Introduction to Large Language Models for Natural Language Processing

Lê Hồng Phương
Data Science Laboratory
VIASM & Vietnam National University, Hanoi
<*phuonglh@vnu.edu.vn*>

June 30, 2023

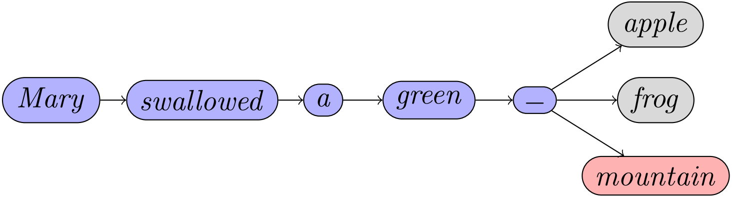
Content

Language Models

- A language model (LM) is a *probability distribution over sequences of words*. Given any sequence of words of length n, a LM assigns a probability $P(w_1, w_2, \ldots, w_n)$ to the sequence.
- An n-gram LM models sequences of words as a Markov process where we want to predict the next word given its history:

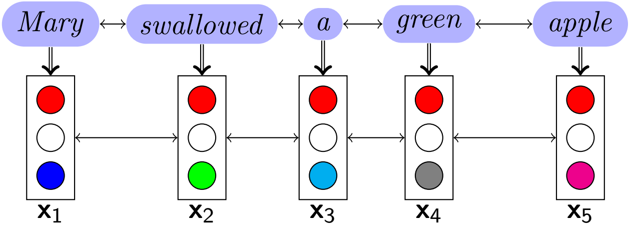
$$P(w_n | w_1, \ldots, w_{n-1})$$

- Text generation from the probabilistic LM:



Neural Language Models

- Neural language models (or *continuous space language models*) use continuous representations or embeddings of words to make their predictions. These models make use of neural networks.
- Each word or token has an associated embedding vector $\mathbf{x}_i \in \mathbb{R}^d$:

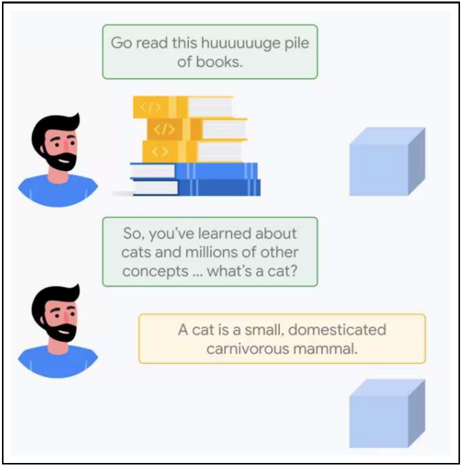


- Static word embeddings: Skip-gram and CBOW (2013), GloVe (2014)
- Dynamic word embeddings: ELMo (2017), BERT (2018)

Large Language Models

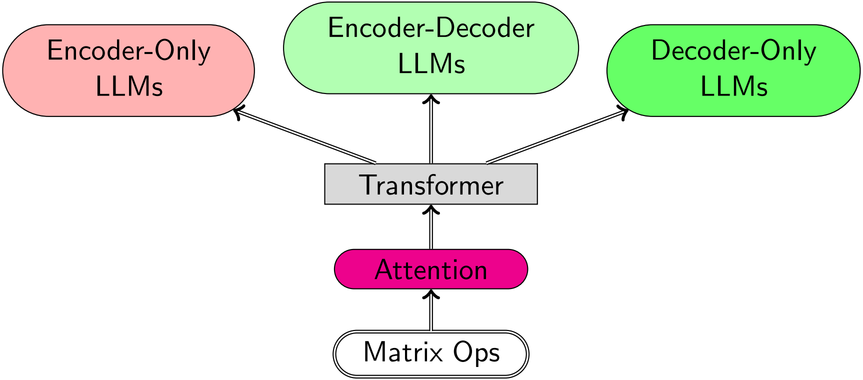
What are LLMs?

- ML algorithms that can **recognize**, **predict**, and **generate** human languages.
- Pretrained on petabyte scale text datasets resulting in large models with **10s to 100s of billions of parameters**.
- LLMs are normally pretrained followed by tuning on a specific task.



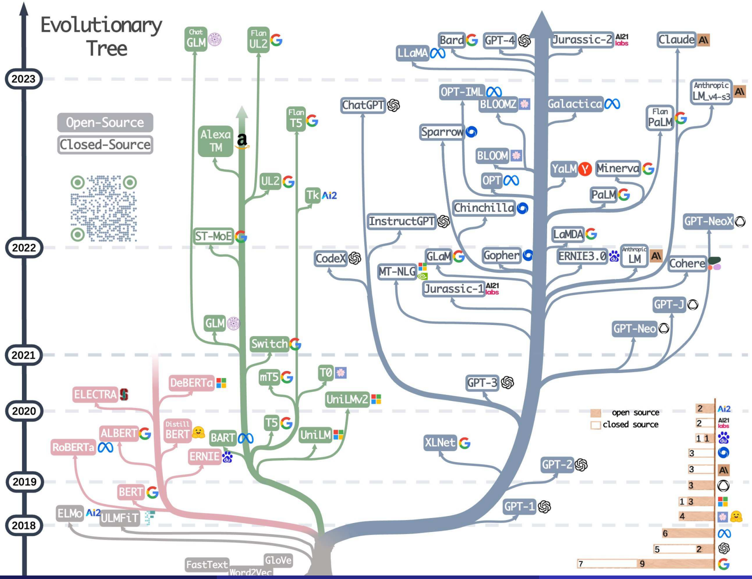
Large Language Models: Evolutionary Tree

- A nice recent survey presenting the evolution of LLMs[1]
- A simplified view:

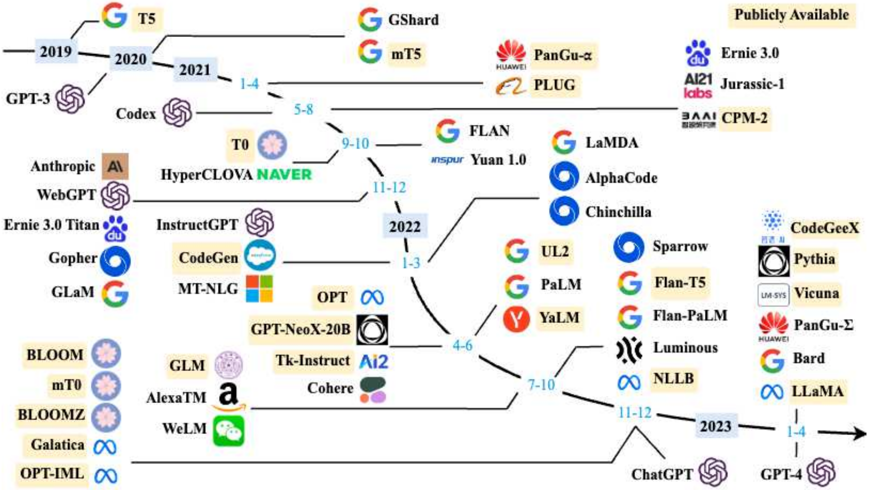


[1]Yang et al. (2023) Harnessing the power of LLMs in practice: a survey on ChatGPT and beyond.

