Different Types of Transformer Models

Sourav Karmakar

souravkarmakar29@gmail.com

Transformer Architecture: Recap

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

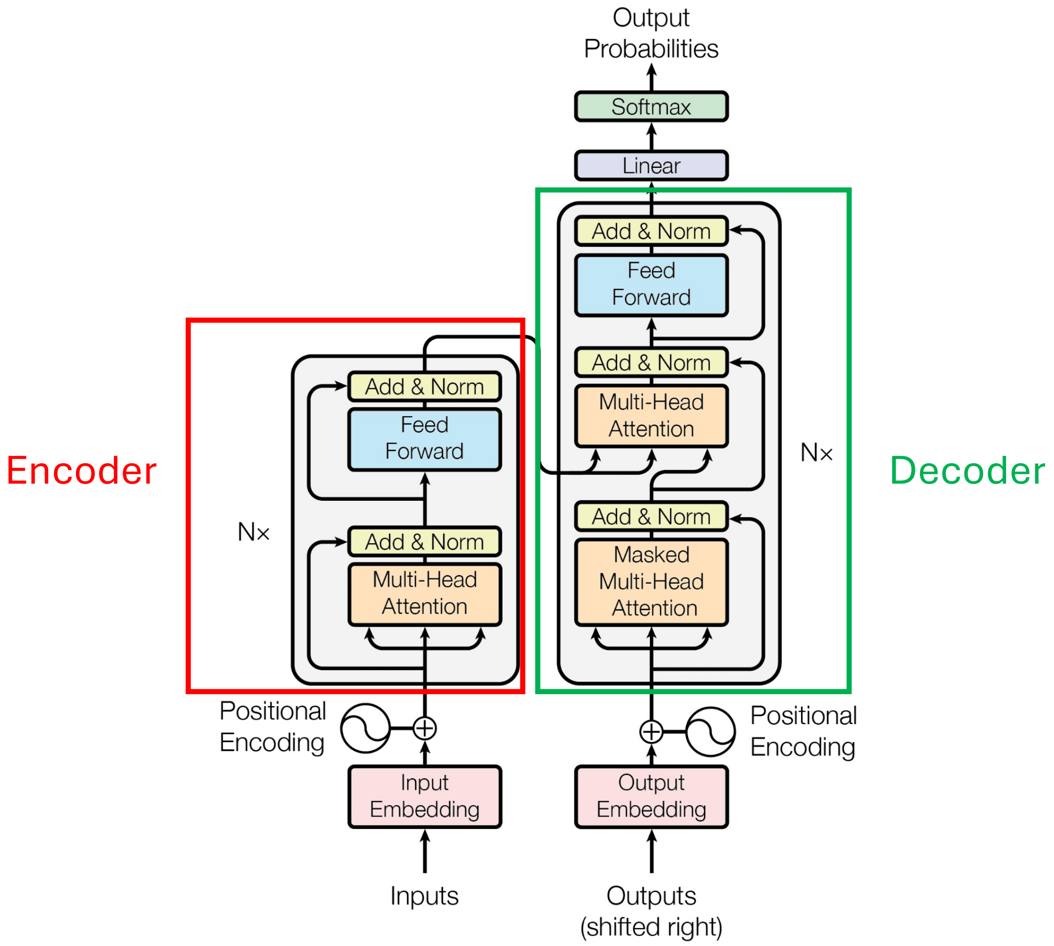
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Paper: Attention is all you need by Vaswani et. al.

Transformer Architecture: Recap

The transformer consists of two parts. One is Encoder and the other is Decoder.



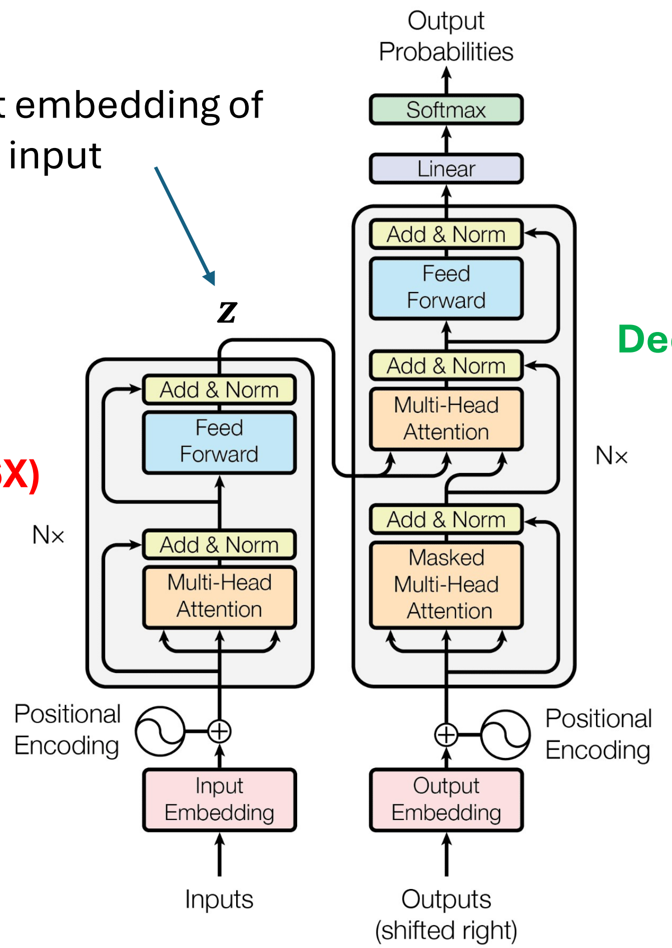
Transformer Architecture: Recap

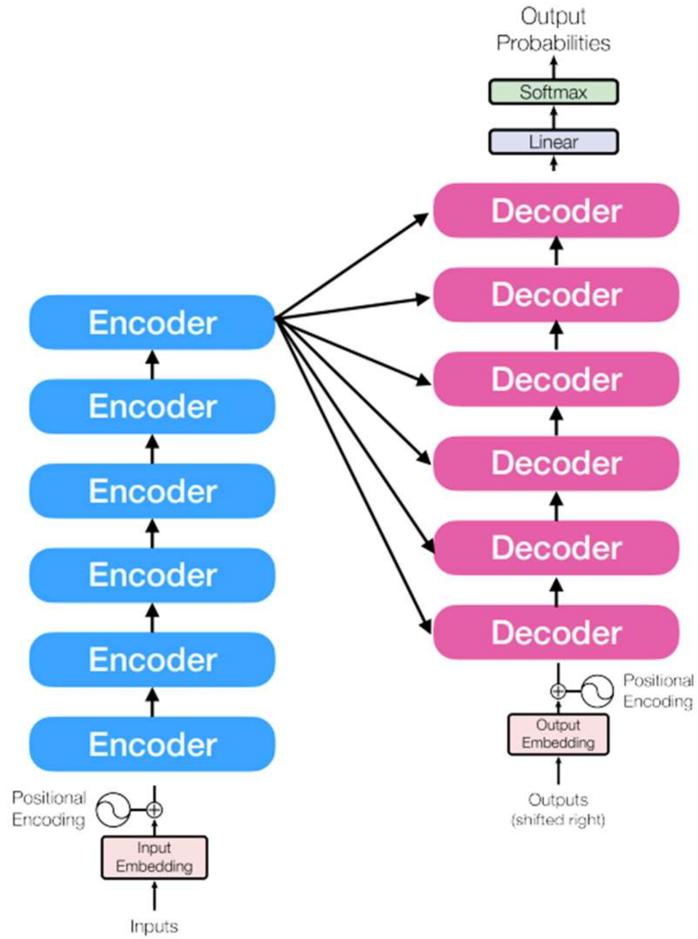
In the original paper there are 6 blocks of **encoder** and 6 blocks of **decoder** networks.

