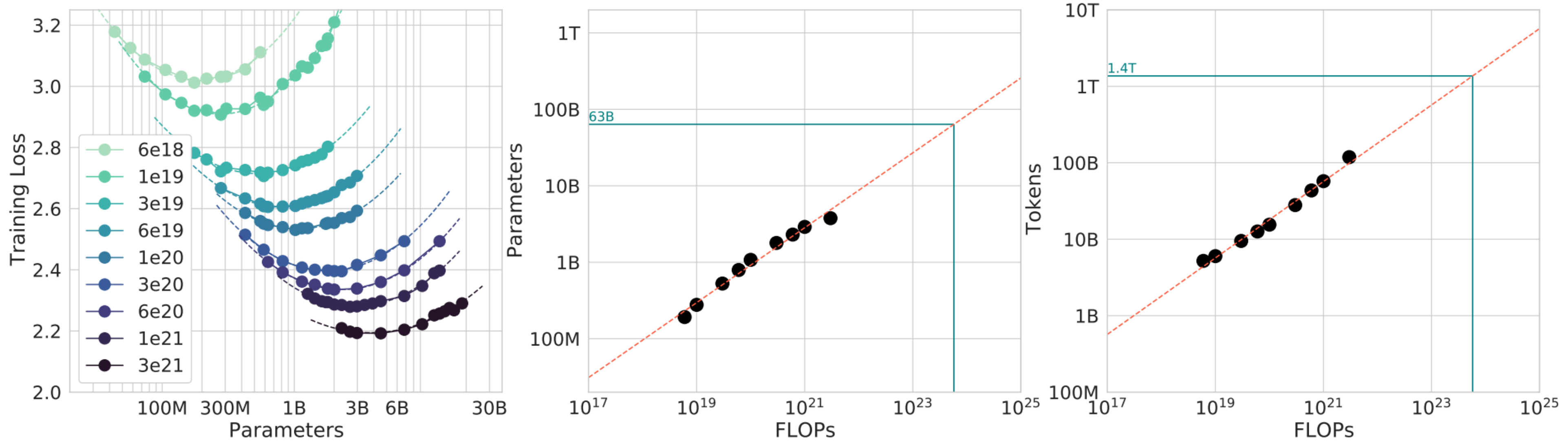
Course 5: Language Models at Inference Time

Introduction

Background

Scaling language models (LMs) is the go-to solution to achieve greater performance [1].



Background

- Evidently, the more you scale, the more compute you need at inference.
- Hardware cost can hinder LLMs useless if no optimization is done.
- Not all optimization techniques are born equal...

What are the different responses to the trade-off between an LLM performance and an LLM througput?

Content

More About Throughput?

Prompt pruning: when KV caching is not enough

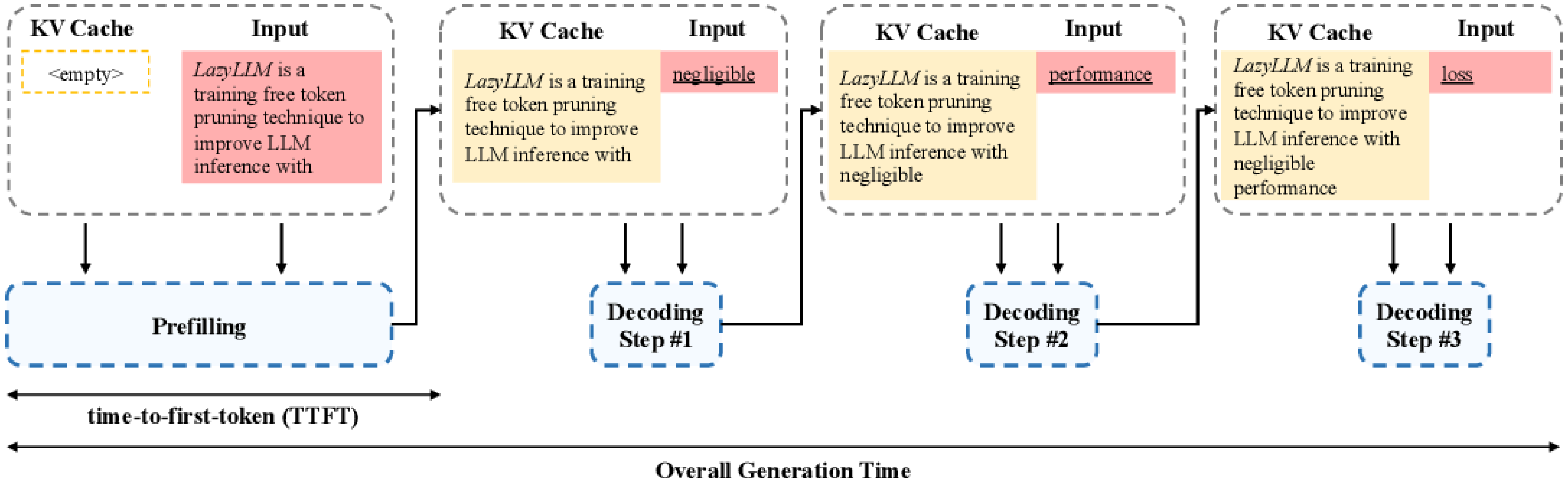
Attention matrices need to be calculated for every token constituing an LLM's prompt, leading to latency.

