LLM Acceleration

Skip Methods: MoD and LayerSkip

Kyungseon Lee

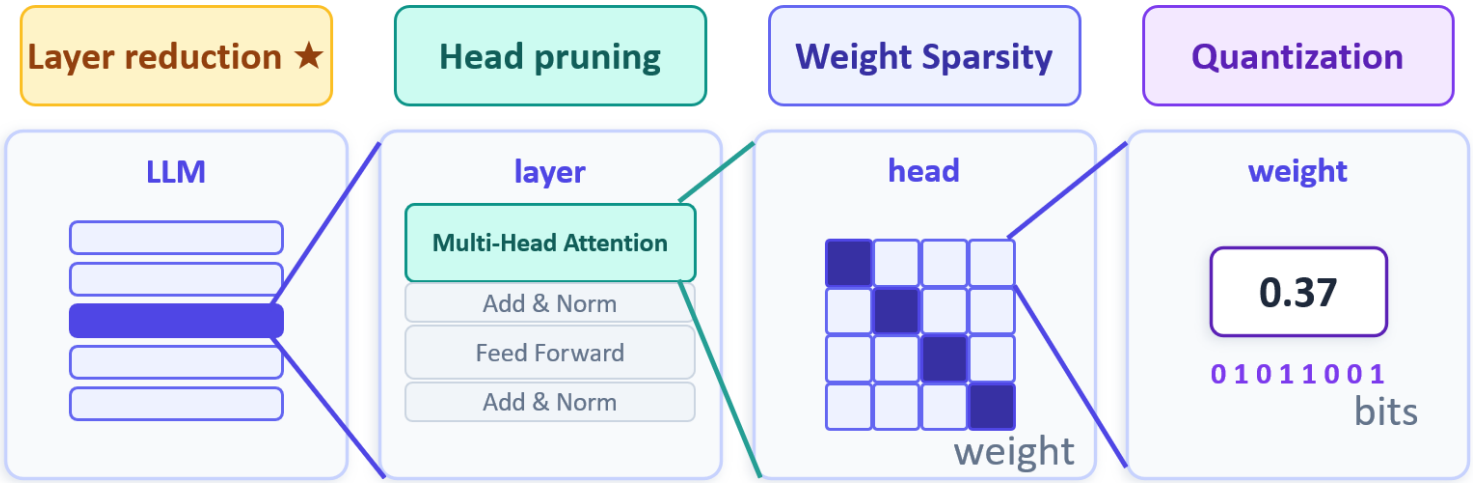
2026-06

Seoul National University

Background

LLM acceleration

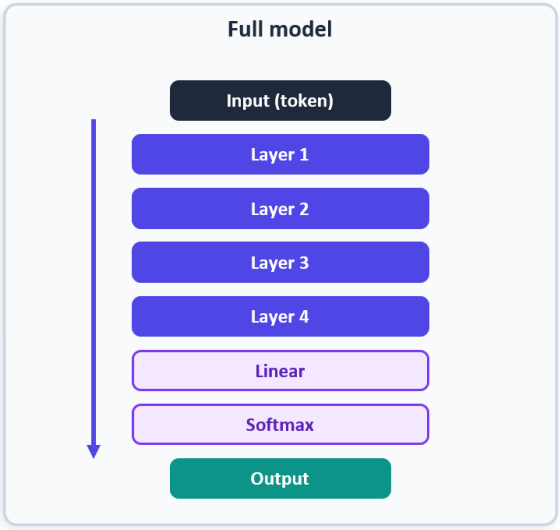
- **Quantization** – shrink the number of bits used to store one weight.
- **Weight Sparsity** – drop weights or activations that take part in the computation.
- **Head pruning** – cut the number of attention heads in a layer.
- **Layer reduction** $\star$ – cut the number of transformer layers a token passes through.

