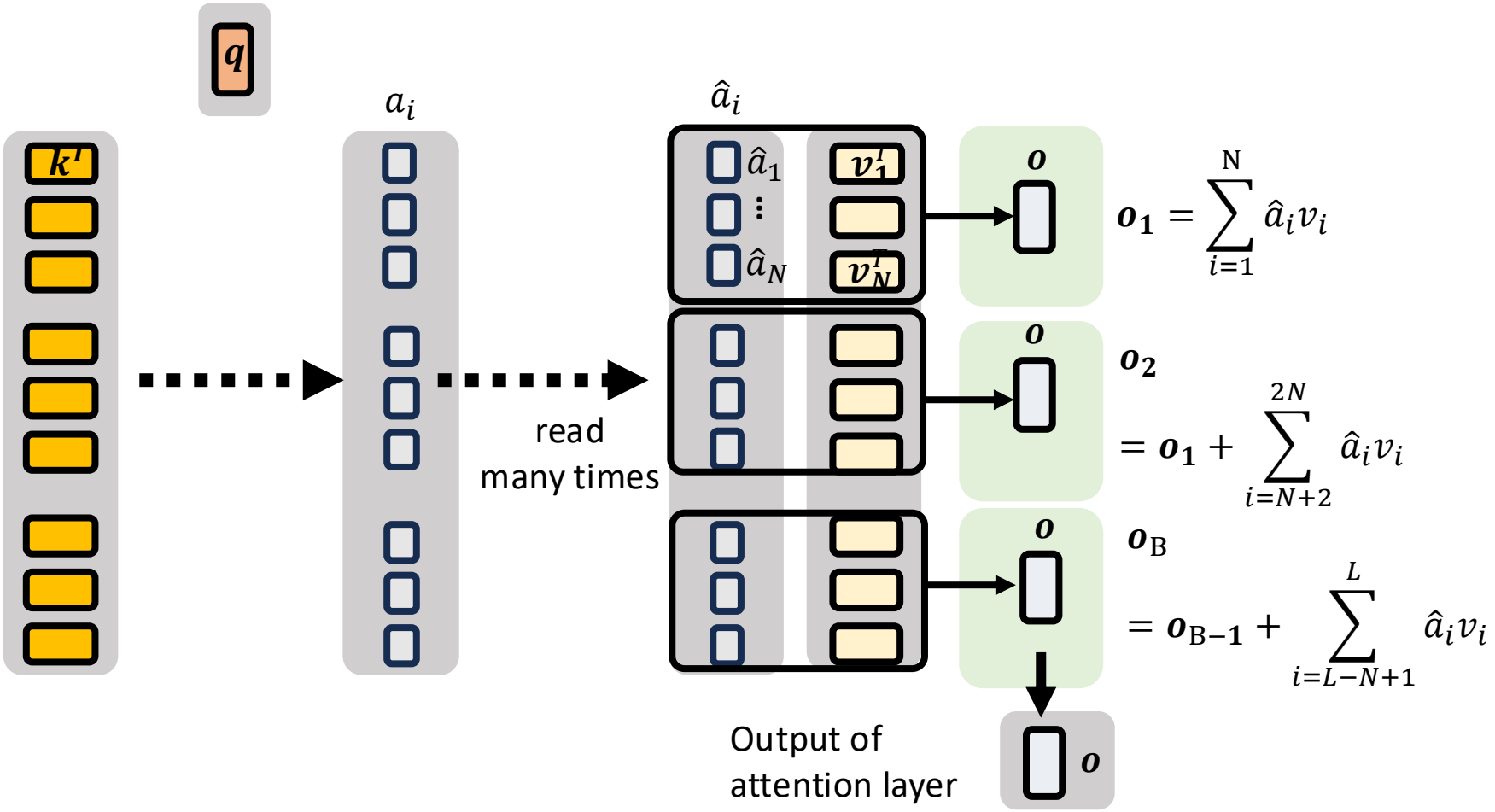
credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

FlashAttention: a running maximum



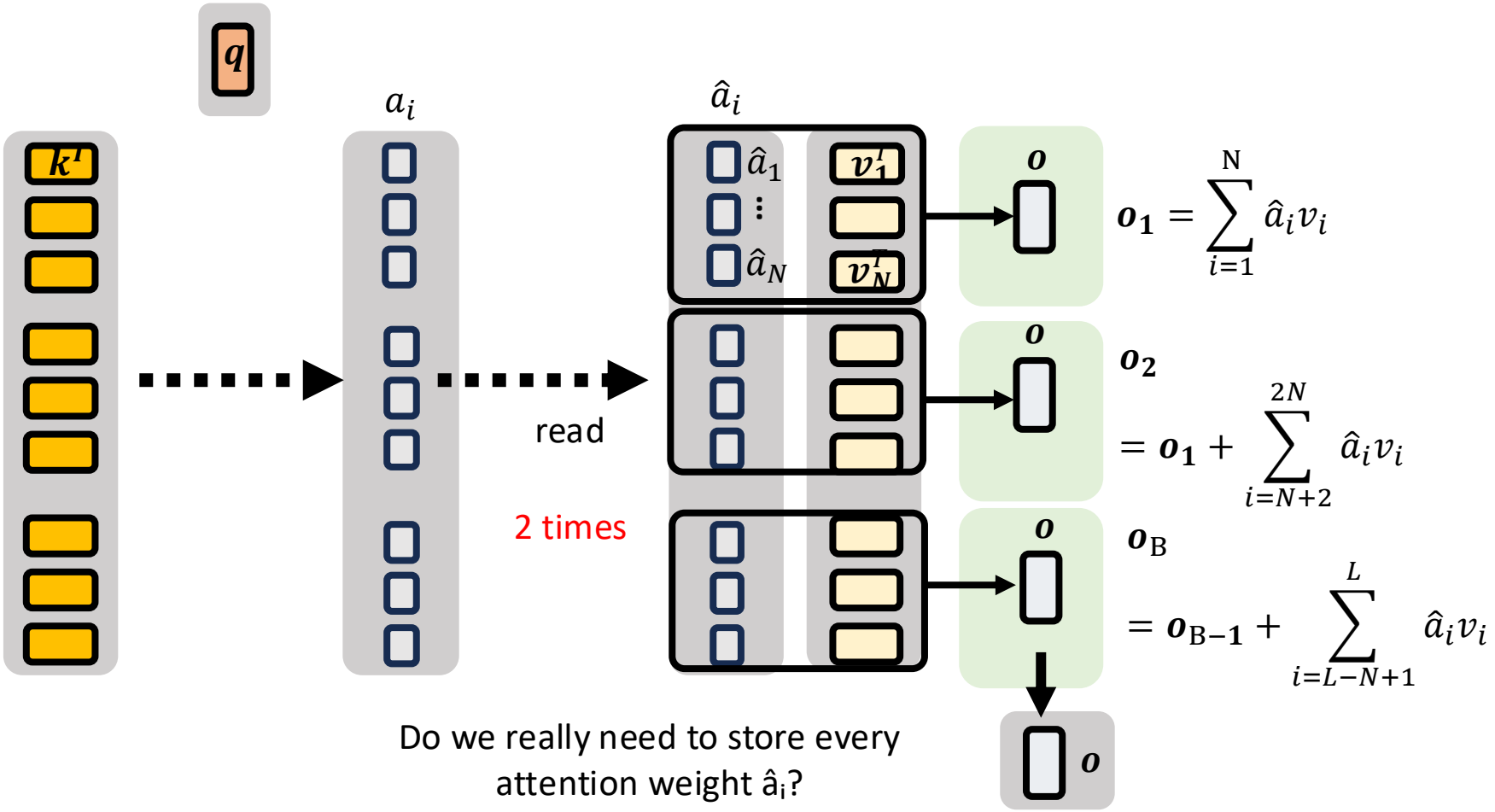
credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

