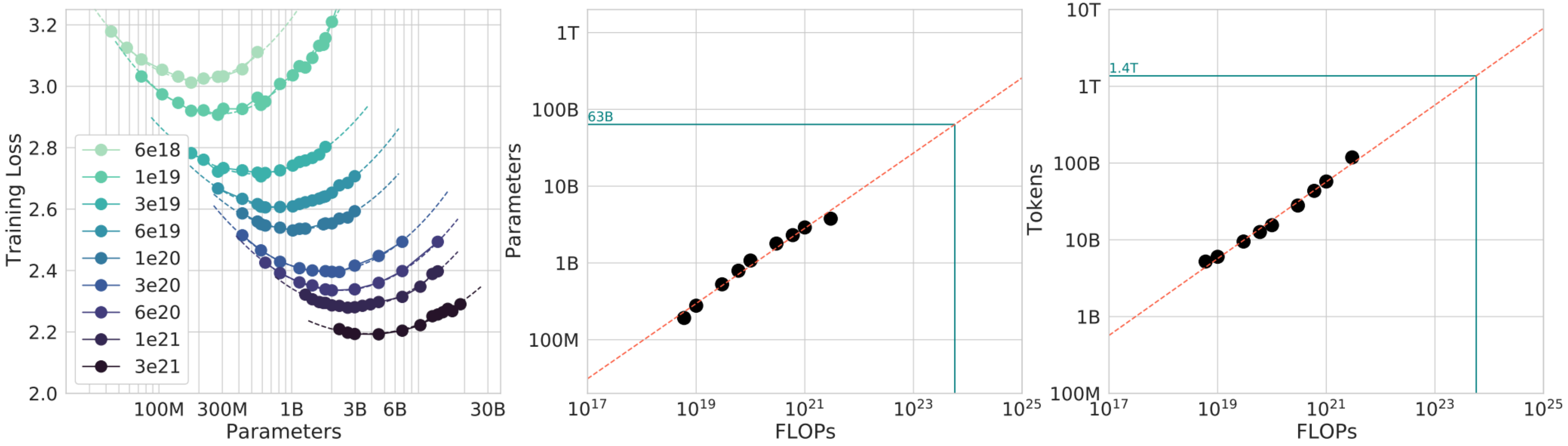
Course 5: Language Models at Inference Time

Introduction

Background

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



Background

- The more you scale, the more compute you need at inference.
- Hardware costs can hinder LLMs 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

1. **More About Throughput?**
    a. Prompt pruning, when KV caching is not enough
    b. Speculative decoding
    c. Layer skip: self speculative decoding
2. **More About Performance?**
    a. Retrieval augmented generation (at inference)
    b. Test-time compute
3. **More About "Balance"?**
    a. Mixture of experts

More About Throughput?

Prompt pruning: when KV caching is not enough

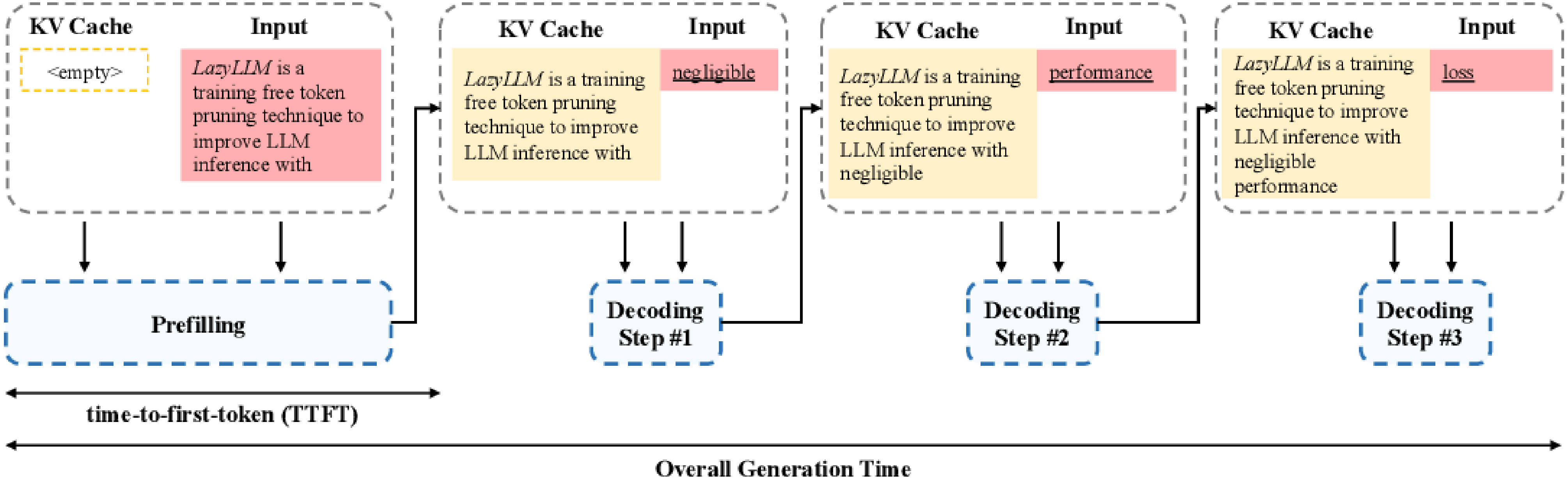
Attention matrices need to be calculated for every token constituting an LLM's prompt, leading to latency.

- 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 caching comes into play?



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.

Prompt pruning: when KV caching is not enough

How to effectively choose tokens to prune out?

