In-context Learning & Scaling

Slido: https://app.sli.do/event/8EJbWibEpnen89kCSSWUnc

Yu Meng
University of Virginia
yumeng5@virginia.edu

Oct 22, 2025

UNIVERSITY of VIRGINIA

Overview of Course Contents

- Week 1: Logistics & Overview
- Week 2: N-gram Language Models
- Week 3: Word Senses, Semantics & Classic Word Representations
- Week 4: Word Embeddings
- Week 5: Sequence Modeling & Recurrent Neural Networks (RNNs)
- Week 6: Language Modeling with Transformers
- Week 8: Transformer and Pretraining
- Week 9: Large Language Models (LLMs) & In-context Learning
- Week 10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Reinforcement Learning for LLM Post-Training
- Week 13: LLM Agents + Course Summary
- Week 15 (after Thanksgiving): Project Presentations

Reminder

- Assignment 4 Out (due 11/03 11:59pm)

(Recap) Pretraining: Motivation

- Before pretraining became prevalent in NLP, most NLP models were trained from scratch on downstream task data
- **Data scarcity**: many NLP tasks do not have large labeled datasets available (costly to obtain)
- **Poor generalization**: models trained from scratch on specific tasks do not generalize well to unseen data or other tasks
- **Sensitivity to noise and randomness**: models are more likely to learn spurious correlations or be affected by annotation errors/randomness in training

(Recap) Pretraining: Motivation

- There are abundant text data on the web, with rich information of linguistic features and knowledge about the world
- Learning from these easy-to-obtain data greatly benefits various downstream tasks

UNIVERSITY of VIRGINIA

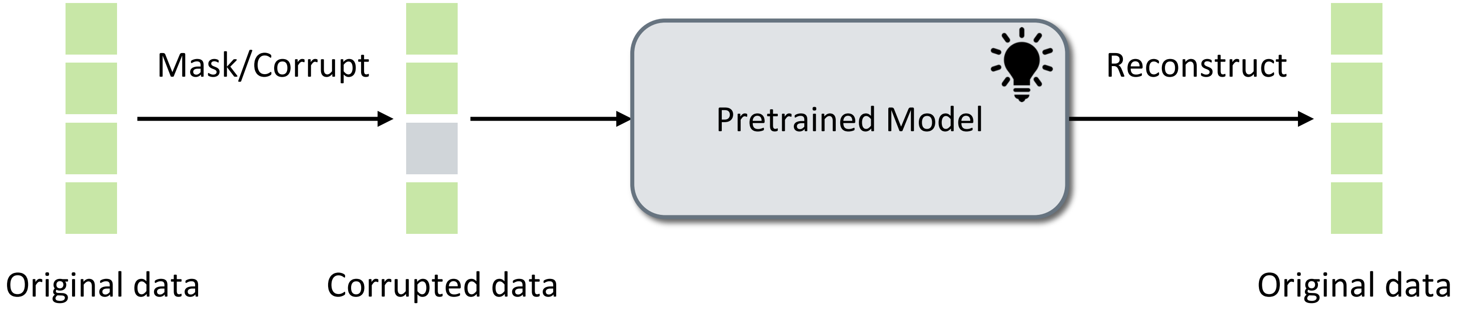
(Recap) Pretraining: Multi-Task Learning

- In my free time, I like to **{<u>run</u>, banana}** (*Grammar*)
- I went to the zoo to see giraffes, lions, and **{<u>zebras</u>, spoon}** (*Lexical semantics*)
- The capital of Denmark is **{<u>Copenhagen</u>, London}** (*World knowledge*)
- I was engaged and on the edge of my seat the whole time. The movie was **{<u>good</u>, bad}** (*Sentiment analysis*)
- The word for "pretty" in Spanish is **{<u>bonita</u>, hola}** (*Translation*)
- 3 + 8 + 4 = **{<u>15</u>, 11}** (*Math*)
- …

Examples from: https://docs.google.com/presentation/d/1hQUd3pF8_2Gr2Obc89LKjmHL0DlH-uof9M0yFVd3FA4/edit#slide=id.g28e2e9aa709_0_1

(Recap) Pretraining: Self-Supervised Learning

- Pretraining is a form of **self-supervised** learning
- Make a part of the input unknown to the model
- Use other parts of the input to reconstruct/predict the unknown part



No Human Supervision Needed!

(Recap) Pretraining + Fine-Tuning

- Pretraining: trained with pretext tasks on large-scale text corpora
- Fine-tuning (continue training): adjust the pretrained model's parameters with fine-tuning data
- Fine-tuning data can have different forms:
  - Task-specific labeled data (e.g., sentiment classification, named entity recognition)
  - (Multi-turn) dialogue data (i.e., instruction tuning)

