NYU

Lecture 08:
Efficient DNN Training, Parameter Efficient Finetuning, Speculative Decoding

Notes

- Lab 2 is due this week, Lab 3 will post this weekend.
- Grade of Lab 1 has been posted.
- Think about the project, discuss with me during office hours or after class.
- Midterm
  - Mar 27, in class.
  - Will cover materials up to this lecture (lecture 8)
  - Discuss about the coverage later this lecture
- Lab 3 will be posted this weekend.
- No Quiz today.

Project Details

- Due on March 20th 11:59pm (1 page)
  - Name + NetID of each group member
  - Project summary (1 paragraph)
  - Project plan (1 paragraph, gantt chart (optional))
  - Individual responsibilities (1 paragraph)
- You should start forming teams of **2–3 students**, which is the **strongly recommended project size**.
- Mark breakdown:
  - Proposal (1 page) 5%
  - Final presentation 10%
  - Final report 10%

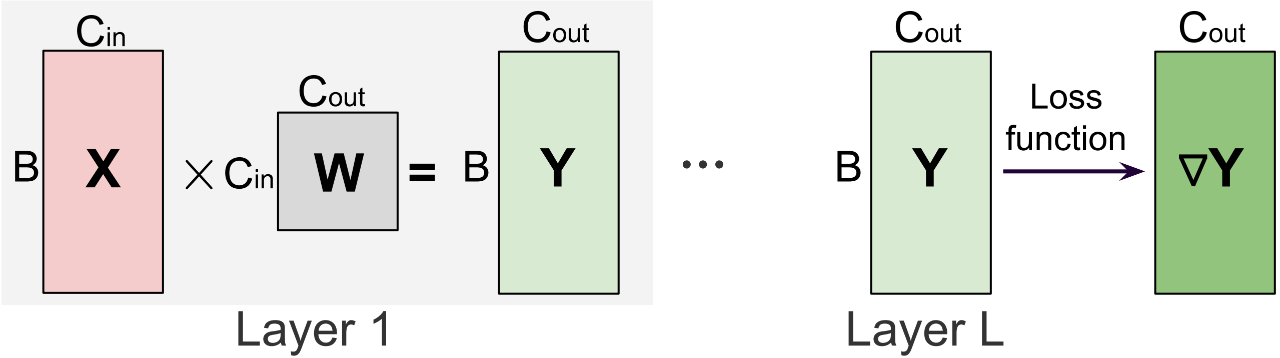
Recap

- Large Model Data Distribution
- Large Model Quantization
- Large Model Pruning
- Low-rank Decomposition for LLM

Topics

- Efficient training of DNNs
  - Efficient computing
  - Efficient storage
- Parameter efficient finetuning
- Speculative Decoding

Forward Pass for Linear Layer



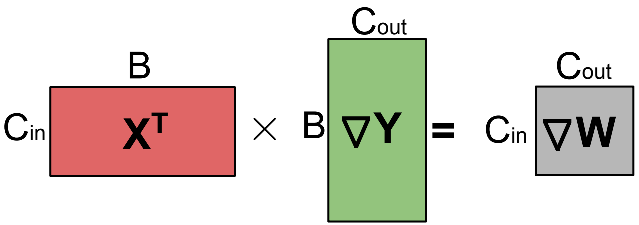
B: batch size　　C_{in}: input channels　　C_{out}: output channels

X: input maps　　**W**: weight filters　　**Y**: output maps

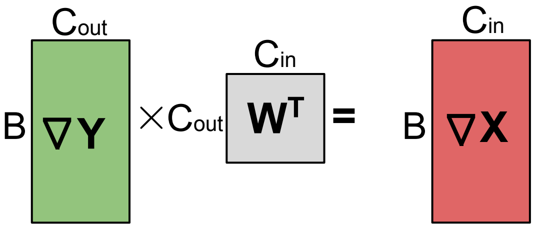
- The fully-connected layer during the forward propagation can be converted into matrix multiplications.

Backward Pass for Linear Layer

Weight Gradient Computation



Data Gradient Computation



X: input maps
∇X: input gradient

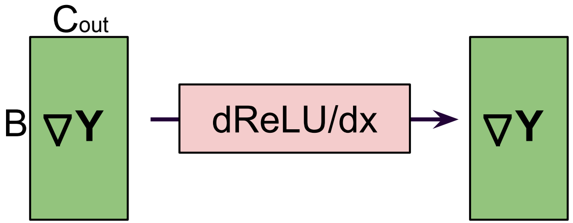
W: weight filters
∇W: weight gradient

Y: output maps
∇Y: output gradient

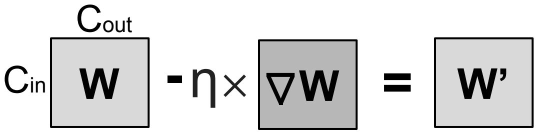
- DNN backward propagation involves two matrix multiplications

Backward Pass for Linear Layer

Data Gradient Computations



Weight Gradient Updates



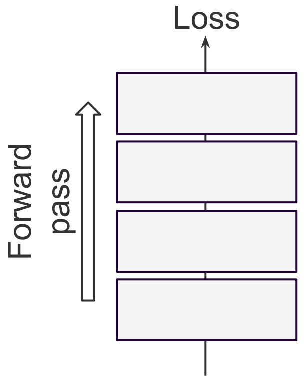
X: input maps
∇X: input gradient

W: weight filters
∇W: weight gradient

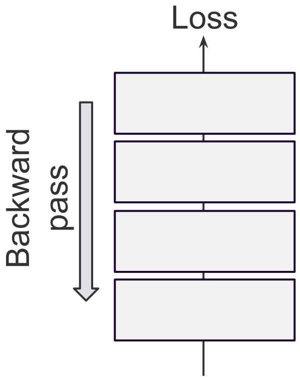
Y: output maps
∇Y: output gradient

- DNN backward propagation involves two matrix multiplications

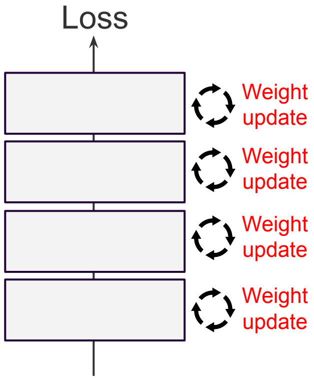
Training Process



```
def forward(self, x):
    ...
return
```



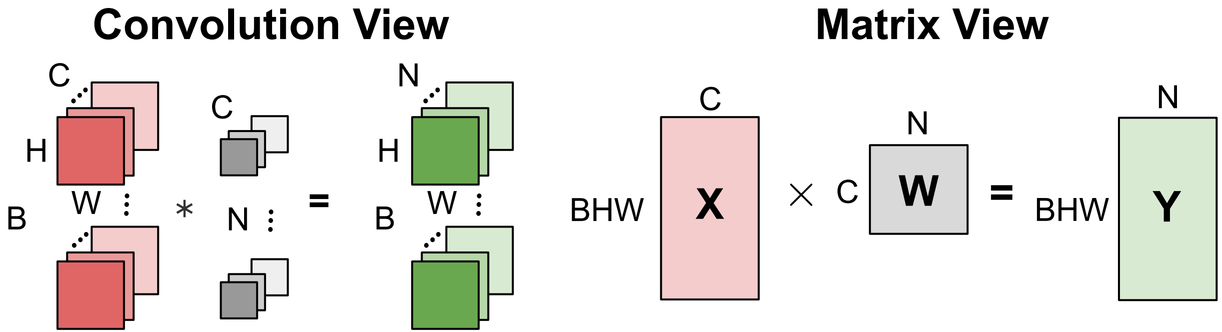
```
loss.backward()
```



```
optimizer.step()
```

Forward Pass for Convolutional Layer

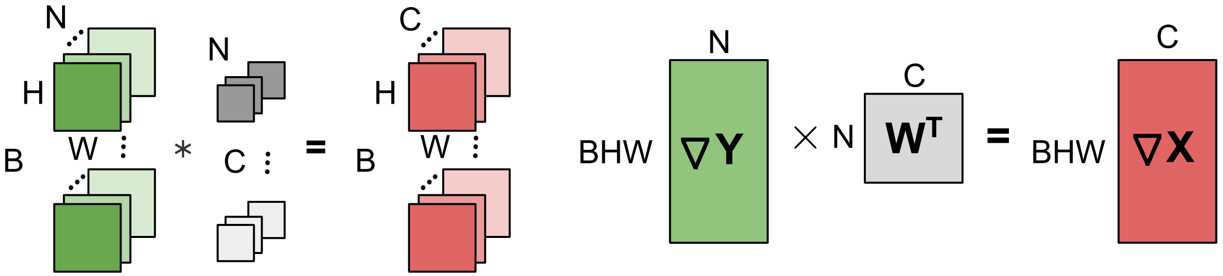
Forward Pass
Compute output **Y**



- Assume a weight kernel size of 1✖1.

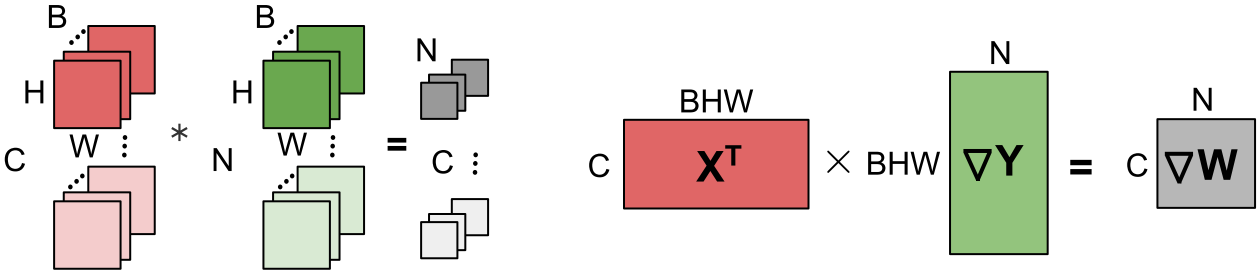
Backward Pass for Convolutional Layer

Backward Pass
Compute Activation gradients **∇X**



Backward Pass for Convolutional Layer

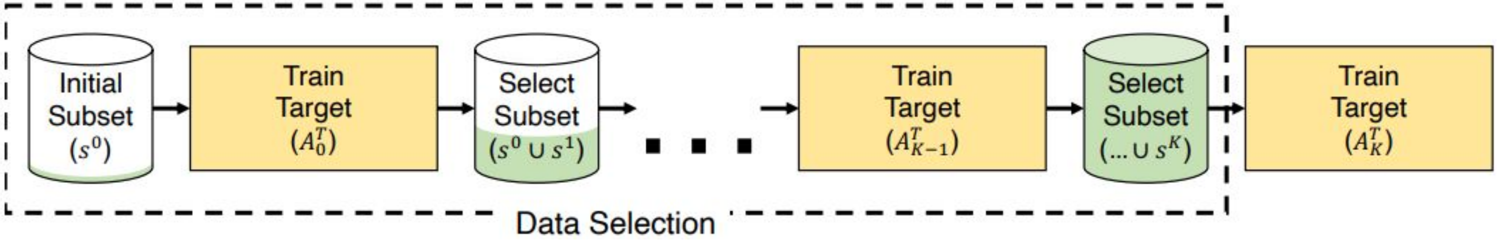
Backward Pass
Compute weight gradients **∇W**



Efficient Computing during Training

- To reduce the training cost of DNN, we can proceed from the following dimensions:
    - Training data sampling
    - Parameter sampling
    - Pruning during training
    - Quantization during training

Training Data Sampling for Efficiency


Initial Subset (s^0)
Train Target (A_0^T)
Select Subset ($s^0 \cup s^1$)
...
Train Target (A_{K-1}^T)
Select Subset ($\ldots \cup s^K$)
Train Target (A_K^T)
Data Selection

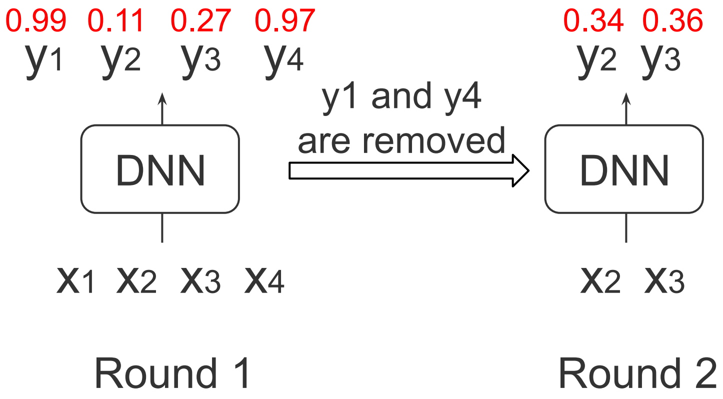
- Assume a total b samples are targeted to be picked. We consider a batch setting with K rounds where we select b/K points in every round.

