Natural Language Processing

Pretraining LMs

Instructor: Yangqiu Song

Slides credits: Anna Goldie, John Hewitt, Carter Blair and Cole Wyeth, Wenhu Chen

Outline

1. Motivating model pretraining from word embeddings
2. Model pretraining three ways
    1. Encoders
    2. Encoder-Decoders
    3. Decoders

Motivating word meaning and context

- Recall the adage we mentioned at the beginning of the course:
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)

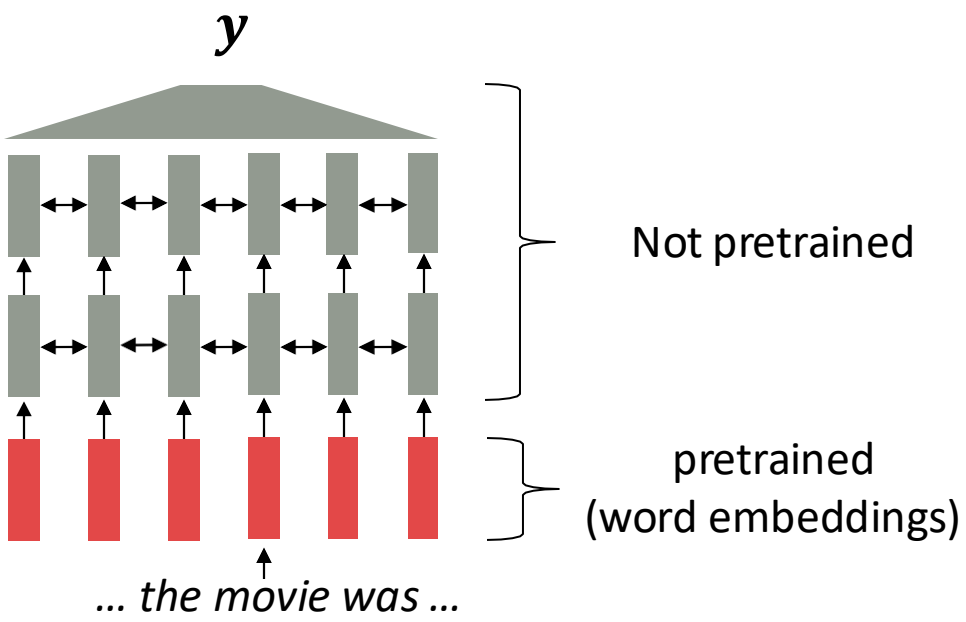
- This quote is a summary of **distributional semantics**, and motivated **word2vec**. But:
  - "… the complete meaning of a word is always contextual, and no study of meaning apart from a complete context can be taken seriously." (J. R. Firth 1935)

- Consider *I* ***record*** *the* ***record***: the two instances of ***record*** mean different things.

[Sennrich et al., 2016, Wu et al., 2016]

Where we were: pretrained word embeddings

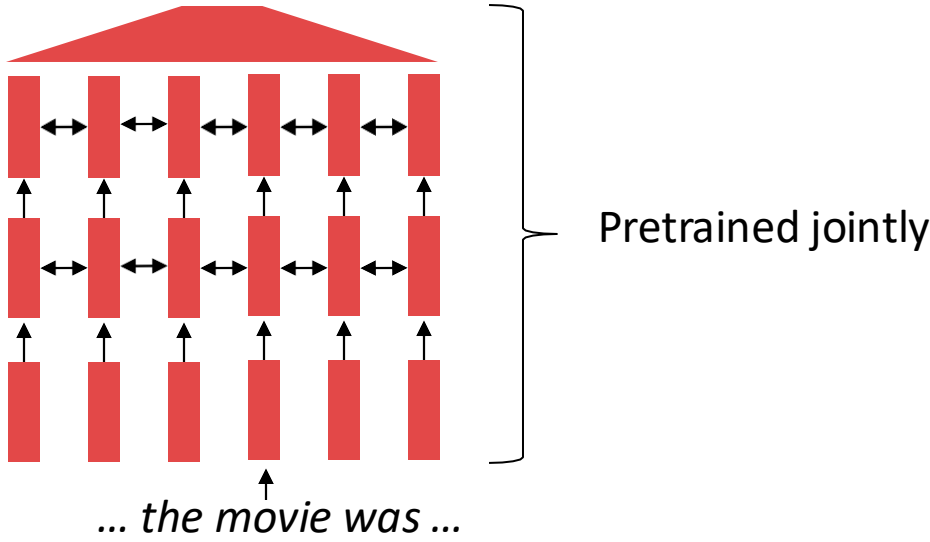
- Circa 2017:
  - Start with pretrained word embeddings (no context!)
  - Learn how to incorporate context in an LSTM or Transformer while training on the task.
- **Some issues to think about:**
  - The training data we have for our **downstream task** (like question answering) must be sufficient to teach all contextual aspects of language.
  - Most of the parameters in our network are randomly initialized!



[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

Where we were: pretrained word embeddings

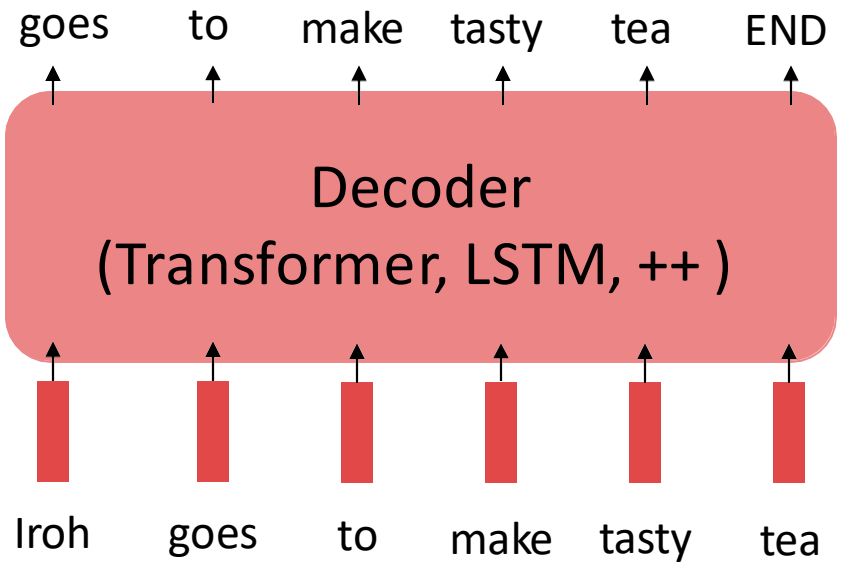
- In modern NLP:
  - All (or almost all) parameters in NLP networks are initialized via **pretraining**.
  - Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.
  - This has been exceptionally effective at building strong:
    - **representations of language**
    - **parameter initializations** for strong NLP models.
    - **Probability distributions** over language that we can sample from



[This model has learned how to represent entire sentences through pretraining]

Pretraining through language modeling [Dai and Le, 2015]

- Recall the **language modeling** task:
  - Model $p_\theta(w_t \mid w_{1:t-1})$, the probability distribution over words given their past contexts.
  - There's lots of data for this! (In English.)
- **Pretraining through language modeling:**
  - Train neural network to perform language modeling on a large amount of text.
  - Save the network parameters.

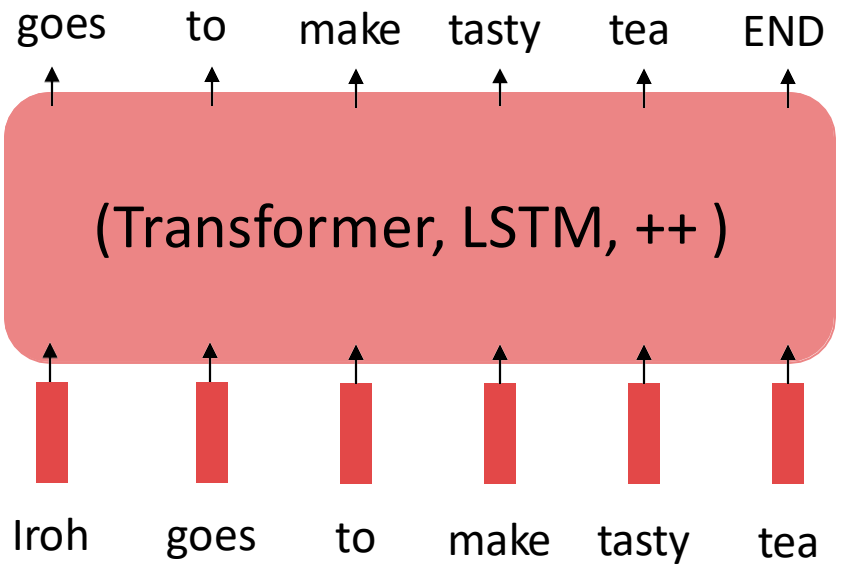


The Pretraining / Finetuning Paradigm

- Pretraining can improve NLP applications by serving as parameter initialization.

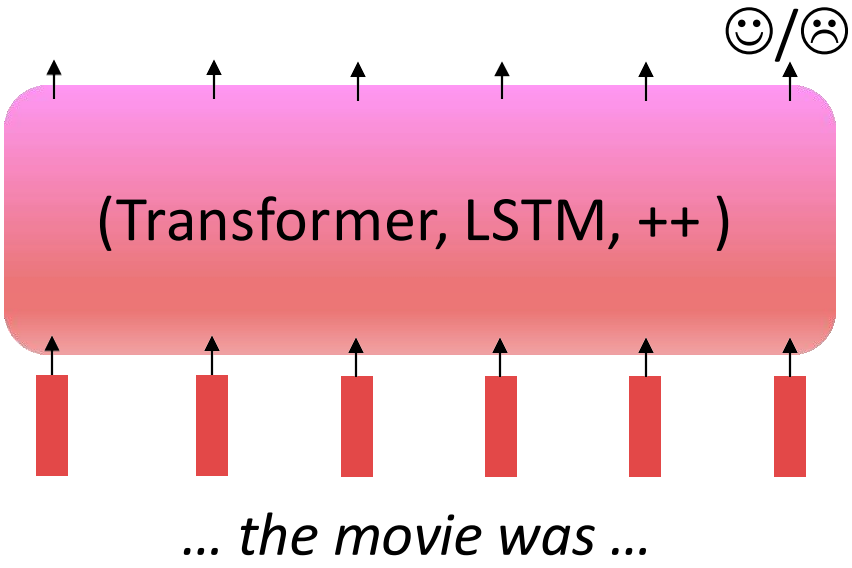
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!



