COMP 3361 Natural Language Processing

Lecture 14: Pre-training and large language models (cont.)

Spring 2025

Announcements

- Assignment 2 is due next Tuesday
  - Join #assignment-2 Slack channel for discussion
- TA will give a coding tutorial on Transformers and assignment 2 this Friday

Lecture plan

- Neural language models: recap
- Traditional to modern NLP
  - Traditional learning paradigm
    - Supervised training/fine-tuning only, NO pre-training
  - Modern learning paradigm
    - Pretrain + fine-tuning, pretrain + prompting/in-context learning
- Pretraining overview: BERT, T5, GPT

Traditional to modern NLP: training paradigm

N-gram language models	→	Neural language models: BERT, GPT
Traditional models: Naive Bayes	→	Neural models: Transformers
Static embeddings: word2vec	→	Contextual embeddings: BERT, GPT
Traditional learning paradigm	→	New learning paradigm: Pretrain, ICL

Question: How to train and use neural language models for different NLP tasks?

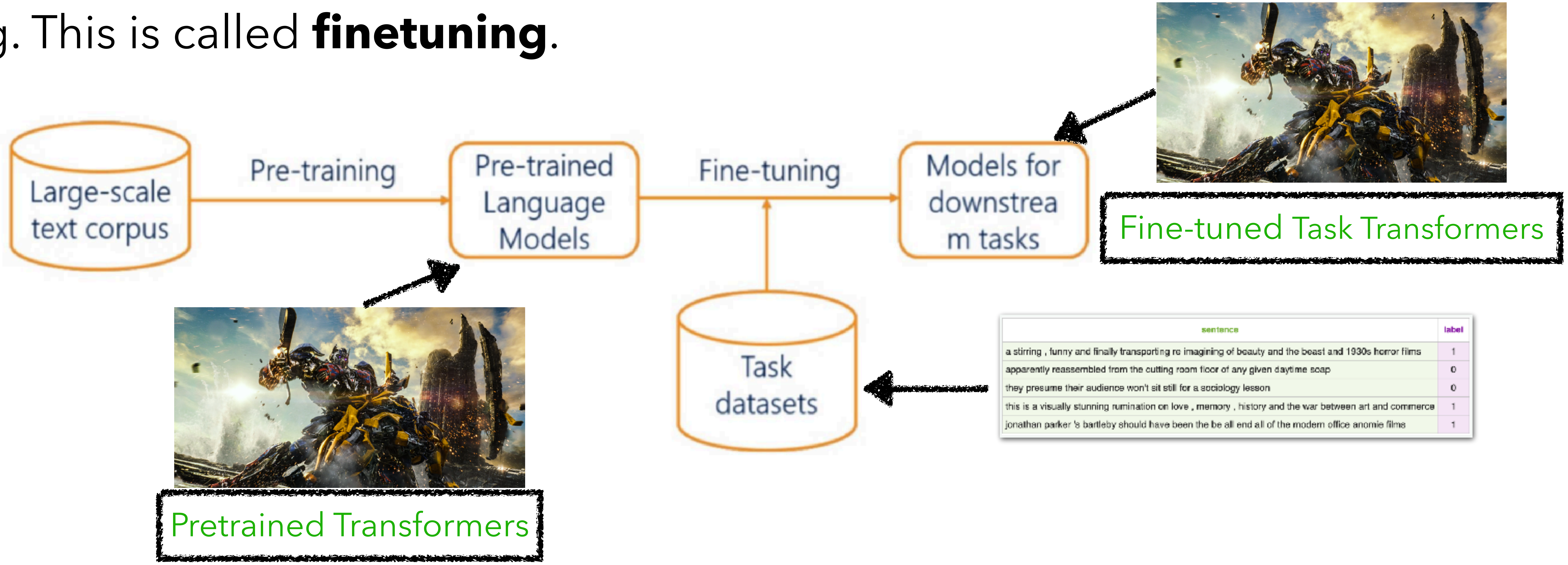
Traditional learning paradigm

- **Supervised training/fine-tuning only, NO pre-training**
  - Collect (x, y) task training pairs
  - Randomly initialize your models f(x) (e.g., vanilla Transformers)
  - Train f(x) on (x, y) pairs

Then you get a trained Transformers **ONLY** for sentiment analysis
The model can be: NB, LR, RNNs, LSTM too

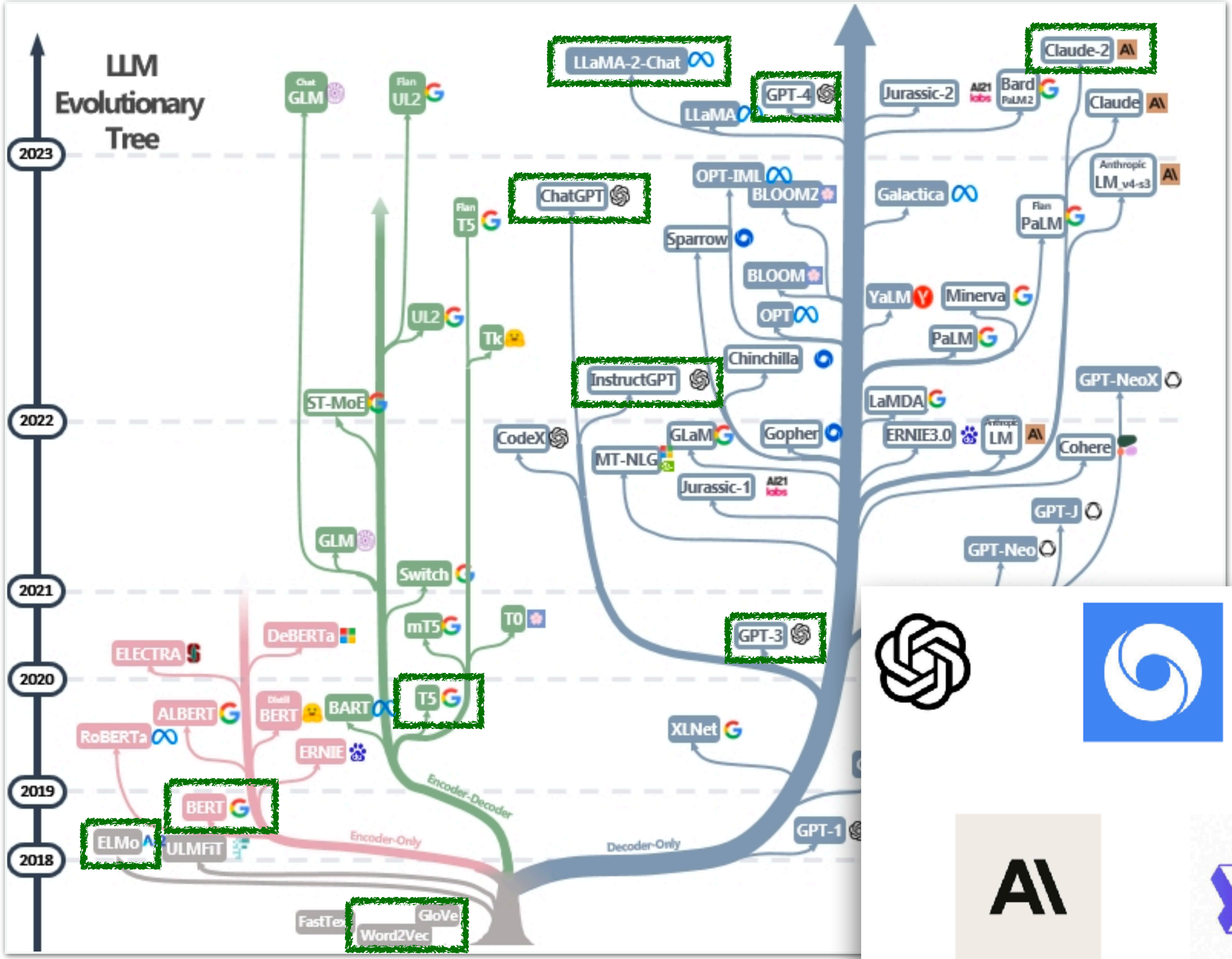
Modern learning paradigm

- **Pre-training + supervised training/fine-tuning**
  - First train Transformer using a lot of general text using unsupervised learning. This is called **pretraining**.
  - Then train the pretrained Transformer for a specific task using supervised learning. This is called **finetuning**.



sentence	label
a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films	1
apparently reassembled from the cutting room floor of any given daytime soap	0
they presume their audience won't sit still for a sociology lesson	0
this is a visually stunning rumination on love , memory , history and the war between art and commerce	1
jonathan parker 's bartleby should have been the be all end all of the modern office anomie films	1

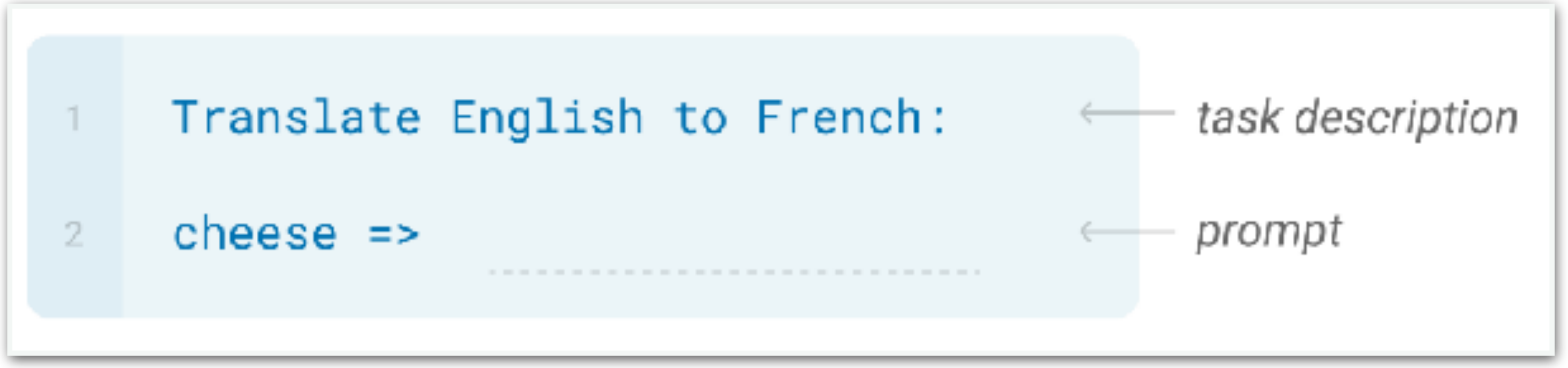
Evolution tree of pretrained LMs



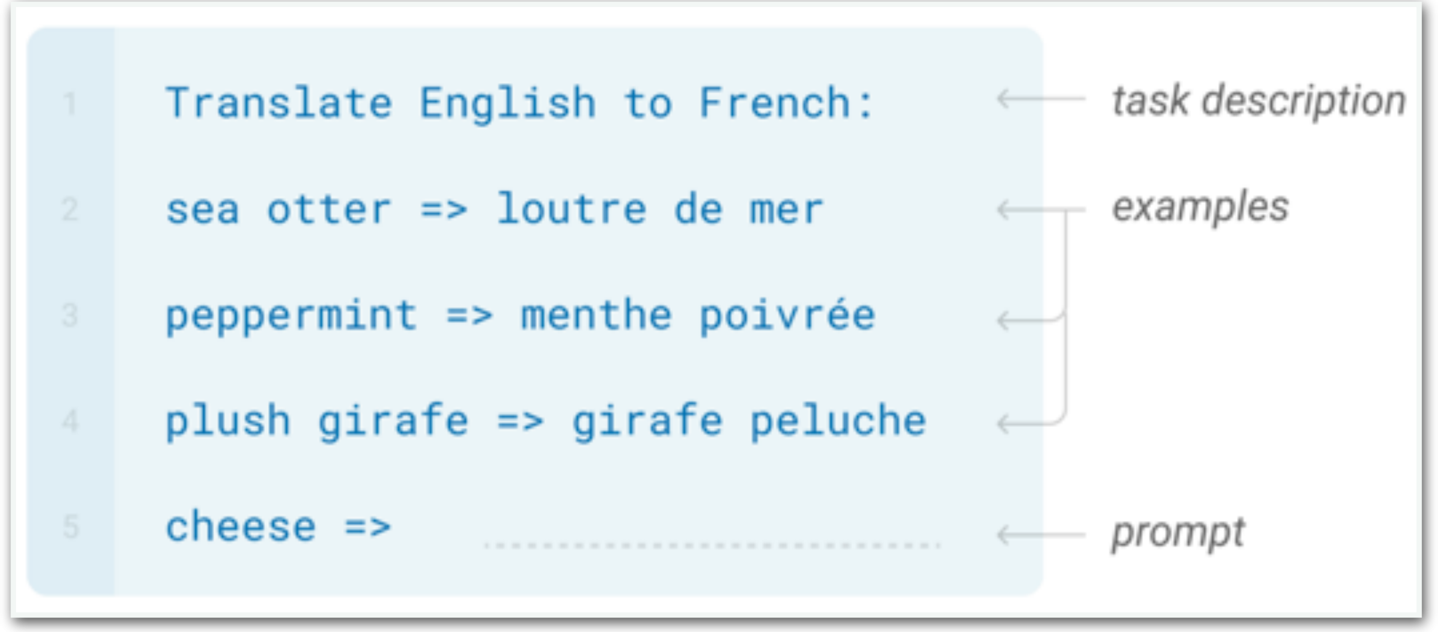
https://github.com/Mooler0410/LLMsPracticalGuide
https://mistral.ai/news/mistral-large/

Latest learning paradigm with LLMs

- **Pre-training + prompting/in-context learning (no training this step)**
  - First train a **large (>7~175B)** Transformer using a lot of general text using unsupervised learning. This is called **large** language model **pretraining**.
  - Then **directly use** the pretrained large Transformer (**no further finetuning/training**) for any different task given only a natural language description of the task or a few task (x, y) examples. This is called **prompting/in-context learning**.