FlashAttention: the output, chunk by chunk



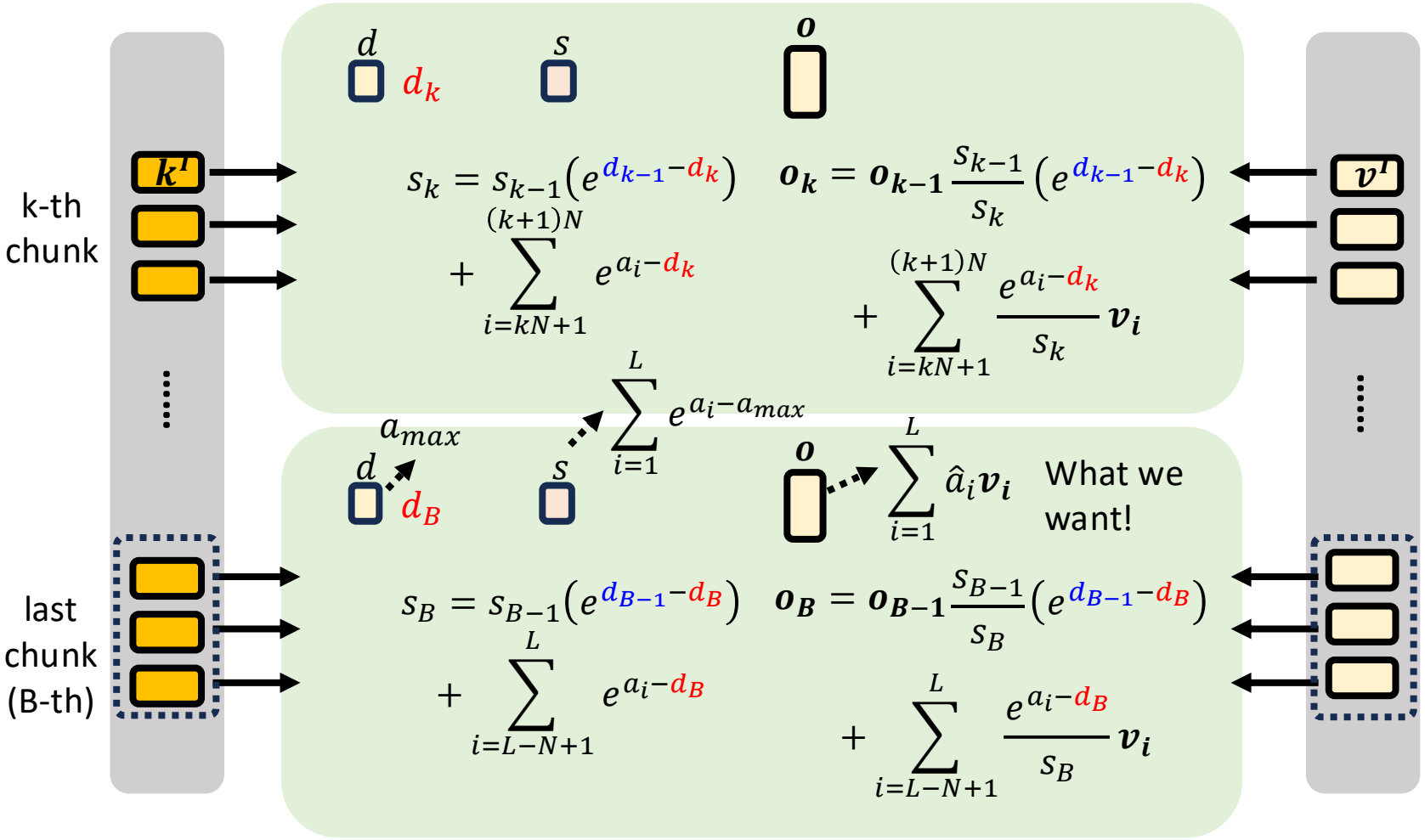
credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

FlashAttention: can we read K and V less?



credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

FlashAttention: one pass, rescale as you go



credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

FlashAttention: work in tiles

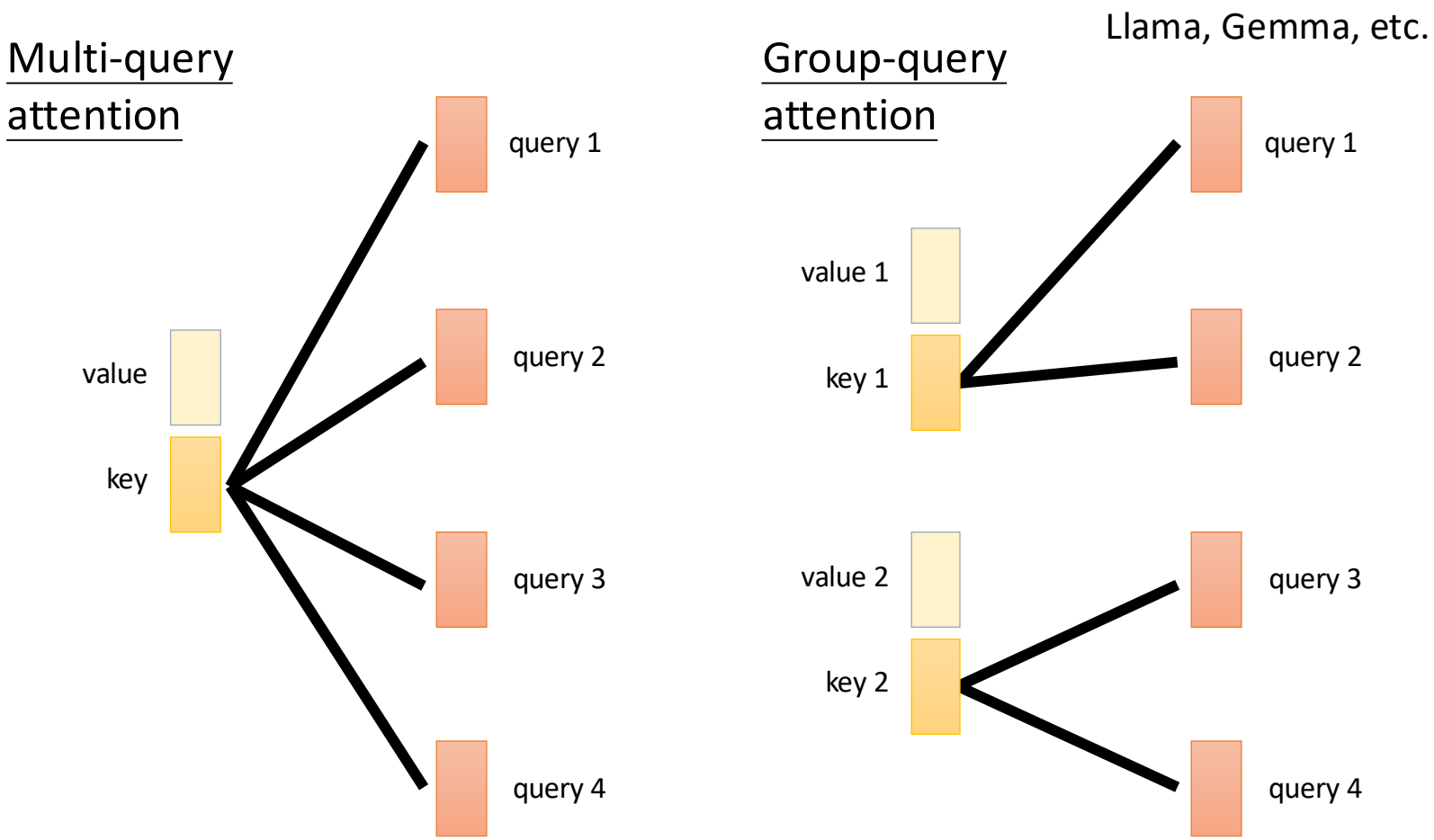
blue: done; green: this tile; gray: to do

- Compute one tile in fast on-chip memory
- Keep a running softmax per row
- Never store the full score matrix
- Exact: same math, less data movement

Same idea as cache blocking: reorder the work to fit fast memory

source: Dao et al., NeurIPS 2022

Sharing K and V: multi-query and grouped-query



credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

GQA: count the cache

$$\text{cache} = 2\,L\,\frac{D}{H}\,H_{kv} \times \text{bytes}$$

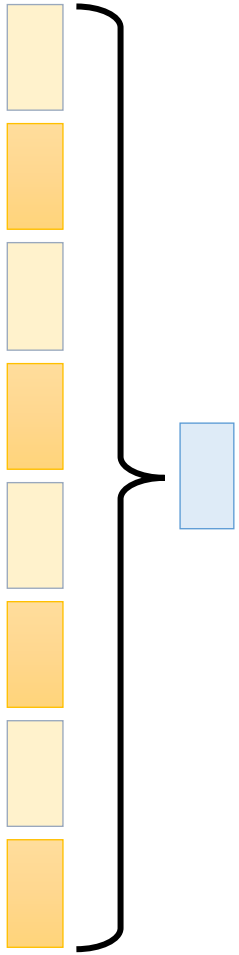
Llama-2-70B, 16-bit	**Per token**	**4096 tokens**
64 KV heads	2.6 MB	10.7 GB
8 KV heads (GQA)	328 KB	1.3 GB

