ELEG 5491: Introduction to Deep Learning

Basics of Large Language Models

Prof. LI Hongsheng

Office: SHB 428
e-mail: hsli@ee.cuhk.edu.hk
web: https://dl.ee.cuhk.edu.hk

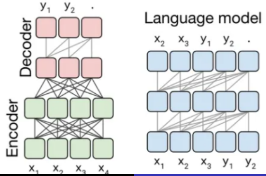
Department of Electronic Engineering
The Chinese University of Hong Kong

March 2023

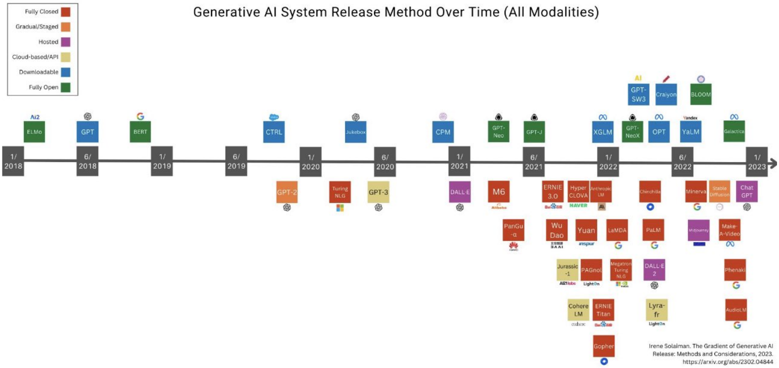
Large Language Models

- Large Language Models (LLM) received increasing attentions in recent years
- Many Natural Language Processing (NLP) tasks have been greatly improved
- Pretrainig is widely used to train LLM and are finetuned on downstream NLP tasks (such as summarization, QA, etc.)

- **BERT** (Bidirectional Encoder Representations from Transformer) series, **T5** (Text-to-Text Transfer Transformer), and **GPT** (Generative Pretrained Transformer) series are representative pertrained LLM
- **Three types of architectures:** 1) encoder-only (BERT), 2) encoder-decoder (T5), 3) decoder-only (GPT)
- **Two training strategies:** 1) pretraining-finetuing (BERT & T5), and 2) pure generative (GPT)



Timeline of Generative Large Language Models



Timeline of Generative Large Language Models

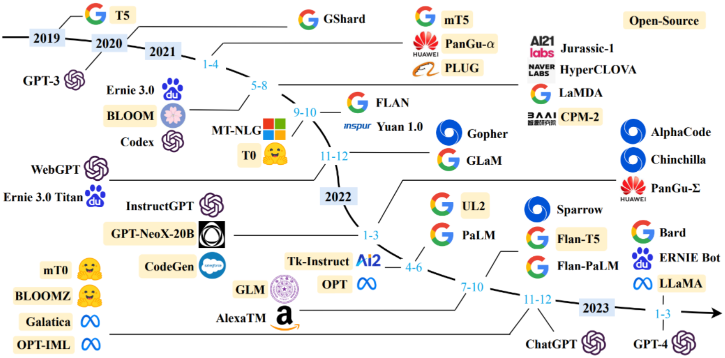


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

WordPiece Tokenizer

- BERT uses WordPiece Tokenizer split words either into the full forms
- For instance, given the following words and their frequencies in the training set

  ("hug", 10), ("pug", 5), ("pun",12), ("bun", 4), ("hugs", 5)

- The initial splitting will be

  ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)

- The pair of wordpieces are gradually merged. Instead of selecting the most frequent pair, WordPiece computes a score for each pair as follows

$$\text{score} = \frac{\text{frequency of pair}}{\text{frequency of first element} \times \text{frequency of second element}}$$

- The most frequent pair is ("##u", "##g") (20 times), but "##u" is very high so its score it not high
- Instead, the first merge is ("##g", "##s") $\rightarrow$ ("##gs")

WordPiece Tokenizer

- Add "##gs" to the vocabulary and apply the merge in the words of the corpus

Corpus

Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs"]

Corpus: ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##gs", 5)

- The next merge ("h", "##u") $\rightarrow$ "hu" (all pairs of X + "##u" end up the same score)

Corpus

Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu"]

Corpus: ("hu" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)

WordPiece Tokenizer

- The next merge is `("hu", "##g")`

Corpus

Vocabulary: `["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]`

Corpus: `("hug", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)`

- Given the vocabulary, WordPiece finds the longest subword that is in the vocabulary, then splits on it

`"hugs"` → `["hug" "##s"]`
`"bugs"` → `["b" "##u "##gs"]`

- When the tokenization gets to a stage where it's not possible to find a subword in the vocabulary, the whole word is tokenized as unknown `"[UNK]"`

`"bums"` → `["b" "##u" "[UNK]"]`

WordPiece Tokenizer

- corpus = ["This is the Hugging Face Course.", "This chapter is about tokenization.", "This section shows several tokenizer algorithms.", "Hopefully, you will be able to understand how they are trained and generate tokens."]
- Initial vocabulary = ['##a', '##b', '##c', '##d', '##e', '##f', '##g', '##h', '##i', '##k', '##l', '##m', '##n', '##o', '##p', '##r', '##s', '##t', '##u', '##v', '##w', '##y', '##z', ',', '.', 'C', 'F', 'H', 'T', 'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y']
- After WordPiece tokenization, Vocabulary = [['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]', '##a', '##b', '##c', '##d', '##e', '##f', '##g', '##h', '##i', '##k', '##l', '##m', '##n', '##o', '##p', '##r', '##s', '##t', '##u', '##v', '##w', '##y', '##z', ',', '.', 'C', 'F', 'H', 'T', 'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab', '##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully', 'Th', 'ch', '##hm', 'cha', 'chap', 'chapt', '##thm', 'Hu', 'Hug', 'Hugg', 'sh', 'th', 'is', '##thms', '##za', '##zat', '##ut']]

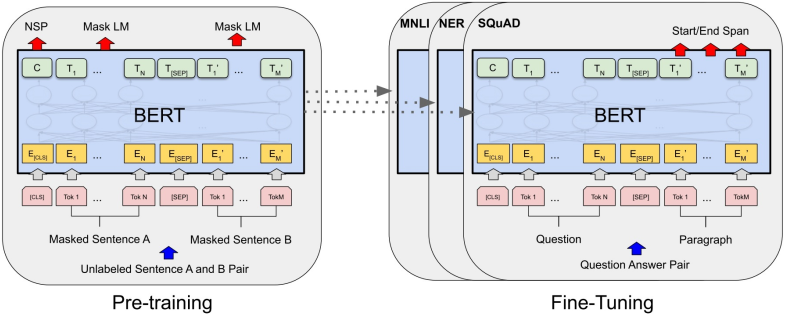
WordPiece Tokenizer

```
"This is the Hugging Face course!"
↓
['Th', '##i', '##s', 'is', 'th', '##e', 'Hugg', '##i', '##n',
'##g', 'Fac', '##e', 'c', '##o', '##u', '##r', '##s', '##e',
'[UNK]']
```

- Introducing WordPiece tokens can effectively mitigate the problem of Out of Vocabulary (VVO)

BERT: Bidirectional Encoder Representations from Transformers

- BERT pretrains a bidirectional transformer with two tasks
  - Masked Language Modeling (MLM)
  - Next Sentence Prediction (NSP)
- When given a new task, the same network architecture is finetuned on new tasks