Large Language Models: Evolutionary Tree



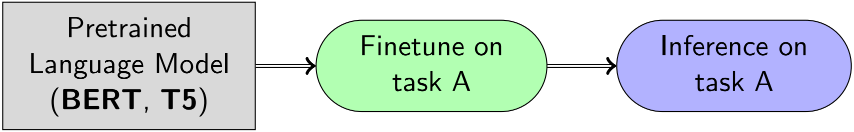
A Timeline of LLMs



2

[2] Zhao et al. (2023) A survey of large language models. `https://arxiv.org/abs/2303.18223`

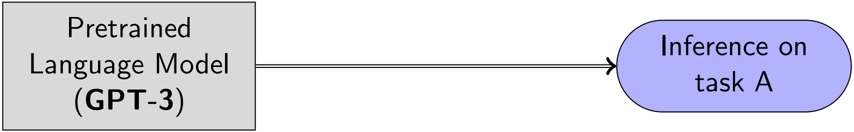
Pretraining – Finetuning Method: 2018–2021



Pretraining – Finetuning:

- Typically requires many task-specific examples
- One specialized model for each task

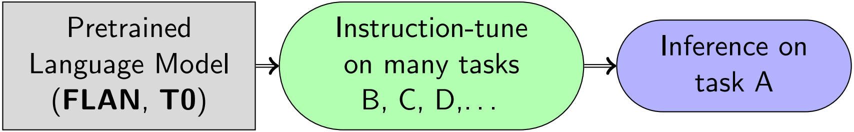
Prompting Method: 2021–2023



Prompting:

- Prompts are used to interact with LLMs to accomplish a task.
- A prompt is a user-provided input. Prompts can include instructions, questions, or any other type of input, depending on the intended use of the model.

Instruction-Tuning Method: 2022–2023



Instruction-tuning:

- Model learns to perform many tasks via natural language instructions
- Inference on unseen tasks

Content

The Attention Mechanism

Currently, the dominant models for nearly all NLP tasks are based on the Transformer architecture. Given any new task in NLP, we usually:

- Take a large Transformer-based pretrained model (BERT, GPT-x, T5-x, etc)
- Fine-tune/Prompt engineer the model on the available data for the downstream task

The core idea behind the Transformer model is the **attention mechanism**, an innovation which was introduced in 2014.[3]

[3] Bahdanau et al. (2014). *Neural machine translation by jointly learning to align and translate.*

The Attention Mechanism

- In machine translation, attention models often assigned high attention weights to cross-lingual synonyms when generating the corresponding words in the target language.
- "*My* *feet* *hurt*" $\rightarrow$ "*j'ai mal au* *pieds*":
    - The model might assign high attention weights to the representation of "*feet*" when generating "*pieds*".[4]

[4] Reference: "*Dive into DL*" – *https://d2l.ai/*

The Attention Mechanism

In 2017, the Transformer architecture was proposed which relies on cleverly arranged attention mechanisms to capture *all relationships* among input and output tokens.[5]

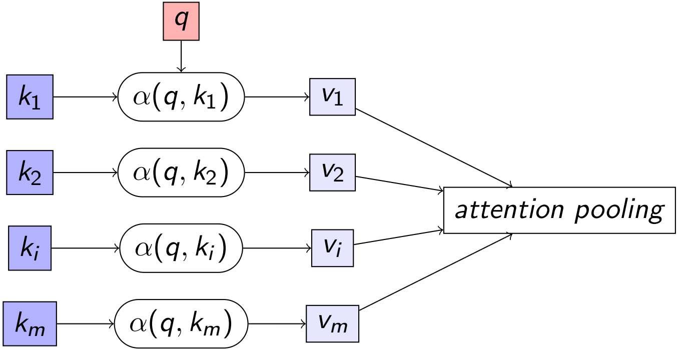
- Denote by $\mathcal{D} = \{(\mathbf{k}_1, \mathbf{v}_1), (\mathbf{k}_2, \mathbf{v}_2), \dots, (\mathbf{k}_m, \mathbf{v}_m)\}$ a database of m tuples (*key*, *value*).
- Denote by $\mathbf{q}$ a *query*.
- The *attention* over $\mathcal{D}$ is defined as

$$\text{Attention}(\mathbf{q}, \mathcal{D}) := \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\, \mathbf{v}_i,$$

where $\alpha(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}, \forall i = 1, \dots, m$ are scalar attention weights.

[5] Vaswani et al. (2017). *Attention is all you need*. NIPS.

Attention Pooling



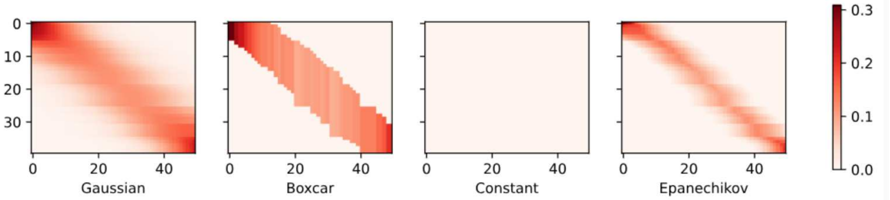
Some special cases:

1. $\alpha(\mathbf{q}, \mathbf{k}_i) \geq 0, \sum_i \alpha(\mathbf{q}, \mathbf{k}_i) = 1$: the weights form a convex combination.
2. $\alpha(\mathbf{q}, \mathbf{k}_j) = 1$, all other weights are 0: sparse attention.
3. $\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{1}{m}$: average pooling.

Attention Pooling By Similarity

Some common similarity kernels $\alpha(\mathbf{q}, \mathbf{k})$:

- Gaussian: $\alpha(\mathbf{q}, \mathbf{k}) = \exp\left(-\frac{1}{2\sigma^2} \| \mathbf{q} - \mathbf{k} \|^2\right)$
- Boxcar: $\alpha(\mathbf{q}, \mathbf{k}) = 1$ if $\| \mathbf{q} - \mathbf{k} \| \leq 1$
- Epanechikov: $\alpha(\mathbf{q}, \mathbf{k}) = \max\{0, 1 - \| \mathbf{q} - \mathbf{k} \|\}$



Scaled Dot Product Attention

From the Gaussian kernel:

$$\begin{aligned}\alpha(\mathbf{q}, \mathbf{k}_i) &= -\frac{1}{2}\| \mathbf{q} - \mathbf{k}_i \|^2 \\ &= \mathbf{q}^\top \mathbf{k}_i - \frac{1}{2}\| \mathbf{q}_i \|^2 - \frac{1}{2}\| \mathbf{k} \|^2\end{aligned}$$

The scaled dot product attention that is used in the Transformers:

$$a(\mathbf{q}, \mathbf{k}_i) = \frac{\mathbf{q}^\top \mathbf{k}_i}{\sqrt{d}}, \quad \mathbf{q}, \mathbf{k} \in \mathbb{R}^d$$

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^m \exp(a(\mathbf{q}, \mathbf{k}_j))}$$

Scaled Dot Product Attention

We often compute attention scores in mini-batches for efficiency. For n queries and m key-value pairs, $\mathbf{q}, \mathbf{k} \in \mathbb{R}^d$, and values $\mathbf{v} \in \mathbb{R}^v$, we use the batch matrix multiplication:

$$\operatorname{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n \times v},$$

for $\mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{m \times d}$, and $\mathbf{V} \in \mathbb{R}^{m \times v}$.