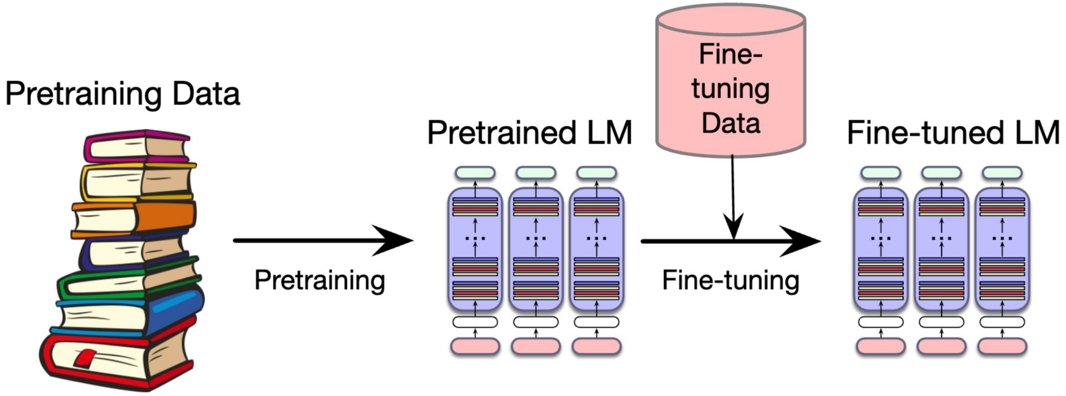


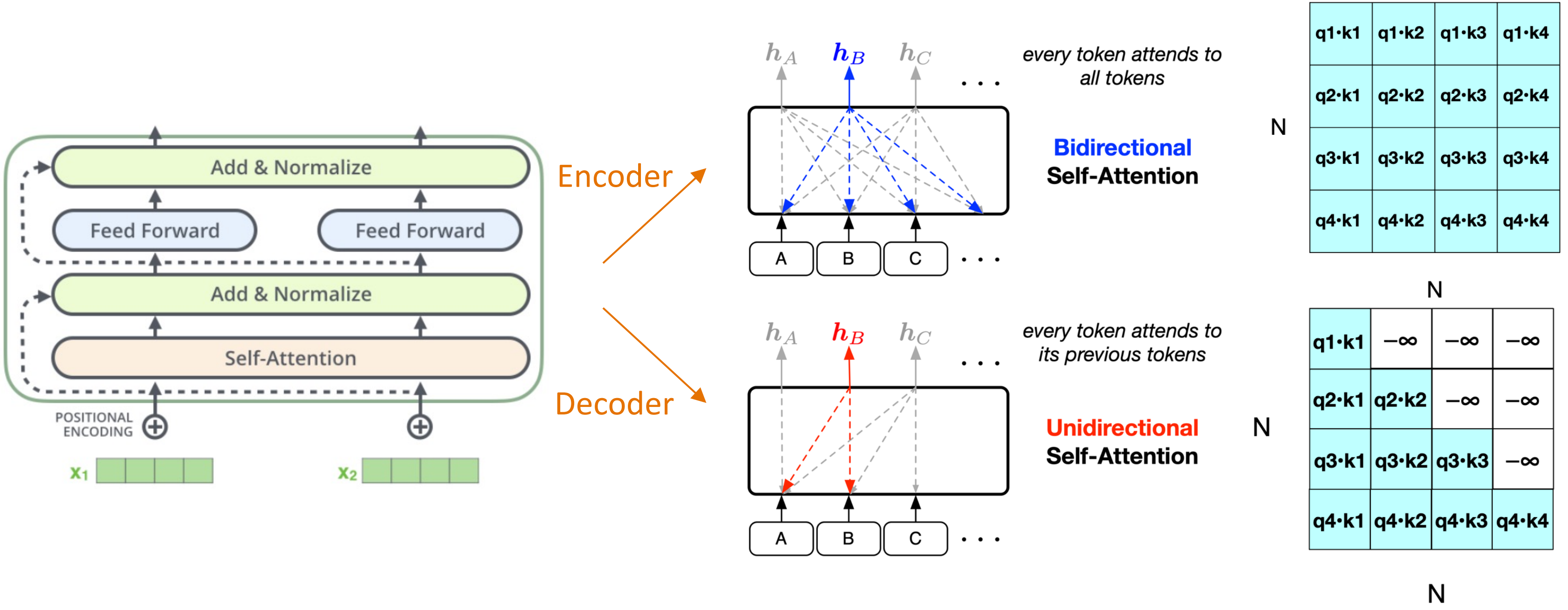
Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

(Recap) Transformer for Pretraining

- Transformer is the common backbone architecture for language model pretraining
- **Efficiency**: Transformer processes all tokens in a sequence simultaneously – fast and efficient to train, especially on large datasets
- **Scalability**: Transformer architectures have shown impressive scaling properties, with performance improving as model size and training data increase (more on this later!)
- **Versatility**: Transformer can be adapted for various tasks and modalities beyond just text, including vision, audio, and other multimodal applications

(Recap) Transformer Architectures

- Based on the type of self-attention, Transformer can be instantiated as
  - Encoder: Bidirectional self-attention
  - Decoder: Unidirectional self-attention



(Recap) Applications of Transformer Architectures

- Encoder (e.g., BERT):
  - Capture bidirectional context to learn each token representations
  - Suitable for natural language understanding (NLU) tasks
- Decoder (modern large language models, e.g., GPT):
  - Use prior context to predict the next token (conventional language modeling)
  - Suitable for natural language generation (NLG) tasks
  - Can also be used for NLU tasks by generating the class labels as tokens

NLU:
Text classification
Named entity recognition
Relation extraction
Sentiment analysis
…

NLG:
Text summarization
Machine translation
Dialogue system
Question answering
…

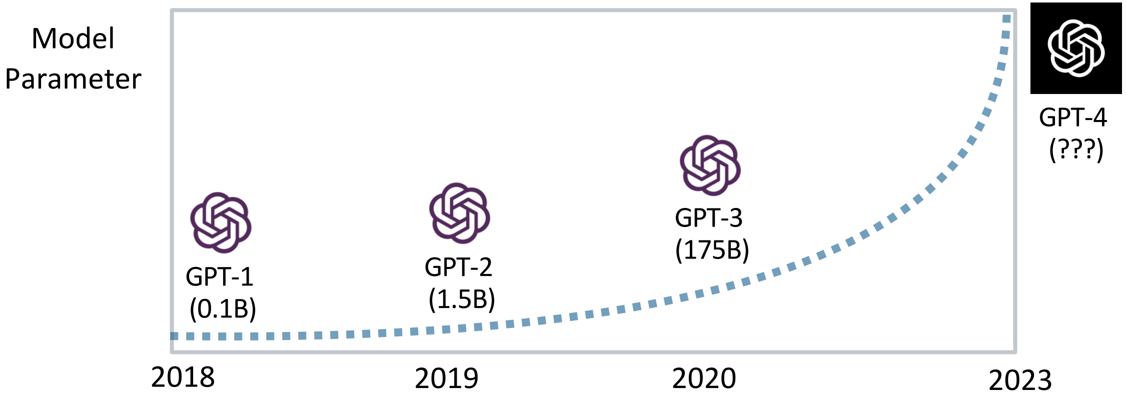
(Recap) Decoder Pretraining

- Decoder architecture is the prominent choice in large language models
- Pretraining decoders is first introduced in GPT (generative pretraining) models
- Follow the standard language modeling (cross-entropy) objective

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\log p_{\boldsymbol{\theta}}(x_i | x_1, x_2, \dots, x_{i-1})$$

(Recap) GPT Series

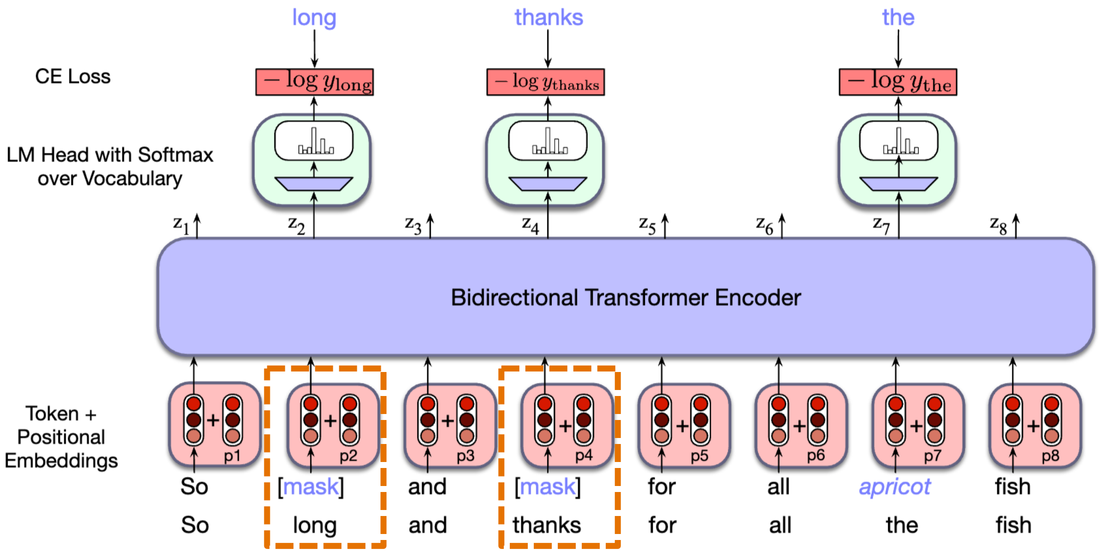
- GPT-1 (2018): 12 layers, 117M parameters, trained in ~1 week
- GPT-2 (2019): 48 layers, 1.5B parameters, trained in ~1 month
- GPT-3 (2020): 96 layers, 175B parameters, trained in several months



Papers: (GPT-1) https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
(GPT-2) https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
(GPT-3) https://arxiv.org/pdf/2005.14165.pdf