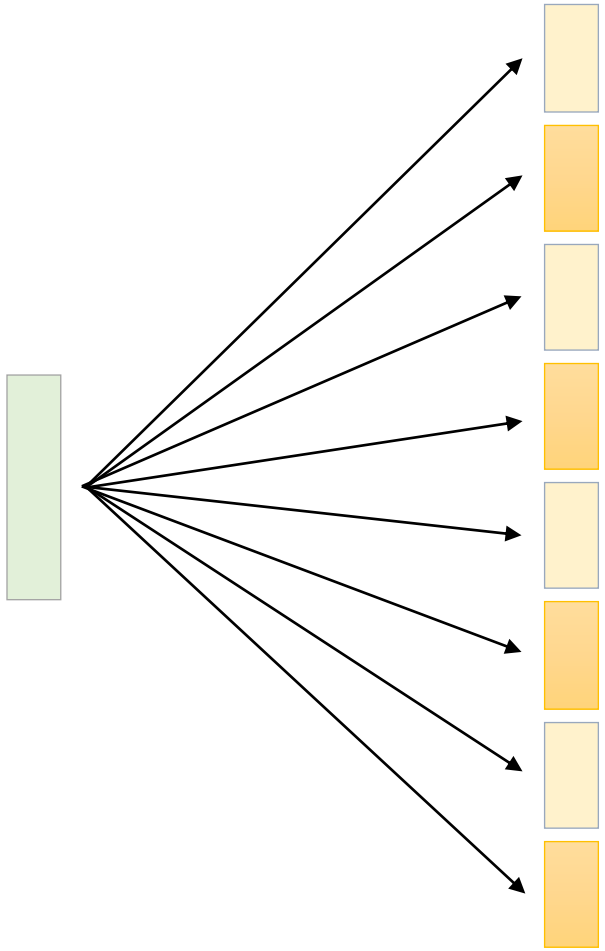
An 8× smaller cache means more users per GPU, or a longer context

source: configuration from UVA GenAI S24 deck; computed

Compressing K and V: multi-head latent attention

Multi-head Latent Attention

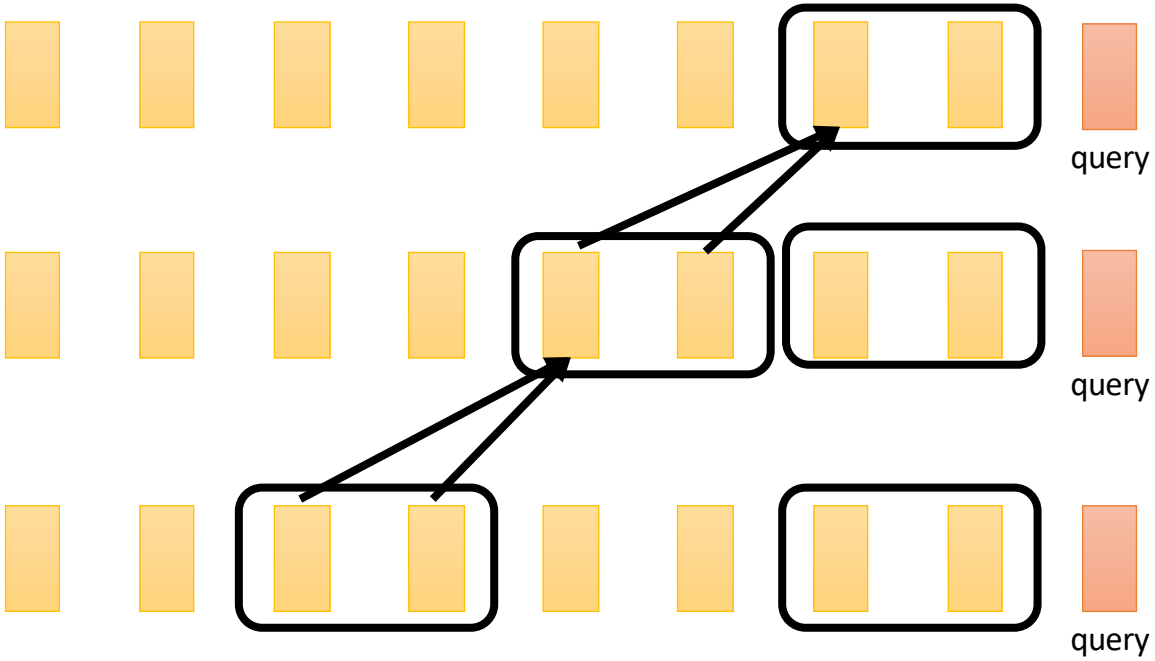




https://arxiv.org/abs/2405.04434

credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

Sliding window attention

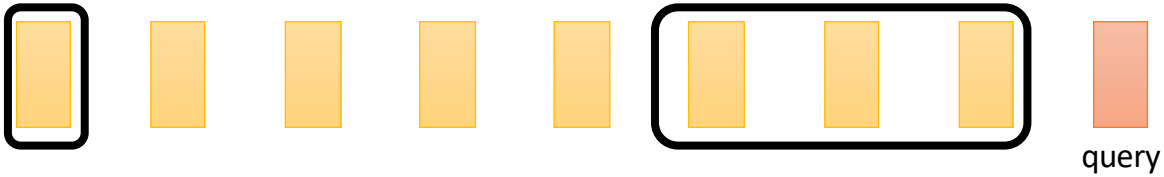


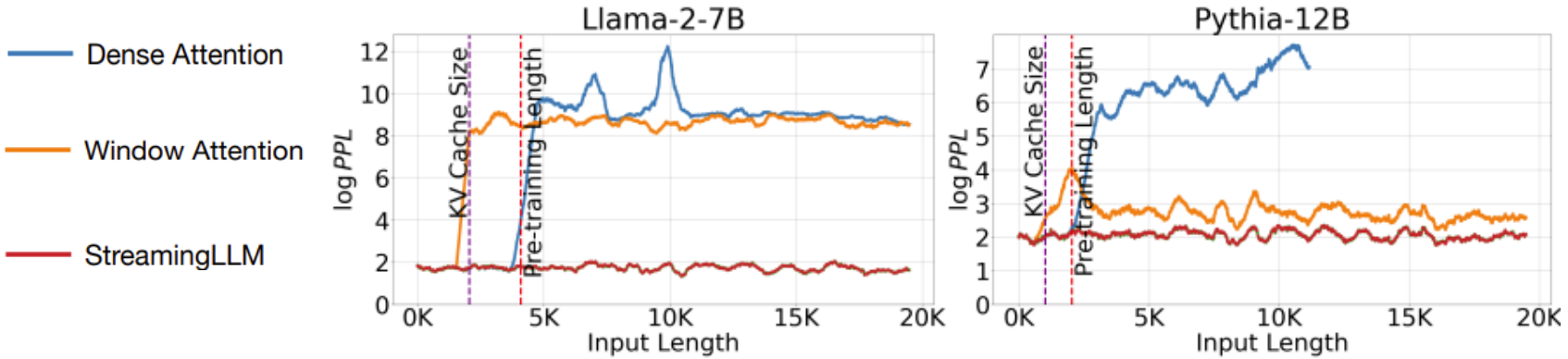
Mistral 7B

credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

StreamingLLM: keep the first tokens too

https://arxiv.org/abs/2309.17453





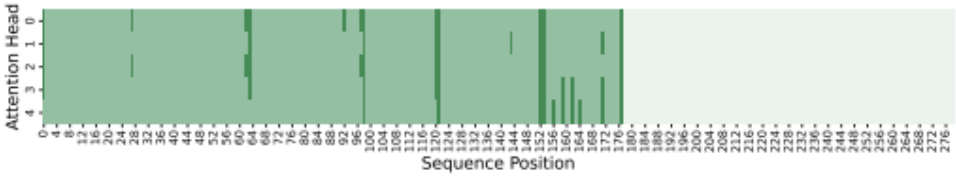
credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

Pruning the KV cache

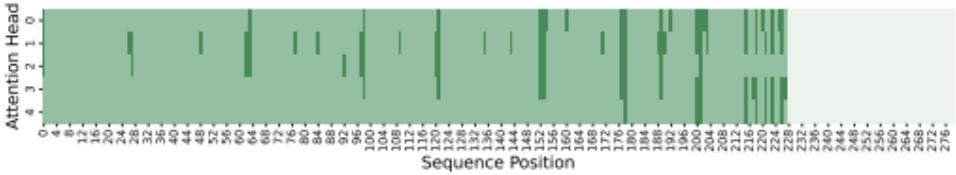
- Few Tokens get Attention
- Few tokens take most attention