Stochastic gradient descent and pretrain/finetune

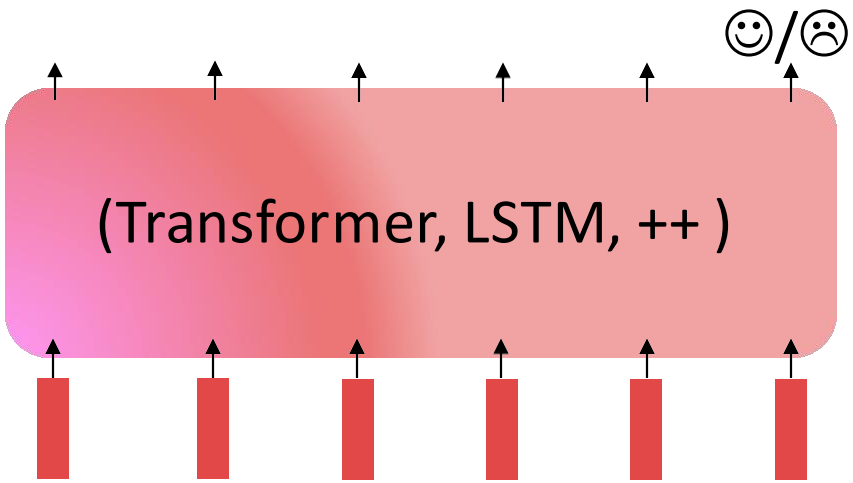
- Why should pretraining and finetuning help, from a "training neural nets" perspective?
- Consider, provides parameters by $\min \mathcal{L}_{\text{pretrain}}$ (The pretraining loss.)
- Then, finetuning approximates $\min \mathcal{L}_{\text{finetune}}$, starting at the pretrained parameters (The finetuning loss)
- The pretraining may matter because stochastic gradient descent sticks (relatively) close to optimal during finetuning.
  - So, maybe the finetuning local minima tend to generalize well!
  - And/or, maybe the gradients of finetuning loss propagate nicely!

Full Finetuning vs. Parameter-Efficient Finetuning

- Finetuning every parameter in a pretrained model works well, but is memory-intensive. But **lightweight** finetuning methods adapt pretrained models in a constrained way.
  - Leads to **less overfitting** and/or **more efficient finetuning and inference.**

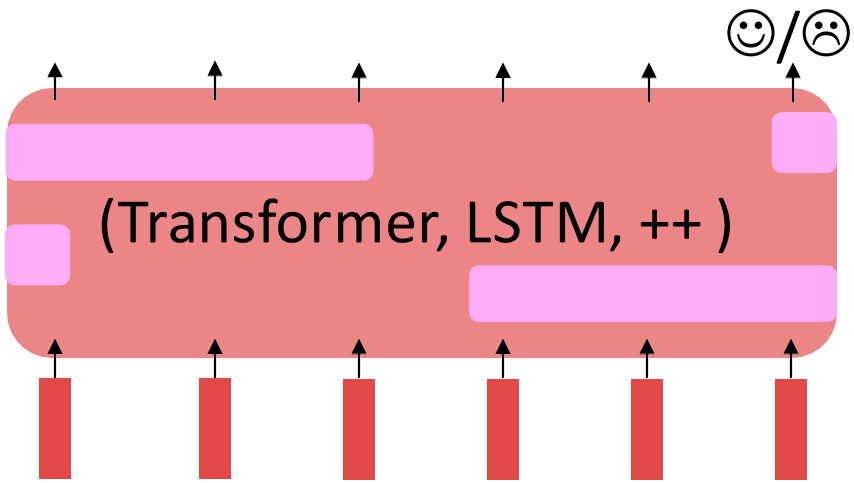
Full Finetuning

Adapt all parameters



Lightweight Finetuning

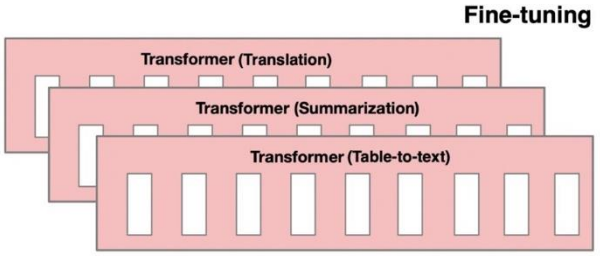
Train a few existing or new parameters



[Liu et al., 2019; Joshi et al., 2020]

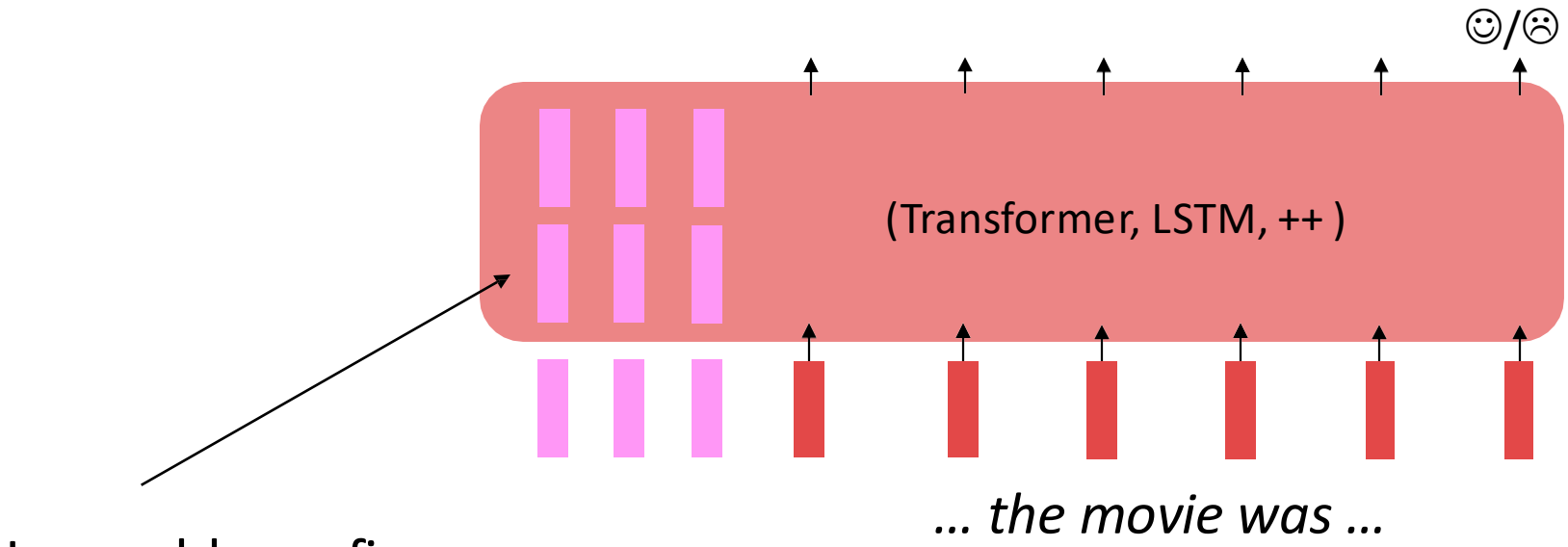
Fine tuning

- With standard fine-tuning, we need to make a **new copy** of the model for each task.
- In the extreme case of a different model per user, we could never store 1000 different full models.



Prefix-Tuning

- Prefix-Tuning adds a **prefix** of parameters, and **freezes all pretrained parameters.**
  - The prefix is processed by the model just like real words would be.
  - Advantage: each element of a batch at inference could run a different tuned model.



[Liu et al., 2019; Joshi et al., 2020]

LoRA

- Low-Rank Adaptation of Large Language Models
  - Freezes the pretrained model weights
  - Injects trainable rank decomposition matrices into each layer of the Transformer architecture

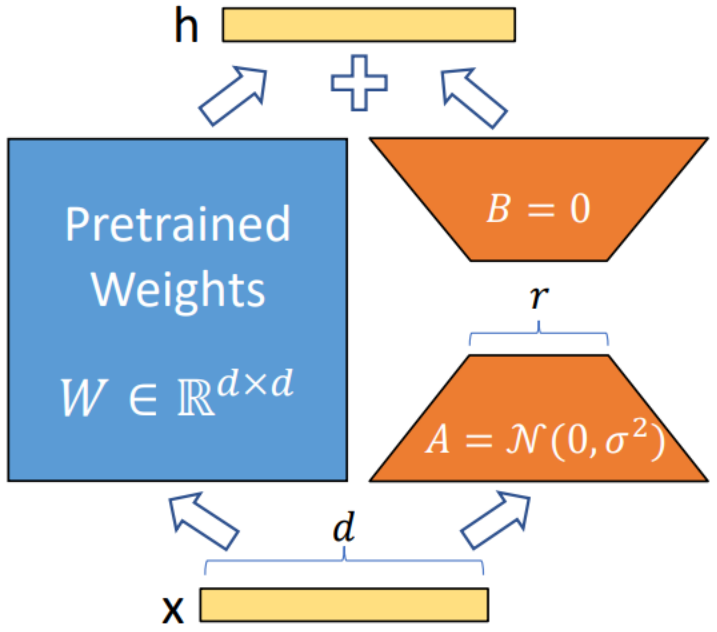


Figure 1: Our reparametrization. We only train A and B.

LoRA: Low-Rank Adaptation of Large Language Models Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen. 2021

Adapter Fine Tuning

- They add **adapter layers** in between the transformer layers of a large model.
- During fine-tuning, they **fix the original model parameters** and only tune the adapter layers.
- No need to store a full model for each task, only the adapter params.
- **3.6%** of parameters needed!