(Recap) Encoder Pretraining: BERT

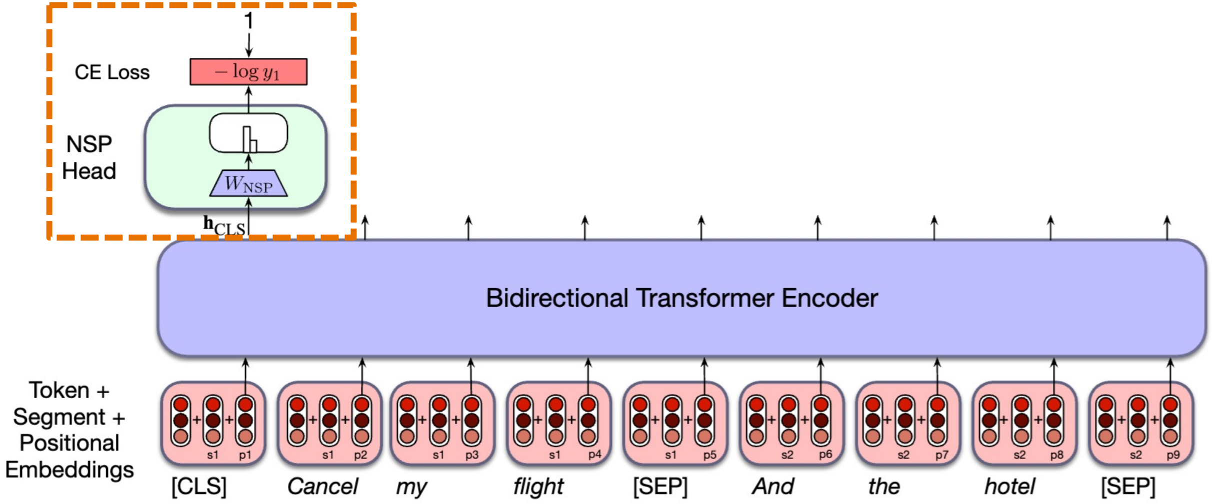
- BERT pretrains encoder models with bidirectionality
- **Masked language modeling** (MLM): With 15% words randomly masked, the model learns bidirectional contextual information to predict the masked words



BERT: https://arxiv.org/pdf/1810.04805.pdf Figure source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

(Recap) Encoder Pretraining: BERT

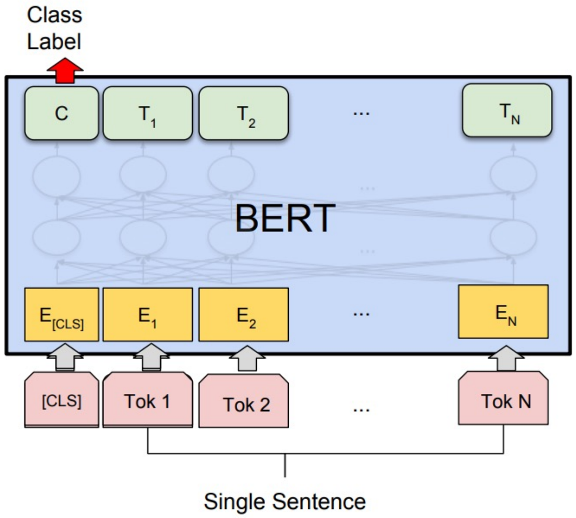
- **Next sentence prediction** (NSP): the model is presented with pairs of sentences
- The model is trained to predict whether each pair consists of an actual pair of adjacent sentences from the training corpus or a pair of unrelated sentences



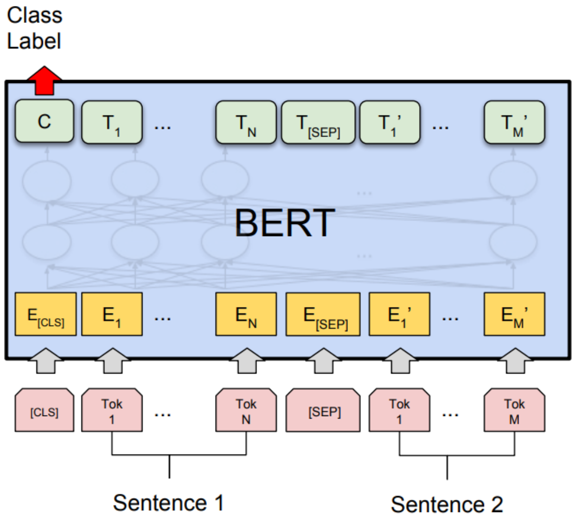
BERT: https://arxiv.org/pdf/1810.04805.pdf Figure source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

(Recap) BERT Fine-Tuning

- Fine-tuning pretrained BERT models takes different forms depending on task types
- Usually replace the LM head with a linear layer fine-tuned on task-specific data



Single sequence classification



Sequence-pair classification

UNIVERSITY of VIRGINIA

(Recap) BERT vs. GPT on NLU tasks

- BERT outperforms GPT-1 on a set of NLU tasks
- Encoders capture **bidirectional** contexts – build a richer understanding of the text by looking at both preceding and following words
- Are encoder models still better than state-of-the-art (large) decoder models?
  - LLMs can be as good as (if not better than) encoder models on NLU: Can ChatGPT Understand Too?
  - The sheer model size + massive amount of pretraining data compensate for LLMs' unidirectional processing

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

UNIVERSITY of VIRGINIA

Agenda

- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation
- In-context Learning
- Scaling Up LLMs

Prompting

- **Prompt**: initial user input/instructions given to the model to guide text generation
- Example (sentiment analysis):

$$P(\text{positive}|\boxed{\texttt{The sentiment of the sentence ``I like Jackie Chan" is:}})$$
$$P(\text{negative}|\boxed{\texttt{The sentiment of the sentence ``I like Jackie Chan" is:}})$$

prompt

- Example (question answering):

$$P(w|\boxed{\texttt{Q: Who wrote the book ``The Origin of Species"? A:}})$$

prompt

- **Prompting**: directly use trained LMs to generate text given user prompts (no fine-tuning)

For good prompting performance, we need **instruction-tuning** (later lectures)

Example source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

Prompt Engineering

- Some LMs (especially small ones) can be sensitive to specific formats of prompts
- Multiple prompts can make sense for the same task, but the resulting model performance might differ

$$P_1(a) = \text{It was \_\_\_\_. } a \qquad P_2(a) = \text{Just \_\_\_\_! } \| \ a$$

$$P_3(a) = a. \text{ All in all, it was \_\_\_\_.}$$

$$P_4(a) = a \ \| \text{ In summary, the restaurant is \_\_\_\_.}$$

Model predicts the masked word

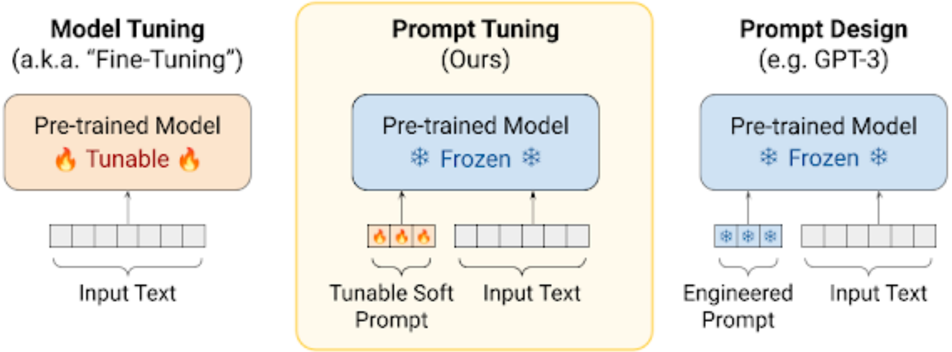
Prompt templates for BERT sentiment classification