Scissorhands: https://arxiv.org/abs/2305.17118

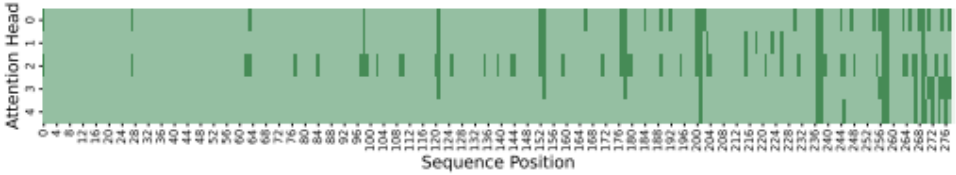
H2O: https://arxiv.org/abs/2306.14048



(a) Attention map at position 178



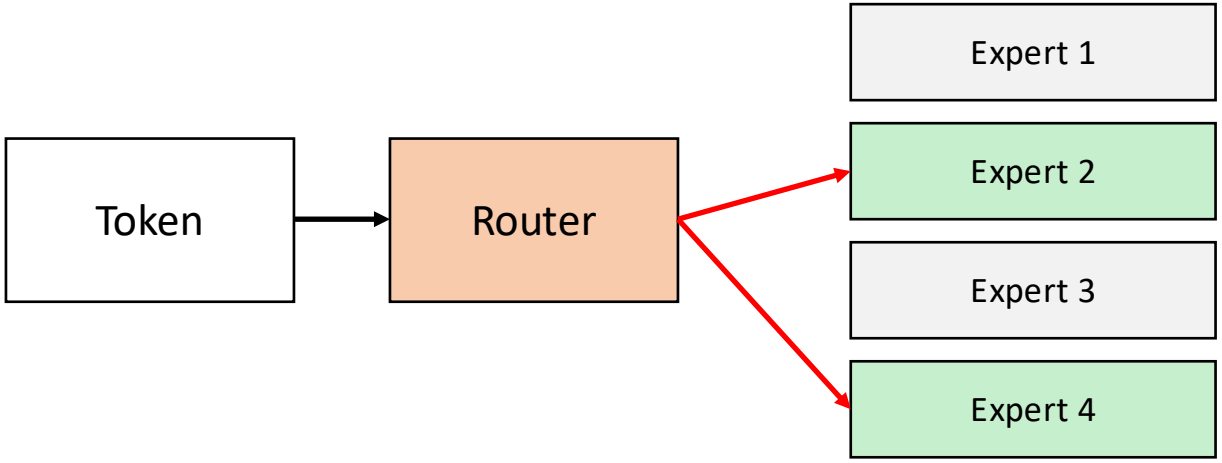
(b) Attention map at position 228



(c) Attention map at position 278

credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

Mixture-of-experts: compute only what you need



DeepSeek-V4-Pro	Parameters
Total (memory)	1.6 T
Active per token	49 B
Active share	3.1%

MoE cuts compute per token. It does not cut hosting memory

source: Shazeer et al. 2017; DeepSeek-V4-Pro model card, as of Oct 2026

Quiz examples: attention

1. Which method here needs no new quality evaluation: FlashAttention, GQA or a sliding window? Why?

2. A model uses 16 heads and a 2.1 GB cache per user. With 4 KV heads, what does the cache become?

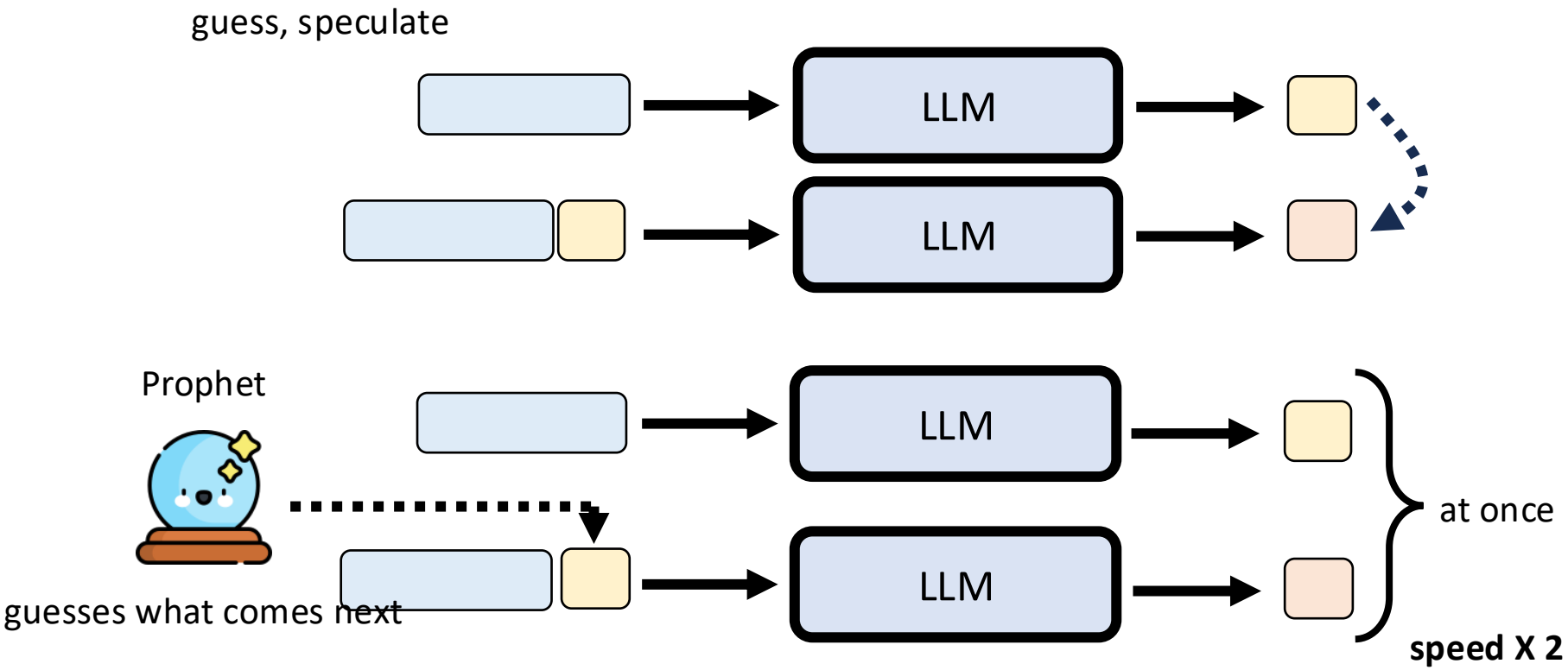
3. Your GPU cannot hold a model. Does switching to a MoE of the same total size help?

Roadmap : Today's Lecture

- Generating text
- Where the cost goes
- Cheaper attention
- **Faster decoding and serving**
- Choosing what to do