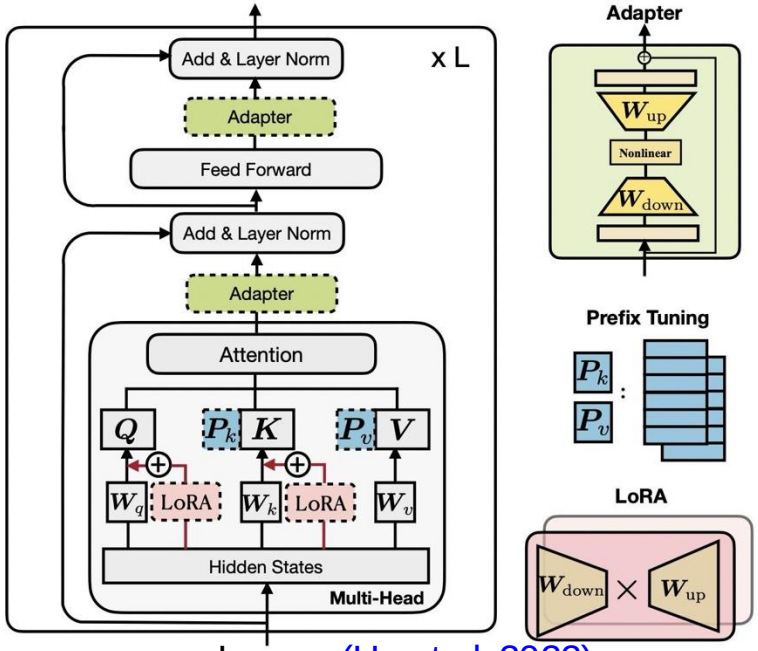


Image: (He et al. 2022)

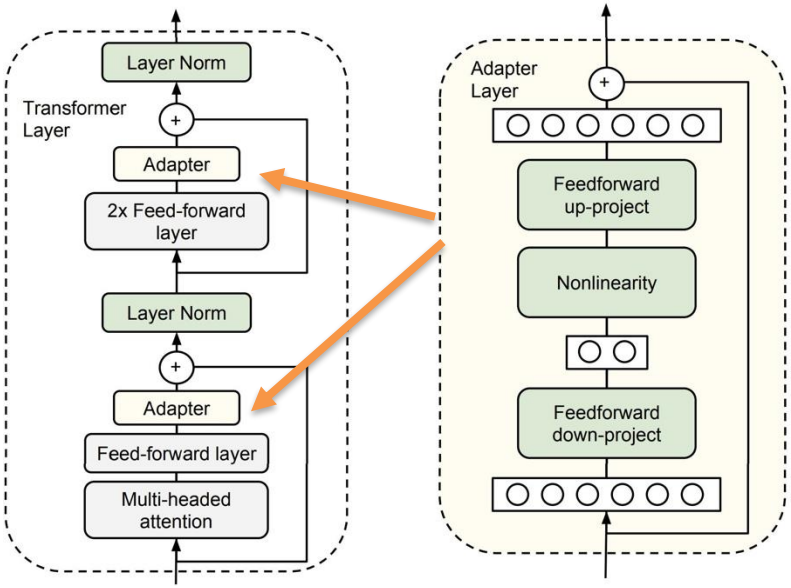


Figure 2. Architecture of the adapter module and its integration with the Transformer. **Left:** We add the adapter module twice to each Transformer layer: after the projection following multi-headed attention and after the two feed-forward layers. **Right:** The adapter consists of a bottleneck which contains few parameters relative to the attention and feedforward layers in the original model. The adapter also contains a skip-connection. During adapter tuning, the green layers are trained on the downstream data, this includes the adapter, the layer normalization parameters, and the final classification layer (not shown in the figure).

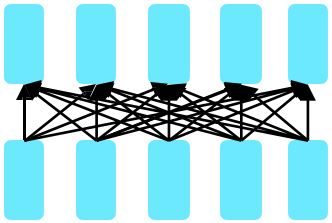
(Houlsby et al. 2019)

Outline

1. Motivating model pretraining from word embeddings
2. Model pretraining three ways
    1. Encoders
    2. Encoder-Decoders
    3. Decoders
3. The Scaling Law

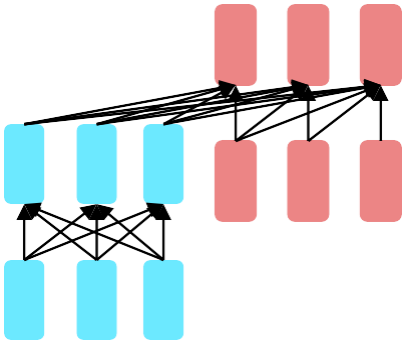
Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.



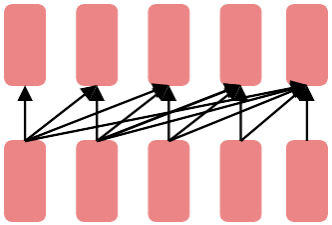
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

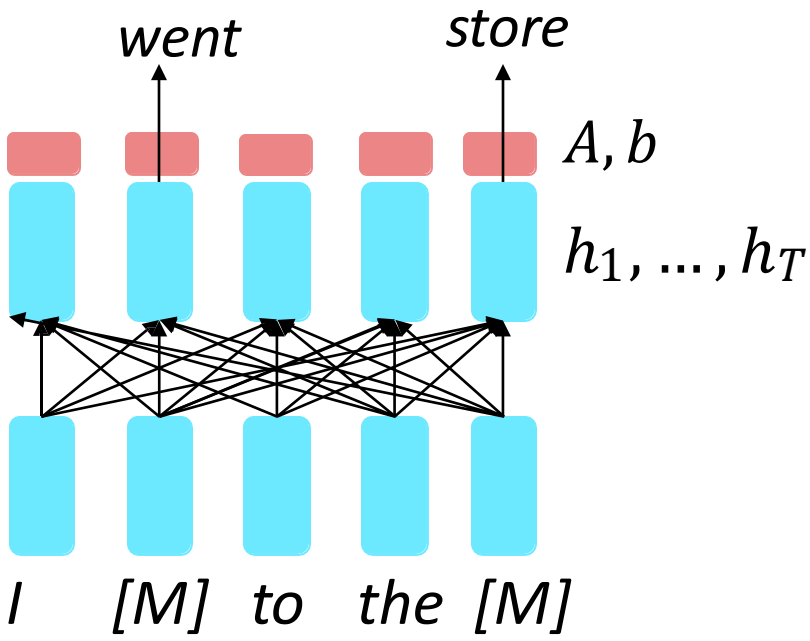


Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining encoders: what pretraining objective to use?

- So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!
- Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.
  - $h_1, \dots, h_T = \text{Encoder}\ w_1, \dots, w_T$
    - $y_i \sim Aw_i + b$
- Only add loss terms from words that are "masked out." If x' is the masked version of x, we're learning $p_\theta(x \mid x')$. Called **Masked LM**.

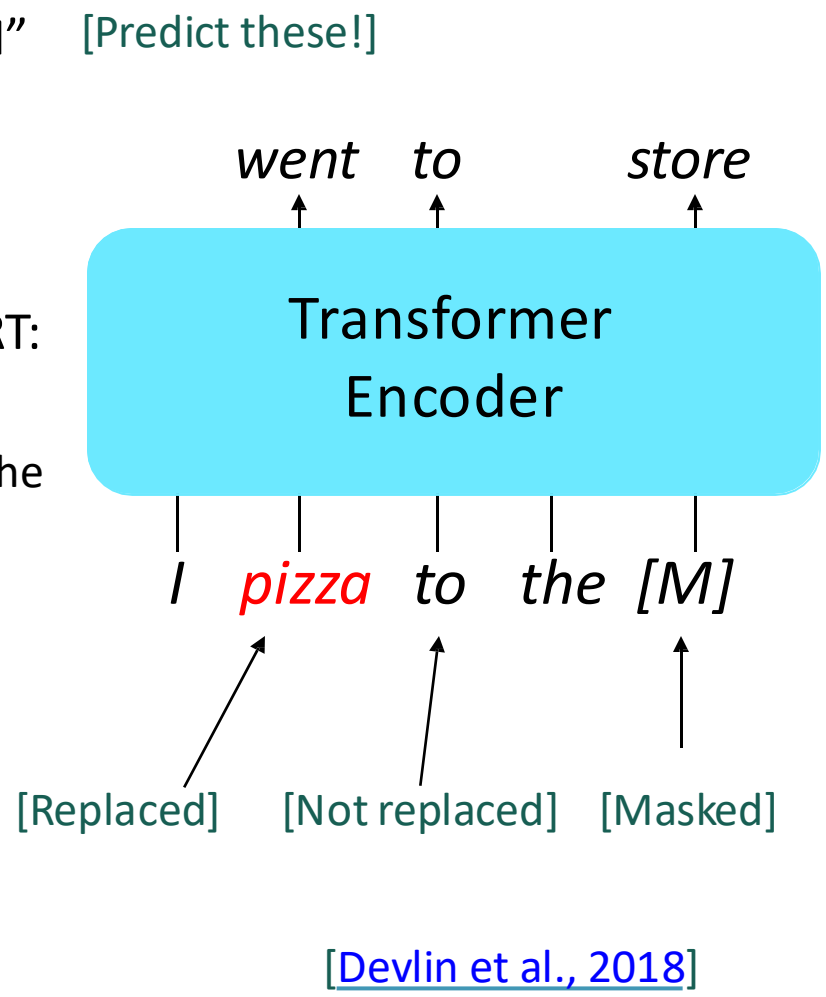


[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

- Devlin et al., 2018 proposed the "Masked LM" objective and **released the weights of a pretrained Transformer**, a model they labeled BERT.

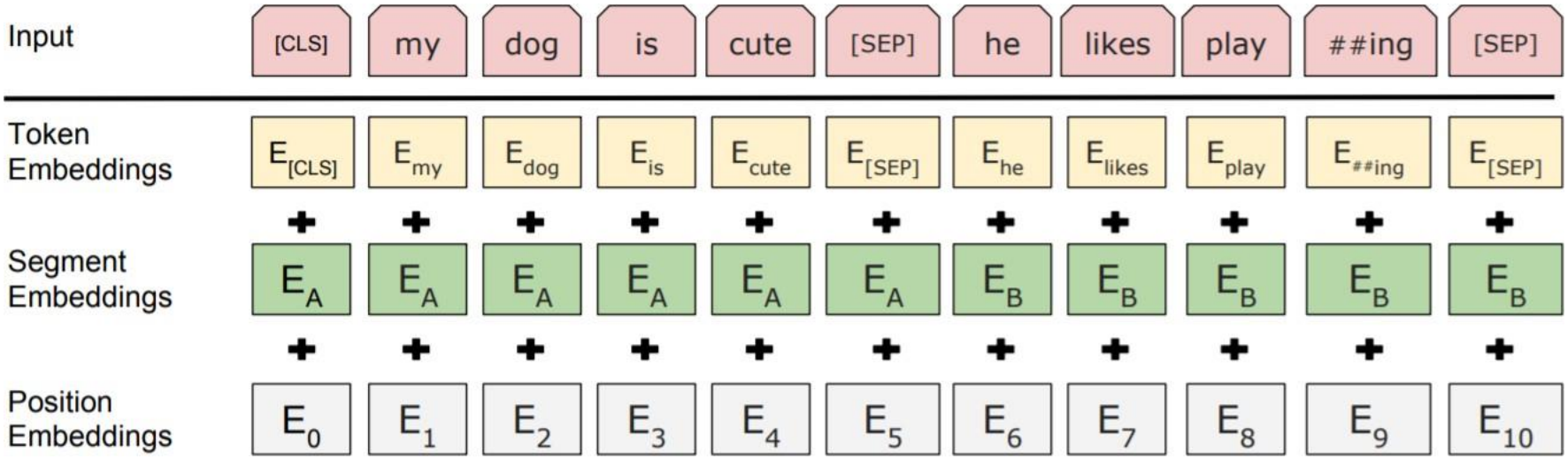
- Some more details about Masked LM for BERT:
  - Predict a random 15% of (sub)word tokens.
    - Replace input word with [MASK] 80% of the time
    - Replace input word with a random token 10% of the time
    - Leave input word unchanged 10% of the time (but still predict it!)
  - Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

- The pretraining input to BERT was two separate contiguous chunks of text:



- BERT was trained to predict whether one chunk follows the other or is randomly sampled.
  - Later work has argued this "next sentence prediction" is not necessary.

