Natural Language Processing (NLP) and Large Language Models (LLMs)
Lecture 8-3: Efficient Adaptation

Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

WISE @ XMU

2025 年 5 月 15 日

Adapted in part from CS 224n, Dive into deep learning

Pretrained foundation models

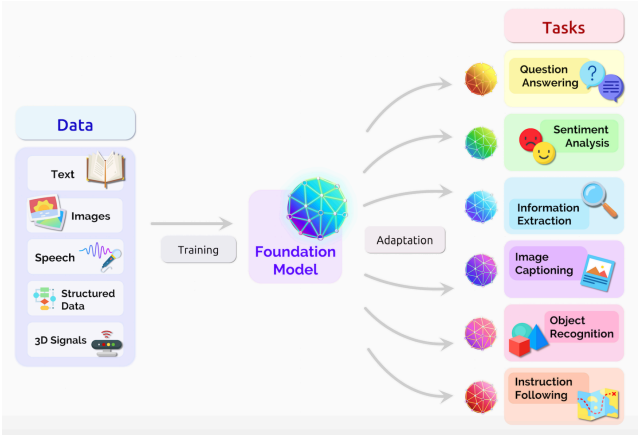


Figure is from Bommasani et al., 2021

Zero-Shot Learning

No examples, pure generalization

Definition:

- Model solves tasks **without any labeled examples**.
- Uses pretrained knowledge (e.g., GPT-3, CLIP).

Example (Text Classification):

Prompt

```
Classify the sentence into [joy, anger, fear]:
"I missed my flight and lost my luggage!"
Output: anger
```

Why It Matters:

- Handles **unseen categories** (e.g., new product reviews).
- No need for task-specific data collection.

One-Shot Learning

Learn from a single example

Definition:

- Model generalizes from **just 1 example**.

Example (Translation):

Prompt

```
Translate English to French:
"Hello" → "Bonjour"
New: "How are you?" →
```

Few-Shot Learning

Learn from a handful of examples

Definition:

- Model uses **2–5 examples** per class.
- Balances prior knowledge and new data.

Example (Translation):

Prompt

```
Translate English to French:
1. "Hello" → "Bonjour"
2. "Goodbye" → "Au revoir"
3. "Thank you" → "Merci"
New: "How are you?" →
```

Why It Matters:

- Practical for low-resource domains.

Few-shot learning v.s. finetuning

- Few-shot learning is sometimes referred to as in-context learning in certain contexts.

Prompt

```
Translate English to French:
1. "Hello" → "Bonjour"
2. "Goodbye" → "Au revoir"
3. "Thank you" → "Merci"
New: "How are you?" →
```

finetuning

```
Translate English to French:
1. "Hello" → "Bonjour"→ gradient update →
2. "Goodbye" → "Au revoir"→ gradient update →
3. "Thank you" → "Merci"→ gradient update
New: "How are you?" →
```

Prompting is not all you need!

Prompt

```
Note that:
1. 4 = 2 + 2
2. 6 = 3 + 3
3. 8 = 3 + 5
New: Show that "Every positive even integer can be written as the sum of two primes."
```

- Some tasks remain challenging for LLMs —and even for humans!
- In particular, LLMs often struggle with reasoning tasks.
- Carefully designed prompts can significantly influence model performance.