Scaled Dot Product Attention

When queries and keys are vectors of different dimensionalities, we can either

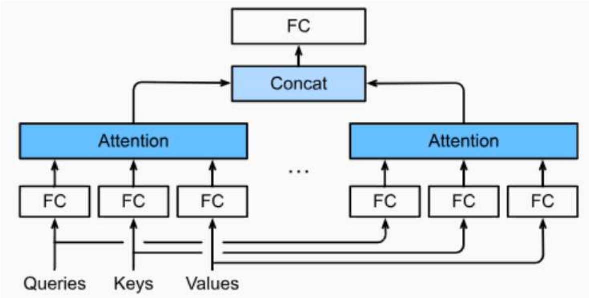
- use a matrix to address the mismatch via $\mathbf{q}^\top \mathbf{M}\,\mathbf{k}$, or
- use additive attention, for $\mathbf{q} \in \mathbb{R}^q, \mathbf{k} \in \mathbb{R}^k$:

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^\top \tanh\left(\mathbf{W}_q\,\mathbf{q} + \mathbf{W}_k\,\mathbf{k}\right) \in \mathbb{R},$$

where $\mathbf{w}_v \in \mathbb{R}^h, \mathbf{W}_q \in \mathbb{R}^{h\times q}, \mathbf{W}_k \in \mathbb{R}^{h\times k}$ are learnable parameters.

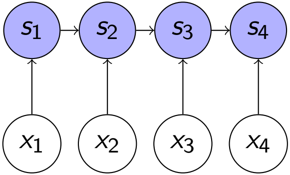
Multi-Head Attention

- We want our model to combine knowledge from different behaviors of the same attention mechanism, such as capturing dependencies of various ranges within a sequence.
  - It is beneficial to allow our attention mechanism to jointly use different representation subspaces of queries, keys, and values.
  - We concatenate multiple attention pooling outputs: *multi-head attention*.

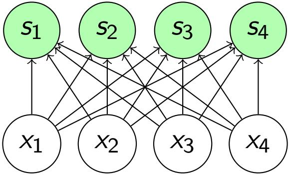


Self-Attention

- In sequence processing, each token has its own query, keys, and values. We can compute, for each token, a representation by building the appropriate weighted sum over the other tokens.
- Each token is attending to each other token $\Rightarrow$ *self-attention models*.



Recurrent Neural Network



Self-Attention

Self-Attention

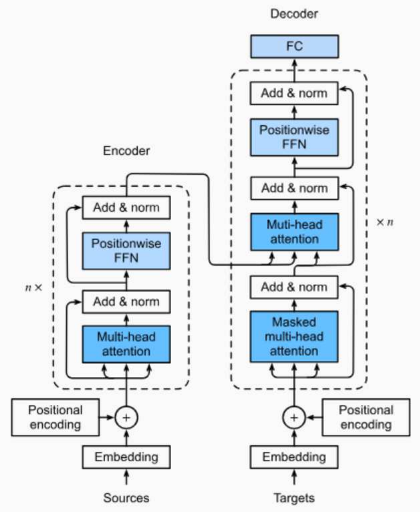
Computation complexity on a sequence of n tokens:

- RNN: multiplication of a weight matrix of size $d \times d$ and a d-dimensional hidden state $\Rightarrow O(nd^2)$; $O(n)$ operations cannot be parallelized.
- Self-attention: a $n \times d$ matrix is multiplied by a $d \times n$ matrix, then the resulting matrix is multiplied by a $n \times d$ matrix $\Rightarrow O(n^2 d)$; Computation can be parallelized with $O(1)$ sequential operation.

Content

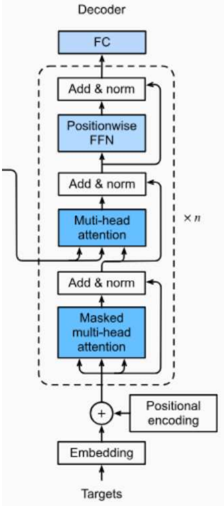
The Transformer Architecture

- The Transformer is composed of an **encoder** and a **decoder**.
- The encoder is a stack of multiple identical layers, where each layer has two sublayers
  - The first is a multi-head self-attention pooling.
  - The second is a positionwise feed-forward network.
  - There is a *residual connection* at both sublayers, inspired by the ResNet.
- The encoder outputs a d-dimensional vector representation for each position of the input sequence.

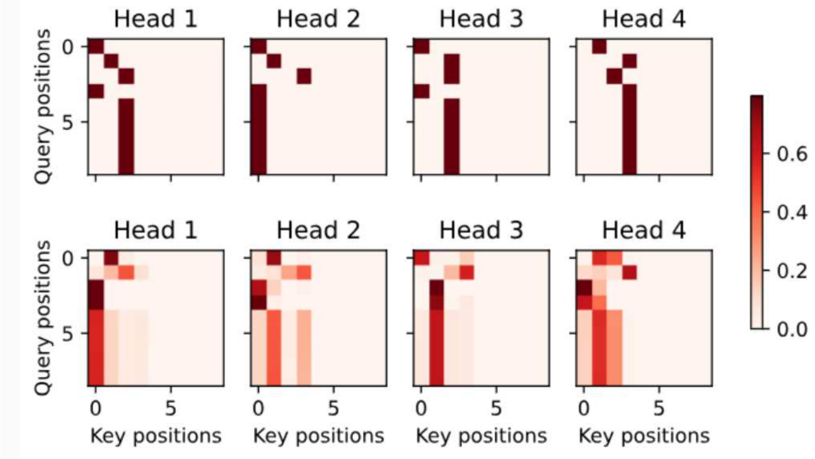


The Transformer Architecture

- The decoder is also a stack of multiple identical layers with residual connections and layer normalizations.
    - Between the two sublayers, the decoder has a third sublayer, called the *encoder-decoder attention*.
    - In the encoder-decoder attention, queries are from the outputs of the previous decoder layer, and the keys and values are from the encoder outputs.
    - In the decoder self-attention, queries, keys, and values are all from the outputs of the previous decoder layer.
- Each position in the decoder is allowed to only attend to all positions up to that position.
    - The *masked attention* preserves the auto-regressive property, ensuring that the prediction only depends on those output tokens that have been generated.