- **Prompt engineering**: designing and refining prompts to achieve desired outcomes from LMs (e.g., manually tune on a validation set)
- A guide on prompt engineering: https://www.promptingguide.ai/

Figure source: https://aclanthology.org/2021.eacl-main.20.pdf

Prompt Tuning

- **Prompt tuning**: instead of manually testing the prompt design, consider prompt tokens as learnable model parameters ("soft prompts")
- Optimize a small amount of prompt token embeddings while keeping the LM frozen



- Prompt tuning is a parameter efficient fine-tuning (PEFT) method

Figure source: https://www.googblogs.com/guiding-frozen-language-models-with-learned-soft-prompts/

Parameter Efficient Fine-tuning (PEFT)

- Fine-tuning all model parameters is expensive

Pretrained weight
(can represent any module) $\boldsymbol{W}_0 \in \mathbb{R}^{d \times d}$

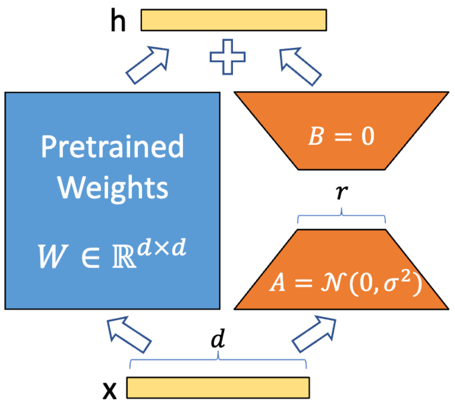
Fine-tuned weight $\boldsymbol{W}^* = \boldsymbol{W}_0 + \Delta \boldsymbol{W}, \quad \Delta \boldsymbol{W} \in \mathbb{R}^{d \times d}$

- Can we update only a small number of model parameters on fine-tuning data?

Parameter Efficient Fine-tuning: LoRA

- Assume the parameter update is **low-rank**
  - **Overparameterization**: large language models typically have many more parameters than strictly necessary to fit the training data
  - **Empirical observation**: parameter updates in neural networks tend to be low-rank in practice
- Solution: approximate weight updates with low-rank factorization

Freeze pretrained weights



$$\Delta \boldsymbol{W} \approx \boldsymbol{B}\boldsymbol{A}, \quad \boldsymbol{B} \in \mathbb{R}^{d \times r}, \quad \boldsymbol{A} \in \mathbb{R}^{r \times d}, \quad r \ll d$$

Low-rank approximation

LoRA: https://arxiv.org/pdf/2106.09685

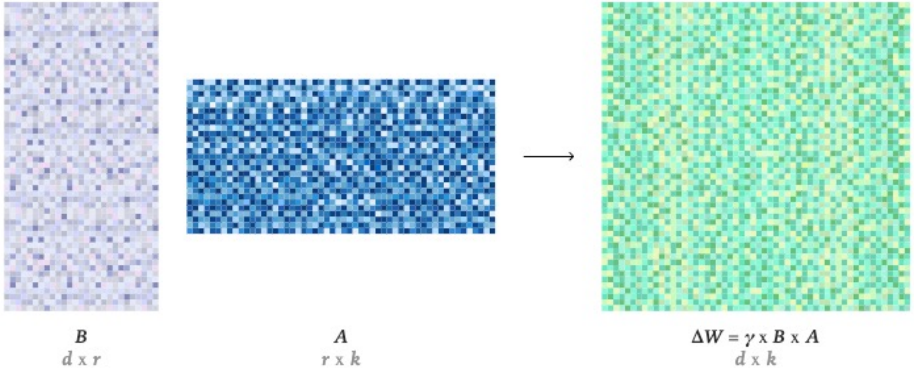
LoRA in Frontier Research

LoRA fine-tuning is effective for frontier LLM post-training (e.g., reinforcement learning)

LoRA Without Regret

John Schulman in collaboration with others at Thinking Machines

Sep 29, 2025



Blog post: https://thinkingmachines.ai/blog/lora/

Further Reading on PEFT

- Parameter-Efficient Transfer Learning for NLP [Houlsby et al., 2019]
- Prefix-Tuning: Optimizing Continuous Prompts for Generation [Li & Liang, 2021]
- The Power of Scale for Parameter-Efficient Prompt Tuning [Lester et al., 2021]
- GPT Understands, Too [Liu et al., 2021]

Agenda

- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation
- In-context Learning
- Scaling Up LLMs

Large Language Models (LLMs)

- The field of LLMs is rapidly evolving!
  - In 2018, BERT-large with 340 million parameters was considered large
  - In 2019, GPT-2 with 1.5 billion parameters was considered very large
  - In 2020, GPT-3 with 175 billion parameters set a new standard for “large”
- In 2025, how should we define LLMs?
- General definition:
  - Transformer-decoder architecture (or variants) that can generate text
  - Pretrained on vast and diverse general-domain corpora
  - With (at least) billions of parameters
  - General-purpose solvers for a wide range of NLP tasks and beyond

Decoding with LLMs

- **Decoding**: convert Transformer representations into natural language tokens
- Autoregressive decoding typically involves iterative **sampling** from LMs' output distributions, until an [EOS] token is generated

$$p_{\boldsymbol{\theta}}(w|x_1, x_2, \ldots, x_{i-1}) = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h}_{i-1}) = \left[\frac{\exp(\boldsymbol{u}_1 \cdot \boldsymbol{h}_{i-1})}{\sum_{j=1}^{|\mathcal{V}|} \exp(\boldsymbol{u}_j \cdot \boldsymbol{h}_{i-1})}, \ldots, \frac{\exp(\boldsymbol{u}_{|\mathcal{V}|} \cdot \boldsymbol{h}_{i-1})}{\sum_{j=1}^{|\mathcal{V}|} \exp(\boldsymbol{u}_j \cdot \boldsymbol{h}_{i-1})}\right]$$

Model parameters   Unembedding matrix   Hidden states at token $i-1$

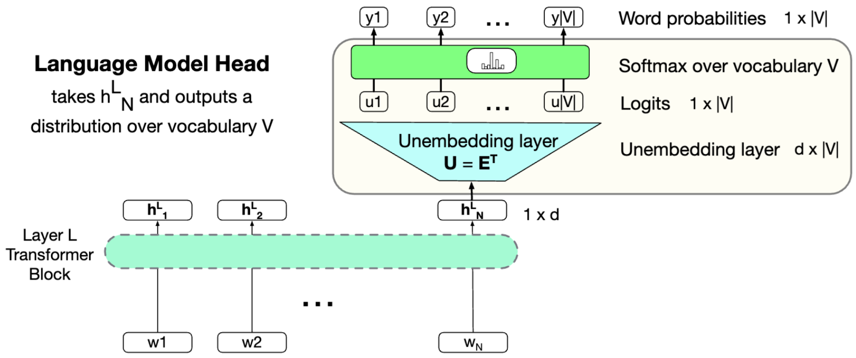


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

UNIVERSITY of VIRGINIA

Greedy Decoding

- Always pick the token with the highest probability estimated by the LM for every step

$$x_i \leftarrow \arg\max_w p_{\boldsymbol{\theta}}(w|x_1, x_2, \ldots, x_{i-1})$$