Few shot Chain-of-thought (COT) prompting

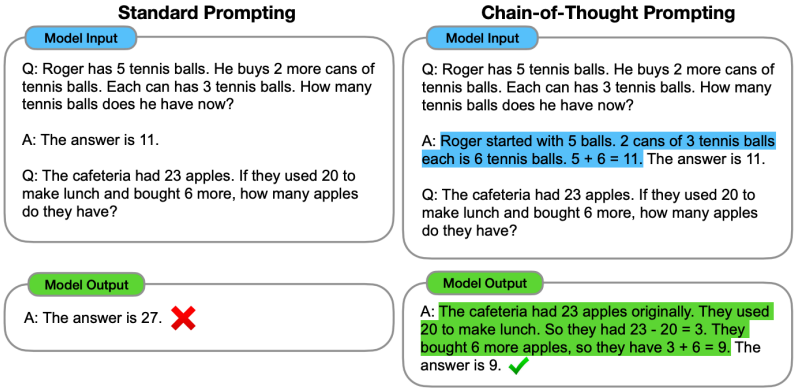
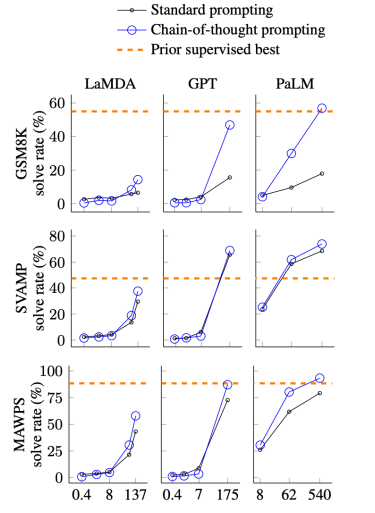


Figure: An example of COT prompting from Wei et al..

COT prompting is an emergent ability of increasing model scale (Wei et al.)



Zero-shot COT prompting

Large Language Models are Zero-Shot Reasoners

Takeshi Kojima
The University of Tokyo
t.kojima@weblab.t.u-tokyo.ac.jp

Shixiang Shane Gu
Google Research, Brain Team

Machel Reid
Google Research*

Yutaka Matsuo
The University of Tokyo

Yusuke Iwasawa
The University of Tokyo

Abstract

Pretrained large language models (LLMs) are widely used in many sub-fields of natural language processing (NLP) and generally known as excellent *few-shot* learners with task-specific exemplars. Notably, chain of thought (CoT) prompting, a recent technique for eliciting complex multi-step reasoning through step-by-step answer examples, achieved the state-of-the-art performances in arithmetics and symbolic reasoning, difficult *system-2* tasks that do not follow the standard scaling laws for LLMs. While these successes are often attributed to LLMs' ability for few-shot learning, we show that LLMs are decent *zero-shot* reasoners by simply adding "Let's think step by step" before each answer. Experimental

Zero-shot COT prompting

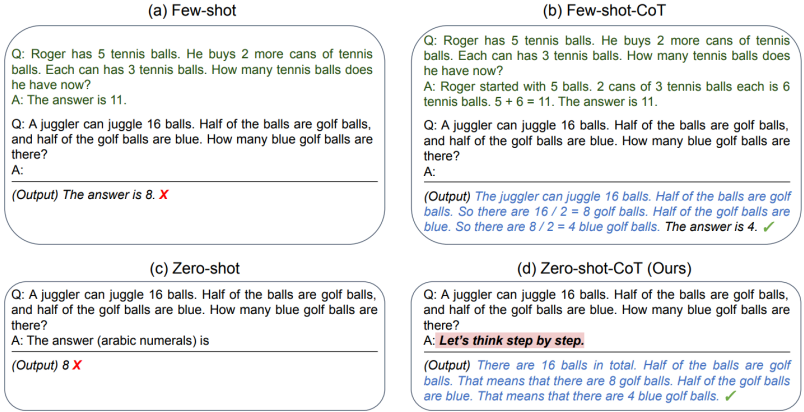


Figure is from Kojima et al.