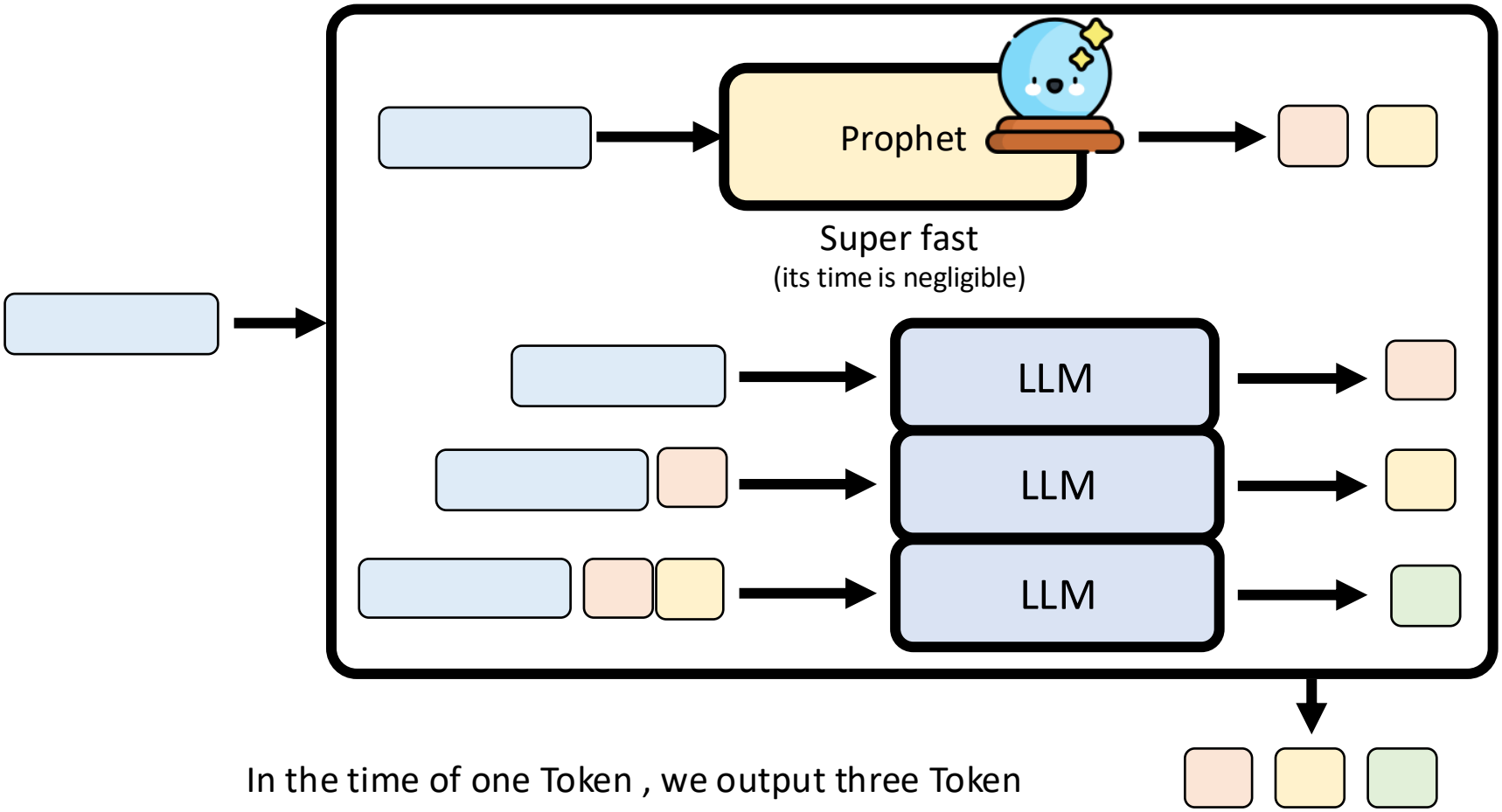
Speculative decoding: guess, then verify

https://arxiv.org/abs/2211.17192
https://arxiv.org/abs/2302.01318



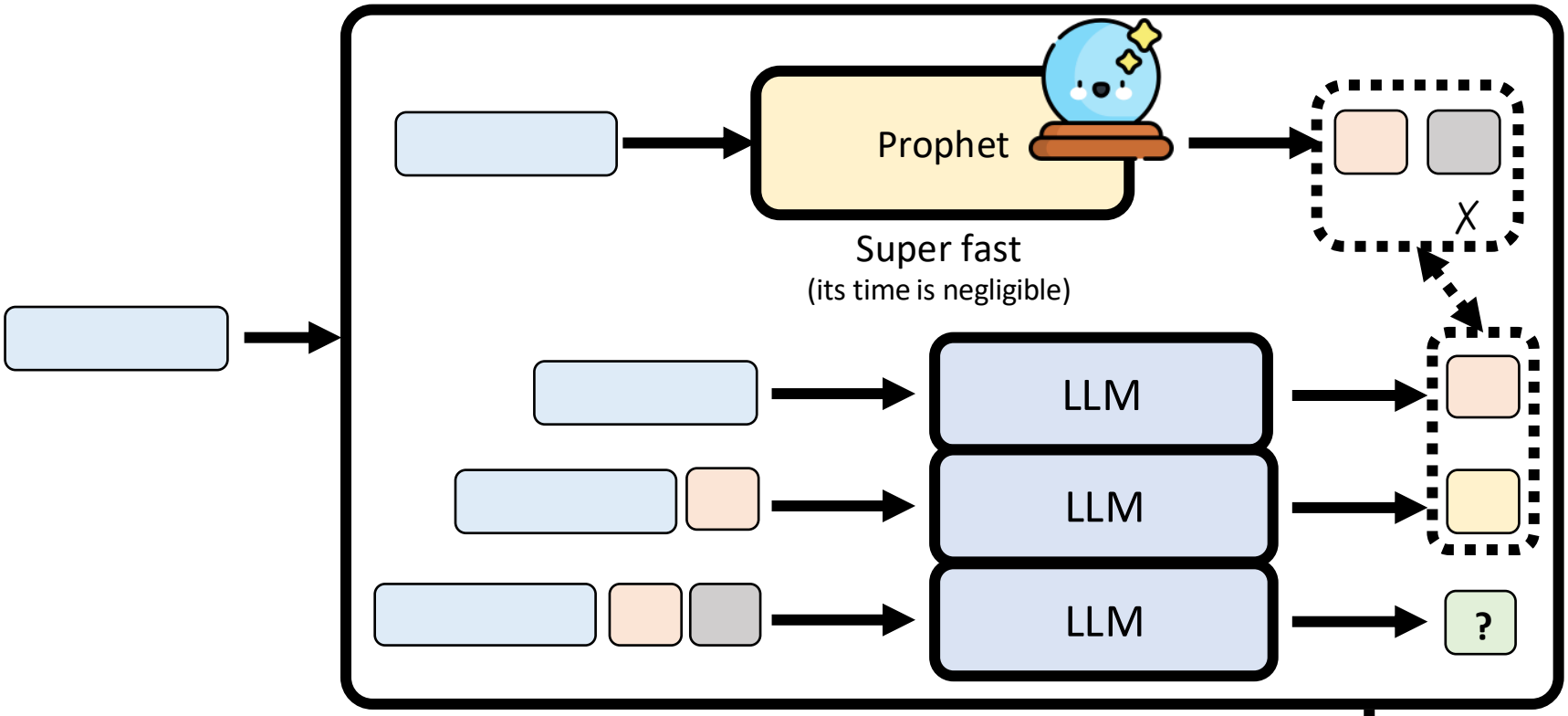
credit: Prof. Hung-yi Lee, NTU GenAI 2024 (generation strategies), translated

