MIT HAN LAB

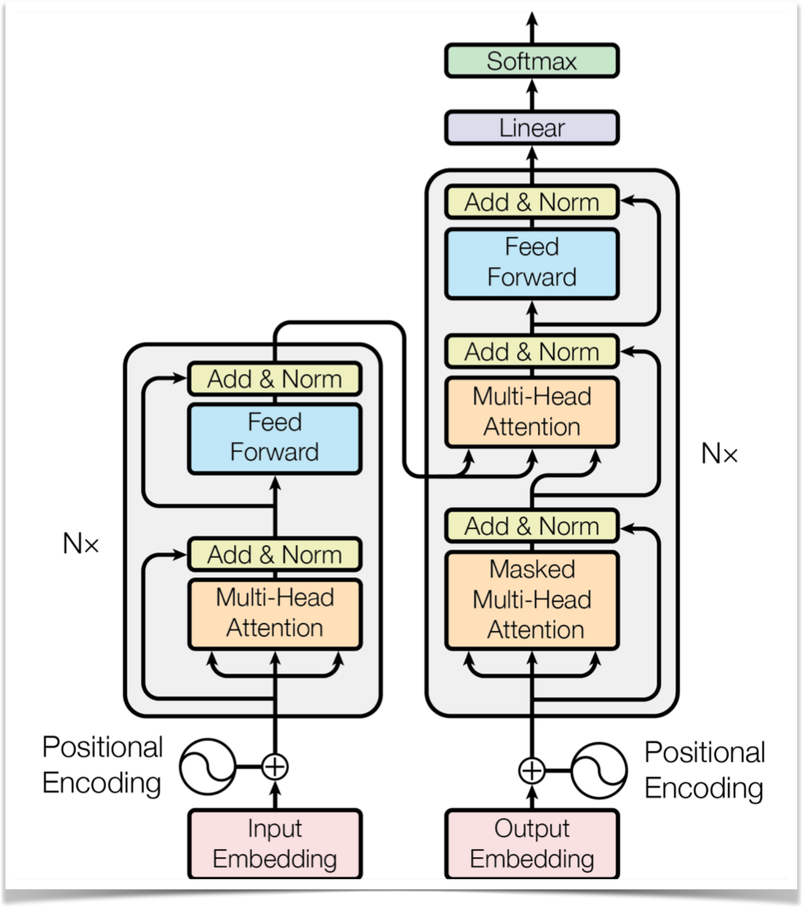
Lecture 20
Efficient Transformers

Song Han
songhan@mit.edu

Lecture Plan

Today we will cover:

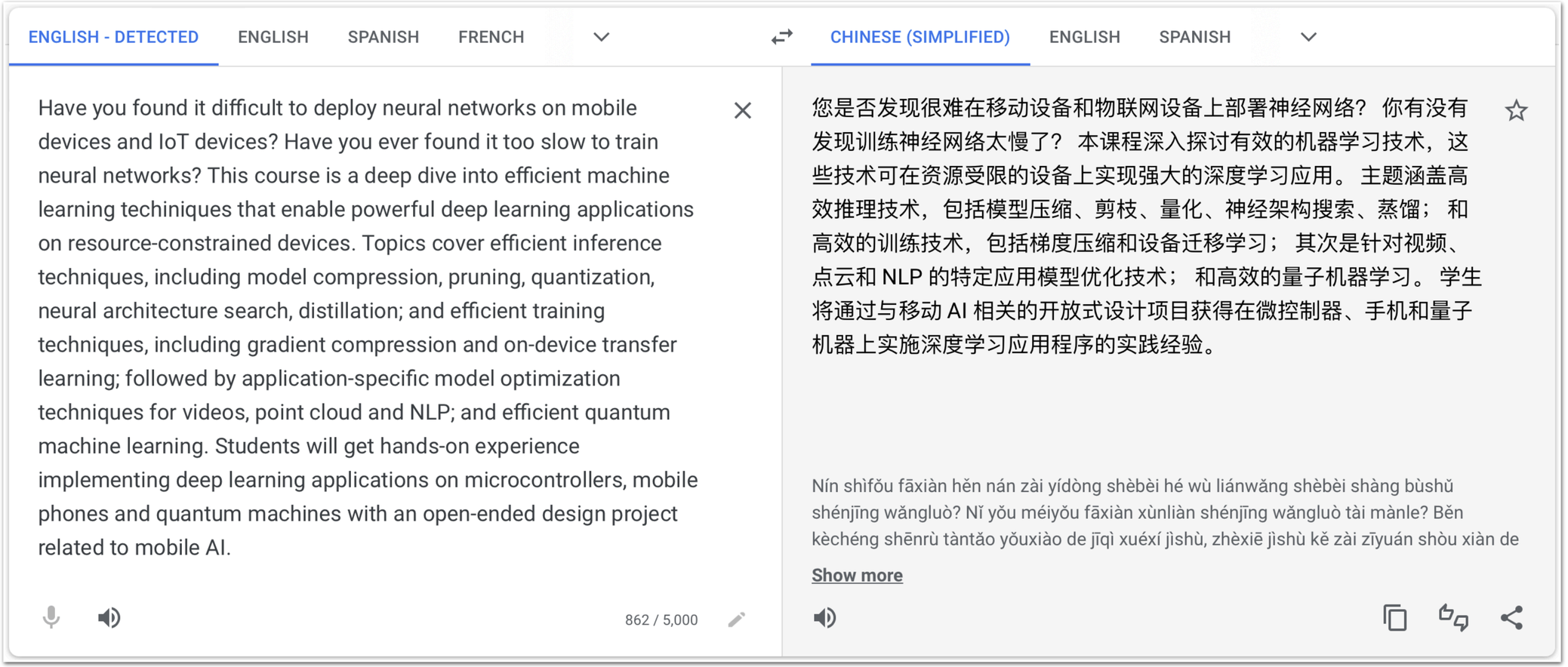
1. Basics and applications of transformers
2. Efficient transformers
3. System support for transformers



I. Transformer Basics

Natural Language Processing

Google Translate — Machine Translation



https://translate.google.com/

Natural Language Processing

Chat Bot — Question Answering

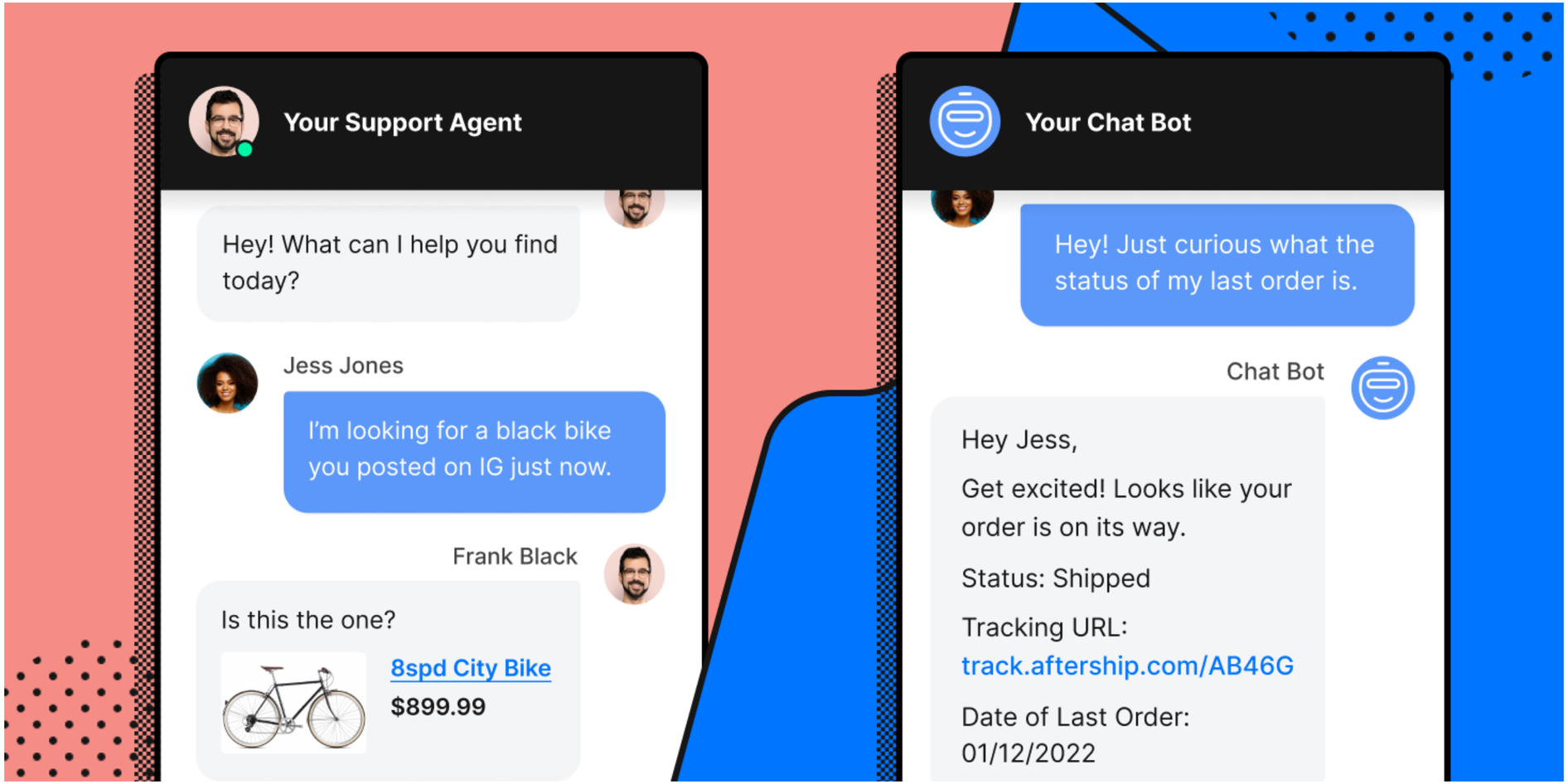


Image credit: https://www.gorgias.com/blog/chatbot-vs-live-chat

Natural Language Processing

GitHub Copilot — Language Modeling

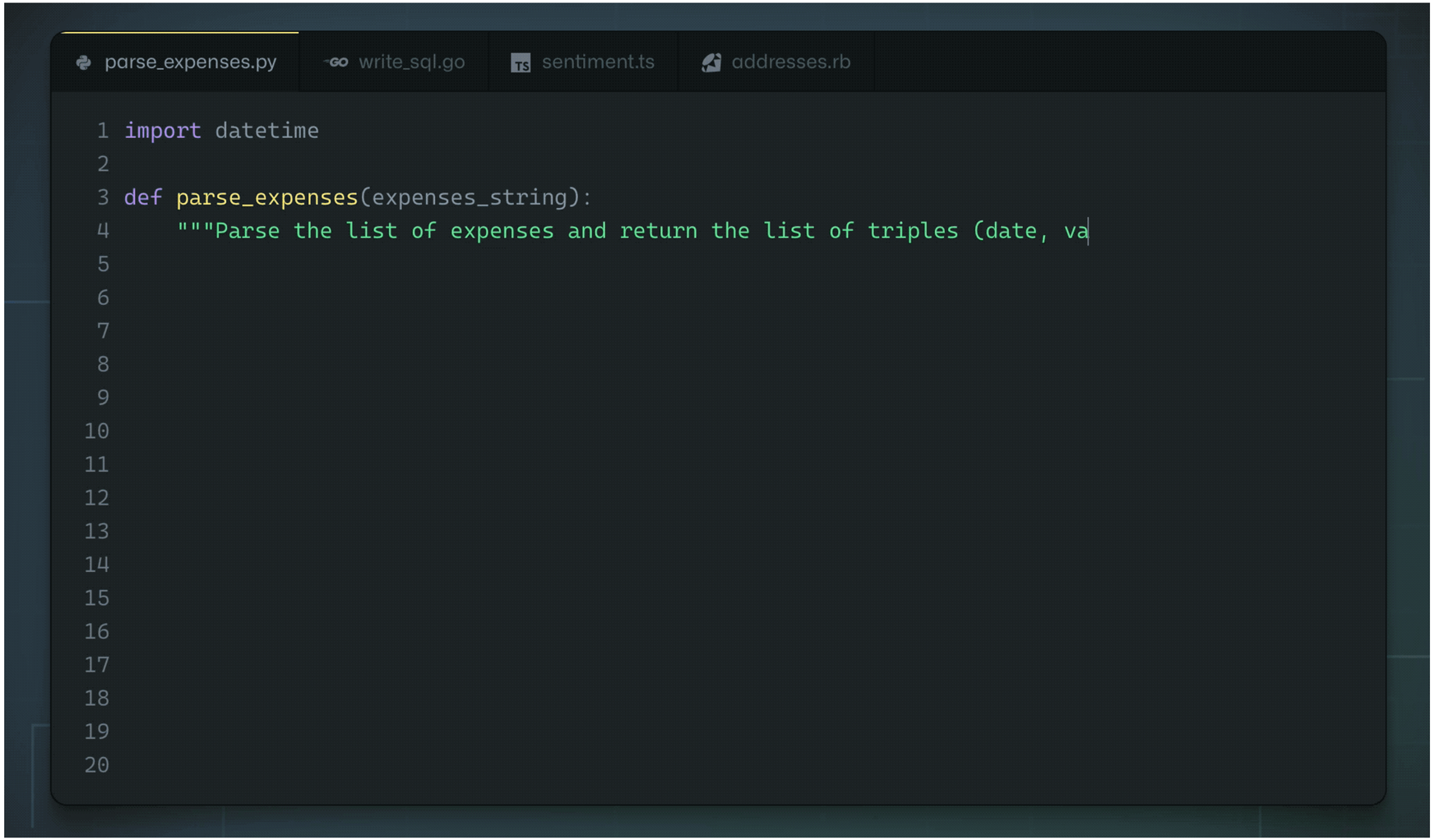
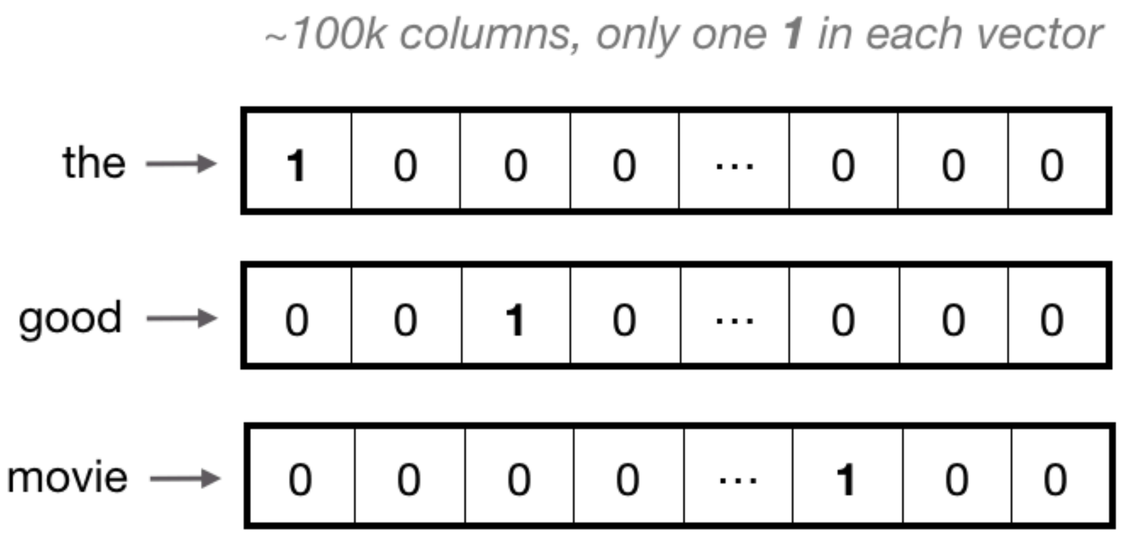


Image credit: https://techcrunch.com/2021/06/29/github-previews-new-ai-tool-that-makes-coding-suggestions/

Word Representation

One-Hot Encoding

- **Key idea**: Representing each word as a vector that has as many values in it as there are words in the vocabulary. Each column in a vector represents one possible word in a vocabulary.



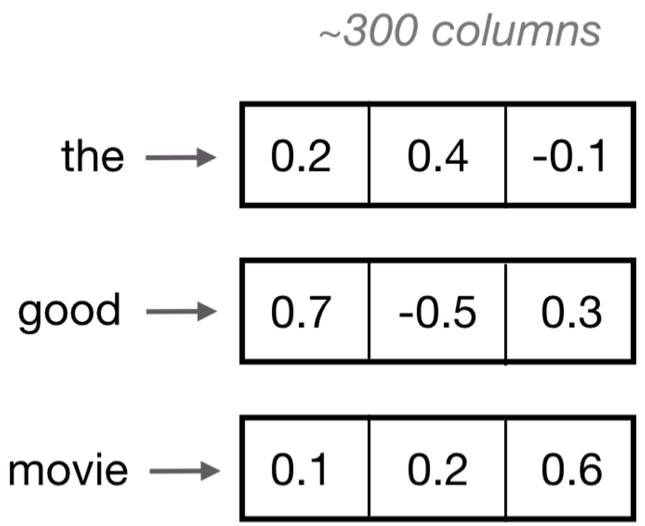
- For **large vocabularies**, these vectors can get very **long**, and they contain all 0's except for one value. This is considered a very sparse representation.

Content credit: https://cezannec.github.io/CNN_Text_Classification/

Word Representation

Word Embedding

- **Key idea**: Map the word index to a **continuous** word embedding through a **look-up table**.

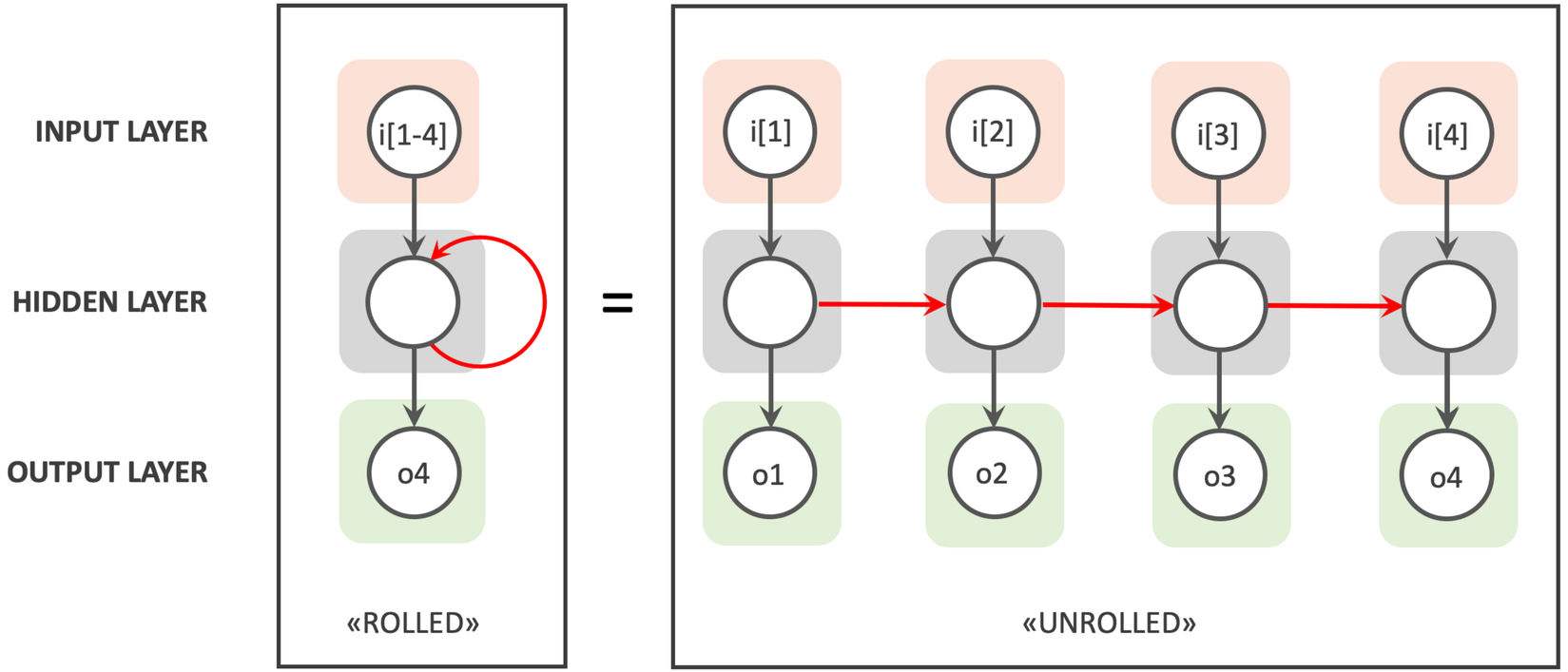


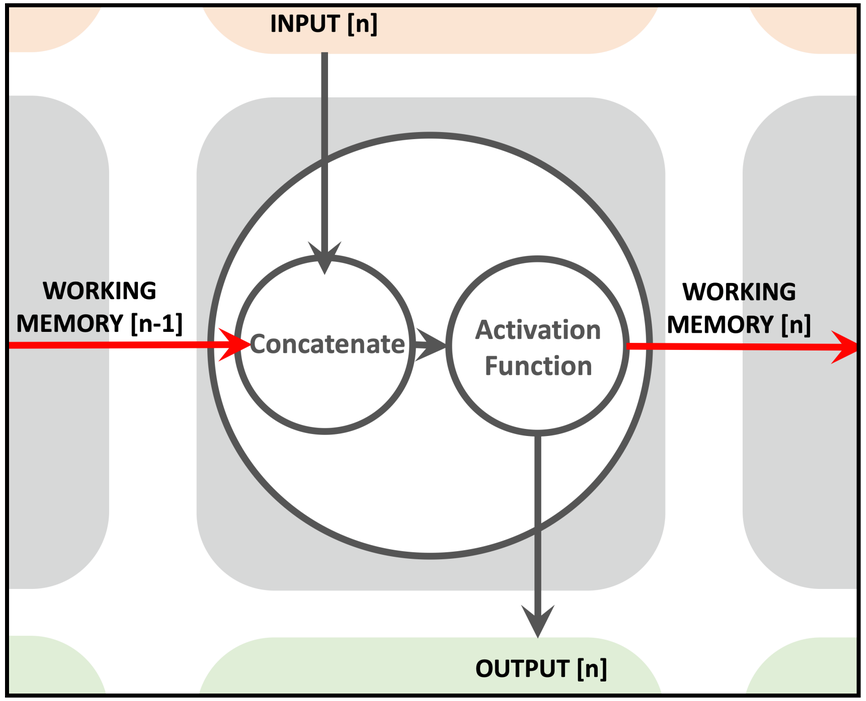
- The word embedding can be trained **end-to-end** with the model for the downstream tasks.
  - Popular pre-trained word embeddings: Word2Vec, GloVe.