[Devlin et al., 2018, Liu et al., 2019]

BERT: Bidirectional Encoder Representations from Transformers

- Two models were released:
  - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.
- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)
- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)
- Finetuning is practical and common on a single GPU
  - “Pretrain once, finetune many times.”

[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

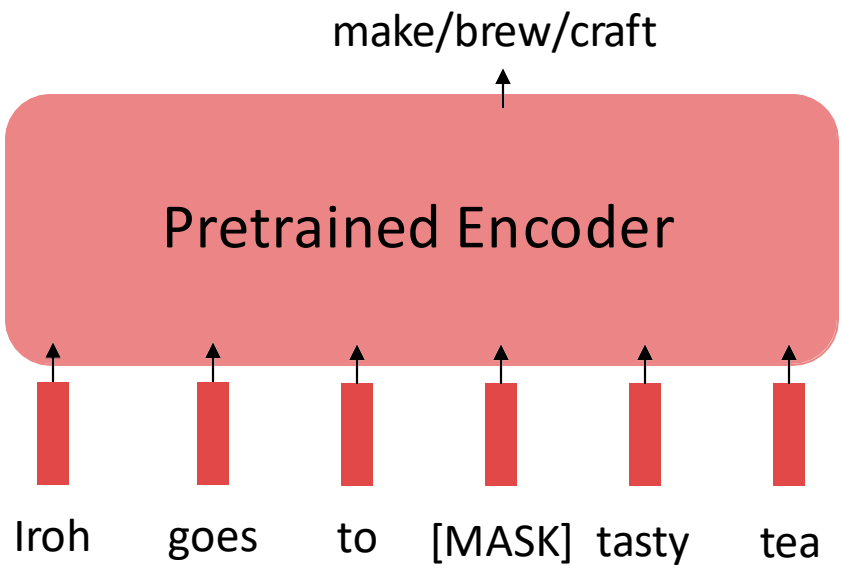
- BERT was massively popular and hugely versatile; finetuning BERT led to new state-of- the-art results on a broad range of tasks.
  - **QQP:** Quora Question Pairs (detect paraphrase questions)
  - **QNLI**: natural language inference over question
  - **SST-2**: sentiment analysis
  - **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
  - **STS-B**: semantic textual similarity
  - **MRPC**: microsoft paraphrase corpus
  - **RTE**: a small natural language inference corpus

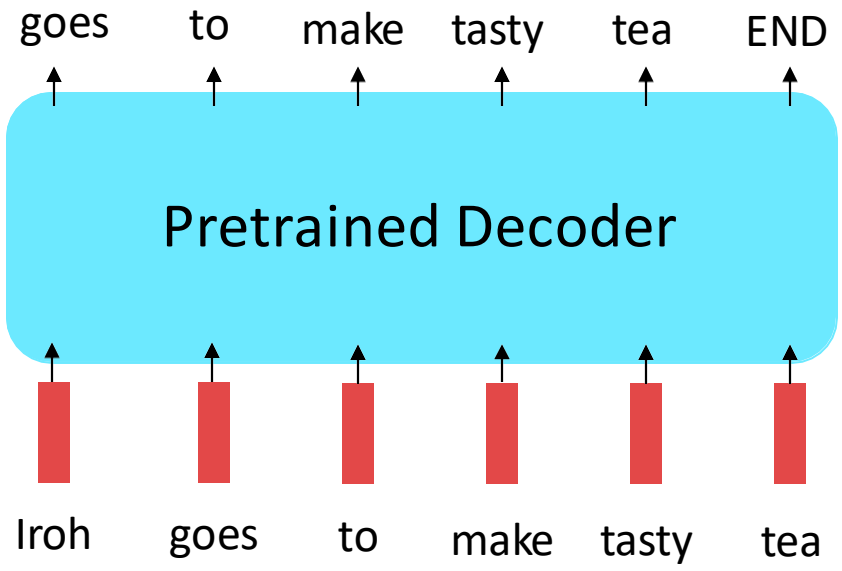
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

[Devlin et al., 2018]

Limitations of pretrained encoders

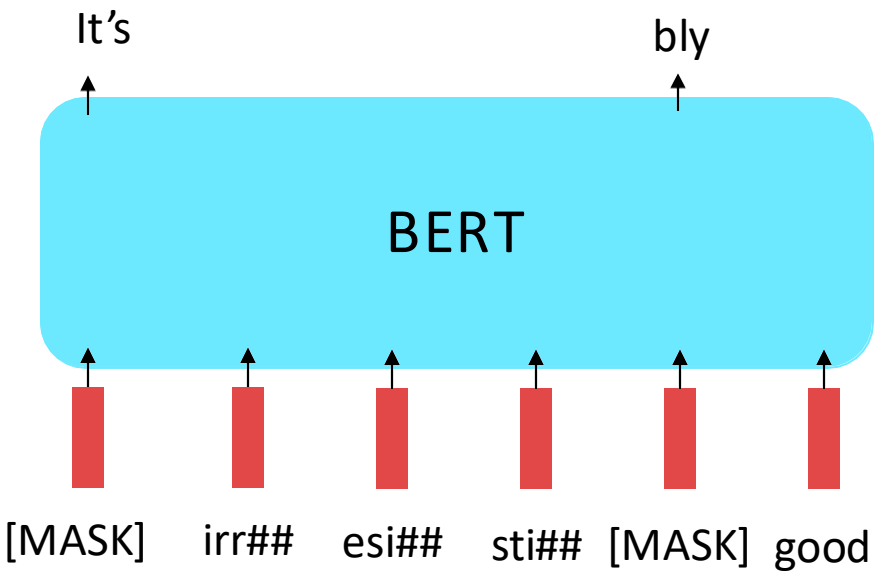
- If your task involves generating sequences, consider using a pretrained decoder; BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.

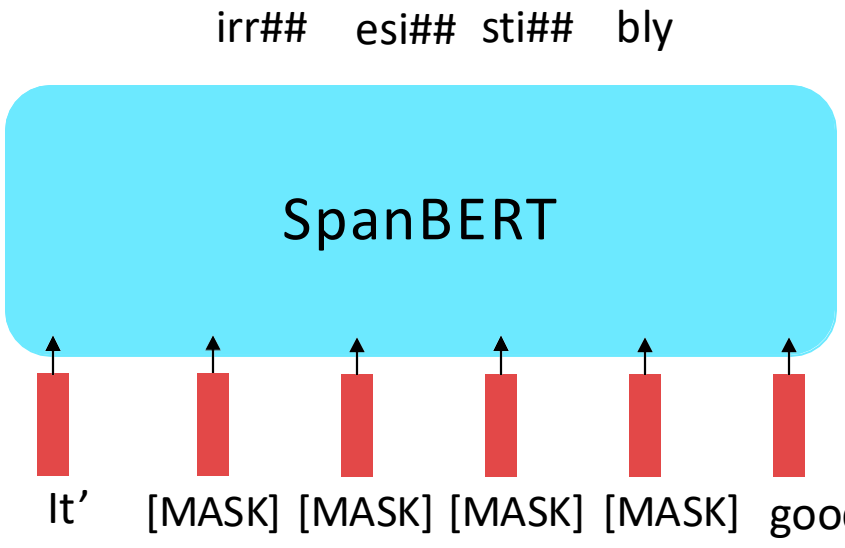




Extensions of BERT

- You'll see a lot of BERT variants like RoBERTa, SpanBERT, DeBERTa, …
- Some generally accepted improvements to the BERT pretraining formula:
  - RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
  - SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task





[Liu et al., 2019; Joshi et al., 2020]

Extensions of BERT

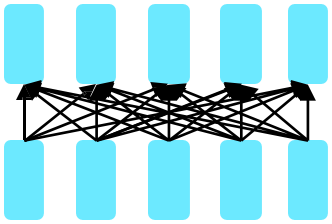
- A takeaway from the RoBERTa paper: more compute, more data can improve pretraining even when not changing the underlying Transformer encoder.

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

[Liu et al., 2019; Joshi et al., 2020]

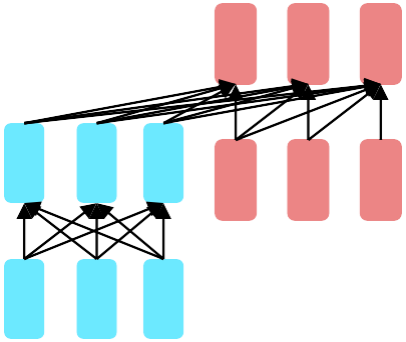
Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.



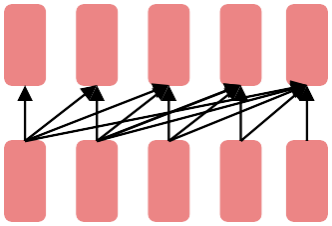
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

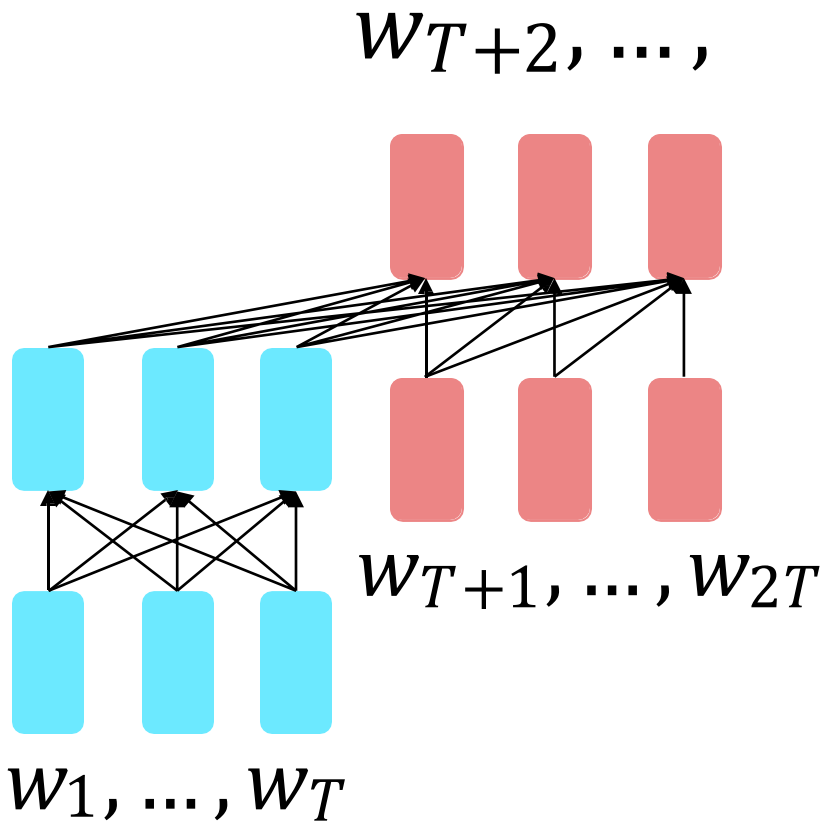
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining encoder-decoders: what pretraining objective to use?