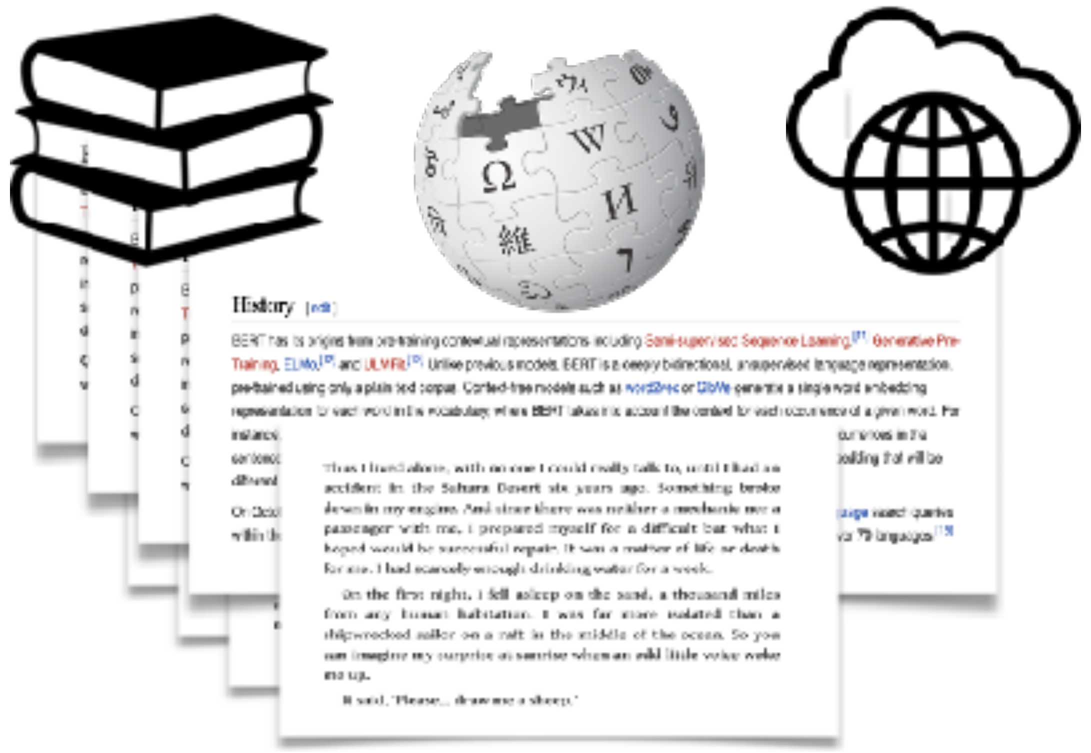
Zero-shot prompting



Few-shot prompting/in-context learning

Pretraining: training objectives?

- During pretraining, we have a large text corpus (**no task labels**)
  - **Key question: what labels or objectives used to train the vanilla Transformers?**

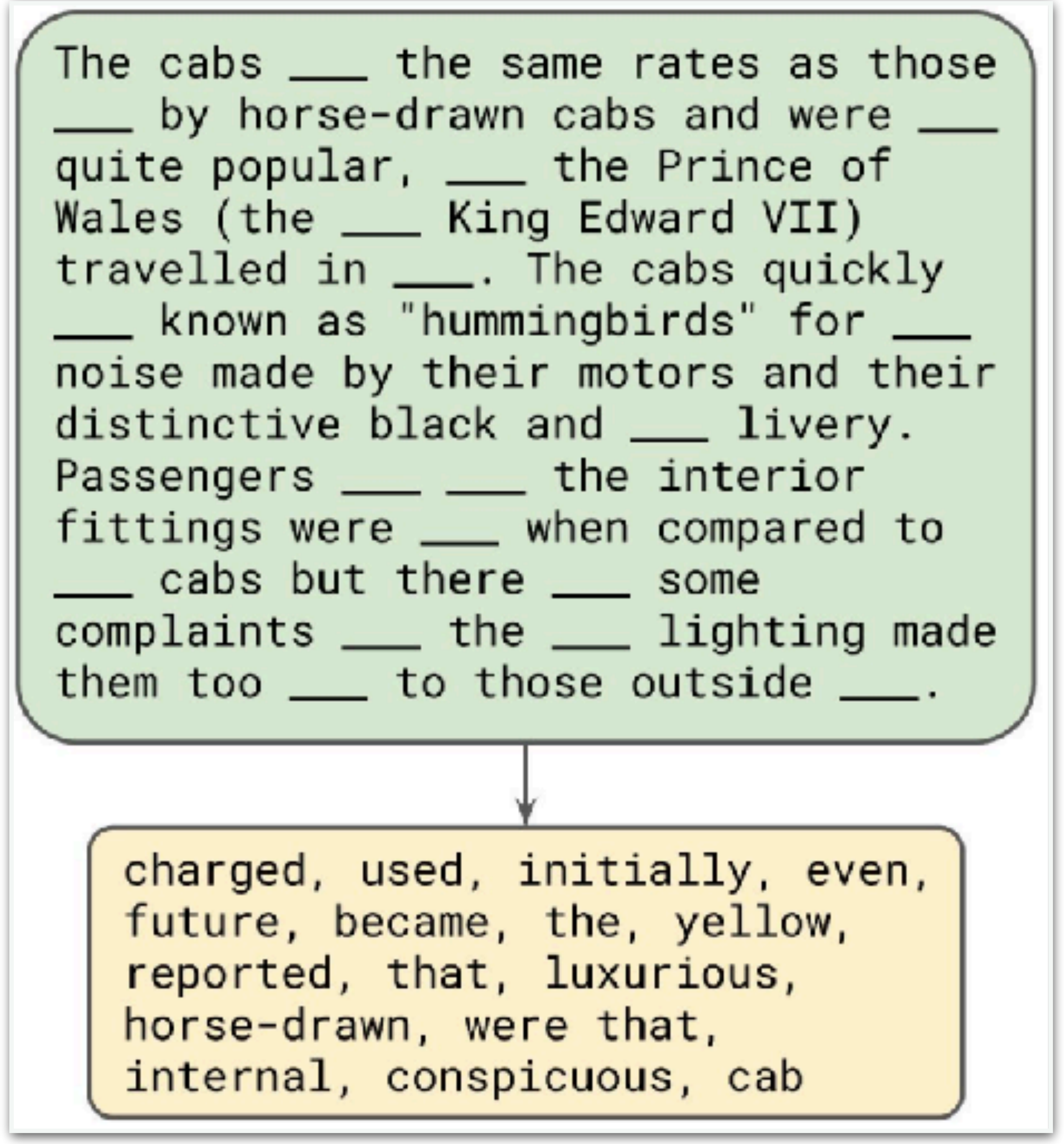


Training labels/objectives?

Pretraining: training objectives?

BERT

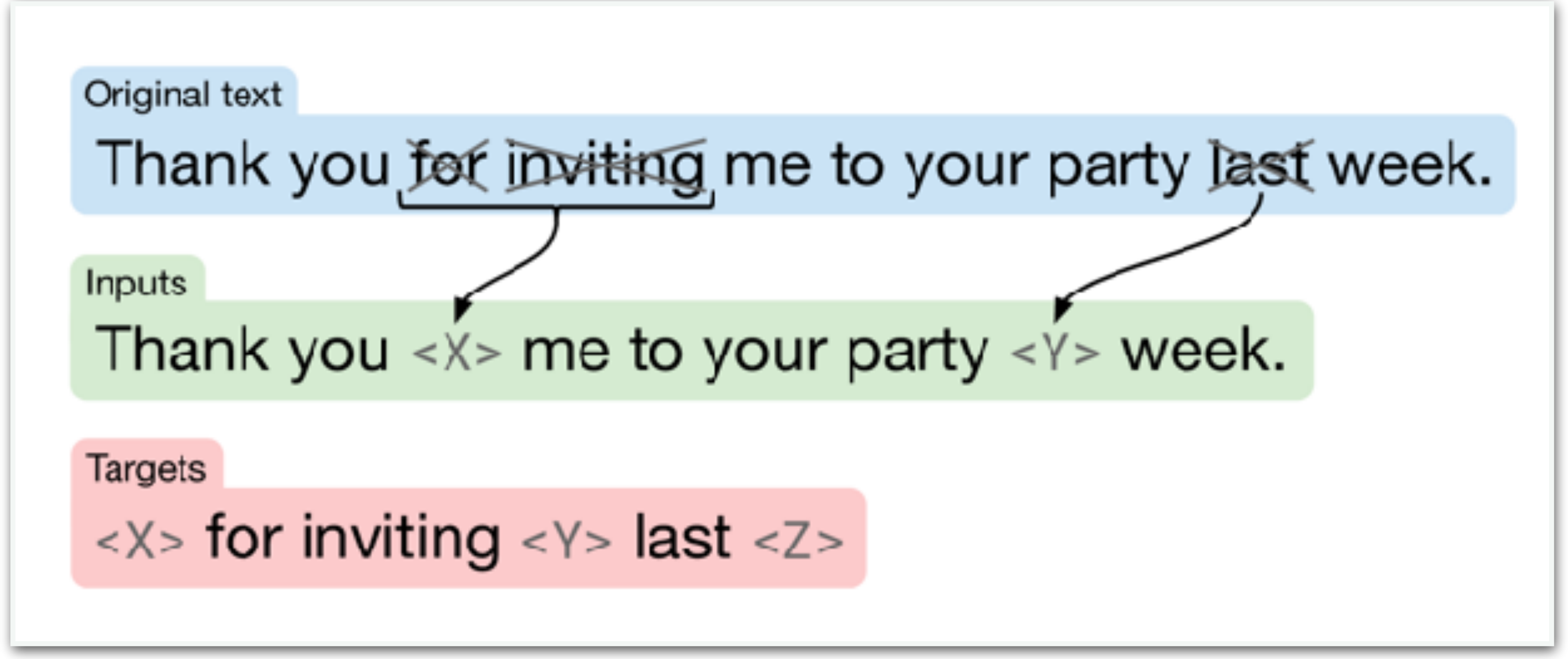
Devlin et al., 2018



Masked token prediction

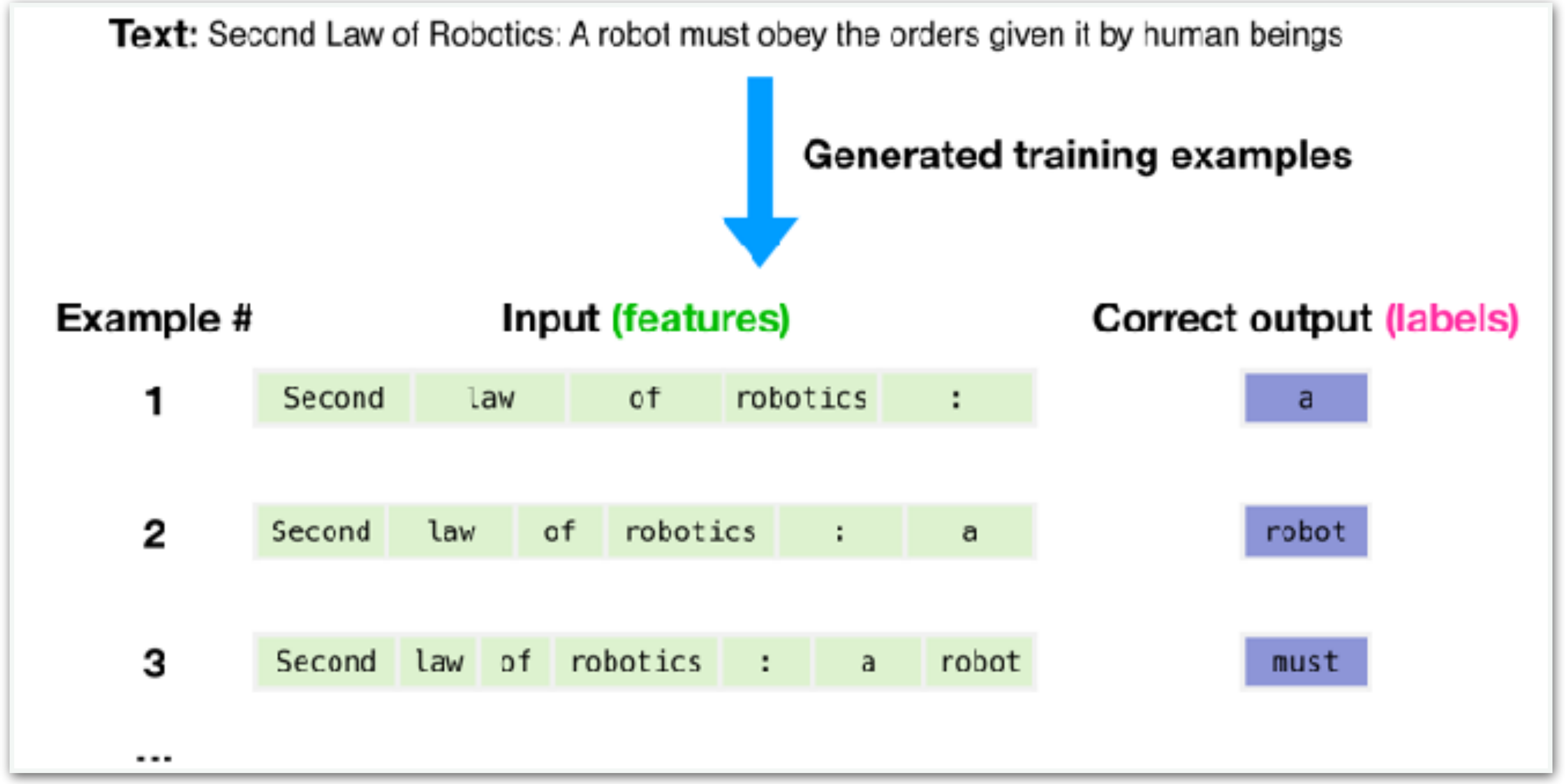
T5

Raffel et al., 2019



Denoising span-mask prediction

GPT - 4



Next token prediction

https://github.com/manueldeprada/Pretraining-T5-PyTorch-Lightning

Pre-training architectures

Encoder

- E.g., BERT, RoBERTa, DeBERTa, …
- **Autoencoder** model
- **Masked** language modeling

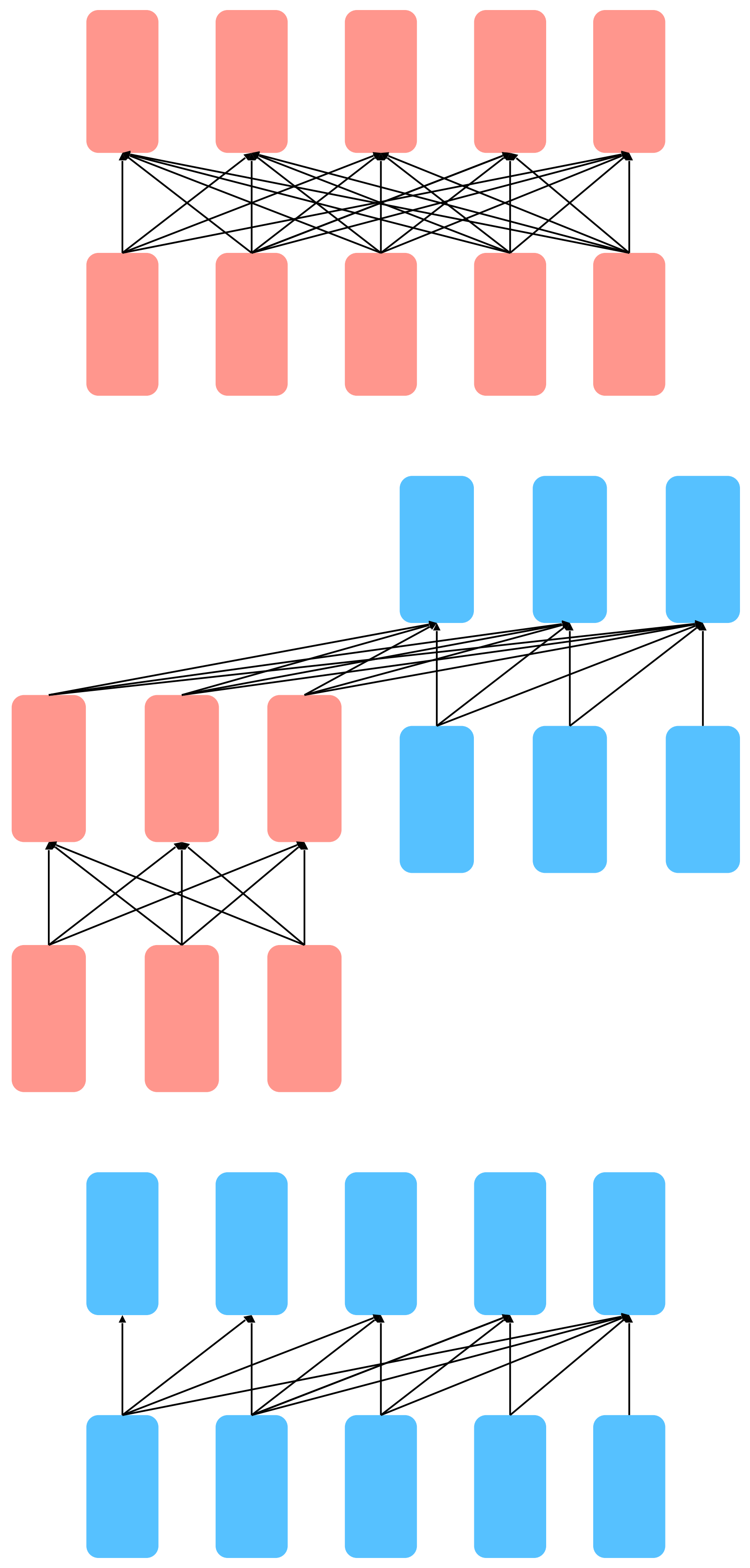
Encoder-Decoder

- E.g., T5, BART, …
- **seq2seq** model

Decoder

- E.g., GPT, GPT2, GPT3, …
- **Autoregressive** model
- **Left-to-right** language modeling