Content credit: https://cezannec.github.io/CNN_Text_Classification/

Pre-Transformer Era

Recurrent Neural Networks (RNNs)



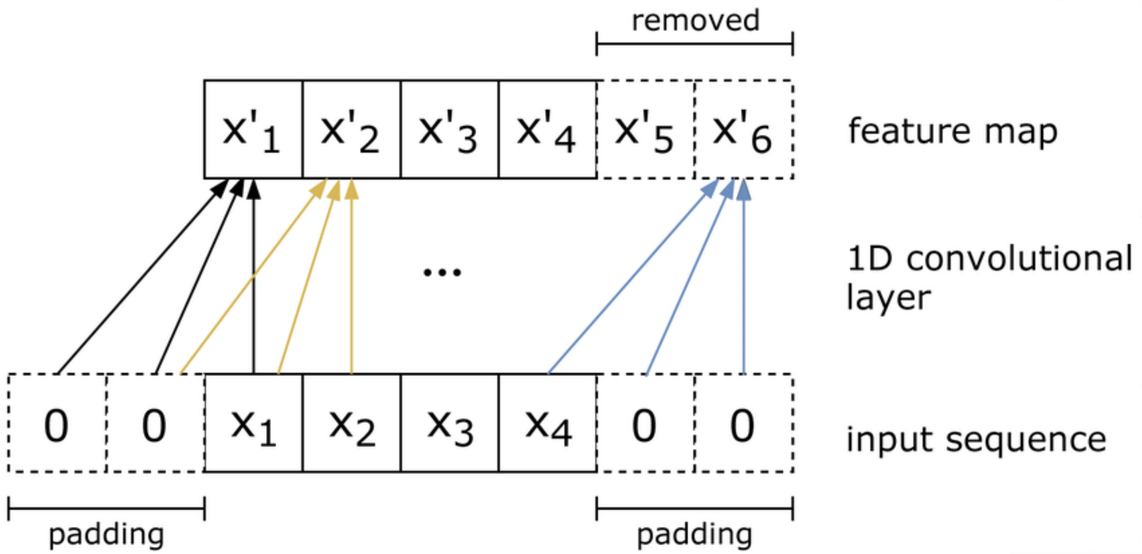


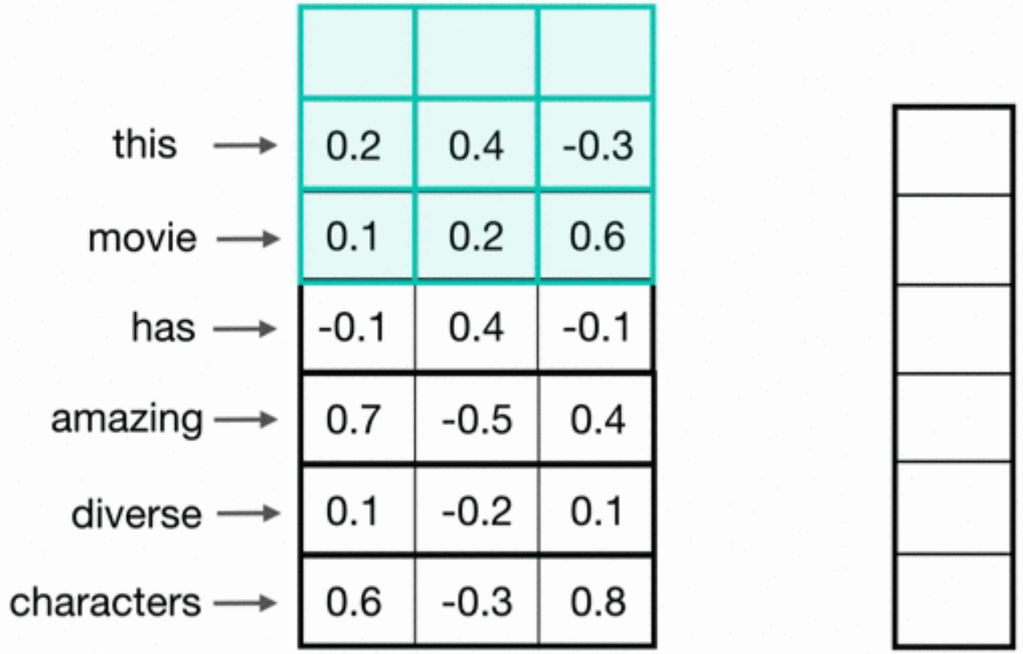
- The "working memory" struggles to retain **long-term** dependancies (can be solved by LSTM).
- There is **strict dependency** across tokens, limiting the scalability.

Content credit: https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks

Pre-Transformer Era

Convolutional Neural Networks (CNNs)

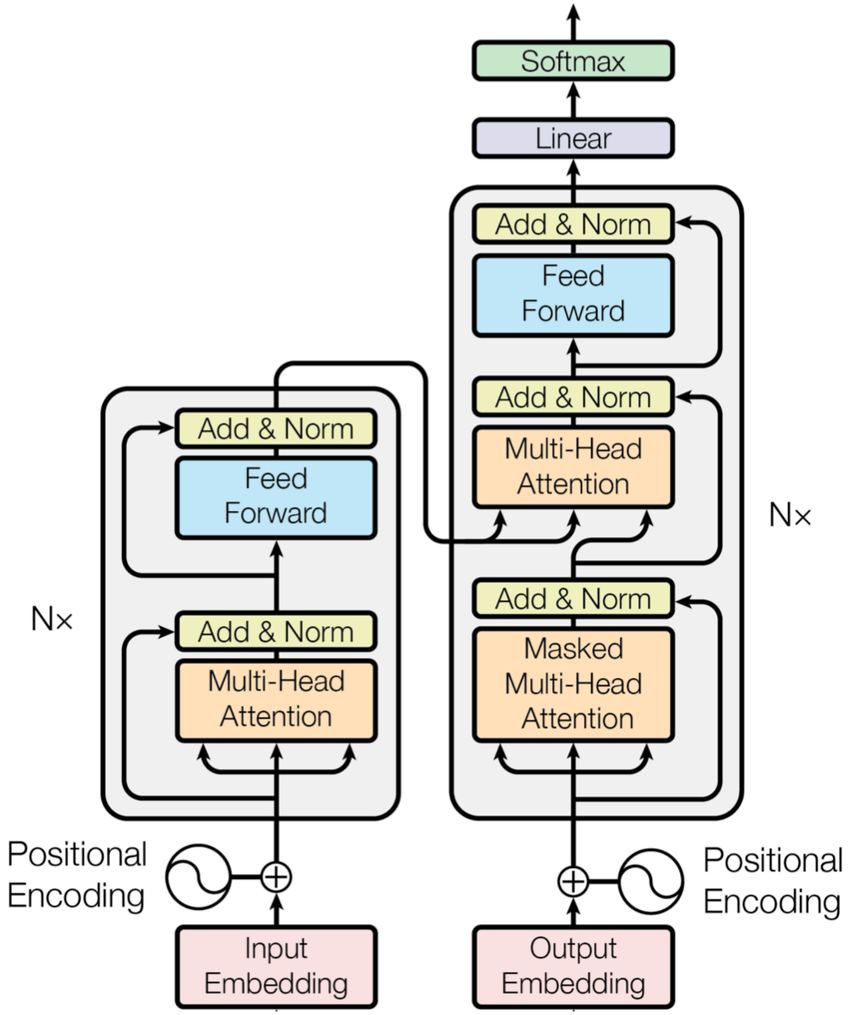




- No dependency between tokens, leading to **better scalability**.
- **Limited context information**, resulting in worse modeling capability.

Image credit: https://cezannec.github.io/CNN_Text_Classification/

Transformer



- Each encoder block has two sub-layers:
  - The first is a **multi-head self-attention** mechanism.
  - The second is a position-wise **fully connected feed-forward** network.
- Each decoder block has an additional third sub-layer:
  - The third is a multi-head attention over the output of the encoder stack.
- A residual connection is added around each of the two sub-layers, followed by layer normalization:

$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$

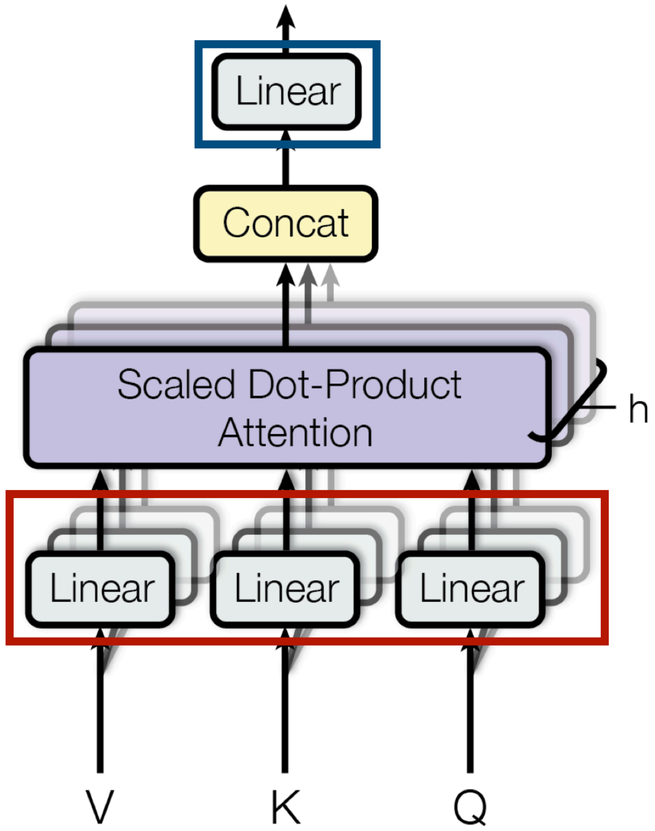
- The decoder generates the output sequence of symbols one element at a time in an **auto-regressive** manner.

Attention Is All You Need [Vaswani *et al.*, 2017]

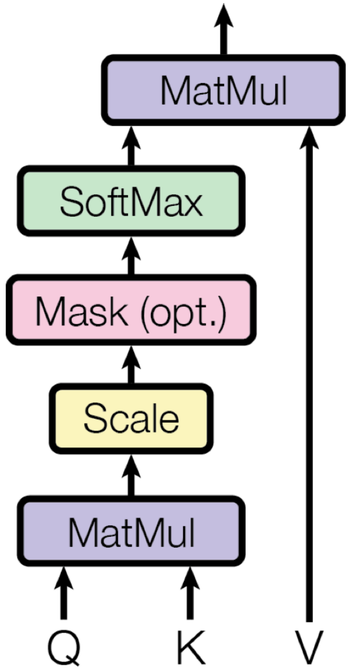
Multi-Head Self-Attention (MHSA)

- **Project** Q, K and V with h **different**, learned linear projections.
- Perform the **scaled dot-product attention** function on each of these projected versions of Q, K and V **in parallel**.
- **Concatenate** the output values.
- **Project** the output values again, resulting in the final values.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Scaled Dot-Product Attention



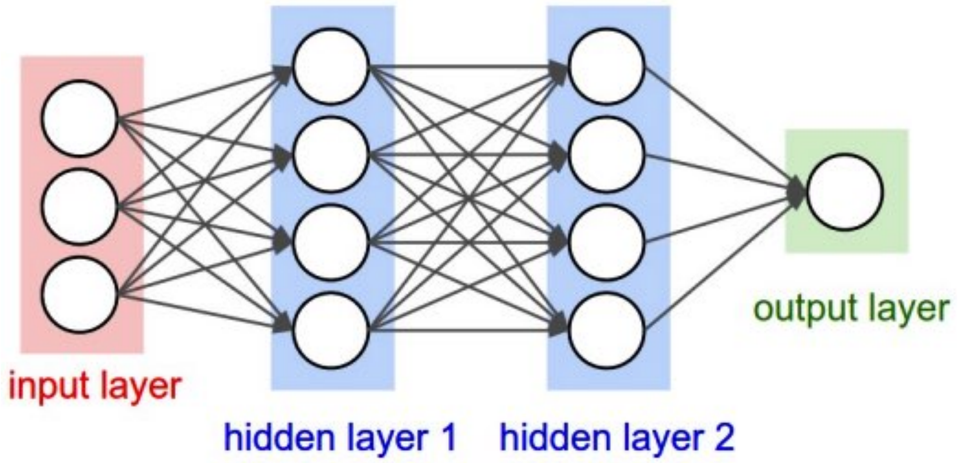
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Feed-Forward Network (FFN)

- Each block in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position **separately** and **identically**.
- This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- The middle hidden size is usually larger than and input and output size (**inverted bottleneck**).



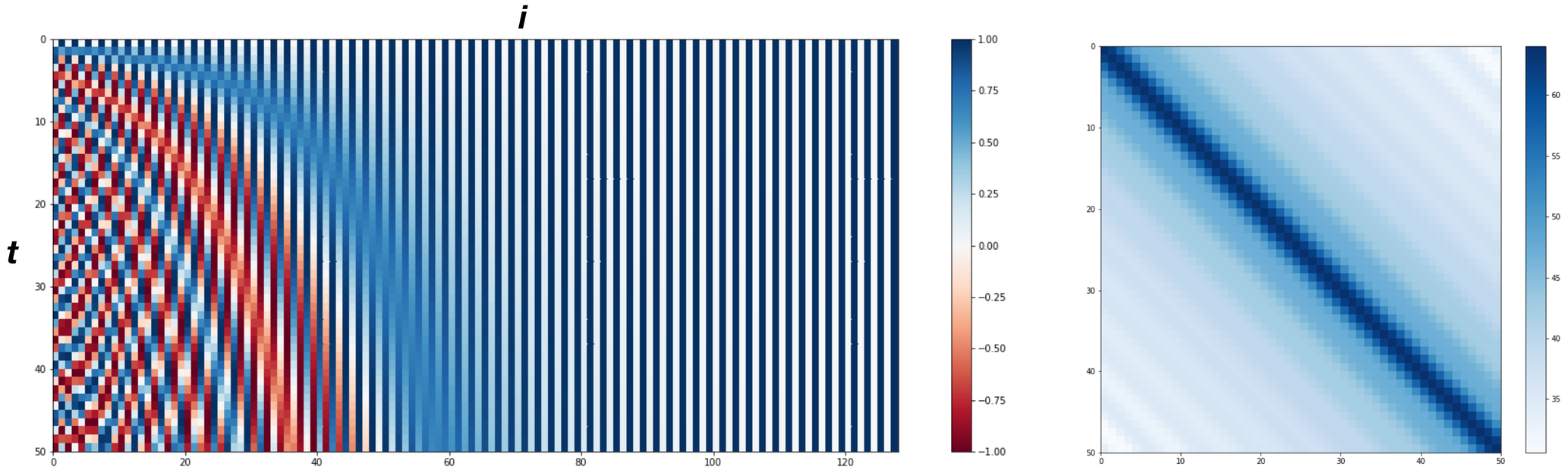
Model	#L	#H	d_{model}	LR	Batch
125M	12	12	768	$6.0e{-4}$	0.5M
350M	24	16	1024	$3.0e{-4}$	0.5M
1.3B	24	32	2048	$2.0e{-4}$	1M
2.7B	32	32	2560	$1.6e{-4}$	1M
6.7B	32	32	4096	$1.2e{-4}$	2M
13B	40	40	5120	$1.0e{-4}$	4M
30B	48	56	7168	$1.0e{-4}$	4M
66B	64	72	9216	$0.8e{-4}$	2M
175B	96	96	12288	$1.2e{-4}$	2M

Image credit: https://ds055uzetaobb.cloudfront.net/brioche/uploads/uzLXsnBLTI-fully_connected_mlp.png?width=4000

Position Embedding (PE)

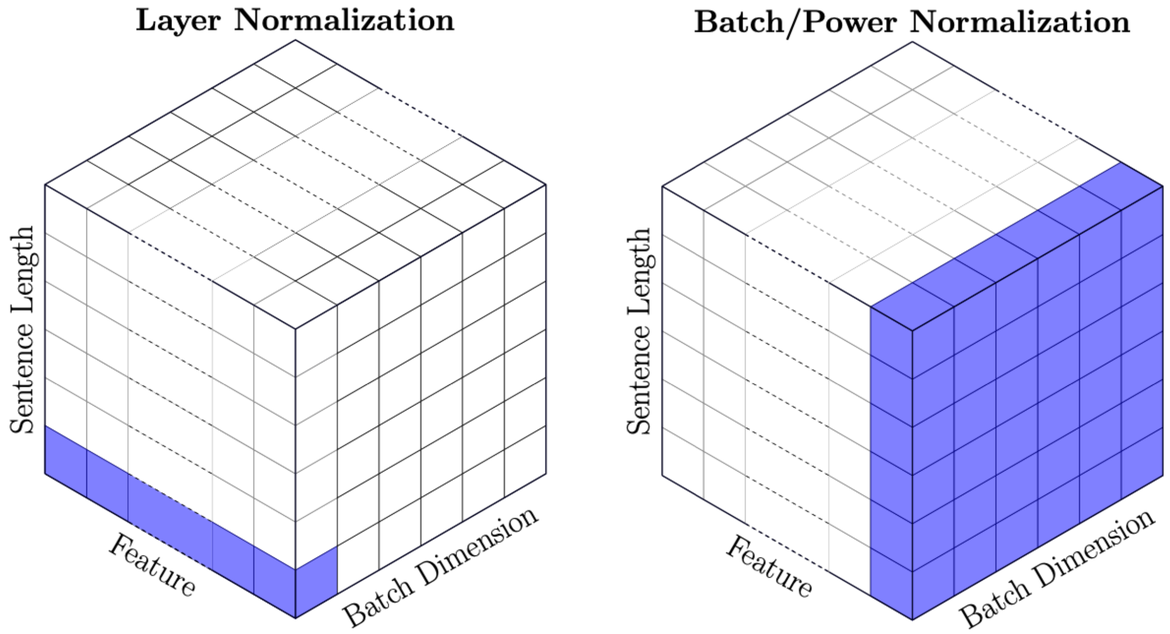
- Positional embedding: Information to each word about its position in the sentence.
  - **Unique** encoding for each word's position in a sentence.
  - Distance between any two positions is **consistent** across sentences with different lengths.
  - **Deterministic** and **generalize** to longer sentences.

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases} \qquad \omega_k = \frac{1}{10000^{2k/d}}$$



Content credit: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Layer Normalization (LN)



- The statistics of NLP data across the **batch** dimension exhibit **large fluctuations** during training.
- Batch normalization (BN) is more efficient as it can be fused into the linear layer.
- There are some efforts towards using BN in transformers.
  - PowerNorm: Rethinking Batch Normalization in Transformers [Shen *et al.*, 2020]
  - Leveraging Batch Normalization for Vision Transformers [Yao *et al.*, 2021]

Image credit: http://proceedings.mlr.press/v119/shen20e/shen20e.pdf

Results

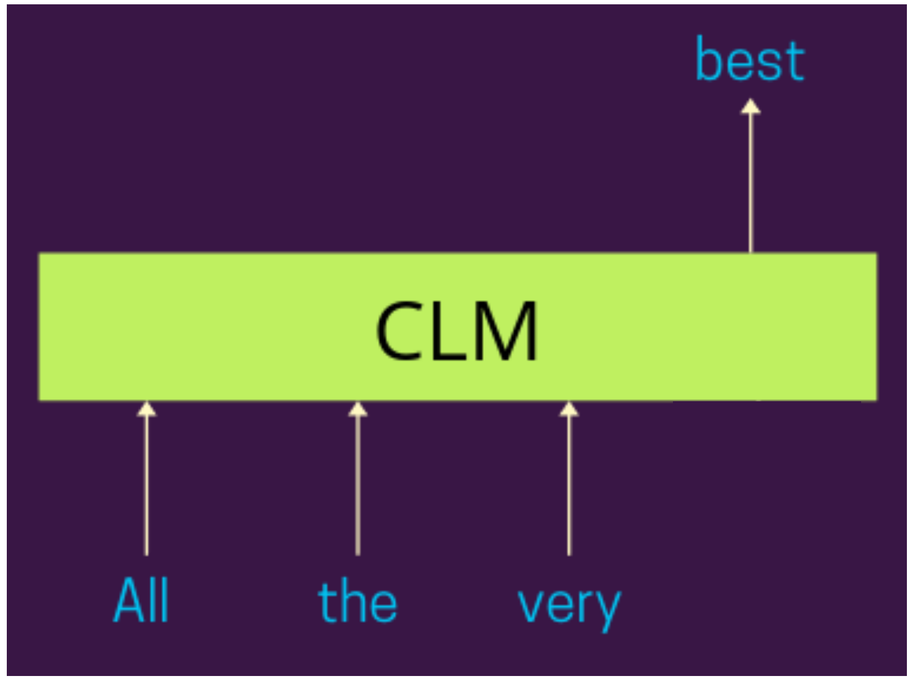
Neural Machine Translation (NMT)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- Transformer surpasses all previously published models and ensembles, at a fraction of the training cost of any of the competitive models.
- BLEU measures the similarity of the machine-translated text to a set of reference translations.

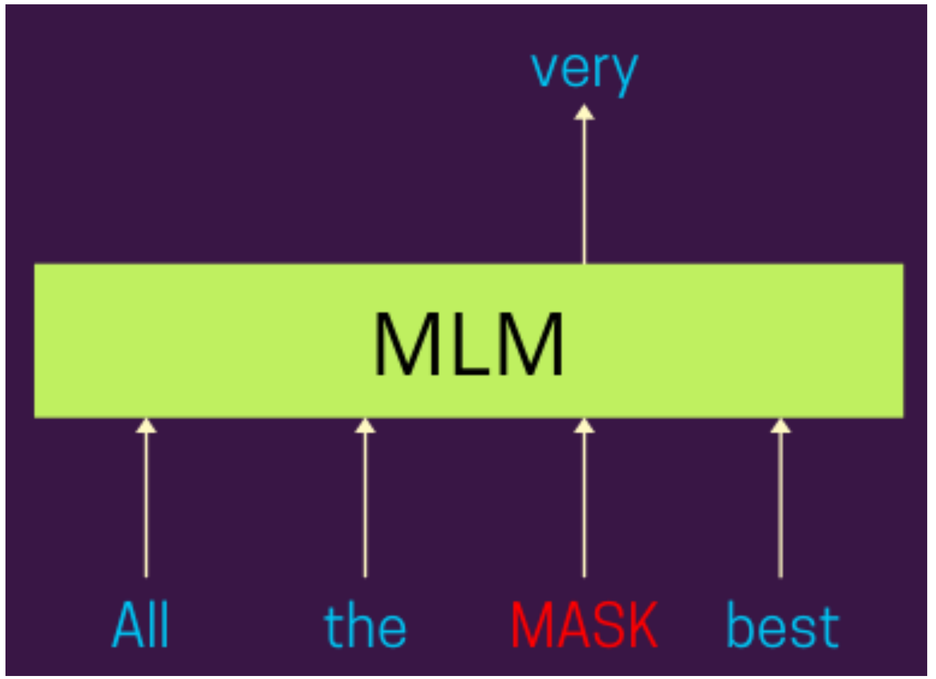
Language Models

- General model of language understanding (trained on large amount of **unlabeled text**) before being fine-tuned on specific NLP tasks (such as machine translation, text summarization, etc.)