- Pros:
    - Simplicity: easy to implement and understand
    - Deterministic: guarantee the same output given the same input
    - Efficient: makes only one (simple) decision at each step w/o additional operations
- Cons:
    - Suboptimal solutions: may not find the globally optimal sequence
    - Lack of diversity: cannot produce multiple outputs given the same input

Top-k Sampling

- Motivation: Instead of choosing the single most probable word to generate, sample from the top-k most likely tokens (candidates) – avoid generating low probability tokens
- k is a hyperparameter (typically 5-10)

Compute the probability distribution only over the top-k tokens

$$p_{\boldsymbol{\theta}}(w|x_1, x_2, \dots, x_{i-1}) = \text{softmax}(\boldsymbol{U}_{\text{top-}k}\boldsymbol{h}_{i-1}) = \left[\frac{\exp(\boldsymbol{u}_1 \cdot \boldsymbol{h}_{i-1})}{\sum_{j=1}^{k} \exp(\boldsymbol{u}_{\text{top-}j} \cdot \boldsymbol{h}_{i-1})}, \dots, \frac{\exp(\boldsymbol{u}_{\text{top-}k} \cdot \boldsymbol{h}_{i-1})}{\sum_{j=1}^{k} \exp(\boldsymbol{u}_{\text{top-}j} \cdot \boldsymbol{h}_{i-1})}\right]$$

Sample from the top-k tokens $\quad x_i \sim p_{\boldsymbol{\theta}}(w|x_1, x_2, \dots, x_{i-1})$

- With $k = 1$, top-k sampling is equivalent to greedy decoding

Nucleus (Top-p) sampling

- Top-k sampling does not account for the shape of the probability distribution
  - For the next-token distribution of “the 46th US president Joe”, top-k sampling may consider more tokens than necessary
  - For the next-token distribution of “the spacecraft”, top-k sampling may consider fewer tokens than necessary
- Nucleus sampling sets cutoff based on the top-p percent of the probability mass
- p is a hyperparameter (typically 0.9)
- Top-p vocabulary is the smallest set of words such that

$$\sum_{w \in \mathcal{V}_{\text{top-p}}} p(w|x_1, x_2, \ldots, x_{i-1}) \geq p$$

- Sample from the top-p vocabulary in a similar way as top-k sampling

Temperature Sampling

- Intuition comes from thermodynamics
  - A system at a high temperature is flexible and can explore many possible states
  - A system at a lower temperature is likely to explore a subset of lower energy (better) states
- Reshape the probability distribution by incorporating a temperature hyperparameter

$$p_{\boldsymbol{\theta}}(w|x_1, x_2, \ldots, x_{i-1}) = \text{softmax}(\boldsymbol{U}\boldsymbol{h}_{i-1}/\tau) = \left[\frac{\exp(\boldsymbol{u}_1 \cdot \boldsymbol{h}_{i-1}/\tau)}{\sum_{j=1}^{|\mathcal{V}|}\exp(\boldsymbol{u}_j \cdot \boldsymbol{h}_{i-1}/\tau)}, \ldots, \frac{\exp(\boldsymbol{u}_{|\mathcal{V}|} \cdot \boldsymbol{h}_{i-1}/\tau)}{\sum_{j=1}^{|\mathcal{V}|}\exp(\boldsymbol{u}_j \cdot \boldsymbol{h}_{i-1}/\tau)}\right]$$

- With $\tau \to 0$, temperature sampling approaches greedy decoding

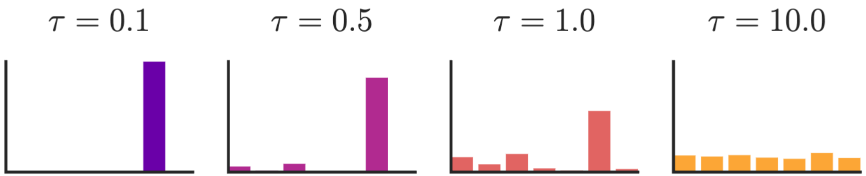


Figure source: https://arxiv.org/pdf/1611.01144v5

Practical Considerations of Decoding Algorithms

- If aiming for simplicity and efficiency without diversity requirements, use greedy decoding
- If multiple responses are required for the same input, use sampling-based decoding
  - Top-p is usually better than Top-k
  - Temperature sampling is commonly used
  - Top-p can be used together with temperature sampling

Agenda

- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation
- In-context Learning
- Scaling Up LLMs

In-context Learning

- In-context learning is a type of few-shot learning
  - User provides a few examples of input-output pairs in the prompt
  - The model uses given examples to predict the output for new, similar inputs
- First studied in the GPT-3 paper
- No model parameter updates

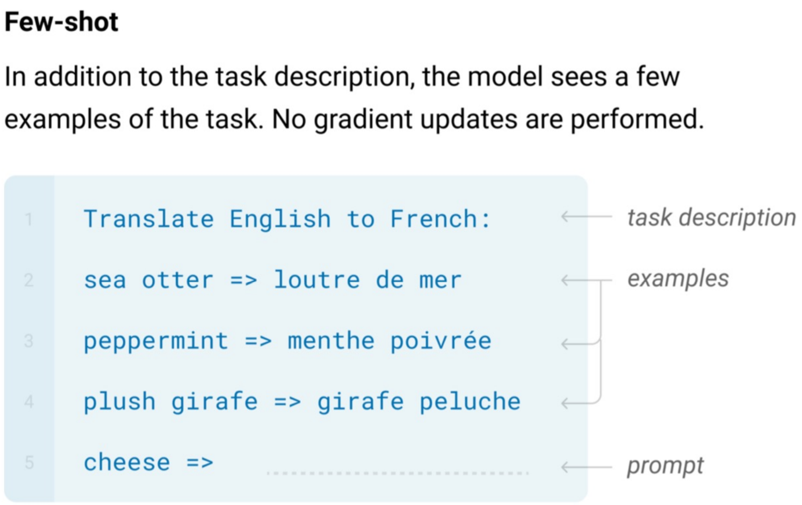


Figure source: https://arxiv.org/pdf/2005.14165

UNIVERSITY of VIRGINIA

In-context Learning Demo

Prompt: Swap the second and the penultimate letter of the following word: pothyn

Swap the second and the penultimate letter of the following word: pothyn

`llama-3.3-70b-instruct`

The word is "pothyn".

The second letter is "o" and the penultimate letter is "y" (the last letter is "n").

Swapping them results in "pythno" and then "pythyn".