Pre-training architectures



Encoder

- Bidirectional; can condition on the future context

Encoder-Decoder

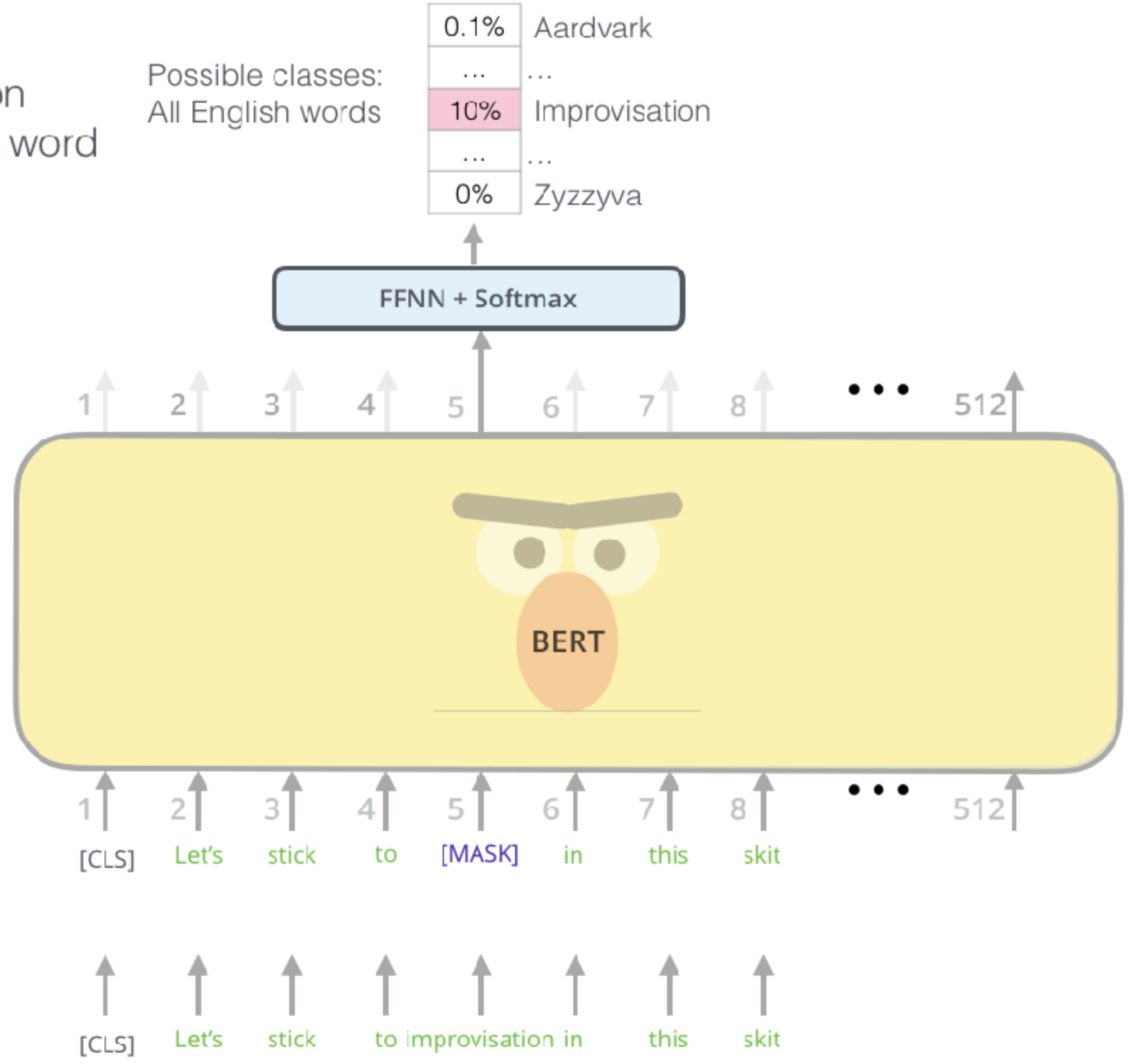
- Map two sequences of different length together

Decoder

- Language modeling; can only condition on the past context

Masked Language Modeling (MLM)

Use the output of the masked word's position to predict the masked word

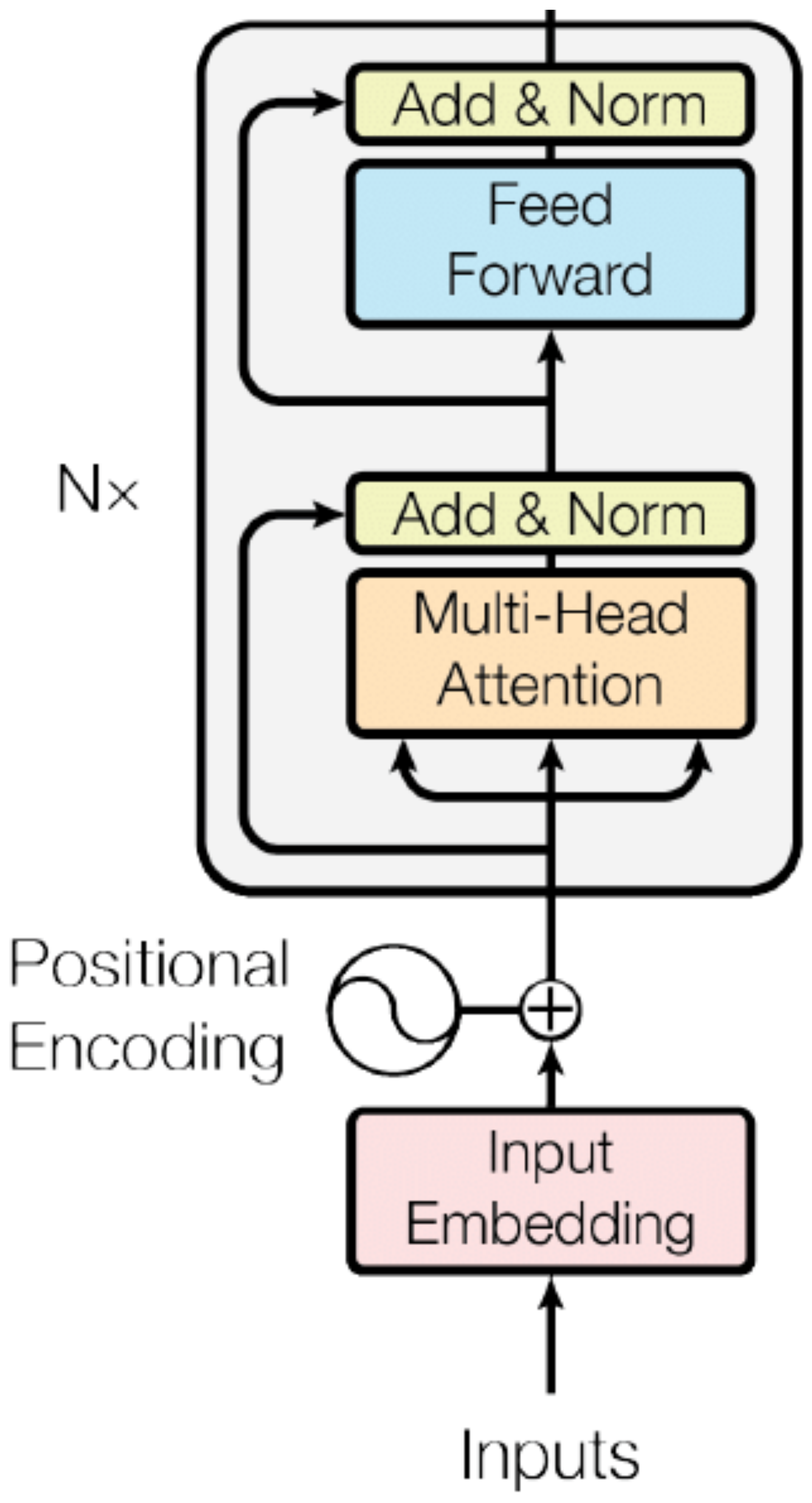


Randomly mask 15% of tokens

Input

http://jalammar.github.io/illustrated-bert/

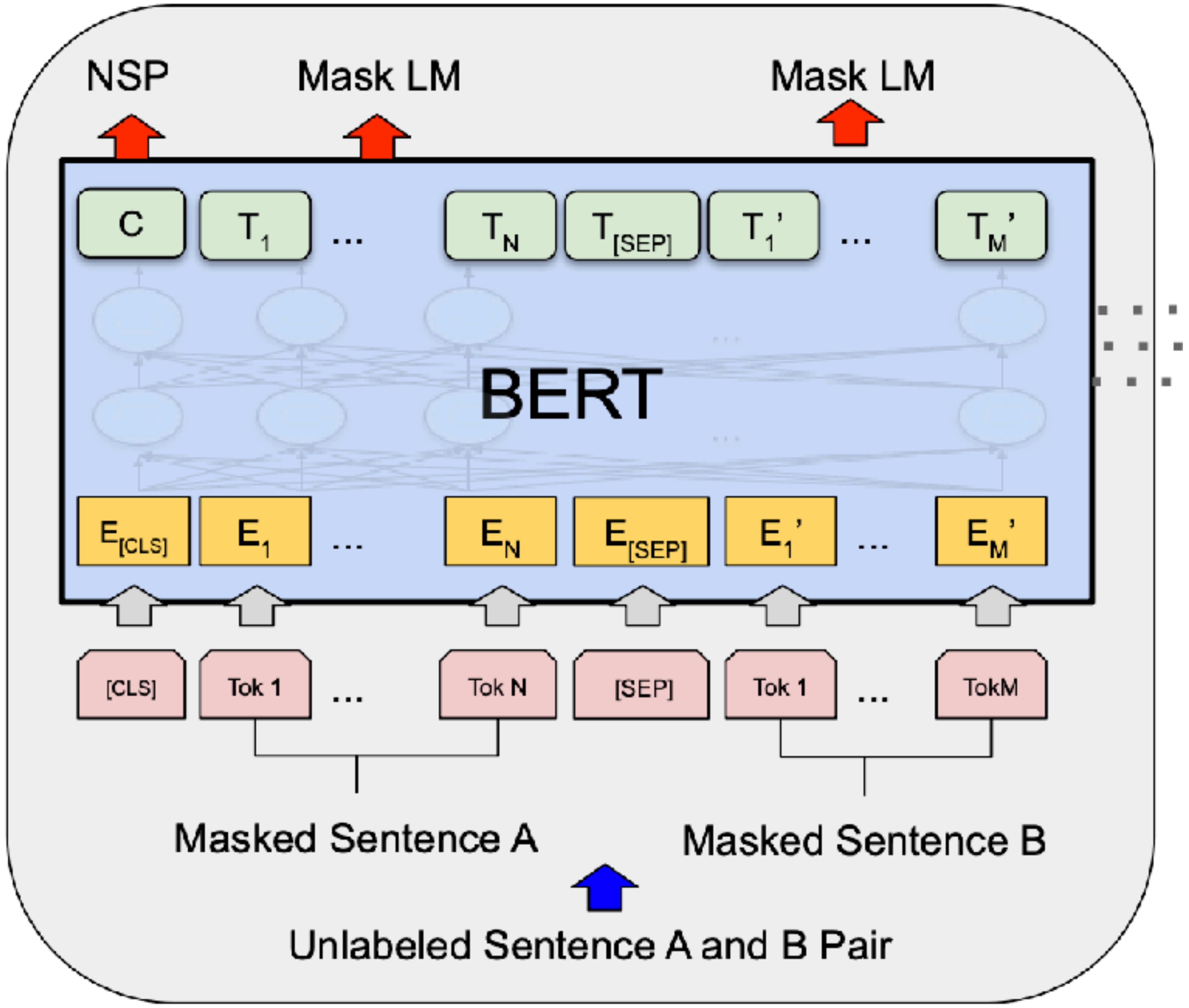
BERT pre-training



- BERT-base: 12 layers, 768 hidden size, 12 attention heads, 110M parameters
- BERT-large: 24 layers, 1024 hidden size, 16 attention heads, 340M parameters

- Training corpus: Wikipedia (2.5B) + BooksCorpus (0.8B)
- Max sequence size: 512 wordpiece tokens (roughly 256 and 256 for two non-contiguous sequences)
- Trained for 1M steps, batch size 128k