- For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$
$$h_{T+1}, \ldots, h_2 = Decoder(w_1, \ldots, w_T, h_1, \ldots, h_T)$$
$$y_i \sim Ah_i + b, i > T$$

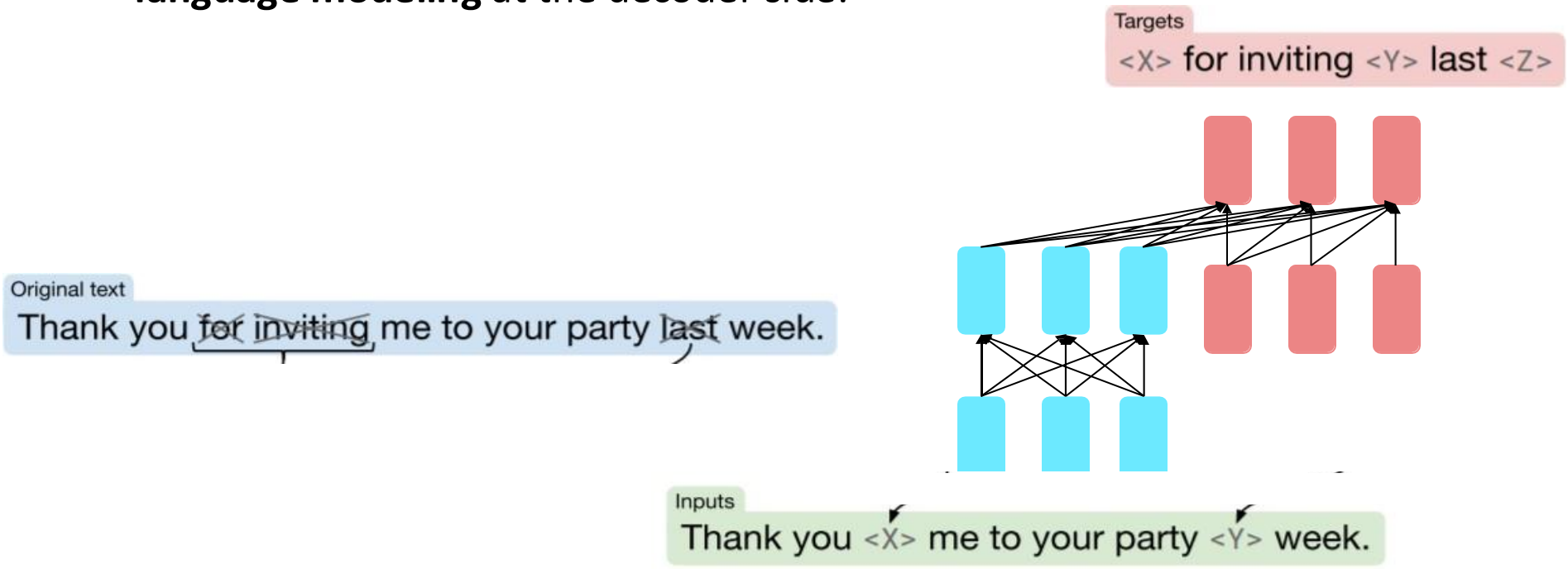
- The **encoder** portion benefits from bidirectional context; the **decoder** portion is used to train the whole model through language modeling.



[Raffel et al., 2018]

Pretraining encoder-decoders: what pretraining objective to use?

- What Raffel et al., 2018 found to work best was **span corruption.** Their model: **T5**.
  - Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!
  - This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



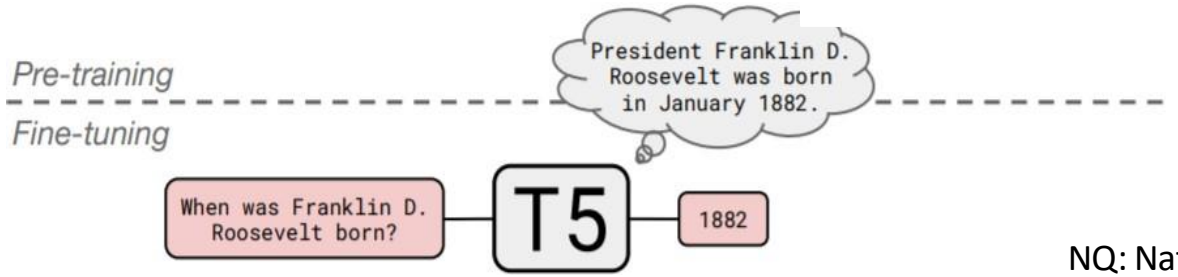
Pretraining encoder-decoders: what pretraining objective to use?

- Raffel et al., 2018 found encoder-decoders to work better than decoders for their tasks, and span corruption (denoising) to work better than language modeling.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Pretraining encoder-decoders: what pretraining objective to use?

- A fascinating property of T5: it can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.



	NQ	WQ	TQA dev	TQA test
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–
T5.1.1-Base	25.7	28.2	24.2	30.6
T5.1.1-Large	27.3	29.5	28.5	37.2
T5.1.1-XL	29.5	32.4	36.0	45.1
T5.1.1-XXL	32.8	35.6	42.9	52.5
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**

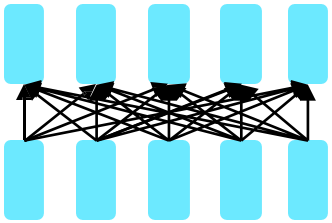
NQ: Natural Questions
WQ: WebQuestions
TQA: Trivia QA
All "open-domain" versions

220 million params
770 million params
3 billion params
11 billion params

[Raffel et al., 2018]

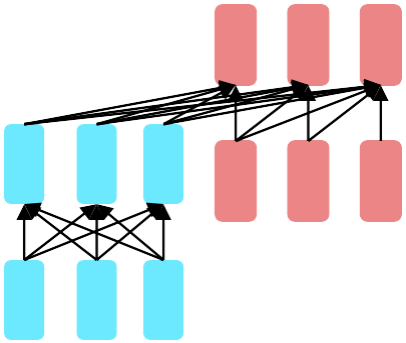
Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.



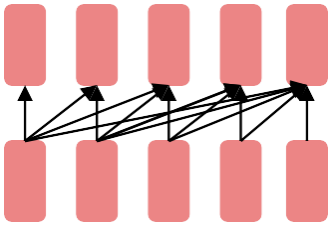
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

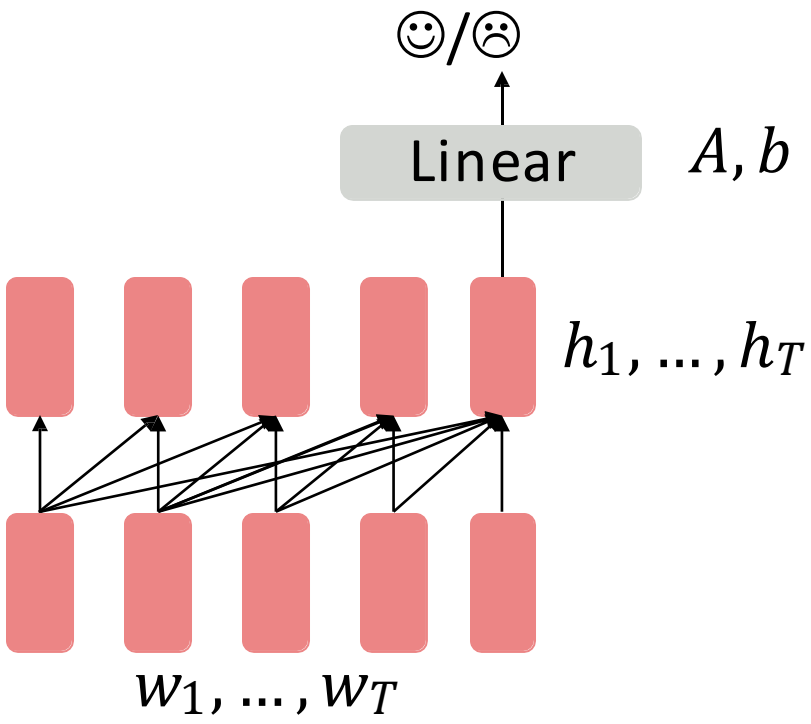


Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining decoders

- When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t | w_{1:t-1})$.
- We can finetune them by training a classifier on the last word's hidden state.
  - $h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$
    - $y \sim Ah_T + b$
- Where A and b are randomly initialized and specified by the downstream task.

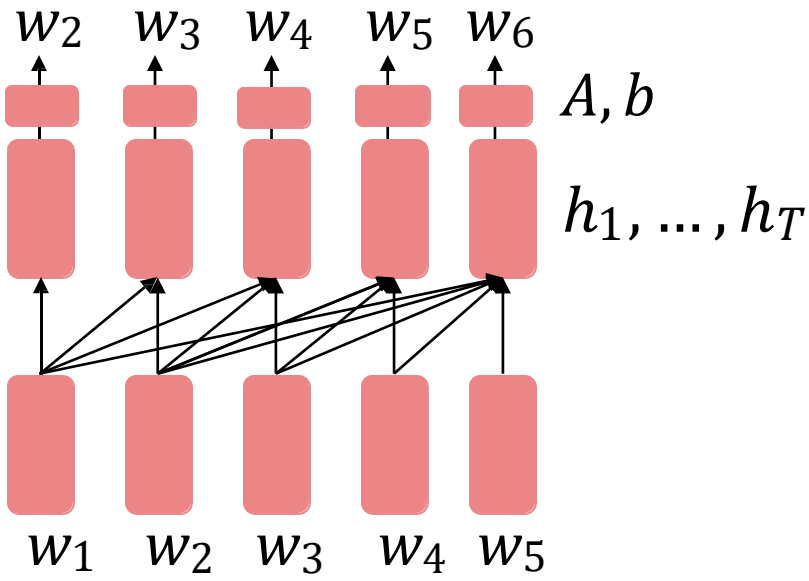


[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Pretraining decoders

- It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t | w_{1:t-1})$

- This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!
  - $h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$
    - $y \sim Ah_T + b$
      - Where A, b were pretrained in the language model!
  - Dialogue (context=dialogue history)
  - Summarization (context=document)



[Note how the linear layer has been pretrained.]