Wrong generation only given the prompt

Figure source: https://lmarena.ai/

In-context Learning Demo

Prompt: Directly answer the last one (swapping the second and the penultimate letter of the following words): tarehd -> thread, revir -> river, pothyn ->

Directly answer the last one (swapping the second and the penultimate letter of the following words): tarehd → thread, revir → river, pothyn →

llama-3.3-70b-instruct

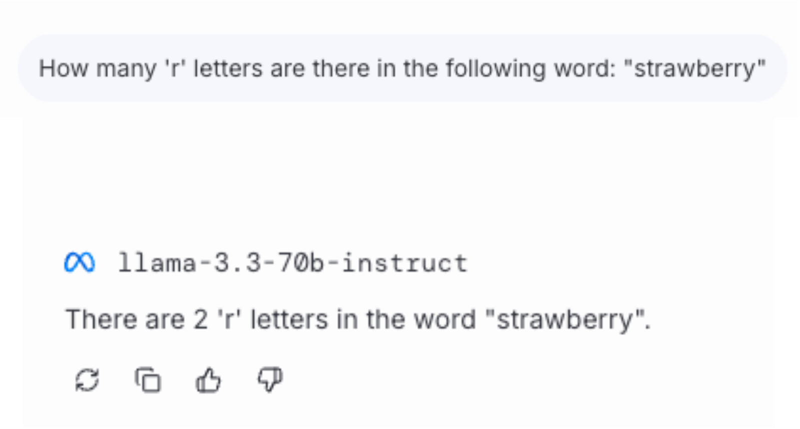
pothyn → python

Correctly learn from the in-context examples

Figure source: https://lmarena.ai/

In-context Learning Demo

Prompt: How many 'r' letters are there in the following word: "strawberry"



Wrong generation only given the prompt

Figure source: https://lmarena.ai/

In-context Learning Demo

Prompt: Count how many 'r' letters are there in the following words: "red": 1, "roar": 2, "strawberry":

Count how many 'r' letters are there in the following words: "red": 1, "roar": 2, "strawberry":

llama-3.3-70b-instruct

Let's count the 'r' letters in the word "strawberry":

1. s - no 'r'
2. t - no 'r'
3. r - 1 'r'
4. a - no 'r'
5. w - no 'r'
6. b - no 'r'
7. e - no 'r'
8. r - 1 'r'
9. r - 1 'r'
10. y - no 'r'

There are 3 'r' letters in the word "strawberry".

Correctly learn from the in-context examples

Figure source: https://lmarena.ai/

Further Reading on In-context Learning

- An Explanation of In-context Learning as Implicit Bayesian Inference [Xie et al., 2021]
- Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? [Min et al., 2022]
- What Can Transformers Learn In-Context? A Case Study of Simple Function Classes [Garg et al., 2022]
- What learning algorithm is in-context learning? Investigations with linear models [Akyurek et al., 2023]

UNIVERSITY of VIRGINIA

Agenda

- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation
- In-context Learning
- Scaling Up LLMs

Scaling Up Pretraining Data

The Pile: 22 sub-datasets (> 800GB), a common choice for pretraining corpus

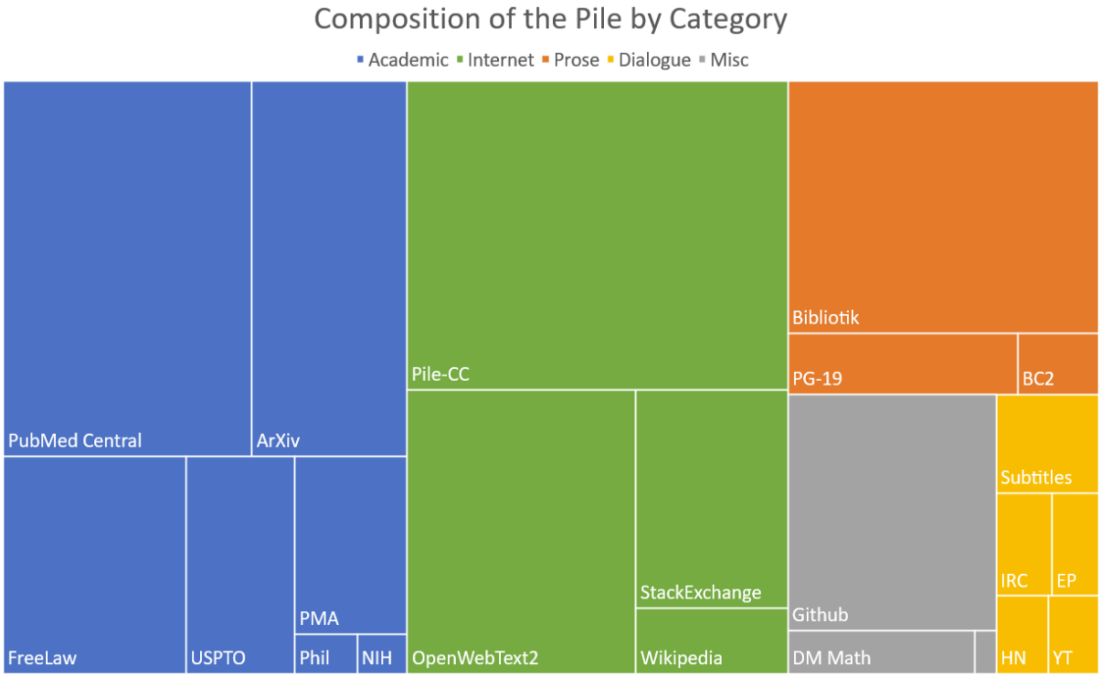
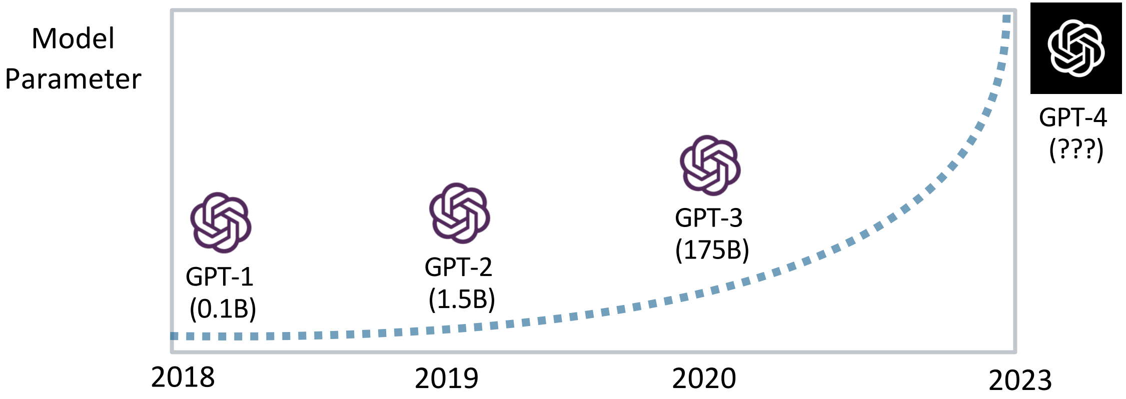


Figure source: https://arxiv.org/pdf/2101.00027

Scaling Up Model Sizes

- GPT-1 (2018): 12 layers, 117M parameters, trained in ~1 week
- GPT-2 (2019): 48 layers, 1.5B parameters, trained in ~1 month
- GPT-3 (2020): 96 layers, 175B parameters, trained in several months



Papers: (GPT-1) https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
(GPT-2) https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
(GPT-3) https://arxiv.org/pdf/2005.14165.pdf

UNIVERSITY of VIRGINIA

Emergent Ability

- Larger models develop **emergent abilities**
  - Skills or capabilities that were not explicitly learned but arise as a result of model capacity
  - Larger models demonstrate surprising abilities in challenging tasks even when they were not explicitly trained for them
- Emergent capabilities typically become noticeable only when the model size reaches a certain threshold (cannot be predicted by small model's performance)