Reasoning Models Are Born to Reason

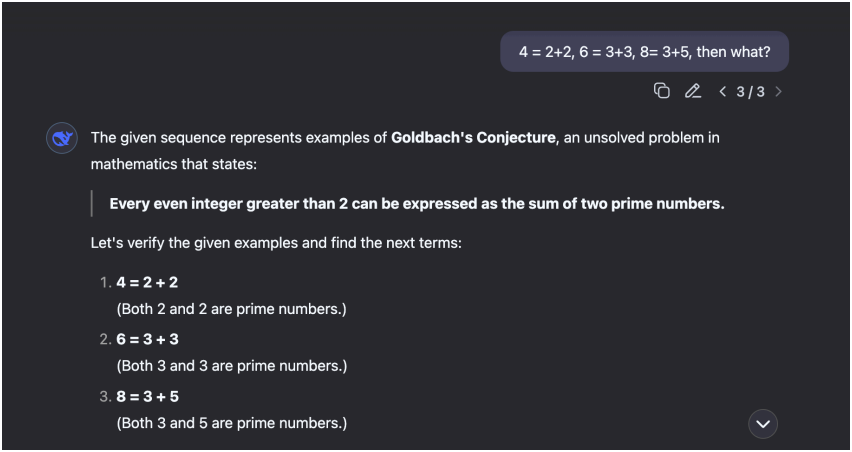
A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

5 / 5

Let's break down the problem step by step.

1. **Total Balls:**
   - The juggler can juggle a total of **16 balls**.
2. **Golf Balls:**
   - Half of the balls are golf balls.
   - Number of golf balls = $\frac{1}{2} \times 16 = 8$.
3. **Blue Golf Balls:**
   - Half of the golf balls are blue.
   - Number of blue golf balls = $\frac{1}{2} \times 8 = 4$.

We don't need "step by step" anymore



Prompt Wording Matters

Do Prompt-Based Models Really Understand the Meaning of Their Prompts?

Albert Webson[1,2] and Ellie Pavlick[1]

{albert_webson, ellie_pavlick}@brown.edu
[1]Department of Computer Science, Brown University
[2]Department of Philosophy, Brown University

mans learn faster when provided with task instructions expressed in natural language. In this study, we experiment with over 30 prompt templates manually written for natural language inference (NLI). We find that models can learn just as fast with many prompts that are intentionally irrelevant or even pathologically misleading as they do with instructively "good" prompts. Further, such patterns hold even for models as large as 175 billion parameters (Brown et al., 2020) as well as the recently

Prompt tuning

The Power of Scale for Parameter-Efficient Prompt Tuning

Brian Lester* **Rami Al-Rfou** **Noah Constant**
Google Research
{brianlester,rmyeid,nconstant}@google.com

Abstract

In this work, we explore "prompt tuning," a simple yet effective mechanism for learning "soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation and can be tuned to incorporate signals from any number of labeled examples. Our end-to-end learned approach outperforms

Recap: Language Models and Loss Functions

- For a sequence $\mathbf{x}$ of arbitrary length, a language model p assigns a probability $p_\theta(\mathbf{x})$ to $\mathbf{x}$, parameterized by θ.
- In downstream tasks such as sentiment analysis, each input sequence is associated with a label.
- A loss function can be defined as $\ell(p_\theta(\mathbf{x}), y)$ to guide model training.

Mathematical Definition of Prompt Tuning

- We prepend a trainable prompt $\mathbf{p}$ to each input sequence $\mathbf{x}$.
- The input sentence now becomes $\mathbf{x}' = [\mathbf{p}, \mathbf{x}]$.
- The loss function is defined on the modified input: $\ell(p(\mathbf{x}'), y) = \ell(p(\mathbf{p}, \mathbf{x}), y)$.
- The prompt $\mathbf{p}$ is updated by minimizing the loss function via stochastic gradient descent (SGD):

$$\mathbf{p}_{\text{new}} = \mathbf{p}_{\text{old}} - \nabla_{\mathbf{p}} \ell(p(\mathbf{p}_{\text{old}}, \mathbf{x}), y).$$

Demo: Prompt Tuning in Practice

- Explore a hands-on example in this notebook: `prompt_tuning.ipynb`.

Limitations of Prompt Tuning

- Generally less accurate than full model fine-tuning.
- Learned prompts are often not human-interpretable.