Causal Language Models (CLM)

(Unidirectional, from left to right)



Masked Language Models (MLM)

(Bidirectional, from both left and right)

Content credit: https://towardsdatascience.com/understanding-masked-language-models-mlm-and-causal-language-models-clm-in-nlp-194c15f56a5

Casual Language Models (CLM) — GPT

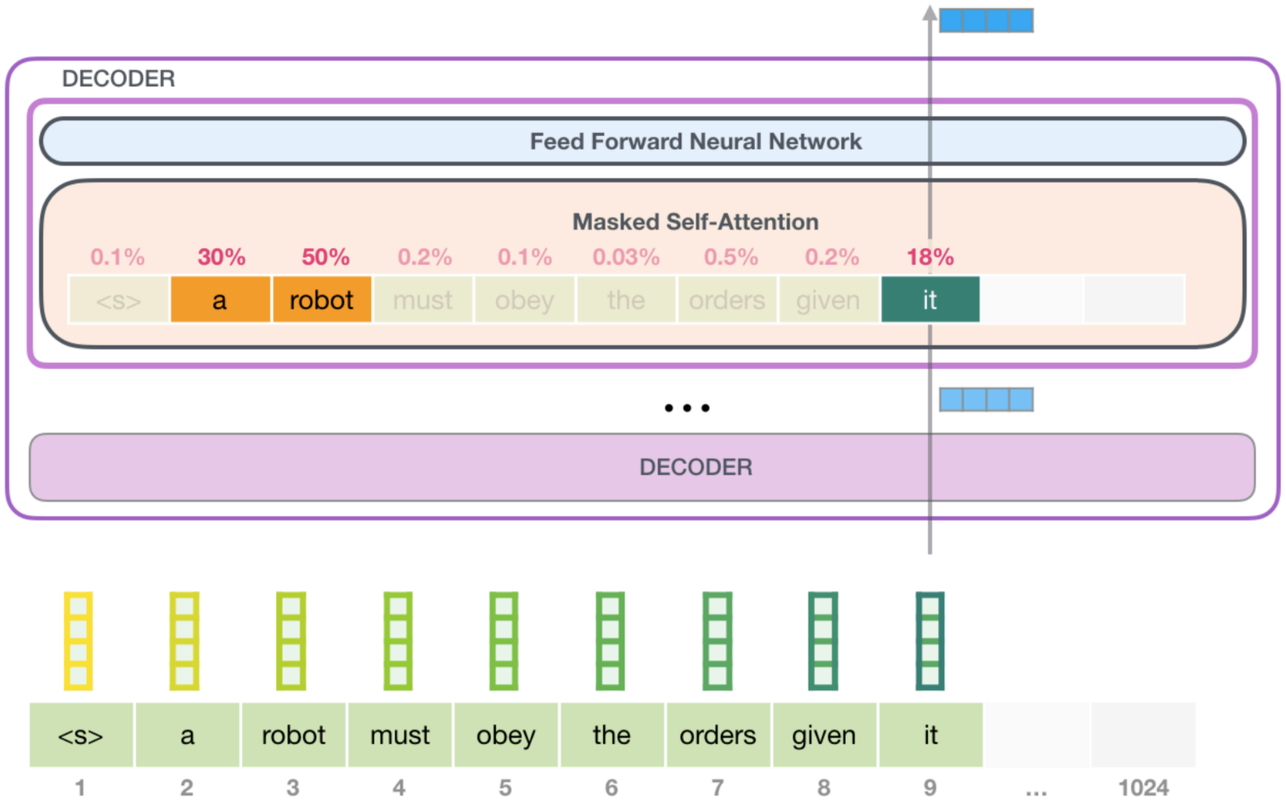


Image credit: https://jalammar.github.io/illustrated-gpt2/

- System works in two stages:
  - First, **pre-train** a transformer model on a very large amount of data in an unsupervised manner (language modeling):

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

  - Then, **fine-tune** this model on much smaller supervised datasets to solve specific tasks.

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y | x^1, \ldots, x^m)$$

Improving Language Understanding by Generative Pre-Training [Radford *et al.*, 2018]

Masked Language Models (MLM) — BERT

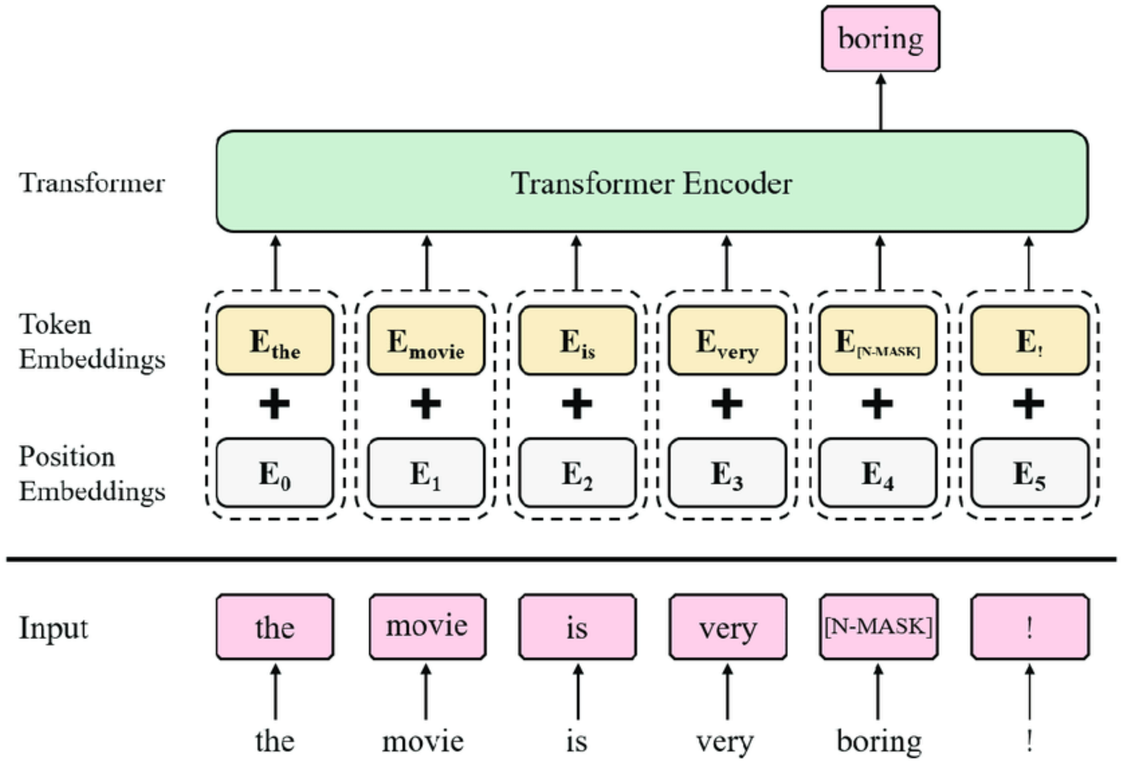


Image credit: https://www.mdpi.com/2073-8994/11/11/1393/htm

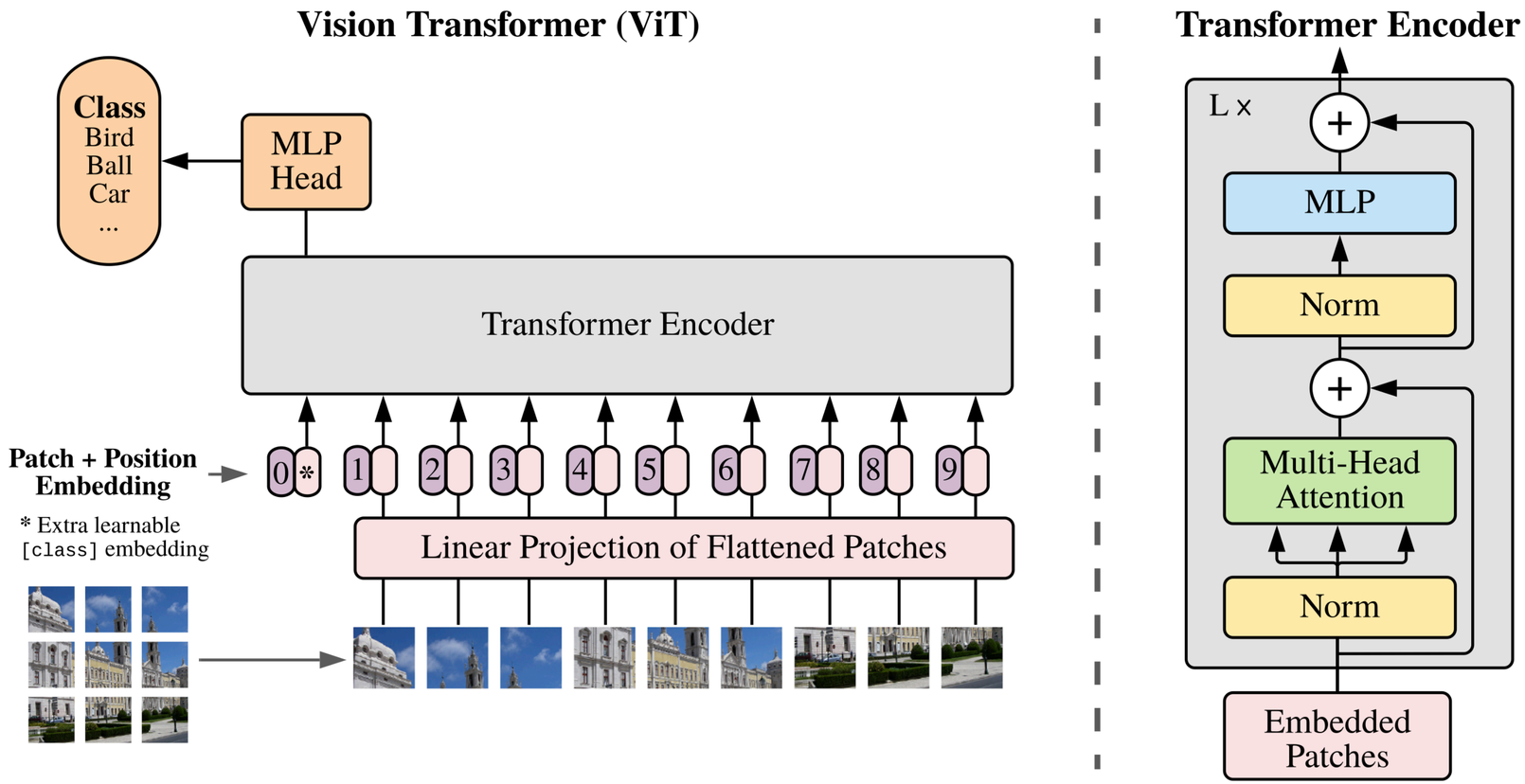
- Task #1: **Masked Language Model (MLM)**
  - Mask some percentage (15%) of the input tokens at random.
  - Predict those masked tokens.

- Task #2: **Next Sentence Prediction (NSP)**
  - Binarized next sentence prediction task.
  - When choosing the sentences A and B for each pre-training example,
    - 50% of the time B is the actual next sentence that follows A (**IsNext**).
    - 50% of the time it is a random sentence from the corpus (**NotNext**).

BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding [Devlin *et al.*, 2018]

II. Applications

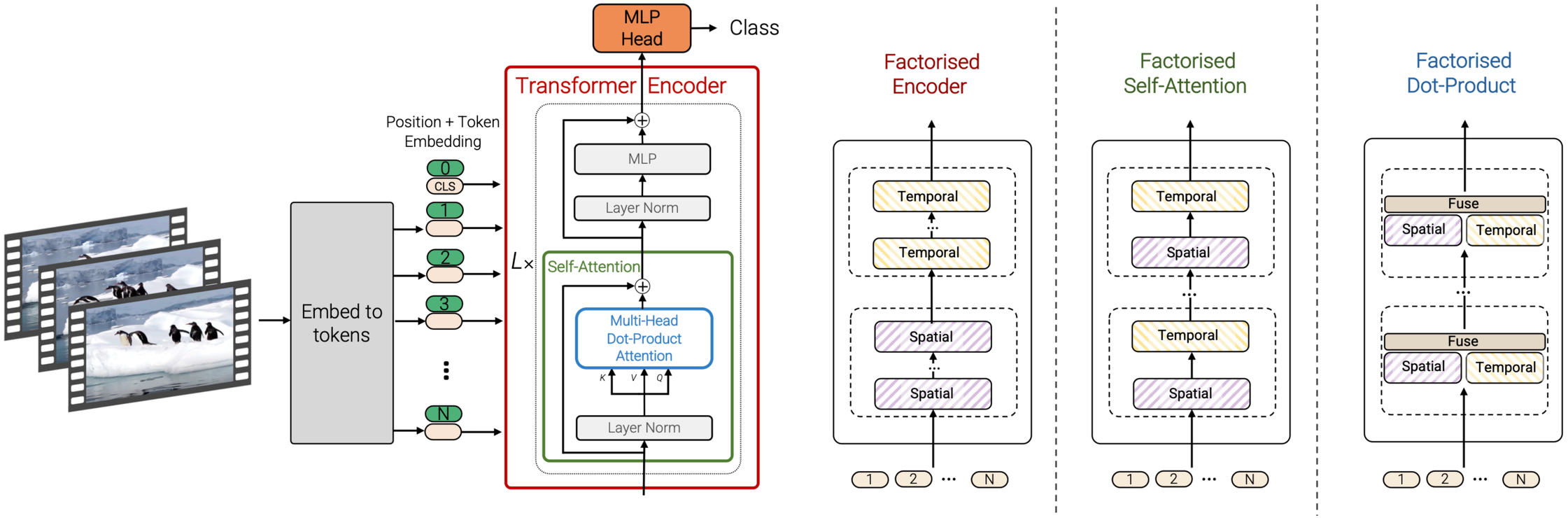
Image Transformer — ViT



- Split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder.

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale [Dosovitskiy *et al.*, 2020]

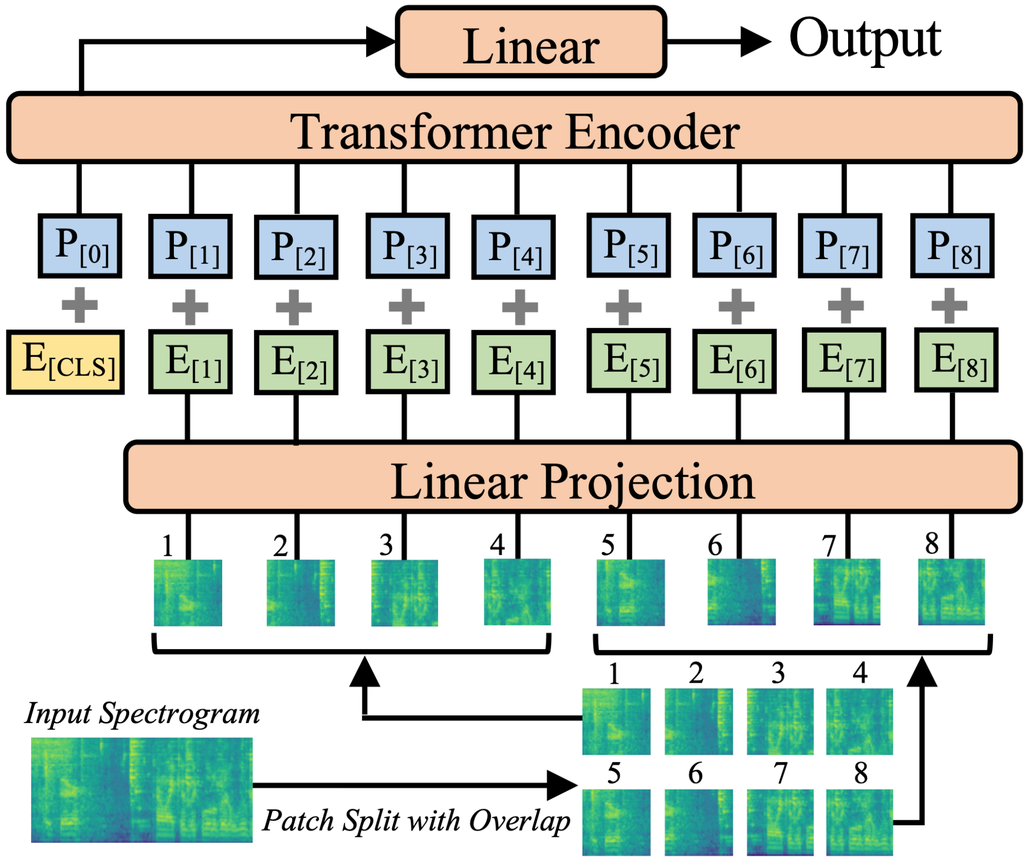
Video Transformer — ViViT



- Extract spatiotemporal tokens from the video, then encoded by a series of transformer layers.
- Factorize the spatial- and temporal-dimensions of the input to handle the long sequences.

ViViT: A Video Vision Transformer [Arnab *et al.*, 2021]

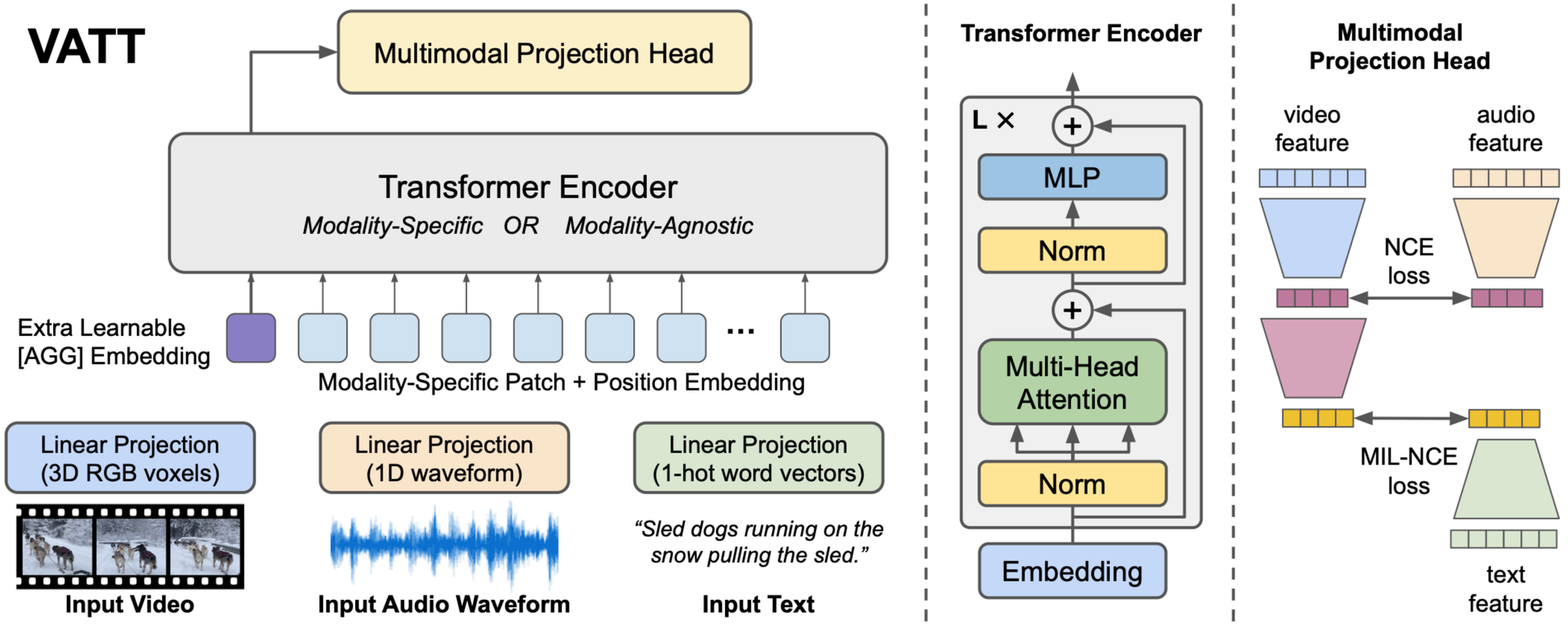
Audio Transformer — AST



- Split the 2D audio spectrogram a sequence of 16×16 patches with overlap, and then linearly projected to a sequence of 1-D patch embeddings.

AST: Audio Spectrogram Transformer [Gong *et al.*, 2021]

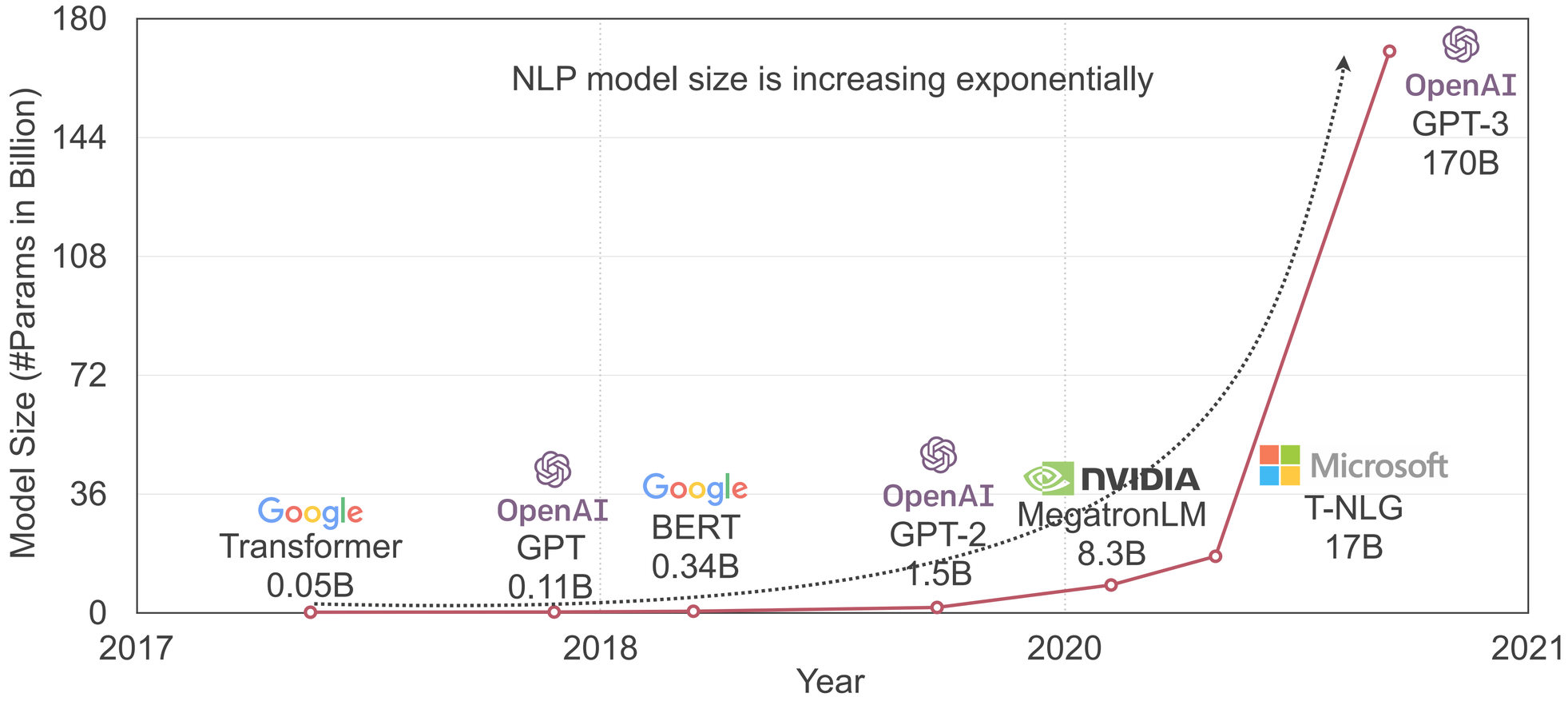
Multi-Modal Transformer — VATT



- Linearly project each modality into a feature vector and feed it into a Transformer encoder.