Latent embedding of whole input

Encoder (6X)

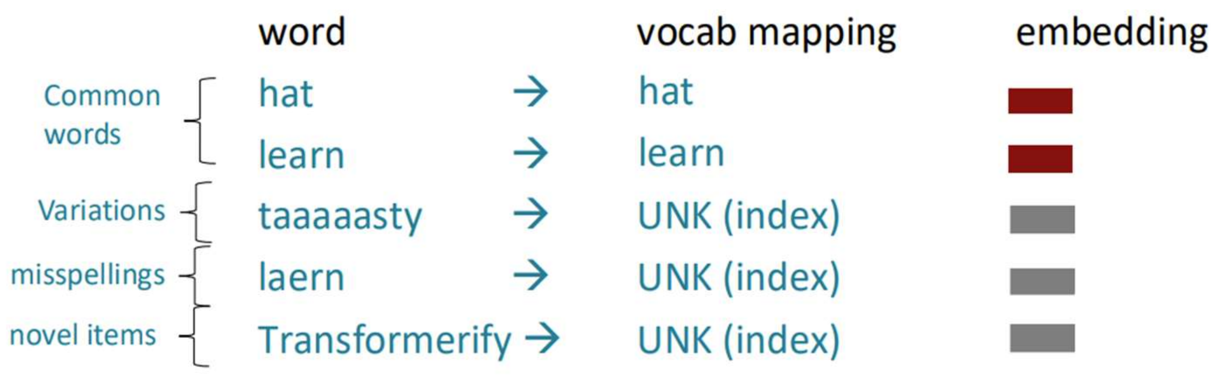
Decoder (6X)





Sub-word Modelling

Let's take a look at the assumptions we've made about a language's vocabulary. We assume a fixed vocab of tens of thousands of words, built from the training set. All *novel* words seen at test time are mapped to a single token **<UNK>**



Sub-word modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.) • The dominant modern paradigm is to learn a vocabulary of parts of words (sub-word tokens). • At training and testing time, each word is split into a sequence of known sub-words.

Byte-pair encoding (BPE) is a simple, effective strategy for defining a sub-word vocabulary.

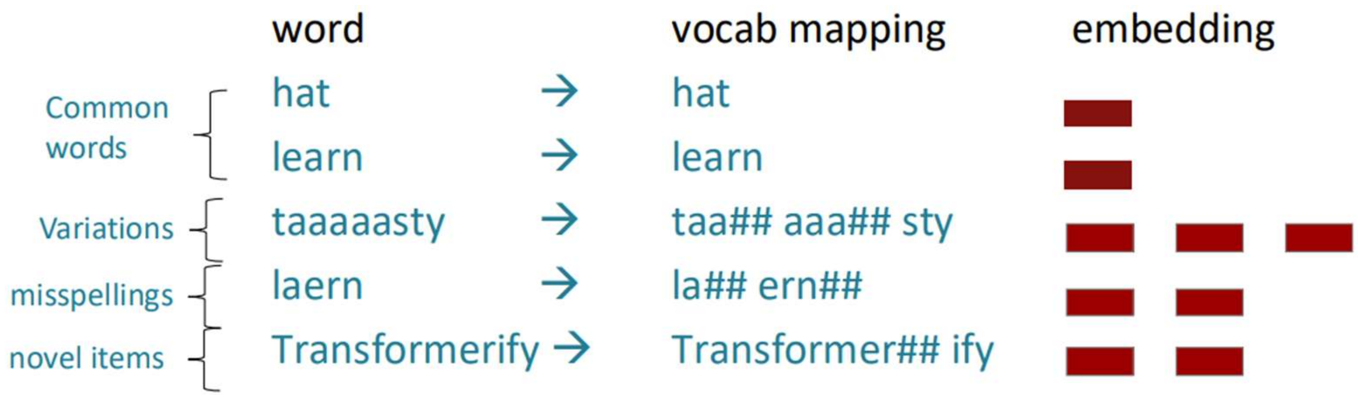
1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
2. Using a corpus of text, find the most common adjacent characters "a,b"; add "ab" as a sub-word.
3. Replace instances of the character pair with the new sub-word; repeat until desired vocab size.

Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

Sub-word Modelling

Common words end up being a part of the sub-word vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, words are split into as many sub-words as they have characters.



Before BPE there were two extreme approaches of tokenization:

1. Word-Level, where each word is a token. This results in a huge vocabulary (~hundreds of thousands words) and rare/unseen words (out of vocabulary)
2. Character-level, where each character is a token. This results in a small vocab, but sequences become very long (harder to learn long term dependencies)

Sub-word Modelling

Let's now take a simplified example of How BPE Works?

- Let's say our training corpus contains these words: low, lower, newest, widest
- Step-1: Start with characters as initial vocabulary. **Vocabulary:** [l, o, w, e, r, n, e, w, s, t, i, d]
  Each word is represented as characters with an end marker _.
  l o w _
  l o w e r _
  n e w e s t _
  w i d e s t _
- Step-2: Count how often the adjacent symbols appear in the corpus: Ex: {(l , o): 2 times, (o, w): 2 times, ... }
- Step-3: Merge the most frequent pairs, ex. Let's merge (l, o) to a new symbol 'lo'. Next merge (lo, w) to low etc.
- Step-4: Repeat the merging process till you get desired number of tokens in vocab. The learnt sub-words in this example could be: ["low", "er", "new", "est", "wide", "st"].

The BPE helps to:

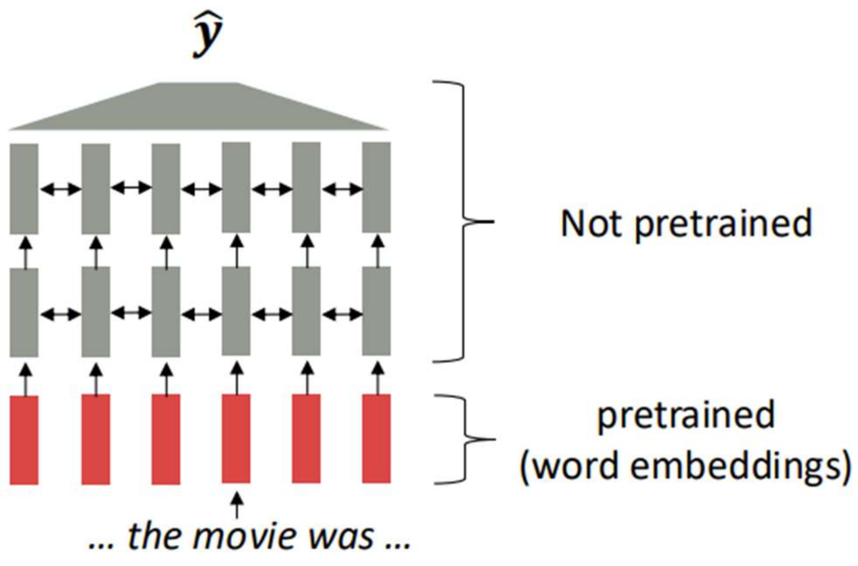
1. Reduce the vocab size (by choosing a middle ground between two extremes of tokenization). In the original transformer paper, the vocab size was around 32000 tokens using BPE.
2. It eliminates "unknown" words. Any new word can be decomposed into known sub-words (e.g. "tranformerization" can be broken down to "transformer" + "ization")

Pretraining: Motivation

Where we were: pretrained word embeddings

Before transformer architecture was popular, to solve NLP problems

- We start with pretrained word embeddings.
- Learn how to incorporate context in an LSTM network while training on the task.
- Sometimes we also learn the word embeddings via embedding layer.



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

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!