Prompting v.s. finetuning

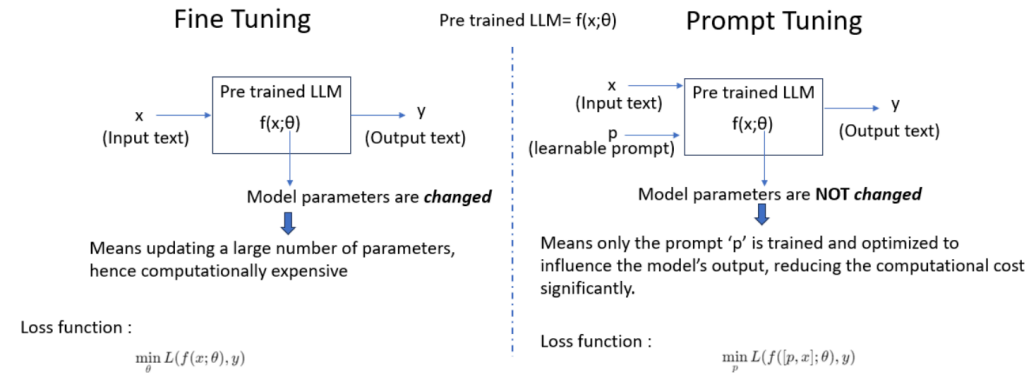


Figure is from https://developer.ibm.com/articles/awb-how-prompt-tuning-works/

Recap: Fine-Tuning

- A foundation model p_{θ_0} is parameterized by θ_0 with pretrained parameters θ_{pre}.
- A new model p_θ is defined with $\theta = (\theta_0, \theta_{\text{new}})$, where θ_{new} can be empty.
- Fully fine-tuning:

$$\widehat{\theta} = \text{Argmin}_\theta \quad L_{\text{finetune}}(p_\theta)$$

  with initialization $\theta_{\text{init}} = (\theta_0 = \theta_{\text{pre}}, \theta_{\text{new,init}})$.
- Fully fine-tuning may be too computationally expensive for large language models (LLMs).
- For instance, for GPT-3, we have $\theta_0 \in \mathbb{R}^d$ where $d = 175\text{B}$.

Recap: Linear Probing

- $p_{\theta_0} = \mathrm{SoftMax}(f_{\theta_0})$, where f_{θ_0} refers to the logits.
- $p_\theta = \mathbf{W} f_{\theta_0} + \mathbf{b}$.
- Only update $\mathbf{W}$ and $\mathbf{b}$ during fine-tuning.

Linear Probing of BERT for Sentiment Analysis

- We have already performed linear probing of BERT for sentiment analysis in Sentiment_finetune_bert.ipynb using the `BertForSequenceClassification` function.
- Now, let's explore what is behind `BertForSequenceClassification` in Linear-probe-BERT.ipynb.

Prefix Tuning (Li & Liang, 2021)

- Prefix tuning is an efficient adaptation method that includes prompt tuning as a special case.
- Recall prompt tuning: $[\mathbf{p}, \mathbf{x}]$, where $\mathbf{p}$ is a prompt updated using backpropagation.
- In prefix tuning, a hidden state $\mathbf{h}_{\theta_{\text{new}}}(\mathbf{p})$ is prepended to the hidden state $\mathbf{h}(\mathbf{x})$ of the input sequence the pretrained model.
- Here