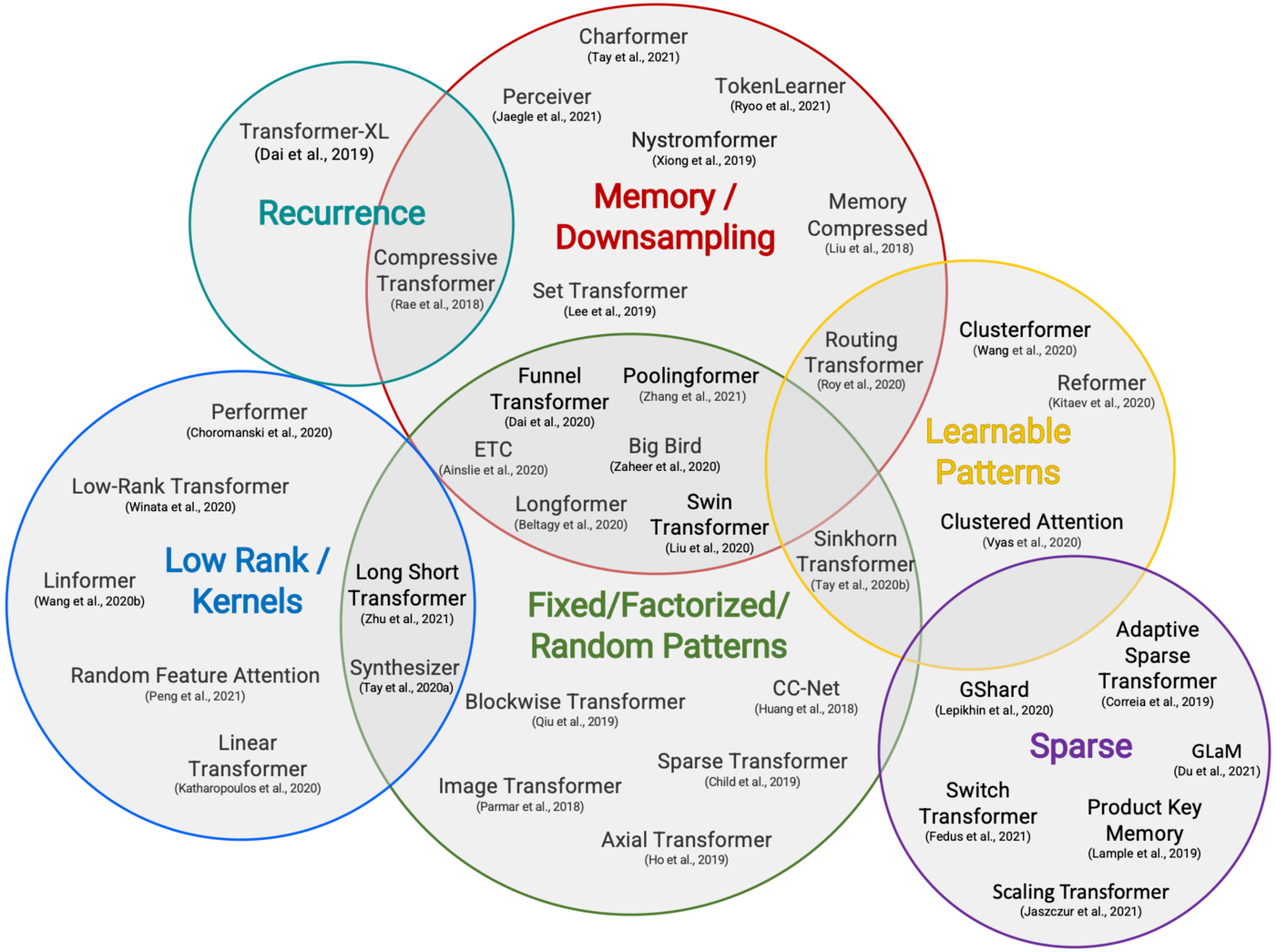
VATT: Transformers for Multimodal Self-Supervised Learning from Raw Video, Audio and Text [Akbari *et al.*, 2021]

Transformers Are Big!



III. Efficient Transformers

Efficient Transformers



Efficient Transformers: A Survey [Tay *et al.*, 2020]

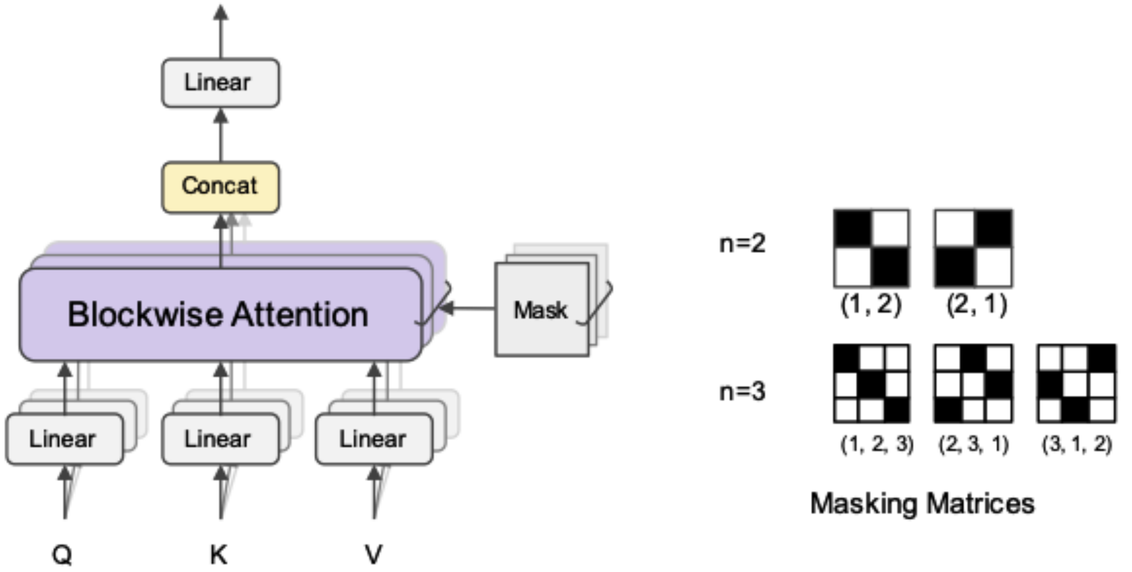
Sparse Attention — BlockBERT

Blockwise Attention

- Key idea: **Chunk input into fixed blocks**.
  - Different heads use different masking matrices.
  - Masking matrices can be any permutation.

- The complexity is reduced from NxN to NxB.
  - B is the block size (e.g., N/2, N/3).

- Key Results:
  - Memory: **18-36%** less
  - Training time: **12-25%** less
  - Inference time: **28%** less

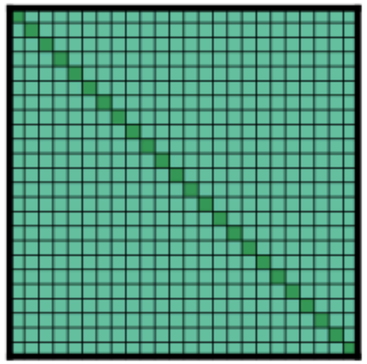


N	Model	Training Time (day)	Memory (per GPU, GB)	Heads Config.	Valid. ppl
512	RoBERTa-1seq	6.62	9.73	-	3.58
	BlockBERT n=2	5.83 (-12.0%)	7.91 (-18.7%)	10:2	3.56
	BlockBERT n=3	5.80 (-12.5%)	7.32 (-23.8%)	8:2:2	3.71
1024	RoBERTa-1seq	9.66	13.39	-	3.60
	BlockBERT n=2	7.51 (-22.3%)	9.73 (-27.3%)	9:3	3.57
	BlockBERT n=3	7.23 (-25.1%)	8.55 (-36.1%)	8:2:2	3.63

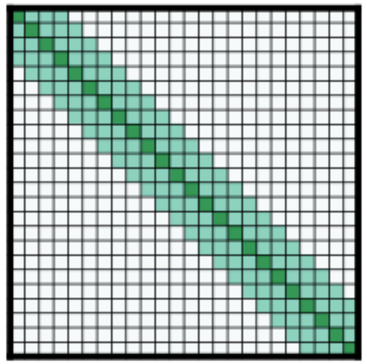
Blockwise Self-Attention for Long Document Understanding [Qiu *et al.*, 2019]

Sparse Attention — LongFormer

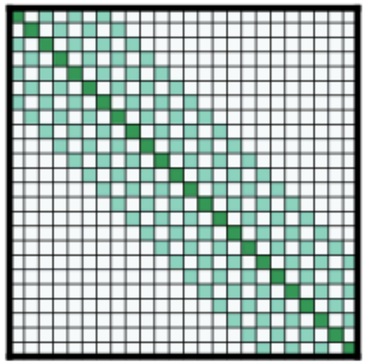
Local Attention + Global Attention



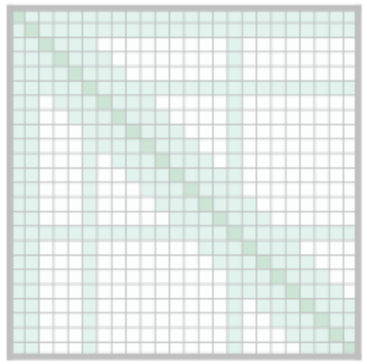
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window



(d) Global+sliding window

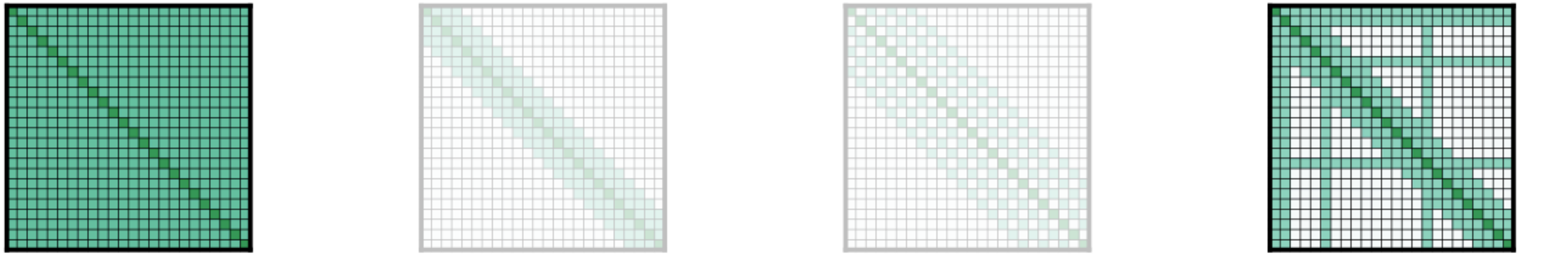
- Attention with **sliding window** (analogous to CNNs):
  - A fixed-size window attention surrounding each token.
  - The complexity is reduced from $O(N^2)$ to **O(N x W)**, where W is the window size.

- Attention with **dilated sliding window** (analogous to dilated CNNs):
  - Dilate the sliding window with gaps of size dilation D.
  - The receptive field is enlarged from W to **W x D**, with the same complexity.

Longformer: The Long-Document Transformer [Beltagy *et al.*, 2020]

Sparse Attention — LongFormer

Local Attention + Global Attention



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

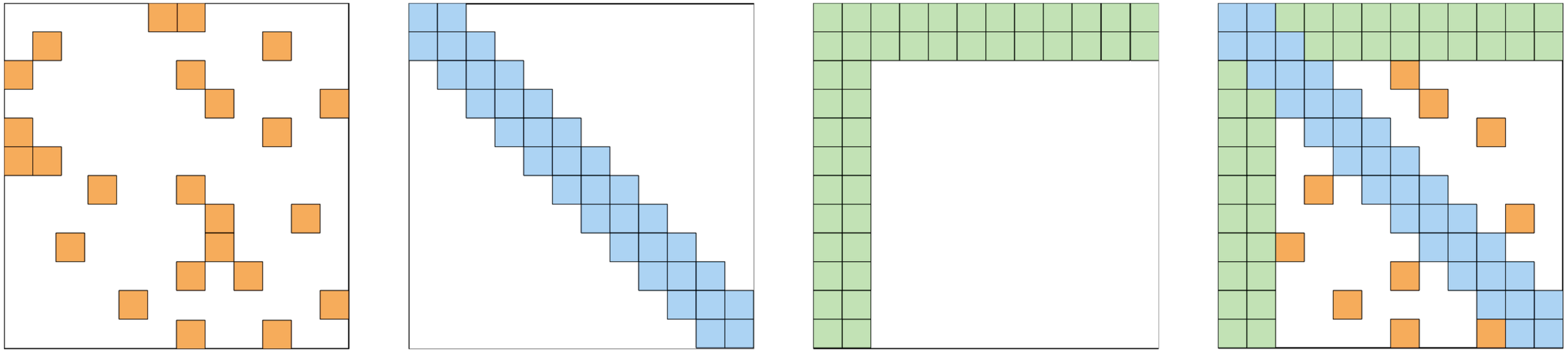
- **Global attention** added on a few **pre-selected** input locations:
  - Classification: The special token ([CLS]), aggregating the whole sequence.
  - QA: All question tokens, allowing the model to compare the question with the document.

- Global attention is applied **symmetrically**:
  - A token with a global attention attends to all tokens across the sequence, and all tokens in the sequence attend to it.

Longformer: The Long-Document Transformer [Beltagy *et al.*, 2020]

Sparse Attention — Big Bird

Random Attention + Local Attention + Global Attention



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

- Random sparse attention:
  - Each query attends over r random number of keys: *i.e.* A(i, ·) = 1 for r randomly chosen keys.
  - Information can flow fast between any pair of nodes (rapid mixing time for random walks).

Big Bird: Transformers for Longer Sequences [Zaheer *et al.*, 2021]

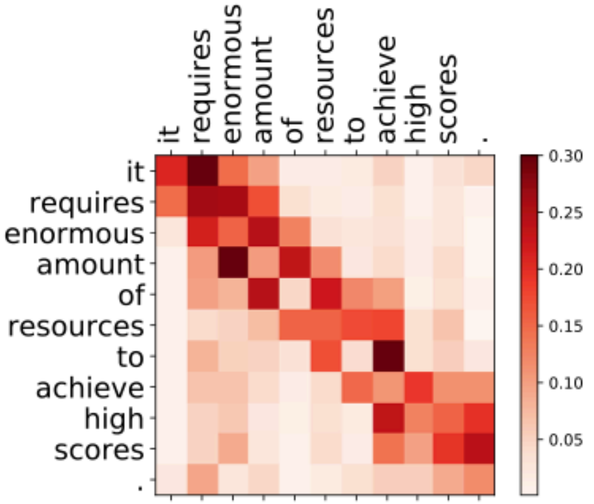
Sparse Attention — Lite Transformer

Local Convolution + Global Attention

- Long-Short Range Attention (LSRA):
  - **Convolution**: Efficiently extract the **local** features.
  - **Attention**: Tailored for **global** feature extraction.

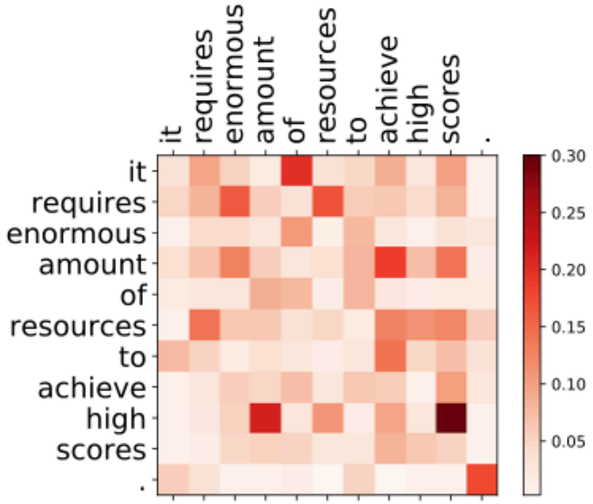
Original Attention

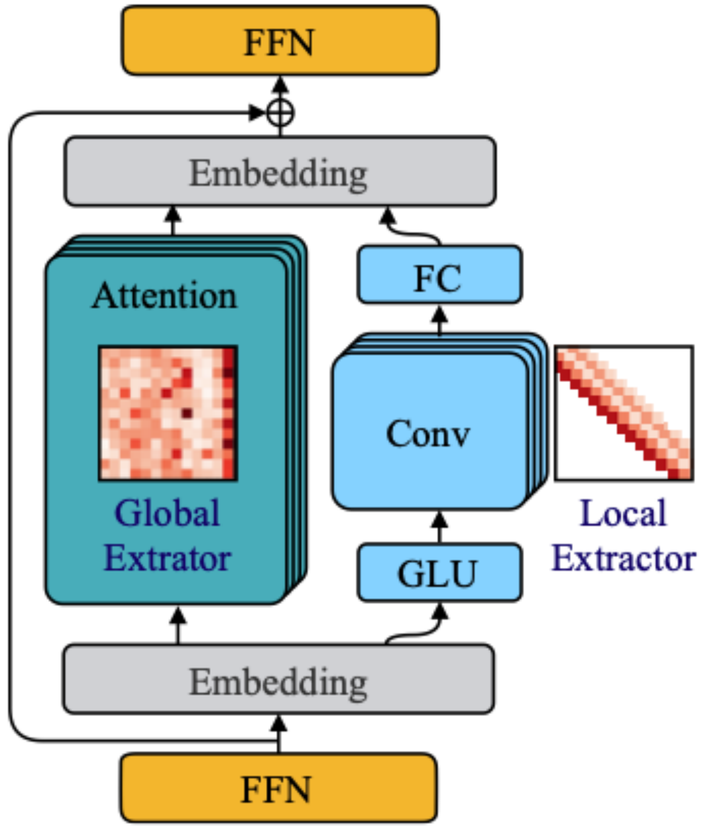
(Too much emphasize on local feature extraction)



Attention in LSRA

(Dedicated for global feature extraction)





Lite Transformer with Long-Short Range Attention [Wu *et al.*, 2020]

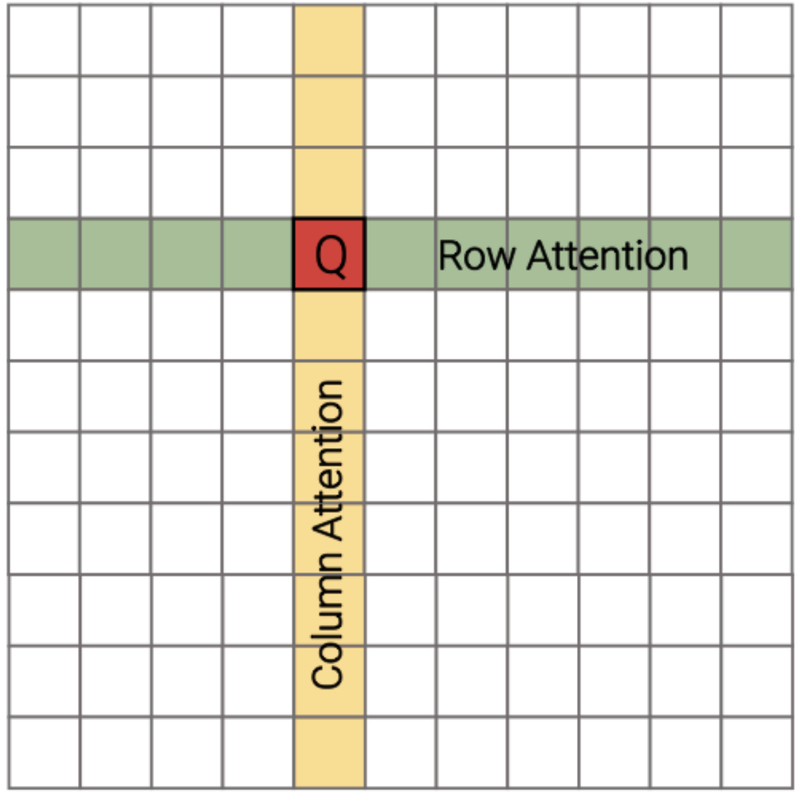
Sparse Attention — Axial Transformer

Row Attention + Column Attention (for 2D Image Recognition)

- The $O(N^2)$ complexity is prohibitive for 2D images:
  - N = H W, where H and W are height and width.

- Key idea: **Factorize attention along different axes**.
  - Each attention mixes information along a particular axis, while keeping information along other axes **independent**.
  - Analogous to spatially separable convolution:
    - Decompose a 3x3 conv into 1x3 and 3x1 convs.

- The complexity is reduced from $O(H^2W^2)$ to $\mathbf{O(H^2 + W^2)}$.



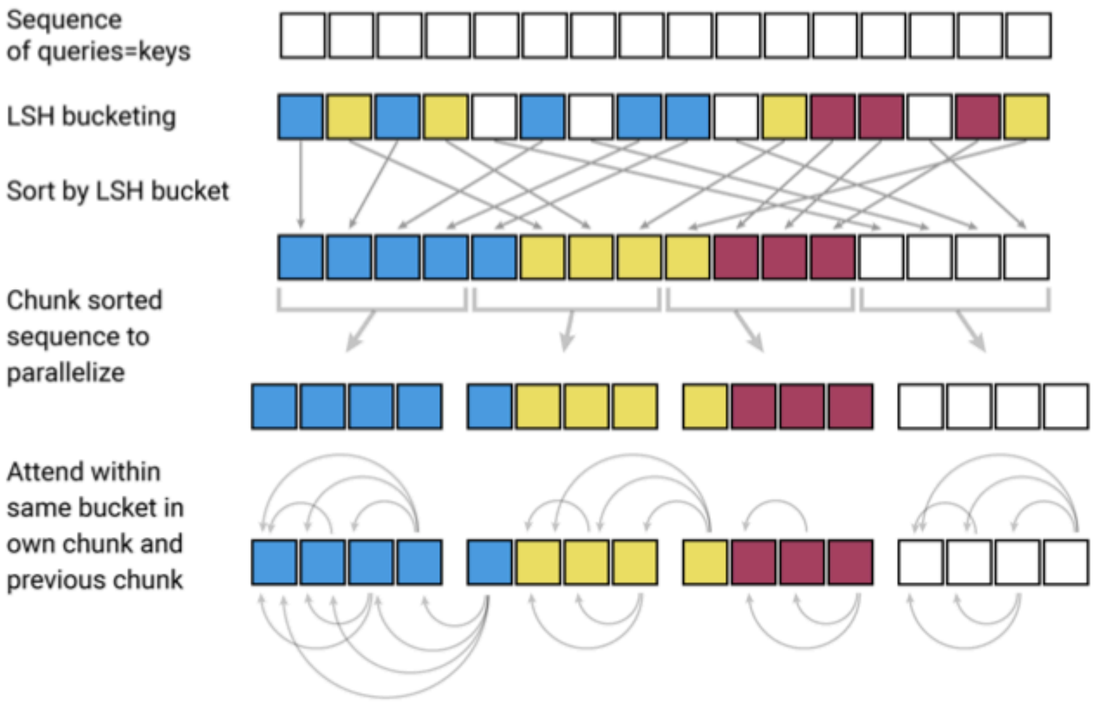
2D Feature Map

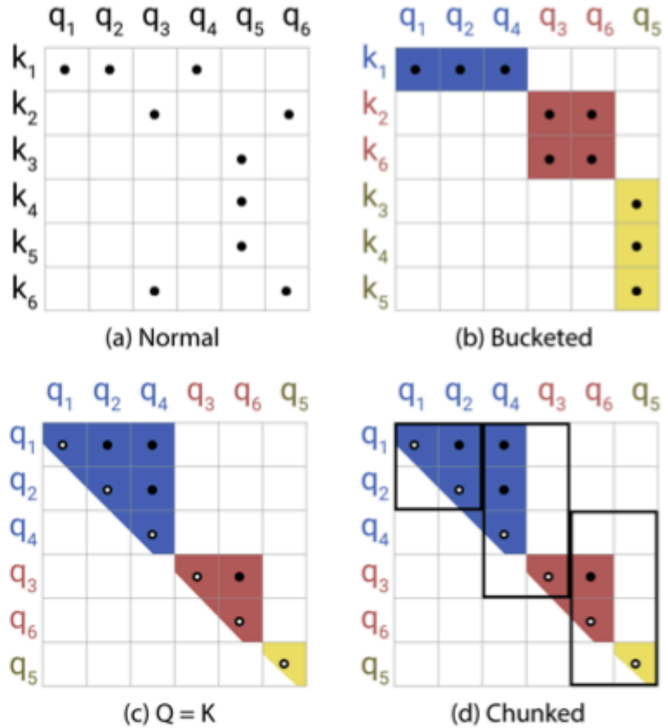
(This is not the attention matrix!)