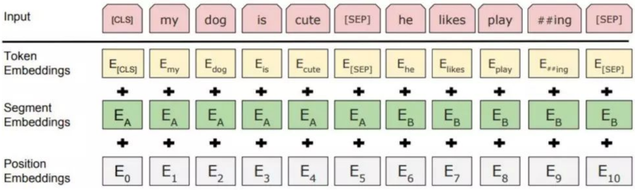


BERT: Bidirectional Encoder Representations from Transformers

- BERT adopts a multi-layer bi-directional transformer architecture
- It adopts two scales in the original paper:

$$BERT_{BASE}: \mathrm{L} = 12, \mathrm{H} = 768, \ \ \mathrm{A} = 12, \text{ Total Parameters} = 110\mathrm{M}$$
$$BERT_{LARGE}: \mathrm{L} = 24, \mathrm{H} = 1024, \ \ \mathrm{A} = 16, \text{ Total Parameters} = 340\mathrm{M}$$

- L - number of transformer layers, H - hidden dimension, A - number of attention heads. FFN has $4H$ dimensions.
- Input Representation:



- WordPiece tokens from a 30,000-token vocabulary + learnable positional embeddings supporting a maximum of 512 input tokens
- A special "[CLS]" token at the beginning for classification-based tasks, which is not used for non-classification tasks

BERT: Bidirectional Encoder Representations from Transformers

- Two ways to distinguish sentences: 1) "[SEP]" token to separate multiple input sentences; 2) learnable sentence tokens E_A and E_B
- For single-sentence input, only E_A is used. Pretraining tasks is the key novelty.
- **Masked Language Modeling (MLM).** Mask out 15% of tokens and replace them as [MASK] tokens in the input sequence and require the transformer to reconstruct the original words/tokens at the topmost layer
- However, there is pretrain-finetune inconsistency: one will never see a [MASK] token during finetuing
- Dataloader
  - 80% inputs use [MASK] to replace 15% words, e.g., `my dog is hairy` $\rightarrow$ `my dog is [MASK]`
  - 10% inputs use random words to replace 15% words, e.g., `my dog is hairy` $\rightarrow$ `my dog is apple`
  - 10% inputs keep the words unchanged, e.g., `my dog is hairy` $\rightarrow$ `my dog is hairy`
- The transformer encoder doesn't know which words have been replaced and is therefore forced to encode context of each token. Only $15\% \times 10\% = 1.5\%$ words are randomly replaced, which doesn't affect the context encoding

BERT: Bidirectional Encoder Representations from Transformers

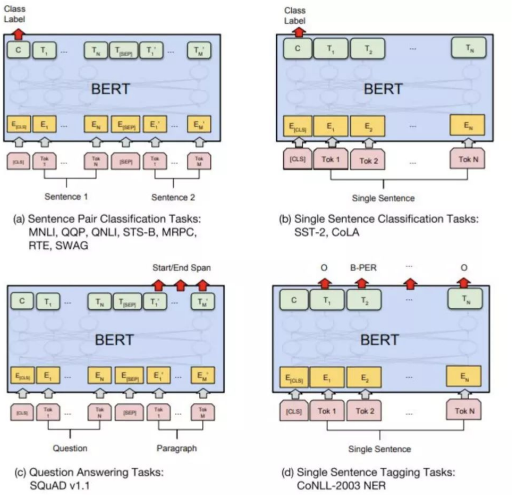
- **Next Sentence Prediction (NSP):** Question Answering (QA) and Natural Language Inference (NLI) require understanding inter-sentence relations
- Two masked sentences A and B are input into the BERT model.
- 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext)
- During pretraining, the `[CLS]` token is used to perform binary classification IsNext or NotNext
- Examples

```
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: Bidirectional Encoder Representations from Transformers

- BERT has shown impressive improvements on various NLP tasks



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

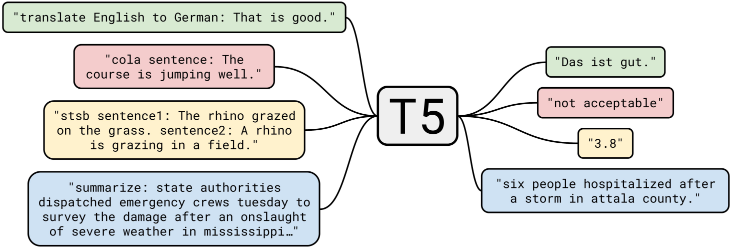
(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

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

T5: Transfer Text-to-Text Transformer

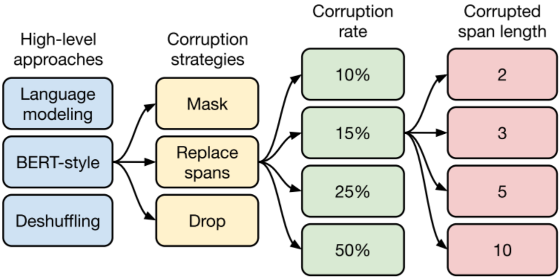
- T5 also adopts the pretraining-finetuning paradigm but it converts all NLP tasks into text-to-text tasks



- Even for regression tasks, it also directly outputs texts. For instance, for Semantic Textual Similarity Benchmark task, it outputs scores with an interval of 0.2 in the range of $[1, 5]$
- For architecture, T5 adopts the architecture of bidirectional encoder $+$ masked-attention decoder

Extensive Search of the Pretraining Task

- T5 searches the optimal unsupervised objectives

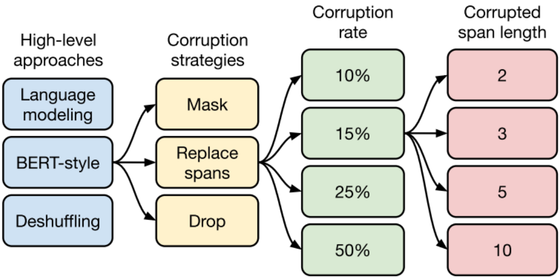


- High-level objectives: BERT-style is optimal

Prefix language modeling	Thank you for inviting	me to your party last week
BERT-style	Thank you $\langle M \rangle \langle M \rangle$ me to your party apple week.	(original text)
Deshuffling	party me for your to . last fun you inviting week Thank	(original text)

Extensive Search of the Pretraining Task

- T5 search the optimal unsupervised objectives



- Corruption: 1) masking, 2) replacing span, 3) dropping. Replacing span gives the optimal results.

noise, mask tokens	Thank you $\langle M \rangle \langle M \rangle$ me to your party $\langle M \rangle$ week.	(original text)
noise, replace spans	Thank you ⟨X⟩ me to your party ⟨Y⟩ week.	⟨X⟩ for inviting ⟨Y⟩ last ⟨Z⟩
noise, drop tokens	Thank you me to your party week	for inviting last

T5 Pretraining Task

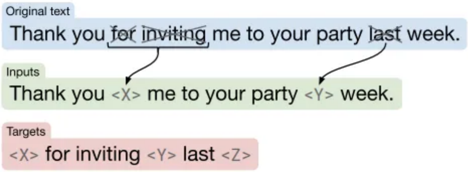
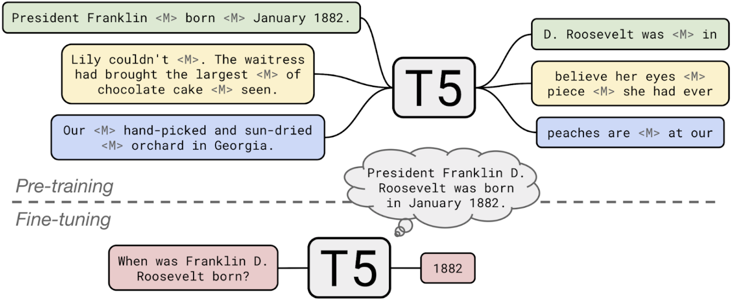


Figure: T5 model's pretraining task of replaceing span.

Pretraining Datasets

- **Common Crawl** corpus contains petabytes of data collected over 12 years of web crawling. The corpus contains raw web page data, metadata extracts and text extracts (20TB raw text data every month)
- The authors clean the data and object **Colossal Clean Crawled Corpus (C4)** dataset
- Clean strategies:
  - Retaining lines that ended in a terminal punctuation mark (i.e. a period, exclamation mark, question mark, or end quotation mark).
  - Discarding any page with fewer than 5 sentences and only retained lines that contained at least 3 words.
  - Removing any page that contained any word on the "List of Dirty, Naughty, Obscene or Otherwise Bad Words"
  - Many of the scraped pages contained warnings stating that Javascript should be enabled so we removed any line with the word Javascript.
  - Remove pages with placeholder "lorem ipsum" text; we removed any page where the phrase "lorem ipsum" appeared
  - Some pages inadvertently contained code. Since the curly bracket "'{" appears in many programming languages (such as Javascript, widely used on the web) but not in natural text, they removed any pages that contained a curly bracket.
  - To deduplicate the data set, they discarded all but one of any three-sentence span occurring more than once in the data set

Pretraining-finetuning



- Unsupervised pre-training is optimal and is comparable with multi-task supervised pretraining and achieved SOTA performances on multiple downstream tasks

Training strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Unsupervised pre-training + fine-tuning	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	39.82	27.65
Multi-task training	81.42	**19.24**	79.78	67.30	25.21	36.30	27.76
Multi-task pre-training + fine-tuning	**83.11**	**19.12**	**80.26**	**71.03**	**27.08**	39.80	**28.07**
Leave-one-out multi-task training	81.98	19.05	79.97	**71.68**	**26.93**	39.79	**27.87**
Supervised multi-task pre-training	79.93	18.96	77.38	65.36	26.81	**40.13**	**28.04**

GPT: Generative Pretrained Transformer

- GPT is a decoder-only and autoregressive (AR) language model, while BERT is an autoencoding language model
- GPT is a left-to-right model that can be used without finetuning while BERT is a bi-directional model that has to be finetuned for generative tasks

Model	Release Time	# Params	Pretrain Data
GPT-1	June 2018	117M	∼5 GB
GPT-2	Feb. 2019	1.5B	40 GB
GPT-3	May 2020	17.5B	45TB

- GPT-1 performs unsupervised pretraining. Given an unlabeled sentence $\mathcal{U} = \{u_1, \ldots, u_n\}$, the objective is to maximize the log likelihood

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

where k is the sliding temporal window size, P denotes the probability, Θ are model parameters, which are optimized via SGD

Byte-Pair Encoding tokenization

- Byte-Pair Encoding (BPE) was initially developed as an algorithm to compress texts
- It is used by quite a few Transformer models, including GPTs, RoBERTa, BART, DeBERTa, etc.
- For generating the vocabular, given the following words with their frequencies

  `("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)`
- The initial splitting is

  `("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)`
- The pair of highest frequencies are iteratively merged and added into the vocabulary until a target size of the vocabulary is achieved
- The first merge is `("u", "g")` $\rightarrow$ `"ug"`

Corpus

`Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]`

`Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)`

Byte-Pair Encoding tokenization

- Next merge is `("u", "n")` $\rightarrow$ `"un"`

Corpus

`Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un"]`
`Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)`

- Next merge is `("h", "ug")` $\rightarrow$ `"hug"`

Corpus

`Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"] Corpus: ("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)`

Byte-Pair Encoding tokenization

- Example

Corpus