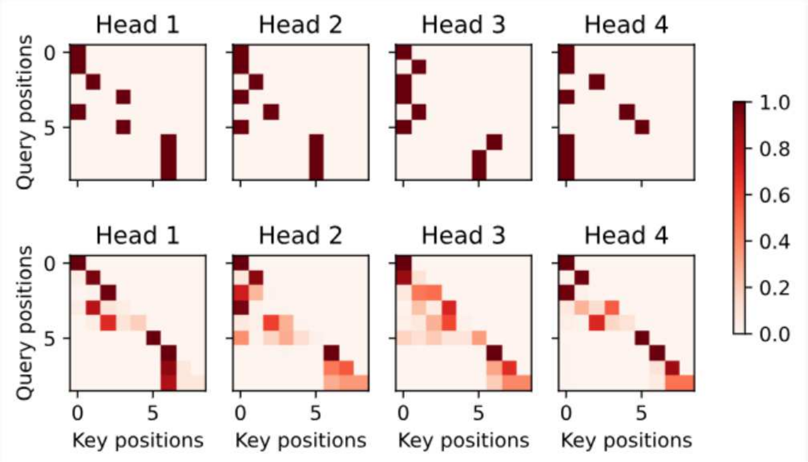
Encoder Self-Attention Weights



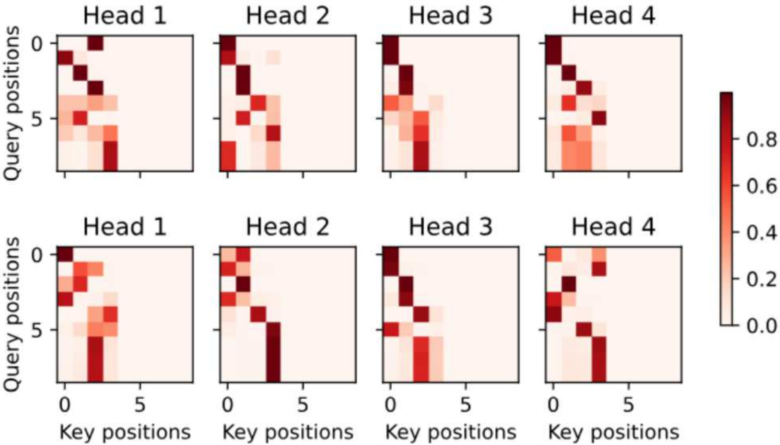
Two layers of multi-head attention weights are presented row by row.[6]

[6] https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html

Decoder Self-Attention Weights



Encoder-Decoder Self-Attention Weights



$$i\text{'}m\ home\ .\Rightarrow [je,\ suis,\ chez,\ moi,\ .]$$

Large-Scale Pretraining with Transformers: Three Modes

Transformers have been extensively pretrained with a wealth of text to learn good representations.

- Originally proposed for MT, the Transformer architecture consists of an encoder for representing input sequences and a decoder for generating target sequences.
- Transformers can be used in three different modes:
  1. **encoder-only**
  2. **encoder-decoder**
  3. **decoder-only**

Content

Encoder-Only Transformer

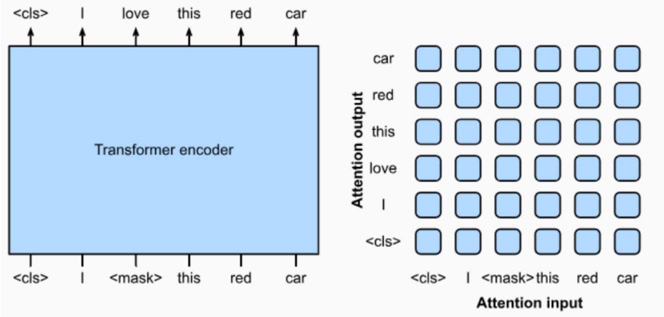
A Transformer encoder consists of self-attention layers, where all input tokens attend to each other.

- A sequence of input tokens is converted into the same number of representations by the encoder.
- These representations can then be further projected into output (e.g., classification).
- This design was inspired by an earlier encoder-only Transformer pretrained on text: BERT.[7]

[7] Devlin et al. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding.

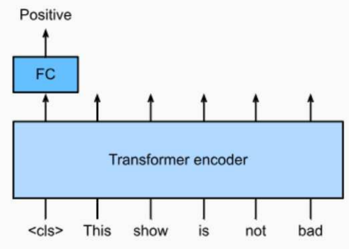
Encoder-Only Transformer: BERT Pretraining

- BERT is pretrained on text sequences using *masked language modeling*: input text with randomly masked tokens is fed into a Transformer encoder to predict the masked tokens.
  - There is no constraint in the attention pattern of Transformer encoders.
  - Prediction of "`love`" depends on input tokens before and after it.
- Large-scale text data can be used for pretraining BERT.



Encoder-Only Transformer: BERT Fine-Tuning

- The pretrained BERT can be fine-tuned to downstream encoding tasks involving single text or text pairs.
  - During fine-tuning, additional layers can be added to BERT with randomized parameters: these parameters and those pretrained BERT parameters will be updated to fit training data of downstream tasks.
  - The general language representations learned by the 340M-parameter BERT from 250B training tokens advanced the SOTA for many NLP tasks.



Derivatives of BERT

Other derivatives of BERT improved model architectures or pretraining objectives:

1. RoBERTa (2019)
    - change key hyperparameters
    - remove the next-sentence pretraining objective of BERT
    - train with much larger mini-batches and learning rates
2. DeBERTa (2021)
    - use disentangled attention mechanism: each word is represented using two vectors that encode its content and position, the attention weights among words are computed using disentangled matrices on their contents and relative positions
    - use enhanced mask decoder to replace the output softmax layer to predict the masked tokens
3. Others
    - ALBERT (2019): enforces parameter sharing
    - SpanBERT (2020): predicts spans of texts
    - ELECTRA (2020): replaced token detection

Multilingual BERT

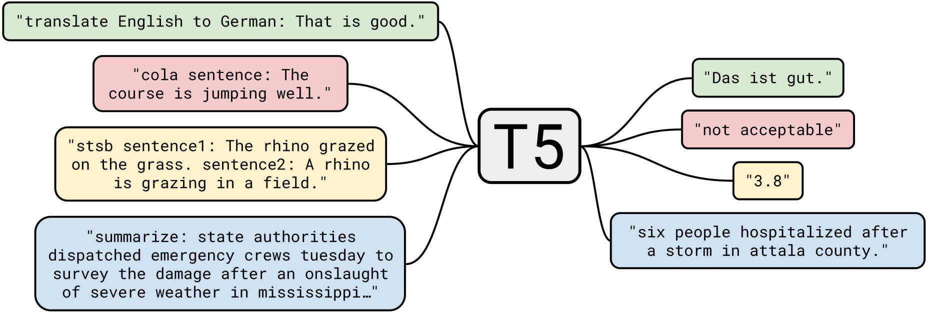
1. mBERT (2019)
   - a multilingual version of BERT trained on 104 languages from the Wikipedia corpus.
   - follows the BERT recipe with the same training architecture and objective
   - 110M to 340M parameters
2. XLM-R (2020)
   - a multilingual language model, trained on 2.5TB of filtered CommonCrawl data of 100 different languages
   - performs particularly well on low-resource languages
   - outperforms mBERT on a variety of cross-lingual benchmarks.
   - 3.5B to 10.7B parameters
3. BERT pretrained models for Vietnamese:
   - ViBERT (10/2020), FPT
   - PhoBERT (11/2020), VinAI