Transformer's attention represents more abstract concept as the computation 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

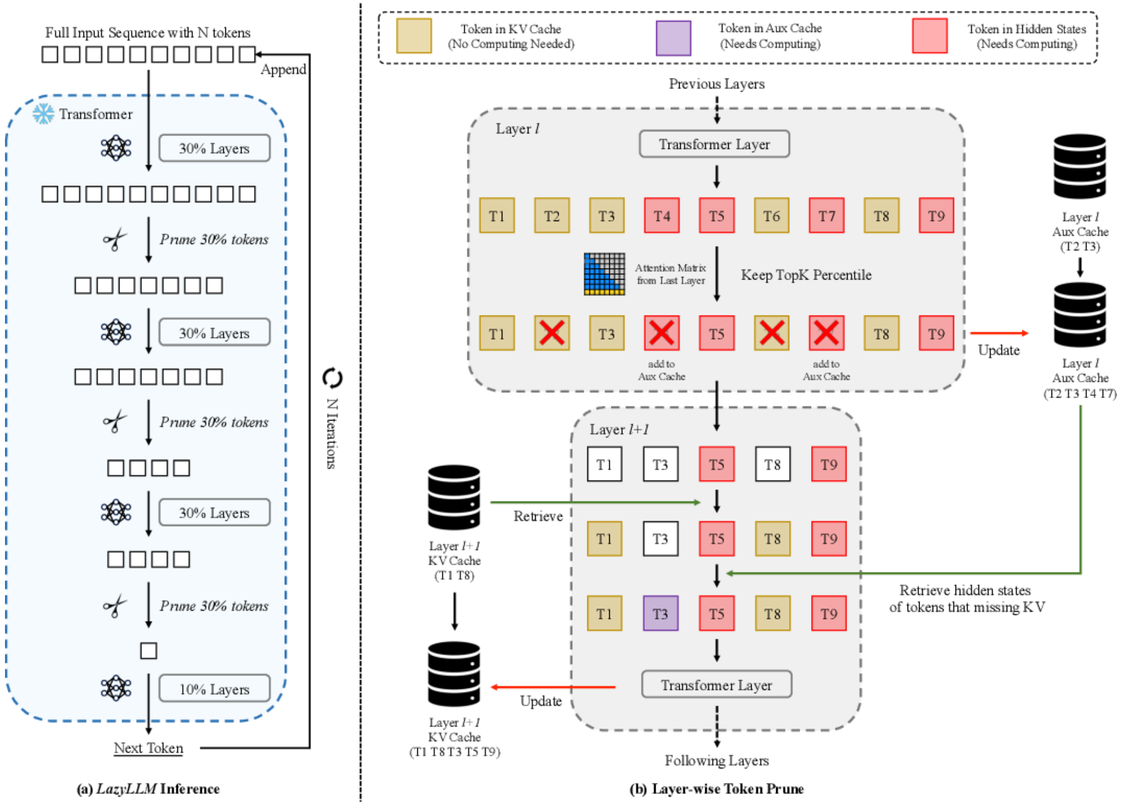
For a given token i, the attention matrix compute the probability of a token $j \leq N$ attending to i accross all H attention heads of a model. This process is repeated accross the $l \leq L$ layers of a model.

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



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] allows an LLM to **"guess" future tokens** while generating current 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

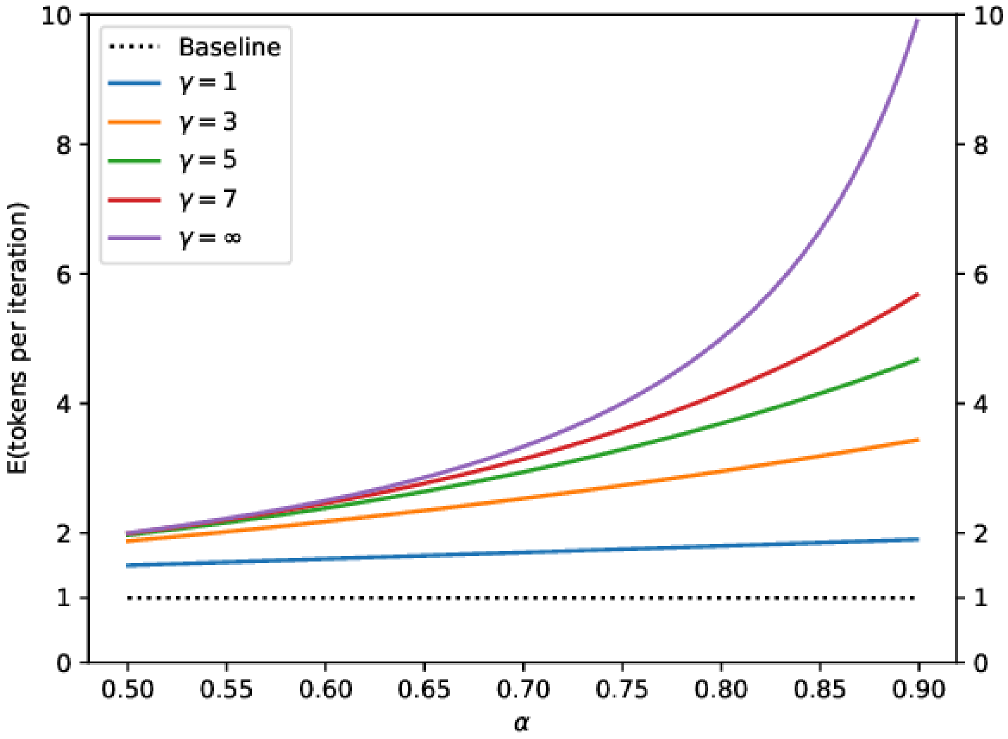
The main model just verifies that the distribution $q(x)$, computed by the assistant is not too far from the distribution $p(x)$ it computes within a forward pass.

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 the 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** $p(x)$ **needs 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 inconveniences:

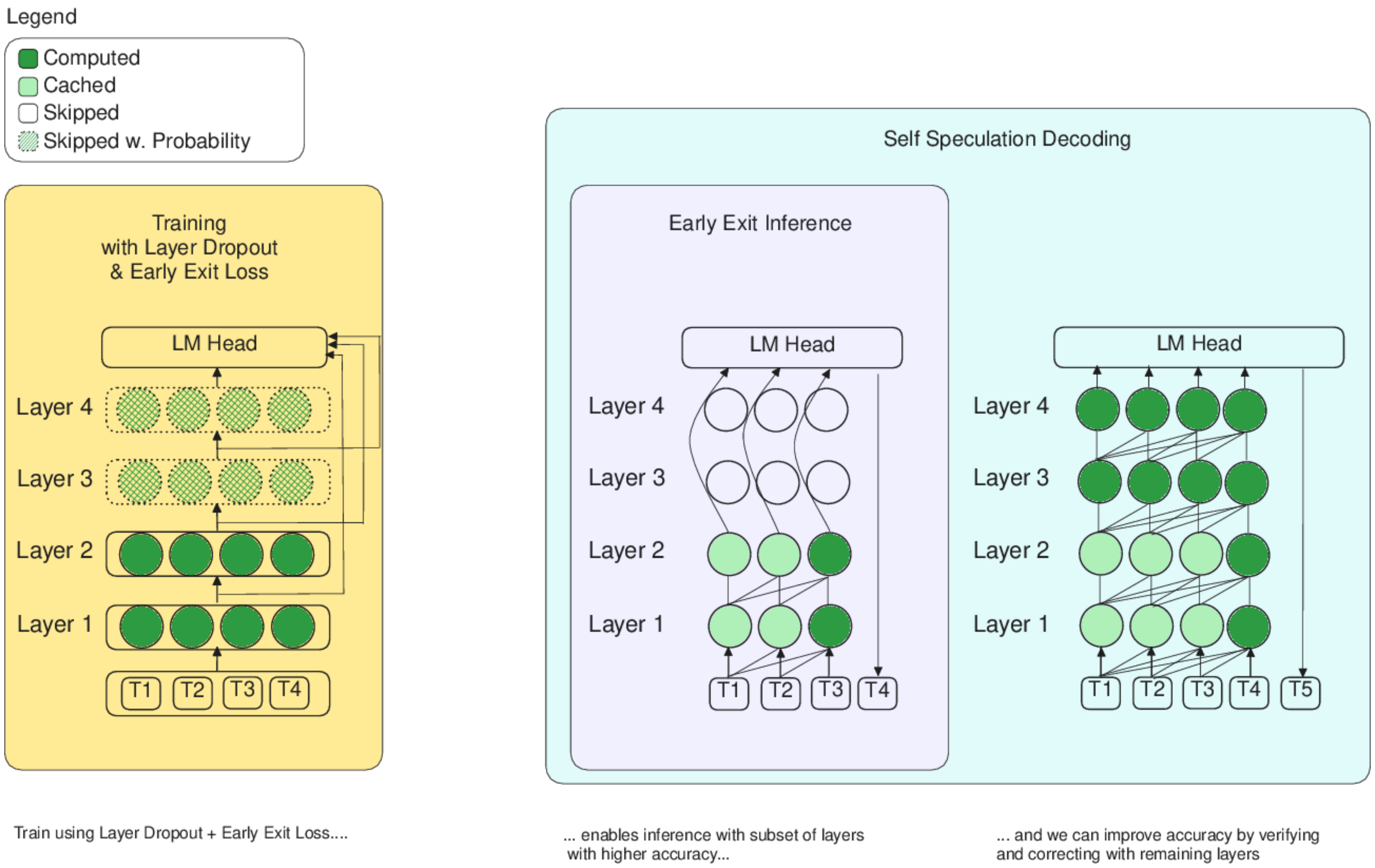
- Loading two models in memory
- Making sure the assistant model outputs a token distribution as close as possible to the main model

Layer skip: self speculative decoding

Why not let the main model do the speculation itself?

Transformer models are believed to be **over-parameterized** 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



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

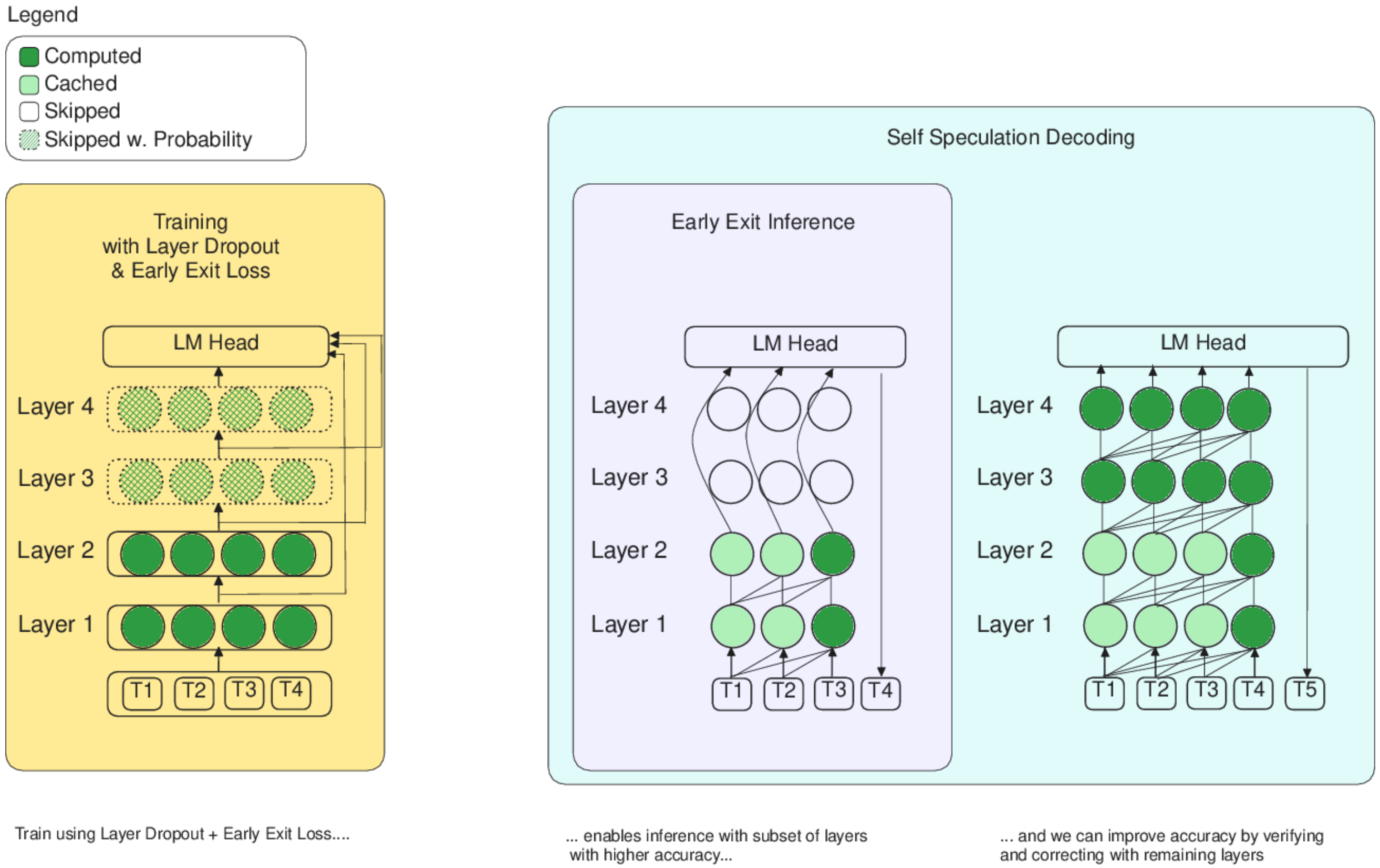
$$C(t,l) = \begin{cases} 1 & \text{if there is no early exit at layer } l \\ 0 & \text{otherwise} \end{cases}$$

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?