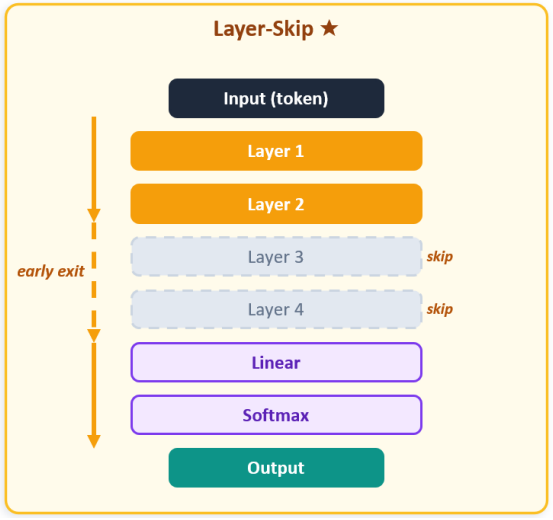


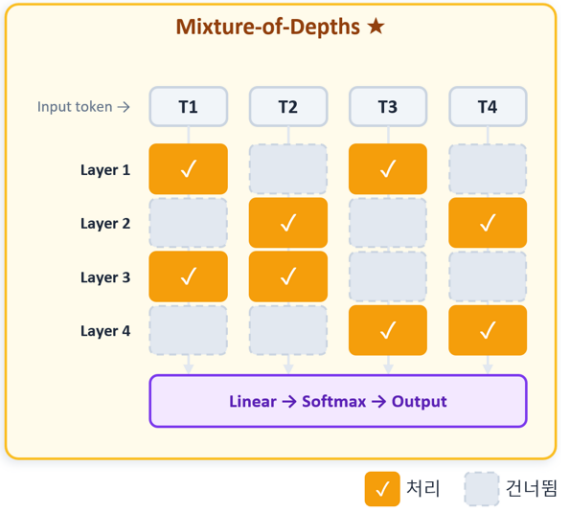
Layer-Skip Family

- Transformer
  - **LayerDrop** (Fan et al., ICLR 2020) – some transformer layers can be dropped at random.
  - **Depth-Adaptive Transformer** (Elbayad et al., ICLR 2020) – if $p(\text{correct} \mid \text{intermediate output})$ is high, use that intermediate output.
- LLM
  - **CALM** (Schuster et al., NeurIPS 2022) – use an intermediate output when it does not hurt overall quality.
  - **CoLT5** (Ainslie et al., EMNLP 2023) – at each layer, send tokens that reduce the total loss the most to the heavy branch.
  - **Mixture-of-Depths** $\star$ (Raposo et al., 2024) – at each layer, process only the tokens that reduce the total loss the most.
  - **LayerSkip** $\star$ (Elhoushi et al., ACL 2024) – use intermediate outputs that already look close to the full-model output.

Layer-Skip Family







- **Mixture-of-Depths** ⋆ (Raposo et al., 2024) – at each layer, process only the inputs that reduce the total loss the most.
- **LayerSkip** ⋆ (Elhoushi et al., ACL 2024) – use intermediate outputs that already look close to the full-model output.

Mixture-of-Depths: Dynamically Allocating Compute in Transformer-Based Language Models

MoD

Router: at every layer, decides which tokens go in as input.

- compute a score $r_{l,t}$; the layer takes only the top-k tokens.

$$r_{l,t} = R(x_{l,t}), \quad x_{l+1,t} = \begin{cases} x_{l,t} + r_{l,t} f_l(x_{l,t}) & \text{if } t \in \text{top-}k, \\ x_{l,t} & \text{otherwise.} \end{cases}$$

- $x_{l,t}$: output for token t at layer l.
- $R(\cdot) : \mathbb{R}^d \to \mathbb{R}$: linear function that scores each token; updated by gradient descent.
- $f_l(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$: self-attention and MLP of layer l.

LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding

LayerSkip: Inference

Early-exit idea: draft tokens from an intermediate layer, then verify with the full model.

1. Generate d tokens at layer E.
2. Verify these tokens with the full model. The compute drops because the full model runs once over the d positions, not d separate times.

$$\hat{y}_{m+j} \stackrel{?}{=} \arg\max_{v \in V} P_{full}(v | \hat{y}_{m:m+j-1}, y_{1:m}),\ j = 1, \cdots, d$$

3. Accept tokens that match the full-model prediction; at the first mismatch, replace it with the full-model prediction.

- $\hat{y}_i, y_i$: draft token from the intermediate layer, and the token generated earlier.
- $v \in V$: vocabulary.
- d: number of draft tokens per round.
- g: LM head.
- L: total number of layers.
- E: hyperparameter; the fixed exit layer.

LayerSkip: Training

Two things are needed: 1. assign a higher drop probability to deeper layers; 2. make intermediate-layer outputs behave like the full-model output.

1. Layer dropout – drop later layers more often during training.

$$x_{l+1,iter} = x_{l,iter} + M(p_{l,iter})\, f_l(x_{l,iter}), \qquad p_{l,iter} = D(l)$$

- l: the l-th layer.
- $M(p)$: Bernoulli variable returning 0 with probability p.
- $D(l)$: grows with l.

2. Early-exit loss – attach a shared LM head g to every layer's output and add a loss term.

$$\mathcal{L}(X, Y, t) = \sum_{l=0}^{L-1} \tilde{e}_{l,iter}\, CE(g(x_{l+1}), Y)\,.$$

- CE: cross-entropy loss.
- $\tilde{e}_{l,iter}$: per-layer loss weight.

References

- Raposo, D., Ritter, S., Richards, B., et al. (2024). *Mixture-of-Depths: Dynamically Allocating Compute in Transformer-Based Language Models*. arXiv:2404.02258.
- Elhoushi, M., Shrivastava, A., Liskovich, D., et al. (2024). *LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding*. arXiv:2404.16710.
- Fan, A., Grave, E., Joulin, A. (2020). *Reducing Transformer Depth on Demand with Structured Dropout*. ICLR.
- Elbayad, M., Gu, J., Grave, E., Auli, M. (2020). *Depth-Adaptive Transformer*. ICLR.
- Schuster, T., Fisch, A., Gupta, J., et al. (2022). *Confident Adaptive Language Modeling*. NeurIPS.
- Ainslie, J., Lei, T., et al. (2023). *CoLT5: Faster Long-Range Transformers with Conditional Computation*. EMNLP.