Axial Attention in Multidimensional Transformers [Ho *et al.*, 2019]

Sparse Attention — Reformer

Learned Bucket-wise Attention



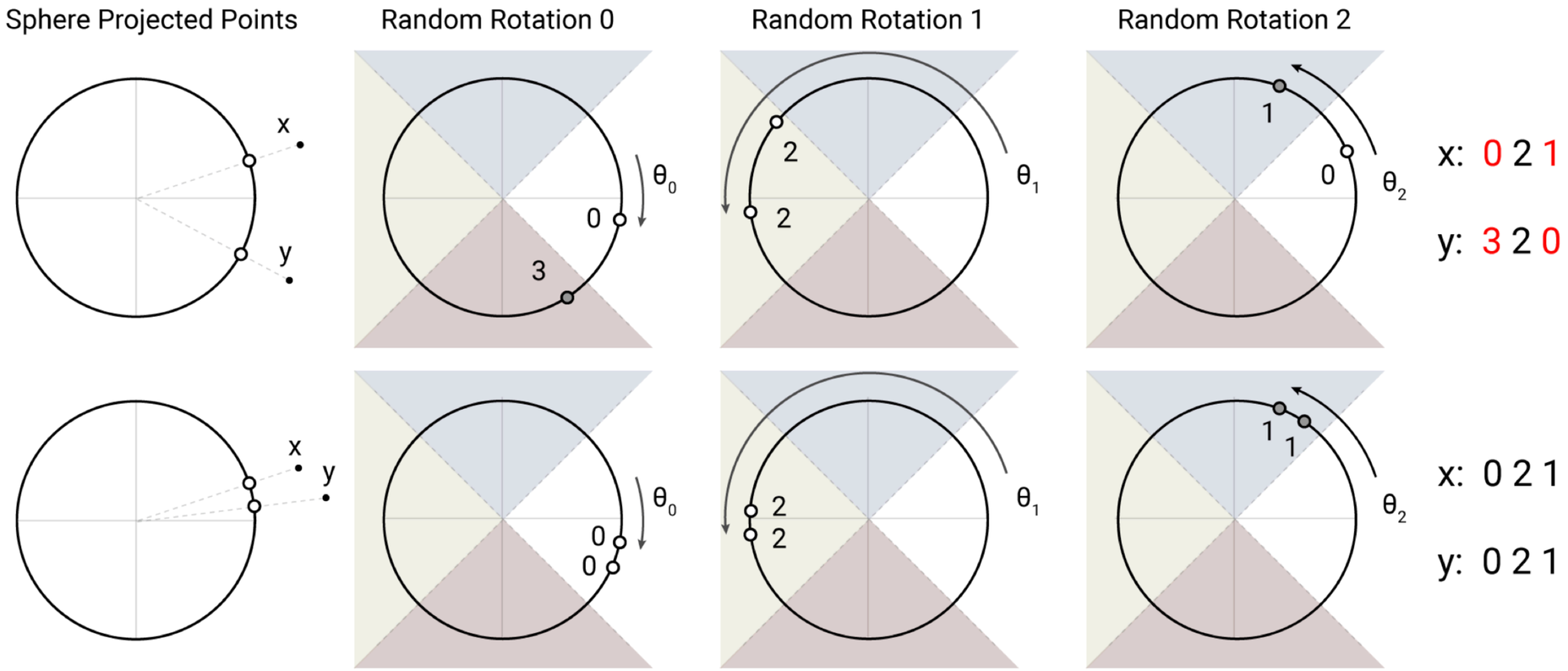


- Assumption: **Q** (queries) = **K** (keys)
- Intuition: Attention weight Q^TK is **large** only if Q_i and K_j are **similar**.
- Key idea: **Group similar vectors into the same bucket** (using locality-sensitive hashing).

Reformer: The Efficient Transformer [Kitaev *et al.*, 2020]

Sparse Attention — Reformer

Locality-Sensitive Hashing (LSH)



- Use **random rotations** of spherically projected points to establish buckets by an argmax over signed axes projections.

$$h(x) = \arg\max([xR; -xR])$$

Reformer: The Efficient Transformer [Kitaev *et al.*, 2020]

Low-Rank Approximations

- Improve efficiency by leveraging **low-rank** approximations of the self-attention matrix.
- The key idea is to assume **low-rank structure** in the N×N matrix.

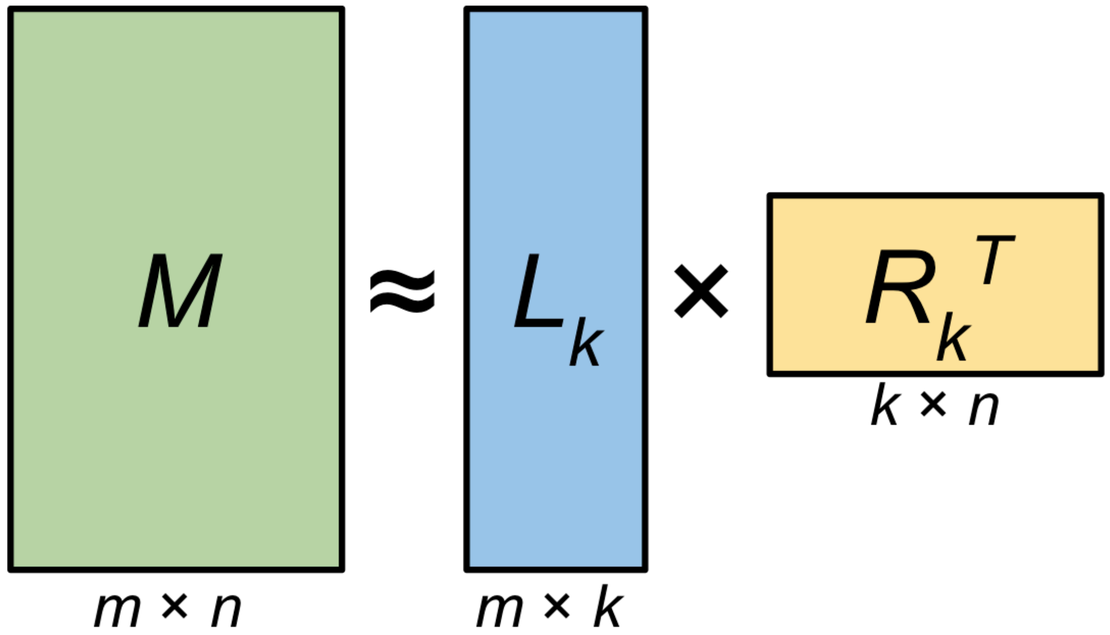
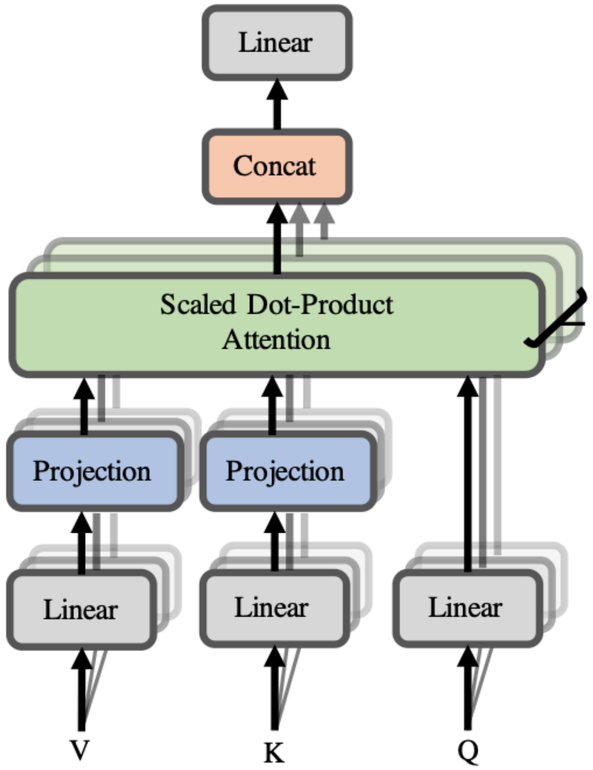
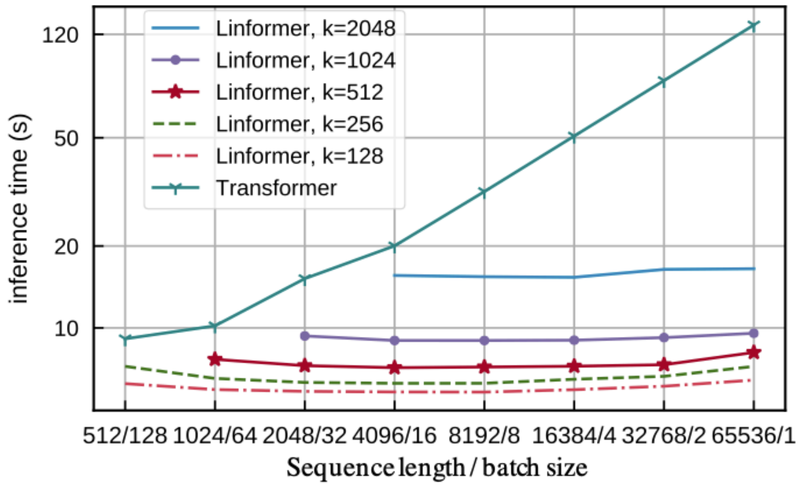


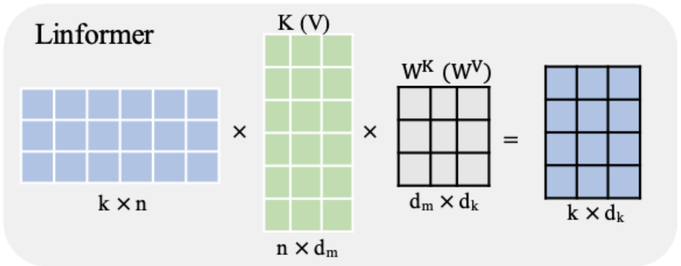
Image credit: https://dustinstansbury.github.io/theclevermachine/assets/images/svd-data-compression/low-rank-approximation.png

Low-Rank Approximations — Linformer

Approximate Self-Attention with Low-Rank Matrix







$$\overline{\text{head}_i} = \text{Attention}(QW_i^Q, \boxed{E_i}KW_i^K, \boxed{F_i}VW_i^V)$$

$$= \underbrace{\text{softmax}\left(\frac{QW_i^Q(E_iKW_i^K)^T}{\sqrt{d_k}}\right)}_{\bar{P}:n\times k} \cdot \underbrace{F_iVW_i^V}_{k\times d},$$

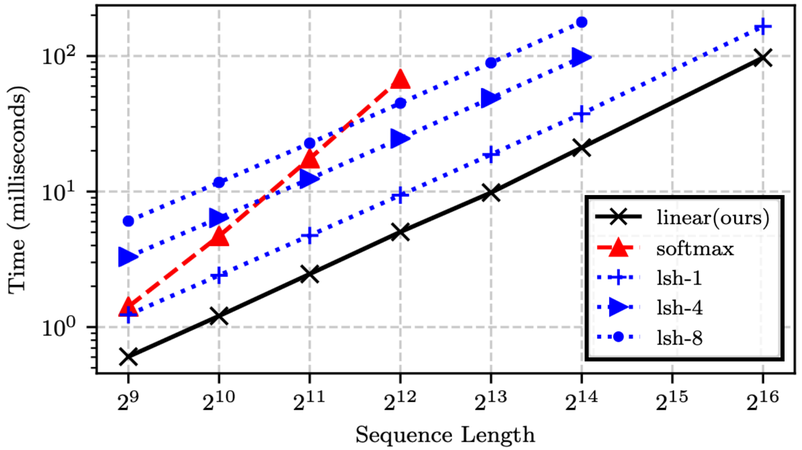
- It projects the **length dimension** (**not the feature dimension**) of keys and values to a lower-dimensional representation (from N to k).
- Low-rank method reduces the memory complexity problem of self-attention (from N N to N×k).

Linformer: Self-Attention with Linear Complexity [Wang *et al.*, 2020]

Kernelization — Linear Transformer

- Generalized attention can be formulated as

$$O_i = \sum_{j=1}^{N} \frac{Sim(Q_i, K_j)}{\sum_{j=1}^{N} Sim(Q_i, K_j)} V_j, \quad \text{where } Q = xW_Q, K = xW_K, V = xW_V$$



- Softmax attention is a special case with $Sim(Q, K) = \exp(\frac{QK^T}{\sqrt{d}})$

- Linearized attention is defined with $Sim(Q, K) = \phi(Q)\phi(K)^T$

$$O_i = \sum_{j=1}^{N} \frac{\phi(Q_i)\phi(K_j)^T}{\sum_{j=1}^{N} \phi(Q_i)\phi(K_j)^T} V_j = \frac{\sum_{j=1}^{N} (\phi(Q_i)\phi(K_j)^T) V_j}{\phi(Q_i) \sum_{j=1}^{N} \phi(K_j)^T} = \frac{\phi(Q_i)(\sum_{j=1}^{N} \phi(K_j)^T V_j)}{\phi(Q_i)(\sum_{j=1}^{N} \phi(K_j)^T)}$$

1 x D, D x D, 1 x D, D x 1

Shared across different O_i

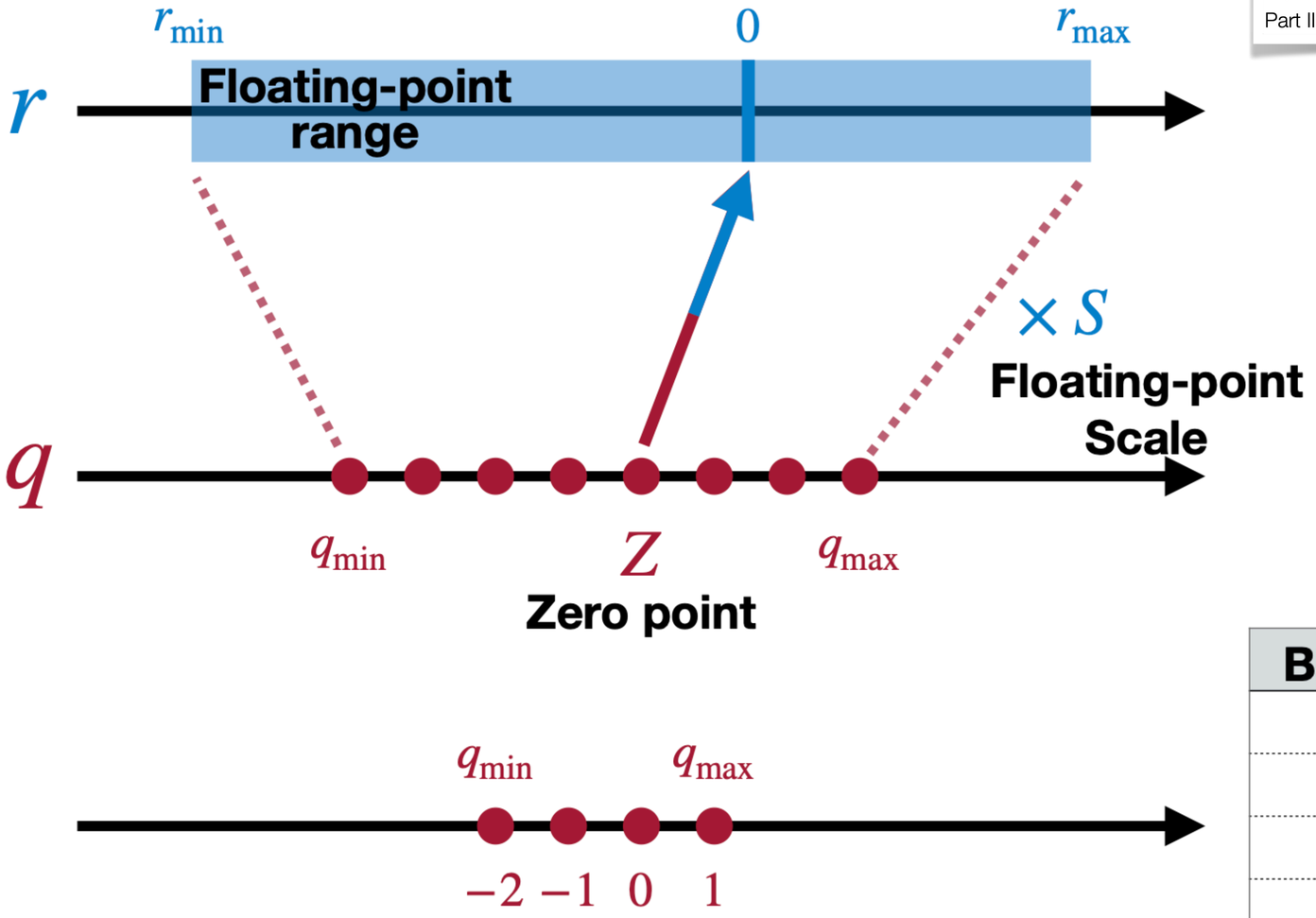
- Complexity is reduced from **$O(N^2D)$** to **$O(ND^2)$**.

Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention [Katharopoulos *et al.*, 2020]

Quantization

Lecture 05
Quantization
Part I

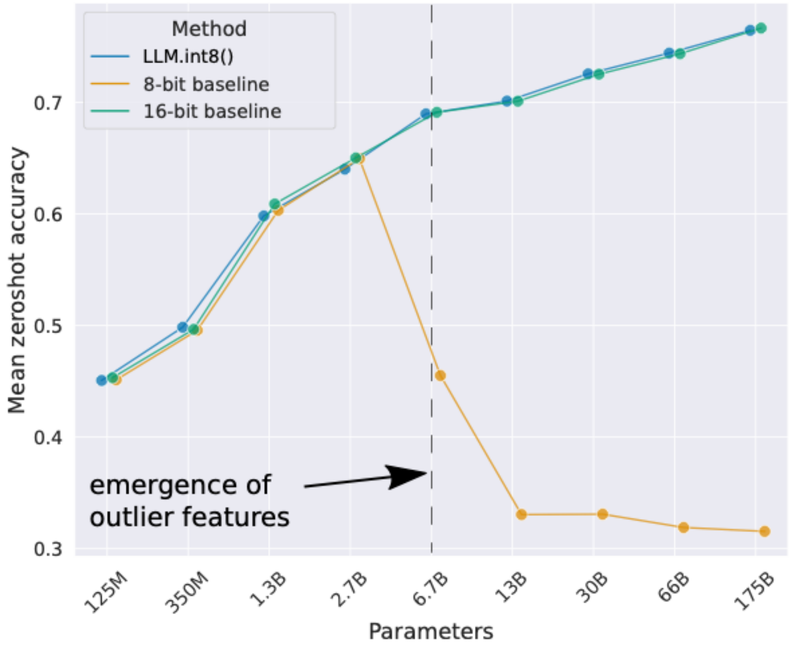
Lecture 06
Quantization
Part II

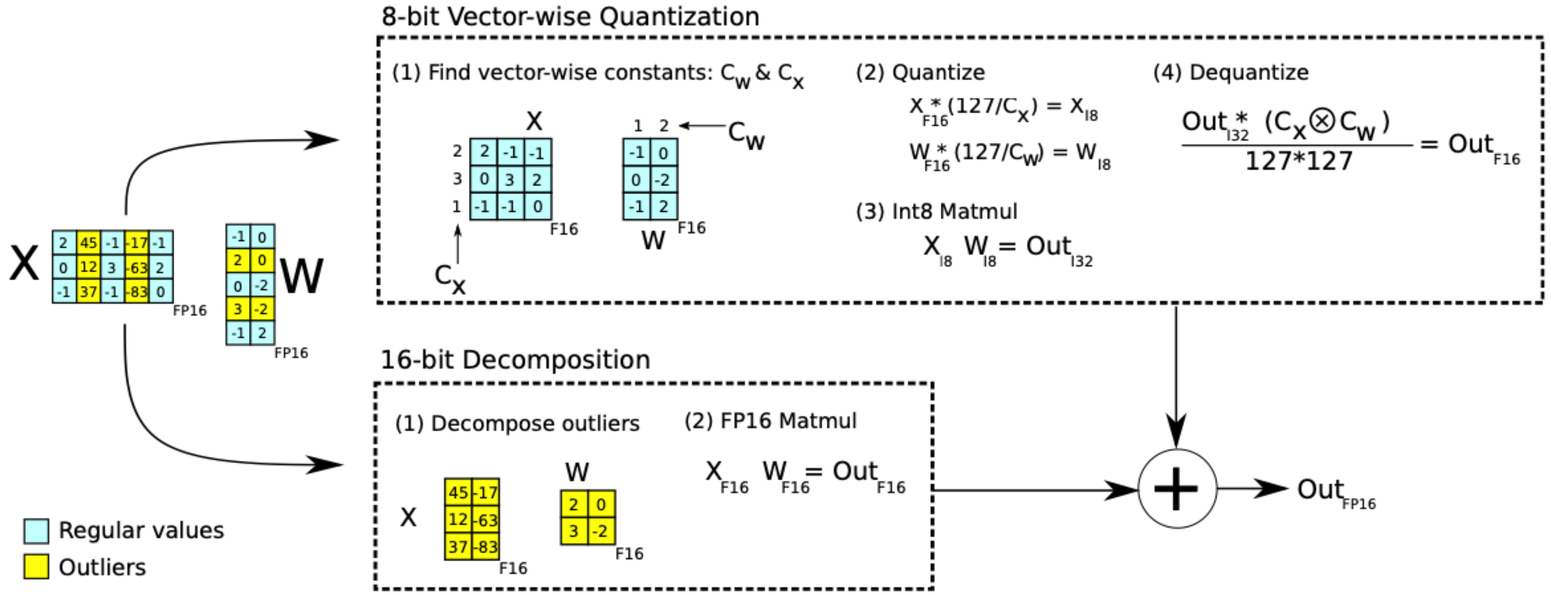


Binary	Decimal
01	1
00	0
11	-1
10	-2

Quantization — LLM.int8()

Mixed-Precision Decomposition





- **Motivation**: Transformers have outlier features that have **large values** (especially large models).
  - They occur in particular hidden dimensions, leading to large quantization error.
- **Key idea**: Separate outlier features into a **separate FP16 MM**, quantize the other values to Int8.
  - Outlier: At least one feature dimension with a magnitude larger than the threshold (6).
  - Token-wise scale factor (for X) and (output) channel-wise scale factor (for W).