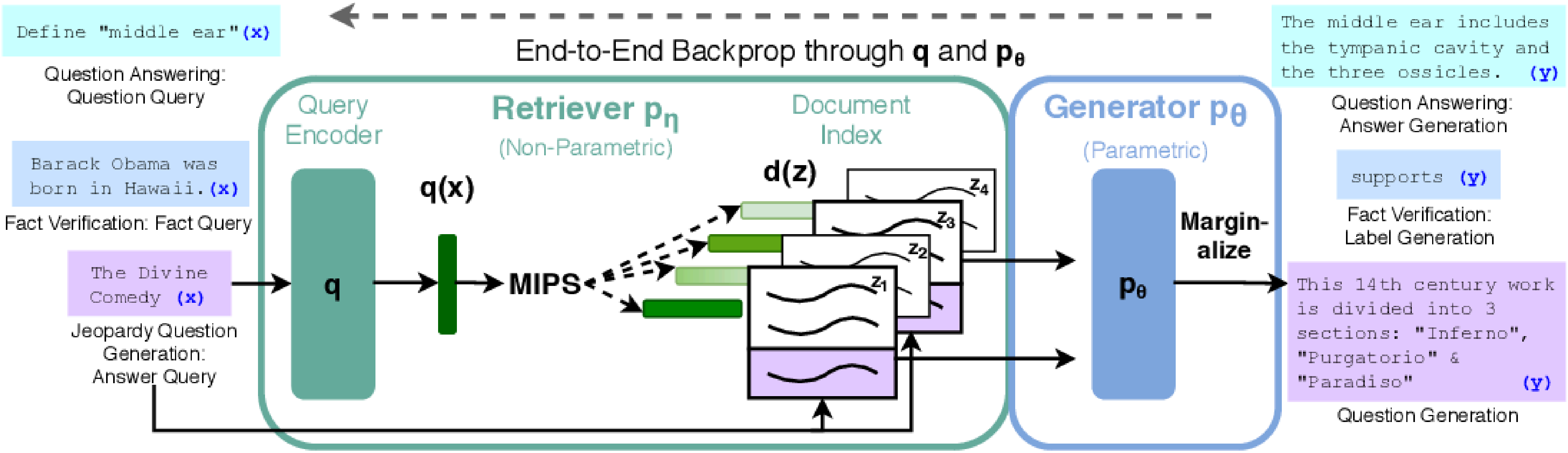
Layer skip: self speculative decoding

- 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)

The goal of retrieval augmented generation (RAG) is to give access to updated knowledge to a model [8].



RAG's intricacies will be discussed in another chapter.

Retrieval augmented generation (at inference)

RAG-sequence model

$$p_{\text{RAG-sequence}}(y|x) \approx \sum_{z \in \text{top-}k} p_\eta(z|x) \prod_i^N p_\theta(y_i|x, z, y_{1:i-1})$$

RAG-token model

$$p_{\text{RAG-token}}(y|x) \approx \prod_i^N \sum_{z \in \text{top-}k} p_\eta(z|x) p_\theta(y_i|x, z, y_{1:i-1})$$

Retrieval augmented generation (at inference)

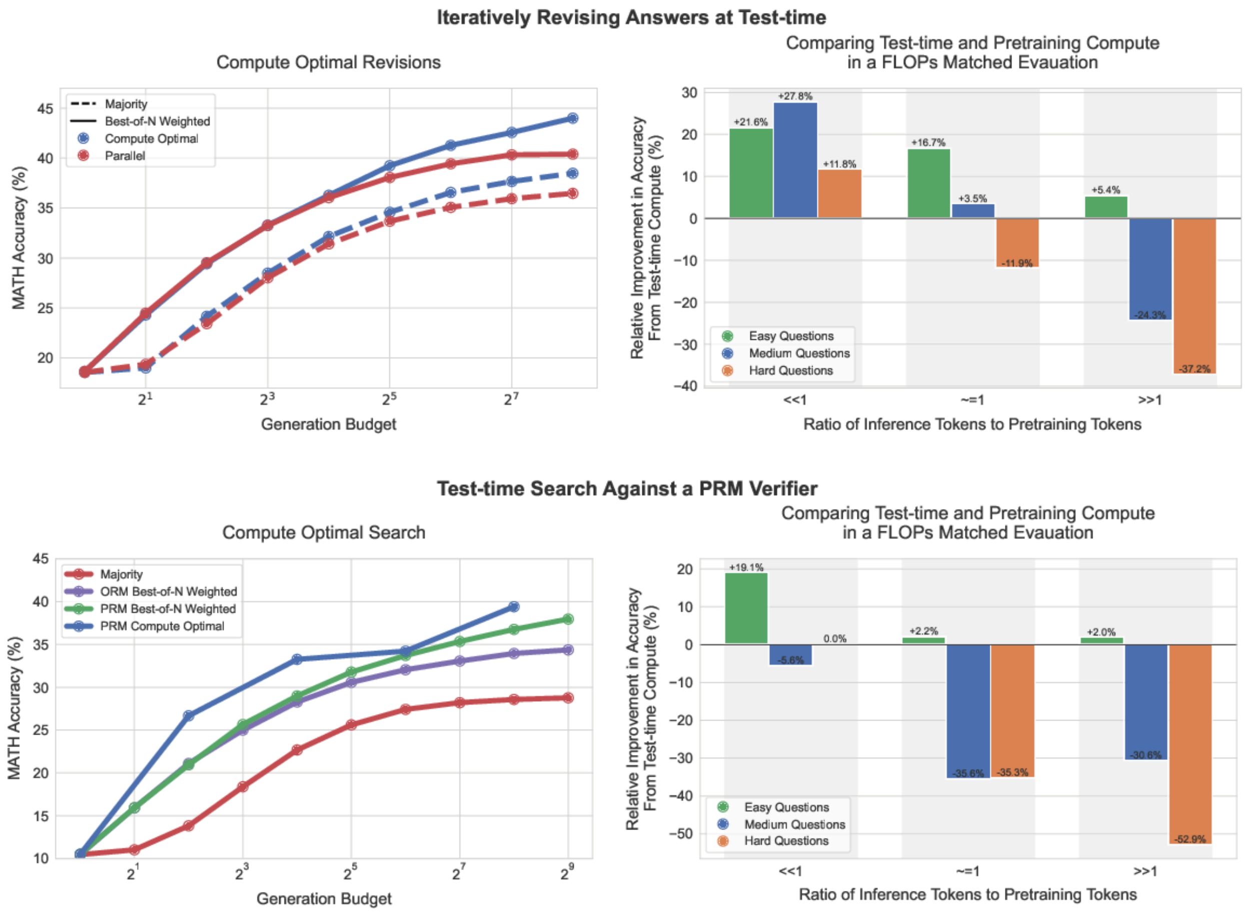
- Although conditioned on retrieved knowledge, output may be a hallucination.
- Most of RAG's performance depends on the chunking method and the retriever.