Speculative decoding: all guesses right



credit: Prof. Hung-yi Lee, NTU GenAI 2024 (generation strategies), translated

Speculative decoding: one guess wrong

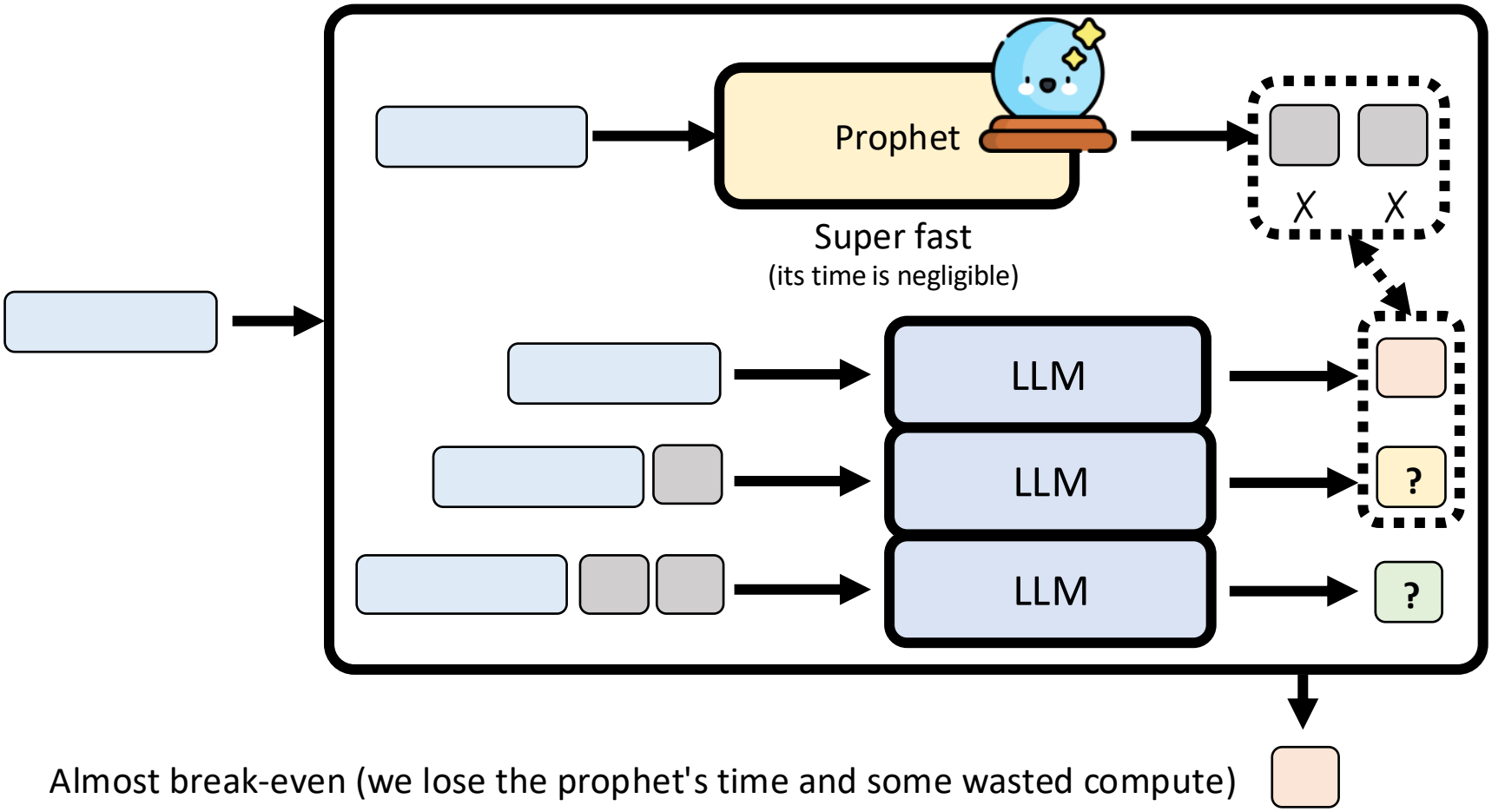


In the time of one Token , we output two Token

still a gain!

credit: Prof. Hung-yi Lee, NTU GenAI 2024 (generation strategies), translated

Speculative decoding: all guesses wrong



Almost break-even (we lose the prophet's time and some wasted compute)

credit: Prof. Hung-yi Lee, NTU GenAI 2024 (generation strategies), translated

Speculative decoding: when does it pay?

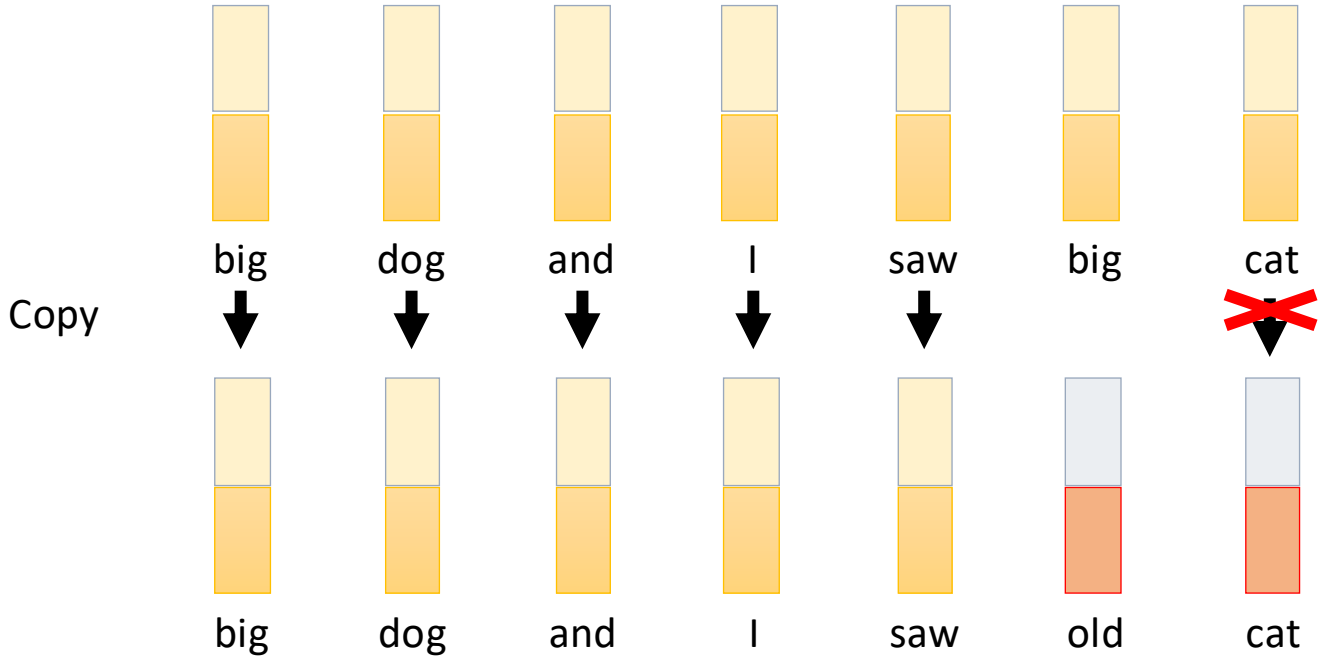
$$\mathbb{E}[\text{tokens per step}] = \frac{1 - \alpha^{\gamma+1}}{1 - \alpha}$$

Acceptance α	**0.5**	**0.8**	**0.9**
Tokens per step (γ = 4)	1.94	3.36	4.10

- Upper bound: ignores draft cost and rejected work
- Best with light load and predictable text
- Under load, speedups often shrink

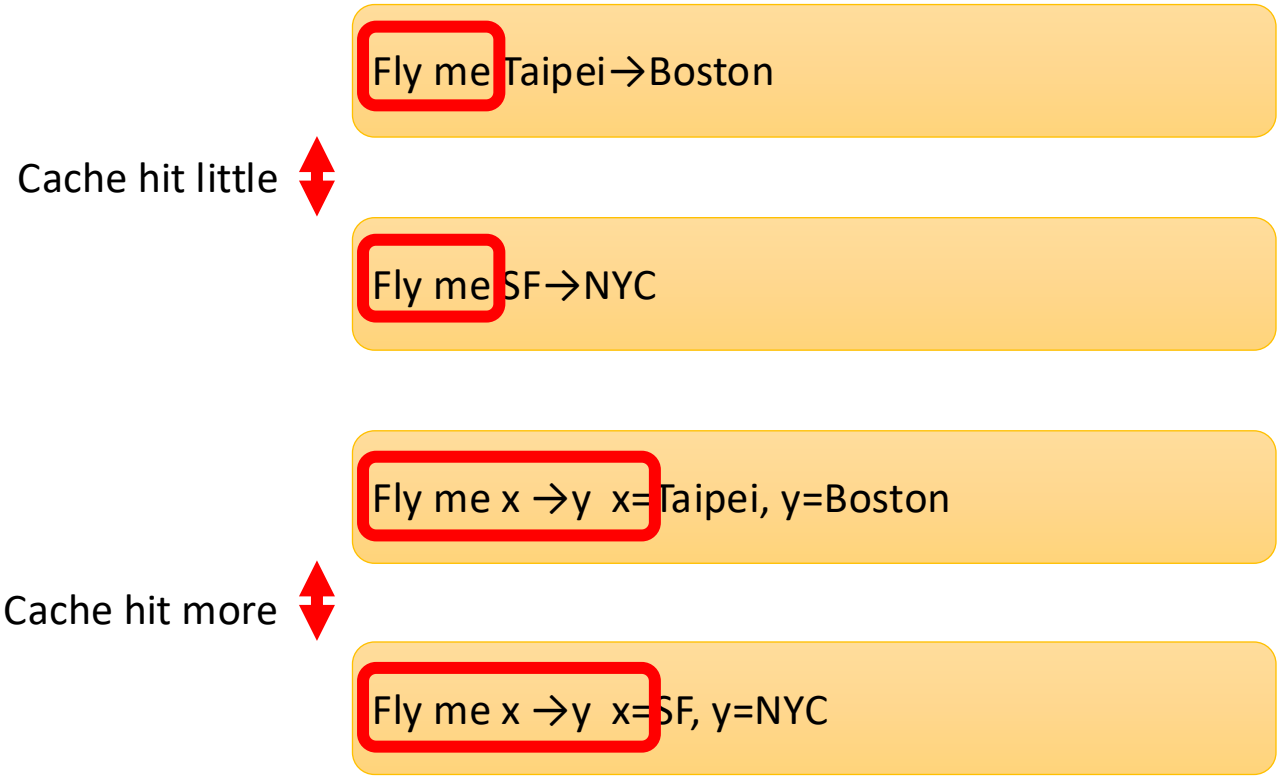
source: Leviathan et al., ICML 2023; Kong et al. 2026 (arXiv 2605.15051); computed