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale [Dettmers *et al.*, 2022]

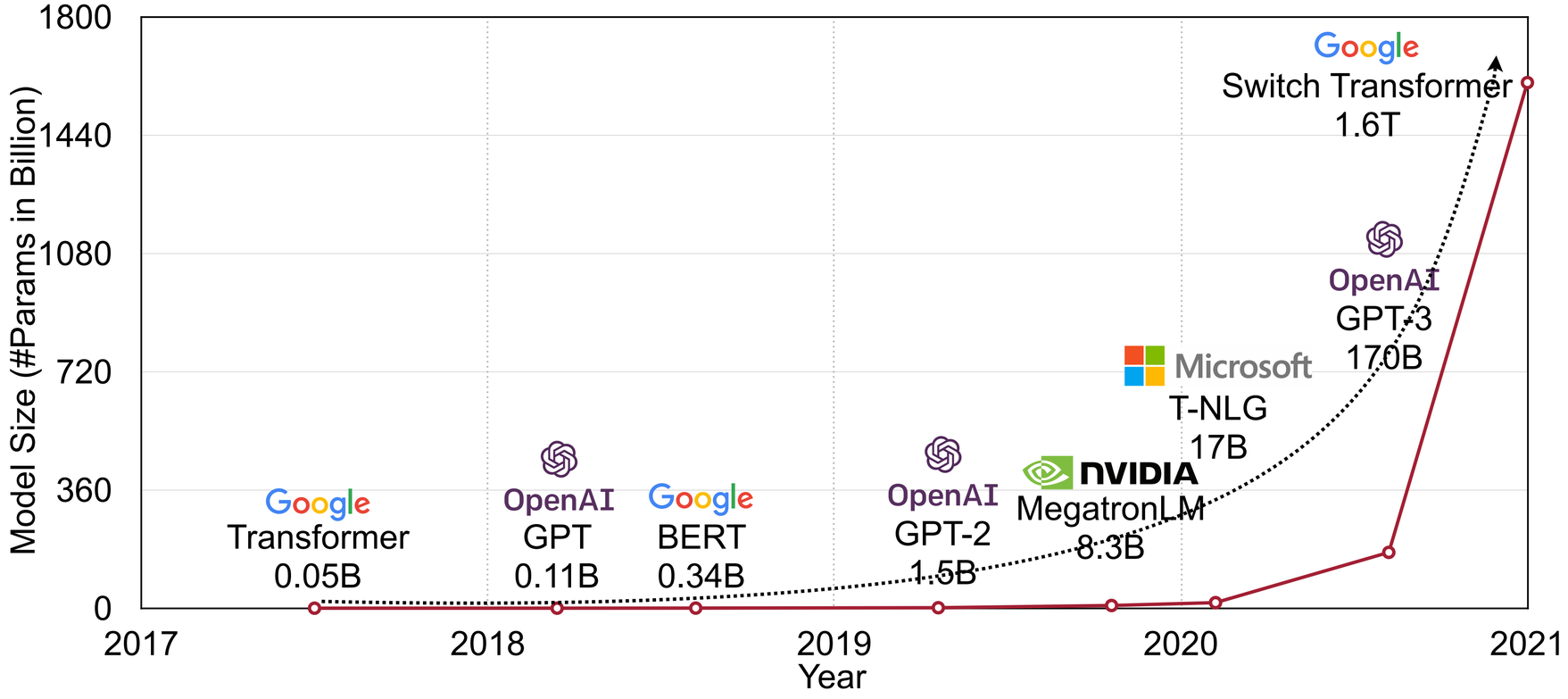
SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models

Guangxuan Xiao*, Ji Lin*, Mickael Seznec, Julien Demouth, Song Han

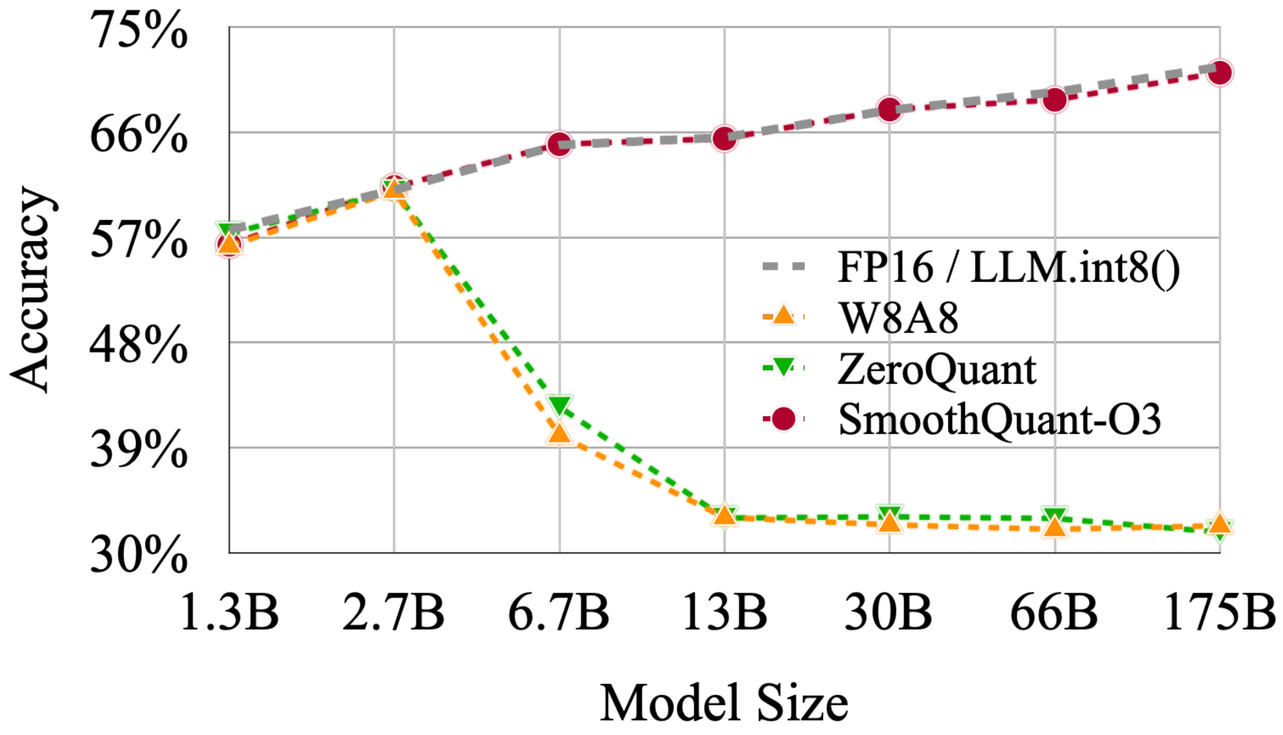
Massachusetts Institute of Technology
NVIDIA

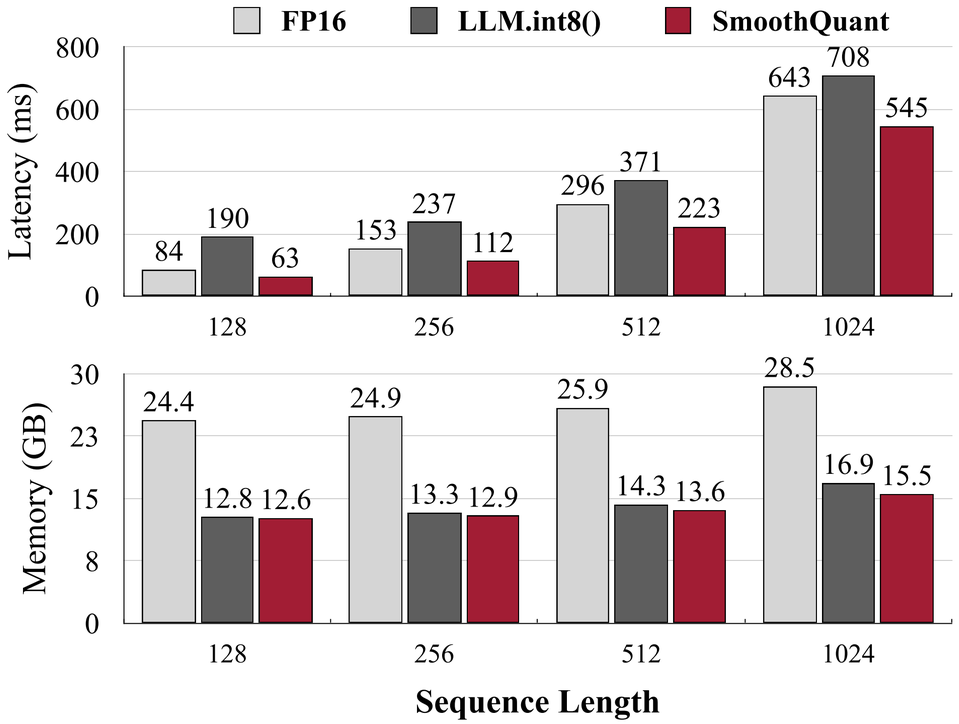
Quantization for LLMs is Important

- NLP model size and computation are increasing exponentially
- Quantizing **both weights and activations** can save memory and computation simultaneously.



Existing Quantization Method is Slow or Inaccurate

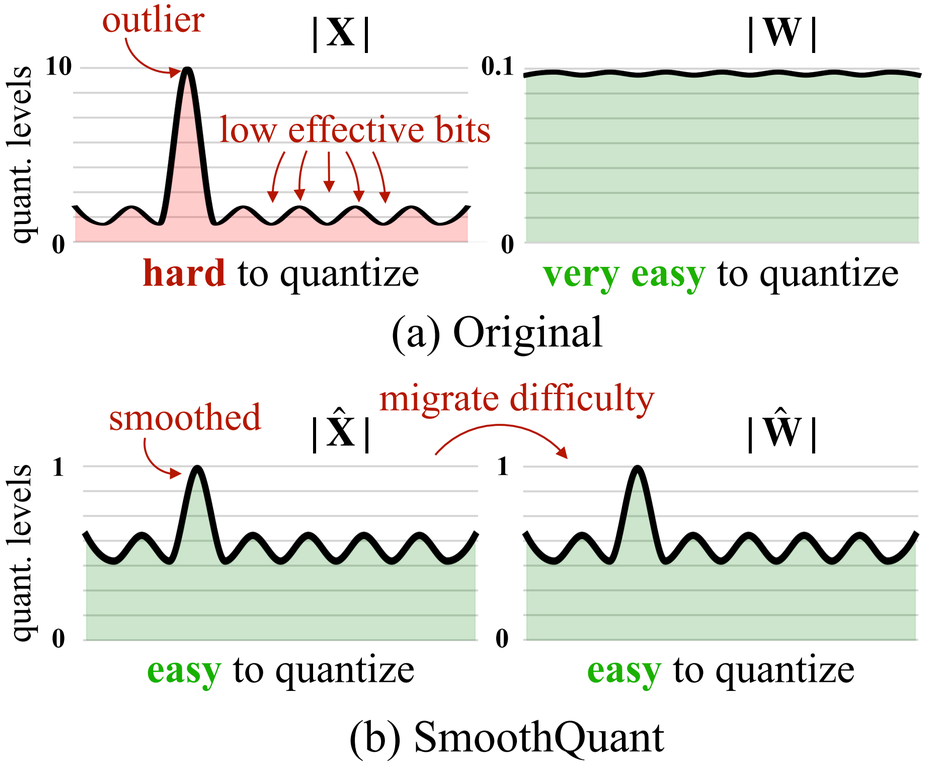




- Systematic outliers emerge in activations when we scale up LLMs beyond 6.7B. Naive but efficient quantization methods will destroy the accuracy.
- The accuracy-preserving baseline (Dettmers et al., 2022) uses FP16 to represent outliers, which needs runtime outlier detection, scattering and gathering. It is even slower than the FP16 inference.

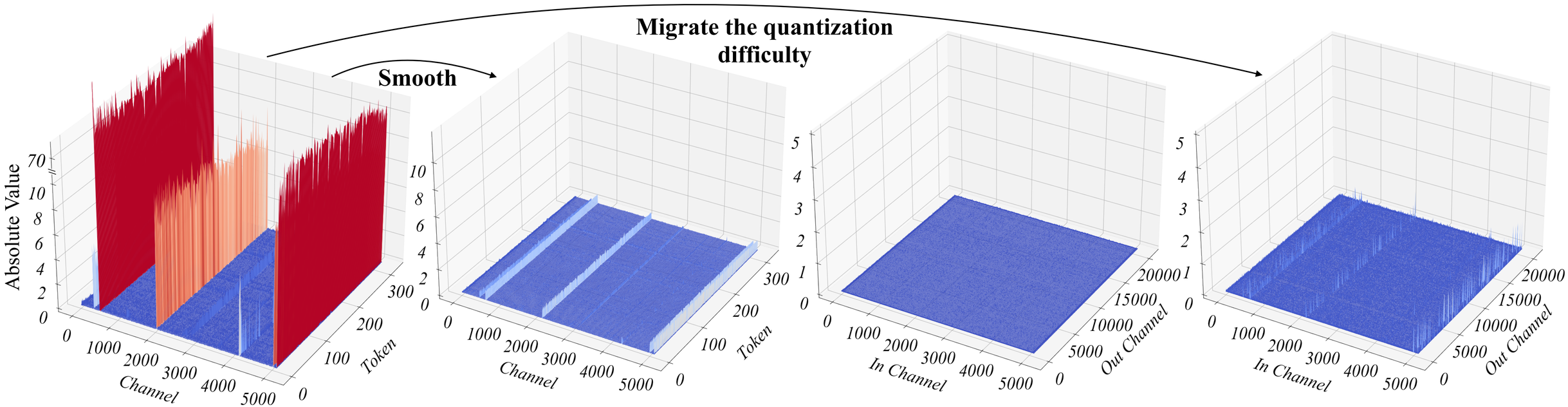
SmoothQuant: Accurate and Efficient Post-Training Quantization for LLMs

	LLM (100B+) Accuracy	Hardware Efficiency
ZeroQuant	✘	✔
Outlier Suppression	✘	✔
LLM.int8()	✔	✘
SmoothQuant	✔	✔



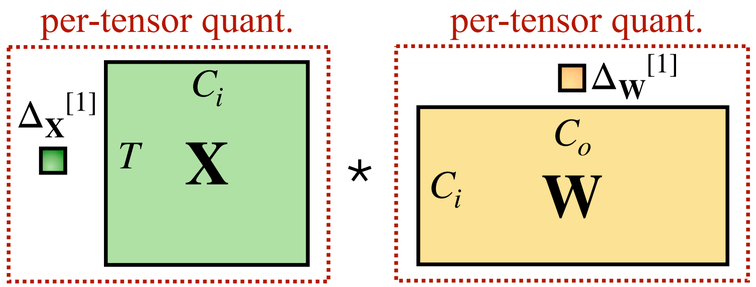
- We propose SmoothQuant, an **accurate** and **efficient** post-training-quantization (PTQ) method to enable 8-bit weight, 8-bit activation (**W8A8**) quantization for LLMs.
- Since **weights are easy** to quantize while **activations are not**, SmoothQuant smooths the activation outliers by **migrating the quantization difficulty from activations to weights** with a mathematically equivalent transformation.

Dissecting the Quantization Difficulty

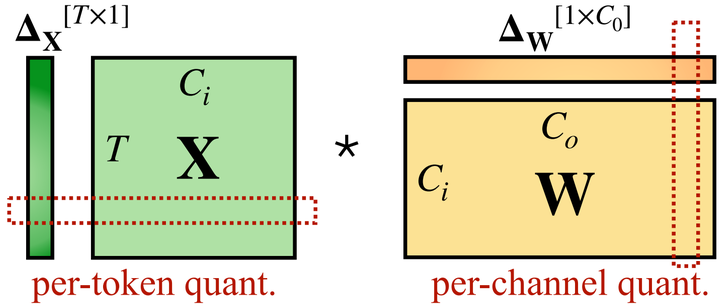


- Activations are harder to quantize than weights
- Outliers make activation quantization difficult
- Outliers persist in *fixed* channels

Quantization Schemes



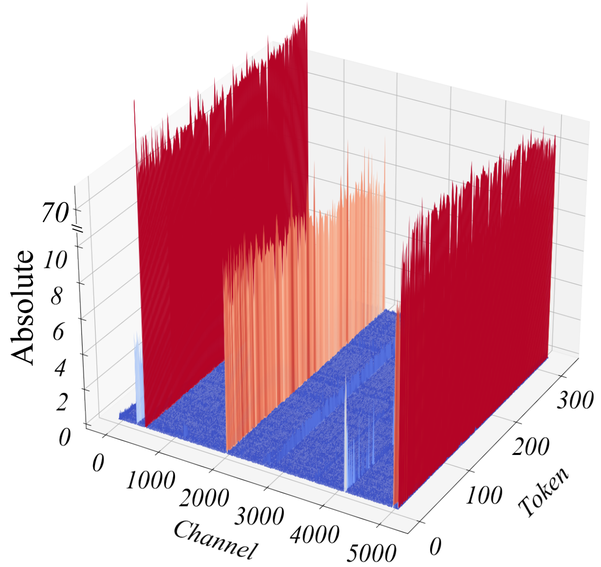
(a) per-tensor quantization



(b) per-token + per-channel quantization

$$\bar{\mathbf{X}}^{\text{INT8}} = \lceil \frac{\mathbf{X}^{\text{FP16}}}{\Delta} \rfloor, \quad \Delta = \frac{\max(|\mathbf{X}|)}{2^{N-1}-1}$$

$$\mathbf{Y} = diag(\boldsymbol{\Delta}_{\mathbf{X}}^{\text{FP16}}) \cdot (\bar{\mathbf{X}}^{\text{INT8}} \cdot \bar{\mathbf{W}}^{\text{INT8}}) \cdot diag(\boldsymbol{\Delta}_{\mathbf{W}}^{\text{FP16}})$$

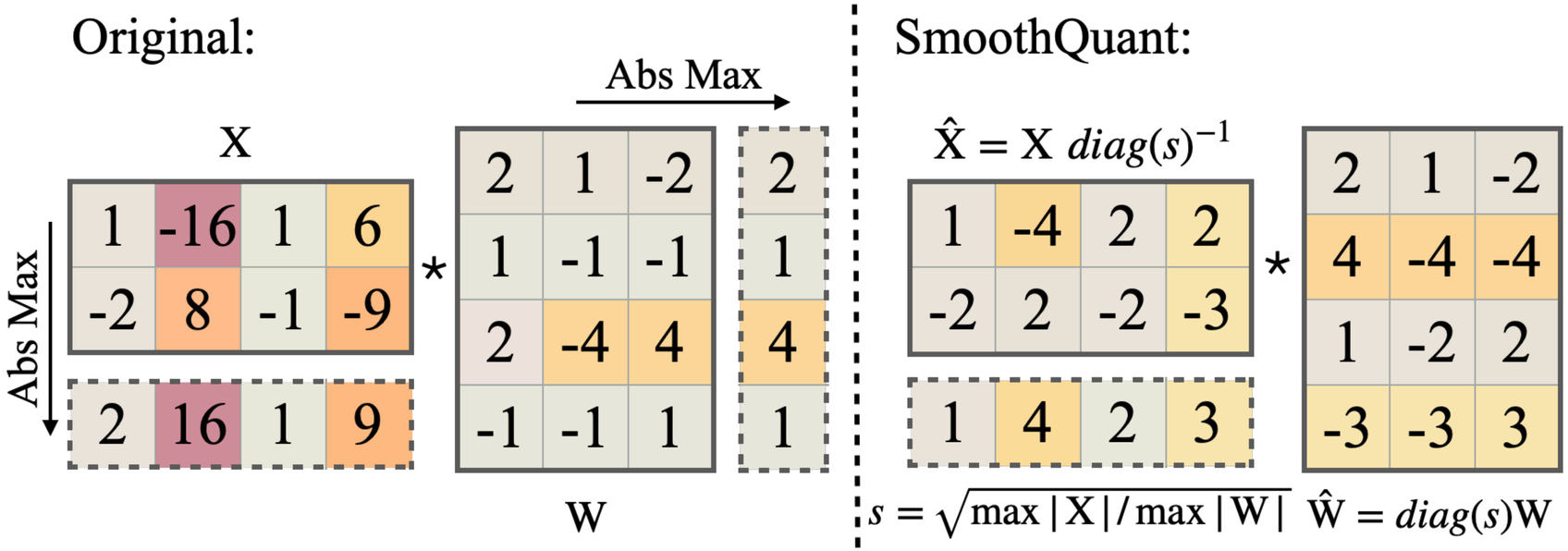


Activation
Hard to quantize

Model size	6.7B	13B	30B	66B	175B
FP16	64.9%	65.6%	67.9%	69.5%	71.6%
INT8 per-tensor	39.9%	33.0%	32.8%	33.1%	32.3%
INT8 per-token	42.5%	33.0%	33.1%	32.9%	31.7%
INT8 per-channel	64.8%	65.6%	68.0%	69.4%	71.4%

Among different activation quantization schemes, only per-channel quantization preserves the accuracy, but it is not compatible with INT8 GEMM kernels.

Activation Smoothing

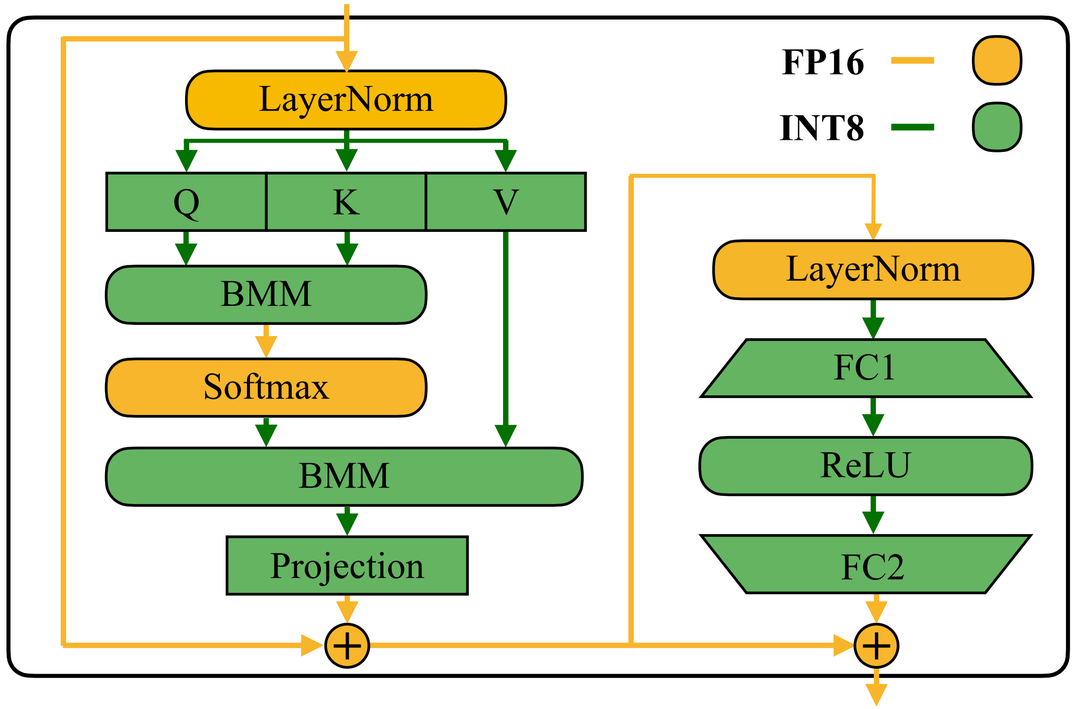


$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha},\ j = 1,2,\ldots,C_i$$

$$\mathbf{Y} = (\mathbf{X}diag(\mathbf{s})^{-1}) \cdot (diag(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}}$$

α: Migration Strength

System Implementation



Method	Weight	Activation
W8A8	per-tensor	per-tensor dynamic
ZeroQuant	group-wise	per-token dynamic
`LLM.int8()`	per-channel	per-token dynamic+FP16
Outlier Suppression	per-tensor	per-tensor static
SmoothQuant-O1	per-tensor	per-token dynamic
SmoothQuant-O2	per-tensor	per-tensor dynamic
SmoothQuant-O3	per-tensor	per-tensor static

- SmoothQuant's precision mapping for a Transformer block.
- All compute-intensive operators, such as linear layers and batched matrix multiplications (BMMs) use INT8 arithmetic.

- Quantization setting of the baselines and SmoothQuant. All weight and activations use INT8 representations unless specified.
- We implement three efficiency levels of quantization settings for SmoothQuant. The efficiency improves from O1 to O3.