Test time compute

The goal is to **allocate more compute at inference** to **"natively" incorporate chain-of-thought** like decoding.

The hypothesis is that **models have good reasoning capabilities** but standard **decoding processes hinder them**.

Test time compute

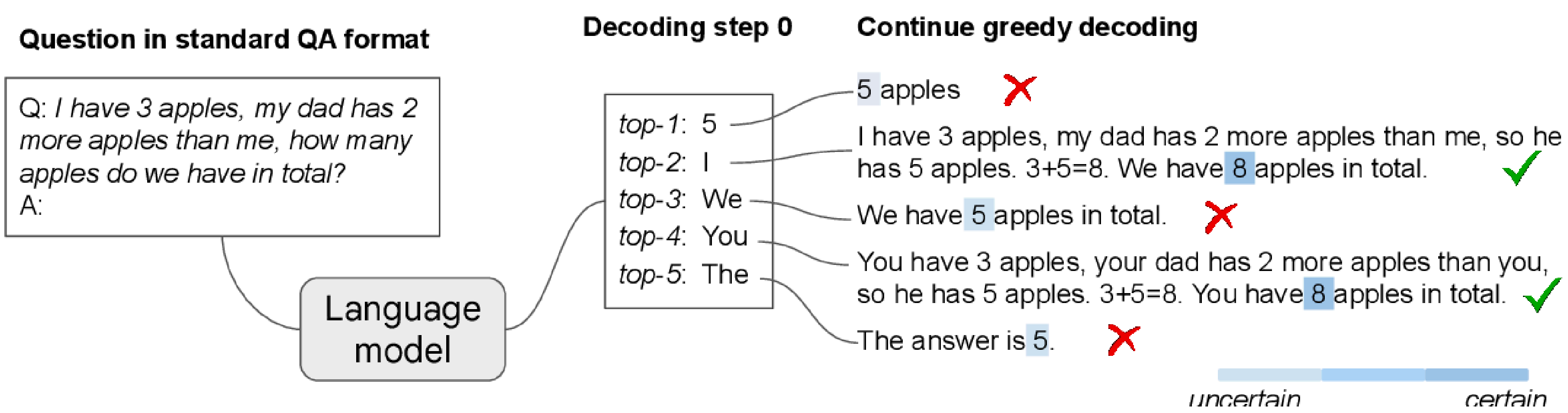


[9]

Test time compute

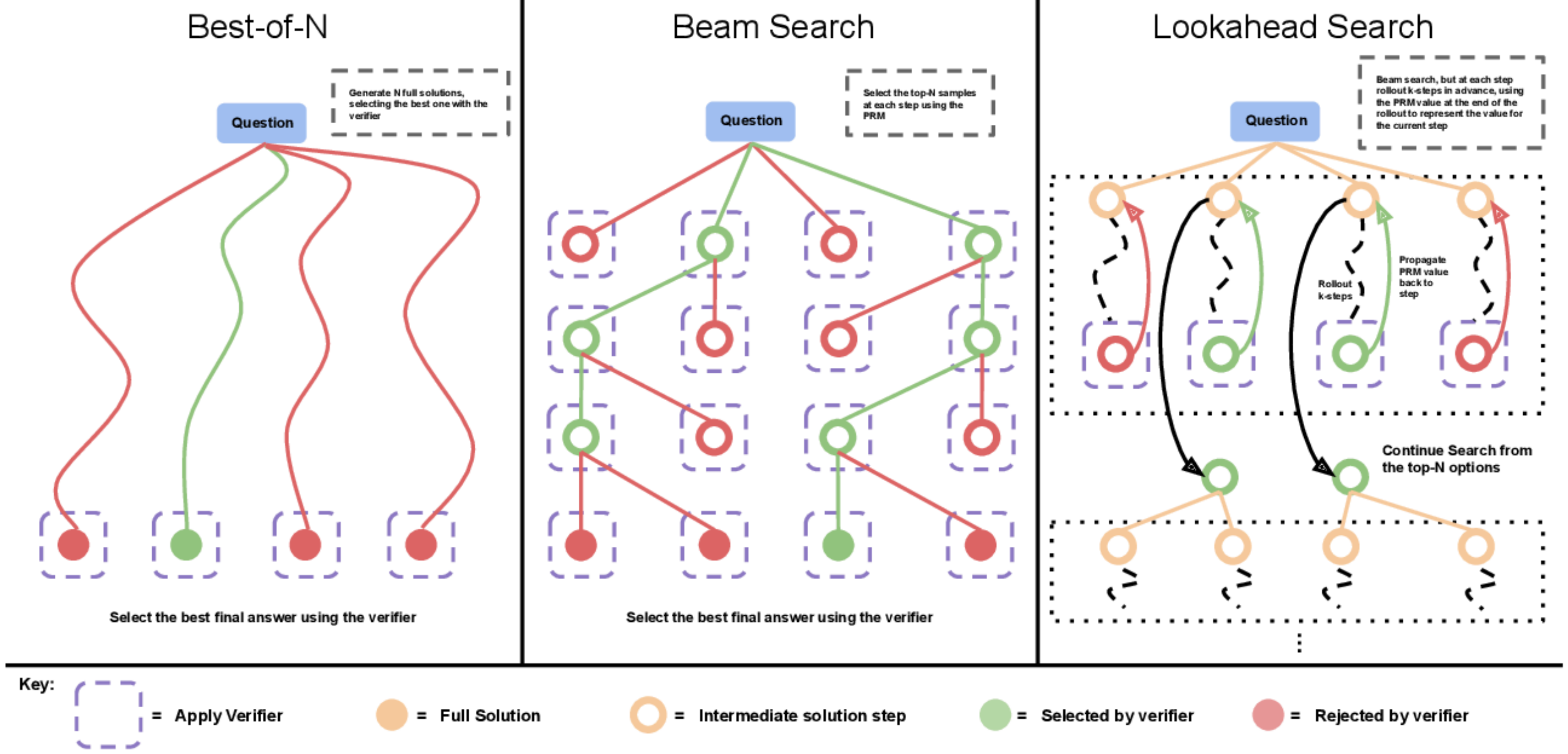
Search against verifiers [10]:

- Most decoding methods stem from greedy decoding.
- There is no "correct" way of selecting the first token when decoding.



test time compute

A **reward model (verifier)** selects the **best answer** based on a **systematic search method**:



Test time compute

Modifying proposal distribution:

Reinforcement learning-like techniques where a **model learns to refine its own answer** to reach the optimal one: look at **ReST** [12] and **STaR** [11].

Unlinke standard decoding, **the model can backtrack to previous steps**.

Test time compute

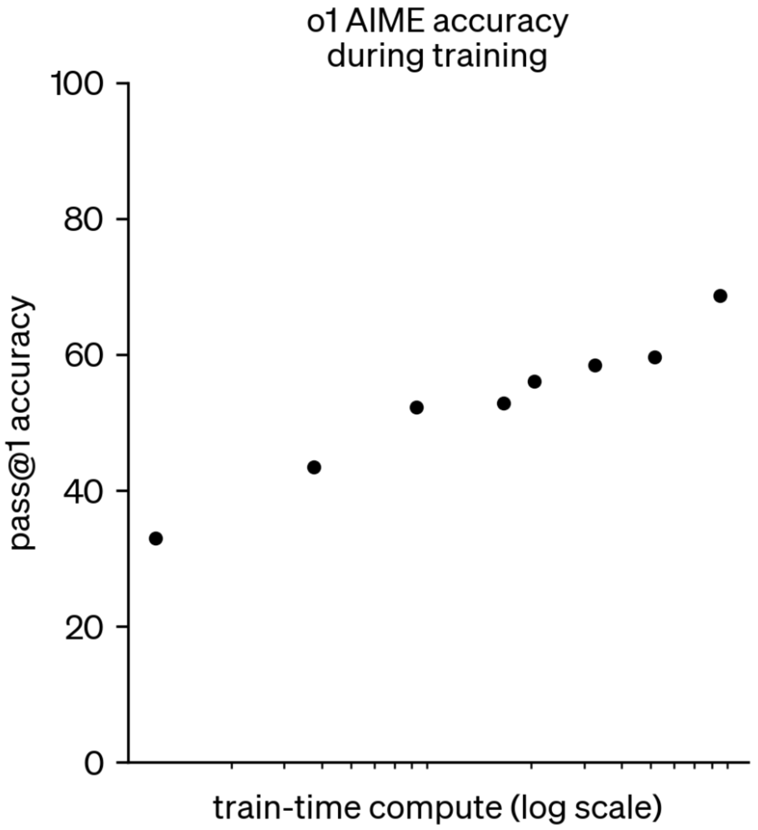
- Borrowing from **ReST**, one could **create candidate responses during inference** and **assess them against a task-specific quality metric** (without updating weights). The highest-quality candidates can then **guide token sampling**.
- **STaR's** multi-path reasoning generation and selection is applicable at test-time by **generating multiple answer paths** and **using consistency checks** or **reranking** to **choose the best response**.

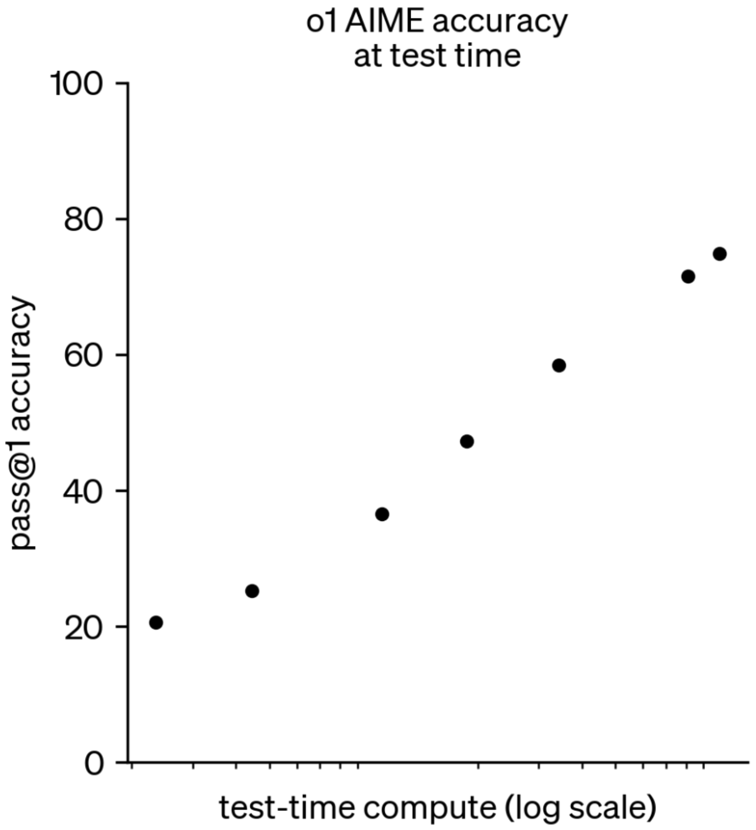
Test time compute

Takeaways (DeepMind's scaling laws):

- Small models (<10b) are better at answering easy questions when given more TTC than pretraning compute.
- Diminishing return on larger models with more TTC than pretraining compute.