- On LLaMa2-70b models, given a long prompt, 23% of the total generation time is accounted for the time to first token (TTFT).
- KV caching is of no-use in that context...

How to reduce that TTFT with minimum performance loss?

Prompt pruning: when KV caching is not enough

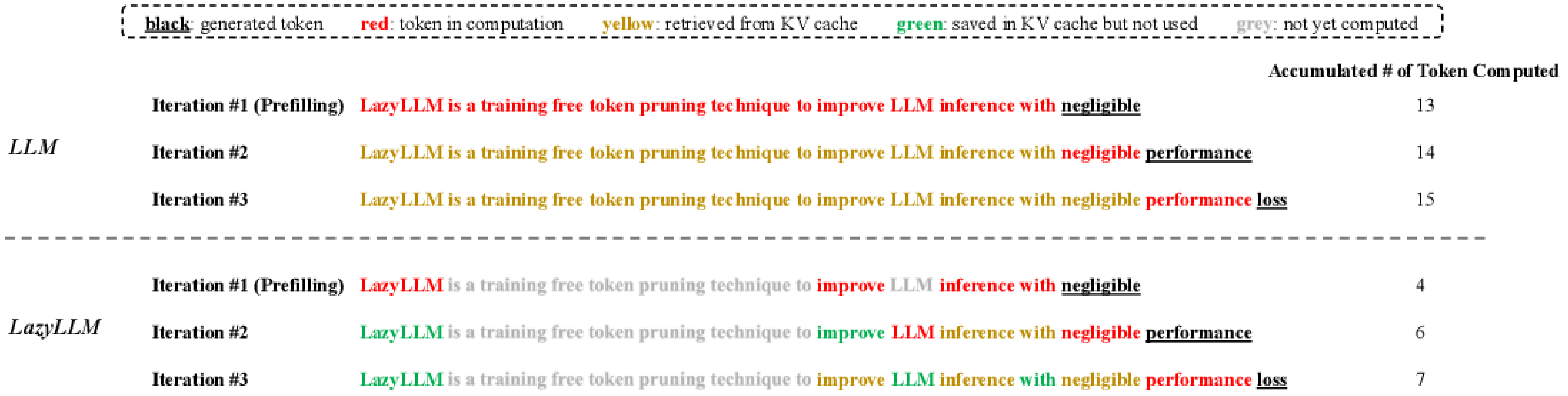
When does KV cachin comes into play?


KV Cache
Input
<empty>
LazyLLM is a training free token pruning technique to improve LLM inference with
Prefilling
KV Cache
Input
LazyLLM is a training free token pruning technique to improve LLM inference with
negligible
Decoding Step #1
KV Cache
Input
LazyLLM is a training free token pruning technique to improve LLM inference with negligible
performance
Decoding Step #2
KV Cache
Input
LazyLLM is a training free token pruning technique to improve LLM inference with negligible performance
loss
Decoding Step #3
time-to-first-token (TTFT)
Overall Generation Time

The above example assume that your model is aware of LazyLLM [2] via its training data.