Prompt caching: reuse a shared prefix



credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

Prompt caching: put what changes last



credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

Prompt caching: providers price it

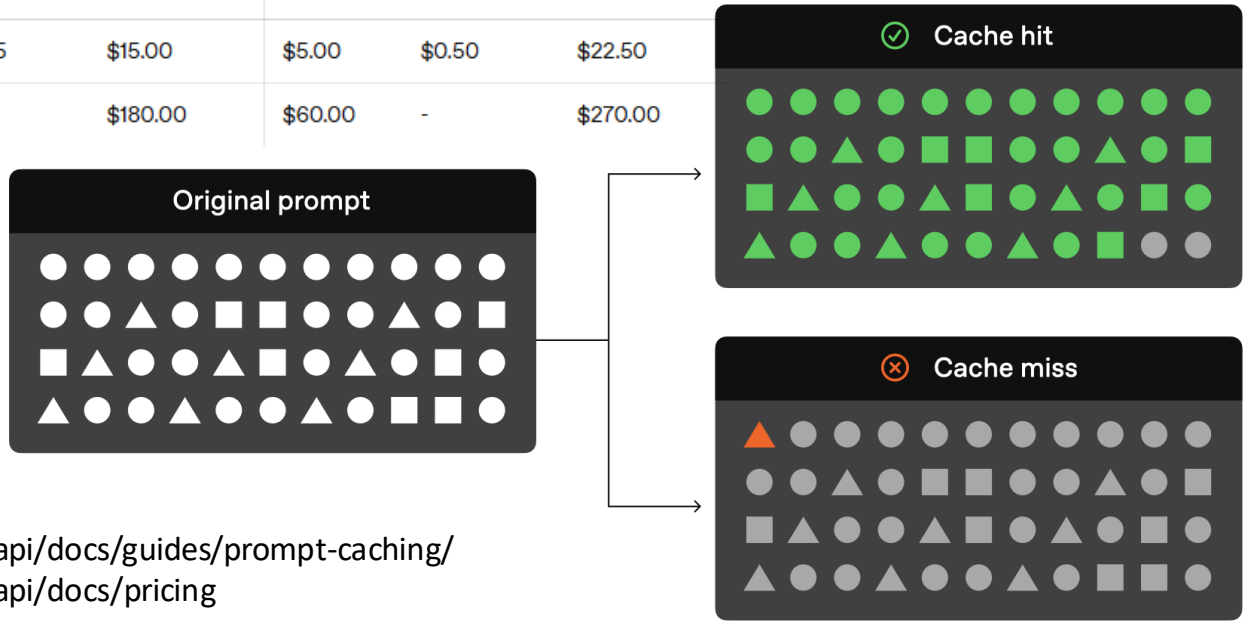
Text tokens

Standard | Batch | Flex | Priority

Prices per 1M tokens.

Our latest models

	Short context			Long context		
Model	**Input**	**Cached input**	**Output**	**Input**	**Cached input**	**Output**
gpt-5.4	$2.50	$0.25	$15.00	$5.00	$0.50	$22.50
gpt-5.4-pro	$30.00	-	$180.00	$60.00	-	$270.00



https://developers.openai.com/api/docs/guides/prompt-caching/
https://developers.openai.com/api/docs/pricing

credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

Continuous batching

Continuous batching: update batches after each decoding iteration, inserting new decode requests when one completes, thereby preventing GPU idle time

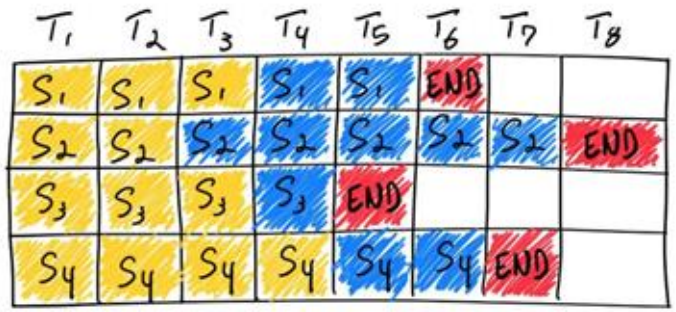
Addition considerations in scheduling:

- Prompt length, order of arrival
- Service Level Objective (SLO): acceptable latency targets for user-facing inference tasks such as chatbot responses or API completion requests

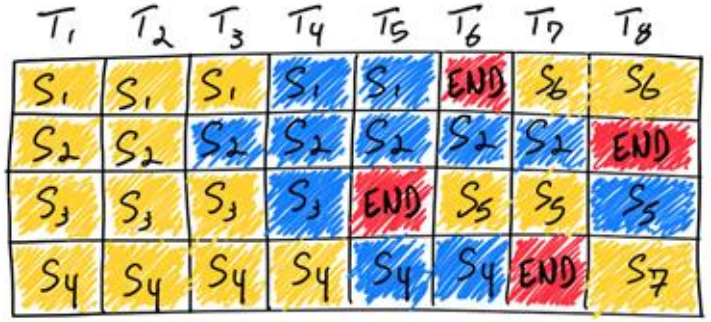
Legend:

- Yellow: prompt token
- Blue: generated token
- Red: end-of-sequence token

Static Batching



Continuous Batching



credit: UVA GenAI S26 student team (R. Khondaker); Miao et al. 2025 survey

Why the KV cache wastes memory

- Motivation: KV cache does not work well with current Mem management

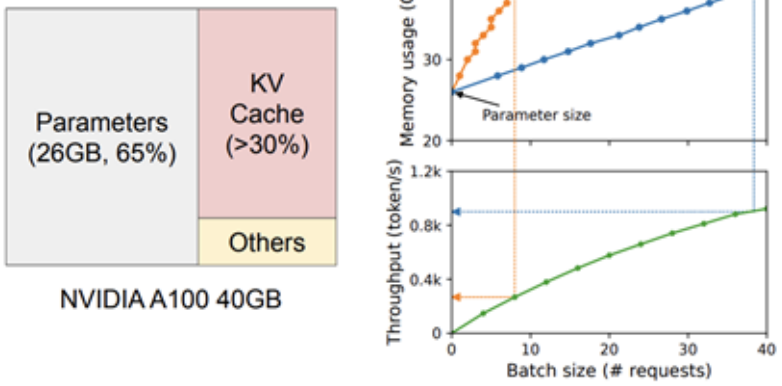


Figure 1. *Left:* Memory layout when serving an LLM with 13B parameters on NVIDIA A100. The parameters (gray) persist in GPU memory throughout serving. The memory for the KV cache (red) is (de)allocated per serving request. A small amount of memory (yellow) is used ephemerally for activation. *Right:* vLLM smooths out the rapid growth curve of KV cache memory seen in existing systems [31, 60], leading to a notable boost in serving throughput.

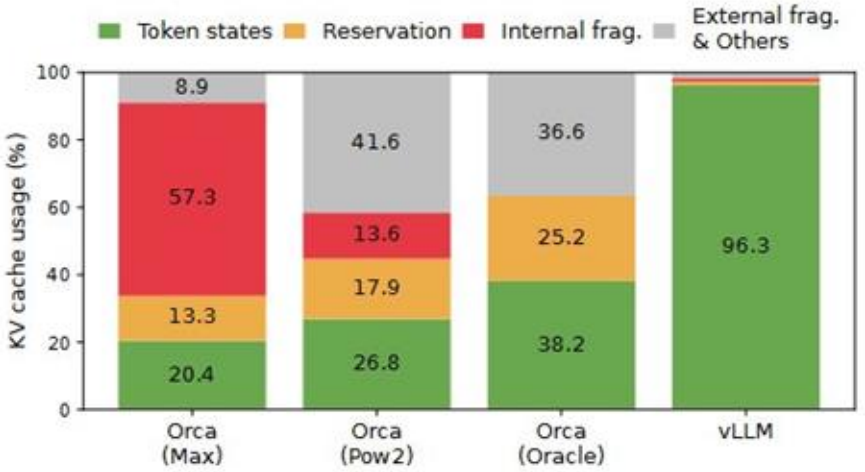


Figure 2. Average percentage of memory wastes in different LLM serving systems during the experiment in §6.2.