$$\mathbf{h}_{\theta_{\text{new}}}(\mathbf{p})$$

  is usually a (fully connected) neural networks

Prefix Tuning for Decoder-Only Models

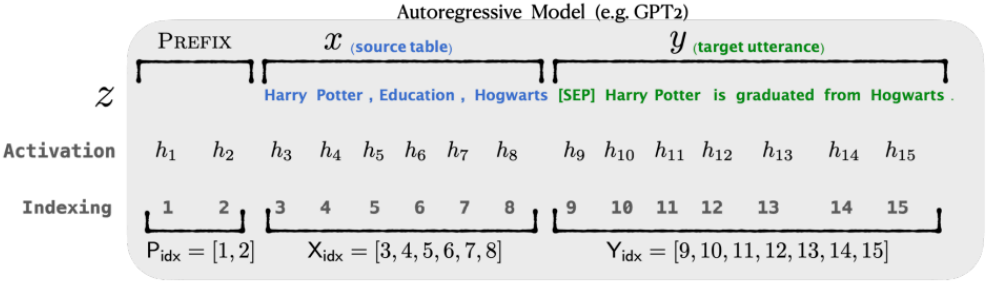


Figure is from Li & Liang, 2021

Prefix Tuning for Encoder-Decoder Models

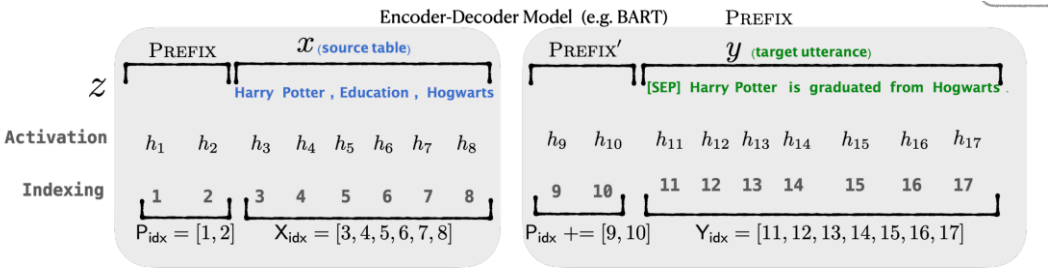


Figure is from Li & Liang, 2021

A Demo of Prefix Tuning

A demo is available in prefix-tuning.ipynb.

Prefix Tuning Meets Attention

- Recall the matrix form of an attention layer:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{SoftMax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}.$$

- Prefix for $\mathbf{K}$: $\mathbf{P}_K$, Prefix for $\mathbf{V}$: $\mathbf{P}_V$.
- Let $\widetilde{\mathbf{K}} = \text{Concat}(\mathbf{P}_K, \mathbf{K})$ and $\widetilde{\mathbf{V}} = \text{Concat}(\mathbf{P}_V, \mathbf{V})$.
- The final output of the attention layer is $\text{Attention}(\mathbf{Q}, \widetilde{\mathbf{K}}, \widetilde{\mathbf{V}})$.

Drawbacks of Prefix Tuning

- Transformers typically have an upper bound on sequence length (why?)
- Prefix tuning reserves part of the sequence length for adaptation, which reduces the available length for downstream task processing.
- As a result, prefix tuning is difficult to optimize, with its performance varying non-monotonically with the number of trainable parameters.

LoRA: low rank adaptation

[Submitted on 17 Jun 2021 (v1), last revised 16 Oct 2021 (this version, v2)]

LoRA: Low-Rank Adaptation of Large Language Models

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example -- deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose Low-Rank Adaptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, no additional inference latency. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at this https URL.

LoRA: low rank adaptation

- For any weight matrix $\mathbf{W} \in \mathbb{R}^{d \times d'}$ in the trainable parameters of a model, let $\mathbf{W}_{\text{pre}}$ be the pretrained parameters.
- Keep $\mathbf{W}_{\text{pre}}$ frozen, and update $\mathbf{W}$ to $\mathbf{W}_{\text{pre}} + \Delta\mathbf{W}$ with $\Delta\mathbf{W} \in \mathbb{R}^{d \times d'}$.
- During finetuning, $\Delta\mathbf{W}$ are trainable parameters (how many trainable parameters?)
- What if $\Delta\mathbf{W}$ has some low rank strucutre?
- Low rank decomposition of $\Delta\mathbf{W}$: $\Delta\mathbf{W} = \mathbf{AB}$ with $\mathbf{A} \in \mathbb{R}^{d \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times d'}$, wher $r \ll \min\{d, d'\}$
- Update $\mathbf{W}$ to $\mathbf{W}_{\text{pre}} + \mathbf{AB}$, only $\mathbf{A}$ and $\mathbf{B}$ are trainable parameters (how many?)