Prompt pruning: when KV caching is not enough

Not all tokens are useful to understand/answer the prompt.



black: generated token red: token in computation yellow: retrieved from KV cache green: saved in KV cache but not used grey: not yet computed

			Accumulated # of Token Computed
LLM	Iteration #1 (Prefilling)	LazyLLM is a training free token pruning technique to improve LLM inference with negligible	13
	Iteration #2	LazyLLM is a training free token pruning technique to improve LLM inference with negligible performance	14
	Iteration #3	LazyLLM is a training free token pruning technique to improve LLM inference with negligible performance loss	15
LazyLLM	Iteration #1 (Prefilling)	LazyLLM is a training free token pruning technique to improve LLM inference with negligible	4
	Iteration #2	LazyLLM is a training free token pruning technique to improve LLM inference with negligible performance	6
	Iteration #3	LazyLLM is a training free token pruning technique to improve LLM inference with negligible performance loss	7

Prompt pruning: when KV caching is not enough

How to effectively choose tokens to prune out?

Transformer's attention represent more abstract concept as the compution is done deeper in its layers [3].

The last attention matrices play an important role in the decision boundaries computed by a transformer-based LM [4].

Prompt pruning: when KV caching is not enough

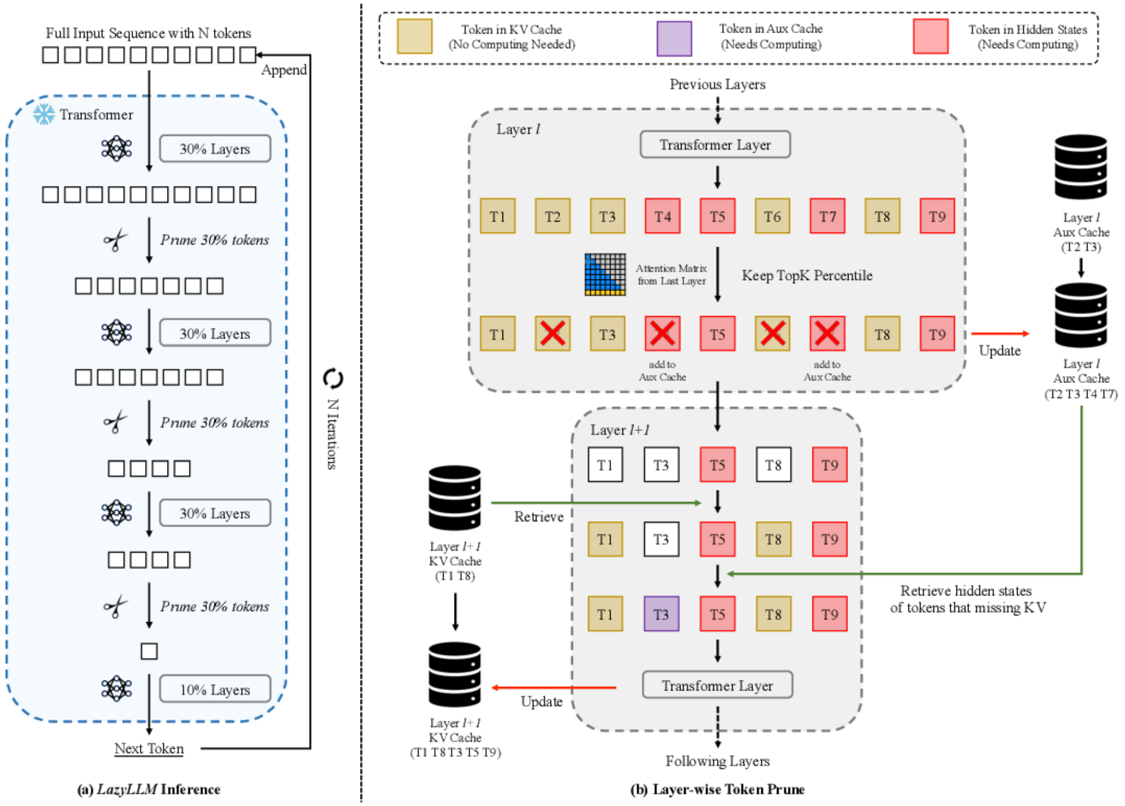
For a given token i, the attention matrix compute the probability of a token $j \leq N$ attending to i accross all H attention heads of a model. This process is repeated accross the $l \leq L$ layers of a model.