BERT pre-training

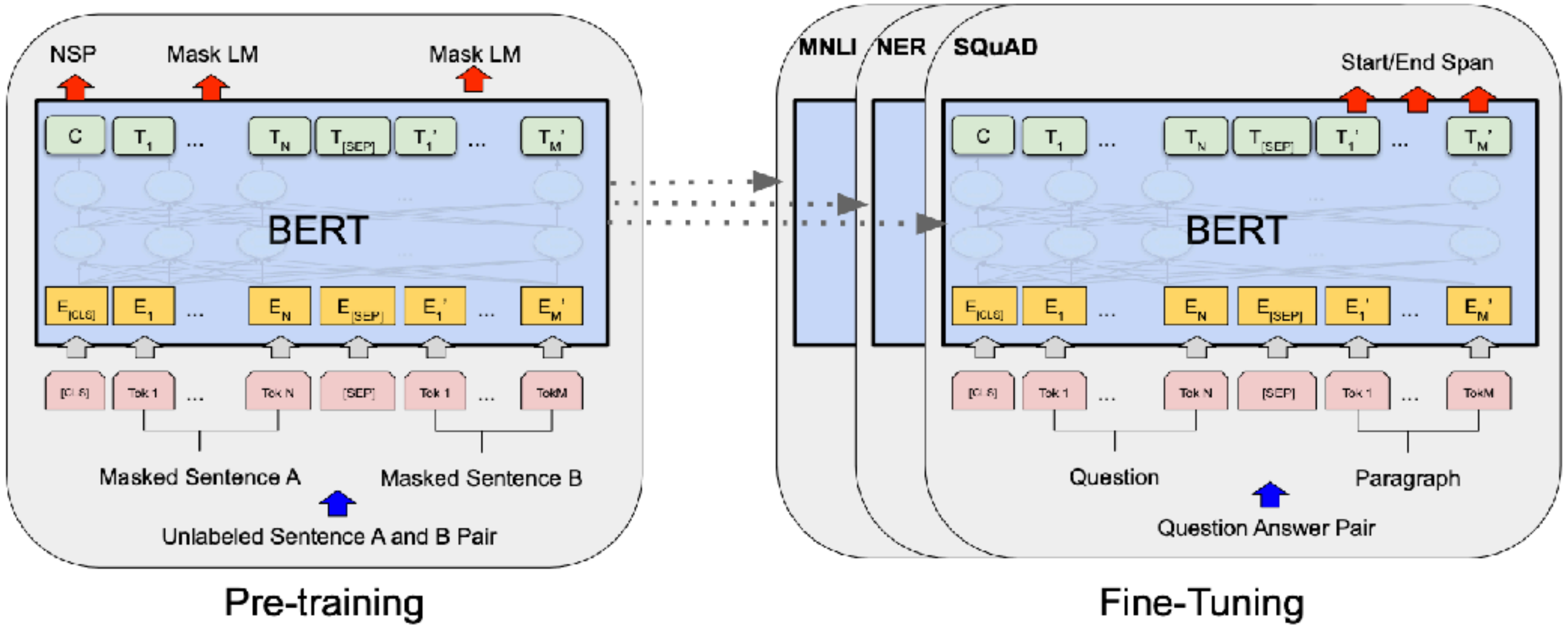


Pre-training

- MLM and NSP are trained together
- [CLS] is pre-trained for NSP
- Other token representations are trained for MLM

Pretraining / fine-tuning

“Pre-train” a model on a large dataset for task X, then “fine-tune” it on a dataset for task Y



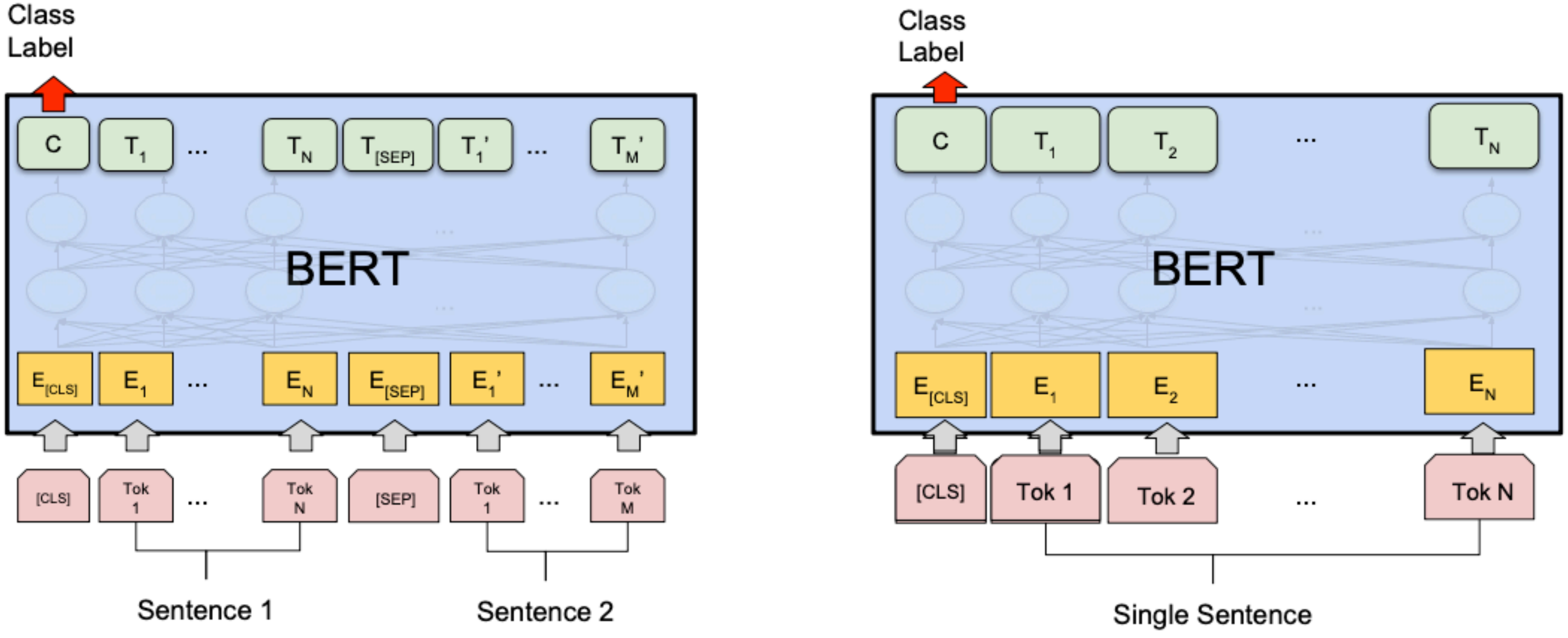
“**Fine-tuning** is the process of **taking the network learned by these pre-trained models**, and **further training the model**, often via an added neural net classifier that takes the top layer of the network as input, to perform some downstream task.”

Fine-tuning is a training process and takes **gradient descent steps**!

BERT fine-tuning

"Pretrain once, finetune many times."

sentence-level tasks



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

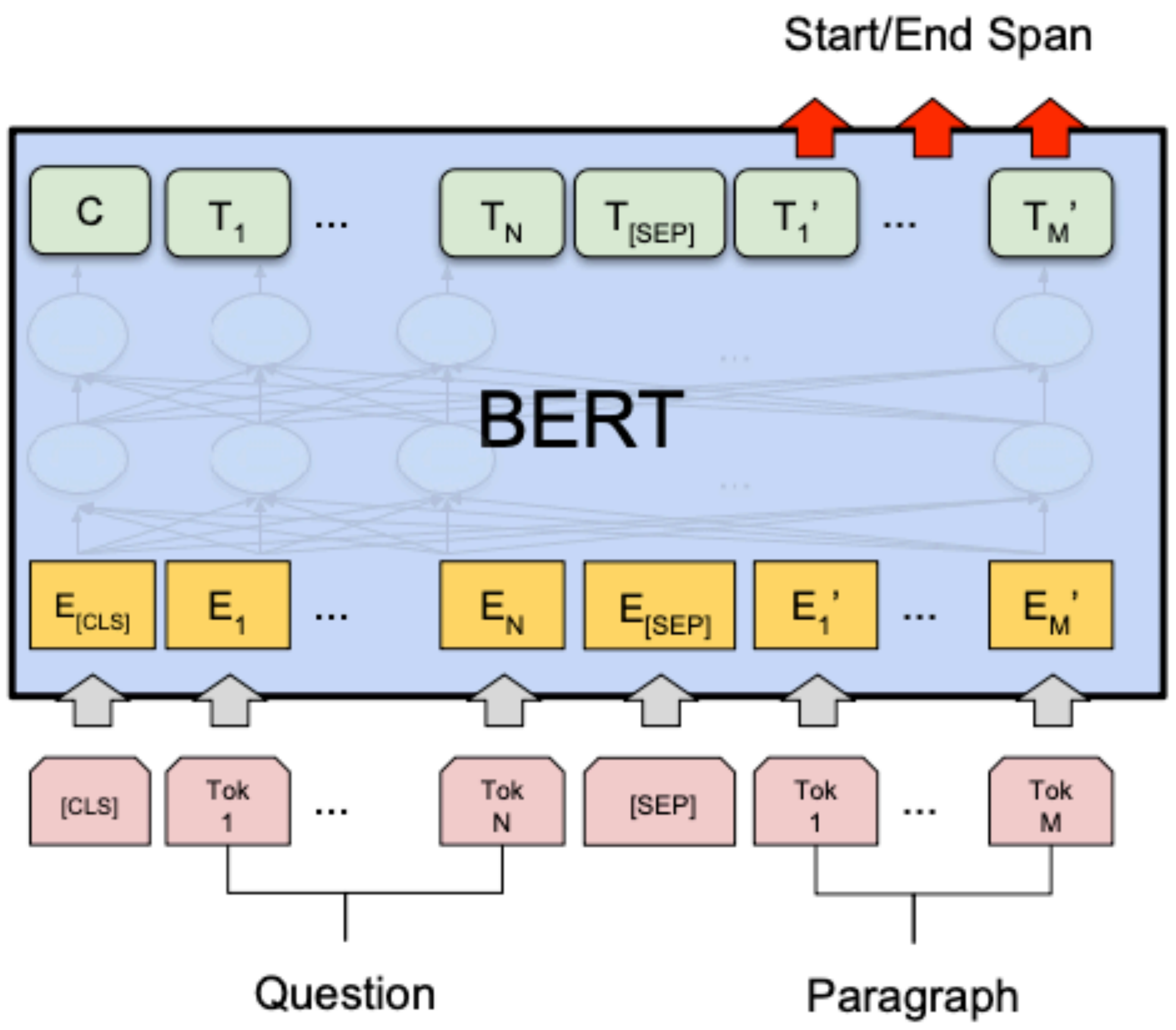
(b) Single Sentence Classification Tasks: SST-2, CoLA

- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI:** natural language inference over question answering data
- **SST-2:** sentiment analysis

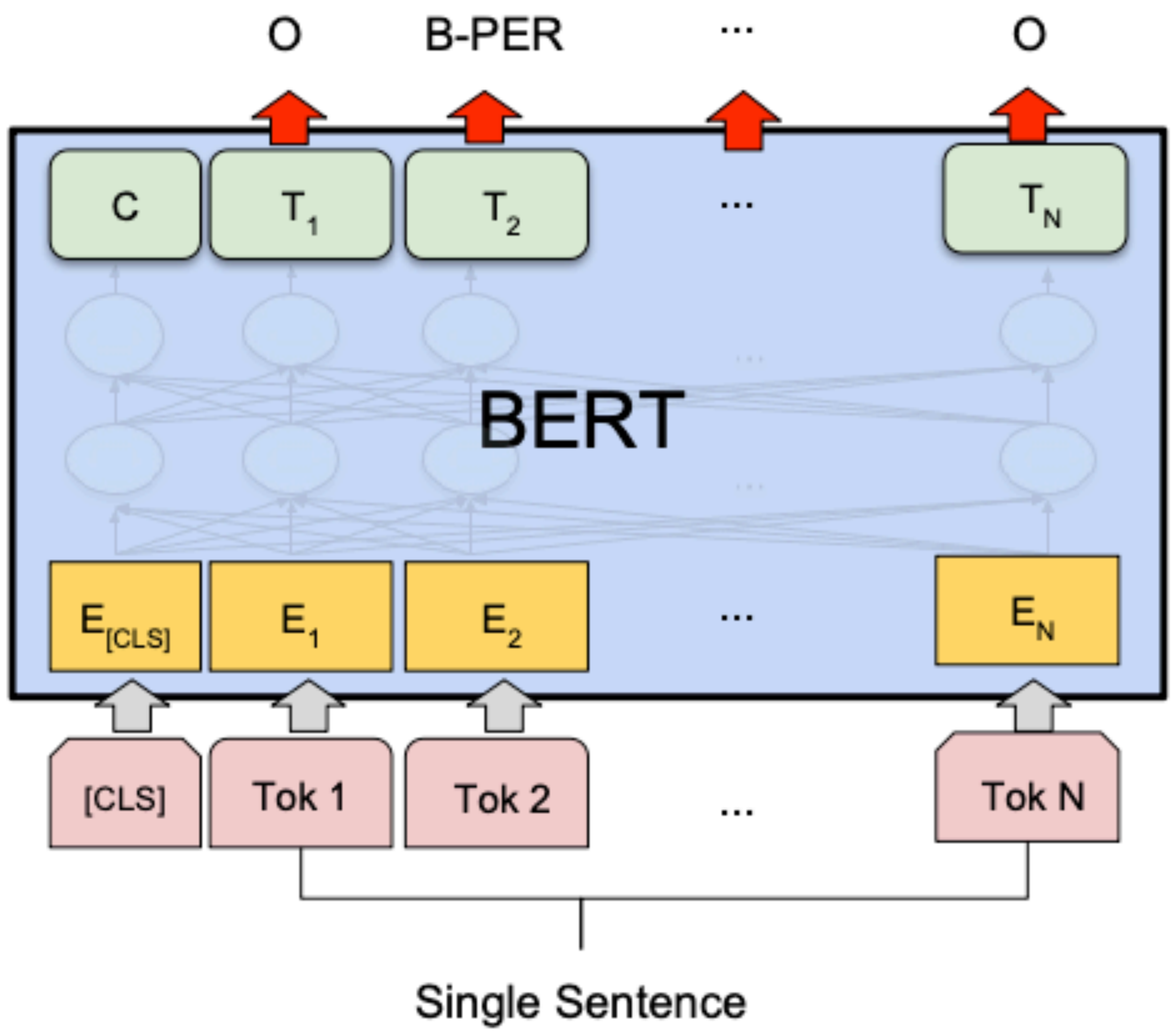
BERT fine-tuning

“Pretrain once, finetune many times.”