Pretraining: Motivation

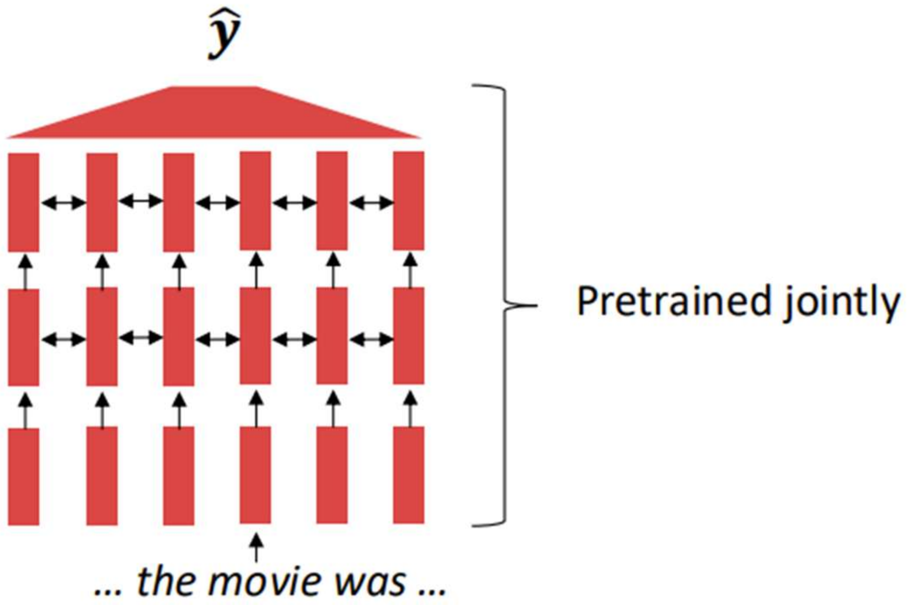
Where we're going: pretraining whole model

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:

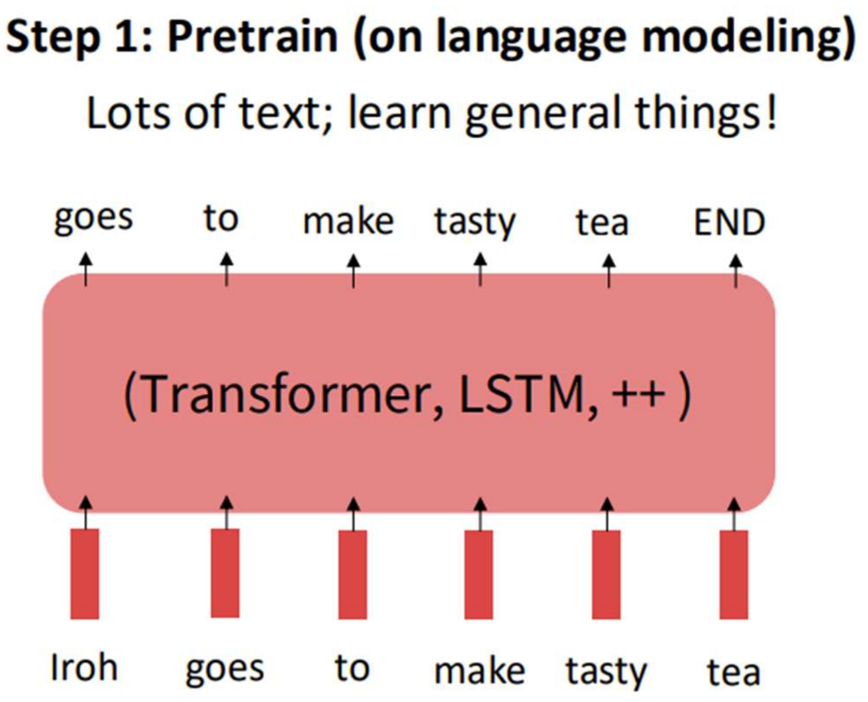
- representations of language.
- parameter initializations for strong NLP models.
- Probability distributions over language that we can sample from



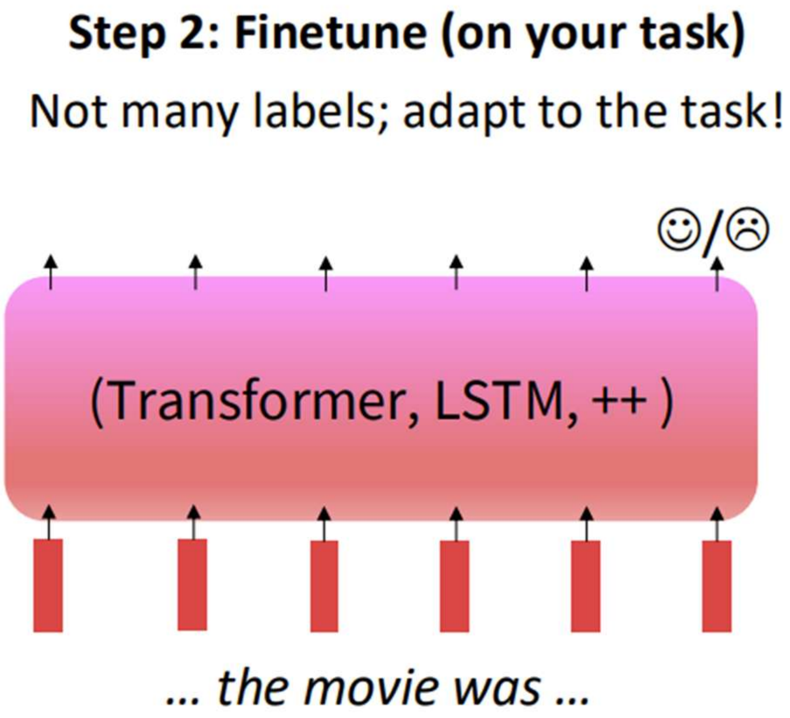
[This model has learned how to represent entire sentences through pretraining]

The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.



Pre-training on large unlabeled datasets
(Self-supervised learning)

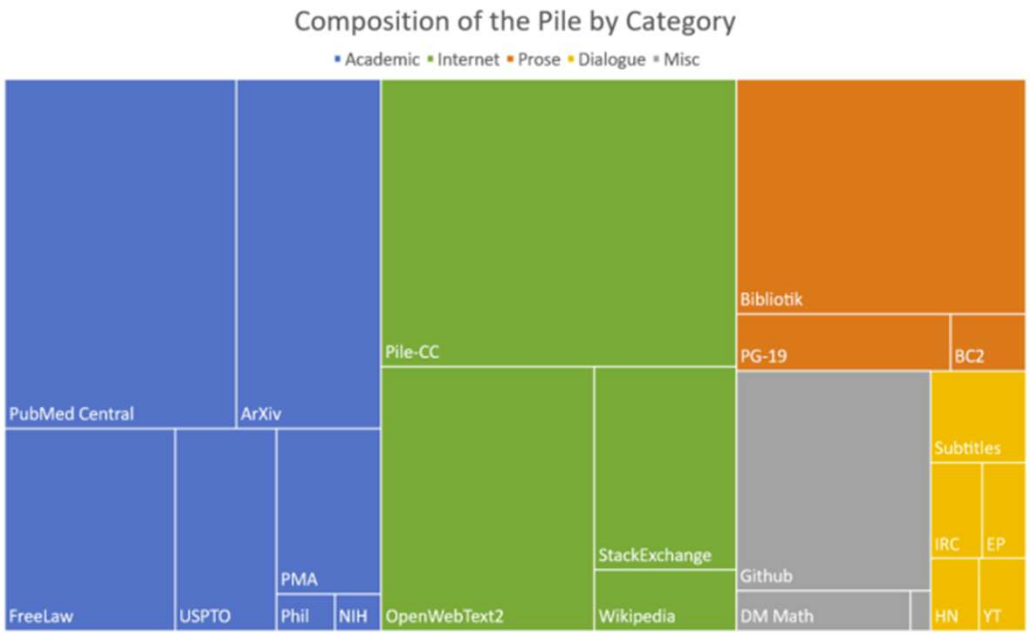


Training for downstream tasks on labeled data
(supervised learning)

Pretraining

Pretraining can be massively diverse

It’s not just about the quantity, but also the incredible diversity of internet text data.



[Gao+ 20]

Source	Doc Type	UTF-8 bytes (GB)	Documents (millions)	Unicode words (billions)	Llama tokens (billions)
Common Crawl	web pages	9,812	3,734	1,928	2,479
GitHub	code	1,043	210	260	411
Reddit	social media	339	377	72	89
Semantic Scholar	papers	268	38.8	50	70
Project Gutenberg	books	20.4	0.056	4.0	6.0
Wikipedia, Wikibooks	encyclopedic	16.2	6.2	3.7	4.3
Total		**11,519**	**4,367**	**2,318**	**3,059**

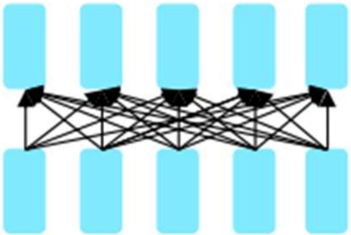
[Soldani+ 24]

The Pile: An 800GB Dataset of Diverse Text for Language Modeling by Gao et. al.