Test time compute



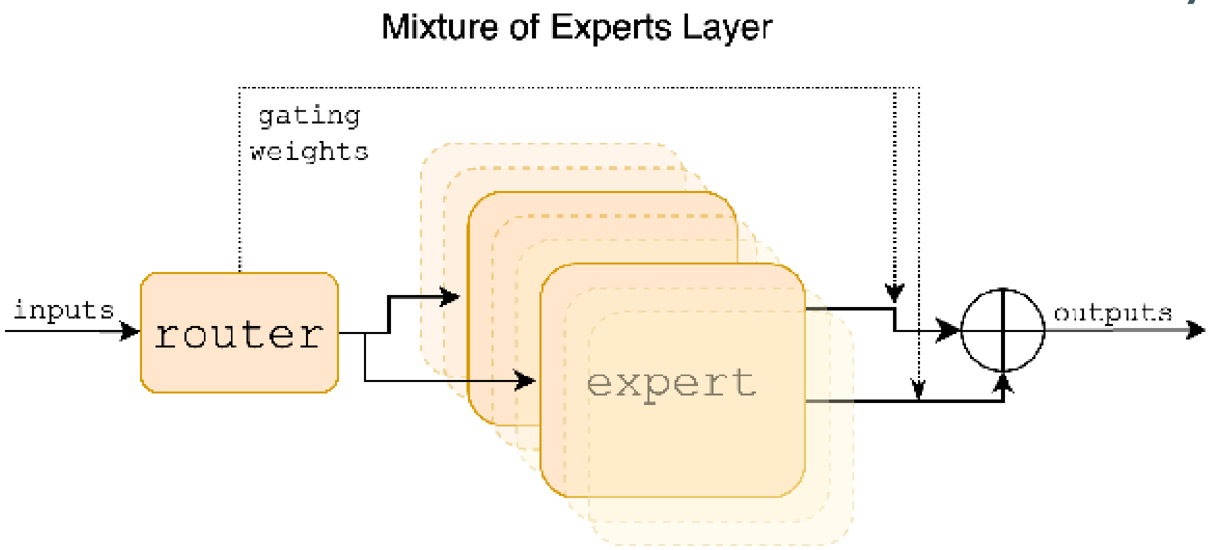


[13]

More About "Balance"?

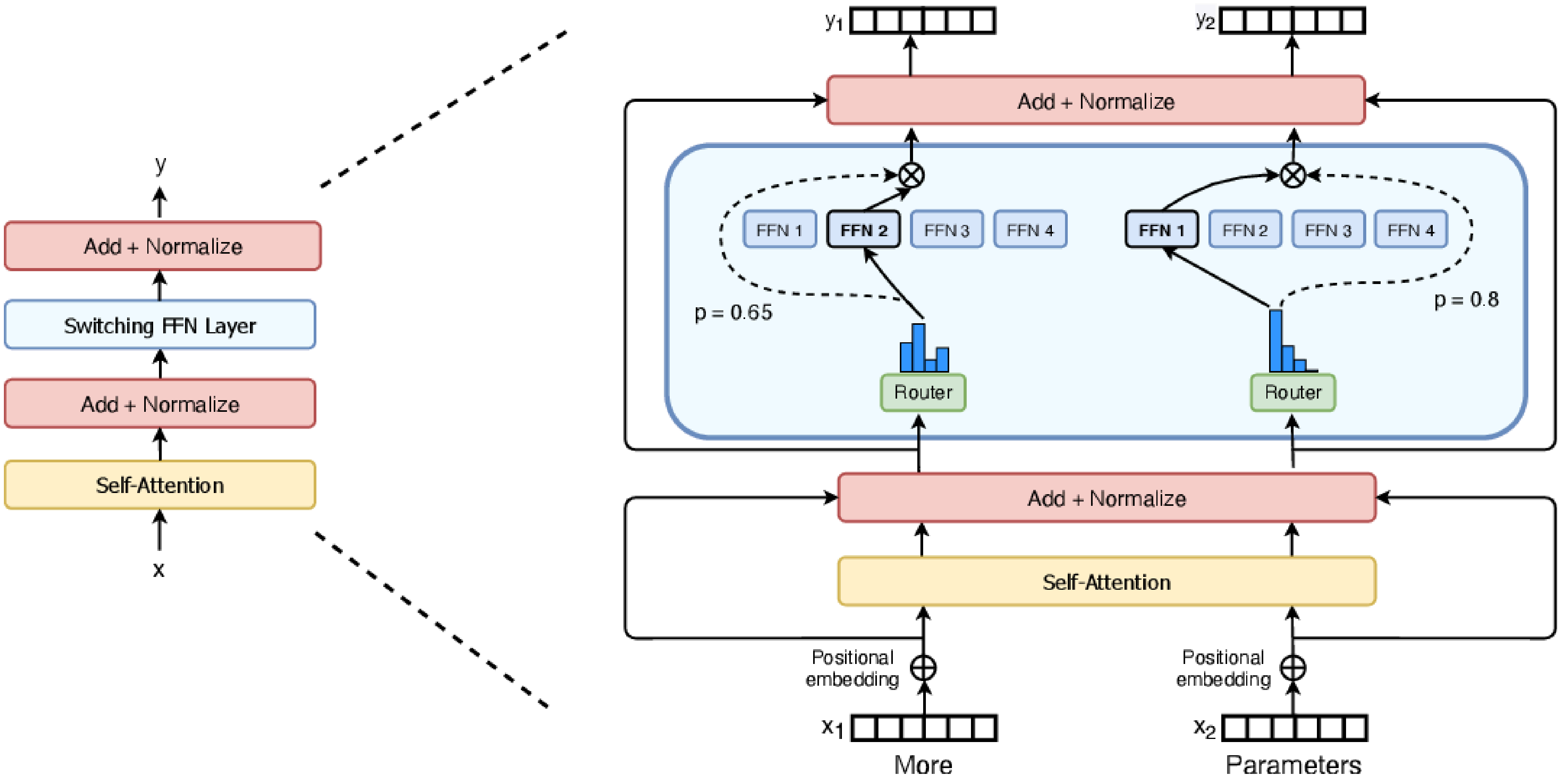
Mixture of experts

Replacing every FFN in a transformers with a MoE layer [14]?