- Training the target model with b/K samples, then evaluate the rest of the sample over the model. Find the batch with the least confidence score. Append it to the training dataset.

NYU SAI LAB

Cody Coleman, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Select via proxy: Efficient data selection for training deep networks, 2019.

Training Data Sampling for Efficiency



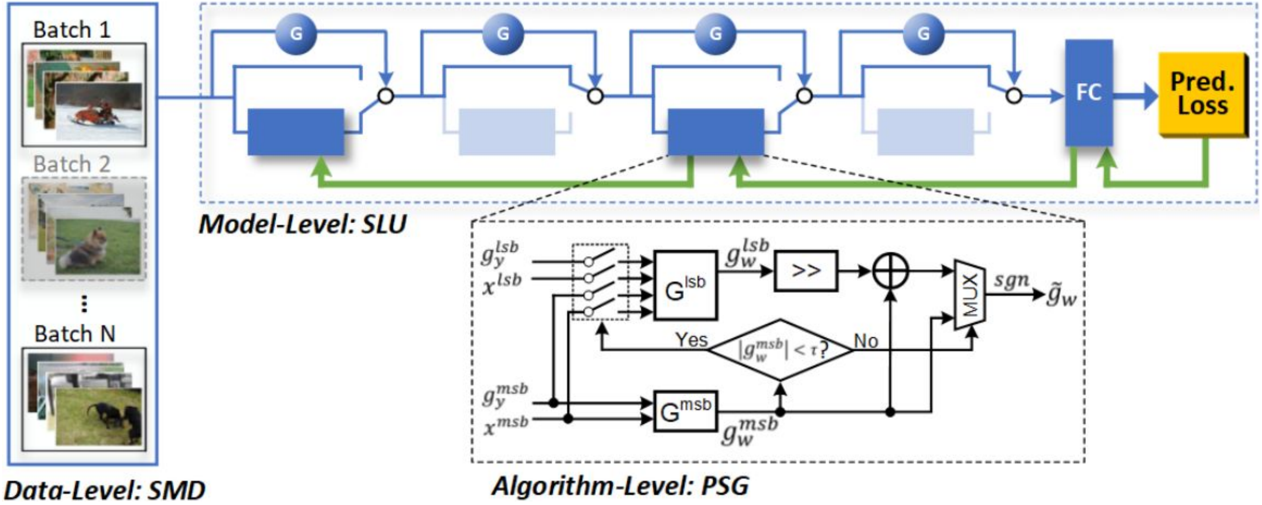
- Assume a total b samples are targeted to be picked. We consider a batch setting with K rounds where we select b/K points in every round.
- Training the target model with b/K samples, then evaluate the rest of the sample over the model. Find the batch with the least confidence score. Append it to the training dataset.

Cody Coleman, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Select via proxy: Efficient data selection for training deep networks, 2019.

Efficient Computing during Training

- To reduce the training cost of DNN, we can proceed from the following dimensions:
    - Training data sampling
    - Parameter sampling
    - Pruning during training
    - Quantization during training

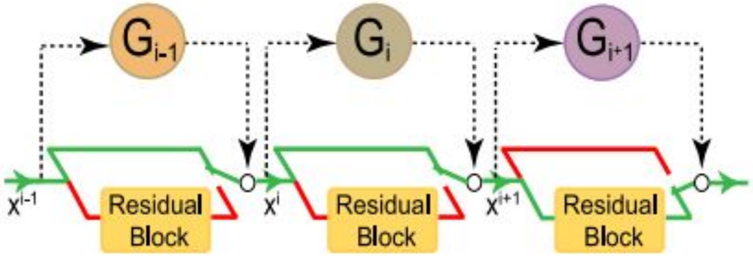
E2-Train



- A stochastic mini-batch dropping strategy is proposed.
- Stochastic minibatch dropping simply skips every mini-batch with a default probability of 0.5.
- For some easy dataset, this will generate negligible impact on performance.

Wang, Yue, et al. "E2-train: Training state-of-the-art cnns with over 80% energy savings." *Advances in Neural Information Processing Systems* 32 (2019).

Dynamically Layer Skipping



- $G_i(x_i) \in \{0, 1\}$ is the gating function for layer i.
- It determines whether to skip to current residual block or not.
- During the training, G and residual blocks are trained together.
- Loss = acc_loss + computation_loss
- We will skip different layers adaptively based on inputs.

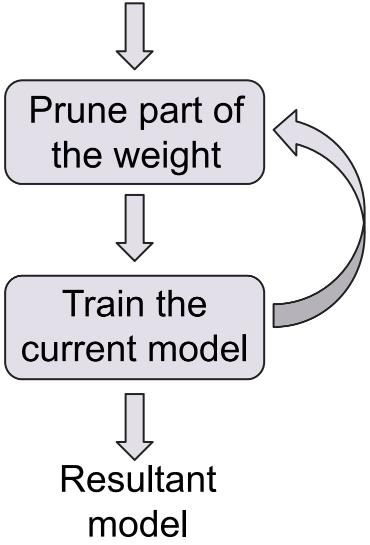
Wang, Xin, et al. "Skipnet: Learning dynamic routing in convolutional networks." *Proceedings of the European conference on computer vision (ECCV)*. 2018.

Wang, Yue, et al. "E2-train: Training state-of-the-art cnns with over 80% energy savings." *Advances in Neural Information Processing Systems* 32 (2019).

Efficient Computing during Training

- To reduce the training cost of DNN, we can proceed from the following dimensions:
  - Training data sampling
  - Parameter sampling
  - Pruning during training
  - Quantization during training

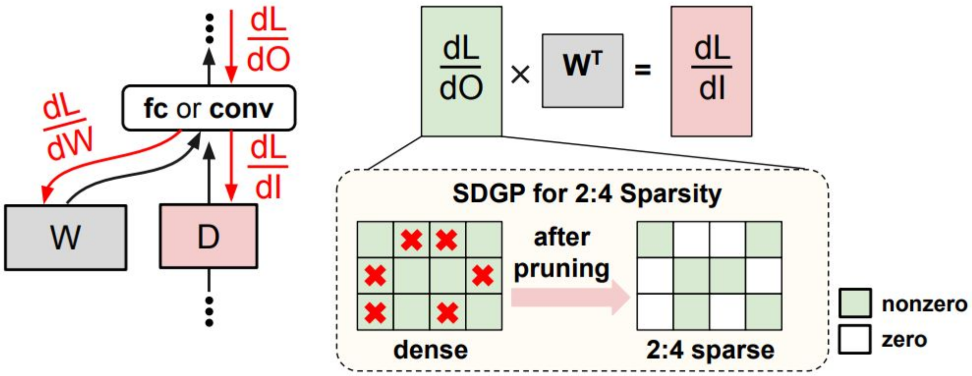
Pruning during Training



Prune part of the weight

Train the current model

Resultant model



- We can remove the unnecessary weight during the DNN training process.

McDanel, Bradley, Helia Dinh, and John Magallanes. "Accelerating dnn training with structured data gradient pruning." *2022 26th International Conference on Pattern Recognition (ICPR)*. IEEE, 2022.

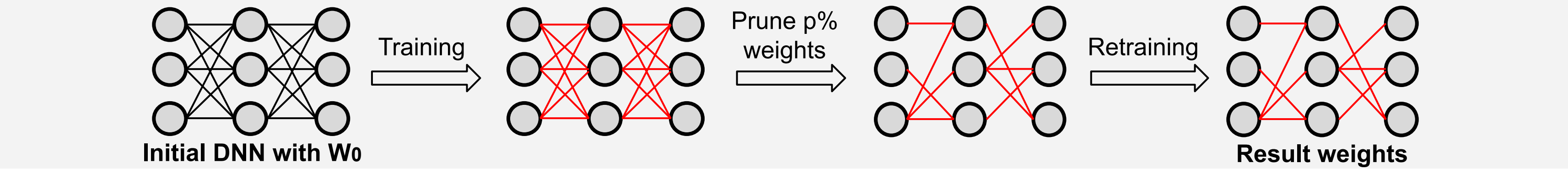
How to Find the Winning Tickets?

- **Iterative Magnitude Pruning (IMP):**
  - Initialized DNN with random weights w_0.
  - While the sparsity level has not reached:
    - Train the DNN with k epochs until convergence
    - prune p% of the nonzero weights.
    - Reinitialize the remaining weights using the values in w_0, finetune the remaining weights for k epochs (**Rewind**).
  - Return the weights.
- Later work has shown that rewind to w_i (i is small) works better for larger networks.

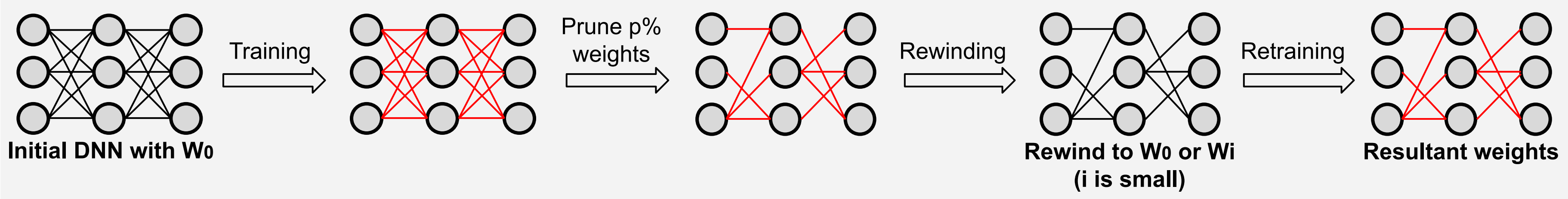
Frankle, Jonathan, et al. "Stabilizing the lottery ticket hypothesis." *arXiv preprint arXiv:1903.01611* (2019).

Weight Rewinding

Conventional iterative pruning



Conventional iterative pruning with weight rewinding

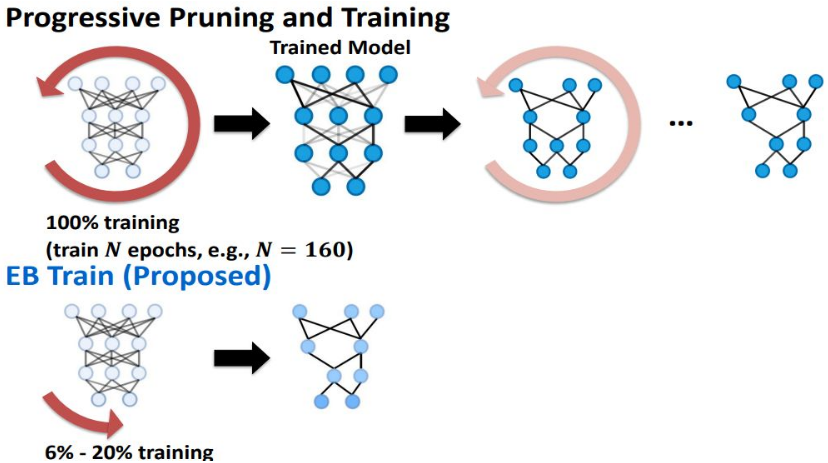


- The pruned architecture itself, rather than a set of inherited “important” weights, is more crucial to the accuracy in the final model, which suggests that in some cases pruning can be useful as an architecture search paradigm.