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

- 2018's GPT was a big success in pretraining a decoder!
- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

- How do we format inputs to our decoder for **finetuning tasks?**
- **Natural Language Inference:** Label pairs of sentences as
  - *entailing/contradictory/neutral*

Premise: *The man is in the doorway*
Hypothesis: *The person is near the door*
} **entailment**

- Radford et al., 2018 evaluate on natural language inference. Here's roughly how the input was formatted, as a sequence of tokens for the decoder.
  - [START] *The man is in the doorway* [DELIM] *The person is near the door* [EXTRACT]
  - The linear classifier is applied to the representation of the [EXTRACT] token.

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

- GPT results on various *natural language inference* datasets.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

GPT-3, In-context learning, and very large models

- So far, we've interacted with pretrained models in two ways:
  - Sample from the distributions they define (maybe providing a prompt)
  - Fine-tune them on a task we care about, and take their predictions.

- Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

- GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters.
  - **GPT-3 has 175 billion parameters.**

GPT-3, In-context learning, and very large models

- Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.
- The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.
- **Input (prefix within a single Transformer decoder context):**

  thanks -> merci
  hello -> bonjour
  mint -> menthe
  otter ->

- **Output (conditional generations):**

  loutre…

Scaling Efficiency: how do we best use our compute

- GPT-3 was **175B parameters** and trained on **300B** tokens of text.
- Roughly, the cost of training a large transformer scales as **parameters*tokens**
- Did OpenAI strike the right parameter-token data to get the best model? No.

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

This 70B parameter model is better than the much larger other models!

The DeepSeek V3!

Training Costs	Pre-Training	Context Extension	Post-Training	Total
in H800 GPU Hours	2664K	119K	5K	2788K
in USD	$5.328M	$0.238M	$0.01M	$5.576M

Table 1 | Training costs of DeepSeek-V3, assuming the rental price of H800 is $2 per GPU hour.

Metrics Comparison

Metric	DeepSeek V3	Llama 3.1
Parameters	671B total (37B active per token)	405B
GPU Type	NVIDIA H800	NVIDIA H100
GPU Count	2,048	Up to 16,000
Training Duration	~2 months	~2.6 months (estimated)
Tokens Processed	14.8T	15.6T
GPU Hours	2.788M	~30.8M
Training Cost	~$5.6M	~$92.4M$–123.2M (estimated)
Cost per Trillion Tokens	~$378K	~$5.93M$–7.90M

Note: Cost estimations uses an average of **$2/hour** for H800 GPUs (DeepSeek V3) and **$3/hour** for H100 GPUs (Llama 3.1) based on rental GPU prices.

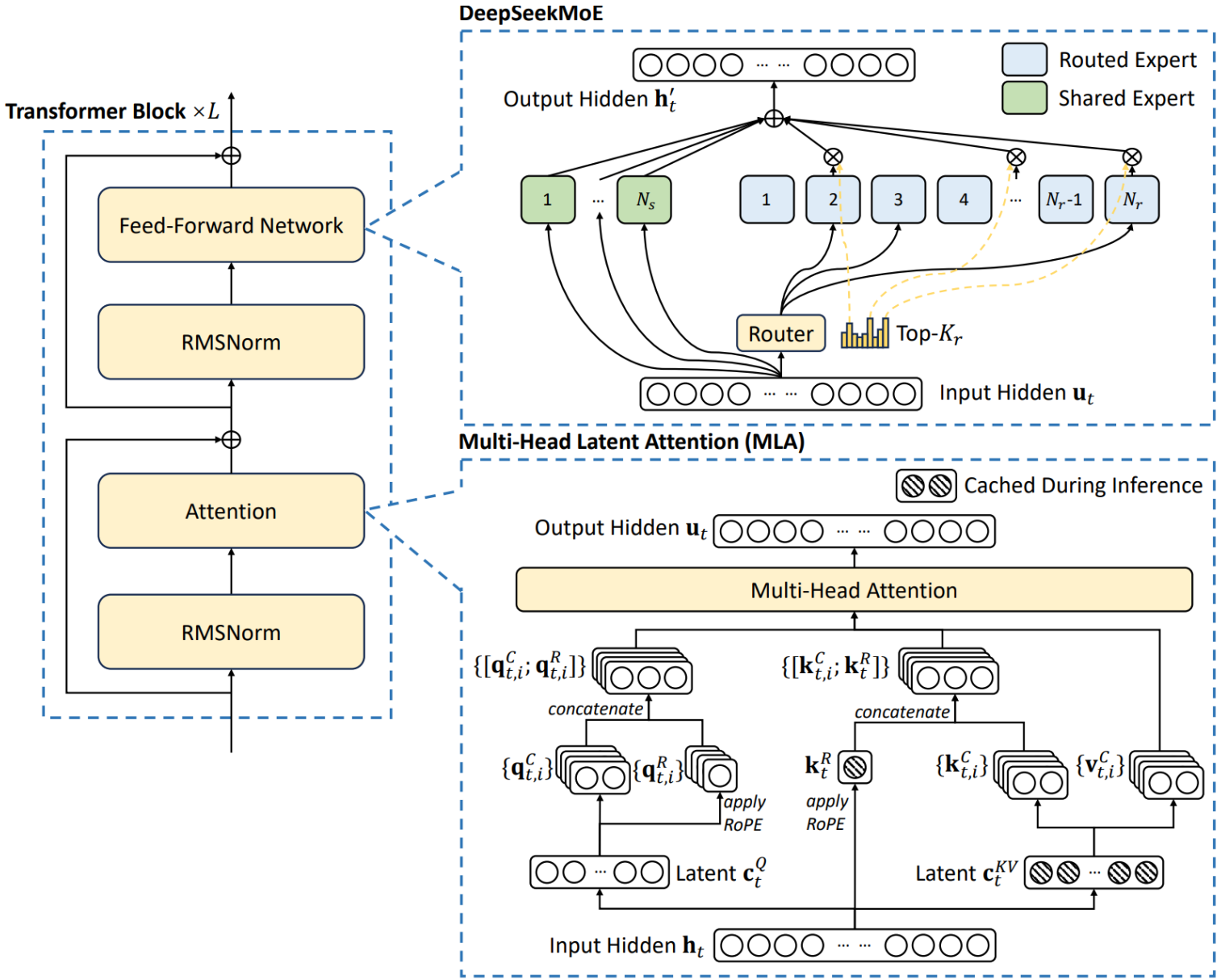
https://arxiv.org/pdf/2412.19437v1
https://apxml.com/posts/training-cost-deepseek-v3-vs-llama-3

Rome was not built in a day, DeepSeek-R1 wasn't either

- What constitute R1?
- Feb of 2024, release of Deepseek Math: Introduce GRPO
- May of 2024, release of Deepseek V2: Introduce **MoE**, **MLA**
- Dec of 2024, release of Deepseek V3: Introduce FP8, **MTP**

All the above techniques constitute the foundation of R1, a robust DeepSeek V3 model.

DeepSeek-V3

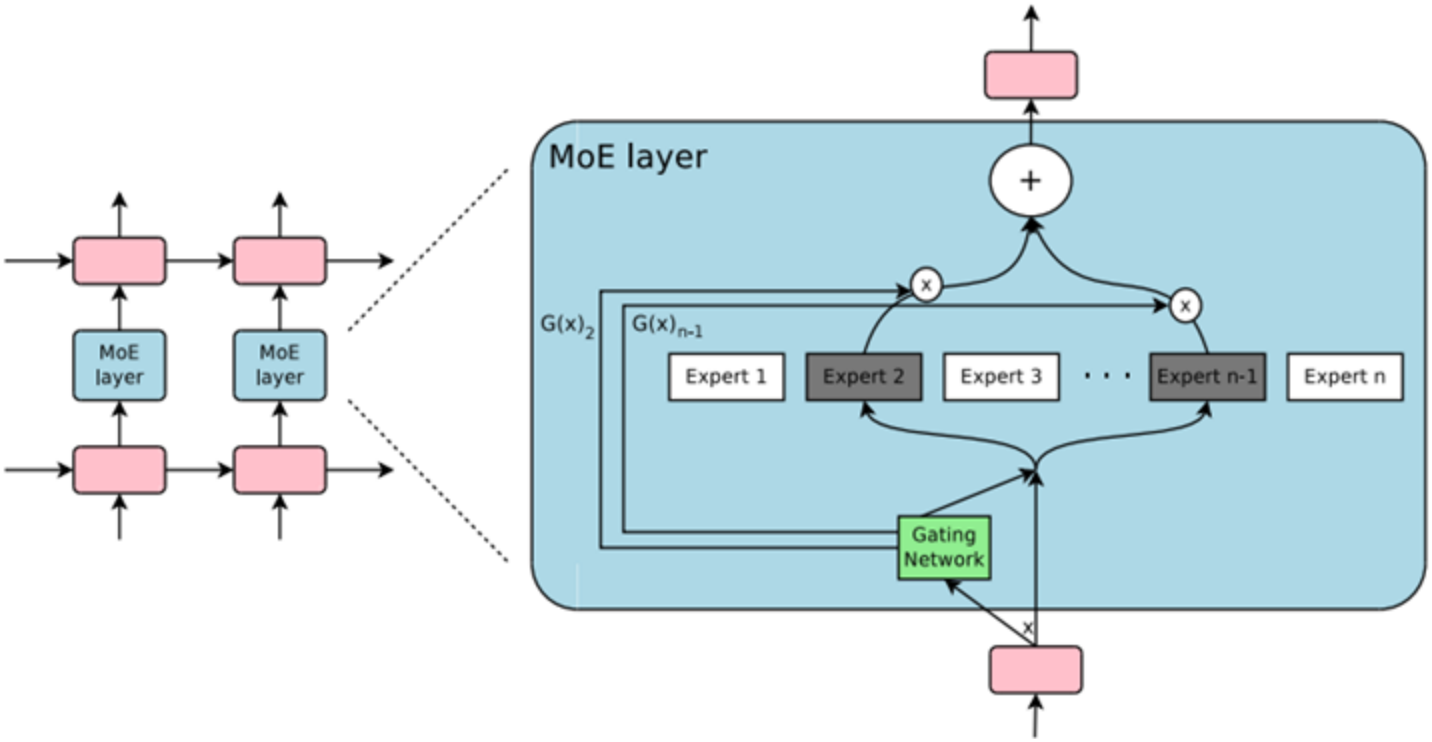


DeepSeek-AI et al., "DeepSeek-V3 Technical Report," Dec. 27, 2024, arXiv: arXiv:2412.19437. doi: 10.48550/arXiv.2412.19437.