Dolma: an Open Corpus of Three Trillion Tokens for Language Model Pretraining Research by Soldani et. al.

Three types of Architectures

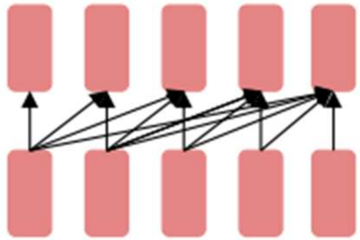
The neural architecture influences the type of pretraining, and natural use cases.



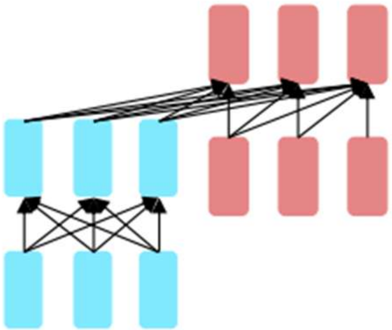
Encoders

- Gets bidirectional context – can condition on future!
- We train them to build strong representations / embeddings. This embeddings are contextual.
- Examples: **BERT** and its variants like **RoBERTa**, **DistilBERT** etc.
- Best suited for: text classification, named entity recognition, sematic similarity etc.
- Not suitable for generating new text.

- Language models, what we've seen so far.
- It is used to generate new texts. It can't be conditioned on future words.
- Examples: modern day LLMs that we use like GPT, CLAUDE, LLaMA etc. all are based on decoder only transformer architecture.



Decoders



Encoder-Decoders

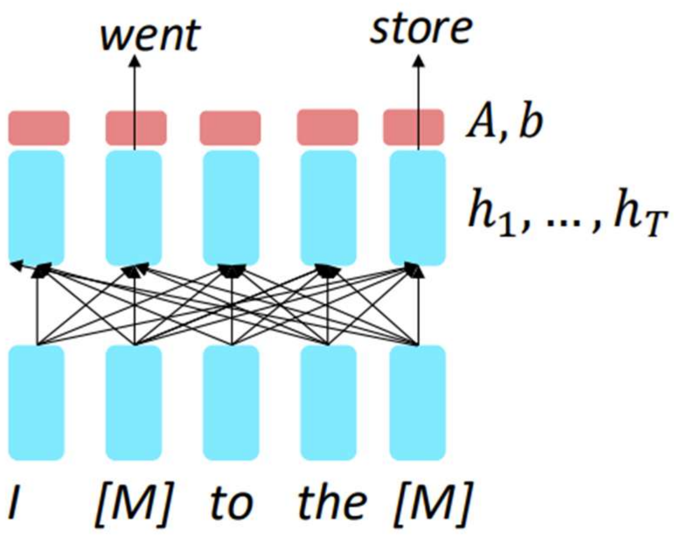
- It takes the best of both worlds (encoder-only and decoder-only)
- Used for various sequence to sequence tasks (for example machine translation, text summarization, paraphrasing etc.)
- Examples: T5, BART etc.

Pretraining Encoder Only Transformers

So far, we've looked at language model pretraining. In case of Language Model pretraining the objective is to predict the next token. 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.

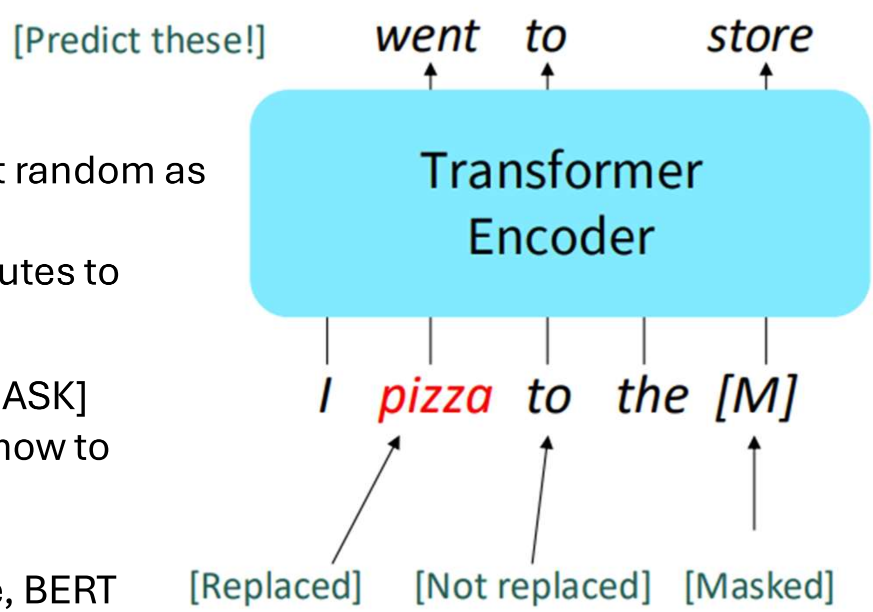
- Only add loss terms from words that are "masked out".
- If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$.
- This methods is called **Masked Language Modelling** (MLM).



Devlin et al., 2018 proposed the "Masked LM" objective and released the weights of a pretrained Transformer, a model they labeled **BERT: Bidirectional Encoder Representations from Transformers**.

BERT

BERT Pretraining Task-1: Masked LM for BERT



- Predict a random 15% of (sub)word tokens.
  - From every input sequence, 15% of the tokens are selected at random as the prediction target.
  - These tokens are the only ones whose prediction loss contributes to training.
- Of those selected 15% tokens, **80%** are replaced by the special [MASK] token. This teaches the model what the [MASK] token means and how to infer missing words using surrounding context.
- To avoid the model *overfitting* to the [MASK] token, **10%** of the time, BERT replaces the selected token with a *random* word from the vocabulary.
  - This prevents the model from simply memorizing that whenever [MASK] appears, it should "predict a missing word."
    It must learn *contextual relationships* even when the corruption isn't marked explicitly.
- In **10%** of the cases, BERT *does not change the token* at all, but still includes it as a prediction target. This further ensures that BERT doesn't rely solely on [MASK] positions to decide what to predict.

BERT

Putting it all together

For every 200 tokens:

- 30 tokens are chosen for prediction.
  - 24 of those (80%) → replaced with `[MASK]`
  - 3 (10%) → replaced with random words
  - 3 (10%) → left unchanged

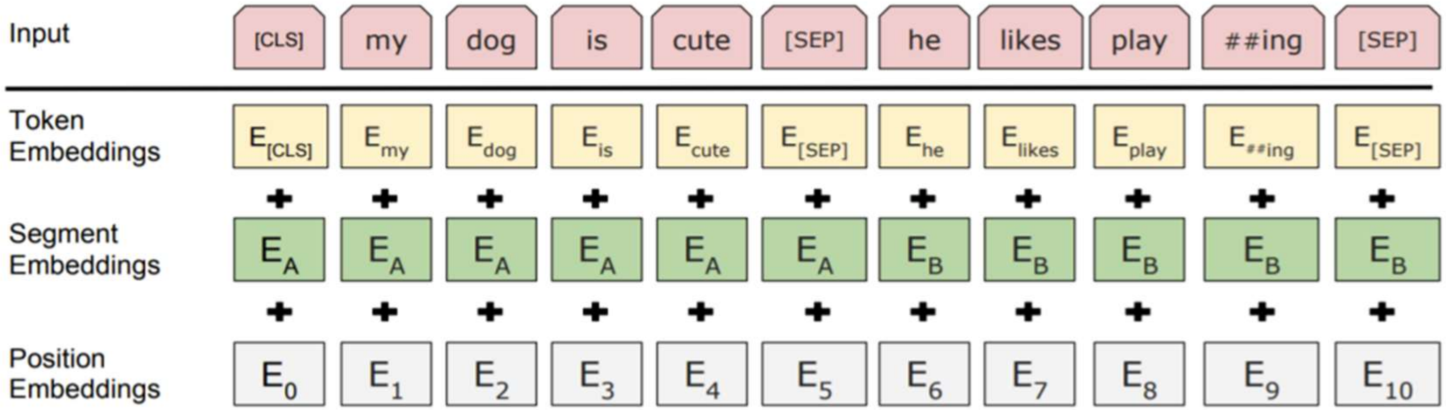
BERT must then **predict the original token** for all 15 positions using the *bidirectional context* of the sentence.
No masks are seen at fine-tuning time!