Content

Encoder-Decoder Transformer

- The decoder autoregressively predicts the target sequence of arbitrary length, token by token, conditional on both encoder output and decoder output:
  - the encoder-decoder cross-attention allows target tokens to attend to all input tokens
  - the masked multi-head attention of decoder allow any target token can attend to past and present tokens in the target sequence
- Two well-known encoder-decoder Transformers, both attempt to reconstruct original text in their pretraining objectives.
  1. BART (2019): emphasizes *noising input* (masking, deletion, permutation, rotation)
  2. T5/mT5 (2020): emphasizes *multitask unification*

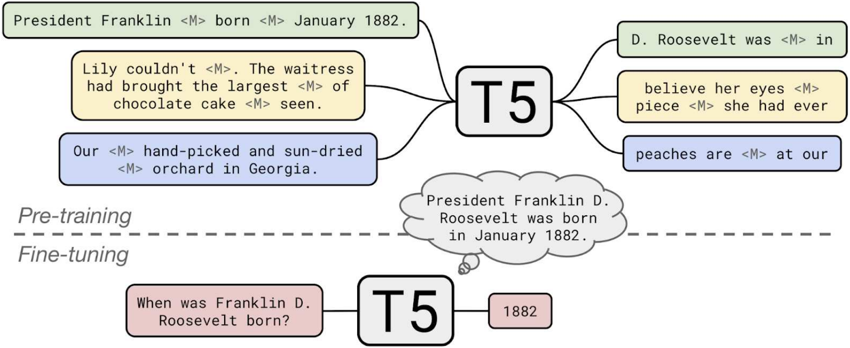
Encoder-Decoder Transformer: T5



- Every task is cast as feeding the model text as input and training it to generate some target text.[8]
- This allows for the use of the same model, loss function, hyperparameters across different tasks.

[8] Raffel et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. JMLR.

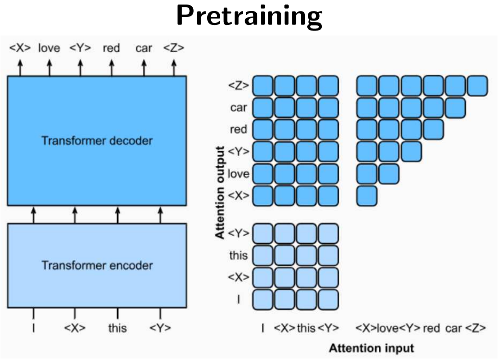
Encoder-Decoder Transformer: T5

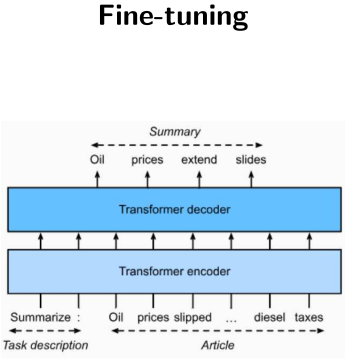


- T5 can be fine-tuned for novel tasks.[9]
- 11B-parameter T5 achieved SOTA on the GLUE, SuperGLUE, SQuAD, and CNN/Daily Mail benchmarks.

[9] https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html

Encoder-Decoder Transformer: T5





Switch Transformer (2022) is based on T5.[10]

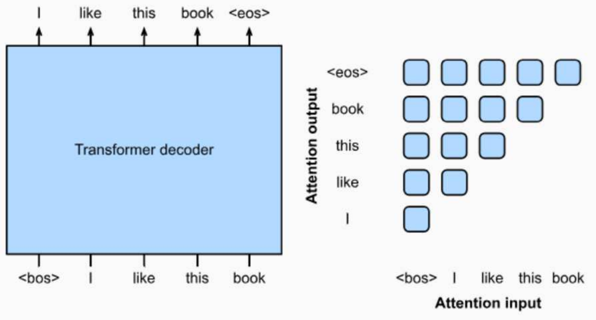
- improve with reduced communication and computational costs
- advance the of LLMs by pre-training up to trillion parameter models, achieved a 4x speedup over the T5-XXL model

[10] Fedus et al. (2022). Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. JMLR.

Content

Decoder-Only Transformer: GPT-2

- Remove the entire encoder and the encoder-decoder cross-attention sublayer from the original encoder-decoder architecture.
- GPT and GPT-2 chooses a Transformer decoder as its backbone.



Decoder-Only Transformer: GPT-2

- GPT (2018)[11]: 100M parameters, need to be fine-tuned for downstream tasks.
- GPT-2 (11/2019)[12]: 1.5B parameters, performed well on multiple other tasks *without updating the parameters or architecture.*[13]

> GPT-2 displays a broad set of capabilities, including the ability to generate conditional synthetic text samples of unprecedented quality, where we prime the model with an input and have it generate a lengthy continuation. In addition, GPT-2 outperforms other language models trained on specific domains (like Wikipedia, news, or books) without needing to use these domain-specific training datasets. On language tasks like question answering, reading comprehension, summarization, and translation, GPT-2 begins to learn these tasks from the raw text, using no task-specific training data. While scores on these downstream tasks are far from state-of-the-art, they suggest that the tasks can benefit from unsupervised techniques, given sufficient (unlabeled) data and compute.

[11] Radford et al. (2018). Improving language understanding by generative pre-training. OpenAI.

[12] Radford et al. (2019). Language models are unsupervised multitask learners. OpenAI Blog.

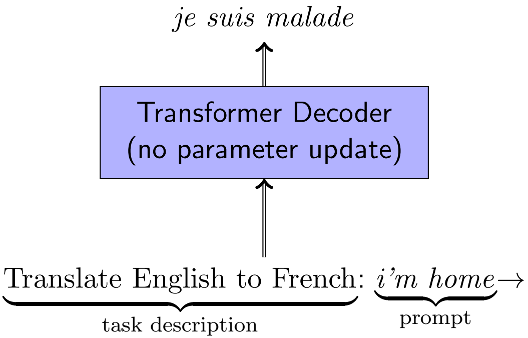
[13] `https://openai.com/research/better-language-models`

Decoder-Only Transformer: GPT-3

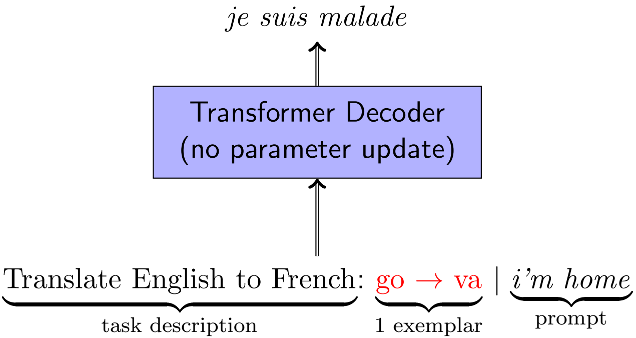
- A pretrained LM may generate the task output as a sequence *without parameter update*, conditional on an input sequence with
  1. the task description
  2. task-specific input-output examples
  3. a prompt (task input)
- This learning paradigm is called in-context learning.[14]
- GPT-3 (2020), 175B parameters:
  - uses the same Transformer decoder architecture in GPT-2 except that attention patterns are sparser at alternating layers
  - pretrained with 300B tokens (40TB of text data)
  - performs better with larger model size, where few-shot performance increases most rapidly

[14] Brown et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems.