The importance of an input token i, at a given layer l can now be computed as

$$s_i^l = \frac{1}{H} \sum_{h=1}^{H} \sum_{j=1}^{N} A_{h,i,j}^l$$

Prompt pruning: when KV caching is not enough

We do not want to have too few tokens and some of them can become relevant later in the decoding process



Full Input Sequence with N tokens
Append
Transformer
30% Layers
Prune 30% tokens
30% Layers
Prune 30% tokens
30% Layers
Prune 30% tokens
10% Layers
Next Token
N Iterations
(a) LazyLLM Inference
Token in KV Cache (No Computing Needed)
Token in Aux Cache (Needs Computing)
Token in Hidden States (Needs Computing)
Previous Layers
Layer l
Transformer Layer
T1 T2 T3 T4 T5 T6 T7 T8 T9
Attention Matrix from Last Layer
Keep TopK Percentile
add to Aux Cache
Update
Layer l Aux Cache (T2 T3)
Layer l Aux Cache (T2 T3 T4 T7)
Layer l+1
Retrieve
Layer l+1 KV Cache (T1 T8)
Retrieve hidden states of tokens that missing KV
Transformer Layer
Update
Layer l+1 KV Cache (T1 T8 T3 T5 T9)
Following Layers
(b) Layer-wise Token Prune

Prompt pruning: when KV caching is not enough

Drawbacks:

- Marginal gain in performance with relatively short prompts.
- Drop in performance in code completion (no stop-words to drop?).

Speculative decoding

An **LLM** can **predict multiple tokens in a single forward pass** :

- **Speculative decoding** [5] allow an LLM to **"guess" future tokens** while generating curent tokens, **all within a single forward pass**.
- By running a draft model to predict multiple tokens, the main model (larger) only has to verify the predicted tokens for "correctness".

Speculative decoding

1. **Prefix**: [BOS]
2. **Assistant**: [BOS] The quick brown sock jumps
3. **Main**: [BOS] The quick brown fox / sock jumps
4. **Assistant**: [BOS] The quick brown fox jumps over the crazy dog
5. **Main**: The quick brown jumps over the lazy / crazy dog
6. …

Speculative decoding

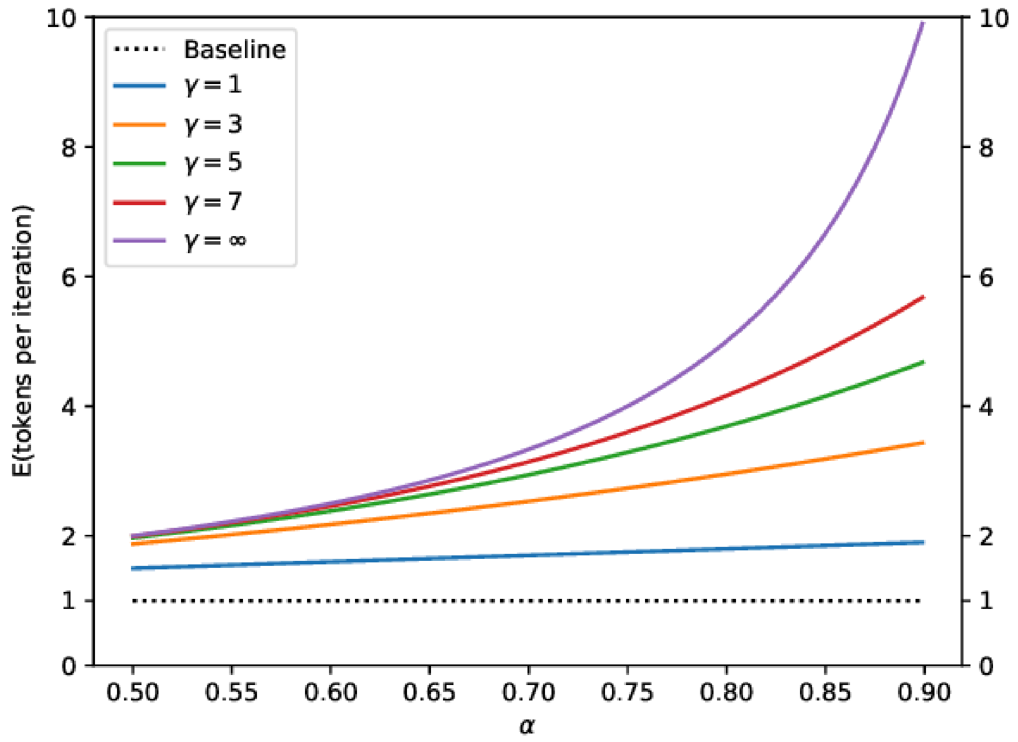
The main model just verifies that the distribution $q(x)$, computed by the assistant is not to far from the distribution $p(x)$ it computes within a forward pass.