NYU SAI LAB

Liu, Zhuang, et al. "Rethinking the value of network pruning." *arXiv preprint arXiv:1810.05270* (2018).

Early-bird Ticket


Progressive Pruning and Training
Trained Model
...
100% training
(train N epochs, e.g., $N = 160$)
EB Train (Proposed)
6% - 20% training

- LTH shows that there exist winning tickets (small but critical subnetworks) for dense, randomly initialized networks, that can be trained alone to achieve a comparable accuracy to the latter in a similar number of iterations.

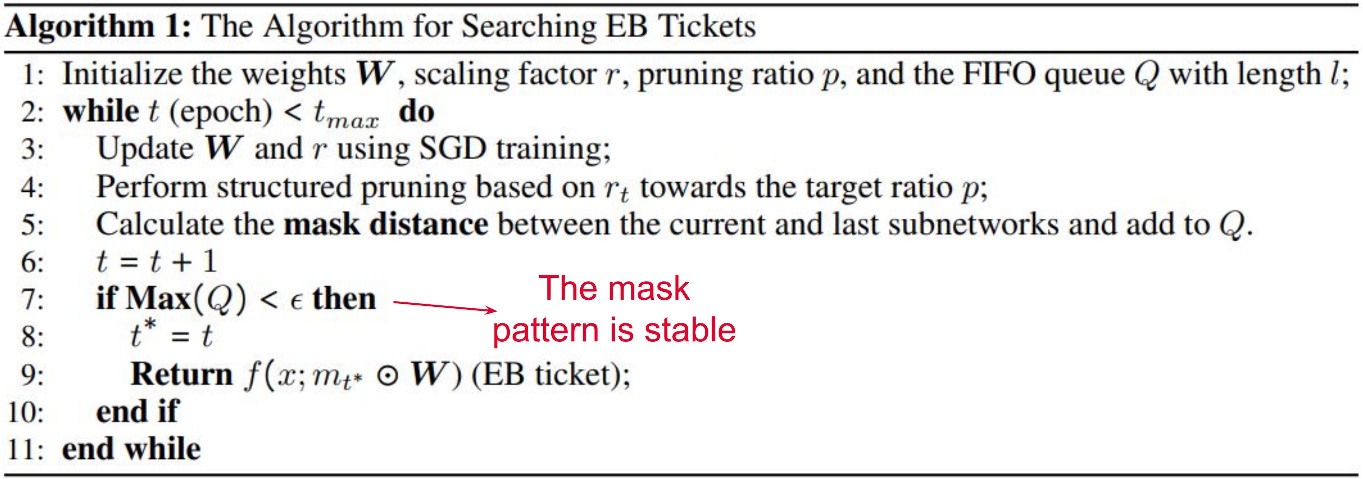
- The winning tickets can be drawn very early in training and with aggressively low-cost training algorithms.

- Early-bird tickets can be founded via low-cost training schemes (e.g., early stopping and low-precision training) at large learning rates

NYU SAI LAB

You, Haoran, et al. "Drawing early-bird tickets: Towards more efficient training of deep networks." *arXiv preprint arXiv:1909.11957* (2019).

Early-bird Ticket



Algorithm 1: The Algorithm for Searching EB Tickets

```
Initialize the weights $W$, scaling factor $r$, pruning ratio $p$, and the FIFO queue $Q$ with length $l$;
while $t$ (epoch) < $t_{max}$ do
    Update $W$ and $r$ using SGD training;
    Perform structured pruning based on $r_t$ towards the target ratio $p$;
    Calculate the mask distance between the current and last subnetworks and add to $Q$.
    $t = t + 1$
    if Max($Q$) < $\epsilon$ then
        $t^* = t$
        Return $f(x; m_{t^*} \odot W)$ (EB ticket);
    end if
end while
```

The mask pattern is stable

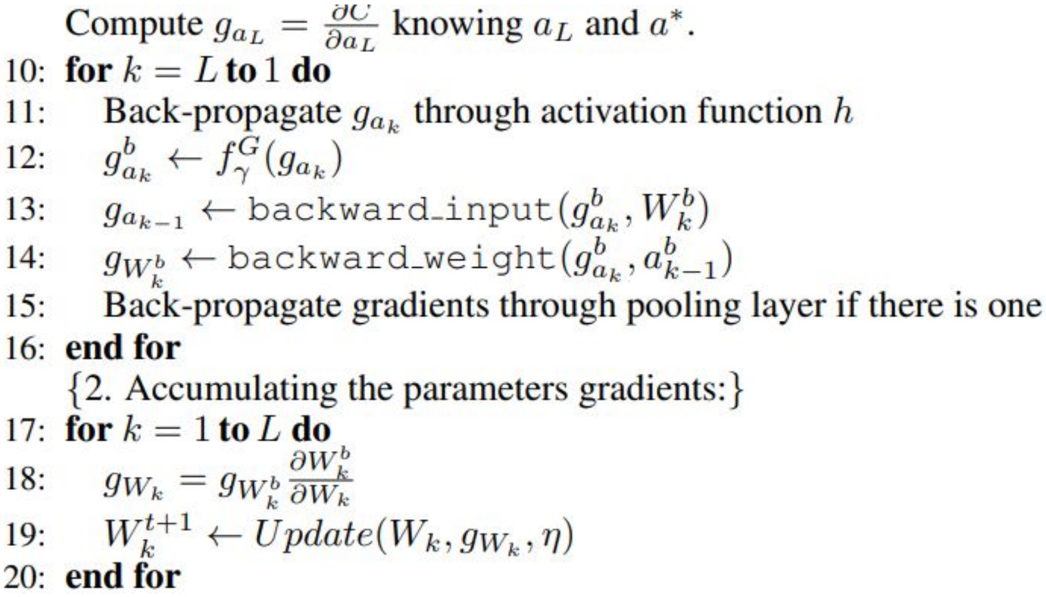
- To search for the lottery ticket, we can early stop the DNN training.

You, Haoran, et al. "Drawing early-bird tickets: Towards more efficient training of deep networks." *arXiv preprint arXiv:1909.11957* (2019).

Efficient Computing during Training

- To reduce the training cost of DNN, we can proceed from the following dimensions:
    - Training data sampling
    - Parameter sampling
    - Pruning during training
    - Quantization during training

DoReFaNet



```
Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$ and $a^*$.
for $k = L$ to 1 do
    Back-propagate $g_{a_k}$ through activation function $h$
    $g^b_{a_k} \leftarrow f^G_\gamma(g_{a_k})$
    $g_{a_{k-1}} \leftarrow \texttt{backward\_input}(g^b_{a_k}, W^b_k)$
    $g_{W^b_k} \leftarrow \texttt{backward\_weight}(g^b_{a_k}, a^b_{k-1})$
    Back-propagate gradients through pooling layer if there is one
end for
{2. Accumulating the parameters gradients:}
for $k = 1$ to $L$ do
    $g_{W_k} = g_{W^b_k} \frac{\partial W^b_k}{\partial W_k}$
    $W^{t+1}_k \leftarrow Update(W_k, g_{W_k}, \eta)$
end for
```

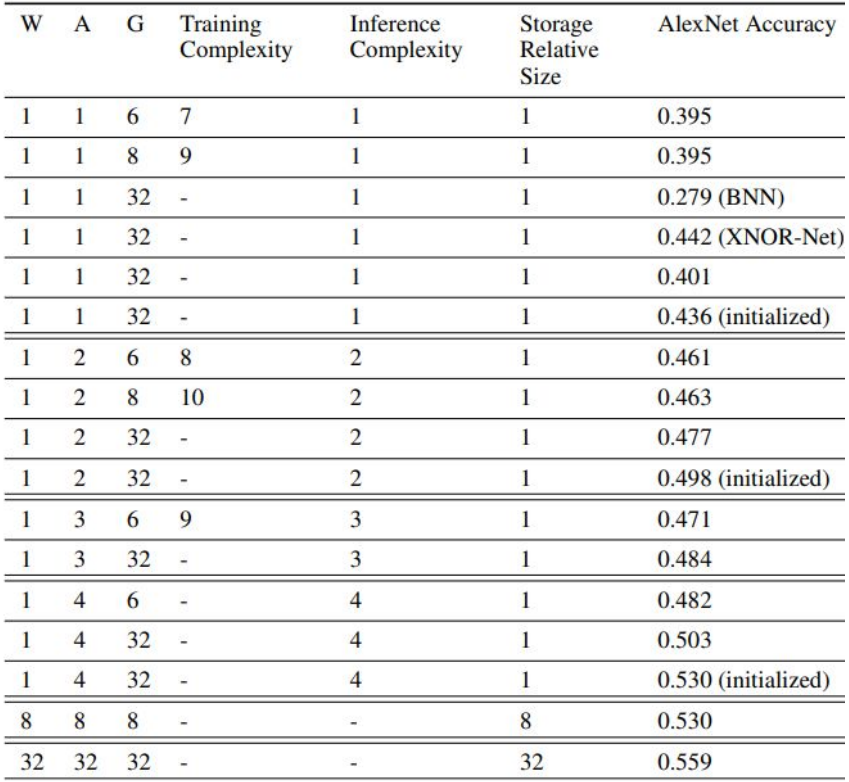
- Linear quantize the weights and activations
- Apply stochastic quantization for the gradients.

NYU SAI LAB

Zhou, Shuchang, et al. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients." *arXiv preprint arXiv:1606.06160* (2016).

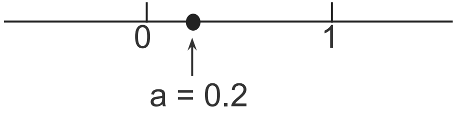
DoReFaNet

- Usually gradients requires far more bitwidth than weight and activation.
- Usually gradient requires **stochastic quantization**.



W	A	G	Training Complexity	Inference Complexity	Storage Relative Size	AlexNet Accuracy
1	1	6	7	1	1	0.395
1	1	8	9	1	1	0.395
1	1	32	-	1	1	0.279 (BNN)
1	1	32	-	1	1	0.442 (XNOR-Net)
1	1	32	-	1	1	0.401
1	1	32	-	1	1	0.436 (initialized)
1	2	6	8	2	1	0.461
1	2	8	10	2	1	0.463
1	2	32	-	2	1	0.477
1	2	32	-	2	1	0.498 (initialized)
1	3	6	9	3	1	0.471
1	3	32	-	3	1	0.484
1	4	6	-	4	1	0.482
1	4	32	-	4	1	0.503
1	4	32	-	4	1	0.530 (initialized)
8	8	8	-	-	8	0.530
32	32	32	-	-	32	0.559

Deterministic and Stochastic Quantization



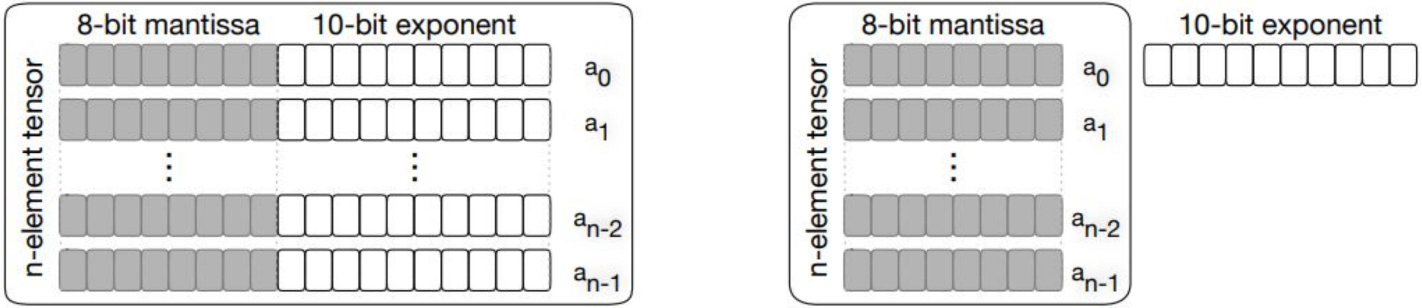
- To quantize a, conventional linear quantization will make q(a) = 0. However, this will cause a bias.
- With stochastic quantization:

$$q(a) = \begin{cases} 1 & \text{for } p = 0.2 \\ 0 & \text{for } p = 0.8 \end{cases}$$

- Stochastic quantization is extremely useful when applying quantization to accelerate DNN training.