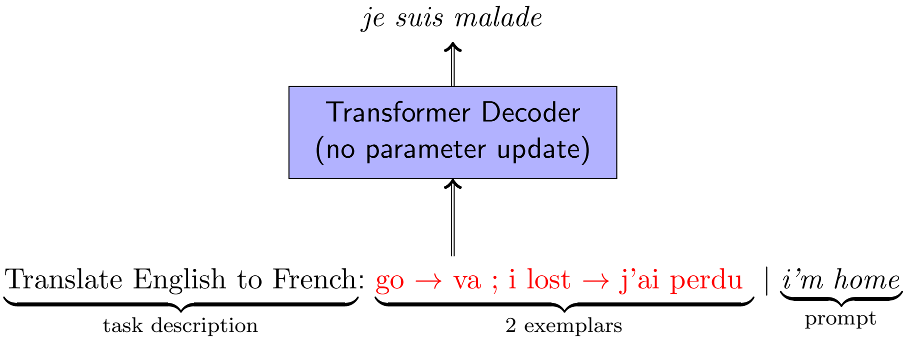
GPT-3 Zero-shot



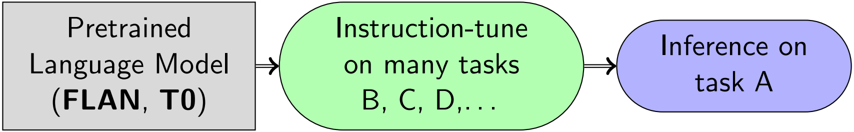
GPT-3 One-shot



GPT-3 Few-shot



Recall: Instruction-Tuning Method

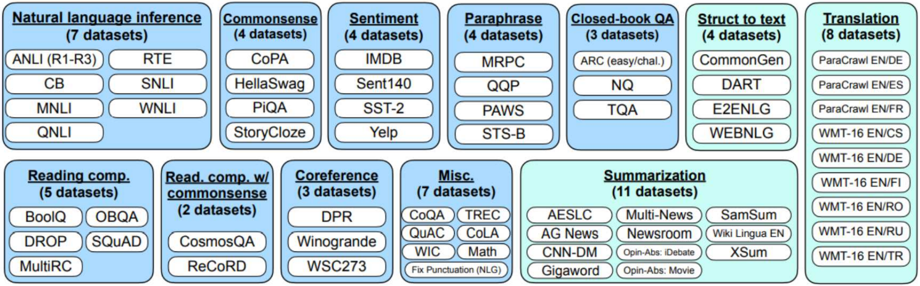


Instruction-tuning:

- Model learns to perform many tasks via natural language instructions
- Inference on unseen tasks
- **Zero-shot learning**

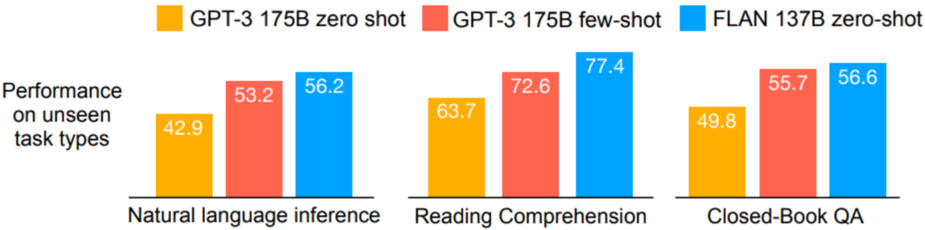
FLAN Instruction-Tuning

- Leverage the intuition that NLP tasks can be described via natural language instructions, such as
  - *Is the sentiment of this movie review positive or negative?*
  - *Translate "how are you" into Chinese.*



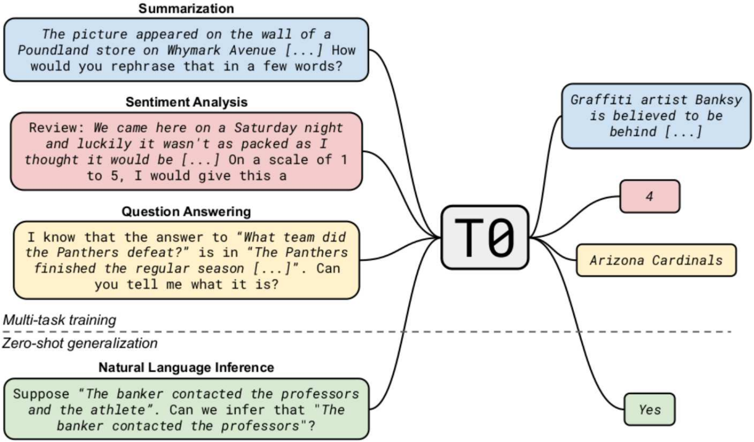
FLAN Instruction-Tuning

Take a pretrained **decoder-only** model (LaMDA-PT, 137B parameters) and perform instruction tuning–finetuning 60+ NLP datasets.[15]



[15] Wei et al. (2022) Finedtuned language models are zero-shot learners. ICLR.

T0 Instruction-Tuning



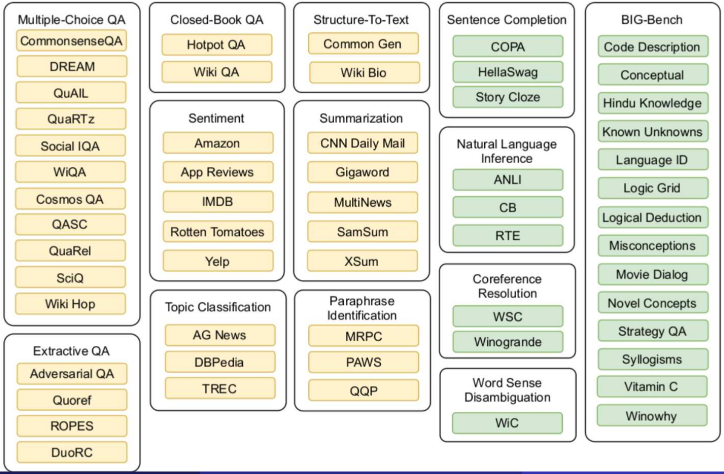
T0 is an **encoder-decoder** model.[16]

[16] Sanh et al. (2022) Multitask prompted training enables zero-shot task generalization. ICLR.

T0 Instruction-Tuning

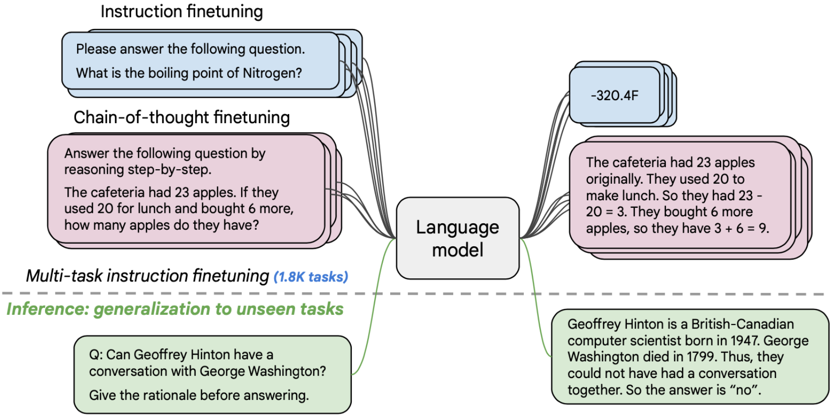
T0 (2022) 11B parameters, trained on multitask mixture of NLP datasets

- Each dataset is associated with multiple prompt templates to format examplars. T0 is as good as FLAN despite being 10x smaller.