Emergent Abilities of Large Language Models

Jason Wei[1] *jasonwei@google.com*
Yi Tay[1] *yitay@google.com*
Rishi Bommasani[2] *nlprishi@stanford.edu*
Colin Raffel[3] *craffel@gmail.com*
Barret Zoph[1] *barretzoph@google.com*
Sebastian Borgeaud[4] *sborgeaud@deepmind.com*
Dani Yogatama[4] *dyogatama@deepmind.com*
Maarten Bosma[1] *bosma@google.com*
Denny Zhou[1] *dennyzhou@google.com*
Donald Metzler[1] *metzler@google.com*
Ed H. Chi[1] *edchi@google.com*
Tatsunori Hashimoto[2] *thashim@stanford.edu*
Oriol Vinyals[4] *vinyals@deepmind.com*
Percy Liang[2] *pliang@stanford.edu*
Jeff Dean[1] *jeff@google.com*
William Fedus[1] *liamfedus@google.com*

[1] *Google Research* [2] *Stanford University* [3] *UNC Chapel Hill* [4] *DeepMind*

Paper: https://arxiv.org/pdf/2206.07682

Experiment Setting

- Consider the **few-shot in-context learning** paradigm
- Consider an ability to be **emergent** when a model has **random** performance until a certain scale, after which performance increases to **well-above random**
- Abilities to test
  - Arithmetic: addition, subtraction, multiplication
  - Transliteration
  - Recover a word from its scrambled letters
  - Persian question answering
  - Question answering (truthfully)
  - Grounded conceptual mappings
  - Multi-task understanding (math, history, law, …)
  - Contextualized semantic understanding

Performance vs. Model Scale

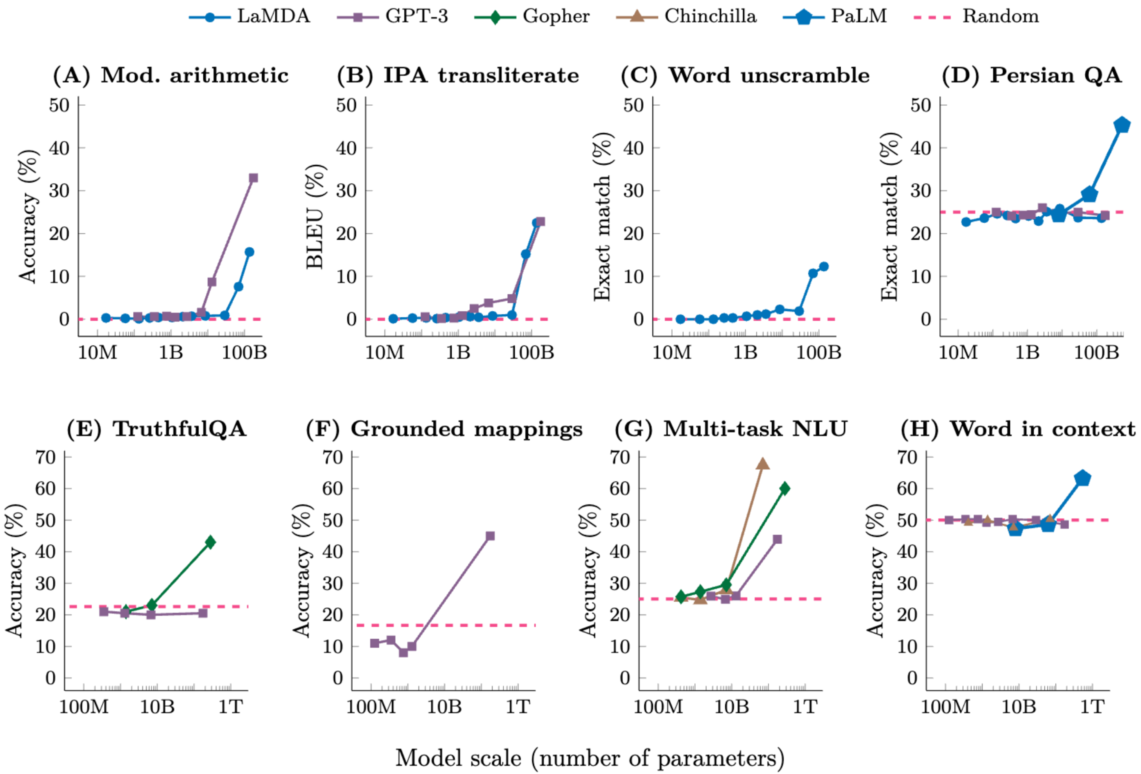


Figure source: https://arxiv.org/pdf/2206.07682

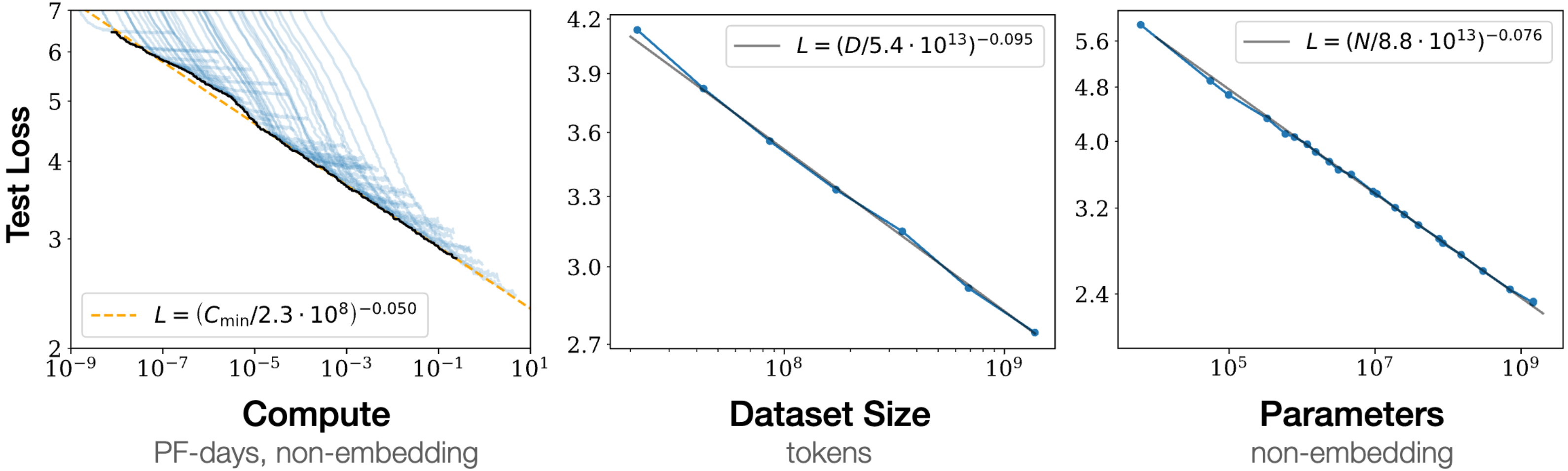
Models exhibit random performance until a certain scale, after which performance significantly increases

Scaling Laws of LLMs

- (Pretrained) LLM performance is mainly determined by 3 factors
  - Model size: the number of parameters
  - Dataset size: the amount of training data
  - Compute: the amount of floating point operations (FLOPs) used for training
- Scaling up LLMs involves scaling up the 3 factors
  - Add more parameters (adding more layers or having more model dimensions or both)
  - Add more data
  - Train for more iterations
- **Scaling laws**: study the correlation between the cross-entropy language modeling loss and the above three factors
- How to optimally allocate a fixed compute budget?