NYU SAI LAB

Training DNNs with Hybrid BFP



(a) FP repr. with an exponent per tensor element.

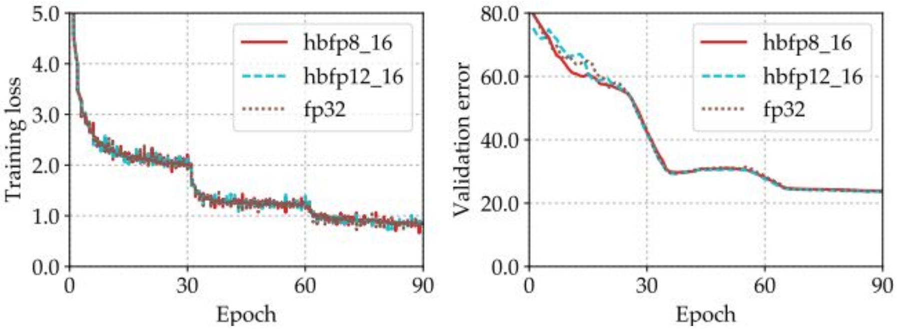
(b) BFP repr. with an exponent per tensor.

- Block floating point format achieves a better hardware efficiency and comparable representation capability than FP.

Drumond, Mario, et al. "Training dnns with hybrid block floating point." *Advances in Neural Information Processing Systems* 31 (2018).

Training DNNs with Hybrid BFP

- Use BFP in all dot-product-based operations present in DNNs (i.e., convolutions, matrix multiplications, and outer products), and floating-point representations for all other operations (i.e., activations, regularizations, etc).
- To minimize data loss in long-lasting training state, the weights are stored with wider mantissas.



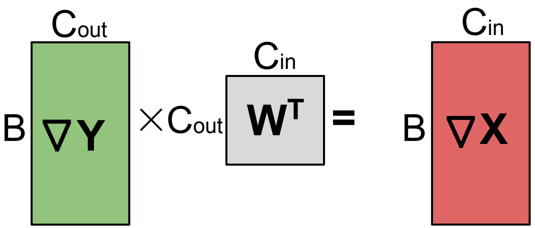
- ResNet-50 trained on ImageNet for 90 epochs.
- 8 bit mantissa, 16 bits weight seems to achieve comparable performance as FP32. A mantissa bitwidth of 12 achieves an even better performance.
- A tile size of 24.

NYU SAI LAB

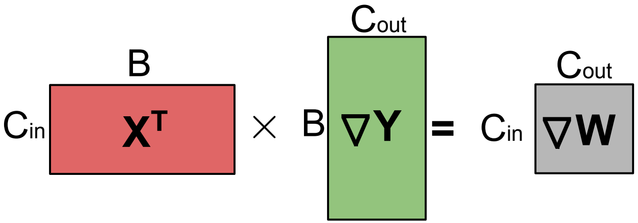
Drumond, Mario, et al. "Training dnns with hybrid block floating point." *Advances in Neural Information Processing Systems* 31 (2018).

Two Copies of Weights

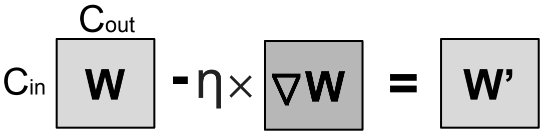
Input data gradient Computation



Weight Gradient Computation

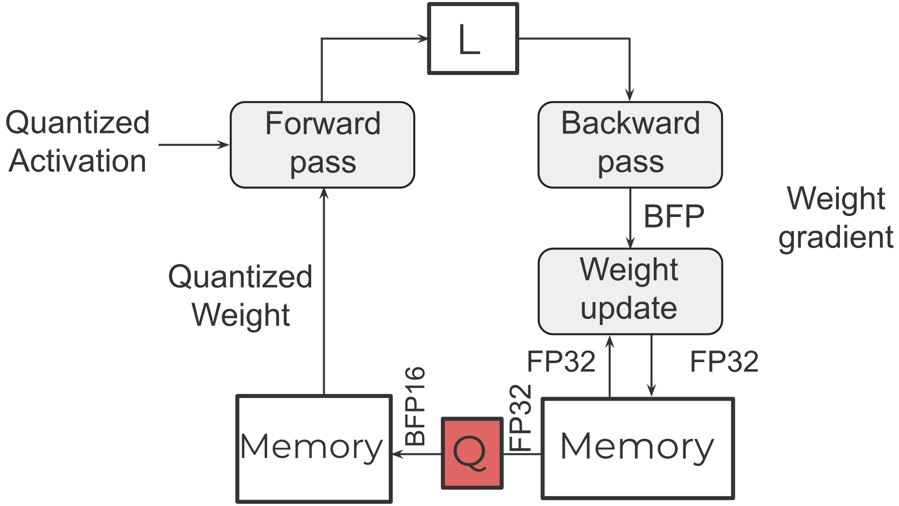


Weight Gradient Updates



- Gradient and forward propagation are performed using BFP.
- Weights are updated using FP.
- Two copies of weights are used.

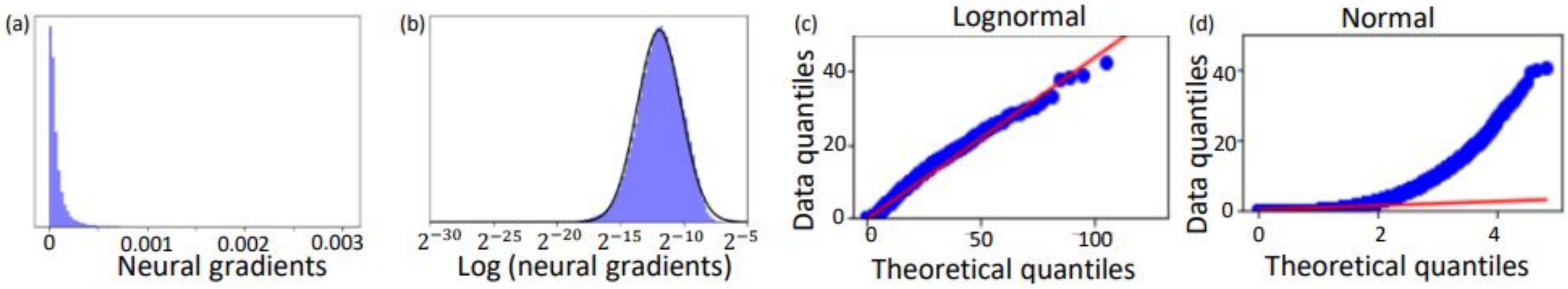
Two Copies of Weights



- Two pieces of copies are needed to be kept in the memory.
- The weight updates are usually performed with higher precision (e.g., FP16).

NYU SAI LAB

Neural Gradients are Near-Lognormal: Improved Quantized and Sparse Training



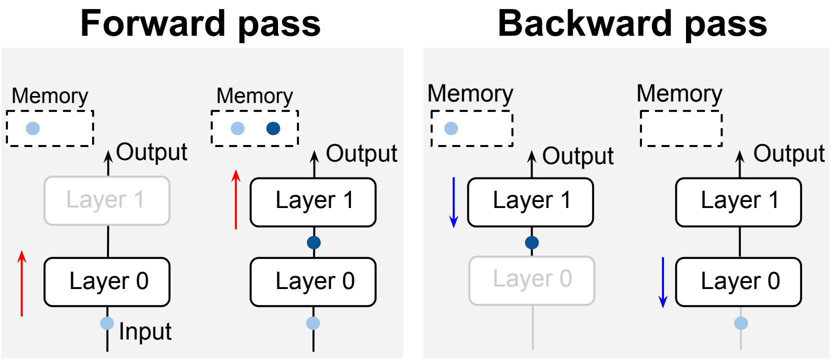
- The distribution of neural gradients is approximately lognormal.
- We can use lognormal regression to determine the optimal quantization setting (e.g., bitwidth, quantization interval).

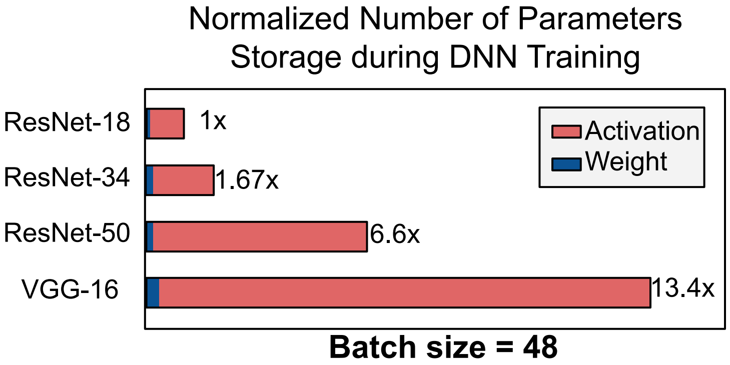
Chmiel, Brian, et al. "Neural gradients are near-lognormal: improved quantized and sparse training." *arXiv preprint arXiv:2006.08173* (2020).

Topics

- Efficient training of DNNs
  - Efficient computing
  - Efficient storage
- Parameter efficient finetuning
- Speculative Decoding

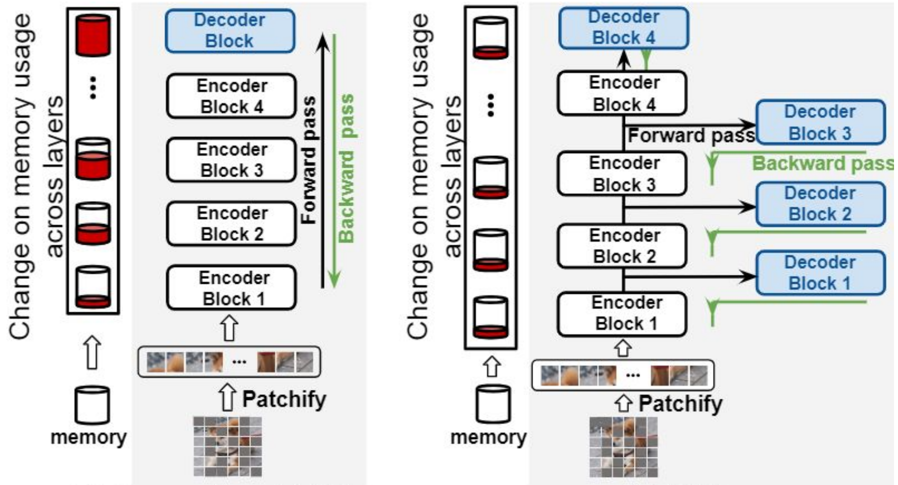
Memory Consumption During Training





- The memory footprint grows proportional with the layer depth. The activation in the early DNN layers need to be stored for a long time.
- Activations consume most of the memory space, approximately 13 times larger than the weights on average.