The expected number of tokens generated within one looop of speculative decoding can be theorithically formulated as:

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

Which is # forward passes' reduction factor.

Speculative decoding



The expected number of tokens generated via speculative decoding as a function of α for various values of γ.

Speculative decoding

In order **to take the most out of speculative decoding**, the distance between $q(x)$ **and** $p(x)$ **need to be minimal**.

How to reduce the distance between $q(x)$ and $p(x)$ when the assistance model is smaller?

- Quantization
- Distillation
- Over-training on the same dataset as the main model

Layer skip: self speculative decoding

Speculative decoding comes with two inconvenients:

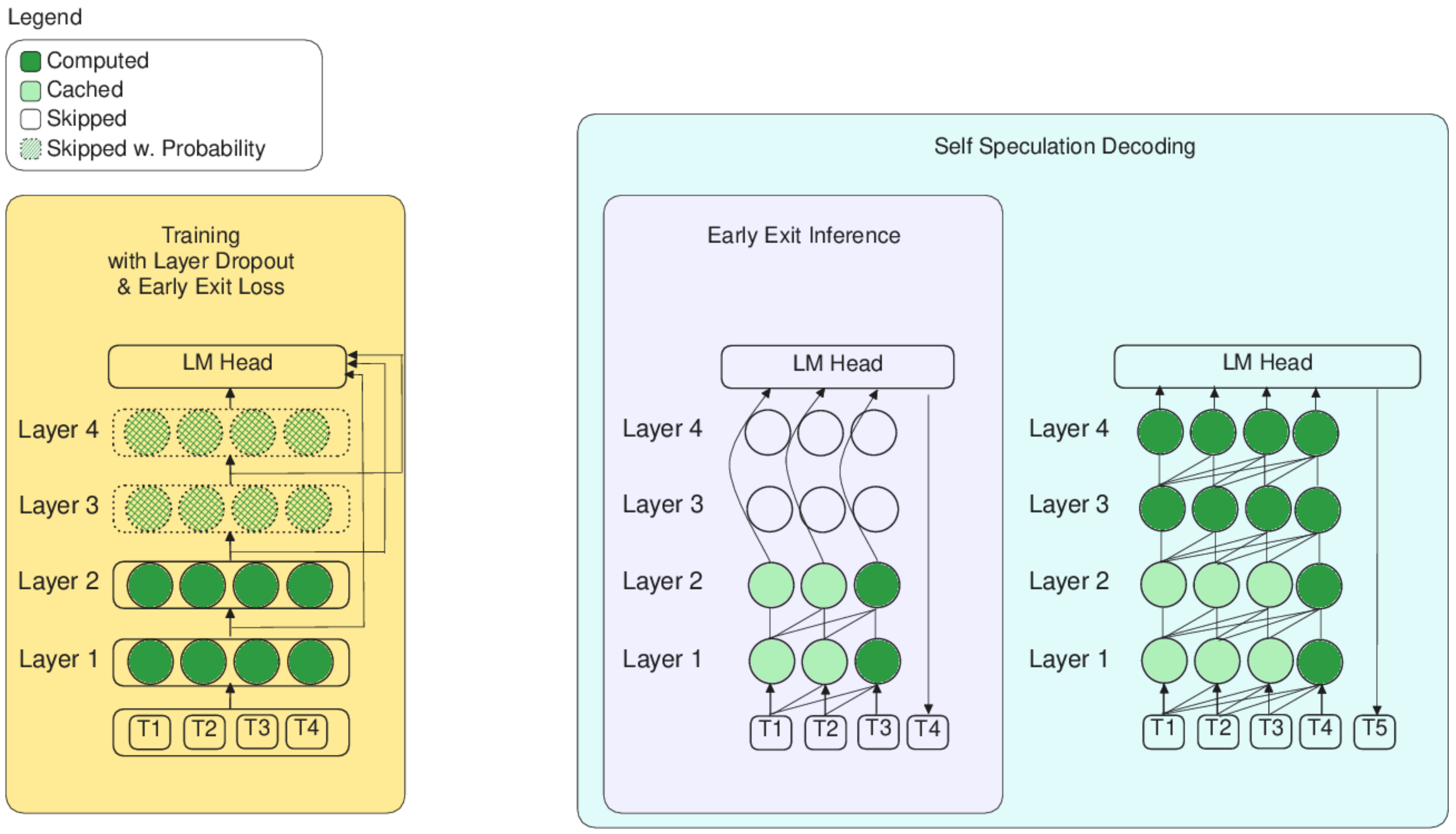
- Loading two models in memory
- Making sure the assistant model output a token distribution as close as possible to the main model