token-level tasks



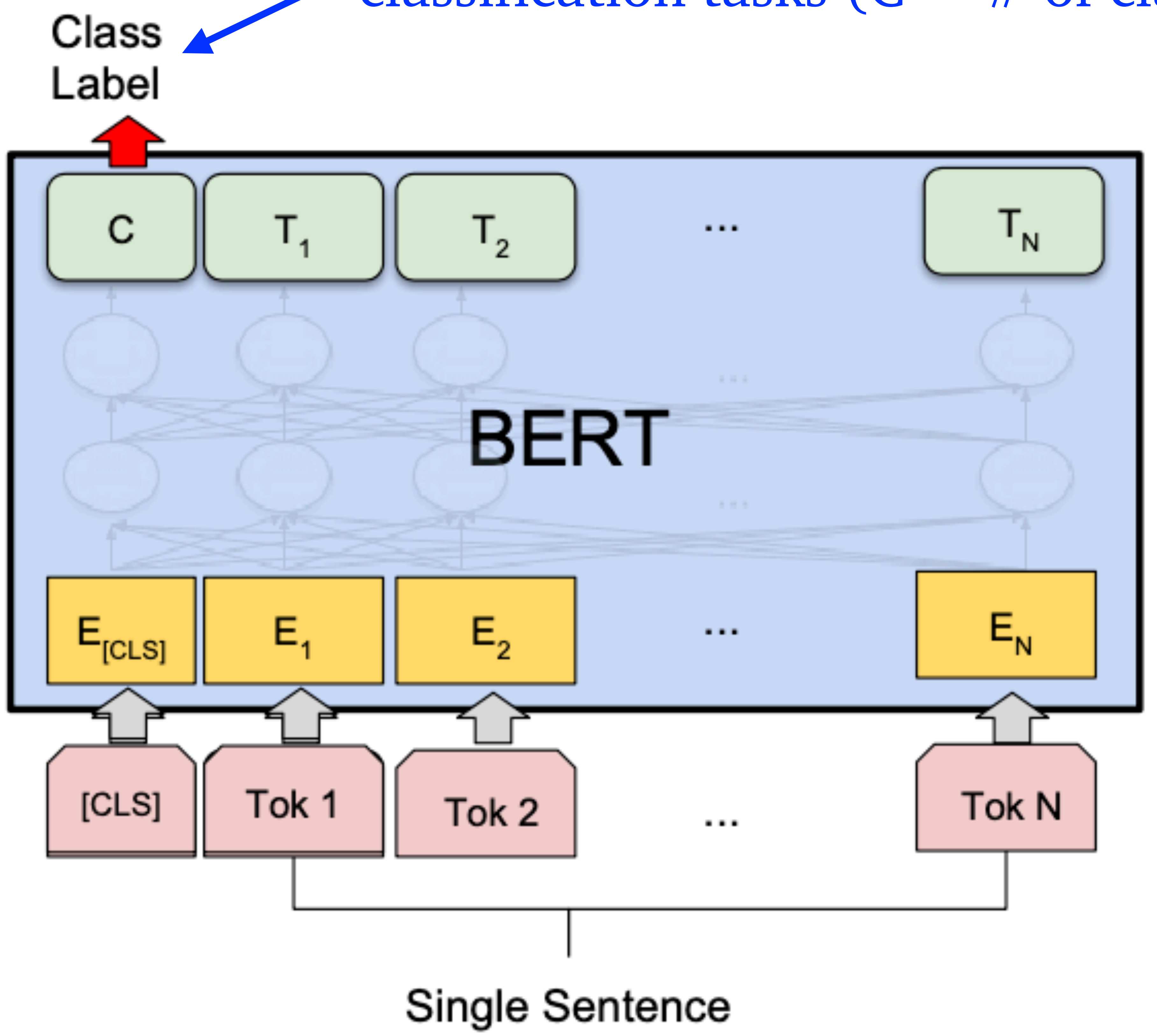
(c) Question Answering Tasks:
SQuAD v1.1



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

Example: sentiment classification

We just need to introduce $C \times h$ parameters for classification tasks (C = # of classes, h = hidden size)!



$$P(y = k) = softmax_k(\mathbf{W}_o \mathbf{h}_{[CLS]})$$

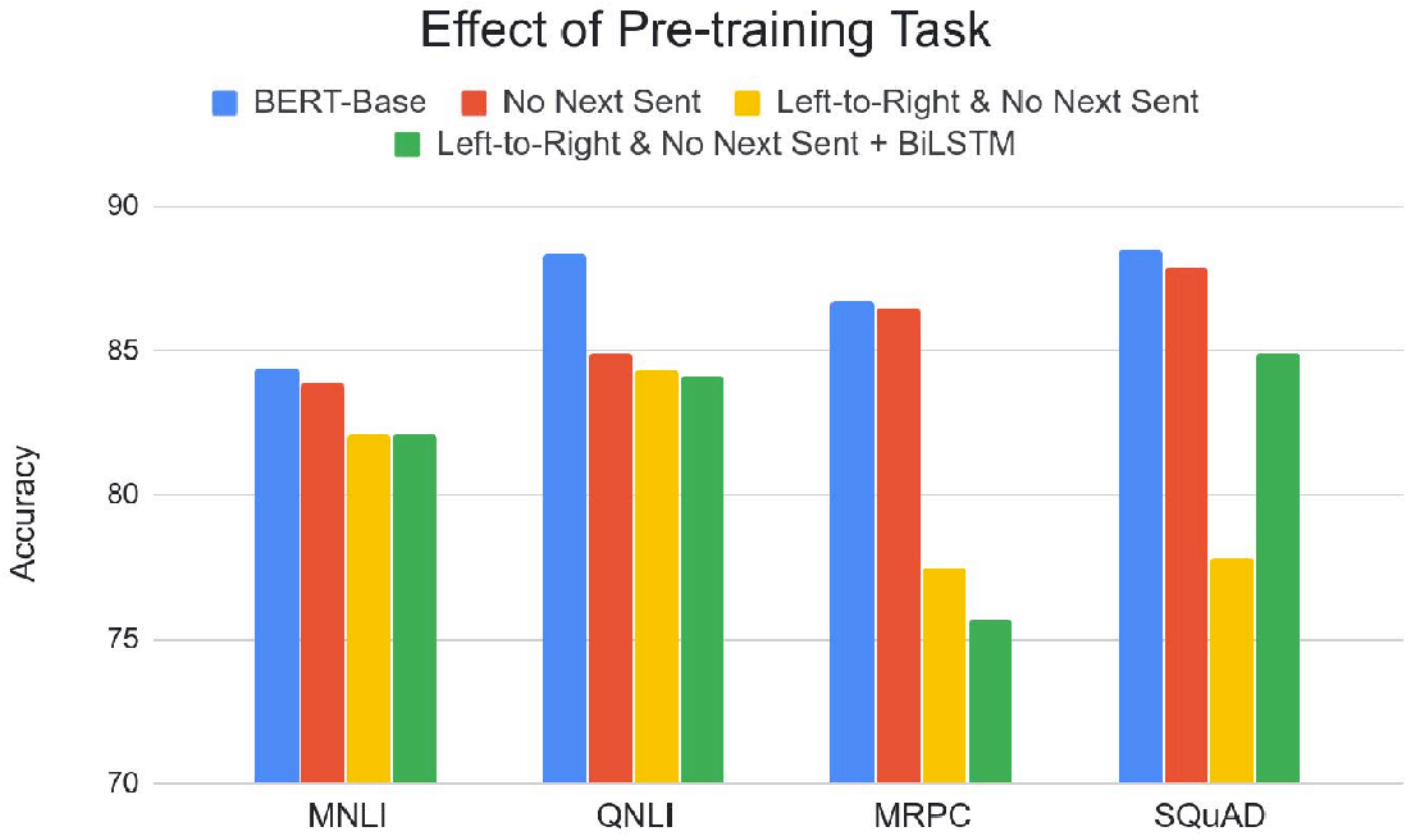
$$\mathbf{W}_o \in \mathbb{R}^{C \times h}$$

All the parameters will be learned together (original BERT parameters + new classifier parameters)

Experimental results: GLUE

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**

Ablation study: pre-training tasks



- MLM >> left-to-right LMs
- NSP improves on some tasks
- Note: later work (Joshi et al., 2020; Liu et al., 2019) argued that NSP is not useful

Ablation study: model sizes

layers ↓ hidden size ↓ # of heads ↙

Hyperparams			Dev Set Accuracy			
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

The bigger, the better!

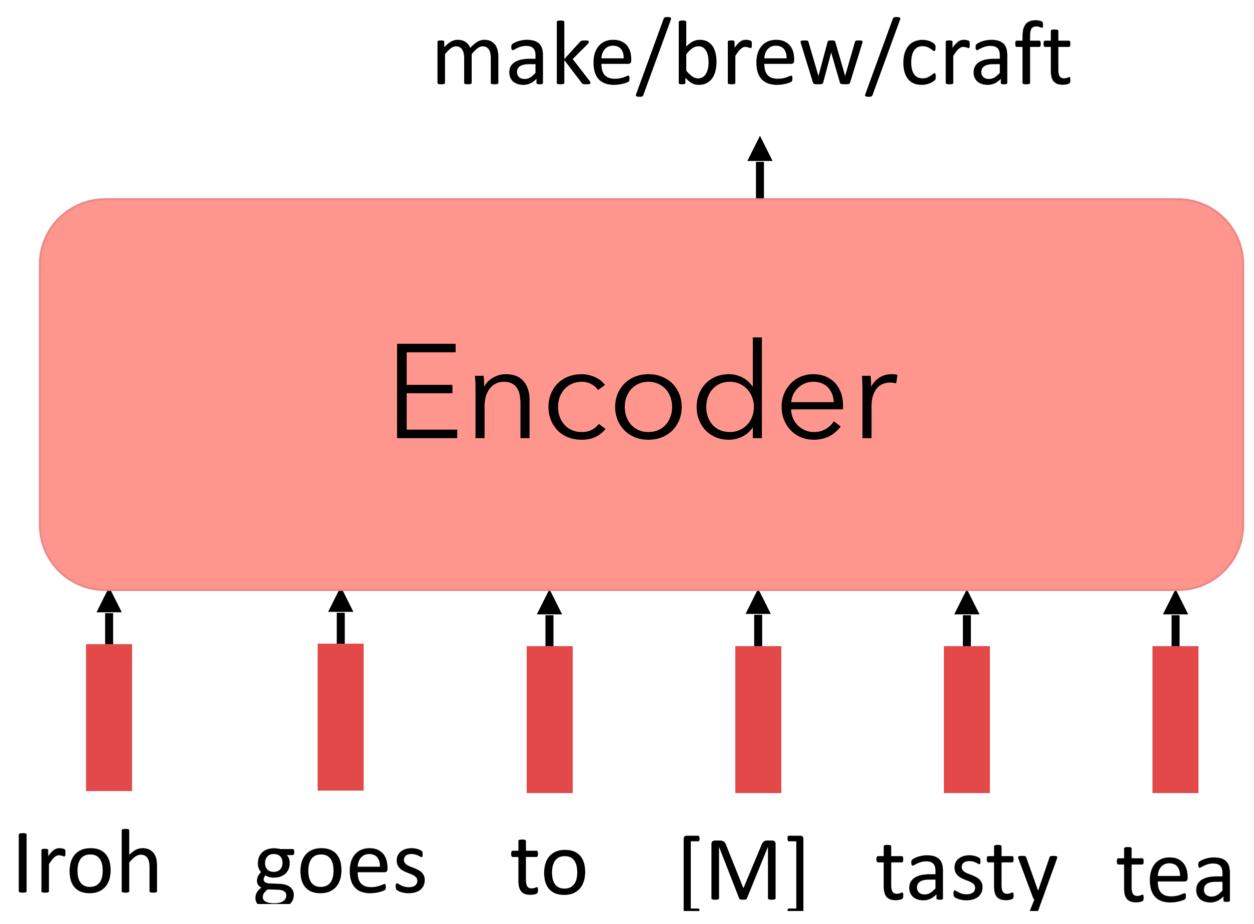
Encoder: other variations of BERT

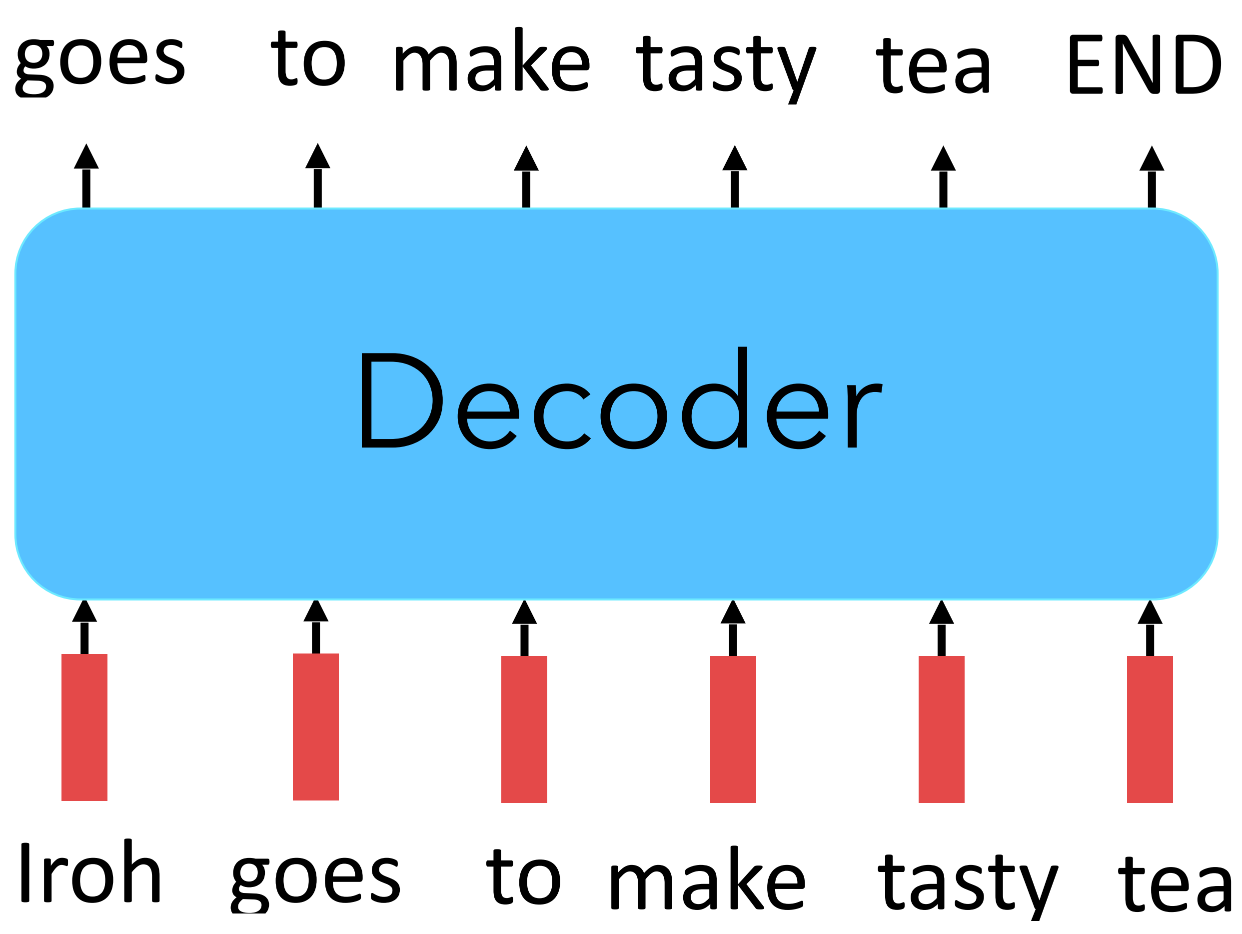
- **ALBERT [Lan et al., 2020]**: incorporates two parameter reduction techniques that lift the major obstacles in scaling pre-trained models
- **DeBERTa [He et al., 2021]:** decoding-enhanced BERT with disentangled attention
- **SpanBERT [Joshi et al., 2019]:** masking contiguous spans of words makes a harder, more useful pre-training task
- **ELECTRA [Clark et al., 2020]:** corrupts texts by replacing some tokens with plausible alternatives sampled from a small generator network, then train a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample or not.
- **DistilBERT [Sanh et al., 2019]:** distilled version of BERT that's 40% smaller
- **TinyBERT [Jiao et al., 2019]:** distill BERT for both pre-training & fine-tuning
- **…**

Encoder: pros & cons

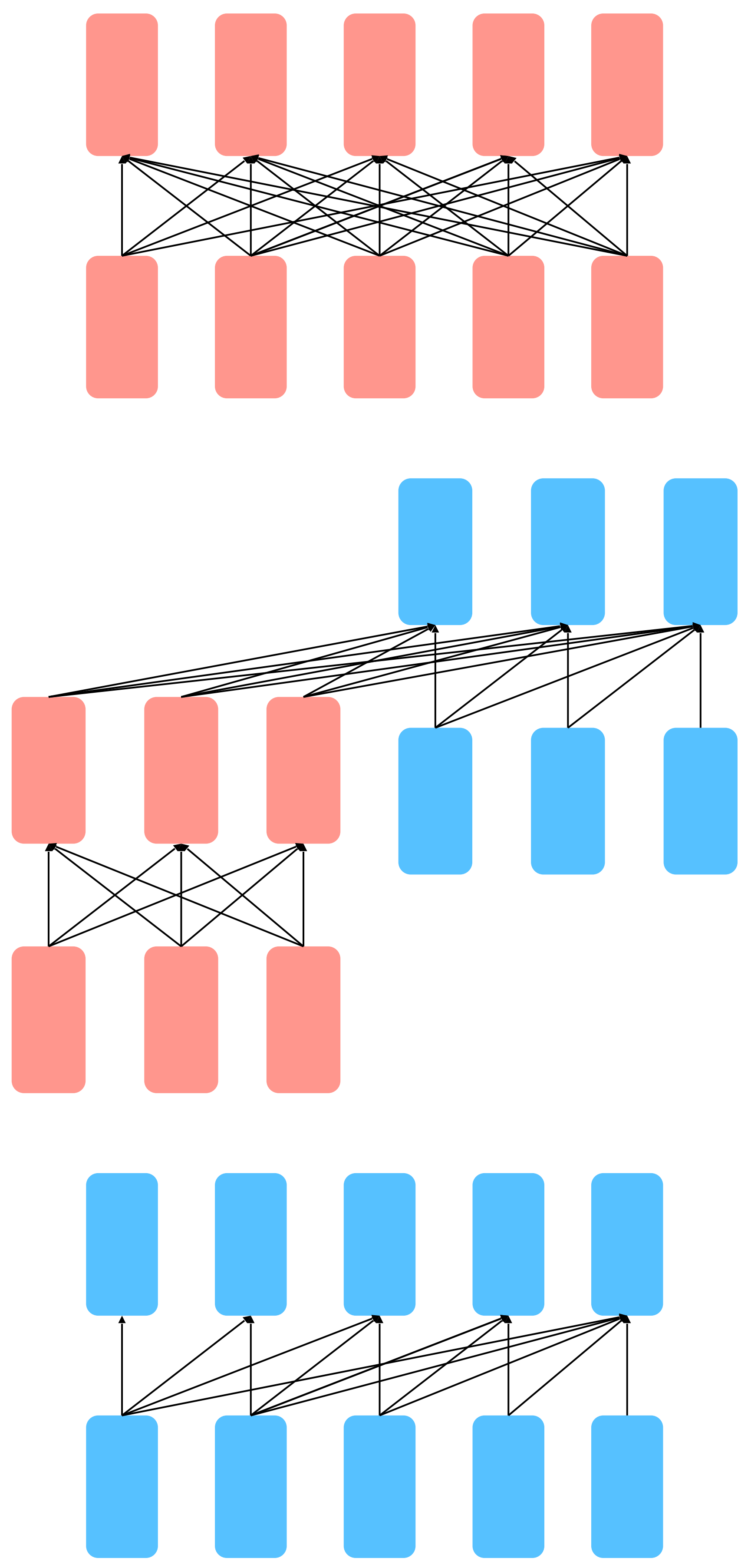
- Consider both left and right context
- Capture intricate contextual relationships

- Not good at generating open-text from left-to-right, one token at a time





Pre-training architectures



Encoder

- Bidirectional; can condition on the future context

Encoder-Decoder

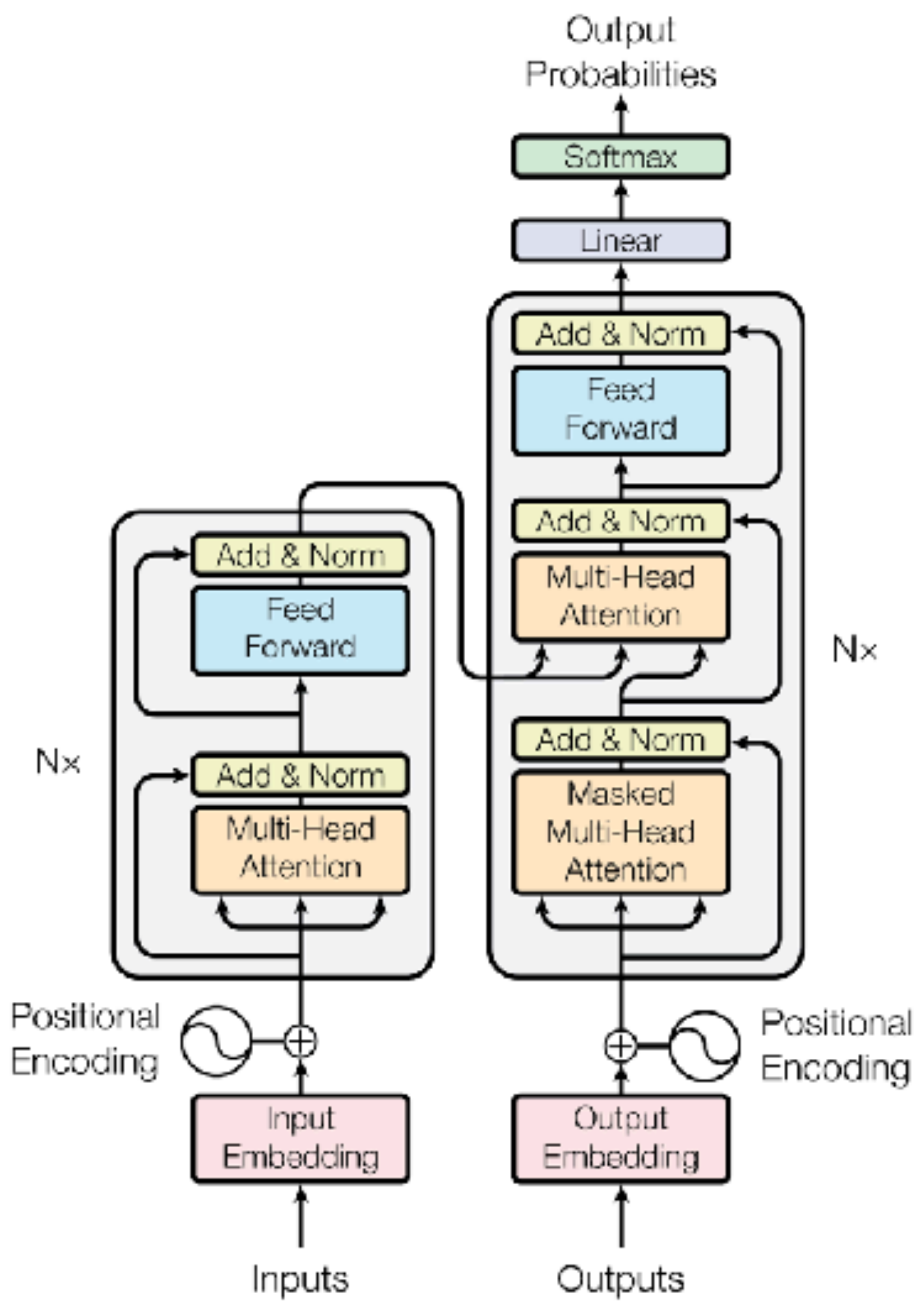
- Map two sequences of different length together

Decoder

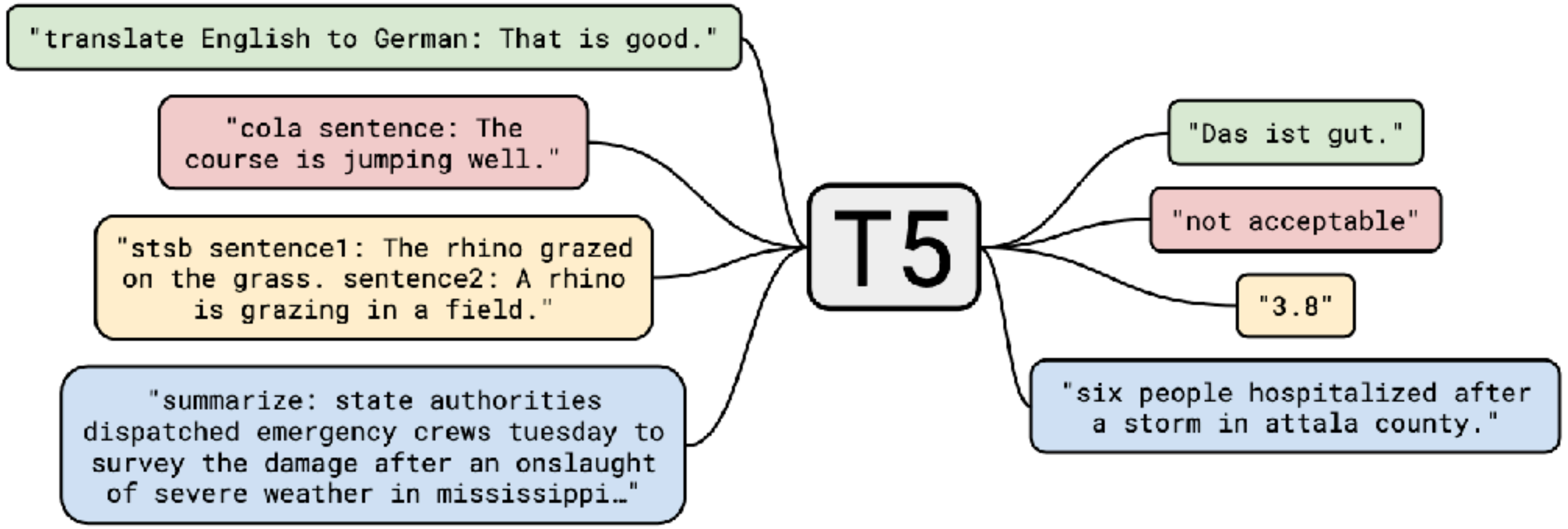
- Language modeling; can only condition on the past context

Text-to-text models: the best of both worlds

- So bar, **encoder-only models (e.g., BERT)** enjoy the benefits of **bidirectionality** but they can't be used to generate text
- **Decoder-only models (e.g., GPT)** can do generation but they are left-to-right LMs..
- Text-to-text models combine the best of both worlds!



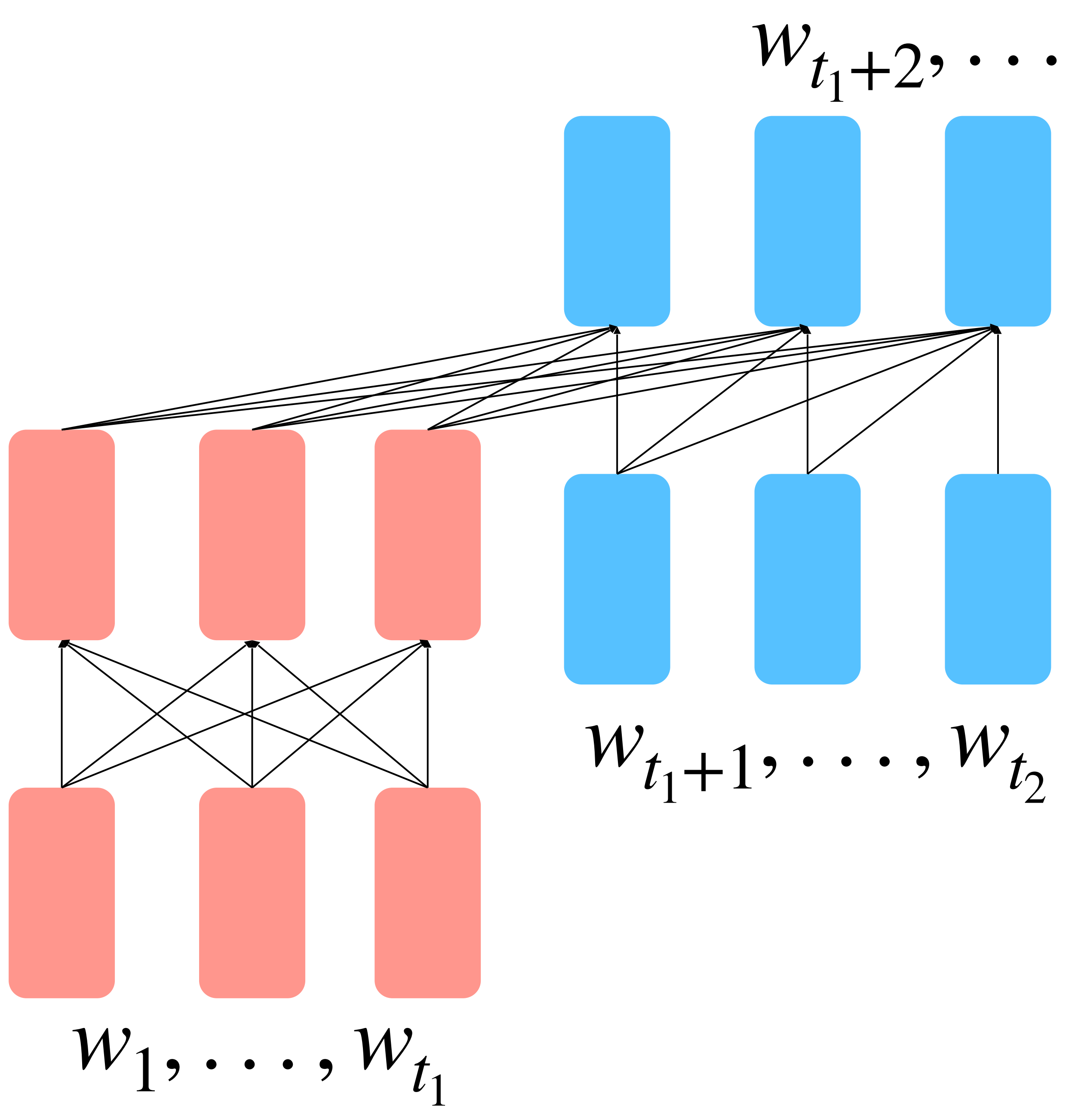
T5 = <u>**T**</u>ext-<u>**t**</u>o-<u>**T**</u>ext <u>**T**</u>ransfer <u>**T**</u>ransformer



(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

Encoder-decoder: architecture

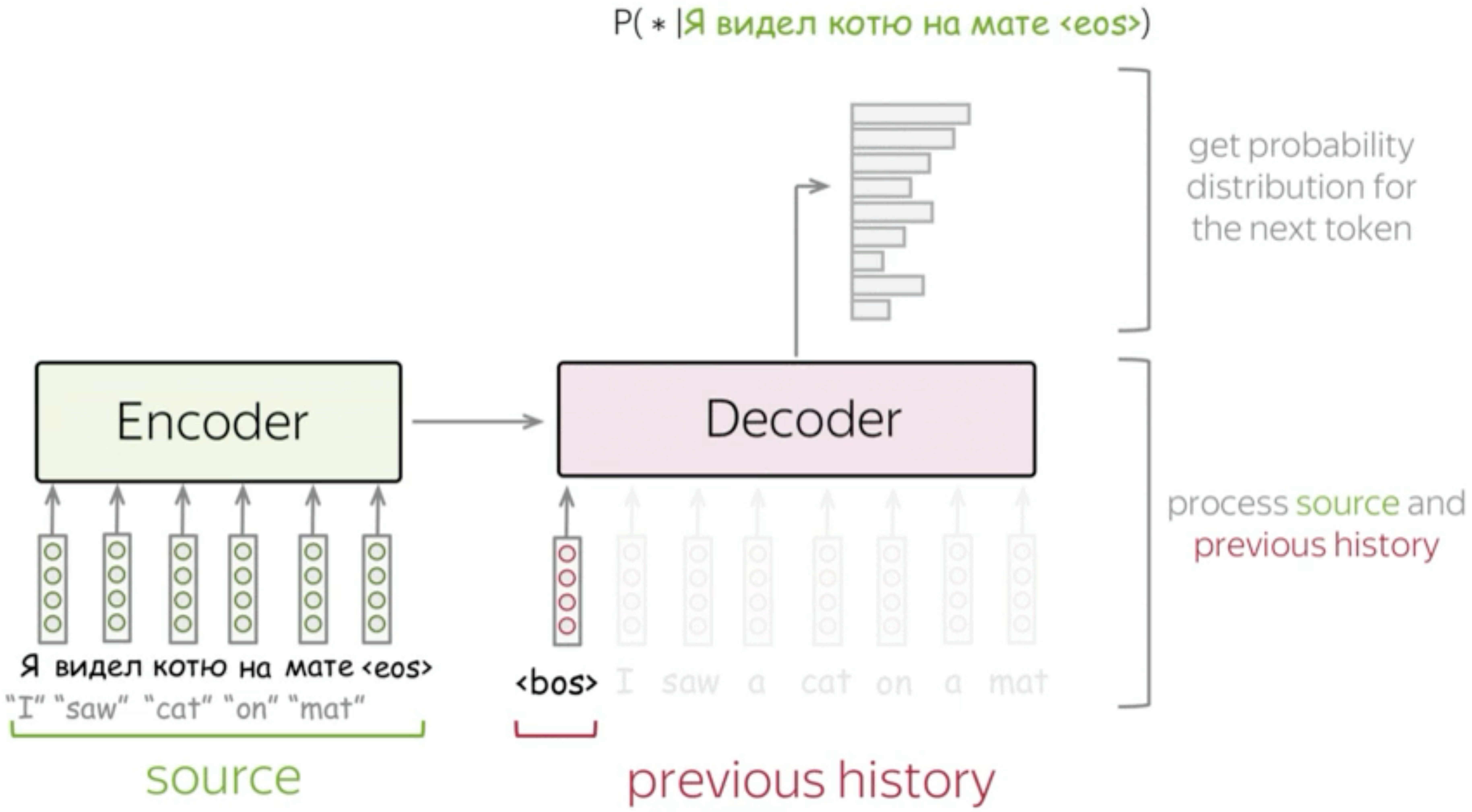
- Moving towards **open-text generation**…
- **Encoder** builds a representation of the source and gives it to the **decoder**
- **Decoder** uses the source representation to generate the target sentence
- The **encoder** portion benefits from **bidirectional** context; the **decoder** portion is used to train the whole model through **language modeling**



$$h_1, \ldots, h_{t_1} = \text{Encoder}(w_1, \ldots, w_{t_1})$$
$$h_{t_1+1}, \ldots, h_{t_2} = \text{Decoder}(w_{t_1+1}, \ldots, w_{t_2}, h_1, \ldots, h_{t_1})$$
$$y_i \sim Ah_i + b, i > t$$

[Raffel et al., 2018]

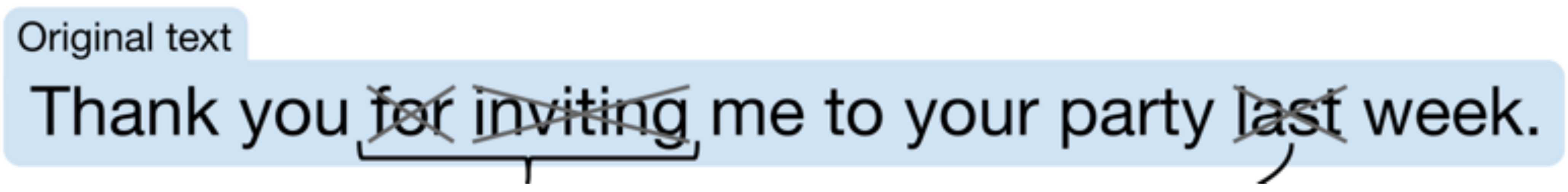
Encoder-decoder: machine translation example

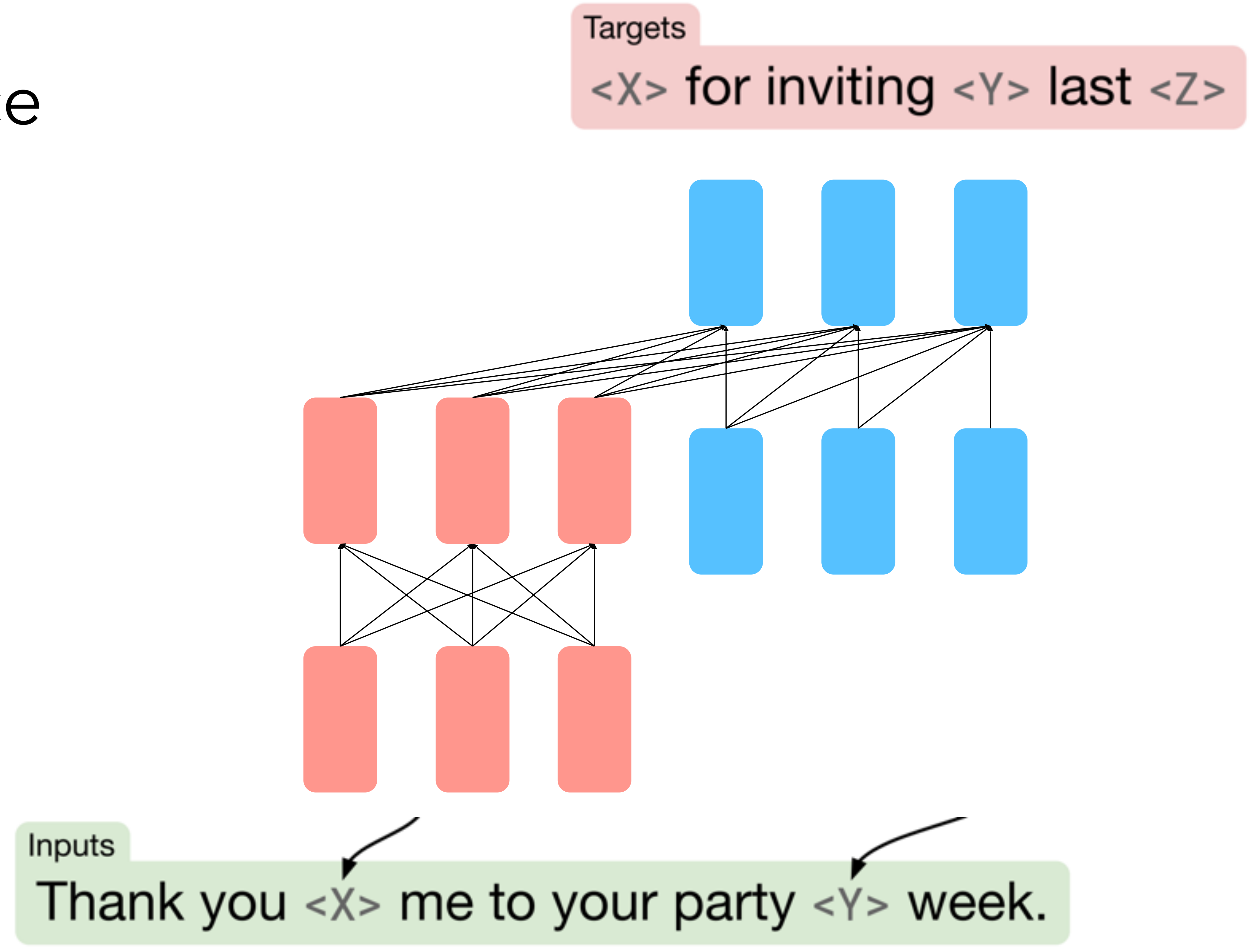


[Lena Viota Blog]

Encoder-decoder: training objective

- **T5 [Raffel et al., 2018]**
- **Text span corruption (denoising):** Replace different-length spans from the input with unique placeholders (e.g., <extra_id_0>); decode out the masked spans.
  - Done during **text preprocessing**: training uses **language modeling** objective at the decoder side





Encoder-decoder: T5

[Raffel et al., 2018]

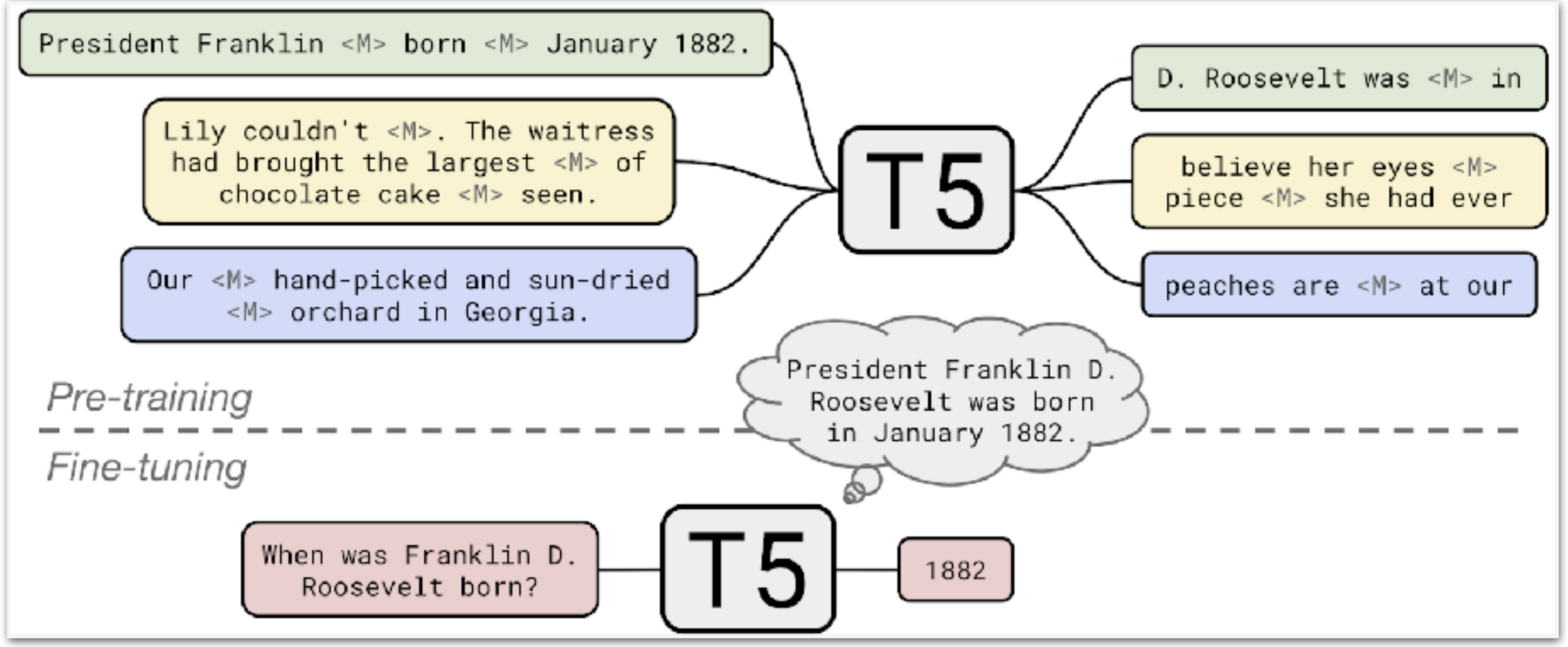
- **Encoder-decoders** works better than decoders
- **Span corruption (denoising)** objective works better than language modeling

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

Encoder-decoder: T5

[Raffel et al., 2018]

- **Text-to-Text:** convert NLP tasks into input/output text sequences
- **Dataset:** Colossal Clean Crawled Corpus (C4), 750G text data!
- **Various Sized Models:**
  - Base (222M)
  - Small (60M)
  - Large (770M)
  - 3B
  - 11B
- **Achieved SOTA with scaling & purity of data**



[Google Blog]

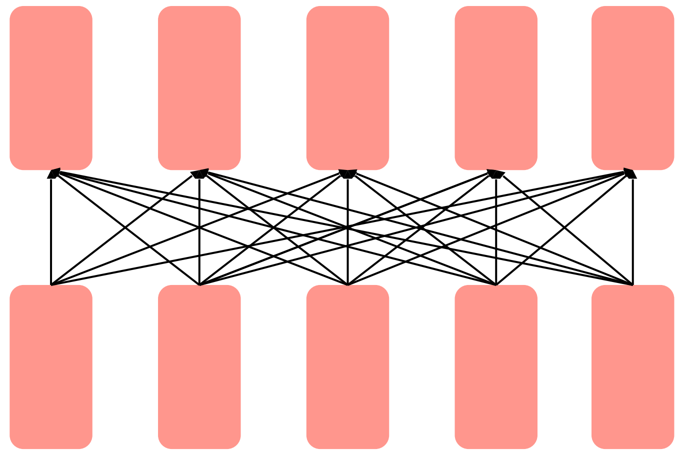
Encoder-decoder: pros & cons

- A nice middle ground between leveraging **bidirectional** contexts and **open-text** generation
- Good for **multi-task** fine-tuning

- Require more **text wrangling**
- **Harder to train**
- **Less flexible** for natural language generation