BIM: Block-Wise Local Learning with Masked Image Modeling

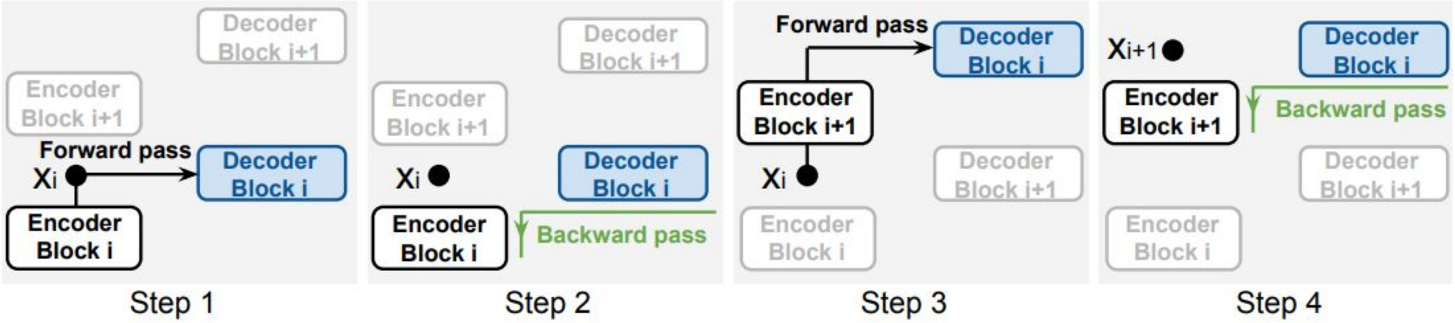


(a) Conventional MIM (b) BIM

- Local exist is introduced during the training process.
- The intermediate results can be discarded once the training process for the current layer is complete.

Luo, Yixuan, Mengye Ren, and Sai Qian Zhang. "BIM: Block-Wise Self-Supervised Learning with Masked Image Modeling." *arXiv preprint arXiv:2311.17218* (2023).

BIM: Block-Wise Local Learning with Masked Image Modeling



- Once the parameter updates in encoder block i and decoder block i are finished, all intermediate features stored in the buffer, except for xi , can be cleared from memory, preserving them for future use.

NYU SAI LAB

Luo, Yixuan, Mengye Ren, and Sai Qian Zhang. "BIM: Block-Wise Self-Supervised Learning with Masked Image Modeling." *arXiv preprint arXiv:2311.17218* (2023).

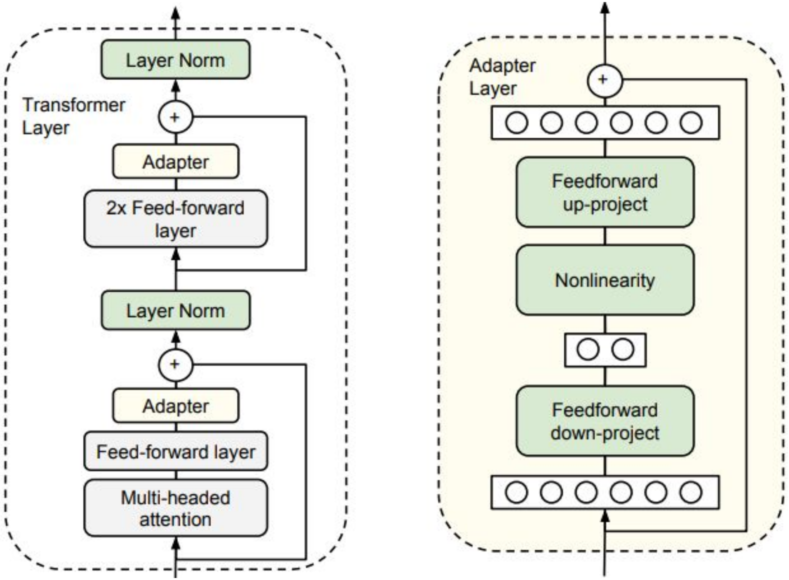
Topics

- Efficient training of DNNs
  - Efficient computing
  - Efficient storage
- Parameter efficient finetuning
- Speculative Decoding

Parameter-efficient Finetuning (PEFT)

- Large models (LMs), often consisting of billions of parameters, require vast amounts of computational resources for execution.
- The expansive scale and computational demands pose considerable challenges when customizing them for particular downstream tasks.
- To better adapt the LMs over the downstream tasks, we can finetune a small portion of the LM parameters. This will make LMs achieve great performance over the downstream tasks while minimizing the training cost.
- Some of the popular PEFT Algorithms:
  - LoRA
  - Adapter
  - BitFit

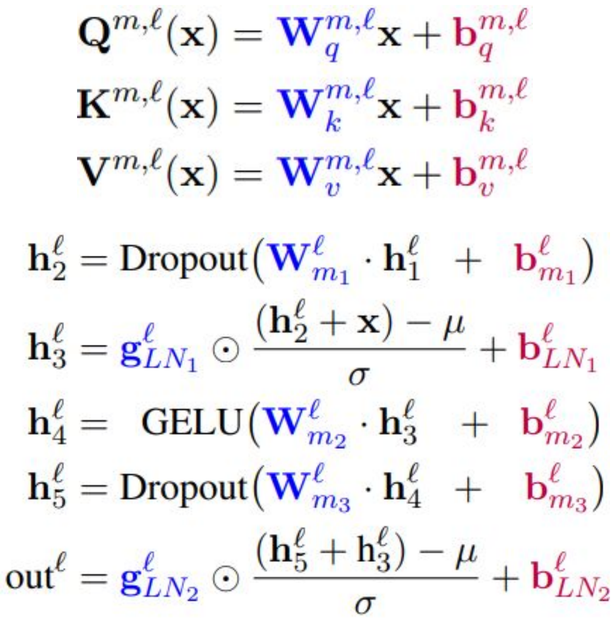
Parameter-Efficient Transfer Learning for NLP



- We add the adapter module twice to each Transformer layer.
- The adapter consists of a bottleneck which contains few parameters relative to the attention and feedforward layers in the original model. The adapter also contains a skip-connection.
- The learnable parameters contributes to around 0.5 − 8% of the parameters of the original model.

Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP." *International conference on machine learning*. PMLR, 2019.

BitFit



$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$

$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$

$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

$$\mathbf{h}_2^\ell = \text{Dropout}\big(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\big)$$

$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell$$

$$\mathbf{h}_4^\ell = \text{GELU}\big(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\big)$$

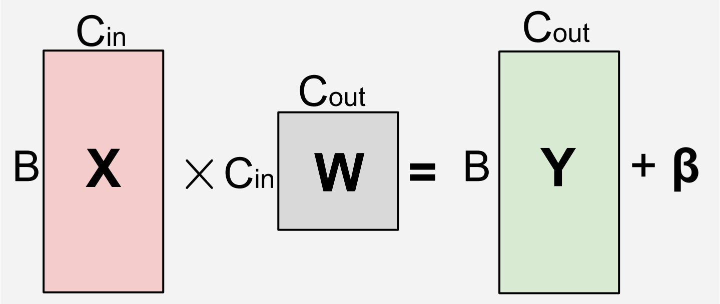
$$\mathbf{h}_5^\ell = \text{Dropout}\big(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\big)$$

$$\text{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell$$

- BitFiT is a sparse-finetuning method where only the bias-terms of the model are being modified.
- Applying BitFit on pre-trained BERT models is competitive with (and sometimes better than) fine-tuning the entire model.
- Bias parameters make up 0.09% of the total number of parameters in BER.

Zaken, Elad Ben, Shauli Ravfogel, and Yoav Goldberg. "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models." *arXiv preprint arXiv:2106.10199* (2021).

Finetune Bias is Cheap



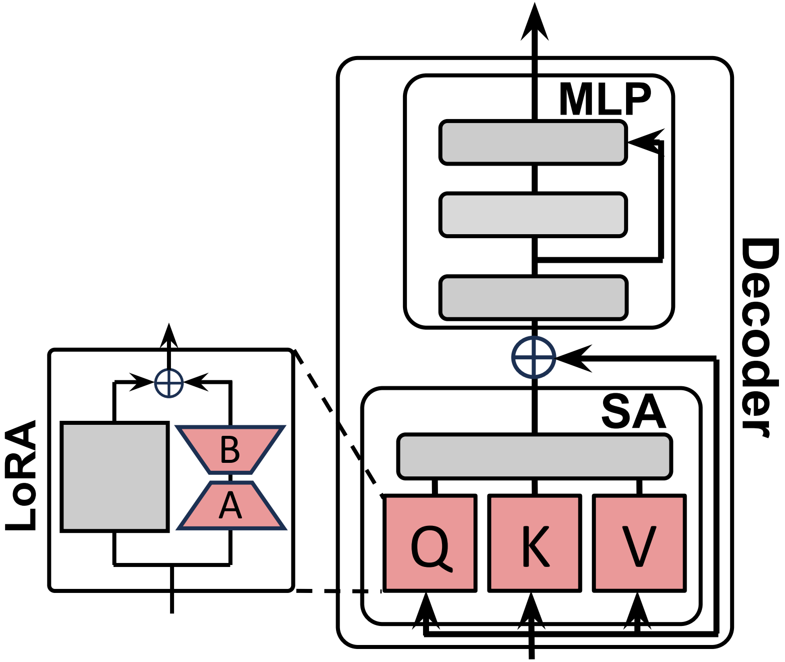
$$\frac{dL}{d\beta} = \sum_{y \in Y} \frac{dL}{dy}$$

- Updating the bias does not require buffering any intermediate results during the forward pass of DNN training.

NYU SAI LAB

Cai, Han, et al. "Tinytl: Reduce activations, not trainable parameters for efficient on-device learning." *arXiv preprint arXiv:2007.11622* (2020).

Low-rank Adaptation (LoRA)

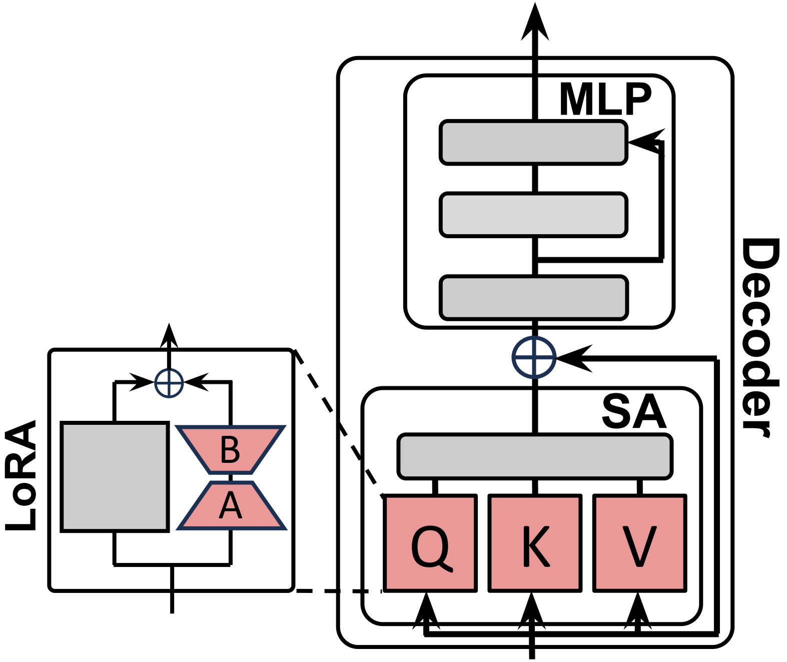


- **LoRA (Low-Rank Adaptation)** is a **PEFT** method for large pre-trained models. Instead of updating all model weights during fine-tuning, LoRA inserts small trainable low-rank matrices into specific layers (usually the attention projections).

- This dramatically reduces memory and compute requirements while maintaining near full fine-tuning performance.

Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

Low-rank Adaptation (LoRA)



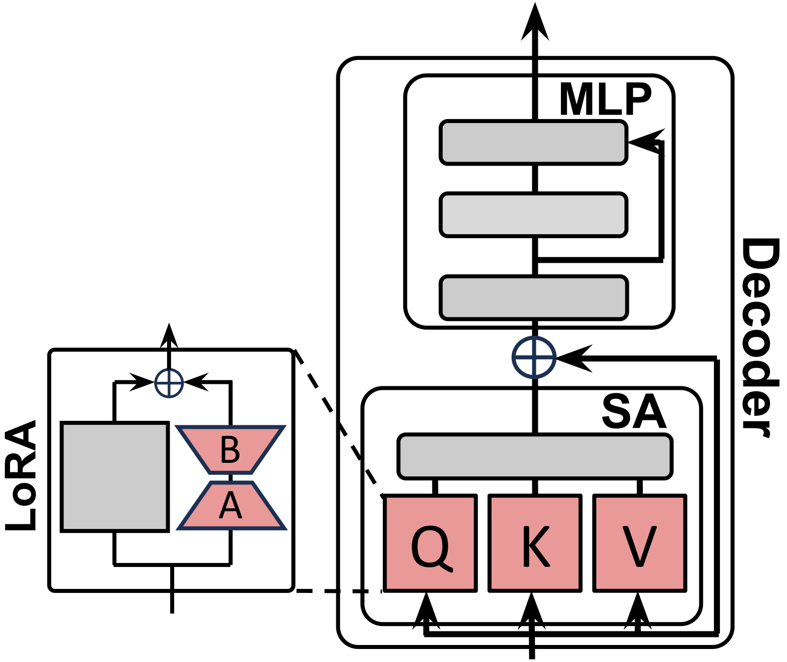
$$h = W_0 x + \Delta W x = W_0 x + BAx$$

- Only the weights within the red blocks are updated.
- Assume the weight matrix has a dimension of E×E, A and B have a size of E×r and r×E, where r << k (e.g., r=4).
- BA can be merged with the original weight W_0, leading to no additional computational and storage cost.

NYU SAI LAB

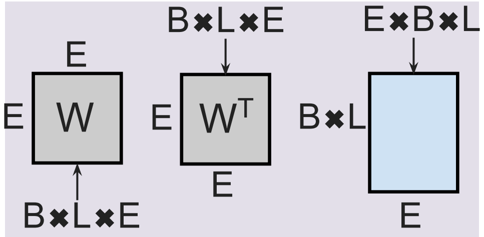
Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

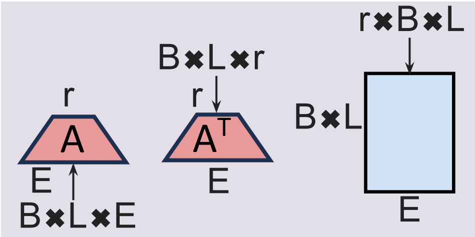
Low-rank Adaptation (LoRA)



$$h = W_0 x + \Delta W x = W_0 x + BAx$$

- Compared with finetuning the entire W_Q, W_K and W_V, this will lead to great compute savings:
  - $3BLE^2$
  - $6BLrE$

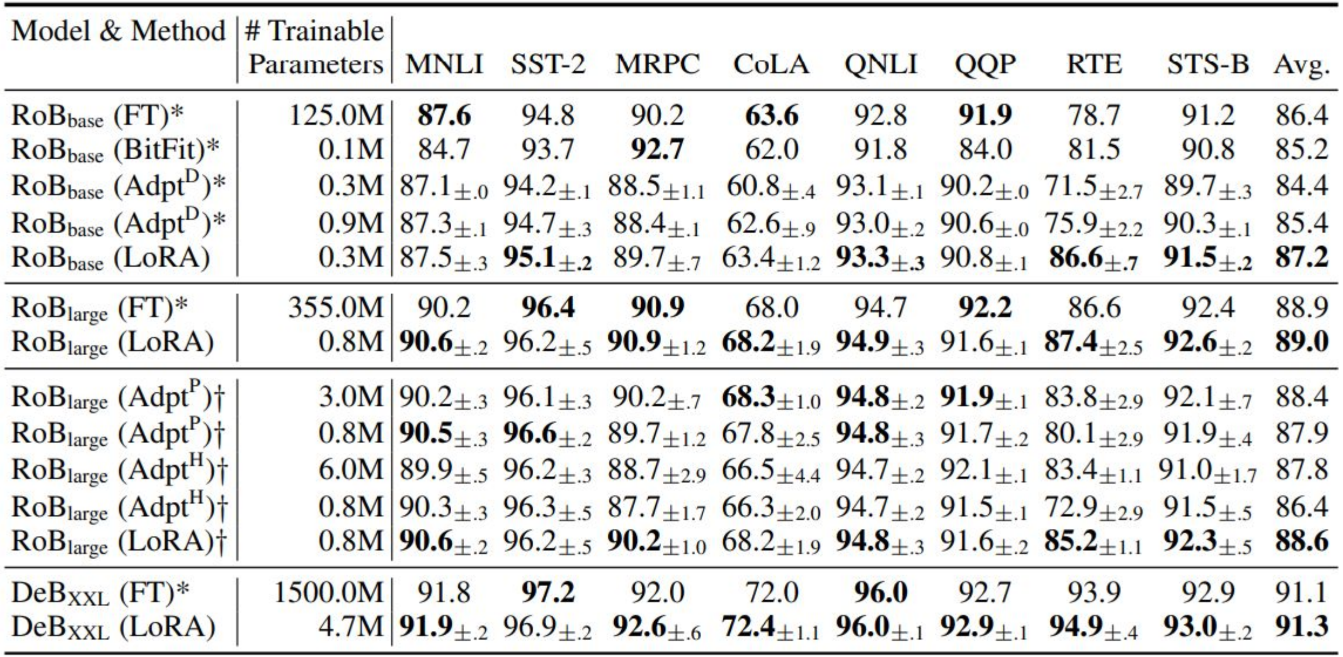




NYU SAI LAB

Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

Low-rank Adaptation (LoRA)



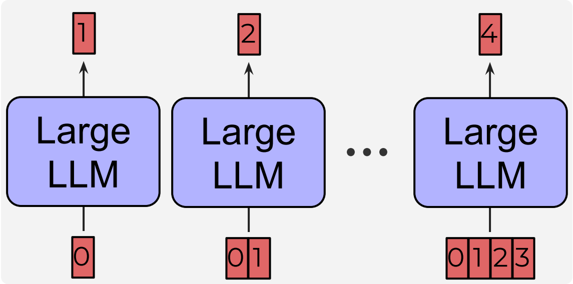
Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB$_{\text{base}}$ (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB$_{\text{base}}$ (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB$_{\text{base}}$ (Adpt$^{\text{D}}$)*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB$_{\text{base}}$ (Adpt$^{\text{D}}$)*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB$_{\text{base}}$ (LoRA)	0.3M	$87.5_{\pm.3}$	**95.1**$_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	**93.3**$_{\pm.3}$	$90.8_{\pm.1}$	**86.6**$_{\pm.7}$	**91.5**$_{\pm.2}$	**87.2**
RoB$_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
RoB$_{\text{large}}$ (LoRA)	0.8M	**90.6**$_{\pm.2}$	$96.2_{\pm.5}$	**90.9**$_{\pm1.2}$	**68.2**$_{\pm1.9}$	**94.9**$_{\pm.3}$	$91.6_{\pm.1}$	**87.4**$_{\pm2.5}$	**92.6**$_{\pm.2}$	**89.0**
RoB$_{\text{large}}$ (Adpt$^{\text{P}}$)†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	**68.3**$_{\pm1.0}$	**94.8**$_{\pm.2}$	**91.9**$_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
RoB$_{\text{large}}$ (Adpt$^{\text{P}}$)†	0.8M	**90.5**$_{\pm.3}$	**96.6**$_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	**94.8**$_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
RoB$_{\text{large}}$ (Adpt$^{\text{H}}$)†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
RoB$_{\text{large}}$ (Adpt$^{\text{H}}$)†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
RoB$_{\text{large}}$ (LoRA)†	0.8M	**90.6**$_{\pm.2}$	$96.2_{\pm.5}$	**90.2**$_{\pm1.0}$	$68.2_{\pm1.9}$	**94.8**$_{\pm.3}$	$91.6_{\pm.2}$	**85.2**$_{\pm1.1}$	**92.3**$_{\pm.5}$	**88.6**
DeB$_{\text{XXL}}$ (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB$_{\text{XXL}}$ (LoRA)	4.7M	**91.9**$_{\pm.2}$	$96.9_{\pm.2}$	**92.6**$_{\pm.6}$	**72.4**$_{\pm1.1}$	**96.0**$_{\pm.1}$	**92.9**$_{\pm.1}$	**94.9**$_{\pm.4}$	**93.0**$_{\pm.2}$	**91.3**

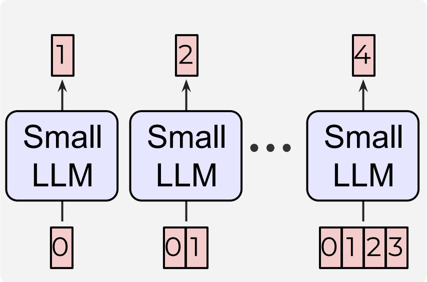
- LoRA achieves better results than Adapter and BitFit.

Topics

- Efficient training of DNNs
    - Efficient computing
    - Efficient storage
- Parameter efficient finetuning
- Speculative Decoding

Speculative Decoding





Accurate but slow

$T_{tot} = NT_1$

Fast but inaccurate

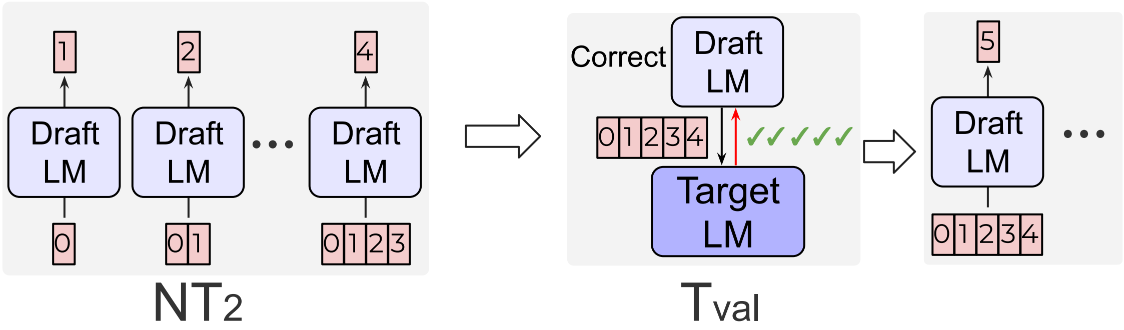
$T_{tot} = NT_2$

- Speculative decoding enables lossless token generation with low latency.

NYU SAI LAB

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

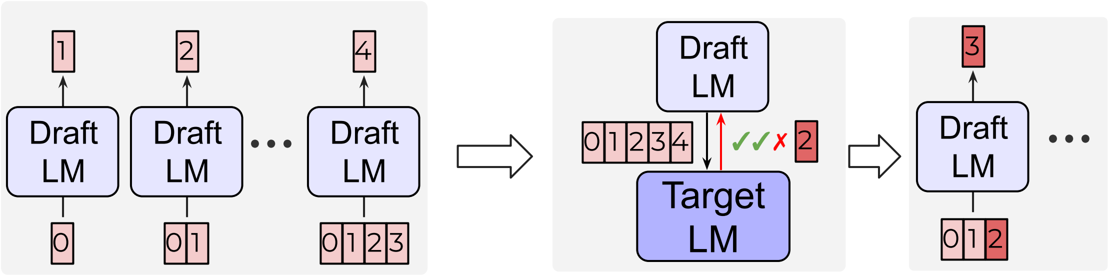
Speculative Decoding



$$T_{tot} = NT_2 + T_{val} < NT_{p,1}$$

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

Speculative Decoding

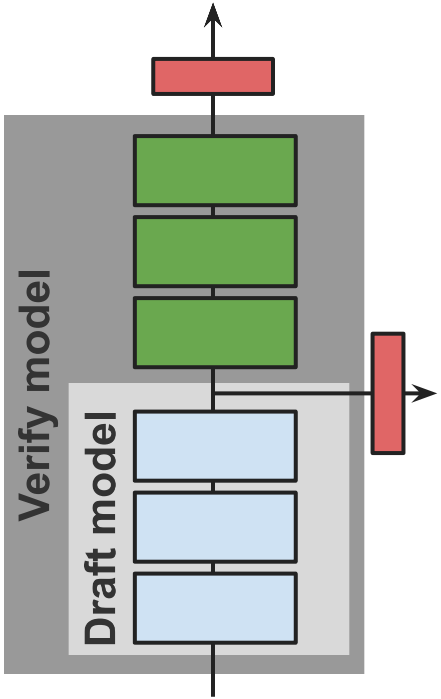


- If the token is incorrect, the target model provides the correct token to the draft model to help it generate subsequent tokens more accurately.
- If the amount of tokens that pass the verification is too low, then it is possible that speculative decoding is slower than autoregressive baseline.