LoRA: low rank adaptation

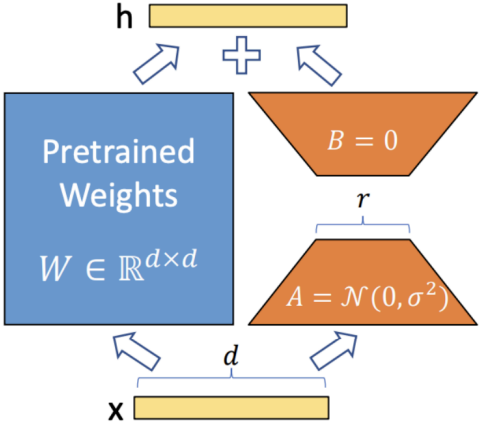


Figure is from https://arxiv.org/pdf/2106.09685

Advantages of LoRA

- As the rank r increases, LoRA converges to full fine-tuning, enabling fine-grained adaptation.
- No additional inference latency: when switching to a new task, recover $\mathbf{W}_{\text{pre}}$ by subtracting $\Delta\mathbf{W}$ and adding a new $\Delta\mathbf{W}'$.

LoRA can be even better than full finetuning

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L ($\text{Adapter}^{\text{L}}$)	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Figure is from `https://arxiv.org/pdf/2106.09685`

Choice of the rank

- A small number of the rank r (such as $r = 1, 2, 4, 8$) can perform well.

	Weight Type	$r=1$	$r=2$	$r=4$	$r=8$	$r=64$
WikiSQL($\pm$0.5%)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm$0.1%)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Fine-tuning with LoRA

- Fine-tuning with LoRA can be easily implemented using the PEFT (Parameter-Efficient Fine-Tuning) library. You can try it out in this lora.ipynb.

Answering the following questions

- Why does an apple fall to the ground?
- Why does a GPS satellite orbit the Earth?
- Does time pass faster on a GPS satellite compared to Earth?

What is RAG?

- No one can know everything —especially specialized knowledge.
- When faced with a question you can't answer, it's often best to "Google it."
- Retrieval-Augmented Generation (RAG) serves as the "Google" for LLMs.

What is RAG?

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Patrick Lewis[†‡], **Ethan Perez**[⋆],

Aleksandra Piktus[†], **Fabio Petroni**[†], **Vladimir Karpukhin**[†], **Naman Goyal**[†], **Heinrich Küttler**[†],

Mike Lewis[†], **Wen-tau Yih**[†], **Tim Rocktäschel**[†‡], **Sebastian Riedel**[†‡], **Douwe Kiela**[†]

[†]Facebook AI Research; [‡]University College London; [⋆]New York University;
plewis@fb.com

What is RAG?

"We definitely would have put more thought into the name had we known our work would become so widespread," Lewis said in an interview from Singapore, where he was sharing his ideas with a regional conference of database developers.

"We always planned to have a nicer sounding name, but when it came time to write the paper, no one had a better idea," said Lewis, who now leads a RAG team at AI startup Cohere.

So, What Is Retrieval-Augmented Generation (RAG)?

Figure is from `https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/`

What is RAG?

- Retrieval-augmented generation (RAG) enhances the accuracy and reliability of generative AI by integrating external, authoritative knowledge sources beyond the model's training data before generating a response.
- RAG extends the capabilities of LLMs to specialized domains or an organization's internal knowledge base.
- It offers a cost-effective way to improve LLM outputs—keeping them relevant, accurate, and context-aware—without requiring model retraining.

Why Do We Need RAG?

Large Language Models (LLMs) tend to respond to every question with high confidence, but they may:

- Provide incorrect answers when they lack the necessary information.
- Offer outdated or generic content when specific or up-to-date information is expected.
- Rely on non-authoritative sources for their responses.
- Misinterpret domain-specific terms due to inconsistencies across training data.