Accuracy on OPT-175B

OPT-175B	LAMBADA	HellaSwag	PIQA	WinoGrande	OpenBookQA	RTE	COPA	**Average↑**	**WikiText↓**
FP16	74.7%	59.3%	79.7%	72.6%	34.0%	59.9%	88.0%	66.9%	10.99
W8A8	0.0%	25.6%	53.4%	50.3%	14.0%	49.5%	56.0%	35.5%	93080
ZeroQuant	0.0%*	26.0%	51.7%	49.3%	17.8%	50.9%	55.0%	35.8%	84648
`LLM.int8()`	74.7%	59.2%	79.7%	72.1%	34.2%	60.3%	87.0%	66.7%	11.10
Outlier Suppression	0.00%	25.8%	52.5%	48.6%	16.6%	53.4%	55.0%	36.0%	96151
SmoothQuant-O1	74.7%	59.2%	79.7%	71.2%	33.4%	58.1%	89.0%	66.5%	11.11
SmoothQuant-O2	75.0%	59.0%	79.2%	71.2%	33.0%	59.6%	88.0%	66.4%	11.14
SmoothQuant-O3	74.6%	58.9%	79.7%	71.2%	33.4%	59.9%	90.0%	66.8%	11.17

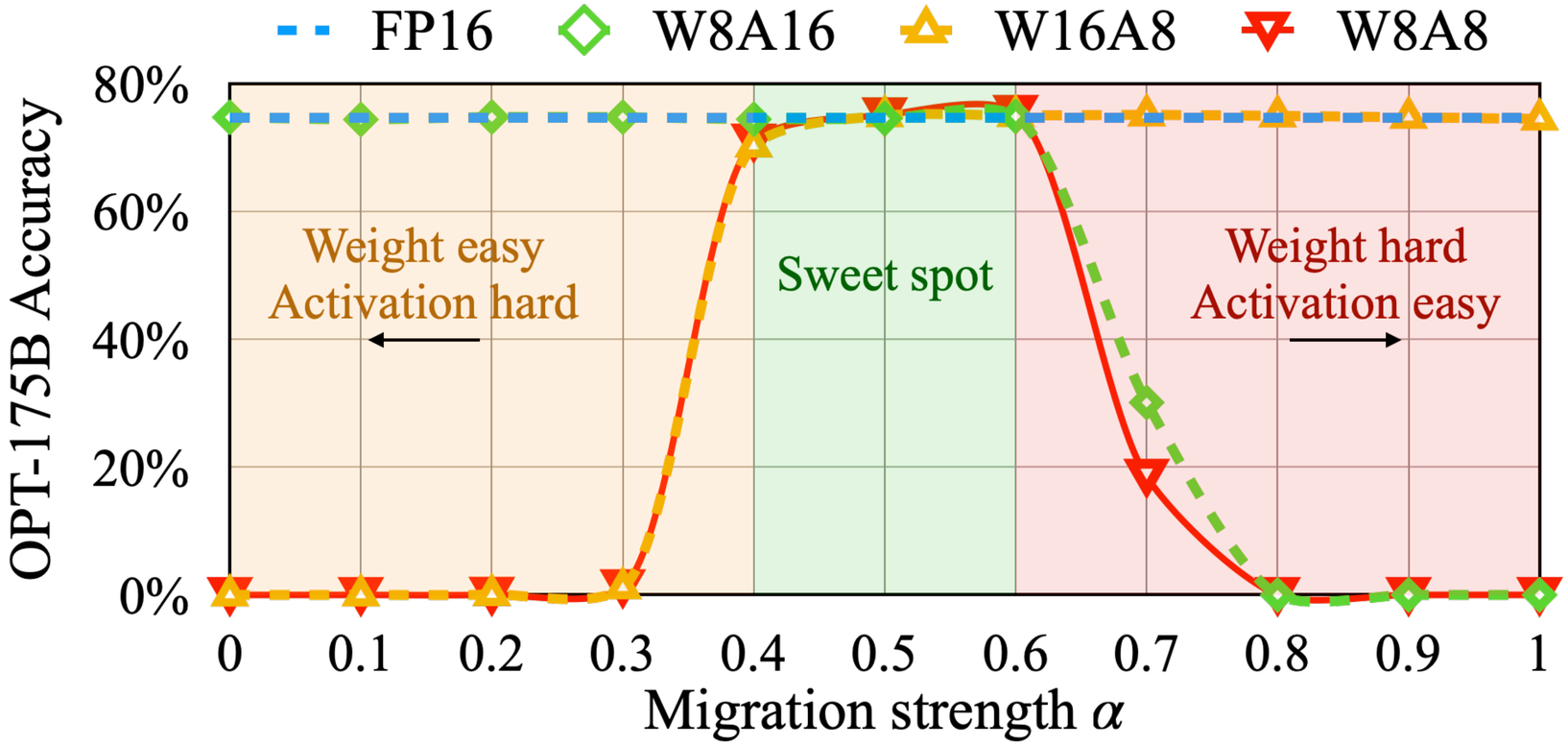
SmoothQuant maintains the accuracy of OPT-175B model after INT8 quantization, even with the most aggressive and most efficient O3 setting.

Accuracy on Different LLMs

Method	OPT-175B	BLOOM-176B	GLM-130B*
FP16	71.6%	68.2%	73.8%
W8A8	32.3%	64.2%	26.9%
ZeroQuant	31.7%	67.4%	26.7%
`LLM.int8()`	71.4%	68.0%	73.8%
Outlier Suppression	31.7%	54.1%	63.5%
SmoothQuant-O1	**71.2%**	68.3%	**73.7%**
SmoothQuant-O2	71.1%	**68.4%**	72.5%
SmoothQuant-O3	71.1%	67.4%	72.8%

SmoothQuant works for different LLMs. We can quantize the 3 largest, openly available LLM models into INT8 without degrading the accuracy.

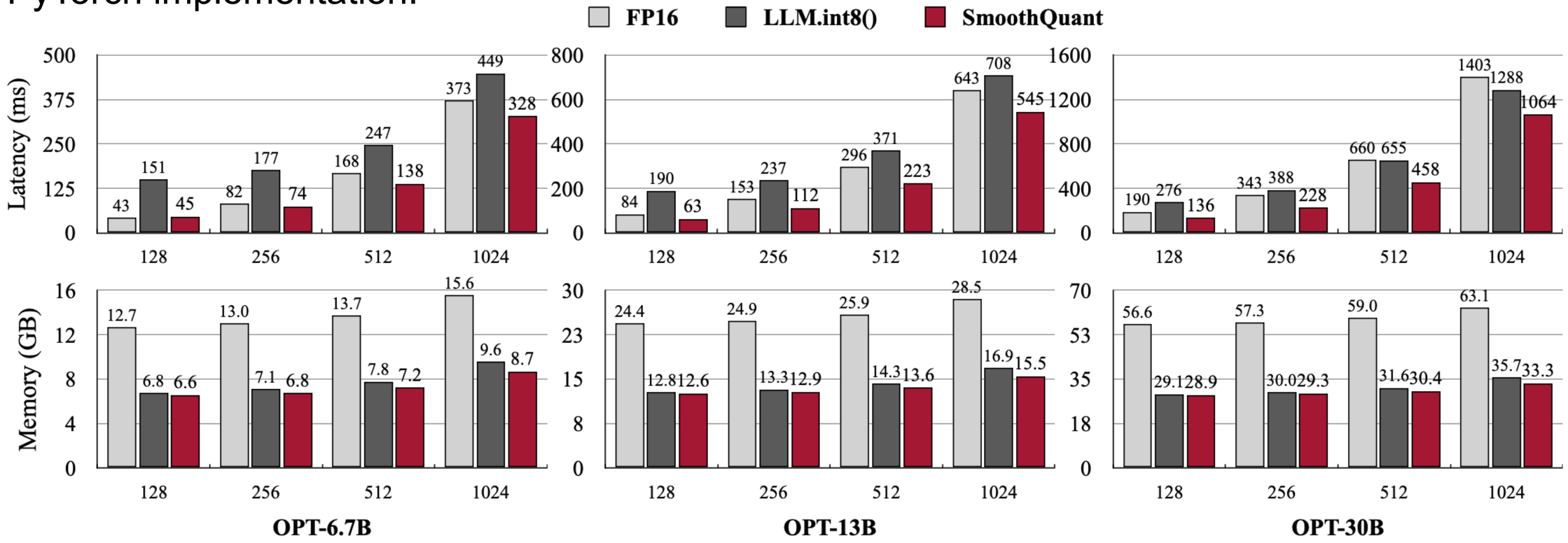
Ablation Study on the Migration Strength α



Migration strength α controls the amount of quantization difficulty migrated from activations to weights. A suitable migration strength α (sweet spot) makes both activations and weights easy to quantize. If the α is too large, weights will be hard to quantize; if too small, activations will be hard to quantize.

Speedup and Memory Saving

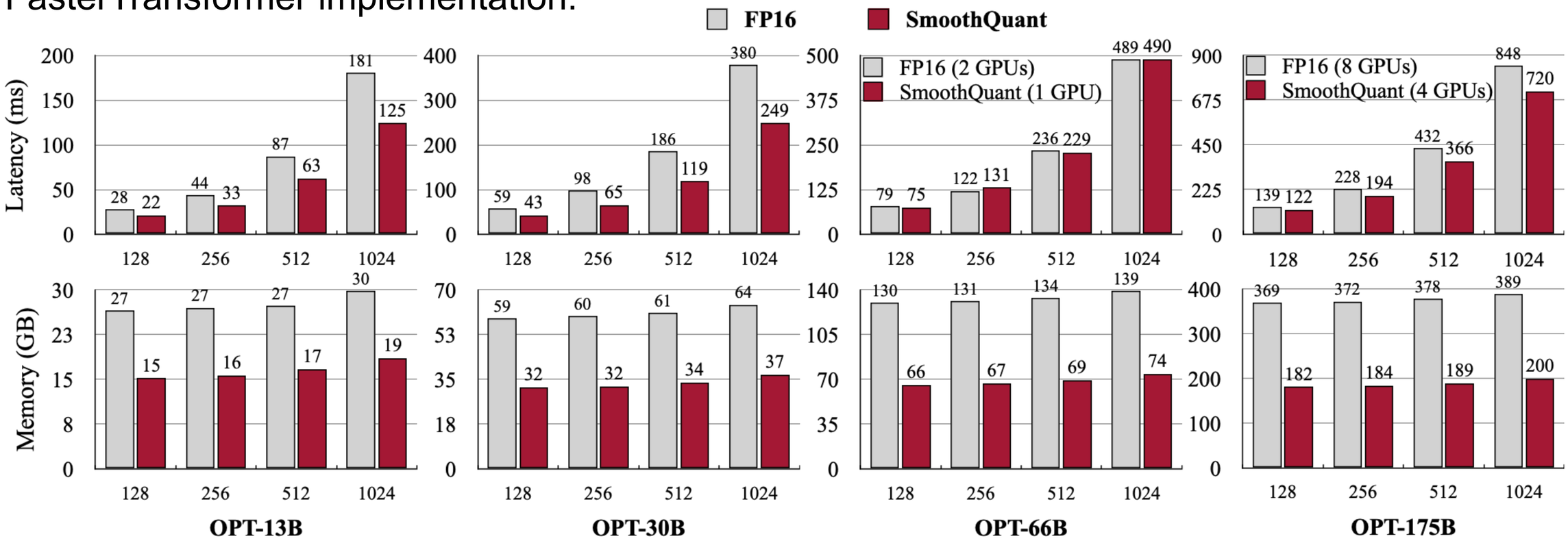
PyTorch implementation:



The PyTorch implementation of SmoothQuant achieves up to **1.51x** speedup and **1.96x** memory saving for OPT models on a single NVIDIA A100-80GB GPU, while **LLM.int8()** slows down the inference in most cases.

Speedup and Memory Saving

FasterTransformer implementation:

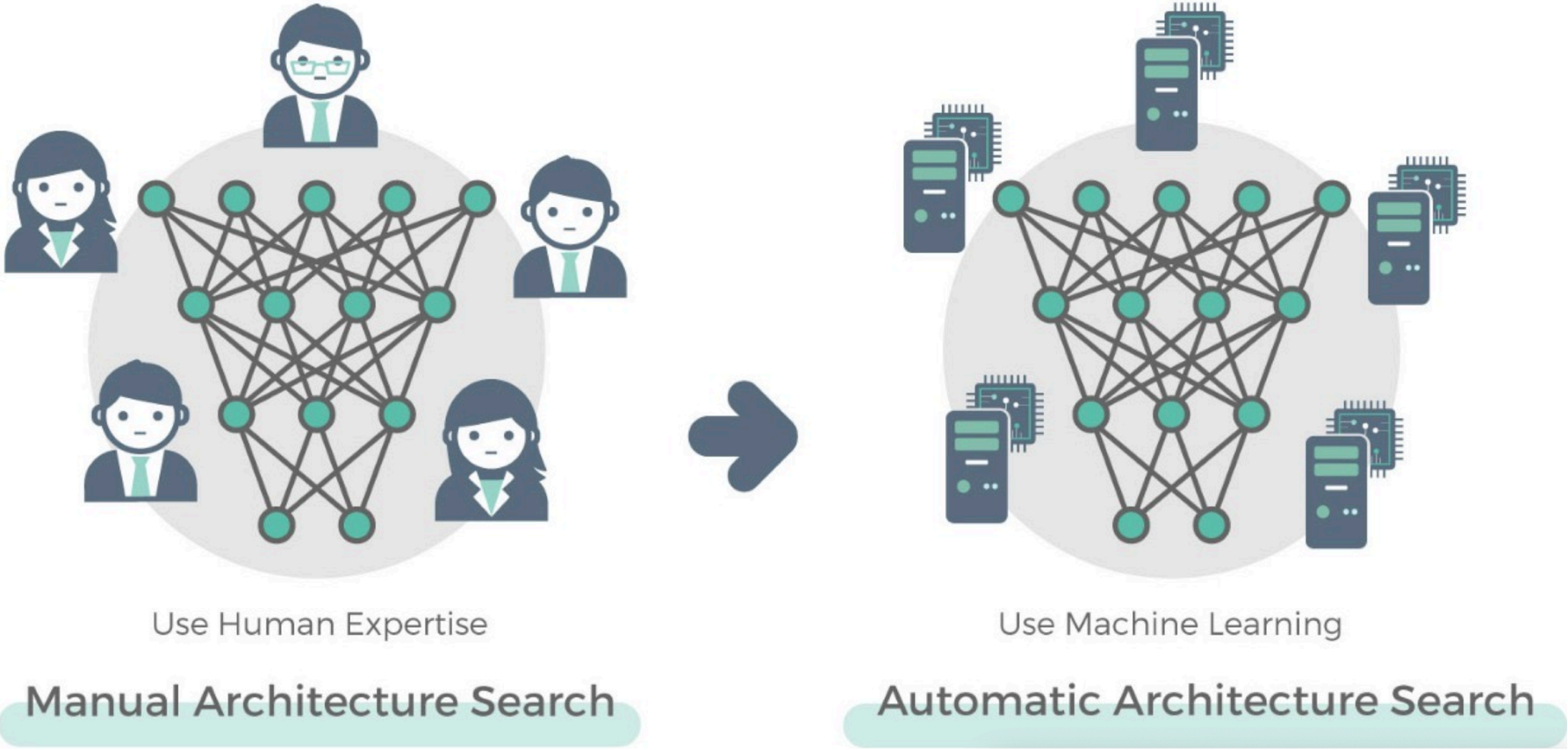


- We integrate SmoothQuant into FasterTransformer, a state-of-the-art Transformer serving framework.
- For smaller models, the latency can be significantly reduced with SmoothQuant by up to **1.56x** compared to FP16.
- For the bigger models (OPT-66B and 175B), we can achieve similar or even faster inference using only **half** number of GPUs. Memory footprint is almost halved compared to FP16.

Neural Architecture Search

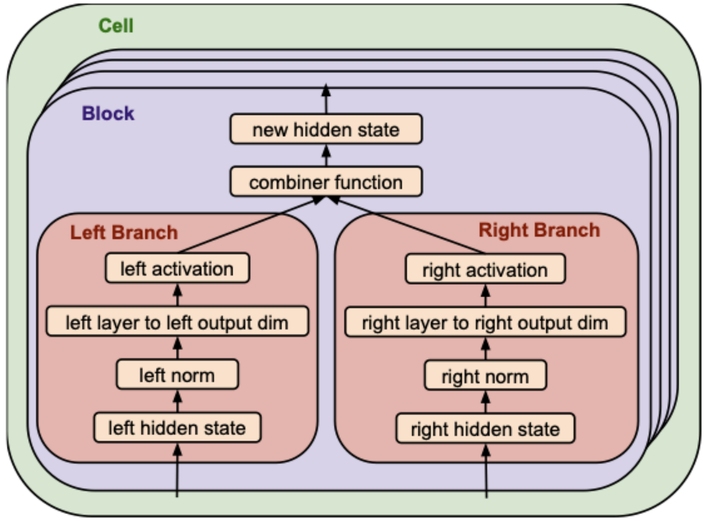
Lecture 07
Neural Architecture Search
Part I

Lecture 08
Neural Architecture Search
Part II



Neural Architecture Search — ET

Evolved Transformer

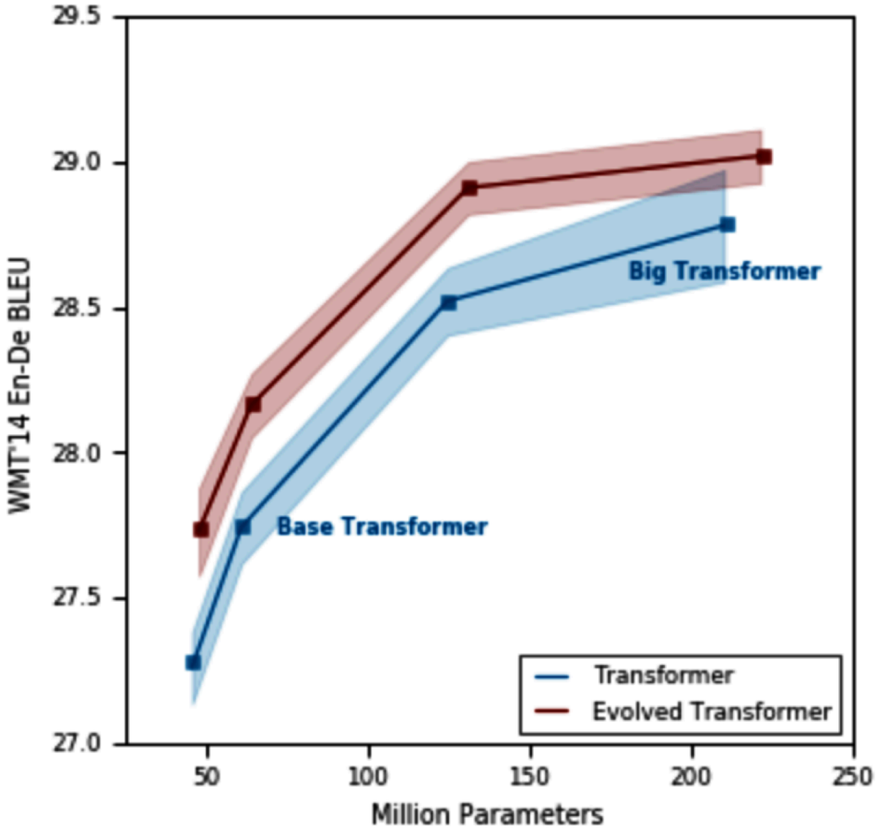


NASNet-style blocks:

- Inputs: Two hidden states.
- Output: New hidden state.

Search space:

- Encoder has 6 blocks.
- Decoder has 8 blocks.

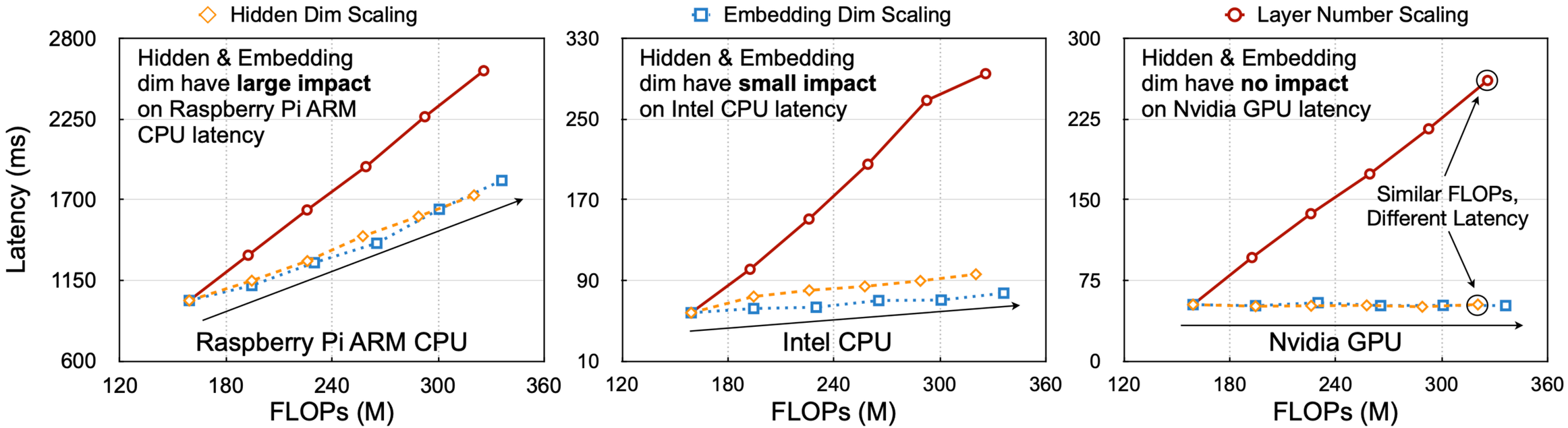


- Tournament selection evolutionary architecture search.
- Dynamically allocate resources to more promising architectures according to their fitness.

The Evolved Transformer [So *et al.*, 2019]

Neural Architecture Search — HAT

Hardware-Aware Transformers

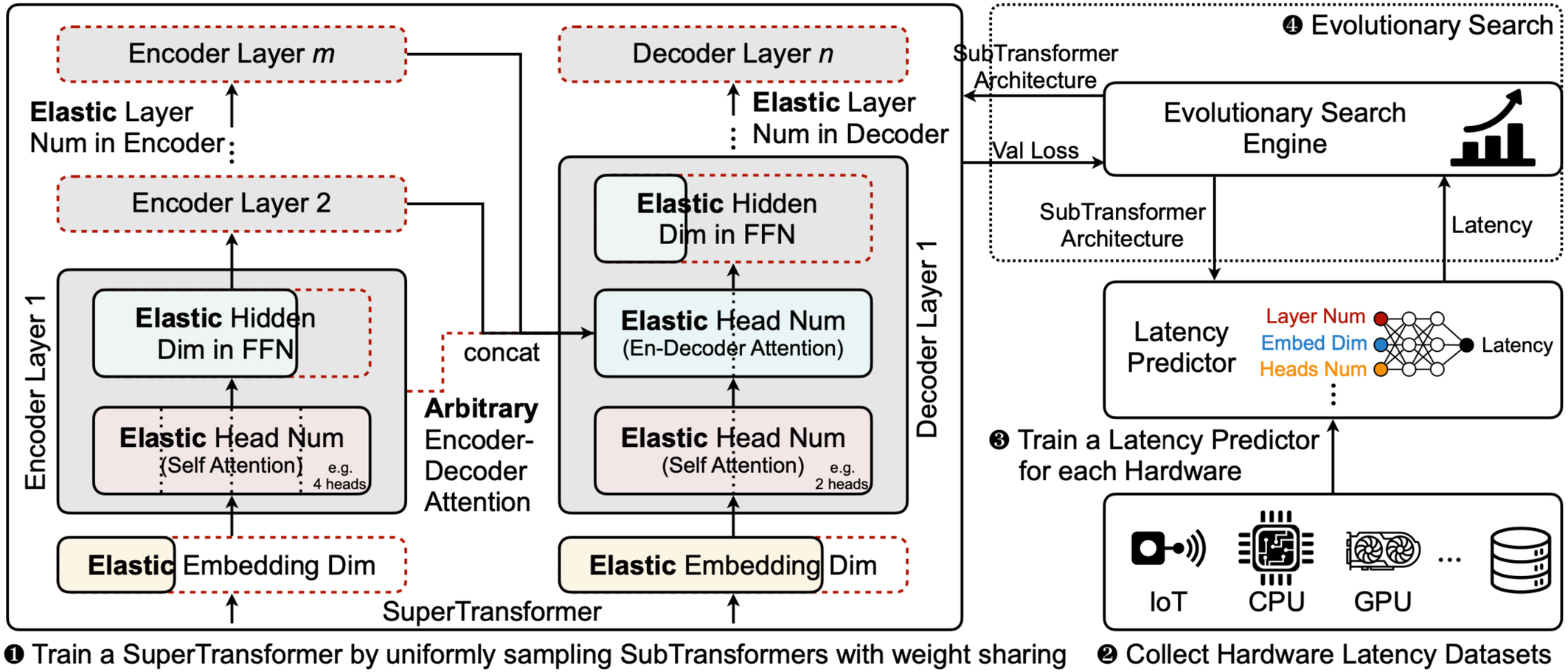


- FLOPs **does not reflect** the real measured latency.
- Latency influencing factors of different hardware are **contrasting**.
- Need **hardware latency** feedback to design **specialized** models for different hardware!

HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, 2020]

Neural Architecture Search — HAT

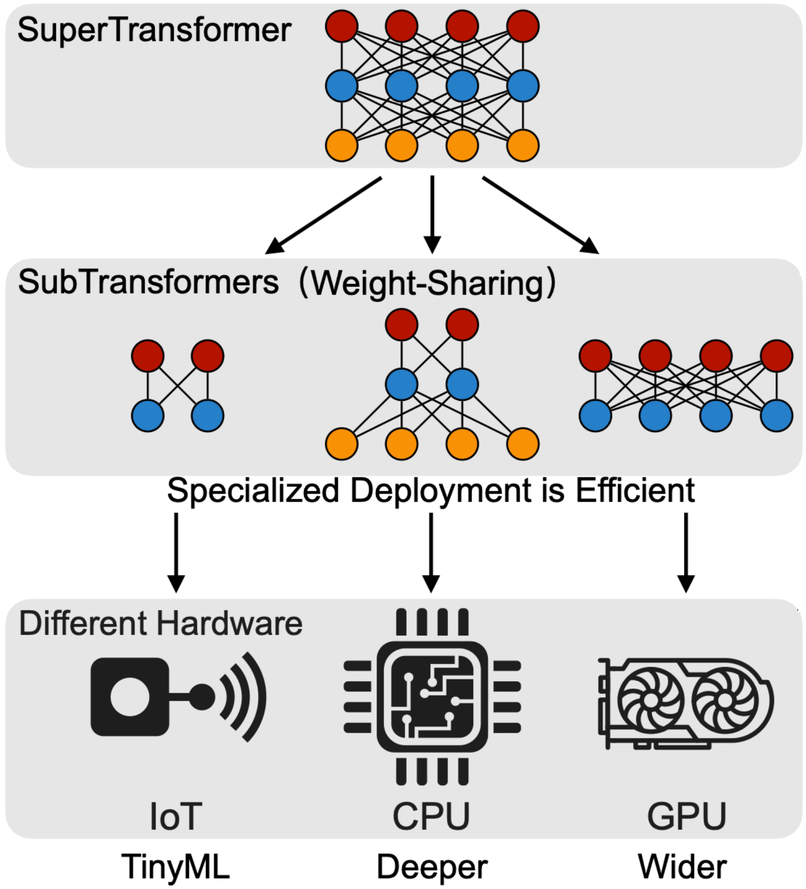
Hardware-Aware Transformers

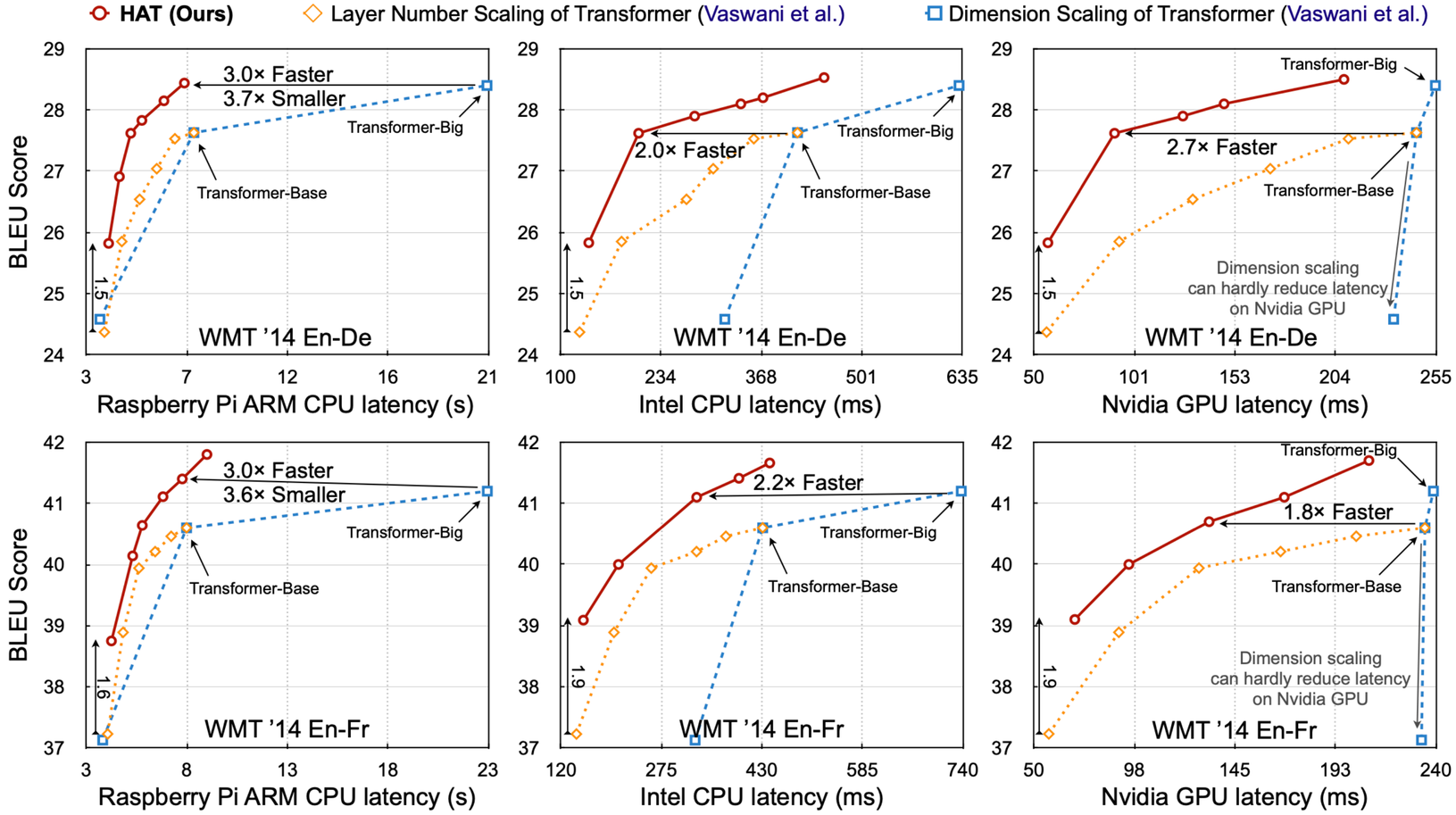


HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, 2020]

Neural Architecture Search — HAT

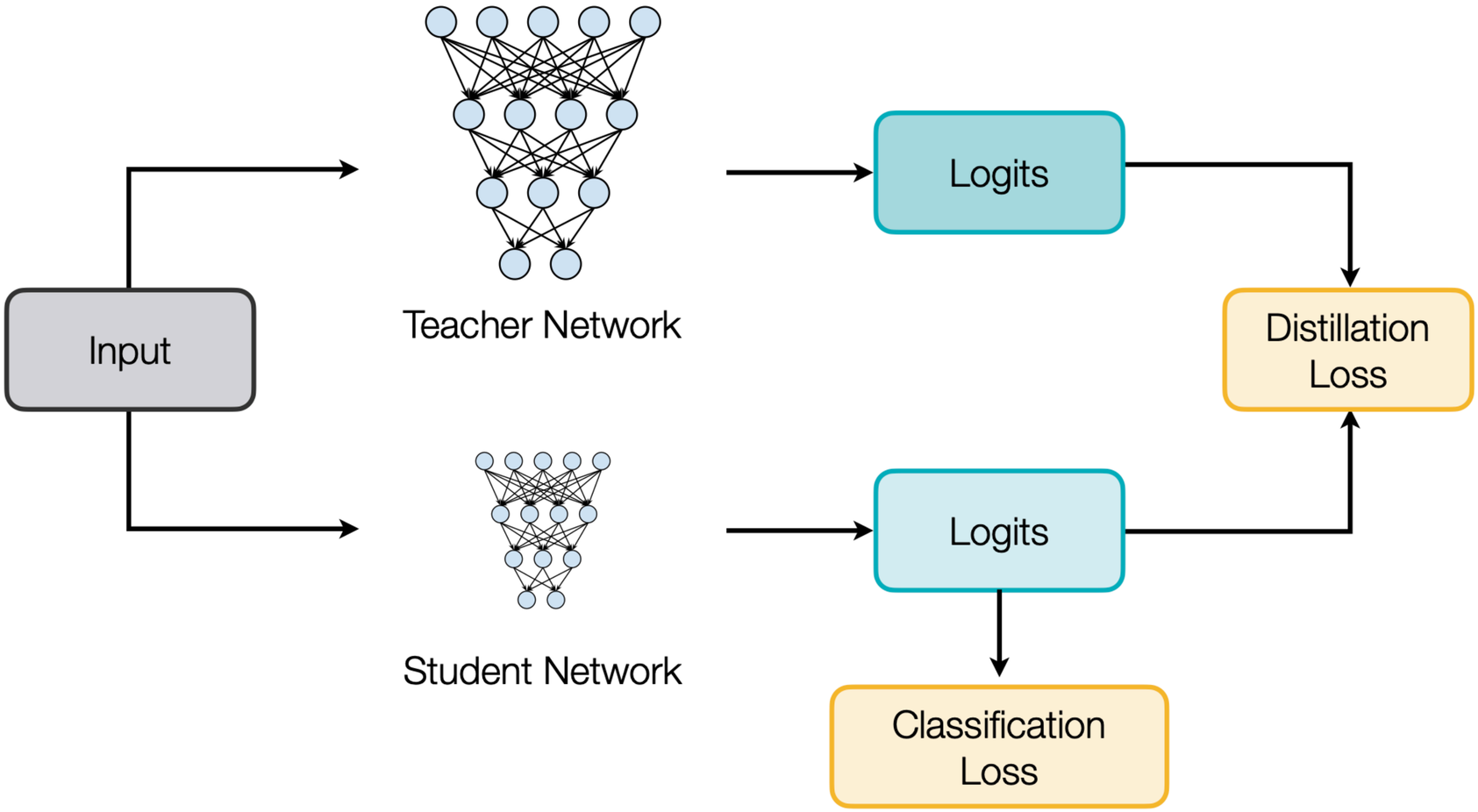
Hardware-Aware Transformers





HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, 2020]

Knowledge Distillation



Knowledge Distillation — DistilBERT/TinyBERT

Feature Distillation + Attention Distillation

- **Attention distillation**: The student learns to fit the matrices of multi-head attention in the teacher network.

$$\mathcal{L}_{\text{attn}} = \frac{1}{h}\sum_{i=1}^{h} \texttt{MSE}(\boldsymbol{A}_i^S, \boldsymbol{A}_i^T)$$

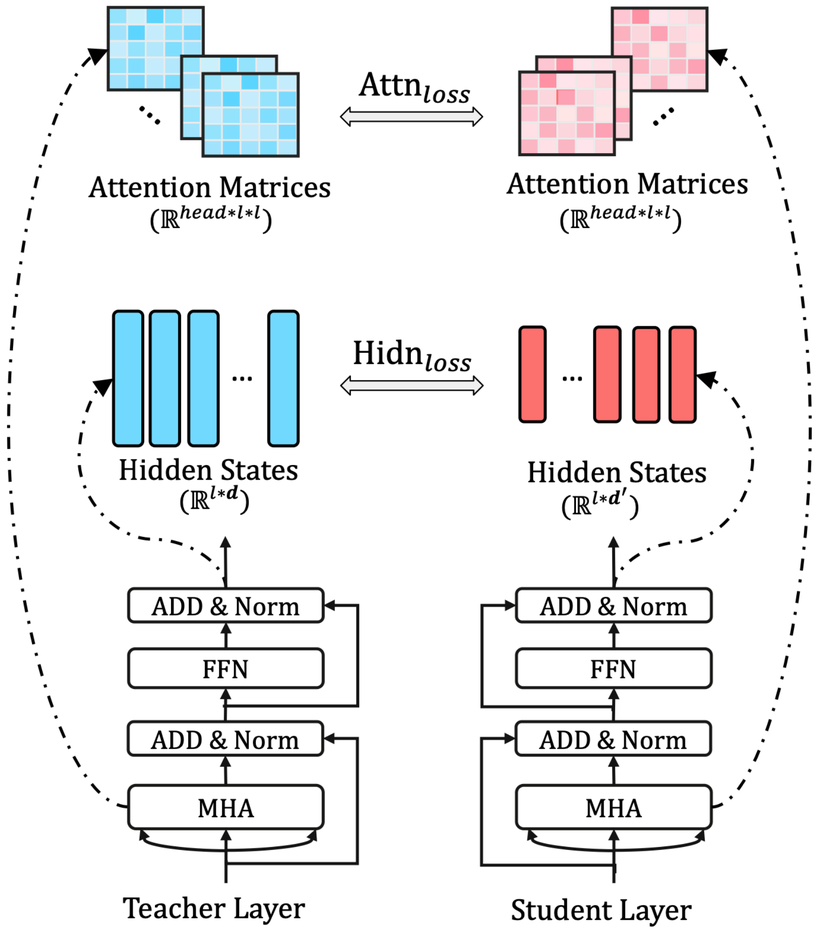
- **Hidden state distillation**: Distill the knowledge from the output of the transformer block.

$$\mathcal{L}_{\text{hidn}} = \texttt{MSE}(\boldsymbol{H}^S \boldsymbol{W}_h, \boldsymbol{H}^T)$$

Linear projection to match the dimension

- **Embedding distillation**: $\mathcal{L}_{\text{embd}} = \texttt{MSE}(\boldsymbol{E}^S \boldsymbol{W}_e, \boldsymbol{E}^T)$

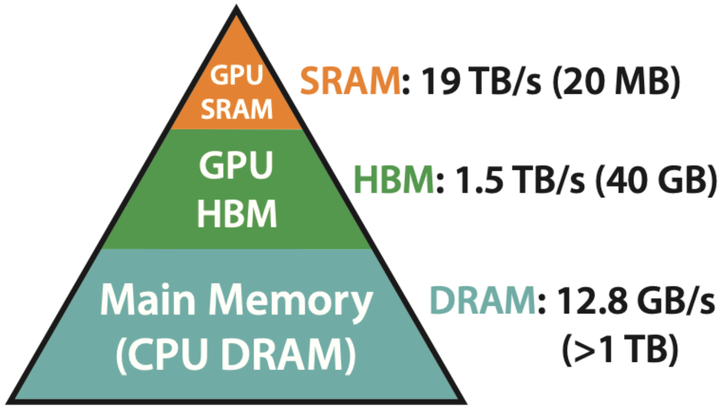
- **Prediction distillation**: $\mathcal{L}_{\text{pred}} = \texttt{CE}(\boldsymbol{z}^T/t, \boldsymbol{z}^S/t)$



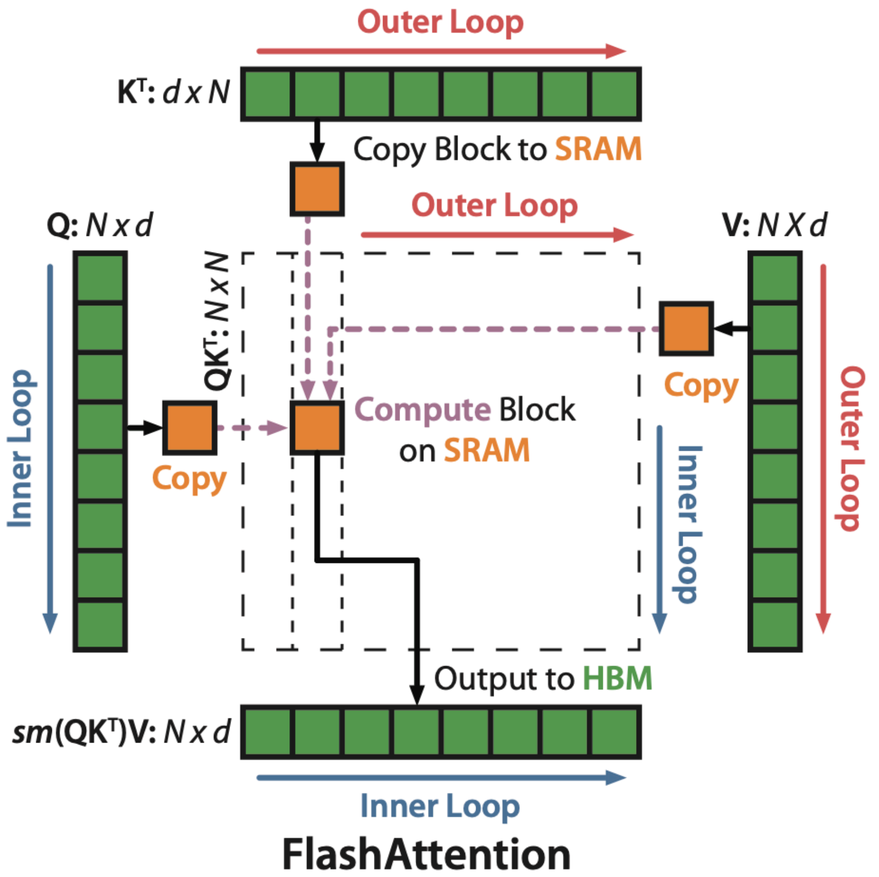
TinyBERT: Distilling BERT for Natural Language Understanding [Jiao *et al.*, 2020]

IV. System Support

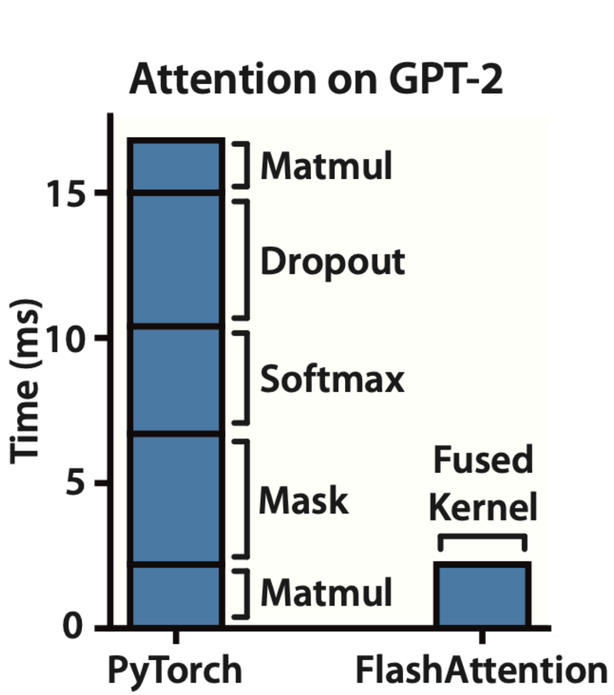
FlashAttention



Memory Hierarchy with Bandwidth & Memory Size



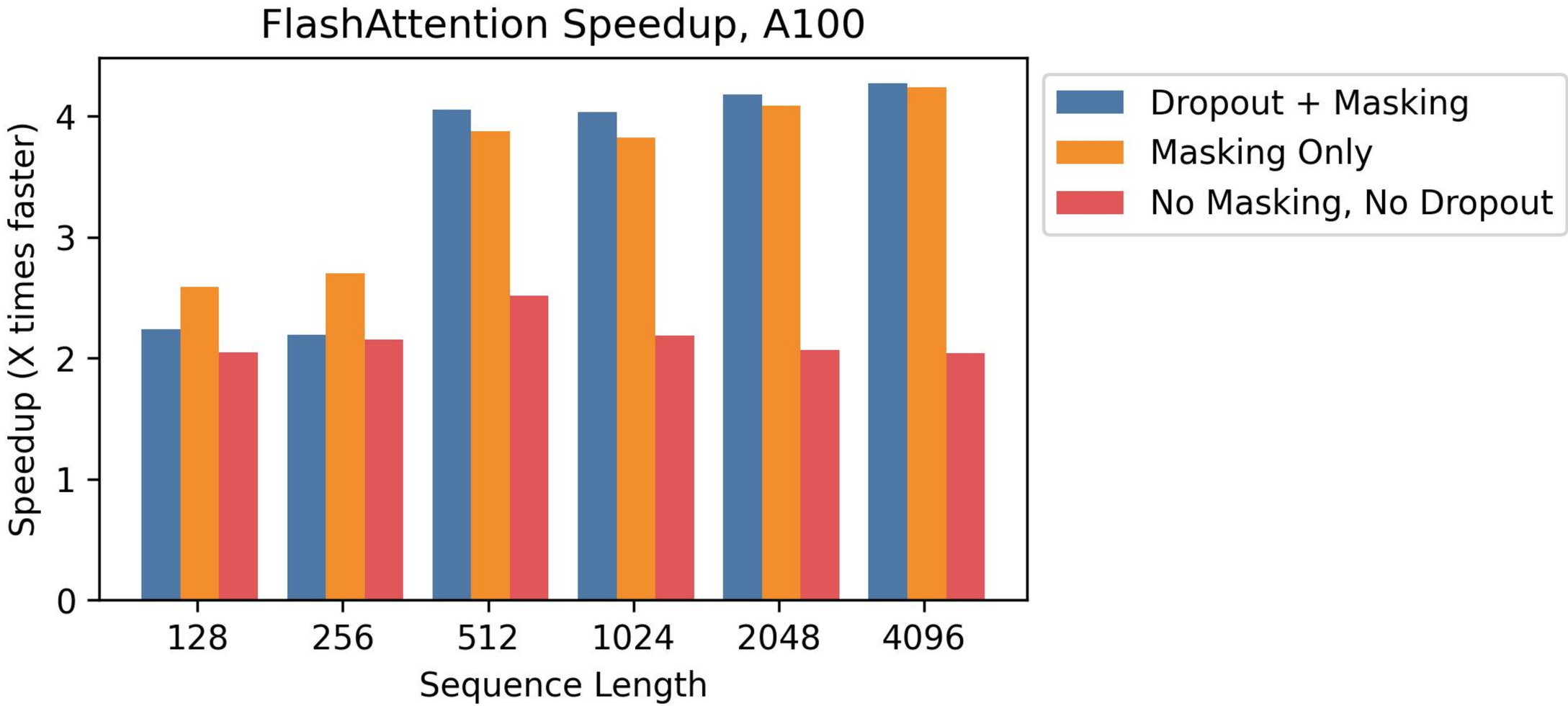
FlashAttention



- Uses **tiling** to reduce the number of **memory reads/writes** between high bandwidth memory (HBM) and on-chip SRAM (which is very expensive).

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness [Dao *et al.*, 2022]

FlashAttention



FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness [Dao *et al.*, 2022]

FasterTransformer

This repository provides a script and recipe to run the highly optimized transformer-based encoder and decoder component, and it is tested and maintained by NVIDIA.

Table Of Contents

deepspeed

Latest News

DeepSpeed trained the world's most powerful language models (MT-530B, BLOOM); learn how.

- [2022/11] Stable Diffusion Image Generation under 1 second w. DeepSpeed MII
- [2022/10] DeepSpeed-MII: instant speedup on 24,000+ open-source DL models with up to 40x cheaper inference
- [2022/09] ZeRO-Inference: Democratizing massive model inference
- [2022/07] Azure and DeepSpeed empower easy-to-use and high-performance model training
- [2022/07] DeepSpeed Compression: A composable library for extreme compression

Extreme Speed and Scale for DL Training and Inference

Colossal-AI

Colossal-AI

Colossal-AI: A Unified Deep Learning System for Big Model Era

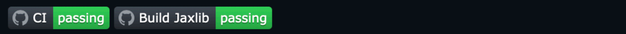
Paper | Documentation | Examples | Forum | Blog

| English | 中文 |

Latest News

- [2022/11] Diffusion Pretraining and Hardware Fine-Tuning Can Be Almost 7X Cheaper
- [2022/10] Use a Laptop to Analyze 90% of Proteins, With a Single-GPU Inference Sequence Exceeding 10,000
- [2022/10] Embedding Training With 1% GPU Memory and 100 Times Less Budget for Super-Large

Alpa



Documentation | Slack

Alpa is a system for training and serving large-scale neural networks.

Scaling neural networks to hundreds of billions of parameters has enabled dramatic breakthroughs such as GPT-3, but training and serving these large-scale neural networks require complicated distributed system techniques. Alpa aims to automate large-scale distributed training and serving with just a few lines of code.

The key features of Alpa include:

Automatic Parallelization. Alpa automatically parallelizes users' single-device code on distributed clusters with data, operator, and pipeline parallelism.

Excellent Performance. Alpa achieves linear scaling on training models with billions of parameters on

Summary of Today's Lecture

In this lecture, we introduce:

- Basics and applications of transformers
- Efficient transformers
- System support for transformers

In the next lecture, we will introduce:

- Basics of quantum computing