NYU SAI LAB

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

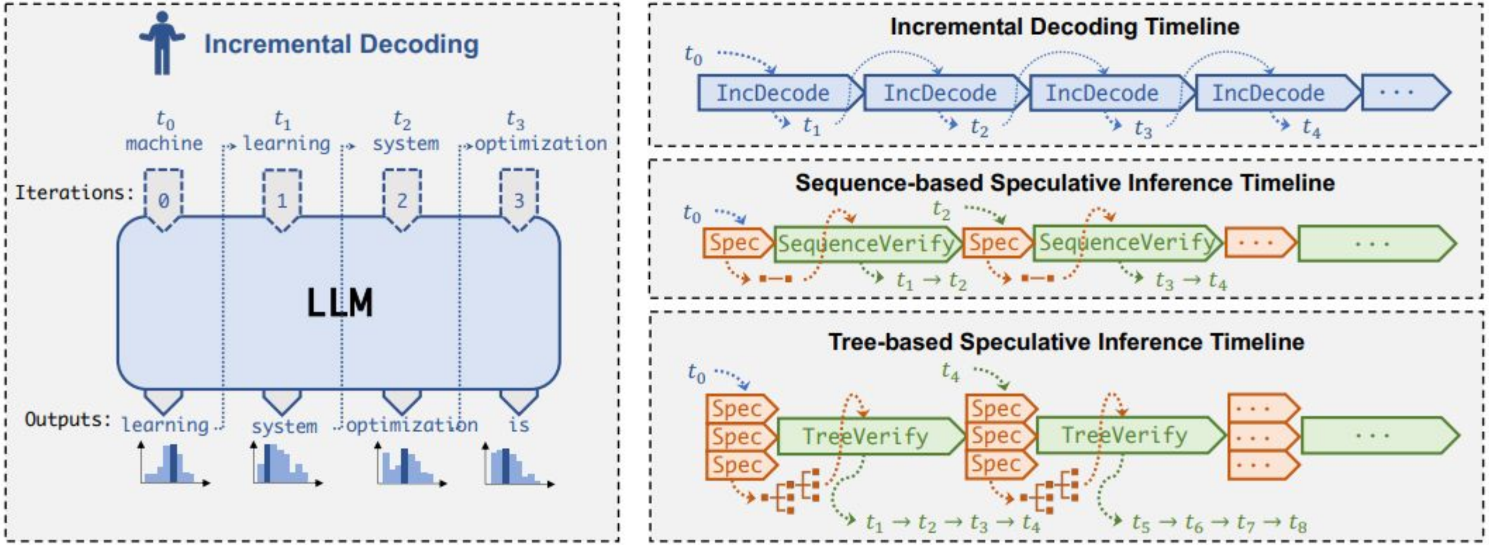
Self-Speculative Decoding



- Self-Speculative decoding the draft model is a subnetwork of the verify model. All the intermediate results from the draft model are reusable.
- No additional network needs to be trained, except a simple classification layer.

Zhang, Jun, et al. "Draft & verify: Lossless large language model acceleration via self-speculative decoding." *arXiv preprint arXiv:2309.08168* (2023).
Elhoushi, Mostafa, et al. "Layer skip: Enabling early exit inference and self-speculative decoding." *arXiv preprint arXiv:2404.16710* (2024).

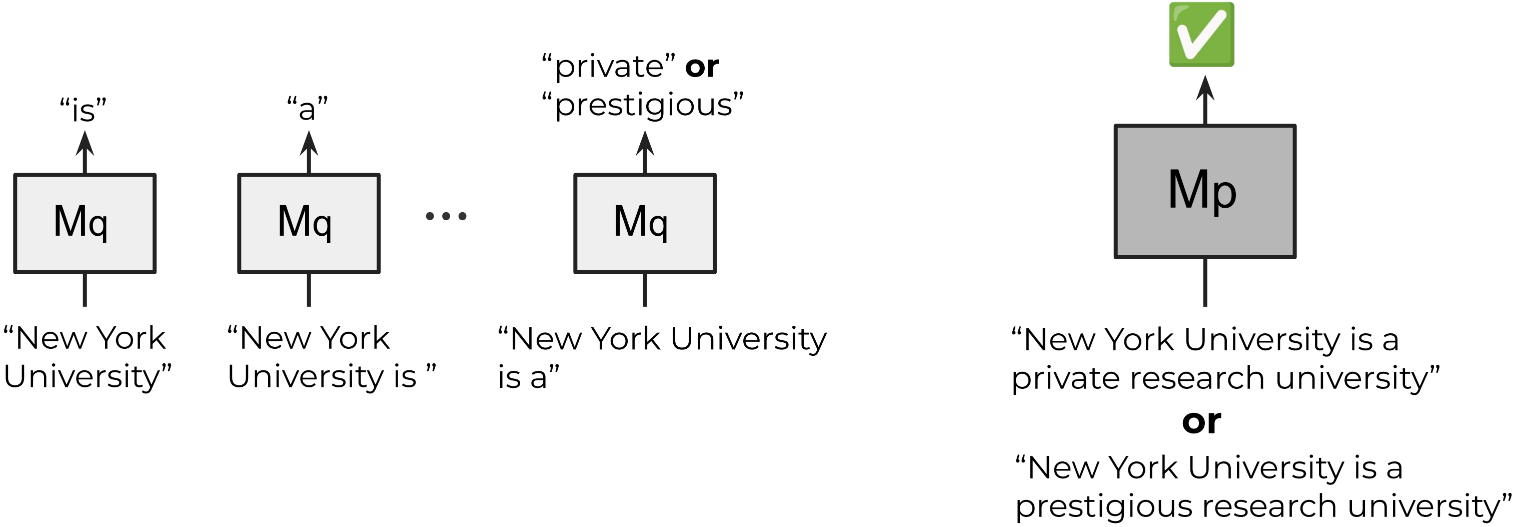
SpecInfer



(a) Incremental decoding. **(b)** Timeline Comparison.

Miao, Xupeng, et al. "SpecInfer: Accelerating Generative Large Language Model Serving with Tree-based Speculative Inference and Verification." *arXiv preprint arXiv:2305.09781* (2023).

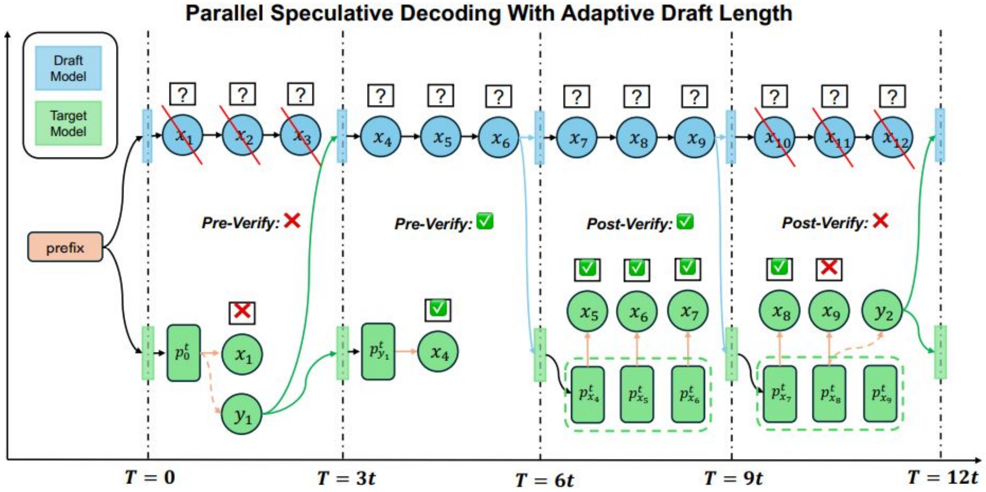
SpecInfer


“is”
Mq
“New York University”
“a”
Mq
“New York University is ”
…
“private” **or** “prestigious”
Mq
“New York University is a”
✅
Mp
“New York University is a private research university”
or
“New York University is a prestigious research university”

NYU SAI LAB

Miao, Xupeng, et al. "SpecInfer: Accelerating Generative Large Language Model Serving with Tree-based Speculative Inference and Verification." *arXiv preprint arXiv:2305.09781* (2023).

Parallel Speculative Decoding

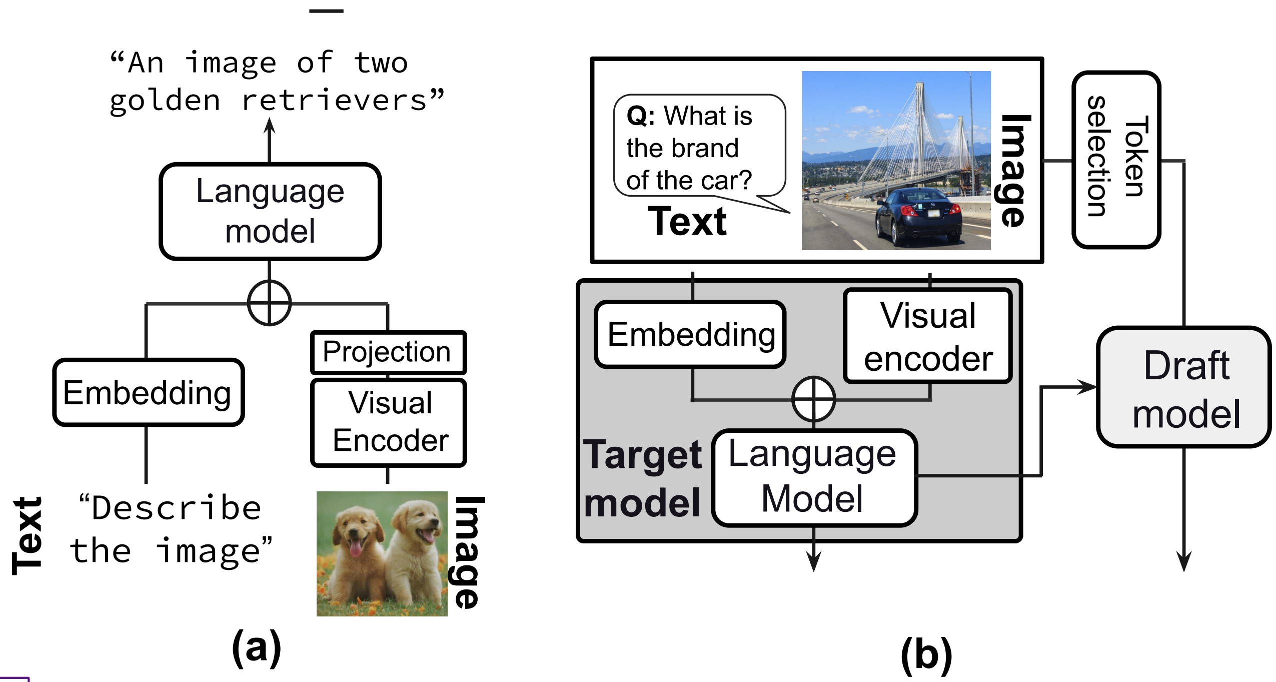


Parallel Speculative Decoding With Adaptive Draft Length

- PEARL is a parallel inference framework based on speculative decoding which utilizes pre-verify and post-verify to achieve adaptive draft length.
- The draft model continues to decode during the verification stage.
- If the verification fails, the windows size will become 1 in the next cycle.

Liu, Tianyu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, and Winston Hu. "Parallel speculative decoding with adaptive draft length." arXiv preprint arXiv:2408.11850 (2024).