Mixture of Expert (MOE)

Challenges with Previous MoE Models:

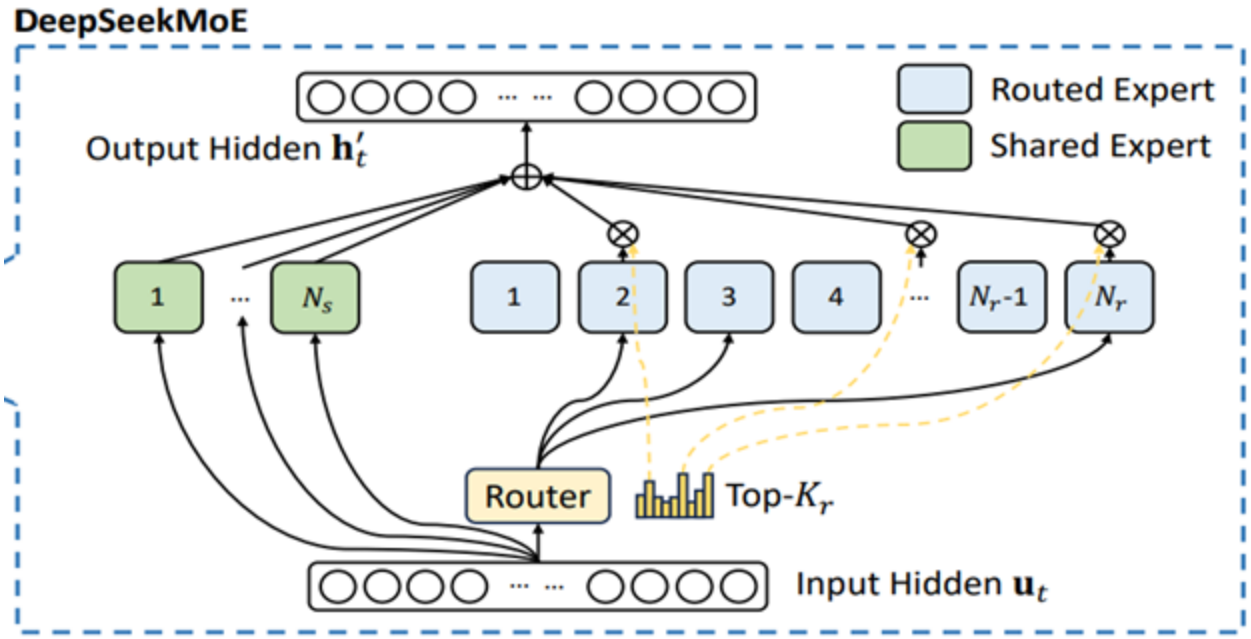
- Training large models is complex and costly.
- Inference and optimization are expensive.
- Earlier models, like Mistral 8x7B, had limited effectiveness due to fewer and smaller experts.



Mixture of Expert (MOE)

DeepSeekMoE

- Large Scale Expert Activation, V2: activated 21B out of 236B, V3: activated 37B out of 671B
- Fine-grained Expert Segmentation
  - (More experts (256+1 shared), less No. of Neuron)
- Use of shared Experts
  - Reduces Knowledge Redundancy among experts



DeepSeek-AI et al., "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model," Jun. 19, 2024, arXiv: arXiv:2405.04434. doi: 10.48550/arXiv.2405.04434.

Mixture of Expert (MOE)

$$\mathbf{h}_t' = \mathbf{u}_t + \sum_{i=1}^{N_s} \mathrm{FFN}_i^{(s)}(\mathbf{u}_t) + \sum_{i=1}^{N_r} g_{i,t}\,\mathrm{FFN}_i^{(r)}(\mathbf{u}_t),$$

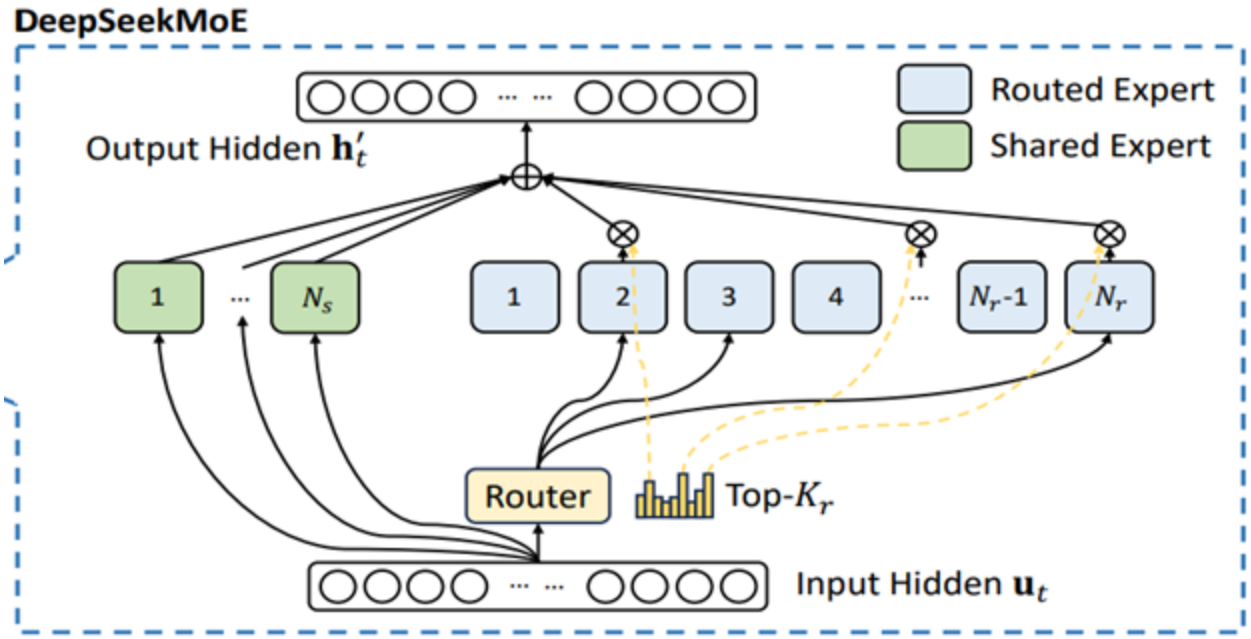
$$g_{i,t} = \frac{g'_{i,t}}{\sum_{j=1}^{N_r} g'_{j,t}},$$

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \mathrm{Topk}(\{s_{j,t} | 1 \leqslant j \leqslant N_r\}, K_r), \\ 0, & \text{otherwise}, \end{cases}$$

$$s_{i,t} = \mathrm{Sigmoid}\left(\mathbf{u}_t^T \mathbf{e}_i\right),$$

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} + b_i \in \mathrm{Topk}(\{s_{j,t} + b_j | 1 \leqslant j \leqslant N_r\}, K_r), \\ 0, & \text{otherwise}. \end{cases}$$

- A bias term b_i for each expert. $b_i \mp \lambda$ if overloaded or underloaded each step
- The gating value, which will be multiplied with the FFN output, is still derived from the original affinity score $s_{i,t}$.



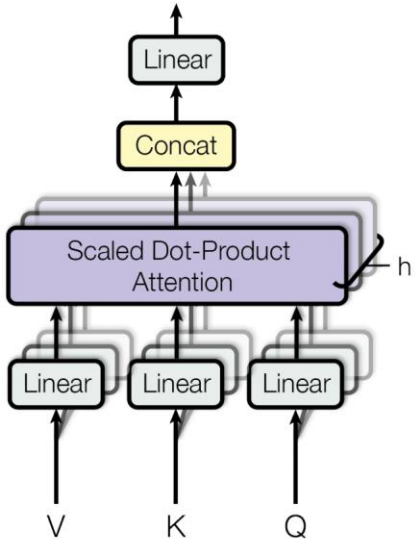
DeepSeek-AI et al., "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model," Jun. 19, 2024, arXiv: arXiv:2405.04434. doi: 10.48550/arXiv.2405.04434.

Recall: Multi-head attention

- The input word vectors could be the queries, keys and values
  - In other words: the word vectors themselves select each other
  - Word vector stack = Q = K = V
- Problem: Only one way for words to interact with one-another
- Solution: Multi-head attention
  - First map Q, K, V into h many lower dimensional spaces via W matrices
  - Then apply attention, then concatenate outputs and pipe through linear layer

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\textbf{where } \text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Multi-Head Latent Attention (MLA)

- Integrates LoRA to reduce memory overhead while retaining context.
- Smaller KV cache, Reasoning with more Context
- Memory Efficiency without performance degrade

$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}}\right)\mathbf{v}_{j,i}^C,$$

$$\mathbf{u}_t = W^O[\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; ...; \mathbf{o}_{t,n_h}],$$

$$\boxed{\mathbf{c}_t^{KV}} = W^{DKV}\mathbf{h}_t,$$

$$[\mathbf{k}_{t,1}^C; \mathbf{k}_{t,2}^C; ...; \mathbf{k}_{t,n_h}^C] = \mathbf{k}_t^C = W^{UK}\mathbf{c}_t^{KV},$$

$$\boxed{\mathbf{k}_t^R} = \mathrm{RoPE}(W^{KR}\mathbf{h}_t),$$

$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R],$$

$$[\mathbf{v}_{t,1}^C; \mathbf{v}_{t,2}^C; ...; \mathbf{v}_{t,n_h}^C] = \mathbf{v}_t^C = W^{UV}\mathbf{c}_t^{KV},$$

$$d_c(\ll d_h n_h)$$

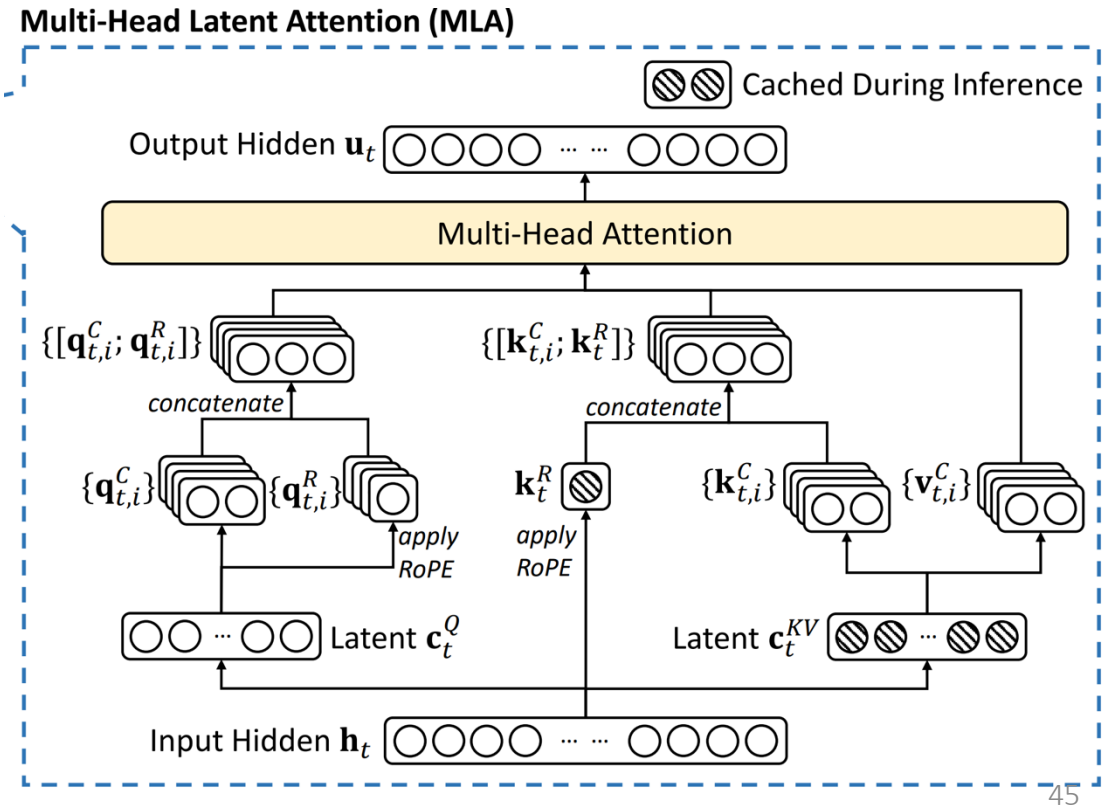
$$\mathbf{c}_t^Q = W^{DQ}\mathbf{h}_t,$$

$$[\mathbf{q}_{t,1}^C; \mathbf{q}_{t,2}^C; ...; \mathbf{q}_{t,n_h}^C] = \mathbf{q}_t^C = W^{UQ}\mathbf{c}_t^Q,$$

$$[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; ...; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R = \mathrm{RoPE}(W^{QR}\mathbf{c}_t^Q),$$