credit: UVA GenAI S24 (A. Benazir, Z. Wang, Tonmoy)

PagedAttention: a block table for the KV cache

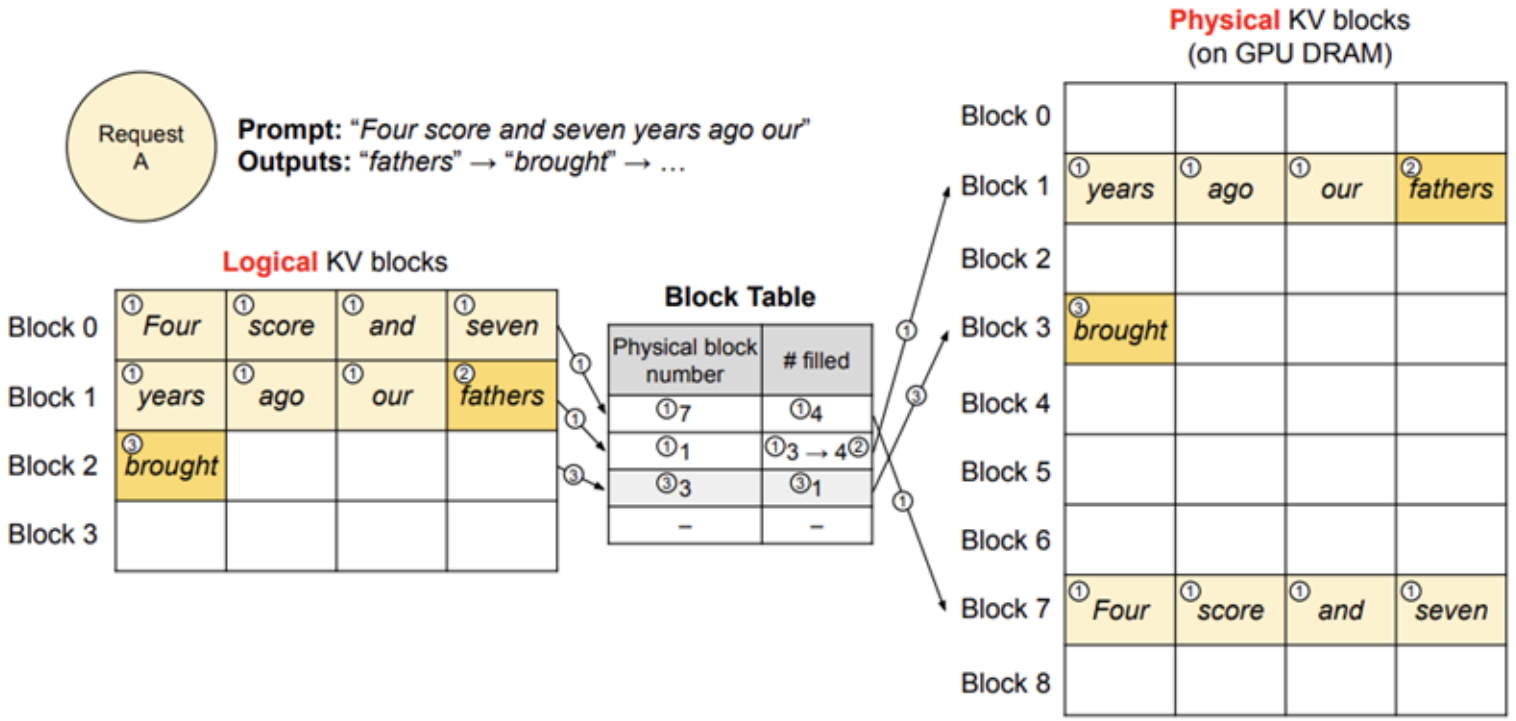


Figure 6. Block table translation in vLLM.

credit: UVA GenAI S24 (A. Benazir, Z. Wang, Tonmoy)

Quiz examples: diagnose the service

1. Text streams smoothly, then freezes for a second whenever a user pastes a long document. Why? What is the fix?

2. Every request begins with the same 3,000-token prompt, except for a timestamp on line 1. What is wasted, and what is the fix?

3. Speculative decoding helped in testing, but production got slower at peak. Why?

Roadmap : Today's Lecture

- Generating text
- Where the cost goes
- Cheaper attention
- Faster decoding and serving
- **Choosing what to do**

Summary: the classic speed-ups

	What it does	Changes Attention	Train?
Flash Attention	moves less data	X	X
KV Cache	stores the computed key and value	X	X
Multi-query attention	several query share key and value	O	O
Group-query attention		O	O
Multi-head Latent Attention	compresses key and value	O	O
Sliding Window Attention	changes the Attention range	O	?
Streaming LLM		O	?
Pruning KV Cache	drops key and value	O	X
Speculative Decoding	a small model guesses the output	X (in theory)	X

credit: Prof. Hung-yi Lee, NTU ML 2026 (LLM inference), translated

Which trick first?

Symptom	Try first
Long wait for the first word	Prefix caching, chunked prefill
Words arrive slowly	Quantize, smaller model, speculative decoding (light load)
GPU sits idle	Continuous batching
Out of memory	Quantize, GQA, paged KV cache
Bill too high	Distill; or use an API at low volume

Measure first: TTFT, TPOT, throughput, utilization

Takeaways

- Generation is a loop: last position → softmax → greedy, sample or beam → append
- Prefill reads the prompt in one pass (compute-bound); decode adds one token per pass (memory-bound)
- The KV cache trades memory for speed; it, not the weights, often limits how many users fit
- FlashAttention: same math, tiles + running softmax, less memory traffic
- GQA / MLA / windows shrink the cache; they change the model
- Speculative decoding, prompt caching, continuous batching, paged memory: same answers, more tokens per GPU

Where this leads: prompting, retrieval (RAG) and test-time compute

Summary of Today: LLM inference

- **Generating text:** the loop; greedy, sampling, temperature, beam search
- **Where the cost goes:** prefill vs decode; the KV cache; memory and speed limits
- **Cheaper attention:** FlashAttention; sharing or compressing K and V; windows
- **Faster decoding and serving:** speculative decoding; prompt caching; batching; paged memory
- **Choosing what to do:** which trick first, and what each one costs

Same trained network throughout: no gradients today, only forward passes, done fast and cheaply.

References

- Prof. Hung-yi Lee, NTU ML 2026: inference (FlashAttention, KV cache, GQA/MLA, prompt caching); NTU GenAI 2024: generation strategies
- Vaswani et al., “Attention Is All You Need”, NeurIPS 2017
- Holtzman et al., “The Curious Case of Neural Text Degeneration”, ICLR 2020
- Dao et al., “FlashAttention”, NeurIPS 2022
- Shazeer, Multi-query attention, 2019; Ainslie et al., “GQA”, EMNLP 2023; DeepSeek-AI, DeepSeek-V2 (MLA), 2024
- Beltagy et al., Longformer, 2020; Jiang et al., Mistral 7B, 2023; Xiao et al., StreamingLLM, ICLR 2024
- Liu et al., Scissorhands, NeurIPS 2023; Zhang et al., H2O, NeurIPS 2023
- Leviathan et al., Speculative Decoding, ICML 2023; Chen et al., Speculative Sampling, 2023
- Yu et al., Orca, OSDI 2022; Kwon et al., vLLM / PagedAttention, SOSP 2023
- Agrawal et al., Sarathi-Serve, OSDI 2024; Zheng et al., SGLang (prefix caching), 2024
- Gemma Team, Gemma 2, 2024 (model sizes)
- Courses: CMU ANLP S26 L09 Decoding; Stanford CS336 L10 Inference; UCSD CSE291A F26 L02; UW CSE599K 24au Serving
- S. Raschka, Build a Large Language Model (From Scratch), 2024; A. Karpathy, nanoGPT
- UVA GenAI S24/S26 student decks (credited on each slide)

EXTRA

Decode Stage has a speed limit

$$\text{tokens/s} \lesssim \frac{\text{memory bandwidth}}{\text{model size in bytes}}$$

7B model	Size	Upper bound at 3 TB/s
16-bit	14 GB	≈ 214 tokens/s
8-bit	7 GB	≈ 429 tokens/s
4-bit	3.5 GB	≈ 857 tokens/s

Fewer bytes to move means faster decode, not only a smaller file

source: illustrative bandwidth; rough upper bound