GPT-2 without RAG may generate

- Try using GPT-2 without RAG in this demo: No-RAG.ipynb

```
What is RAG?

Rag is an acronym for RAG-free software. It's an acronym for RAG-free software,

R
```

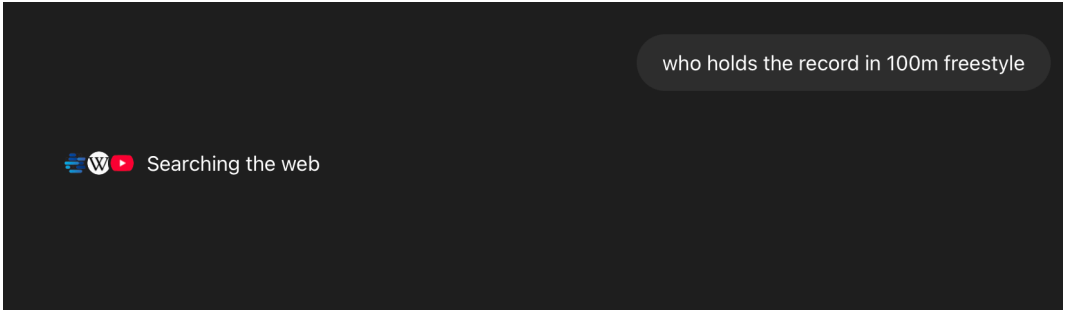
The answer can be clarified using RAG

```
Generated Answer:
Context: RAG enables language models to access external documents. GPT-2 is an autoregressive transformer for text generation.
Question: What is RAG?
Answer: RAG is a language model written in Lisp that generates high-level functions for many kinds of text. Each of these functio
```

RAG build the trust of LLMs

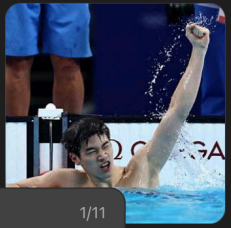
- Retrieval-augmented generation gives models sources they can cite, like footnotes in a research paper, so users can check any claims. That builds trust.
- What's more, the technique can help models clear up ambiguity in a user query. It also reduces the possibility that a model will give a very plausible but incorrect answer, a phenomenon called hallucination.
- RAG allows developers to provide the latest research, statistics, or news to the generative models. They can use RAG to connect the LLM directly to live social media feeds, news sites, or other frequently-updated information sources. The LLM can then provide the latest information to the users.

RAG in GPT-4o



RAG in GPT-4o

As of May 2025, the men's 100-meter freestyle world record is held by Pan Zhanle of China. He set this record on July 31, 2024, during the Paris Olympics, clocking an astonishing time of 46.40 seconds. This performance broke his own previous world record of 46.80 seconds, established earlier that year at the World Championships in Doha . Kyodo News English +10 KTemoc +3



Pan's Olympic swim was not only a person
achievement. His margin of victory—1.08 s
Chalmers—was the largest in the event sin
ance .
Wikipedia +6

← → 1/11

Kyodo News English

Olympics: China's Pan Zhanle breaks own men's 100m freestyle world record

For those interested in witnessing this remarkable feat, here's a video capturing Pan Zhanle's world-record swim: The Times +6

RAG pipeline

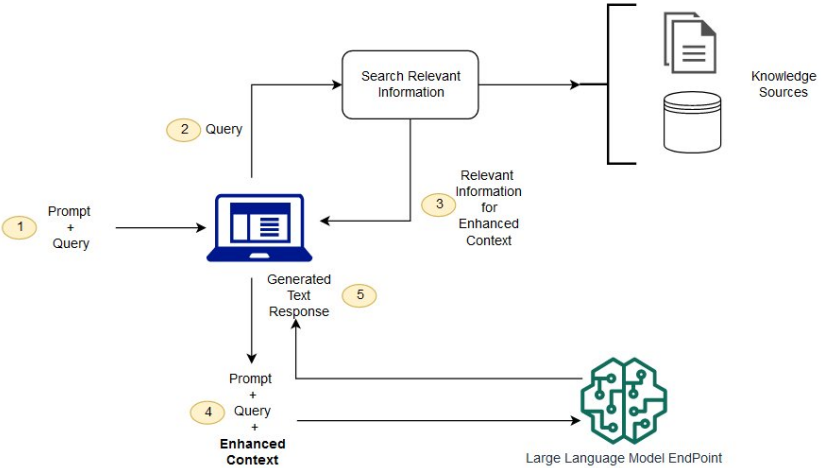


Figure is from https://aws.amazon.com/what-is/retrieval-augmented-generation/

RAG via FAISS

Facebook AI Similarity Search (FAISS) is a library for efficient similarity search and clustering of dense vectors.

```
!pip3 install faiss-cpu
```