DREAM

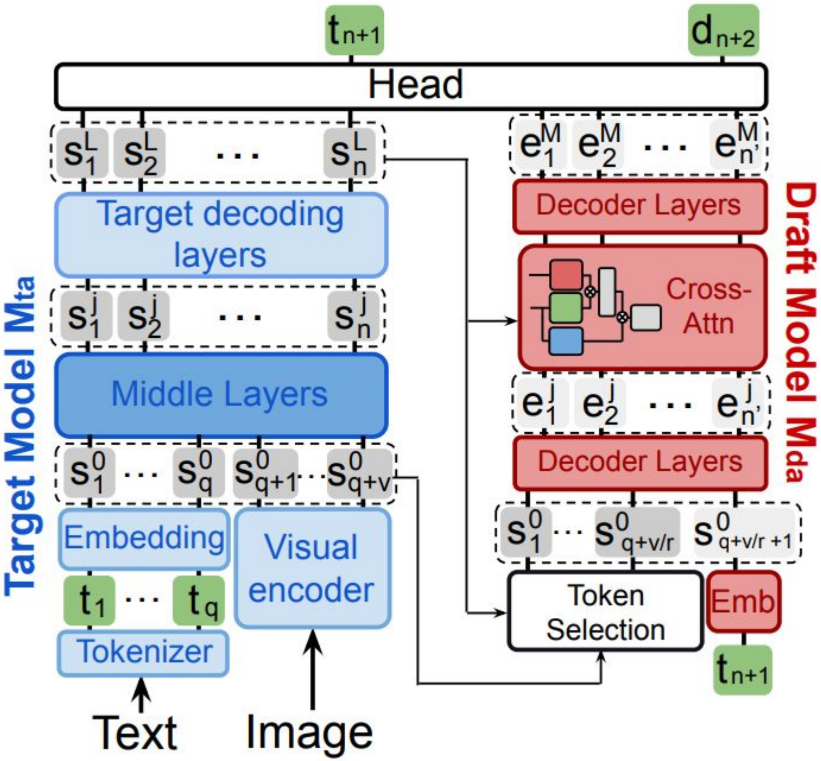


Hu, Yunhai, et al. "DREAM: Drafting with Refined Target Features and Entropy-Adaptive Cross-Attention Fusion for Multimodal Speculative Decoding." *arXiv preprint arXiv:2505.19201* (2025).

DREAM

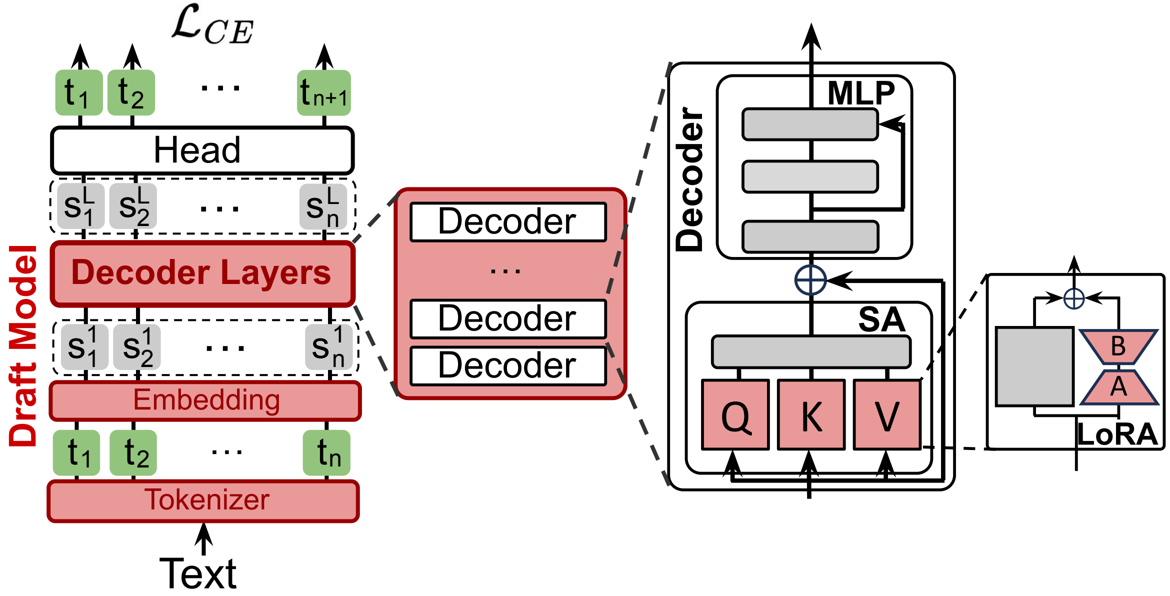
- During operation, the target model will send their intermediate results to the draft model to better guide the generation of the draft model.

- The visual tokens will also be pruned to remove the redundant tokens to reduce the processing latency of the draft model.



Hu, Yunhai, et al. "DREAM: Drafting with Refined Target Features and Entropy-Adaptive Cross-Attention Fusion for Multimodal Speculative Decoding." *NeurIPS* (2025).

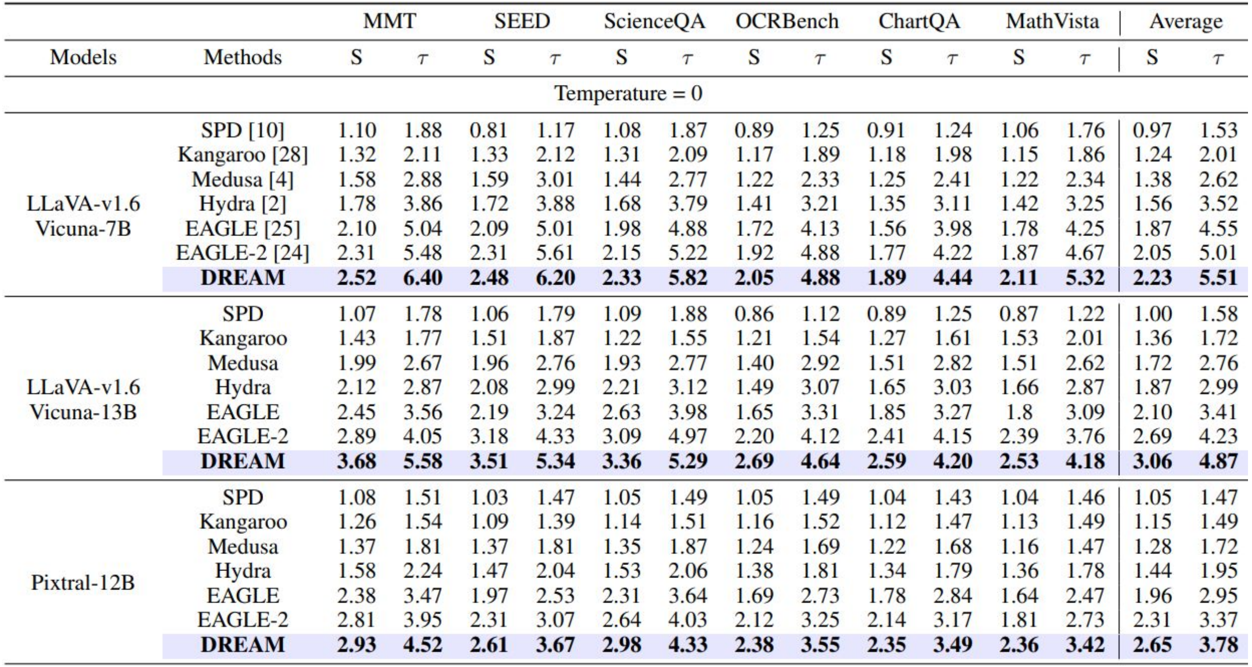
Speculative Decoding with Finetuning



- The draft model will be trained using the dataset with cross-entropy loss to achieve a better acceptance ratio.

Hu, Yunhai, et al. "DREAM: Drafting with Refined Target Features and Entropy-Adaptive Cross-Attention Fusion for Multimodal Speculative Decoding." *NeurIPS* (2025).

DREAM



		MMT		SEED		ScienceQA		OCRBench		ChartQA		MathVista		Average	
Models	Methods	S	τ	S	τ	S	τ	S	τ	S	τ	S	τ	S	τ
							Temperature = 0								
LLaVA-v1.6 Vicuna-7B	SPD [10]	1.10	1.88	0.81	1.17	1.08	1.87	0.89	1.25	0.91	1.24	1.06	1.76	0.97	1.53
	Kangaroo [28]	1.32	2.11	1.33	2.12	1.31	2.09	1.17	1.89	1.18	1.98	1.15	1.86	1.24	2.01
	Medusa [4]	1.58	2.88	1.59	3.01	1.44	2.77	1.22	2.33	1.25	2.41	1.22	2.34	1.38	2.62
	Hydra [2]	1.78	3.86	1.72	3.88	1.68	3.79	1.41	3.21	1.35	3.11	1.42	3.25	1.56	3.52
	EAGLE [25]	2.10	5.04	2.09	5.01	1.98	4.88	1.72	4.13	1.56	3.98	1.78	4.25	1.87	4.55
	EAGLE-2 [24]	2.31	5.48	2.31	5.61	2.15	5.22	1.92	4.88	1.77	4.22	1.87	4.67	2.05	5.01
	DREAM	**2.52**	**6.40**	**2.48**	**6.20**	**2.33**	**5.82**	**2.05**	**4.88**	**1.89**	**4.44**	**2.11**	**5.32**	**2.23**	**5.51**
LLaVA-v1.6 Vicuna-13B	SPD	1.07	1.78	1.06	1.79	1.09	1.88	0.86	1.12	0.89	1.25	0.87	1.22	1.00	1.58
	Kangaroo	1.43	1.77	1.51	1.87	1.22	1.55	1.21	1.54	1.27	1.61	1.53	2.01	1.36	1.72
	Medusa	1.99	2.67	1.96	2.76	1.93	2.77	1.40	2.92	1.51	2.82	1.51	2.62	1.72	2.76
	Hydra	2.12	2.87	2.08	2.99	2.21	3.12	1.49	3.07	1.65	3.03	1.66	2.87	1.87	2.99
	EAGLE	2.45	3.56	2.19	3.24	2.63	3.98	1.65	3.31	1.85	3.27	1.8	3.09	2.10	3.41
	EAGLE-2	2.89	4.05	3.18	4.33	3.09	4.97	2.20	4.12	2.41	4.15	2.39	3.76	2.69	4.23
	DREAM	**3.68**	**5.58**	**3.51**	**5.34**	**3.36**	**5.29**	**2.69**	**4.64**	**2.59**	**4.20**	**2.53**	**4.18**	**3.06**	**4.87**
Pixtral-12B	SPD	1.08	1.51	1.03	1.47	1.05	1.49	1.05	1.49	1.04	1.43	1.04	1.46	1.05	1.47
	Kangaroo	1.26	1.54	1.09	1.39	1.14	1.51	1.16	1.52	1.12	1.47	1.13	1.49	1.15	1.49
	Medusa	1.37	1.81	1.37	1.81	1.35	1.87	1.24	1.69	1.22	1.68	1.16	1.47	1.28	1.72
	Hydra	1.58	2.24	1.47	2.04	1.53	2.06	1.38	1.81	1.34	1.79	1.36	1.78	1.44	1.95
	EAGLE	2.38	3.47	1.97	2.53	2.31	3.64	1.69	2.73	1.78	2.84	1.64	2.47	1.96	2.95
	EAGLE-2	2.81	3.95	2.31	3.07	2.64	4.03	2.12	3.25	2.14	3.17	1.81	2.73	2.31	3.37
	DREAM	**2.93**	**4.52**	**2.61**	**3.67**	**2.98**	**4.33**	**2.38**	**3.55**	**2.35**	**3.49**	**2.36**	**3.42**	**2.65**	**3.78**

Hu, Yunhai, et al. "DREAM: Drafting with Refined Target Features and Entropy-Adaptive Cross-Attention Fusion for Multimodal Speculative Decoding." *arXiv preprint arXiv:2505.19201* (2025).

What Makes an Ideal Draft Model?

- Ideally, the draft model should have:
  - High acceptance rate
  - Low execution latency
- This is exactly the goal of DNN pruning, quantization, knowledge distillation, dynamic computing…