$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R],$$

$$d_c'(\ll d_h n_h)$$

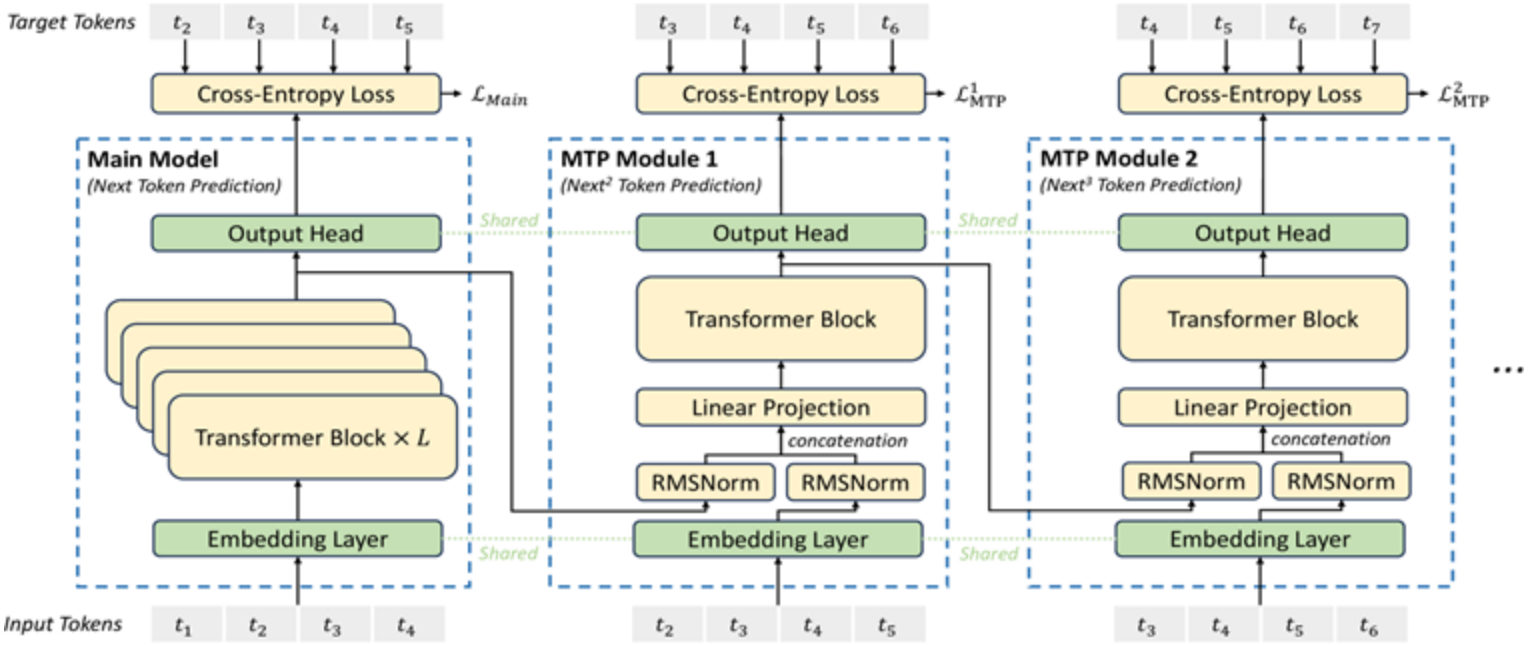


RoPE: Rotary Positional Embedding

DeepSeek-AI et al., "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model," Jun. 19, 2024, arXiv: arXiv:2405.04434. doi: 10.48550/arXiv.2405.04434.

Multi-Token Prediction (MTP)

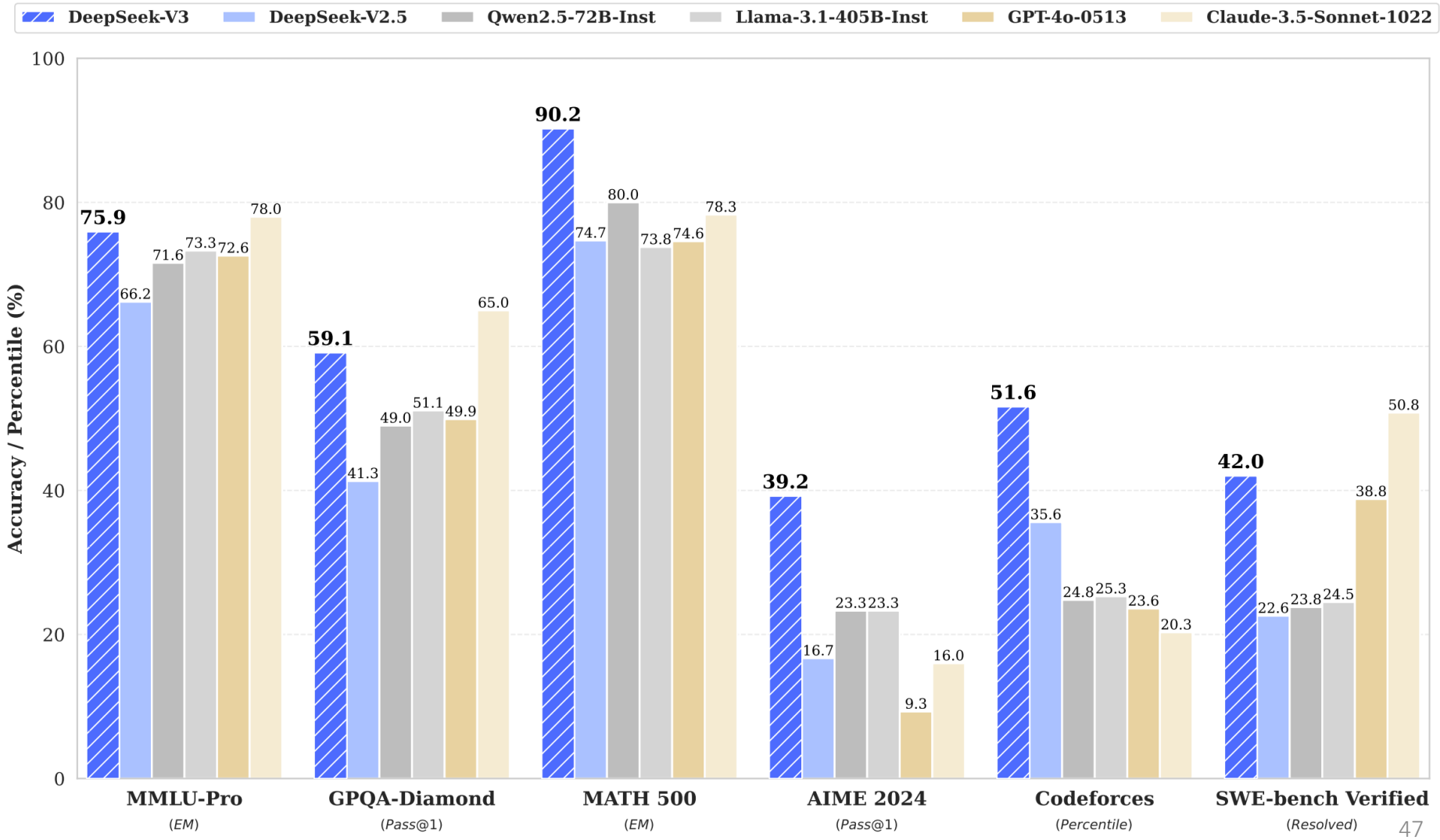
- Transformer: Next Token Prediction
  - Predict one token at a time sequentially.
  - Greedy strategy: Always pick the token with the highest probability for the next step.
- Improve Reasoning Efficiency
  - By predicting more than one token
  - Joint optimization: Considers future context while predicting the current token
- Planning-ahead: Models learn to "think ahead," avoiding dead ends.



The multi-token prediction depth D is set to 1

DeepSeek-AI et al., "DeepSeek-V3 Technical Report," Dec. 27, 2024, arXiv: arXiv:2412.19437. doi: 10.48550/arXiv.2412.19437.

Main Results



Other System Level Optimization

- Training Framework
  - DualPipe and Computation-Communication Overlap
  - Efficient Implementation of Cross-Node All-to-All
  - Communication Extremely Memory Saving with Minimal Overhead
- FP8 Training
  - Mixed Precision Framework
  - Improved Precision from Quantization and Multiplication
  - Low-Precision Storage and Communication
- Inference and Deployment
  - Prefilling
  - Decoding
- Suggestions on Hardware Design
  - Communication Hardware
  - Compute Hardware

3.2.2. *Efficient Implementation of Cross-Node All-to-All Communication*

In order to ensure sufficient computational performance for DualPipe, we customize efficient cross-node all-to-all communication kernels (including dispatching and combining) to conserve the number of SMs dedicated to communication. The implementation of the kernels is co-designed with the MoE gating algorithm and the network topology of our cluster. To be specific, in our cluster, cross-node GPUs are fully interconnected with IB, and intra-node communications are handled via NVLink. NVLink offers a bandwidth of 160 GB/s, roughly 3.2 times that of IB (50 GB/s). To effectively leverage the different bandwidths of IB and NVLink, we limit each token to be dispatched to at most 4 nodes, thereby reducing IB traffic. For each token, when its routing decision is made, it will first be transmitted via IB to the GPUs with the same in-node