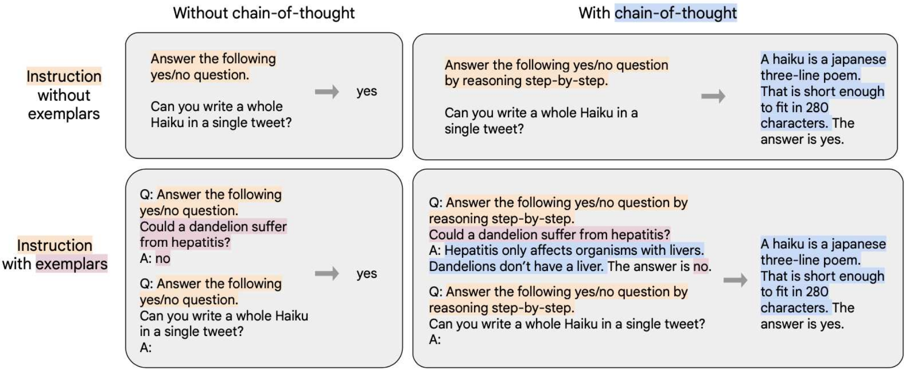


FLAN Instruction-Tuning – Chain of Thoughts

- FLAN-PaLM 540B instruction-finetuned on 1.8K tasks outperforms PaLM 540B by a large margin (+9.4% on average).
- FLAN-PaLM 540B achieves SOTA performance on several benchmarks, such as 75.2% on five-shot MMLU.



FLAN Instruction-Tuning – Chain of Thoughts



FLAN Instruction-Tuning – Chain of Thoughts

-	Random	25.0
-	Average human rater	34.5
May 2020	GPT-3 5-shot	43.9
Mar. 2022	Chinchilla 5-shot	67.6
Apr. 2022	PaLM 5-shot	69.3
Oct. 2022	**Flan-PaLM 5-shot**	**72.2**
	Flan-PaLM 5-shot: CoT + SC	**75.2**
-	Average human expert	89.8
	Jun. 2023 forecast (Hypermind)	73.2
	Jun. 2024 forecast (Hypermind)	75.0
	Jun. 2023 forecast (Metaculus)	82.7
	Jun. 2024 forecast (Metaculus)	87.6

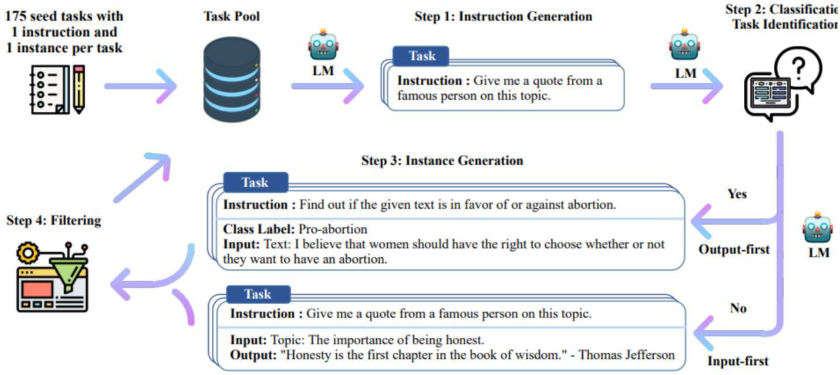
Table 1: Average 5-shot MMLU scores (%) for 57 tasks with model and human accuracy comparisons (Hendrycks et al., 2020). Forecasts were made in July 2022 by competitive human forecasters, regarding a single model (Steinhardt, 2021); see `https://prod.hypermind.com/ngdp/en/showcase2/showcase.html?sc=JSAI` and `https://www.metaculus.com/questions/11676/mmlu-sota-in-2023-2025/`. CoT + SC: chain-of-thought prompting with self-consistency (Wang et al., 2022b).

[17]

[17] Chung et al. (2022) Scaling instruction-finetuned language models.

Self-Instruct

Self-Instruct is a framework for improving the instruction-following capabilities of pretrained LLMs by bootsrapping off their own generations.[18]



[18] Wang et al. (2023) Aligning language models with self-generated instructions. ACL

Self-Instruct

- SuperNI is a benchmark consisting of 119 tasks with 100 instances in each task.
- Self-Instruct data is freely available (52K instructions).
- By finetuning GPT-3 on this data leads to a 33% absolute improvement over the original GPT-3.

Model	# Params	ROUGE-L
Vanilla LMs		
T5-LM	11B	25.7
GPT3	175B	6.8
Instruction-tuned w/o SUPERNI		
T0	11B	33.1
GPT3 + T0 Training	175B	37.9
$\text{GPT3}_{\text{SELF-INST}}$ (Ours)	175B	39.9
InstructGPT_{001}	175B	**40.8**
Instruction-tuned w/ SUPERNI		
T*k*-INSTRUCT	11B	46.0
GPT3 + SUPERNI Training	175B	49.5
$\text{GPT3}_{\text{SELF-INST}}$ + SUPERNI Training (Ours)	175B	**51.6**

Table 3: Evaluation results on *unseen* tasks from SUPERNI (§4.3). From the results, we see that 1 SELF-INSTRUCT can boost GPT3 performance by a large margin (+33.1%) and 2 nearly matches the performance of InstructGPT_{001}. Additionally, 3 it can further improve the performance even when a large amount of labeled instruction data is present.

SuperNI[19], Self-Instruct[20]

[19] Wang et al. (2022) Super-NaturalInstruction: Generalization via declarative instructions on 1600+ tasks. EMNLP

[20] `https://github.com/yizhongw/self-instruct`

Content

Emergent Ability

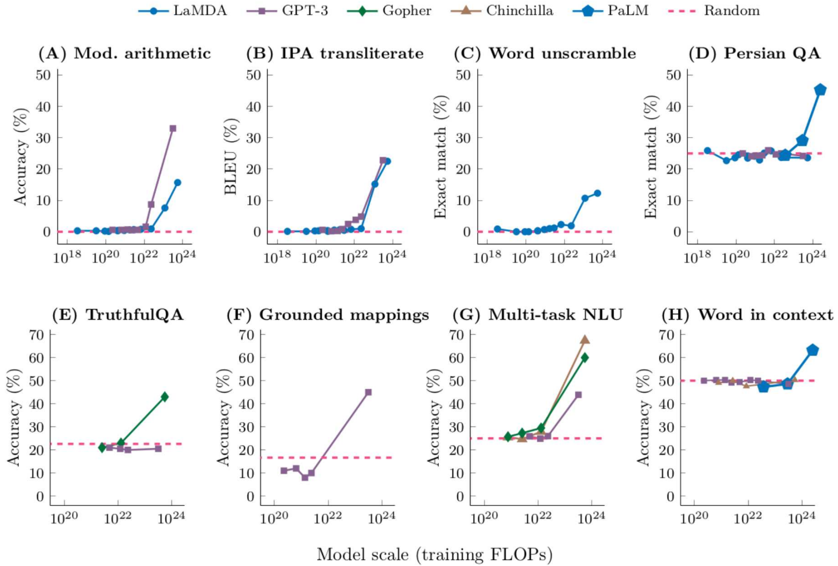
An ability is emergent if it is not present in smaller models but is present in larger models.[21]