显示隐藏的输出项

```
[ ] import os
    os._exit(00)  # Force restart
```

```
from transformers import RagTokenizer, RagRetriever, RagTokenForGeneration

tokenizer = RagTokenizer.from_pretrained("facebook/rag-token-nq")
retriever = RagRetriever.from_pretrained("facebook/rag-token-nq", index_name="exact", use_dummy_dataset=False)
model = RagTokenForGeneration.from_pretrained("facebook/rag-token-nq", retriever=retriever)

input_dict = tokenizer.prepare_seq2seq_batch("who holds the record in 100m freestyle", return_tensors="pt")

generated = model.generate(input_ids=input_dict["input_ids"])
print(tokenizer.batch_decode(generated, skip_special_tokens=True)[0])
```

RAG with custom external documents

- You can use a custom external documents, as exampled in RAG.ipynb

Probability Models of RAG

- Let $\mathbf{x}$ represent the input sequence (prompt), and $\mathbf{y}$ represent the sequence to be generated.
- The objective is to find a language model that maximizes the probability $p(\mathbf{y}|\mathbf{x})$.
- In RAG, each retrieved document $\mathbf{z}$ is treated as a latent variable.
- The probability model is expressed as:

$$p_{\mathrm{RAG}}(\mathbf{y}|\mathbf{x}) = \int_{\mathbf{z}} p_\eta(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{y}|\mathbf{z},\mathbf{x}) \approx \sum_i p_\eta(\mathbf{z}_i|\mathbf{x}) p_\theta(\mathbf{y}|\mathbf{z}_i,\mathbf{x}).$$

Probability Models of RAG

- **Retriever:** $p_\eta(\mathbf{z}_i|\mathbf{x}) \propto \exp(\mathrm{emb}(\mathbf{z}_i)^T \mathrm{emb}(\mathbf{x}))$, where $\mathrm{emb}(\cdot)$ represents the sentence embedding of a sequence, typically computed by an LLM such as BERT.
- Select K documents from a large database $\{\mathbf{z}_i\}_i$ that maximize $p_\eta(\mathbf{z}_i|\mathbf{x})$.
- **Generator** p_θ**:** Typically an encoder-decoder model, such as BART.

Overview of RAG

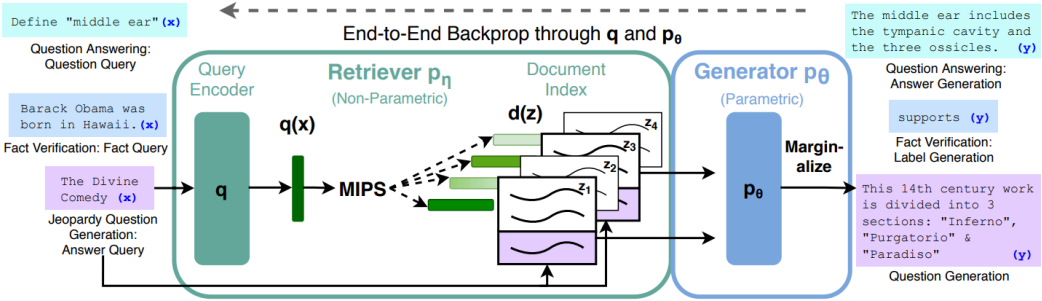


Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

Figure is from `https://arxiv.org/pdf/2005.11401`

What's next

- Benchmarking and Model Evaluation
- Alignment, RLHF,...