Why this design matters

- Real downstream tasks **don't contain `[MASK]` tokens** — so forcing the model to also predict unmasked and random tokens helps bridge the gap between pretraining and fine-tuning.
- The combination of these three replacements teaches BERT to be:
  - **Context-aware** (thanks to bidirectionality),
  - **Robust** to noise and corruption,
  - **Flexible** in understanding natural language.

BERT

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



BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

BERT Pretraining Task-2: Next Sentence Prediction

Balanced binary classification task (50% **IsNext** and 50% **NotNext**)

Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

BERT

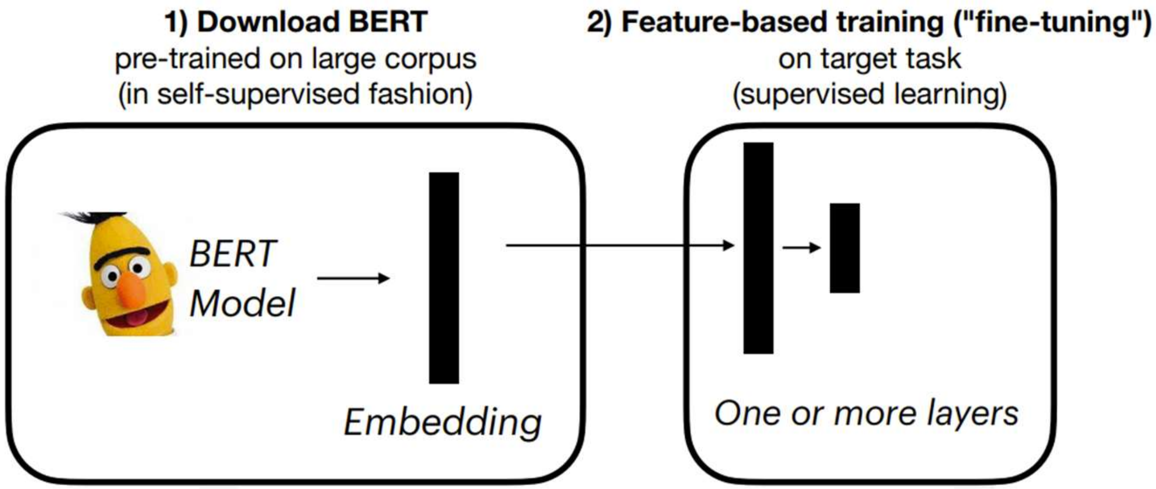
Details about BERT

- 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."

There are lots of different variants of BERT:

- RoBERTa (Robust, generalizes better)
- DistilBERT (memory efficient)
- SpanBERT (good at predicting continuous spans of texts such as multi-word phrases)

Etc.



Pretraining Decoder Only Transformers

- Decoder are the Language Models i.e. they learn to predict the next token. i.e. it is used to model $p_\theta(w_t \mid w_{1:t-1})$.
- It's natural to pretrain decoders as language models and then use them as generators. Hence, these are also called Generative Models.
- The present-day generative language models are decoder only language model. Which are good autoregressive (1-word-at-a-time) models.

GPT -1 : Generative Pretrained transformers

2018's GPT was a big success in pretraining a decoder! Developed by Radford et. al. at OpenAI in the year 2018.

- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with vocab size around 30000.
- Trained on BooksCorpus: over 7000 unique books.
- Contains long spans of contiguous text, for learning long-distance dependencies.

GPT-1

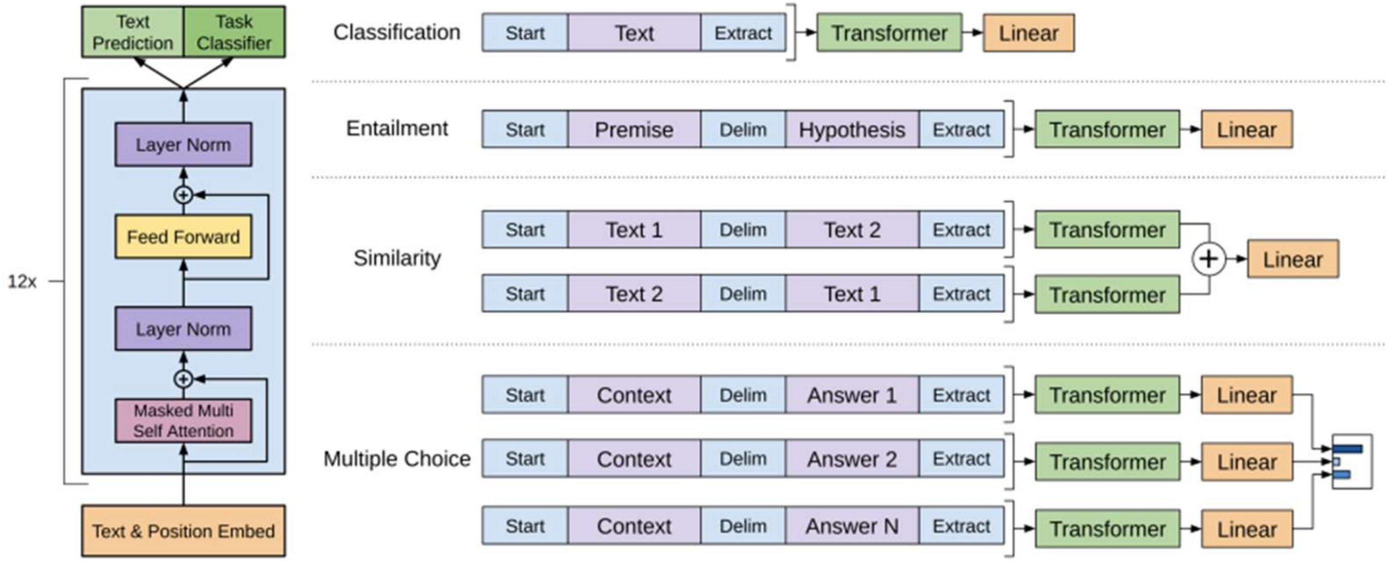


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

2-step training process ("semi-supervised")

1. Generative pre-training (on unlabeled data); unsupervised/"self-supervised" learning
2. Discriminative fine-tuning (on labeled data), supervised learning

GPT-1

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.

GPT-2

GPT-2 is a **1.5 Billion parameters** language model developed by OpenAI in the year 2019 by Radford et. al.

GPT-2 demonstrates the concept that Language Models are Unsupervised Multi-task Learners. i.e. It means that large language models learn from **unlabeled text (unsupervised)** by predicting the next word, and through this single training objective, they **implicitly learn to perform many different tasks** (translation, summarization, reasoning, etc.) without explicit task-specific supervision.

Key architecture:

- Overall, similar to GPT-1 (which is based on original Transformer decoder)
- Some small rearranging of layer norm and residual layers
- Increase vocabulary size from 30,000 to 50,257
- Increase context size from 512 to 1024 tokens
- Overall, 1.5 billion instead of 117 million parameters

Training:

- WebText (millions of webpages)
- Emphasized on data quality
- 8 million documents.
- In contrast to GPT-1, no specific instruction / rearranging for specific tasks

GPT-3

GPT-3 is a **175 Billion parameters** language model developed by OpenAI in the year 2020 by Brown et. al.

GPT-3 demonstrates the concept that Language Models are Few-shot Learners. i.e. In GPT-3, the model can **perform new tasks from just a few examples given in the prompt**—without additional training.
This shows that GPT-3 has **learned generalized language patterns** during pretraining, enabling *few-shot learning* through in-context examples.