Layer skip: self speculative decoding

Why not let the main model do the speculation itself?

Transformer models are believed to be **over-parametrized** and the **last layers specialized** on computing the decision boundaries **before projecting on the LM head**. Maybe we can make **each layer able to project on the LM head**, thus skipping layers [6] and allowing for an **early exit** at inference [7].

Layer skip: self speculative decoding



Train using Layer Dropout + Early Exit Loss....

... enables inference with subset of layers with higher accuracy...

... and we can improve accuracy by verifying and correcting with remaining layers

Layer skip: self speculative decoding

The hidden state of a token t, at layer $l+1$ is stochastically given by

$$x_{l+1,t} = x_{l,t} + M(p_{l,t}) \times f_l(x_{l,t})$$

Where M is a masking function with a probability of skipping

$$p_{l,t} = S(t) \times D(l) \times p_{max}$$

$$D(l) = e^{\frac{l \times ln(2)}{L-1}}$$

$$S(t) = e^{\frac{t \times ln(2)}{T-1}}$$

Layer skip: self speculative decoding

How is the loss computed?

$$\mathcal{L}_{total} = \sum_{l=0}^{l=L-1} \tilde{e}(t,l) \times \mathcal{L}_{CE}$$

Where $\tilde{e}(t,l)$ is a normalized per-layer loss scale

$$\tilde{e}(t,l) = \frac{C(t,l) \times e(l)}{\sum_{i=0}^{i=L+1} C(t,i) \times e(i)}$$