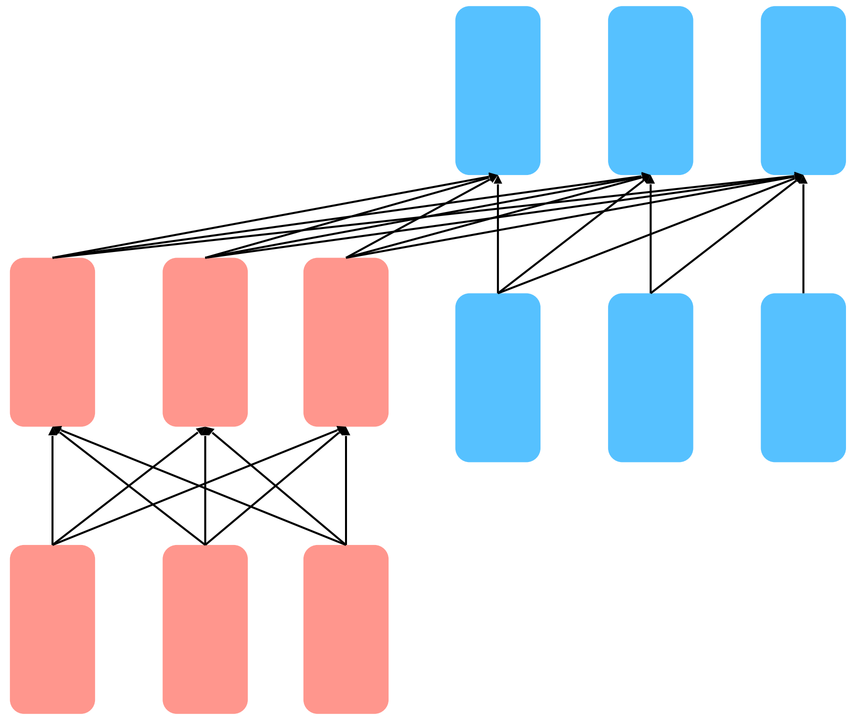
Pre-training architectures

Encoder



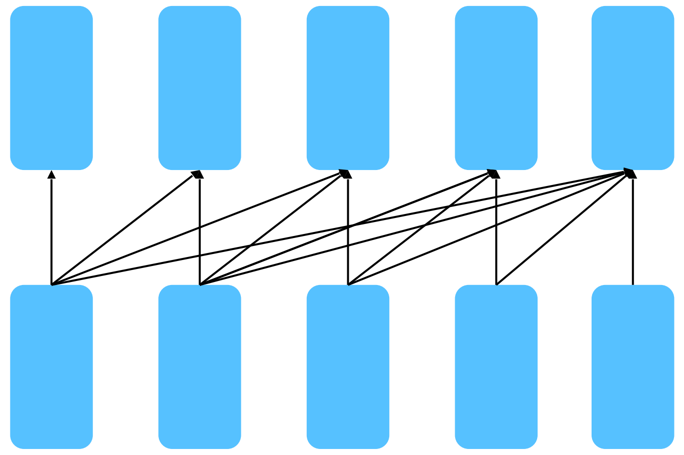
- Bidirectional; can condition on the future context

Encoder-Decoder



- Map two sequences of different length together

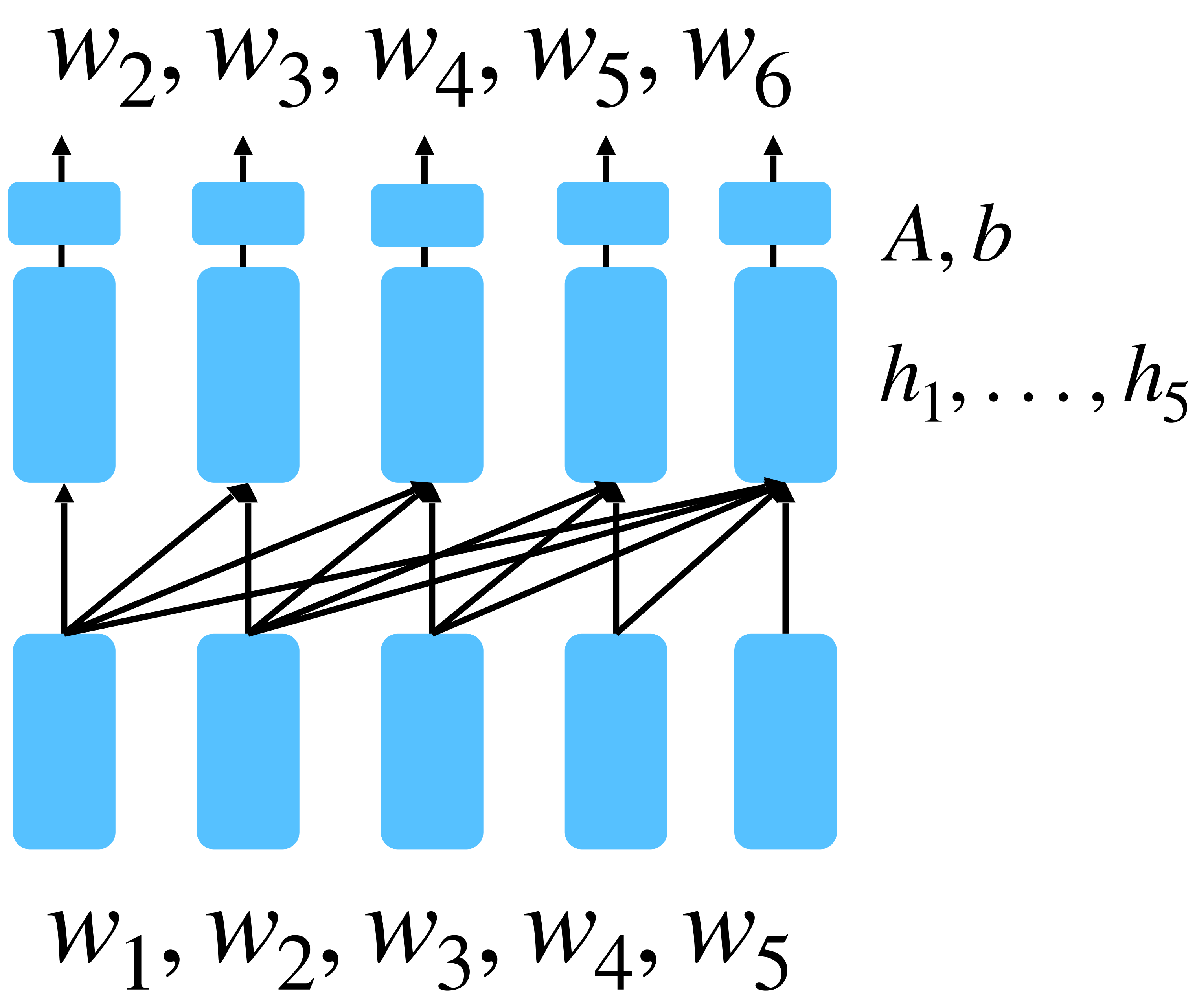
Decoder



- Language modeling; can only condition on the past context

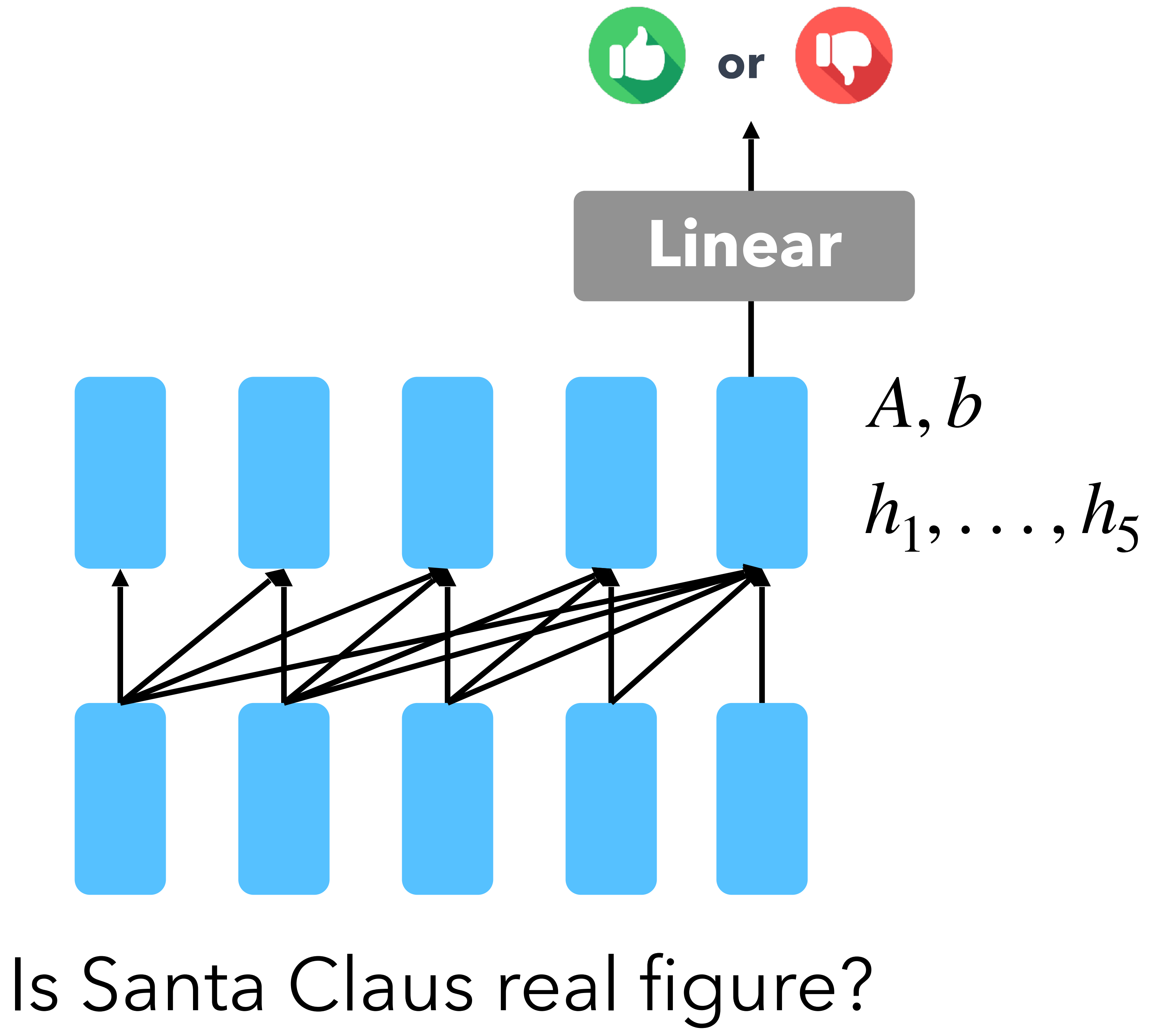
Decoder: training objective

- Many most famous generative LLMs are **decoder-only**
  - e.g., GPT1/2/3/4, Llama1/2
- **Language modeling!** Natural to be used for **open-text generation**
- **Conditional LM:** $p(w_t \mid w_1, \ldots, w_{t-1}, x)$
  - Conditioned on a source context x to generate from left-to-right
- Can be fine-tuned for **natural language generation (NLG)** tasks, e.g., dialogue, summarization.



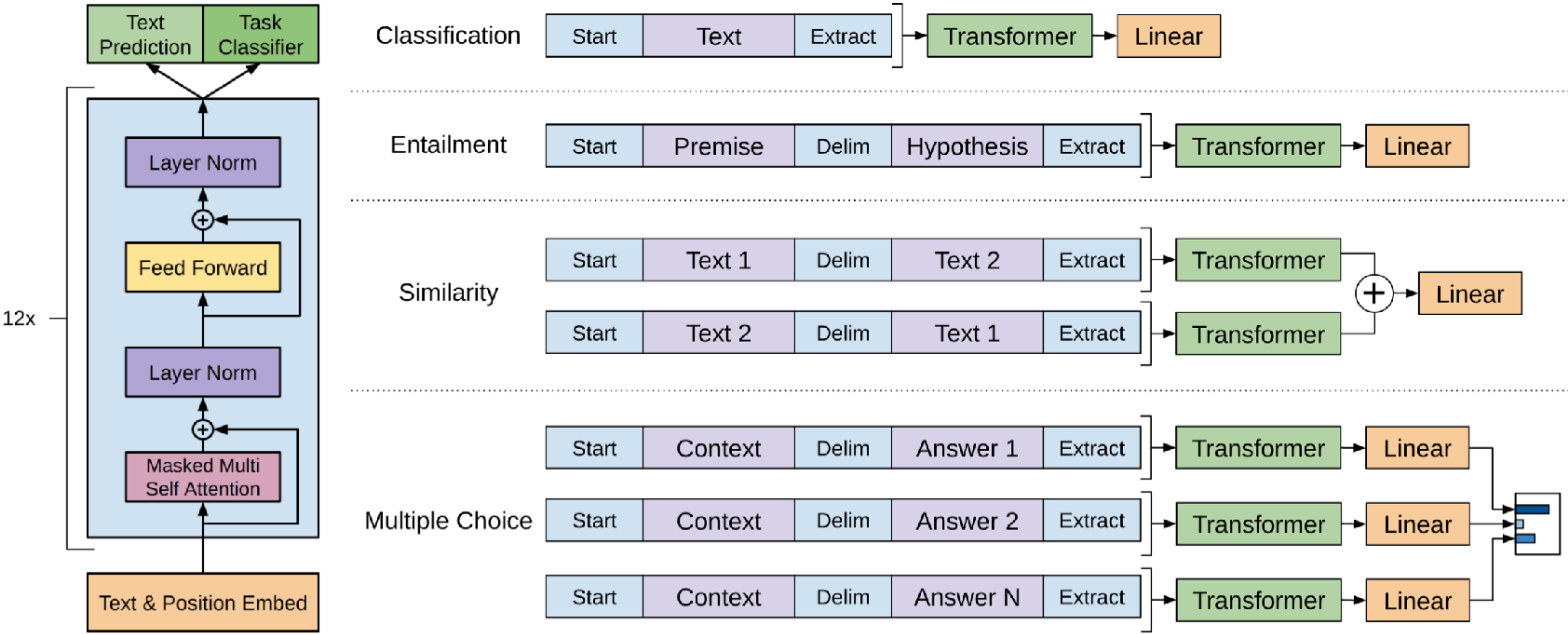
Decoder: training objective

- Customizing the pre-trained model for downstream tasks:
  - Add a **linear layer** on top of the last hidden layer to make it a classifier!
  - During fine-tuning, trained the randomly **initialized linear layer**, along with **all parameters** in the neural net.



Decoder: GPT

Generative **P**re-trained **T**ransformer [Radford et al., 2018]



How to use these pre-trained models?

Transformers

Transformers

Search documentation ⌘K

V4.27.2 EN 92,354

- CANINE
- CodeGen
- ConvBERT
- CPM
- CTRL
- DeBERTa
- DeBERTa-v2
- DialoGPT
- DistilBERT
- DPR
- ELECTRA

DistilBERT

All model pages distilbert | Hugging Face Spaces

Overview

The DistilBERT model was proposed in the blog post Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT, and the paper DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than *bert-base-uncased*, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.

```
>>> from transformers import AutoTokenizer

>>> tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")


>>> def tokenize_function(examples):
...     return tokenizer(examples["text"], padding="max_length", truncation=True)


>>> tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

```
>>> from transformers import AutoModelForSequenceClassification

>>> model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=5)
```

How to pick a proper architecture for a given task?

- Right now **decoder-only** models seem to dominant the field at the moment
    - e.g., GPT1/2/3/4, Mistral, Llama1/2
- T5 (seq2seq) works well with multi-tasking
- **Picking the best model architecture remains an open research question!**