Key architecture:

- Overall, similar to GPT-2
- 175 billion instead 1.5 billion parameters in GPT-2 (because GPT-3 has more layers, larger context size etc.)
- Double the context size (2048 instead of 1024)
- Larger word embeddings (12.8k instead of 1.6k)

Training

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

GPT-3

Implicit Task Learning (In context Learning)

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

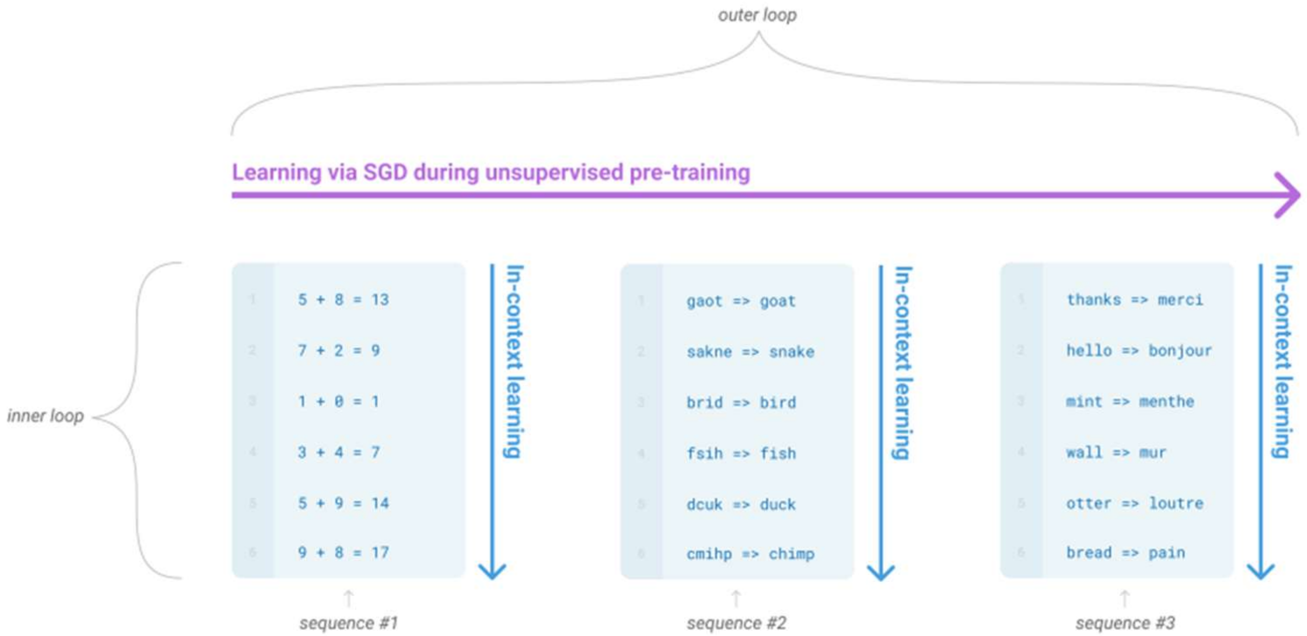


Figure 1.1: Language model meta-learning. During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term "in-context learning" to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

GPT-3: Showing Examples vs Fine Tuning

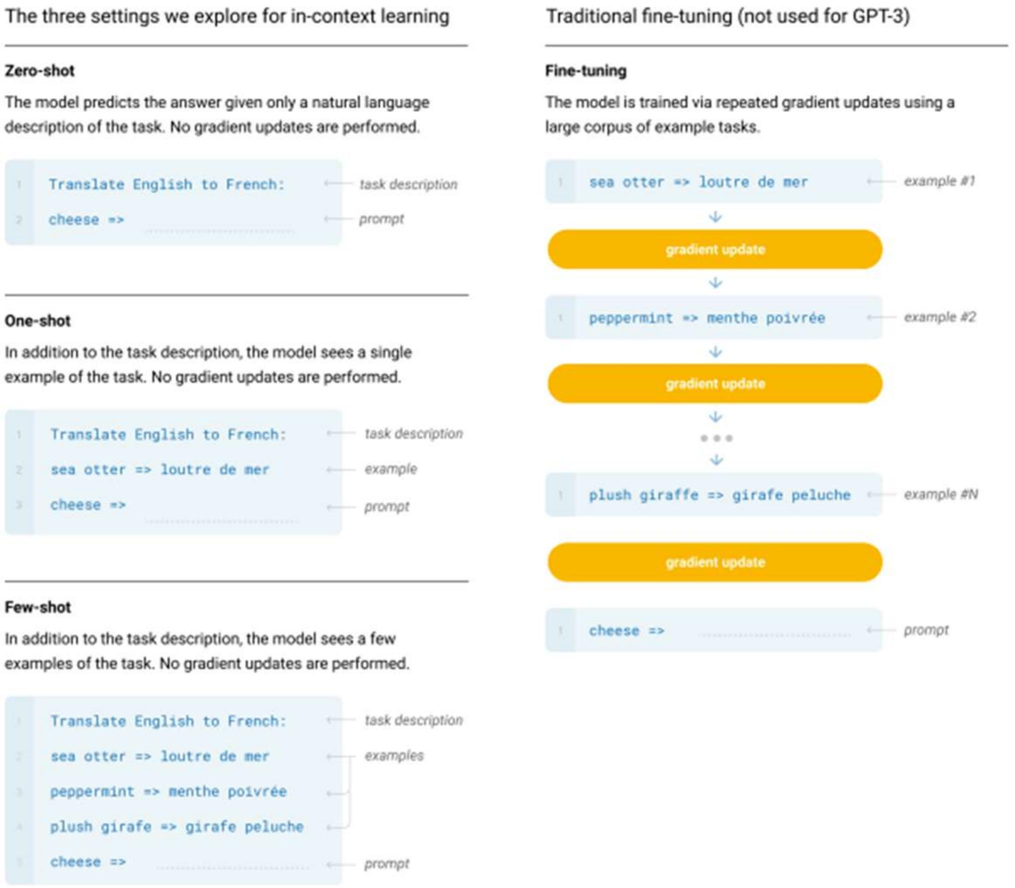
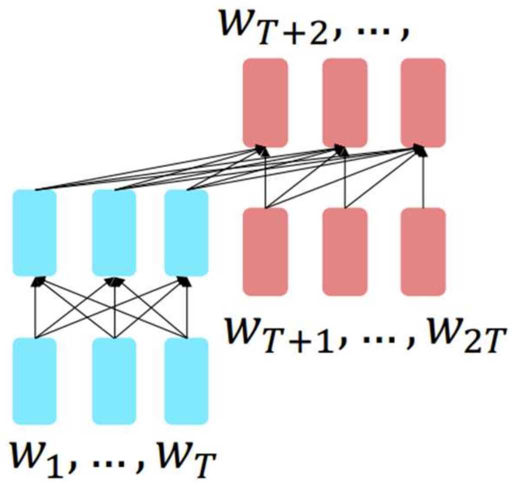


Figure 2.1: Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning. The panels above show four methods for performing a task with a language model – fine-tuning is the traditional method, whereas zero-, one-, and few-shot, which we study in this work, require the model to perform the task with only forward passes at test time. We typically present the model with a few dozen examples in the few shot setting. Exact phrasings for all task descriptions, examples and prompts can be found in Appendix G.

Pretraining Encoder-Decoder Transformers

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



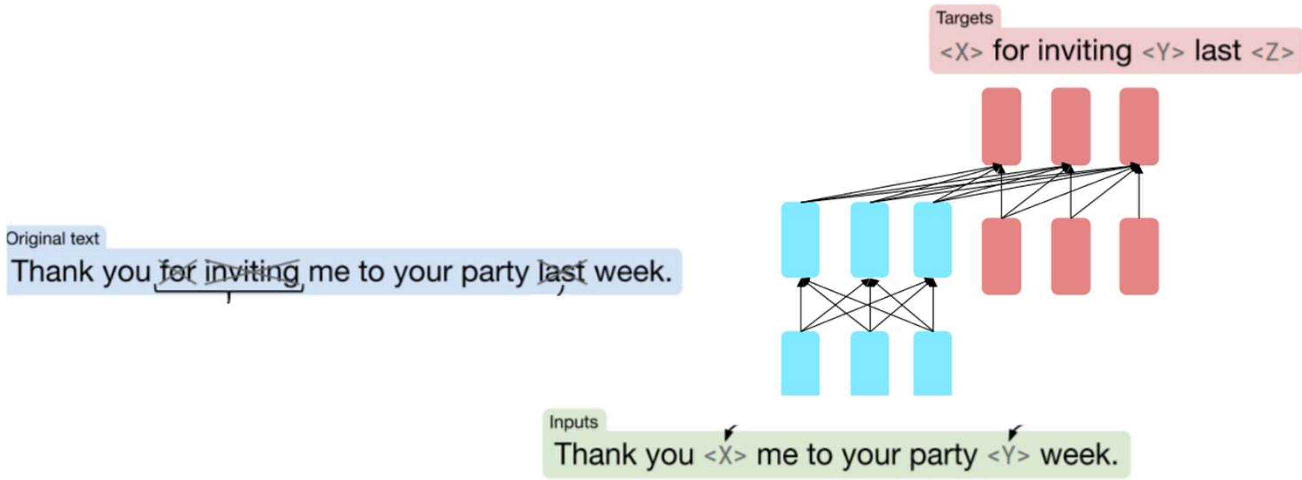
$$h_1, h_2, \ldots, h_T = Encoder(w_1, w_2, \ldots, w_T)$$

$$h_{T+1}, h_{T+2}, \ldots, h_{2T} = Decoder\,(w_1, w_2, \ldots, w_T, h_1, \ldots, h_T)$$

What Raffel et al., 2019 found to work best was span corruption. Their model is called **T5:** Text to Text Transfer Transformer.

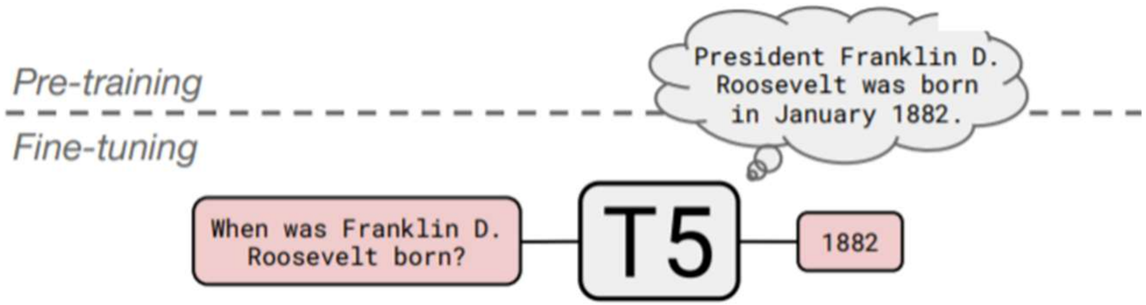
Span corruption: Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



Text to Text Transfer Transformer (T5)

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



Performance of T5:

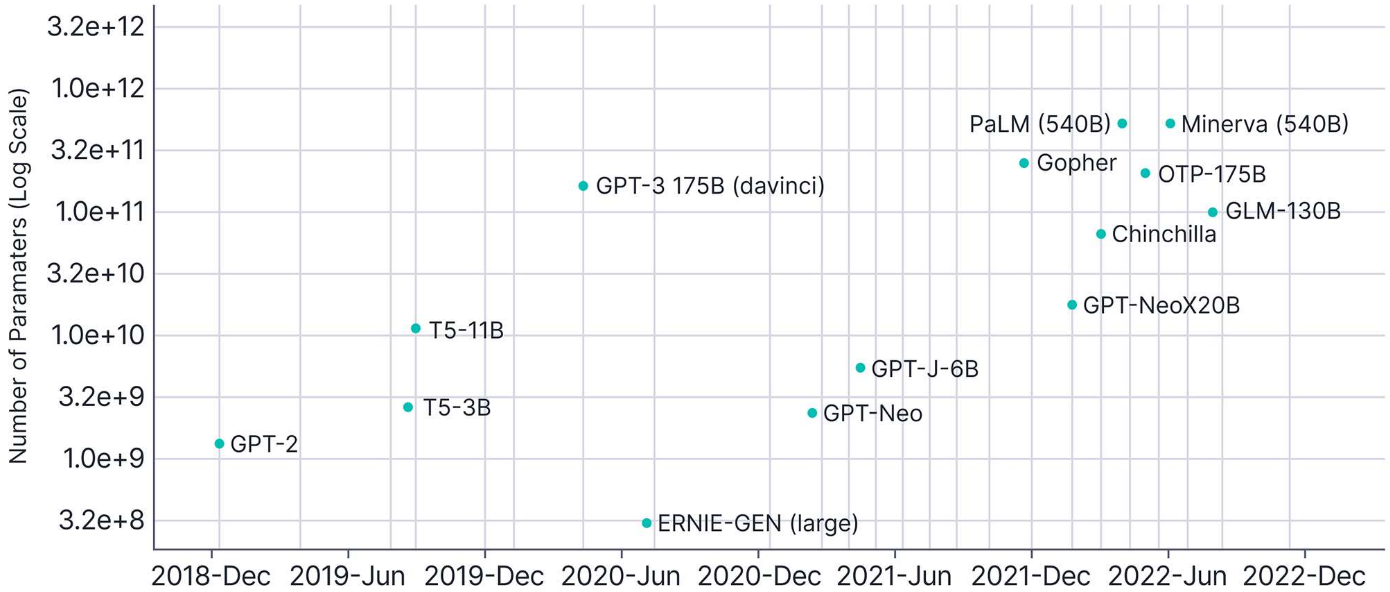
	NQ	WQ	TQA		
			dev	test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

NQ: Natural Questions
WQ: WebQuestions
TQA: Trivia QA

Large Language Models

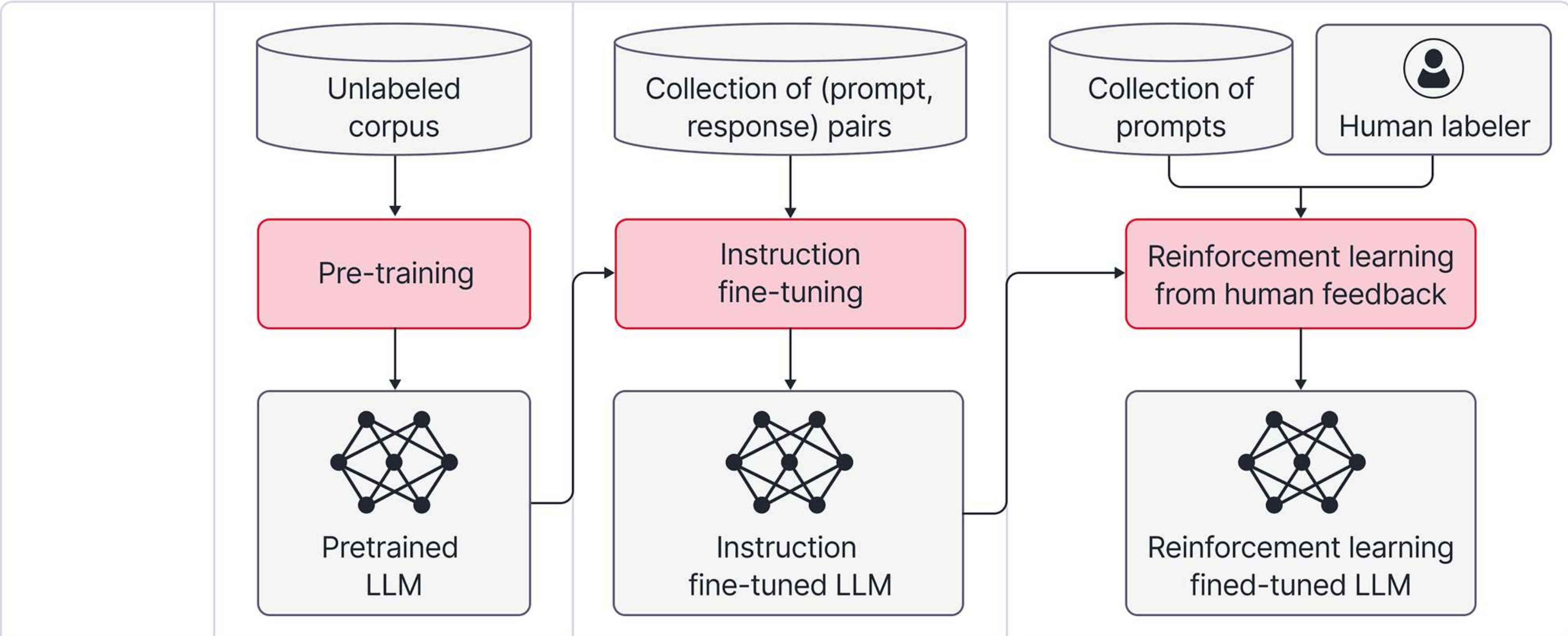
What is a Large Language Model (LLM)?