Layer skip: self speculative decoding

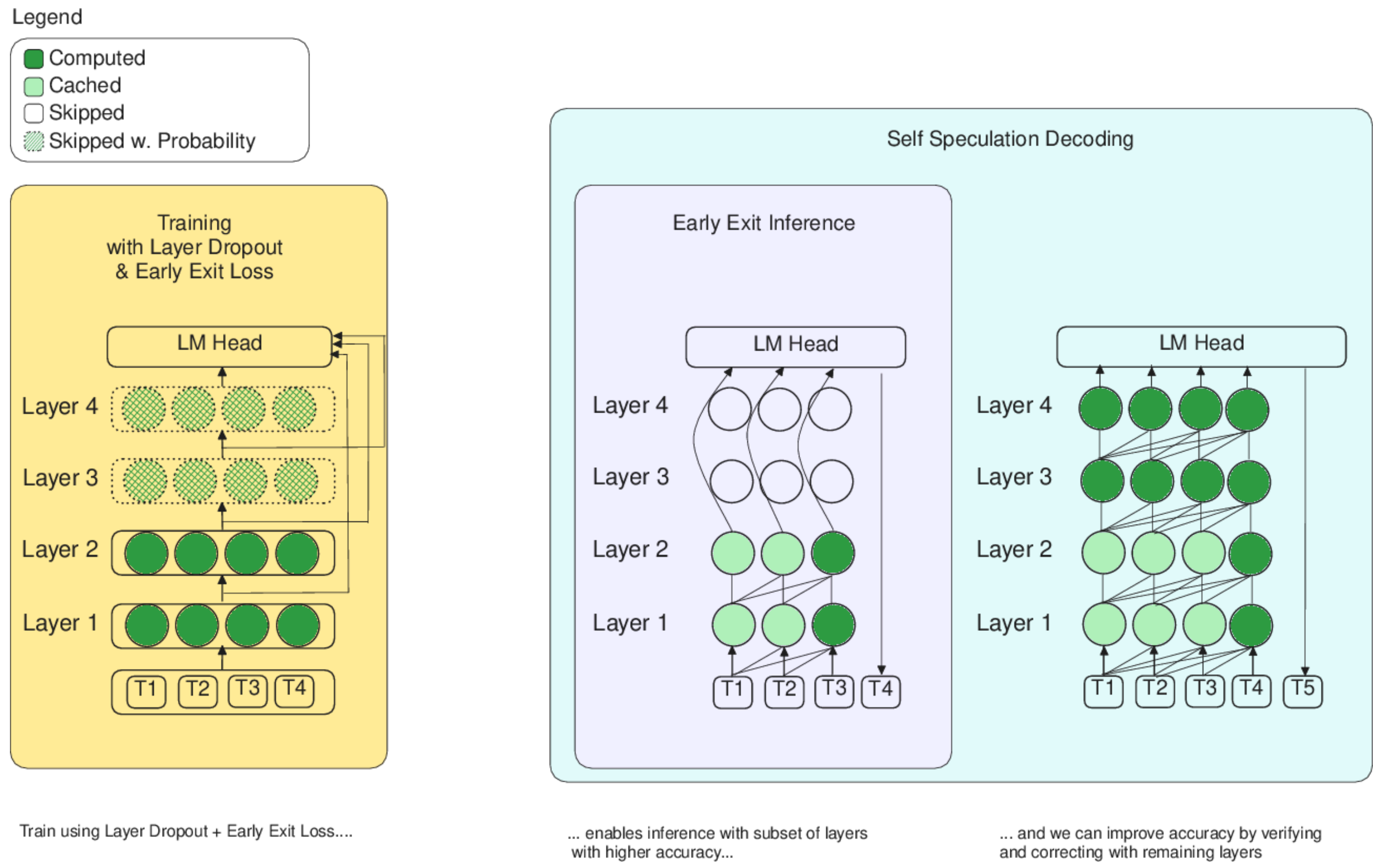
$$C(t,l) = \begin{cases} 1 & \text{if there is no ealr exit at layer } l \\ 0 & \text{otherwise} \end{cases}$$

e is a scale that increases across layers, penalizing later layers, as predicting in later layers is easier.

$$e(l) = \begin{cases} \sum_{i=0}^{i=l} i & \text{if } 0 \leq l \leq L-1 \\ L - 1 + \sum_{i=0}^{i=L-2} i & \text{if } l = L-1 \end{cases}$$

Layer skip: self speculative decoding

How does this change inference?



Train using Layer Dropout + Early Exit Loss....

... enables inference with subset of layers with higher accuracy...

... and we can improve accuracy by verifying and correcting with remaining layers

Layer skip: self speculative decoding

- 10% speed-up
- A single KV cache => low memory overhead
- The main model is still competitive when the last transformer layer is used for prediction despite a different training technique.

More About Performance?

Retrieval augmented generation (at inference)

References

[1] Hoffmann, Jordan, et al. "Training compute-optimal large language models." arXiv preprint arXiv:2203.15556 (2022).

[2] Fu, Qichen, et al. "Lazyllm: Dynamic token pruning for efficient long context llm inference." arXiv preprint arXiv:2407.14057 (2024).

[3] Jawahar, Ganesh, Benoît Sagot, and Djamé Seddah. "What does BERT learn about the structure of language?." ACL 2019-57th Annual Meeting of the Association for Computational Linguistics. 2019.

[4] Chung, Hyung Won, et al. "Rethinking embedding coupling in pre-trained language models." arXiv preprint arXiv:2010.12821 (2020).

[5] Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." International Conference on Machine Learning. PMLR, 2023.

[6] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[7] Elhoushi, Mostafa, et al. "Layer skip: Enabling early exit inference and self-speculative decoding." arXiv preprint arXiv:2404.16710 (2024).