Divide one FFN network with M parameters into N experts with $M' = \frac{M}{N}$ parameters each.

Mixture of experts



Mixture of experts

- Reduced computation during training and inference since we only need to run $1/N$th of the FFN weights.
- Unstable during training: can struggle to generalize, thus prone to overfitting.
- Load balancing is crucial: we do not want a subset of experts to be under-utilized.

Mixture of experts

A learned gating network G decides which experts E to send a part of the input:

$$y = \sum_{i=1}^{n} G(x)_i \times E_i(x)$$

Where $G(x)_i$ denotes the n-dimensional output of the gating network for the i-th expert, and $E_i(x)$ is the output of the i-th expert network

Mixture of experts

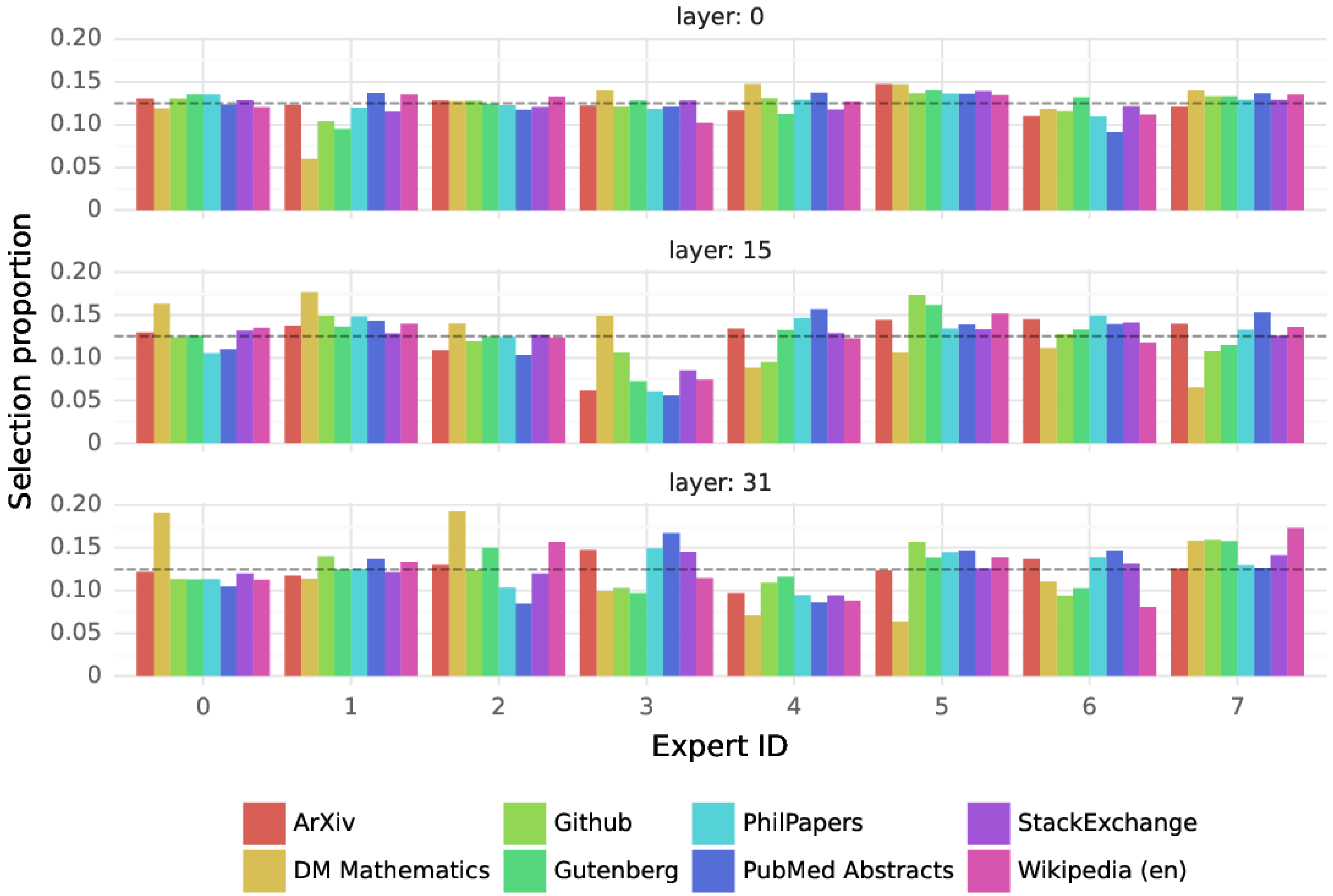
A popular gating function is the softmax function over the top-k logits.

$$G(x) := \text{softmax}(\text{top-}k(x \cdot W_g))$$

In order to have a sparse vector as output

$$\text{top-}k(x \cdot W_g) = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ of } x \cdot W_g \\ -\infty & \text{otherwise} \end{cases}$$

Mixture of experts



Mixture of experts

Questions?

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).

[8] Lewis, Patrick, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks." Advances in Neural Information Processing Systems 33 (2020): 9459-9474.

[9] Snell, Charlie, et al. "Scaling llm test-time compute optimally can be more effective than scaling model parameters." arXiv preprint arXiv:2408.03314 (2024).

[10] Wang, Xuezhi, and Denny Zhou. "Chain-of-thought reasoning without prompting." arXiv preprint arXiv:2402.10200 (2024).

[11] Zelikman, Eric, et al. "Star: Bootstrapping reasoning with reasoning." Advances in Neural Information Processing Systems 35 (2022): 15476-15488.

[12] Gulcehre, Caglar, et al. "Reinforced self-training (rest) for language modeling." arXiv preprint arXiv:2308.08998 (2023).

[13] Learning to Reason with LLMs (2024).

[14] Jiang, Albert Q., et al. "Mixtral of experts." arXiv preprint arXiv:2401.04088 (2024).