Model sizes:

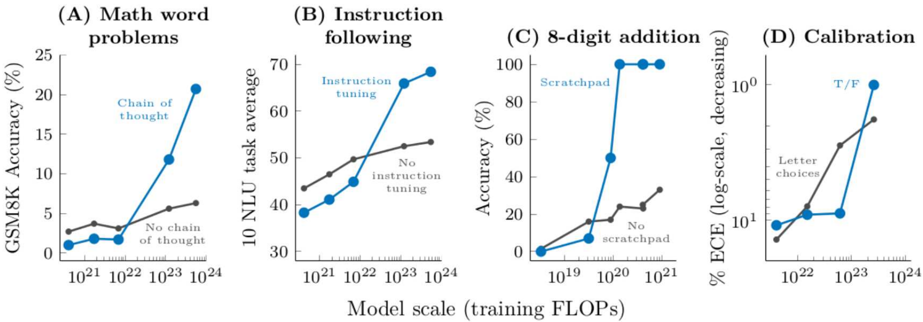
- GPT-3: $2 \cdot 10^{22}$ training FLOPs (13B parameters)
- LaMDA: 10^{23} training FLOPs (68B parameters)
- Gopher: $5 \cdot 10^{23}$ training FLOPs (280B parameters)
- PaLM: $2.5 \cdot 10^{24}$ FLOPs (540B parameters)

[21] Wei et al. (2022) Emergent abilities of large language models. TMLR.

Emergent Ability: Few-shot Prompting Setting



Emergent Ability: Specialized Prompting or Finetuning



- **Multi-step reasoning**: *chain-of-thought prompting*, guiding LLMs to produce a sequence of intermediate steps before giving the final answer.
- **Program execution**: computational tasks involving multiple steps, such as adding large numbers or executing computer programs.
- **Model calibration**: measures whether models can predict which questions they will be able to answer correctly.

Emergent Abilities

	Emergent scale			
	Train. FLOPs	Params.	Model	Reference
Few-shot prompting abilities				
• Addition/subtraction (3 digit)	2.3E+22	13B	GPT-3	Brown et al. (2020)
• Addition/subtraction (4-5 digit)	3.1E+23	175B		
• MMLU Benchmark (57 topic avg.)	3.1E+23	175B	GPT-3	Hendrycks et al. (2021a)
• Toxicity classification (CivilComments)	1.3E+22	7.1B	Gopher	Rae et al. (2021)
• Truthfulness (Truthful QA)	5.0E+23	280B		
• MMLU Benchmark (26 topics)	5.0E+23	280B		
• Grounded conceptual mappings	3.1E+23	175B	GPT-3	Patel & Pavlick (2022)
• MMLU Benchmark (30 topics)	5.0E+23	70B	Chinchilla	Hoffmann et al. (2022)
• Word in Context (WiC) benchmark	2.5E+24	540B	PaLM	Chowdhery et al. (2022)
• Many BIG-Bench tasks (see Appendix E)	Many	Many	Many	BIG-Bench (2022)
Augmented prompting abilities				
• Instruction following (finetuning)	1.3E+23	68B	FLAN	Wei et al. (2022a)
• Scratchpad: 8-digit addition (finetuning)	8.9E+19	40M	LaMDA	Nye et al. (2021)
• Using open-book knowledge for fact checking	1.3E+22	7.1B	Gopher	Rae et al. (2021)
• Chain-of-thought: Math word problems	1.3E+23	68B	LaMDA	Wei et al. (2022b)
• Chain-of-thought: StrategyQA	2.9E+23	62B	PaLM	Chowdhery et al. (2022)
• Differentiable search index	3.3E+22	11B	T5	Tay et al. (2022b)
• Self-consistency decoding	1.3E+23	68B	LaMDA	Wang et al. (2022b)
• Leveraging explanations in prompting	5.0E+23	280B	Gopher	Lampinen et al. (2022)
• Least-to-most prompting	3.1E+23	175B	GPT-3	Zhou et al. (2022)
• Zero-shot chain-of-thought reasoning	3.1E+23	175B	GPT-3	Kojima et al. (2022)
• Calibration via P(True)	2.6E+23	52B	Anthropic	Kadavath et al. (2022)
• Multilingual chain-of-thought reasoning	2.9E+23	62B	PaLM	Shi et al. (2022)
• Ask me anything prompting	1.4E+22	6B	EleutherAI	Arora et al. (2022)

Emergent Abilities

- A dataset of 4,550 questions and solutions from problem sets, midterm exams, and final exams across all MIT EECS courses.
  - GPT-3.5 successfully solves a third of the entire MIT curriculum.
  - GPT-4, with prompt engineering, achieves a perfect solve rate on a test set excluding questions based on images.

Exploring the MIT Mathematics and EECS Curriculum Using Large Language Models

Sarah J. Zhang* MIT sjzhang@mit.edu

Sam Florin* MIT sflorin@mit.edu

Ariel N. Lee Boston University ariellee@bu.edu

Eamon Niknafs Boston University en@bu.edu

Andrei Marginean MIT atmargi@mit.edu

Annie Wang MIT annewang@mit.edu

Keith Tyser Boston University ktyser@bu.edu

Zad Chin Harvard University zadchin@college.harvard.edu

Yann Hicke Cornell University ylh8@cornell.edu

Nikhil Singh MIT nsingh1@mit.edu

Madeleine Udell Stanford University udell@stanford.edu

Yoon Kim MIT yoonkim@mit.edu

Tonio Buonassisi MIT buonassi@mit.edu

Cost

- OPT and BLOOM (175B parameters) training required 34 days on 992 A100 80GB.
- LLaMA of Facebook (65B parameters) used 2,048 A100-80GB for a period of approximately 5 months, cost around 2,638 MWh.[22]
    - `https://github.com/facebookresearch/llama`
- PaLM's (Google's 540B LLM) training costs around $9M to $17M.
    - `https://blog.heim.xyz/palm-training-cost/`

[22] Touvron et al. (2023) LLaMA: Open and efficient foundation language models.

Conclusion

- LLMs have changed the NLP field completely in the last 5 years.
  - Single task $\Rightarrow$ multitask
  - Monolingual processing $\Rightarrow$ multilingual processing
  - Small models $\Rightarrow$ very large models
  - Small corpora $\Rightarrow$ collosal corpora
- Core technologies of LLMs:
  - Attention mechanism $\rightarrow$ the Transformers and their variants
  - Distributed and parallel processing using GPU/TPU
  - Large-scale optimization algorithms
- Current active research directions:
  - Model scaling; improved model architectures and training
  - Data scaling and selection;
  - Better techniques for understanding of prompting
  - Understanding emergence