- A *language model* is built to process and understand a text input *(prompt)*, and then generate a text output *(response)* accordingly.
- These models are usually trained on an extensive corpus of unlabeled text, allowing them to learn general linguistic patterns and acquire a wide knowledge base.
- The primary distinction between a *regular language model* and a *large language model* lies in the number of parameters used. While there is no universally agreed-upon definition, a good rule of thumb proposed by Zhao et al. (2023) is that LLMs should have a minimum of ten billion parameters.



Number of Parameters of selected Large Language models over time.

How to train an LLM



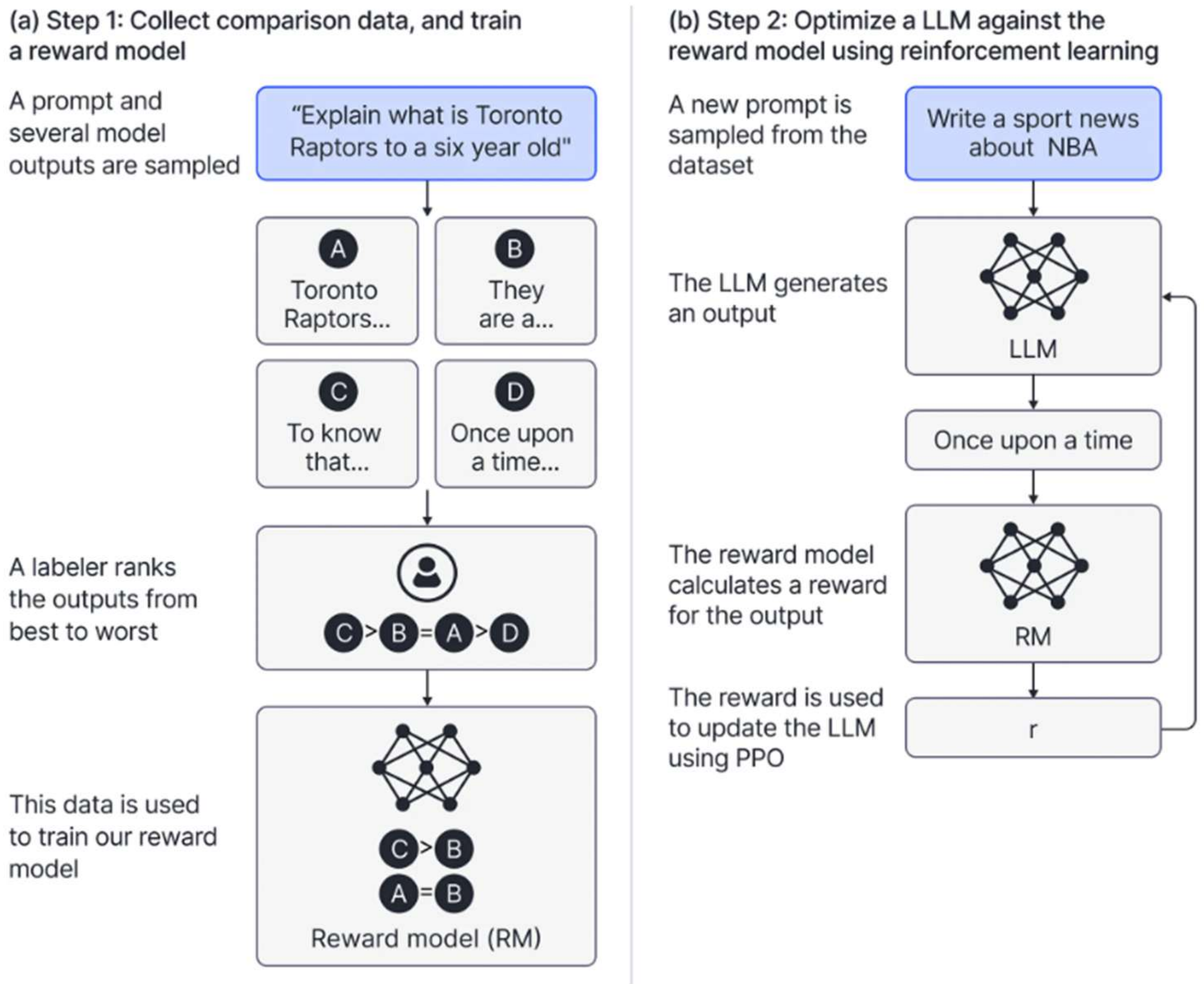
Dataset size	x00 billions to 1.x trillion tokens	~xk to x0k (prompt, response)	~x0k prompt
Example of models	GPT-3, LLaMA, Falcon, BLOOM	Dolly-v2, Falcon-instruct	Claude, GPT-4, ChatGPT
	(a)Pre-training	(b) Instruction fine-tuning	(c) Reinforcement learning from human feedback

Instruction Finetuning

Open datasets available for Instruction Finetuning

Dataset Name	Size	Language	Description	Source
Dolly 2.0 Dataset	15,000 prompts	English	A human-generated instruction dataset designed for training instruction-following LLMs.	Databricks Blog
Alpaca Dataset	52,000 prompts	English	Generated using OpenAI's text-davinci-003; tailored for instruction-following training.	Stanford CRFM
FLAN Collection	1.8M examples	Multilingual	A diverse set of tasks and instructions aimed at enhancing model generalization across tasks.	Google Research
OpenOrca Dataset	3.3M examples	English	Combines multiple datasets to improve reasoning capabilities and instruction-following in LLMs.	OpenOrca

RLHF: Basic Idea



Which LLM is better

There is no single model which outperforms other in all the tasks. This field is rapidly evolving and new models are developed at rapid pace.

Following table shows the leaderboard of different language models in lmarena leaderboard.

Leaderboard Overview

See how leading models stack up across text, image, vision, and beyond. This page gives you a snapshot of each Arena, you can explore deeper insights in their dedicated tabs. Learn more about it here.

Text

5 days ago

Rank (UB)	Model	Score	Votes
1	gemini-2.5-pro	1451	54,087
1	claude-opus-4-1-20250805-thi…	1447	21,306
1	claude-sonnet-4-5-20250929-t…	1445	6,287
1	gpt-4.5-preview-2025-02-27	1441	14,644
2	chatgpt-4o-latest-20250326	1440	40,013
2	o3-2025-04-16	1440	51,293
2	claude-sonnet-4-5-20250929	1438	6,144
2	gpt-5-high	1437	23,580

WebDev

1 day ago

Rank (UB)	Model	Score	Votes
1	GPT-5 (high)	1478	5,848
1	Claude Opus 4.1 thinking-16k…	1472	5,312
1	Claude Opus 4.1 (20250805)	1462	5,582
4	Claude Sonnet 4.5 (thinking …	1421	1,337
4	Gemini-2.5-Pro	1401	11,022
4	GLM-4.6	1398	5,442
4	DeepSeek-R1-0528	1394	4,800
5	Claude Sonnet 4.5	1385	4,127

Thank You