```
corpus = [ "This is the Hugging Face Course.",
"This chapter is about tokenization.",
"This section shows several tokenizer algorithms.",
"Hopefully, you will be able to understand how they are trained
and generate tokens." ]
```

- Pre-tokenization

Raw word frequencies

```
{'This': 3, 'Ġis': 2, 'Ġthe': 1, 'ĠHugging': 1, 'ĠFace': 1,
'ĠCourse': 1, '.': 4, 'Ġchapter': 1, 'Ġabout': 1,
'Ġtokenization': 1, 'Ġsection': 1, 'Ġshows': 1, 'Ġseveral':
1, 'Ġtokenizer': 1, 'Ġalgorithms': 1, 'Hopefully': 1, ',': 1,
'Ġyou': 1, 'Ġwill': 1, 'Ġbe': 1, 'Ġable': 1, 'Ġto': 1,
'Ġunderstand': 1, 'Ġhow': 1, 'Ġthey': 1, 'Ġare': 1,
'Ġtrained': 1, 'Ġand': 1, 'Ġgenerate': 1, 'Ġtokens': 1 }
```

Byte-Pair Encoding tokenization

- Initial vocabulary

  {'< |endoftext| >', ',', '.', 'C', 'F', 'H', 'T', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'y', 'z', 'Ġ'}

- Determine the pair to be merged: ('Ġ, 't') → 'Ġt'

  ('T', 'h'): 3, ('h', 'i'): 3, ('i', 's'): 5, ('Ġ', 'i'): 2, ('Ġ', 't'): 7, ('t', 'h'): 3

- Eventually, 19 merge rules are learned

  ('Ġ', 't')→'Ġt', ('i', 's')→'is', ('e', 'r')→'er', ('Ġ', 'a')→'Ġa', ('Ġt', 'o')→'Ġto', ('e', 'n')→'en', ('T', 'h')→'Th', ('Th', 'is')→'This', ('o', 'u')→'ou', ('s', 'e')→'se', ('Ġto', 'k')→'Ġtok', ('Ġtok', 'en')→'Ġtoken', ('n', 'd')→'nd', ('Ġ', 'is')→'Ġis', ('Ġt', 'h')→'Ġth', ('Ġth', 'e')→'Ġthe', ('i', 'n')→'in', ('Ġa', 'b')→'Ġab', ('Ġtoken', 'i')→'Ġtokeni'

Byte-Pair Encoding tokenization

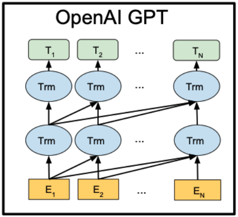
- Final vocabulary

```
['&lt;|endoftext|&gt;', ',', '.', 'C', 'F', 'H', 'T', 'a', 'b',
'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm', 'n', 'o',
'p', 'r', 's', 't', 'u', 'v', 'w', 'y', 'z', 'Ġ', 'Ġt', 'is',
'er', 'Ġa', 'Ġto', 'en', 'Th', 'This', 'ou', 'se', 'Ġtok',
'Ġtoken', 'nd', 'Ġis', 'Ġth', 'Ġthe', 'in', 'Ġab', 'Ġtokeni']
```

- To tokenize a new text, we pre-tokenize it, split it, then apply all the merge rules learned

```
"This is not a token." → ['This', 'Ġis', 'Ġ', 'n', 'o', 't',
'Ġa', 'Ġtoken', '.']
```

GPT-1



- Supervised finetuning: Given m input tokens $\{x^1, \ldots, x^m\}$ and its label y. Input the tokens into the pretrained model and obtain the encoded feature vector h_l^m and use an FC layer to obtain the final prediction

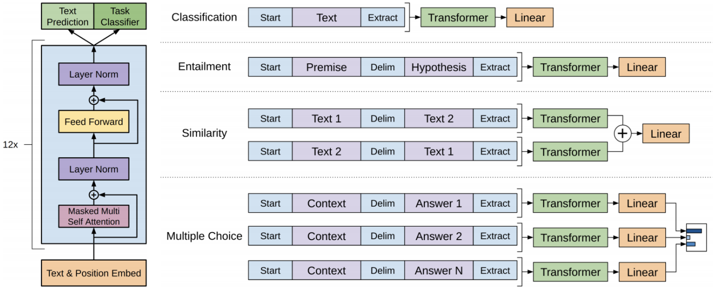
$$P\left(y \mid x^1, \ldots, x^m\right) = \operatorname{softmax}\left(h_l^m W_y\right),$$

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

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda L_1(\mathcal{C}),$$

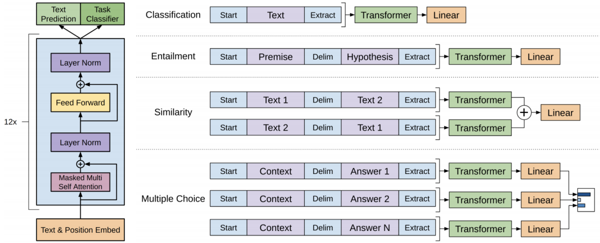
where λ is generally set as 0.5 to maintain the capability of next-word generation

GPT-1



- During finetuning, only the output W_y layer and delimiter token is finetuned
- GPT-1 uses byte pair encoding (BPE) + special tokens to form the vocabulary:
  - $< s >$: Start
  - $< e >$: Extract (End)
  - $< p >$: Premise
  - $< h >$: Hypothesis
  - Delimiter

GPT-1



- **Classification:** Start and end tokens to the two ends of the original sequence as input into the transformer. The final FC layer for prediction
- **Entailment:** Add premise and hypothesis (separated by delimiter). Add start and end tokens to the two ends. Use Transformer and the final FC layer for prediction
- **Text Similarity:** Input the two sentences A and B and swap their order and reinput into the two sentences. Concatenate the two embeddings after the transformer and input into the final FC layer for estimation
- **Multiple Choices:** Convert n-choice question into n binary questions and input into the transformer. Choose the answer with the highest probability.

- 12 Transformer layers with masked attention and 12 multi-attention heads
- 768-d word embeddings and positional embeddings
- FFN 3072-d embeddings
- GELU as the activation function

- **BooksCorpus** training dataset with 70 unpublished books. The books are never seen before and can better test the model's generalization capability on downstream tasks
- Achieve state-of-the-art performances in 9 out of 12 supervised tasks and also show great zero-shot capability on unseen tasks

Table : Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

GPT-2

- GPT-2 aims at utilize the same learning model to tackle multiple tasks
- With task conditioning, GPT-2 can handle zero-shot learning tasks. New downstream tasks don't need to provide any training labels and just use instruction to understand the task
- Problem setups of GPT-2 vs. GPT-1

$$p(\text{output}|\text{input}) \text{ vs. } p(\text{output}|\text{input};\text{task})$$

- Every task is a sub-set of natural language modeling. During training, the answers of the tasks are given. During inference, no answer is provided
- **Reading Comprehension**

$$d_1, d_2, \ldots, d_N\texttt{, "Q:", } q_1, q_2, \ldots, q_M\texttt{, "A:"}$$

- **Summarization**

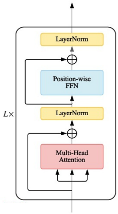
$$d_1, d_2, \ldots, d_N\texttt{, "TL;DR:"}$$

- **Translation**

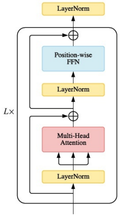
```
Sentence 1 in Language A, "=", Sentence 1 in Language B
Sentence 2 in Language A, "=", Sentence 2 in Language B
Sentence 3 in Language A, "="
```

GPT-2

- GPT-2 utilizes 40GB **WebText** dataset, which consists of articles from Reddit 8M posts with high upvotes and excludes Wikipedia articles
- 48 transformer layers
- The vocabulary is expanded to 50,257. The context size is expanded from 512 to 1024 tokens and a larger batch size of 512
- Move the normalization before each sub-block and add a layer normalization after the final transformer block
- A modified initialization which accounts for the accumulation on the residual path with model depth is used. Weights of residual layers are scaled at initialization by a factor of $1/\sqrt{N}$ where N is the number of residual layers

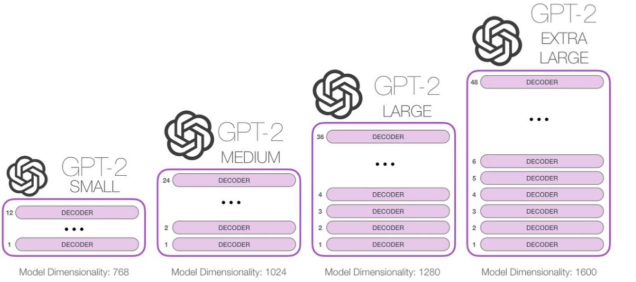


(a) post-LN

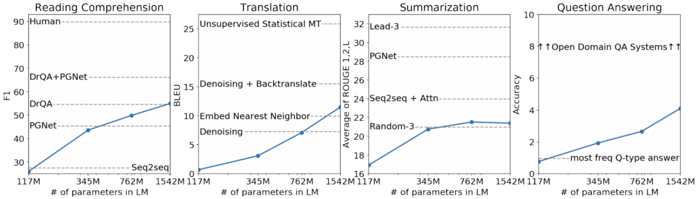


(b) pre-LN

GPT-2

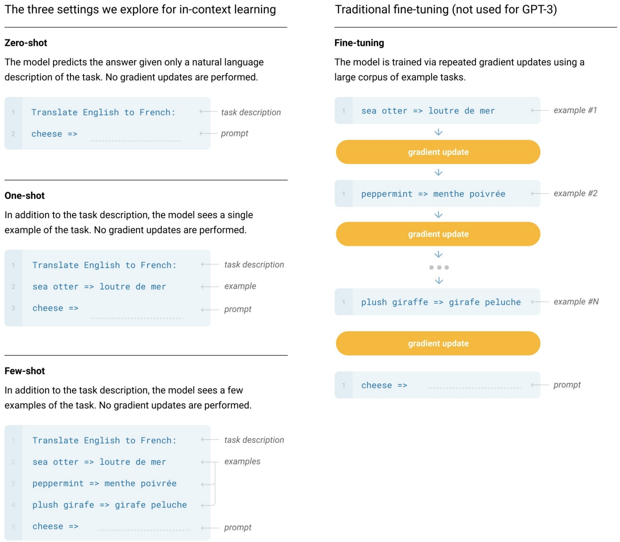
- 

- Results



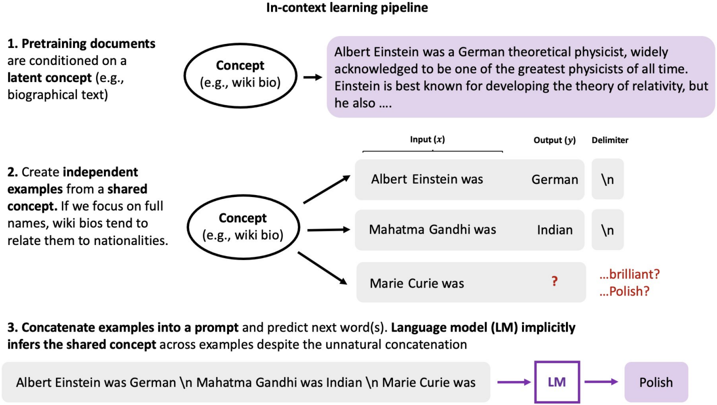
GPT-3

- GPT-3 is a huge model and has 175B parameters, 45TB training data, and a training cost of 12M USD
- It explores three training strategies, zero-shot, one-shot, and few-shot learning



GPT-3's in-context learning (ICL) capability

- GPT-3 is a huge model and has 175B parameters, 45TB training data, and a training cost of 12M USD
- After pretraining, GPT-3 can magically have the ability to be adapted by a few instruction-answer examples



GPT-3

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

Figure: Datasets used to train GPT-3.

GPT-3

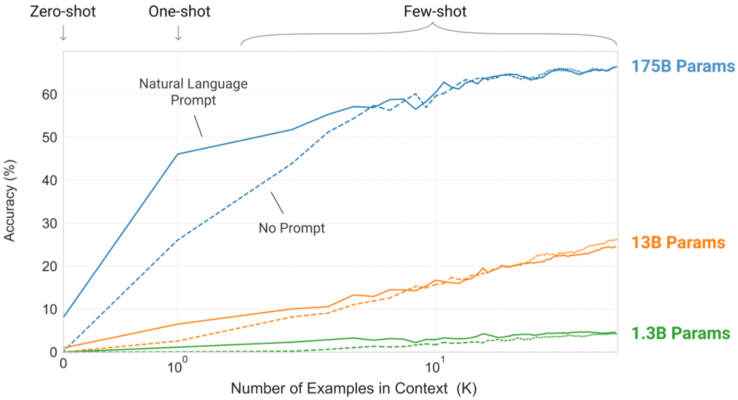


Figure: In-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description. The steeper "in-context learning curves" for large models demonstrate improved ability to learn a task from contextual information.

GPT-3 Results

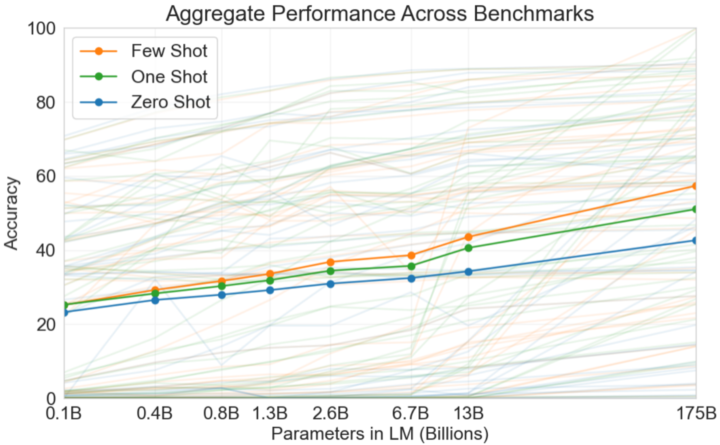
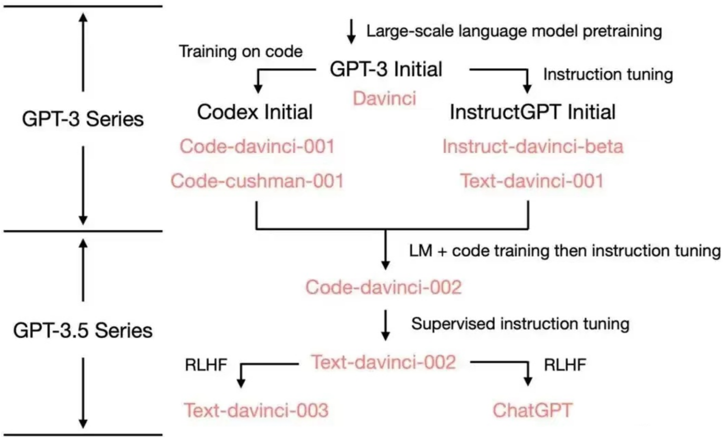


Figure: Aggregate performance for all 42 accuracy-denominated benchmarks While zero-shot performance improves steadily with model size, few-shot performance increases more rapidly, demonstrating that larger models are more proficient at in-context learning.

InstructGPT

- The GPT-3 model can be coaxed to perform natural language tasks using carefully engineered text prompts
- But these models can also generate outputs that are untruthful, toxic, or reflect harmful sentiments
- This is because GPT-3 is trained to predict the next word on a large dataset of Internet text, rather than to safely perform the language task that the user wants
- To make the GPT models safer, more helpful, and more aligned, OpenAI introduces **InstructGPT** and uses an existing technique called **reinforcement learning from human feedback (RLHF)** to finetune a pretrained GPT model

Timeline of GPT-3 and GPT-3.5



GPT-3 series

	Codename	Training Method	Model Size	Corpus Size	Release Time	Remarks
GPT-3	davinci	Unsupervised Pretraining	175B	570GB	May 2020	Weak language understanding ability
CodeX	code-davinci-001	Unsupervised pretraining on code	12B	179GB code from Github	Jul. 2021	Code completion for copilot
InstructGPT	text-davinci-001	Finetuning on supervised tasks	175B	13k SFT, 33k RM, 31k RL	Mar. 2022	Strong zero-shot capability
GPT-3.5	code-davinci-002	Unsupervised pretraining on code	>175B?	179GB code from Github	Jul. 2022	Emerging capability of chain-of-thoughts
	text-davinci-002	Based on code-danvinci-002, finetuning on supervised tasks	>175B?	>7.7k human annotations	Jul. 2022	Much stronger zero-shot capability
	text-davinci-003	Based on text-danvinci-002, add RLHF and focus on in-context learning	>175B?	>7.7k human annotations	Dec. 2022	Safer generation results
ChatGPT		Based on text-danvinci-002, add RLHF and focus on multi-round conversation	>175B?	>7.7k human annotations	Dec. 2022	Safer generation results

InstructGPT Method

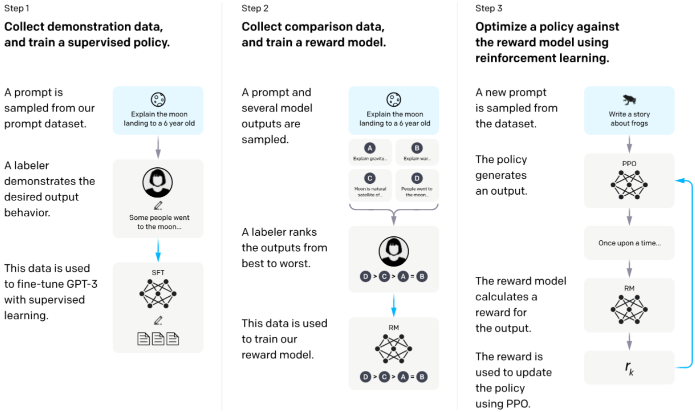


Figure: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of the models. In Step 2, boxes A-D are samples from the models that get ranked by labelers.

InstructGPT

- **SFT dataset:** The prompt dataset consists primarily of text prompts submitted to the OpenAI API on the Playground interface
- In addition, OpenAI hired 40 human labelers to write prompts themselves and requested labelers to write three kinds of prompts:
  - **Plain:** We simply ask the labelers to come up with an arbitrary task, while ensuring the tasks had sufficient diversity
  - **Few-show:** We ask the labelers to come up with an instruction, and multiple query/response pairs for that instruction
  - **User-based:** We had a number of use-cases stated in waitlist applications to the OpenAI API. We asked labelers to come up with prompts corresponding to these use cases
- These prompts are very diverse and include generation, question answering, dialog, summarization, extractions, and other natural language tasks

Table 1: Distribution of use case categories from our API prompt dataset.

Use-case	(%)
Generation	45.6%
Open QA	12.4%
Brainstorming	11.2%
Chat	8.4%
Rewrite	6.6%
Summarization	4.2%
Classification	3.5%
Other	3.5%
Closed QA	2.6%
Extract	1.9%

Table 2: Illustrative prompts from our API prompt dataset. These are fictional examples inspired by real usage—see more examples in Appendix A.2.1.

Use-case	Prompt
Brainstorming	List five ideas for how to regain enthusiasm for my career
Generation	Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home.
Rewrite	This is the summary of a Broadway play: """ {summary} """ This is the outline of the commercial for that play: """

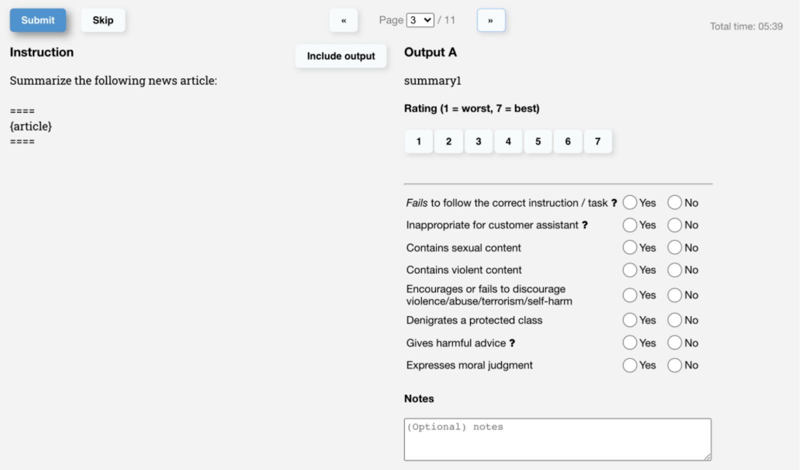
InstructGPT

Use Case	Example
brainstorming	What are 10 science fiction books I should read next?
classification	Take the following text and rate, on a scale from 1-10, how sarcastic the person is being (1 = not at all, 10 = extremely sarcastic). Also give an explanation {text} Rating:
classification	This is a list of tweets and the sentiment categories they fall into. Tweet: {tweet_content1} Sentiment: {sentiment1} Tweet: {tweet_content2} Sentiment: {sentiment2}
classification	{java code} What language is the code above written in?
classification	You are a very serious professor, and you check papers to see if they contain missing citations. Given the text, say whether it is missing an important citation (YES/NO) and which sentence(s) require citing. {text of paper}

Use Case	Example
generation	Here's a message to me: — {email} — Here are some bullet points for a reply: — {message} — Write a detailed reply
generation	This is an article about how to write a cover letter when applying for jobs: — It's important to spend some time
generation	write rap lyrics on the topics mentioned in this news article: — {article} —
rewrite	This is the summary of a Broadway play: """ {summary} """ This is the outline of the commercial for that play: """

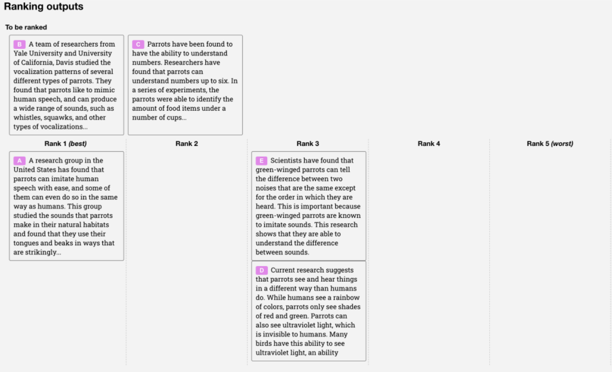
InstructGPT

Human annotation interface



InstructGPT

- Reward modeling (RM) dataset: a dataset of prompts submitted to early InstructGPT models on OpenAI API and human annotated instructions and use partially trained InstructGPT for answering. Ask human labelers to rank the answers and train a reward model
- PPO dataset: Use the reward model above to finetune the InstructGPT model



InstructGTP

SFT Data			RM Data			PPO Data		
split	source	size	split	source	size	split	source	size
train	labeler	11,295	train	labeler	6,623	train	customer	31,144
train	customer	1,430	train	customer	26,584	valid	customer	16,185
valid	labeler	1,550	valid	labeler	3,488			
valid	customer	103	valid	customer	14,399			

Figure: Data sizes of ChatGPT.

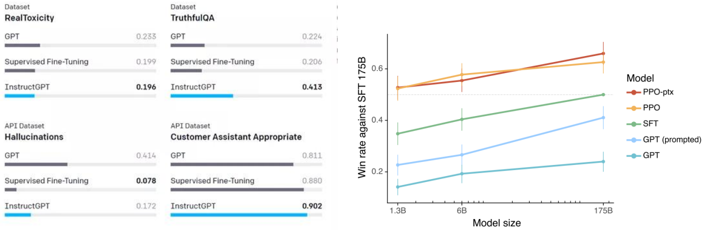


Figure: (Left) InstructGPT generates safer, cleaner, more truthful, less toxic outputs. (Right) Human evaluations on OpenAI API prompt distribution. InstructGPT (PPO-ptx) and its variant without pretraining (PPO) outputs GPT-3 baselines (GPT, GPT prompted) significantly.

Discussions

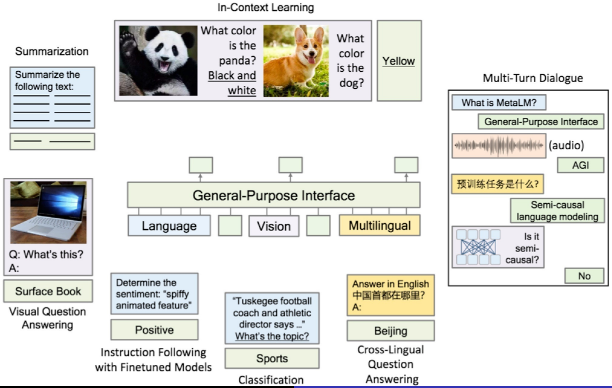
- When using the pretraining-finetuing paradigm, BERT or T5 methods still show better performance than the generative models
- However, AR language model + Prompting (e.g., GPT models) is currently the dominating methods
- Key reasons:
  - Generative model can unify all types of outputs via word generation since Google's T5
  - If using zero shot/few show prompting and abandon actual finetuning, AR model is a must
- Why zero shot/few shot prompting is preferred?
- LLMs should already encode all knowledge seen during training and are generally quite huge. Such LLMs are unlikely to be finetuned repeatedly at different scenarios (the cost is simply too high)
- Given the huge pretrained LLM, zero shot/few shot prompting can help the LLM to unleash its full capability. However, few shot prompting is actually not a natural way for humans to define a task. When better interface is introduced, it might be abandoned

InstructGPT: Create New and Natural Interface for LLM

- Compared with GPT-3.5, InstrctGPT uses instructions to replace few-shot prompts
- Does InstructGPT inject new knowledge into LLM? Yes and no.
- **No:** the new data amount used in InstructGPT is so few compared with the data size used for pretraining LLM
- **Yes:** the newly injected instructions are mostly used to "teach" the GPT model to be polite, non-toxic, and to follow human preferences. Such capability might be already embedded in the pretrained LLM but need to be specifically motivated
- Make GPT adapt human preference instead of requiring humans to adapt GPT. Instructions are more natural than few-shot prompting

Some future research directions

- How to better integrate multi-modal data into LLM and absorb multi-modal knowledge, to handle multi-modal inputs and handle cross-modal tasks (pretraining or end tasks) in other fields
- How to better design interfaces (e.g., in-context learning & instructions) to better motivate the embedded capability of LLM to handle tasks (e.g., chain of thoughts)



Knowledge in LLM

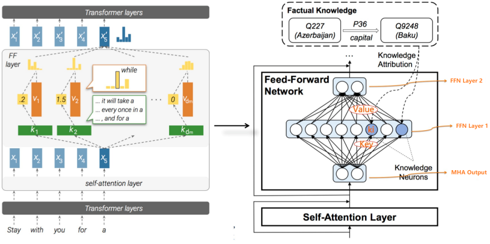
- Two types of knowledge in LLM: language knowledge and word knowledge
- **Language knowledge:** grammar, parts of speech of words, syntax, semantics of natural languages
- **World knowledge.** 1) Factual knowledge: "Biden is the US President", ""; 2) Common sense knowledge: "humans have two eyes", "the sun rises from the east"
- LLM can learn knowledge from training text data
- World knowledge is mostly memorized in the intermediate and high layers, especially intermediate layers, of the Transformers
- For a BERT-series model, a training corpus of 10M-100M words is enough for learning language knowledge

Knowledge in LLM

- Where does Transformer store the knowledge? Apparently, knowledge must be stored in the network weights
- For a Transformer, 1/3 parameters are in the multi-head attention (MHA) layers and 2/3 parameters are in the FFN layers
- MHA layers are mostly used to model inter-word or inter-knowledge relations for information aggregation. It is mostly establishing the relations between words and knowledge and they are unlikely to store knowledge
- Therefore, LLM models mostly store the knowledge in FFN layers
- In the paper "Transformer Feed-Forward Layers Are Key-Value Memories", the authors regard FFN as key-value memories to store knowledge

Knowledge in LLM

- The output feature from MHA sub-layer is input into the FFN sub-layer
- If FFN stores the knowledge, the MHA output feature is fed into the FFN and can serve as the key to retrieve the knowledge in FFN
- In the first FC layer, the FFN determines which neurons/knolwedge is acitvated. In the second FC layer, the FFN aggregates the merged knowledge
- This paper also made a similar conclusion: lower FFN layers store wording and sentence patterns, and intermediate and higher FFN layers encode world knowledge



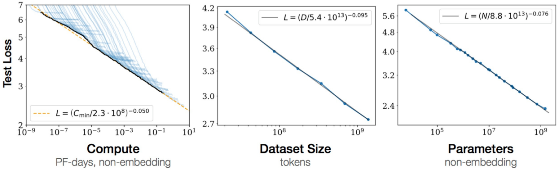
From: 1.Transformer Feed-Forward Layers Are Key-Value Memories
2.Knowledge Neurons in Pre-trained Transformers

Revising the Knowledge in LLM

- **Method 1:** In the paper "Towards Tracing Factual Knowledge in Language Models Back to the Training Data", authors try to track down which training data cause the LLM learn the knowledge. Revise the training data and re-training again
- However, finetuning a LLM would be very time-consuming and is impractical as the knowledge in LLM needs to be constantly updated
- **Method 2:** In the paper "Modifying Memories in Transformer Models", based on knowledge to be revised, build a finetuing dataset and finetune the LLM.
- Again, it requires constantly finetuing
- **Method 3:** In the paper "Locating and Editing Factual Associations in GPT" and "Mass-Editing Memory in a Transformer". They localize parameters related to knowledge to be revised and directly modify the knowledge-related parameters

Scaling up LLMs

- In OpenAI's 2020 paper "Scaling Laws for Neural Language Models", they have investigated how to scale up the LLM
- Separately scaling up training data size, model parameters, training time (epochs) all lead to the decrease of test loss
- How to balance the three aspects. OpenAI chose to increase the data size and model parameters, but uses early stop to decrease the training time
- If the total computational resources increases by 10×, then the model parameters and data size should be scaled up by 5.5× and 1.8×, respectively



- DeepMind's 2022 paper "Training Compute-Optimal Large Language Models" has a similar conclusion: If the total computational resources increases by 10×, then the model parameters and data size should be scaled up by 3.3× and 3.3×, respectively

Scaling up LLMs

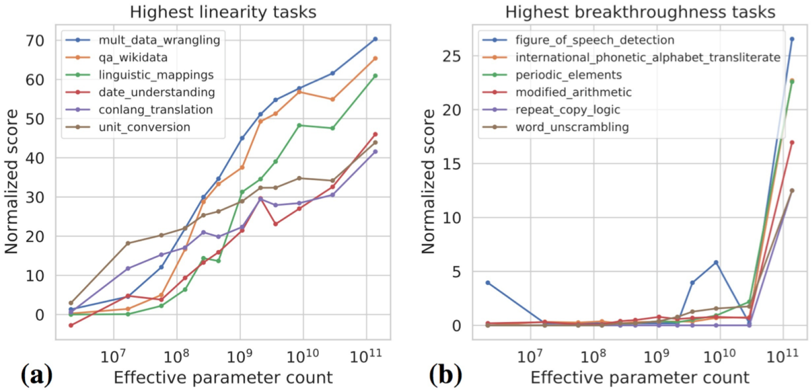


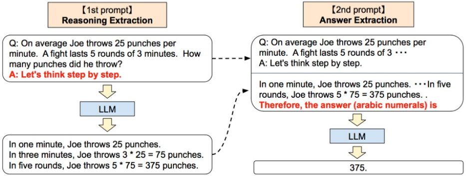
Figure: (Left) Effect of scaling up LLMs. (Right) **Emergent ability** when the LLM parameter size reaches a certain threshold.

Improving Reasoning Capability of LLMs

- When scaling up LLMs to a certain size, they show reasoning ability to some extent
- Two categories of methods to improve the reasoning ability: 1) **Prompt-based methods:** to study how to develop better prompts to activate the underlying reasoning ability of the LLM; 2) **Code-based training data (e.g., OpenAI's ChatGPT):** including code into the text training data to enhance the reasoning ability.
- We will discuss more on the first category of methods
- There are three lines of research on prompt engineering in this direction

Improving Reasoning Capability of LLMs

- **Line 1: Zero-shot CoT.** Directly appending prompts about reasoning to the question and into the answer as introduced in the 2022 paper "Large language models are zero-shot reasoners"
- Following the question, adding the prompt "Let's think step by step"
- Before outputting the final answer, adding the prompt "Therefore, the answer (arabic numbrals) is"

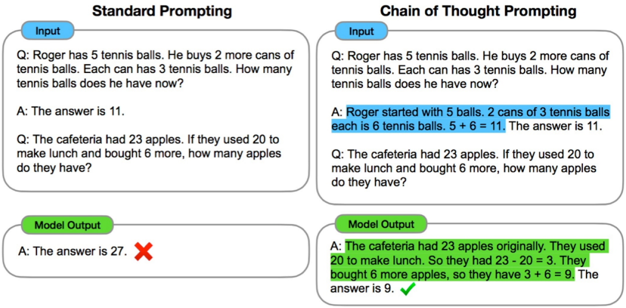


	Arithmetic					
	SingleEq	AddSub	MultiArith	GSM8K	AQUA	SVAMP
zero-shot	74.6/**78.7**	**72.2/77.0**	17.7/22.7	10.4/12.5	22.4/22.4	58.8/58.7
zero-shot-cot	**78.0/78.7**	69.6/74.7	**78.7/79.3**	**40.7/40.5**	**33.5/31.9**	**62.1/63.7**
	Common Sense		Other Reasoning Tasks		Symbolic Reasoning	
	Common SenseQA	Strategy QA	Date Understand	Shuffled Objects	Last Letter (4 words)	Coin Flip (4 times)
zero-shot	**68.8/72.6**	12.7/**54.3**	49.3/33.6	31.3/29.7	0.2/-	12.8/53.8
zero-shot-cot	64.6/64.0	**54.8**/52.3	**67.5/61.8**	**52.4/52.9**	**57.6/-**	**91.4/87.8**

Figure: Zero-shot chain of thoughts.

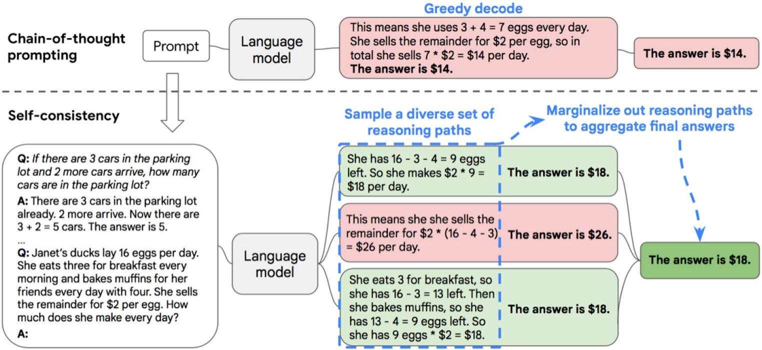
Improving Reasoning Capability of LLMs

- Why it works? It might be because there are a large number of such sentences in training corpus. Such a prompt can activate the underlying reasoning capability of the pretrained LLM
- **Line 2: Few-shot CoT.** In the 2022 paper "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models", it provides few-shot prompts to LLM to "teach" LLM how to properly reason



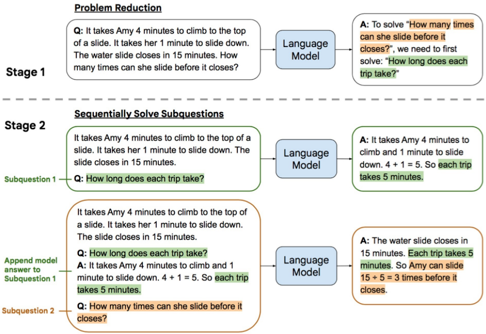
Improving Reasoning Capability of LLMs

- In the 2022 paper "Self-Consistency Improves Chain of Thought Reasoning in Language Models"
- Use CoT to provide several reasoning paths and choose the one with the largest marginal probability



Improving Reasoning Capability of LLMs

- **Line 3: Divide and Conquer.** In the 2022 paper "On the Advance of Making Language Models Better Reasoners"
- Use the original question to generate a prompt: "To solve" the original question ", we need to first solve:", generated subquestion
- We then obtain the subanswer of the subquestion, append the subquestion+subanswer as the new prompt, and ask the final question



Improving Reasoning Capability of LLMs

- Integrating code in the pretraining phase also improves the reasoning capability of LLMs
- code-davinci-002 improves over GPT-3 davinci over 20-50% accuracy

Method	GSM8K	AsDiv	MultiArith	SVAMP	SingleEq	CommonsenseQA	StrategyQA	CLUTRR
Previous SOTA (Fine-tuning)	57[a]	75.3[b]	60.5[c]	57.4[d]	32.5[e]	91.2[f]	73.9[g]	67.0[h]
9–12 year olds (Cobbe et al., 2021)	60	-	-	-	-	-	-	-
LaMDA 137B:								
Greedy Decode	17.1	49.0	51.8	38.9	56.6	57.9	65.4	-
Self-Consistency	27.7	58.2	75.7	53.3	-	63.1	67.8	-
PaLM 540B:								
Greedy Decode	56.5	74.0	94.7	79.0	79.5	79.0	75.3	-
Self-Consistency	74.4	81.9	99.3	86.6	-	**80.7**	**81.6**	-
GPT-3 davinci (175B):								
Greedy Decode	8.7	31.4	31.4	21.2	38.2	48.2	59.3	33.6
Self-Consistency	18.9	52.8	68.6	44.6	59.6	57.4	65.9	42.5
DIVERSE	30.9 (+12.0)	57.6 (+4.8)	87.6 (+19.0)	46.9 (+2.3)	65.1 (+5.5)	75.0 (+17.6)	67.9 (+2.0)	92.5 (+50.0)
text-davinci-002:								
Greedy Decode	37.1	60.8	70.7	60.0	73.3	65.5	60.3	18.4
Self-Consistency	58.2	76.9	88.4	78.2	87.2	72.9	70.7	15.8
DIVERSE	70.2 (+12.0)	83.5 (+6.6)	96.4 (+8.0)	82.7 (+4.5)	86.5 (-0.7)	79.2 (+6.3)	73.1 (+2.4)	68.5 (+52.7)
code-davinci-002:								
Greedy Decode	55.3	75.5	88.8	70.5	87.5	73.4	73.8	32.9
Self-Consistency	76.7	86.2	98.6	85.8	93.7	77.3	78.3	35.6
DIVERSE	**82.3** (+5.6)	**88.7** (+1.5)	**99.8** (+1.2)	**87.0** (+1.2)	**94.9** (+1.2)	79.9 (+2.6)	77.7 (-0.6)	**95.9** (+60.1)

Table 2: The comparison of DIVERSE, *Greedy Decode* and *Self-Consistency*. The previous SOTA results (fine-tuned on non-gigantic pretrained transformers) are: a: Cobbe et al. (2021), b: Miao et al. (2020), c: Roy and Roth (2015), d: Pi et al. (2022), e: Hu et al. (2019a), f: Xu et al. (2021), g: Chowdhery et al. (2022), h: Sinha et al. (2019). The parameter number of either text-davinci-002 or code-davinci-002 is hidden to us.

References

- https://huggingface.co/course/chapter6/5?fw=pt
- https://huggingface.co/course/chapter6/6?fw=pt
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, NAACL-HLT 2019.
- Alec Radford et al., Language Models are Unsupervised Multitask Learners, NeurIPS 2018.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu, Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer , JMLR 2020.
- Tom B. Brown et al., Language Models are Few-Shot Learners, NeurIPS 2020.
- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." ICLR 2015.
- Long Ouyang et al., Training language models to follow instructions with human feedback, arXiv:2203.02155, 2022.
- OpenAI, GPT-4 Technical Report.
- BERTnesia: Investigating the capture and forgetting of knowledge in BERT
- When Do You Need Billions of Words of Pre-training Data?
- Transformer Feed-Forward Layers Are Key-Value Memories

References

- Ekin Akyürek, Tolga Bolukbasi, Frederick Liu, Binbin Xiong, Ian Tenney, Jacob Andreas, Kelvin Guu, Towards Tracing Factual Knowledge in Language Models Back to the Training Data, EMNLP 2018
- Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, Sanjiv Kumar, Modifying Memories in Transformer Models, arXiv:2012.00363.
- Kevin Meng, David Bau, Alex Andonian, Yonatan Belinkov, Locating and Editing Factual Associations in GPT, NeurIPS 2022.
- Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, David Bau, Mass Editing Memory in a Transformer, arXiv:2210.07229.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, Dario Amodei, Scaling Laws for Neural Language Models, arXiv:2001.08361.
- Aarohi Srivastava et al., Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models, arXiv:2206.04615.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, Yusuke Iwasawa, Large Language Models are Zero-Shot Reasoners, arXiv:2205.11916.

References

- Wei et al., Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, arXiv:2201.11903.
- Wang et al., Self-Consistency Improves Chain of Thought Reasoning in Language Models, ICLR 2023.
- Li et al., On the Advance of Making Language Models Better Reasoners, arXiv:2206.02336.
- Zhou et al., Least-to-Most Prompting Enables Complex Reasoning in Large Language Models, ICLR 2023.
- Suzgun et al., Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them, arXiv:2210.09261.
- Zhang et al., Automatic Chain of Thought Prompting in Large Language Models, ICLR 2023.