Scaling Laws of LLMs

Performance has a power-law relationship with each of the three scale factors (model size, dataset size, compute) when not bottlenecked by the other two



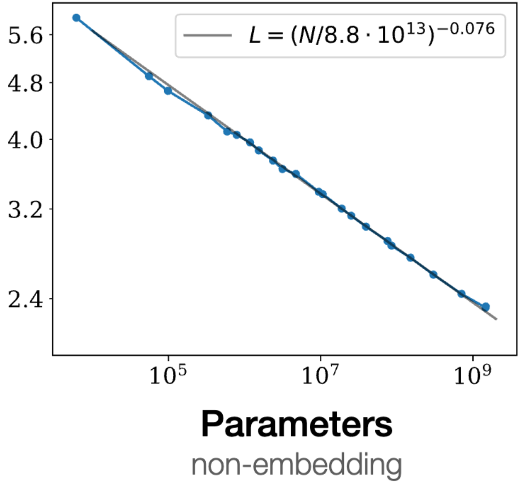
Paper: https://arxiv.org/pdf/2001.08361

Scaling Model Parameters

- Language model loss vs. models with a limited number of parameters (N)
  - Only count non-embedding parameters
  - Infinite compute: trained to convergence
  - Infinite dataset: trained with sufficiently large datasets
- Performance depends strongly on scale, weakly on model shape (depth vs. width)

$$\mathcal{L}(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad \alpha_N \approx 0.076, \quad N_c \approx 8.8 \times 10^{13}$$

Model parameters (non-embedding)

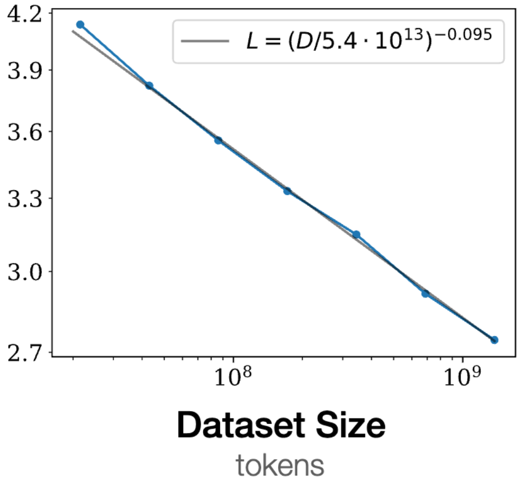


Scaling Dataset Size

- Language model loss vs. a limited dataset size (D)
  - Infinite model size: sufficiently large model
  - With appropriate early stopping: avoid overfitting to the training data

$$\mathcal{L}(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}, \quad \alpha_D \approx 0.095, \quad D_c \approx 5.4 \times 10^{13}$$

Dataset size
(# of tokens)

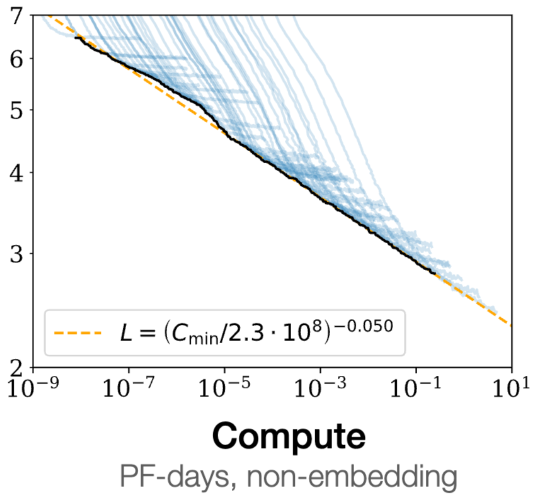


Scaling Training Compute

- Language model loss vs. a limited amount of compute (C)
  - Infinite dataset size: sufficiently large training corpus
  - Optimal model size: can effectively learn the data and not excessively compute-consuming

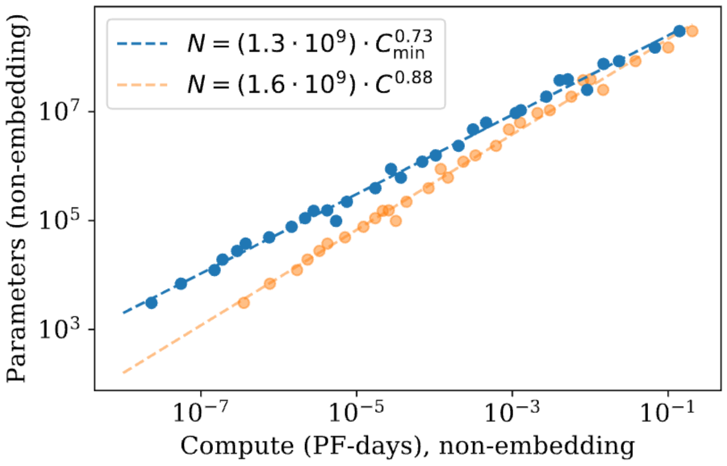
$$\mathcal{L}(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}, \quad \alpha_C \approx 0.050, \quad C_c \approx 3.1 \times 10^8$$

Compute
(# Peta-FLOP days)

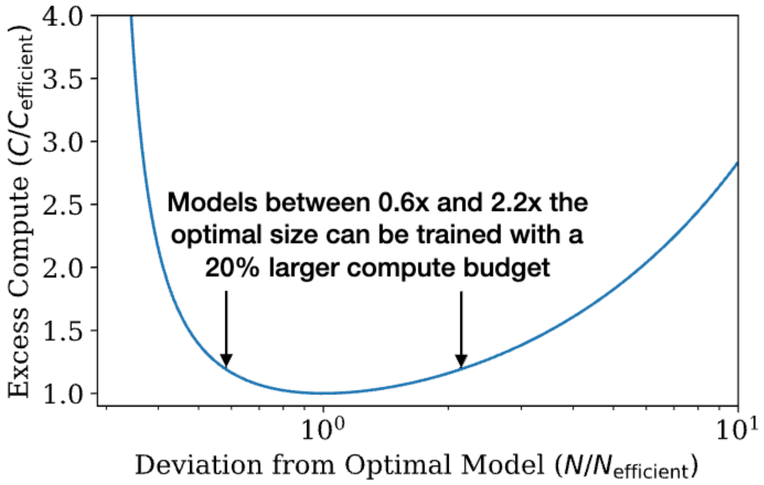


Optimal Model Size

- Given a specific amount of training compute C, what's the optimal model size $N(C)$ that leads to minimal language modeling loss?
- $N(C)$ can be fit with a power-law wrt C
- Additional compute needs to be used when model size is suboptimal



$$N(C_{\min}) \propto (C_{\min})^{0.73}$$



Further Reading on Scaling LLMs

- Training Compute-Optimal Large Language Models [Hoffmann et al., 2022]
- Scaling Data-Constrained Language Models [Muennighoff et al., 2023]
- Are Emergent Abilities of Large Language Models a Mirage? [Schaeffer et al., 2023]

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu