Efficiency & Advanced Techniques

Quantization, MoE, linear attention, LoRA, RAG, reasoning

Zdeněk Kasner, Michal Novák

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Today's learning outcomes

After today's class, you should be able to:

- **Describe the main bottlenecks for deploying and scaling up** the Transformer architecture.
- **Describe the ideas behind efficient algorithms** such as parameter quantization, (Q)LoRA, linear attention, and mixture of experts.
- **Implement a simple retrieval-augmented generation** pipeline.
- **Use LLM reasoning** to achieve better outputs for complex problems.

Efficiency

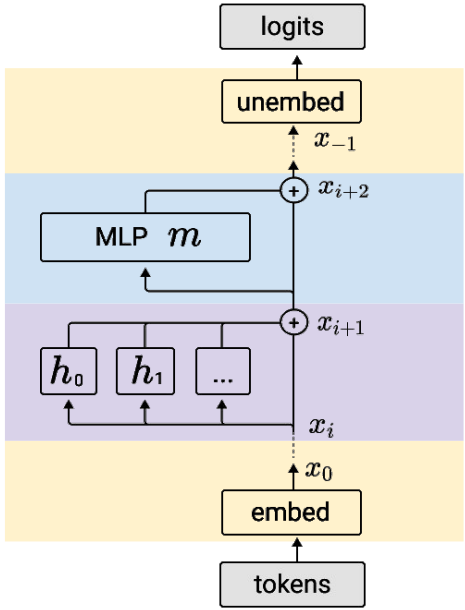
Transformer: algorithmic complexity

image: https://transformer-circuits.pub/2021/framework/index.html

Variables: hidden state dimension (D), sequence length (N), vocabulary (V).

	Time	Space
final projection (unembedding)	$N \cdot V \cdot D$ each state projected to logits over the vocabulary	$V \cdot D + V \cdot N$ (un)embedding matrix 📦 output logits 🧮
MLP	$N \cdot D^2$ projection matrix 📦	$D^2 + N \cdot D$ activations 🧮
multi-head attention	$N \cdot D^2 + N^2 \cdot D$ queries, keys, values attention matrix 📦	$N^2 + N \cdot D + D^2$ projected values 🧮 Q, K, V matrices 📦
embedding	$N \cdot D$ each token is embedded	$V \cdot D + N \cdot D$ embedding matrix 📦 embeddings 🧮

📦 = model parameters, 🧮 = computed values



LLM bottlenecks: Computational throughput

Time complexity: $N \cdot D^2 + \mathbf{N^2 \cdot D} + N \cdot V \cdot D$

long context $N^2 \gg D^2$

happens only once

Computing **multi-head attention** is the main limitation when scaling to longer input contexts.

Solution: Efficient attention, mixture of experts

par.	*min*	*max*
N	512	128,000+
D	768	4,096+
V	30,000	100,000+
blocks	8	128+

Typical hyperparameter values

LLM bottlenecks: Disk space

stored only once

Space complexity: $V \cdot D + D^2$

(model parameters 📦)

MLPs typically take up majority of model parameters as they are repeated in each layer, unlike the embedding matrix.

Solution: Parameter quantization, (Q)LoRA

par.	*min*	*max*
N	512	128,000+
D	768	4,096+
V	30,000	100,000+
blocks	8	128+

Typical hyperparameter values

LLM bottlenecks: Memory

gets more important with growing input context size

Space complexity: $N^2 + N \cdot D + N \cdot V$

(activations 🧮)

For loading a model to GPU memory, we need to account for both model parameters (→fixed) and for computed activations (→depend on the input context size).

Solution: Parameter quantization, (Q)LoRA

par.	*min*	*max*
N	512	128,000+
D	768	4,096+
V	30,000	100,000+
blocks	8	128+

Typical hyperparameter values

LLM bottlenecks: Training vs. inference

In terms of resource requirements, **pre-training ≫ finetuning ≫ inference.**

Dataset size:

- Finetuning: up to 100s GB.
- Pre-training: 10T tokens → **50TB.**

Memory requirements:

- Inference: 2 bytes per parameter → 14 GB for a 7B model.
- Training: 2 bytes per parameter + 2 bytes per gradient + 12 bytes per optimizer weight (Adam) → **112 GB for a 7B model.**

(Note that we typically want to run inference on consumer hardware.)

Efficient algorithms: overview

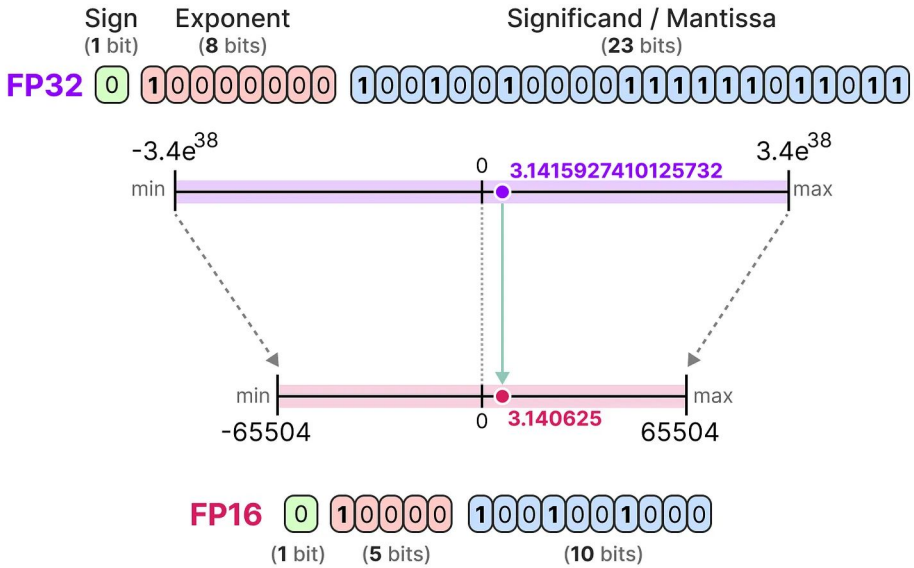
Algorithm	What it targets	Training memory	Training speed	Inference memory	Inference speed
Parameter quantization	Reducing parameter size	–	–	✓ ✓	✓
Mixture of Experts (MoE)	Reducing active parameter count	–	✓ ✓	–	✓
Linear attention	Faster attention using linear algebra	✓	✓	✓	✓
FlashAttention	Faster attention using HW optimizations	✓ ✓	✓	✓	✓ ✓
(Q)LoRA	Efficient model fine-tuning	✓	✓	–	–

Parameter quantization

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Model parameters are floating point numbers. How do we store them? And can we store them more efficiently?

- **float64** – native Python (typically not used on GPUs)
- **float32** – baseline precision
- **float16** – half precision



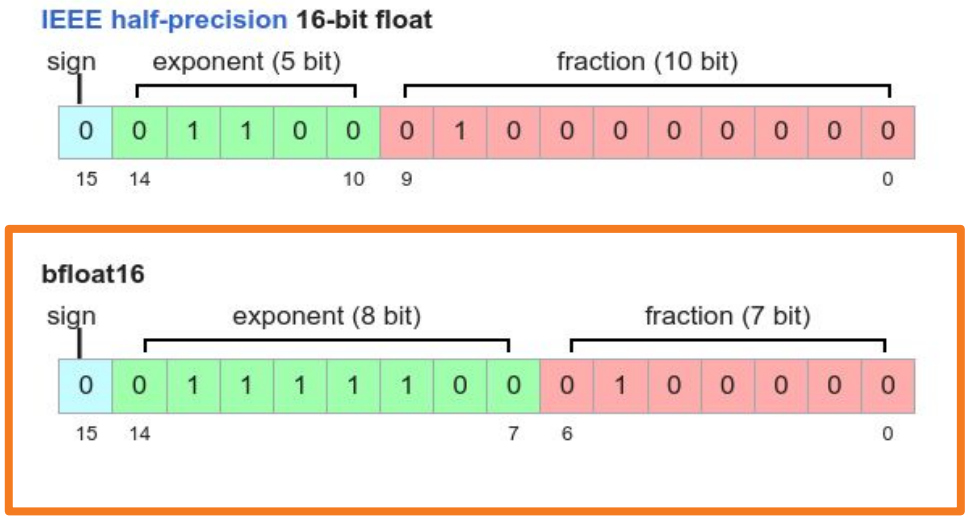
- For inference, float16 typically causes only little performance degradation.
- For training, float16 can lead to vanishing gradients → mixed-precision training.

Parameter quantization

https://en.wikipedia.org/wiki/Bfloat16_floating-point_format

bfloat16

- Developed by Google Brain (→“brain float”).
- 16-bit float optimized for ML models.
- Greater dynamic range than float16 (→supports outlier weights) at the cost of lower precision.

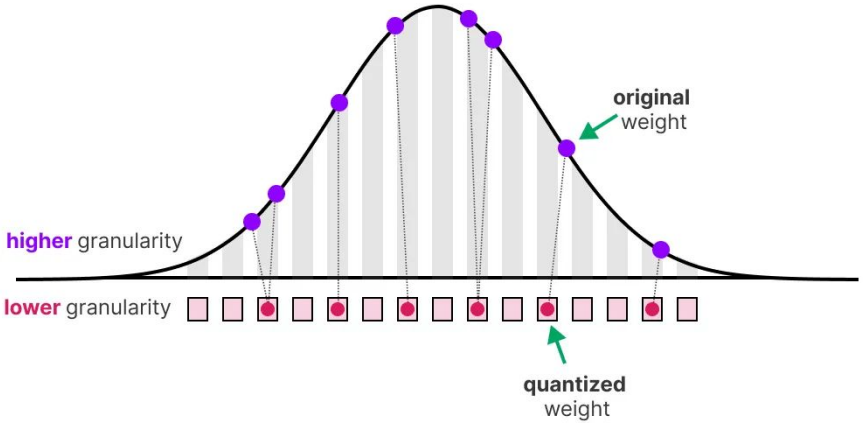


Parameter quantization

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Can we go further?

👷 **Bob:** Let's use **int8**! It is an 8-bit numerical format. We'll save twice as much space compared to float16.



Parameter quantization

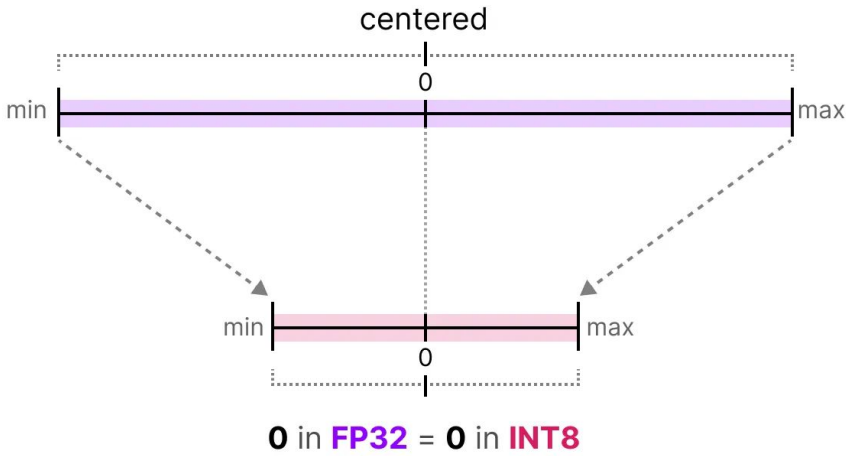
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

👩‍🔬 **Alice:** But we cannot just round the number, can we? Floats have a huge dynamic range. How do we squeeze them into (-127, 127)?

👷 **Bob:** We don't. We only use the range of parameters that we have in our model weights.

👩‍🔬 **Alice:** The highest weight in my model can be still larger than 127.

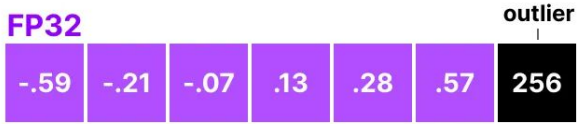
👷 **Bob:** But it will not be that large. Now we can squeeze them!



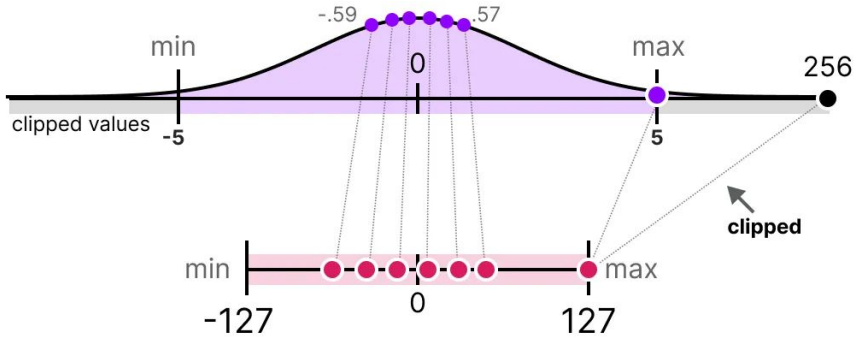
Parameter quantization

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

👩‍🔬 **Alice:** I am not convinced. What if some of my weights are outliers? Wouldn't this map most of the weights to zero (or near zero)?



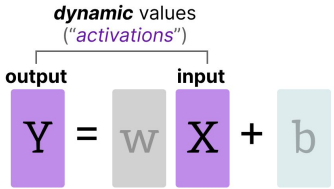
👷 **Bob**: Ahem, right... Let's just clip the outliers?



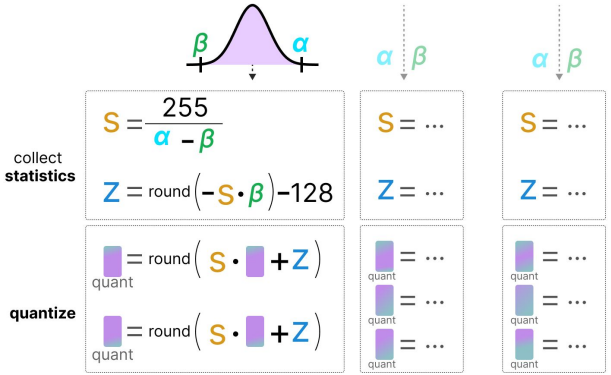
Parameter quantization

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

👩‍🔬 **Alice:** Ok, and how will you pre-compute the range for activations? These are dynamic.



👷 **Bob:** Dynamically?



Parameter quantization

https://arxiv.org/abs/2310.11453

👩‍🔬 **Alice:** I give up. Just go on and quantize your model to 1-bit if it works for you.

👱‍♀️ **Bob:** Can I? Awesome!

BitNet: Scaling 1-bit Transformers for Large Language Models

Hongyu Wang*†‡ Shuming Ma*† Li Dong† Shaohan Huang†
Huaijie Wang§ Lingxiao Ma† Fan Yang† Ruiping Wang‡ Yi Wu§ Furu Wei†◇
† Microsoft Research ‡ University of Chinese Academy of Sciences § Tsinghua University
https://aka.ms/GeneralAI

Parameter quantization

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Quantization in practice:

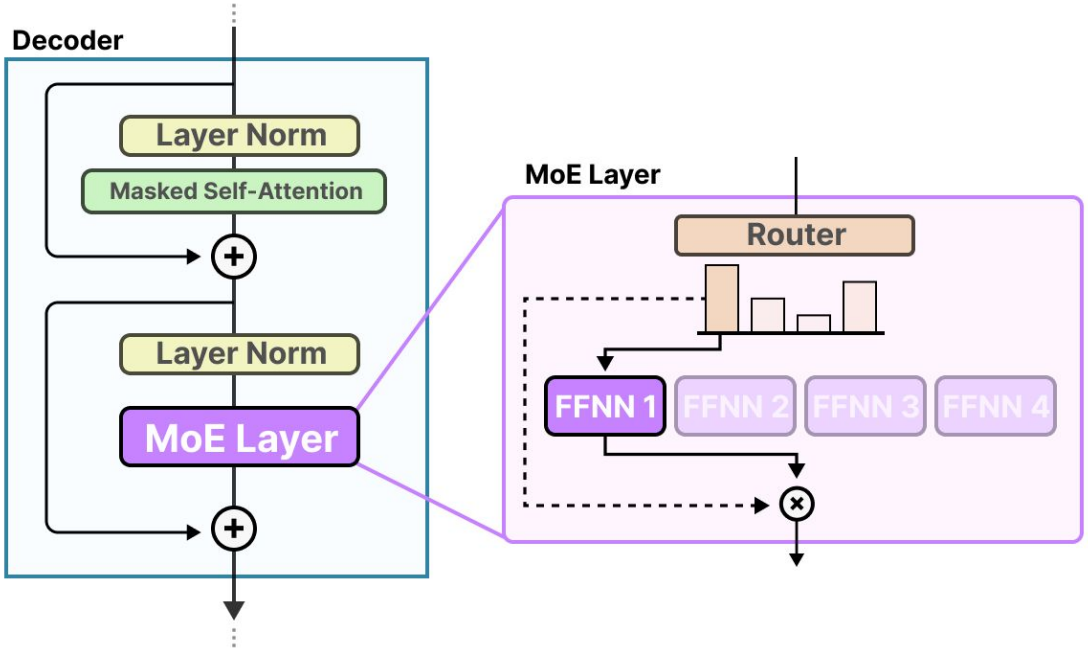
- **GPTQ**
  - Quantization method that uses a few extra tricks to go robustly beyond 8-bit quantization (6-bit, 4-bit, 2-bit).
- **GGUF**
  - File format that performs block-wise quantization → enables offloading parts of the model to CPU.

Models in Ollama are quantized by default to 4-bit.

Mixture of experts (MoE)

A Visual Guide to Mixture of Experts (MoE)

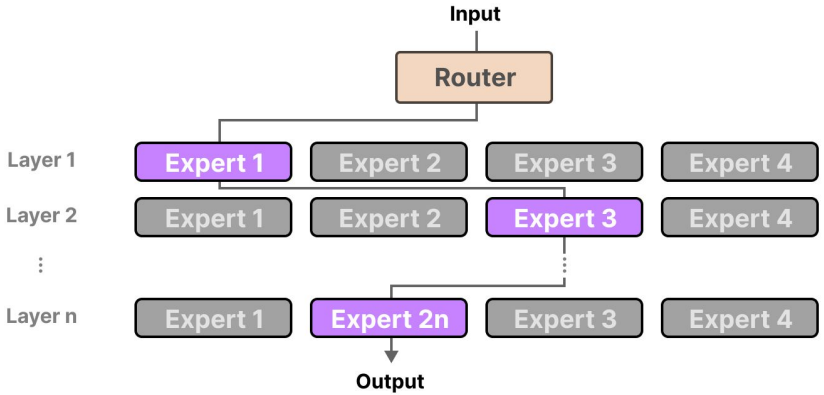
“Experts” = multiple feed-forward networks in each MLP layer of the Transformer.



Mixture of experts (MoE)

https://arxiv.org/pdf/2401.04088
A Visual Guide to Mixture of Experts (MoE)

We can choose a different expert at each **layer**...



...and a different set of experts for each **token**.

```
class MoeLayer(nn.Module):
    def __init__(self, experts: List[nn.Module],
        super().__init__()
        assert len(experts) > 0
        self.experts = nn.ModuleList(experts)
        self.gate = gate
        self.args = moe_args
```

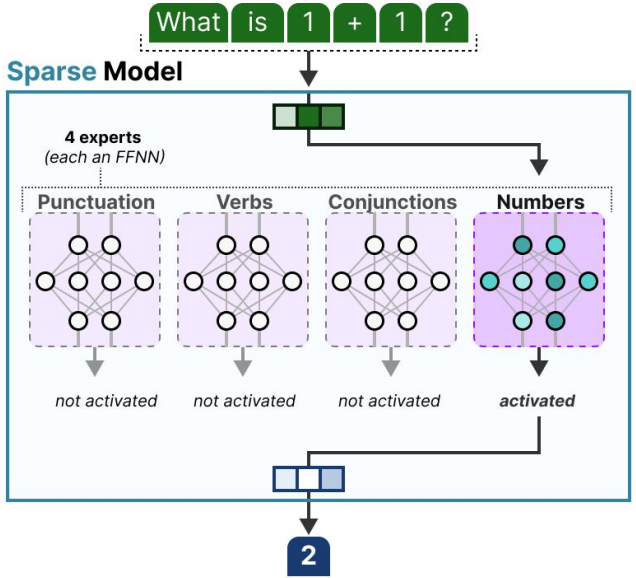
Mixture of experts (MoE)

A Visual Guide to Mixture of Experts (MoE)

Why it's a good idea?

- **Individual experts** can specialize to solving certain kinds of problems.
- **Faster training** for the same number of total parameters (→ we backpropagate only through the selected experts).
- **Faster inference** (although we still need to load the full model into memory).

8x7B → 8 standalone models, approx. equivalent to a 47B dense model (Not 56B, as other parts of the model than MLPs are shared.)



Linear attention

https://haileyschoelkopf.github.io/blog/2024/linear-attn/

Let's revisit the O(N^2) attention. Can we do better?

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(QK^T\right)V$$

Let's zoom in! 🕵️

$$\text{Softmax}\left(QK^T\right)V = \frac{\exp(QK^T)}{\sum_{i=1}^{L} \exp(QK_i^T)}V$$

very expensive way to measure similarity between queries and keys

Linear attention

https://haileyschoelkopf.github.io/blog/2024/linear-attn/

We can use a cheaper way of computing similarity:

$$\frac{\text{sim}(Q,K)}{\sum_{i=1}^{L}\text{sim}(Q,K_i)}V \qquad \text{sim}(Q,K) = \phi(Q)\cdot\phi(K) = \phi(Q)\phi(K)^T$$

Φ = optional dimensionality mapping, can be also identity

This also allows us to re-arrange matrix multiplications:

$Q \cdot K \in R^{NxN}$

$$\frac{\phi(Q)\phi(K)^T}{\sum_{i=1}^{L}\phi(Q)\phi(K_i)^T}V = \frac{\phi(Q)(\phi(K)^T V)}{\phi(Q)\sum_{i=1}^{L}\phi(K_i)^T}$$

$K \cdot V \in R^{DxD}$

Linear attention

https://www.reddit.com/r/MachineLearning/comments/10eolfl/d_why_arent_we_all_using_linear_transformers/

The resulting operations are $O(N \cdot D^2)$, which is linear in sequence length.

Wo-hoo! 🎉

r/MachineLearning • 2 yr. ago
currentscurrents Top 1% Poster

[D] Why aren't we all using linear transformers?

There's a bunch of them - Linformer, Longformer, Performer, Nystromformer, Big Bird, etc etc. Plus a bunch more that have similar goals but don't necessarily aim for linear complexity, like memory-augmented transformers.

As far as I know, none of them have really seen much use. Even for image problems, which have very long input sizes, people are using regular transformers with tokenization schemes.

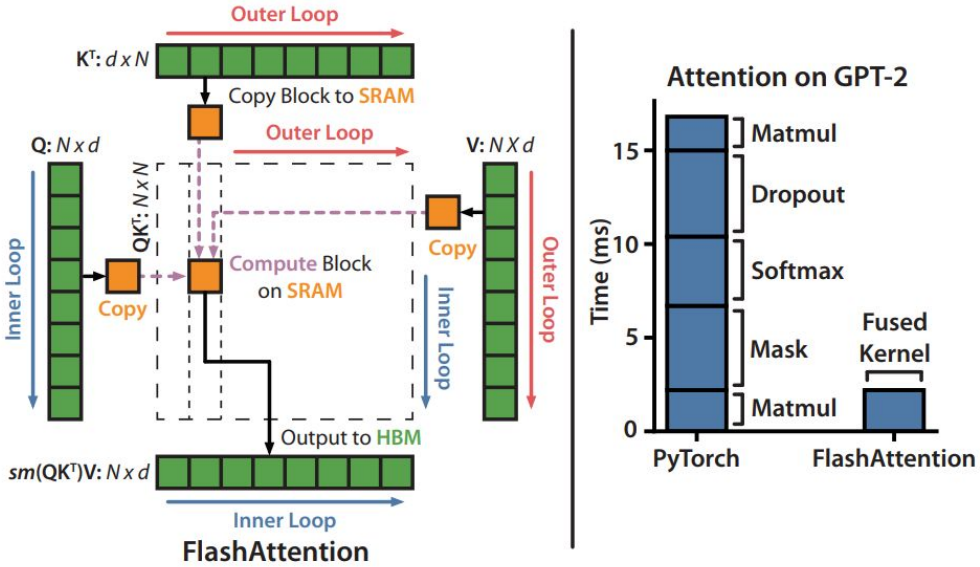
- Am I wrong? Are they actually good, or are at least some of them better than regular transformers?
- If not, what's wrong with them? Do they have lower accuracy? Are they slower to train?

FlashAttention

https://arxiv.org/abs/2205.14135

Approximations such as linear attention can lead to degraded performance.

Meanwhile, **hardware optimizations** of the full attention mechanism can go a long way.



FlashAttention

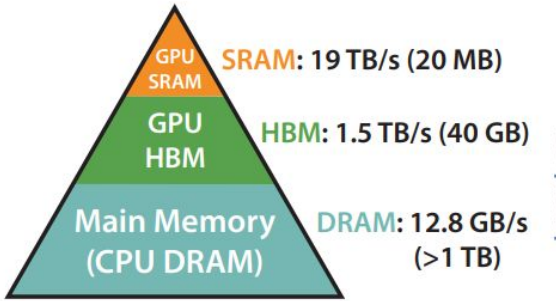
https://arxiv.org/abs/2205.14135

FlashAttention re-arranges the operations to use the underlying hardware more efficiently.

It is:

- **Fast**: 2-3x faster than baselines.
- **Memory-efficient**: linear in sequence length.
- **Exact**: uses no approximations.

It is now commonly implemented in LLM frameworks.



Memory Hierarchy with Bandwidth & Memory Size

Low-rank adaptation (LoRA)

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

👩‍🎨 **Alice:** Remember the 112 GB of GPU memory I would need to finetune a 7B model? Where should I get that?

👷 **Bob**: Buy a new GPU cluster!

👩‍🎨 **Alice:** ...

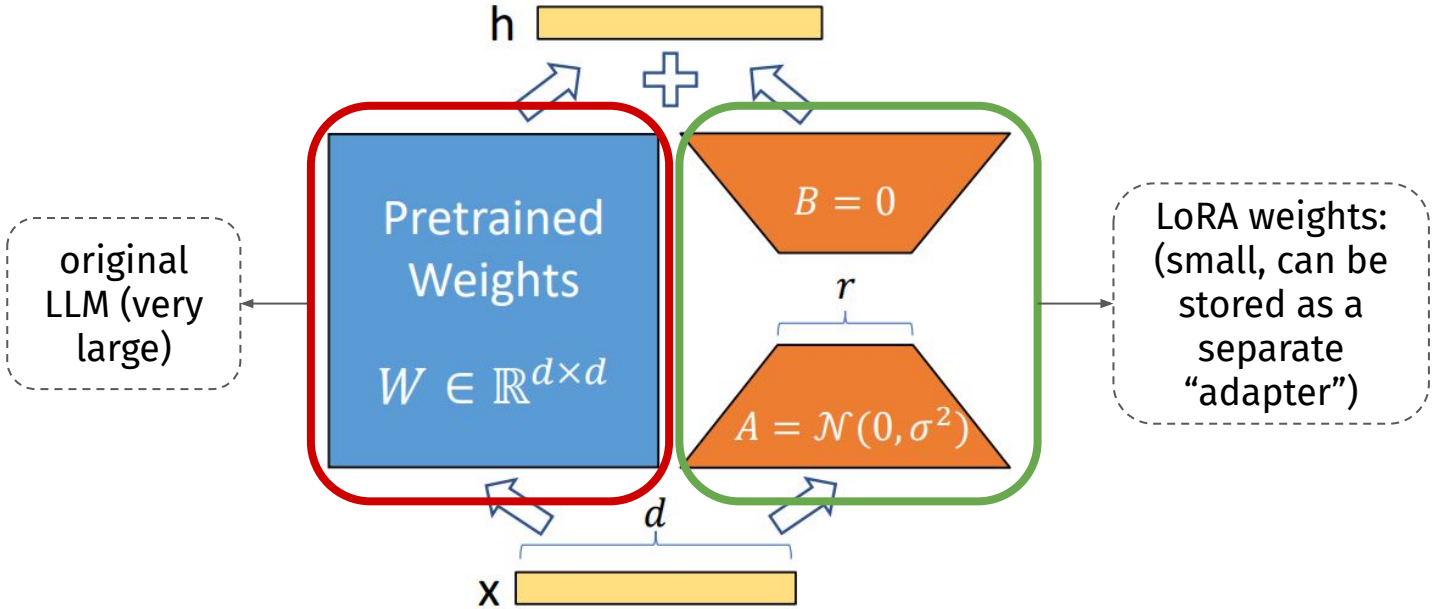
👷 **Bob**: Or use efficient finetuning methods?

Key ideas of LoRA:

- Instead of updating model weights directly, we can keep the Δ (the “diff”) in a separate matrix.
- The Δ matrix has a low rank → it can be approximated by two smaller matrices.

Low-rank adaptation (LoRA)

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



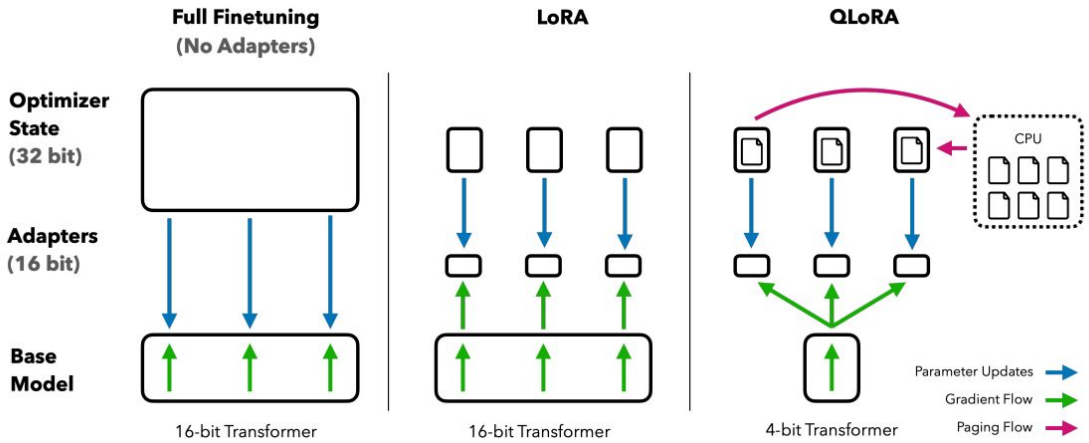
For inference, LoRA weights are applied to the model → no latency issues.

QLoRA: finetuning quantized models

https://arxiv.org/pdf/2305.14314

Can we apply LoRA to quantized models for even higher efficiency?

Yes: **QLoRA** does that with (1) careful parameter quantization and (2) optimizing CPU-GPU memory transfers.



LoRA and QLoRA in practice

All things being equal, full finetuning > LoRA > QLoRA in terms of model performance.

→ The goal is to make the most out of the memory you have at your disposal.

Starting points:

- https://huggingface.co/blog/4bit-transformers-bitsandbytes
- https://www.kaggle.com/code/neerajmohan/finetuning-large-language-models-using-qlora

Method	Bits	~ memory required for finetuning a 7B model
Full finetuning	32	120GB
Full finetuning	16	60GB
LoRA	16	16GB
QLoRA	8	10GB
QLoRA	4	6GB
QLoRA	2	4GB

https://github.com/hiyouga/LLaMA-Factory?tab=readme-ov-file#hardware-requirement

Retrieval-Augmented Generation (RAG)

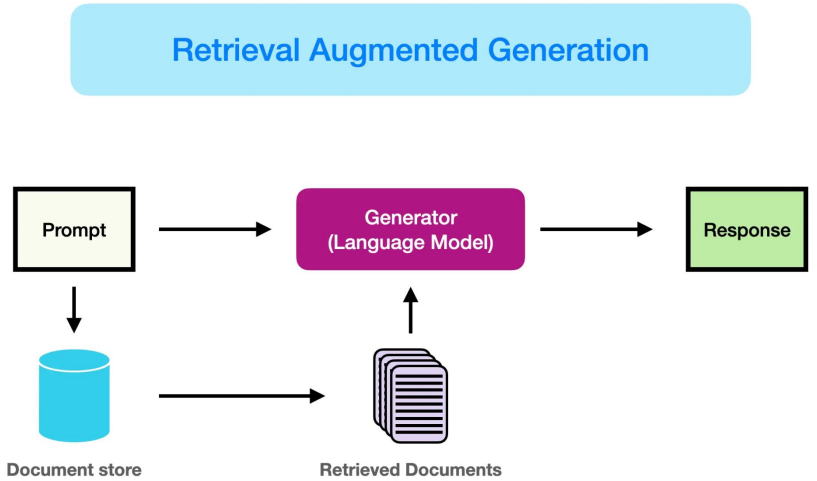
Retrieval-Augmented Generation (RAG)

https://www.promptingguide.ai/research/rag

The knowledge saved in model parameters is lossy and incomplete, and the models tend to hallucinate when they don't know the answer. What can we do about it?

Retrieval-augmented generation (RAG):

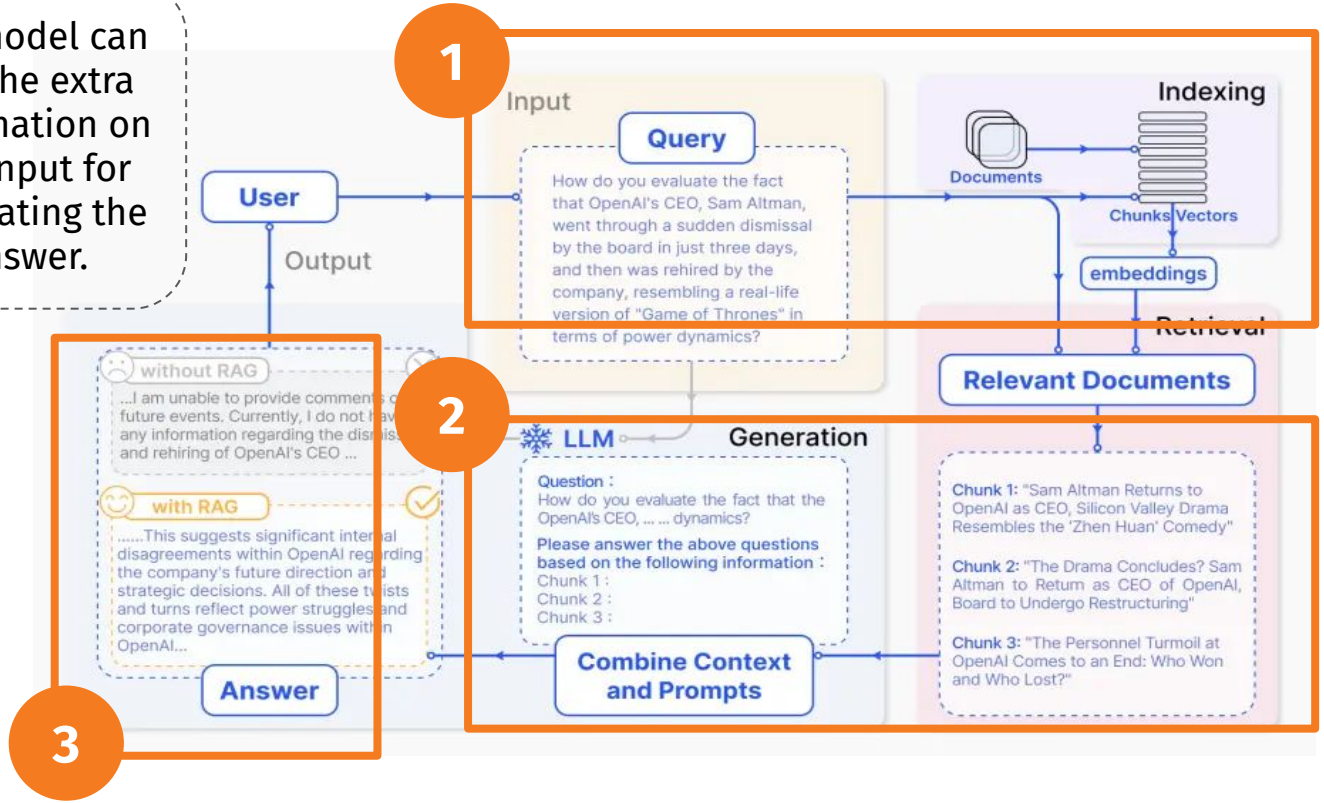
- Retrieve additional documents related to user prompt (websites/ files / database entries / code / ...).
- Concatenate the most relevant documents with the prompt.
- Let the LLM generate the answer.



Retrieval-Augmented Generation (RAG)

https://www.promptingguide.ai/research/rag

The model can use the extra information on the input for generating the answer.



Semantic search in the knowledge base based on the prompt

The most relevant documents appended to the prompt

Retrieval-Augmented Generation (RAG)

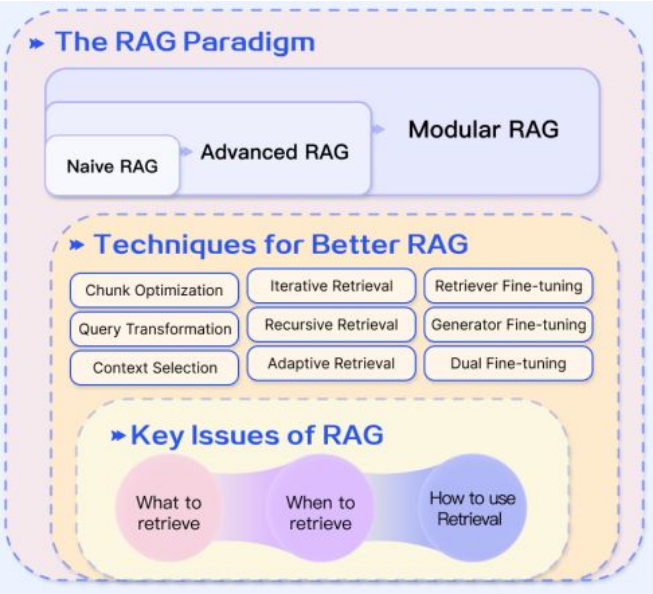
https://arxiv.org/abs/2312.10997

Pros:

- Allows searching private knowledge bases.
- Can reduce the amount of hallucinated facts.
- Makes use of long input context windows.

Cons:

- The model tries to account for all the documents, even the irrelevant ones → the answer may be worse than without RAG.
- Can give a false sense of “groundedness”.

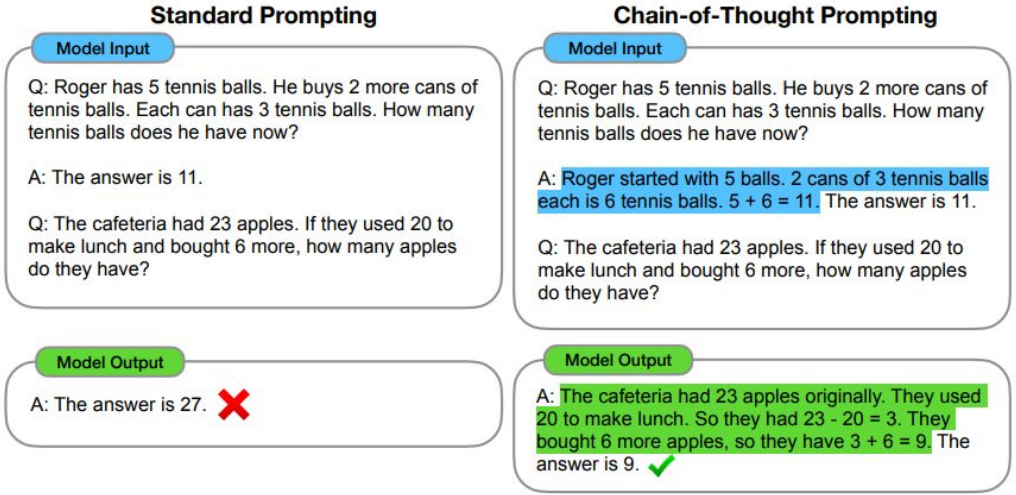


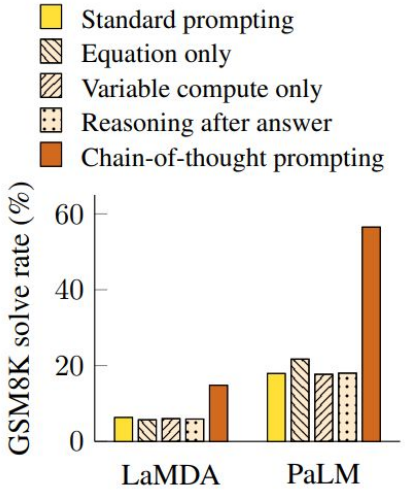
Reasoning

Chain-of-thought Prompting

https://arxiv.org/pdf/2201.11903

Wei et al., (2022): *"The models struggle with math problems. What if we showed them how to do intermediate reasoning steps?"*

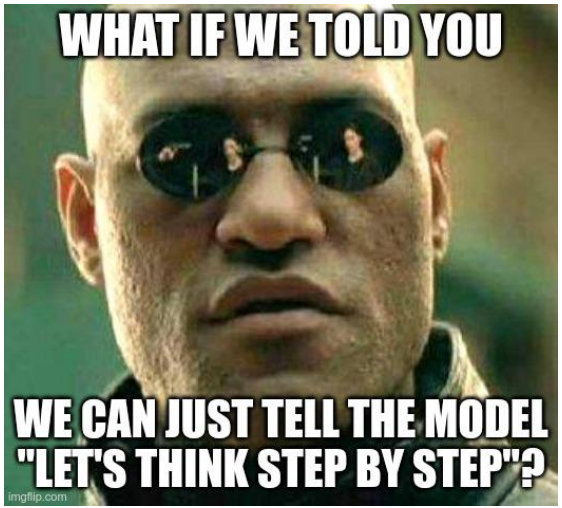




Chain-of-thought Prompting

https://arxiv.org/pdf/2205.11916

Kojima et al (2022): *"Finding good examples of intermediate reasoning steps can be tricky. Do we need to do that at all?"*



(c) Zero-shot

Q: 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?
A: The answer (arabic numerals) is

(Output) 8 ✗

(d) Zero-shot-CoT (Ours)

Q: 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?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Chain-of-thought Prompting

Chain-of-thought prompting:

- ...is nowadays a standard prompting method.
- ...generally increases performance on problems requiring multi-step reasoning.
- ...does not require more than appending (a variant of) *"Think step-by-step"* to the prompt.

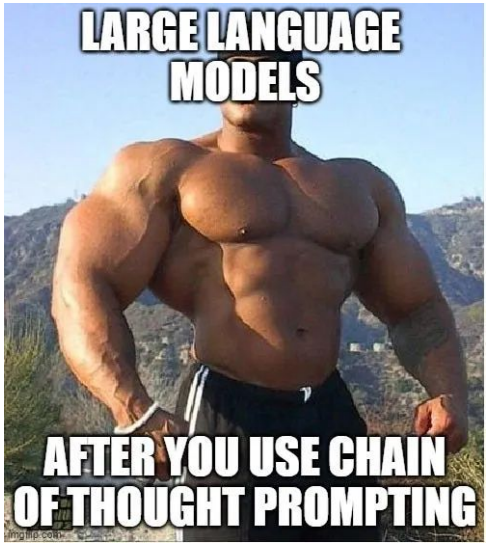


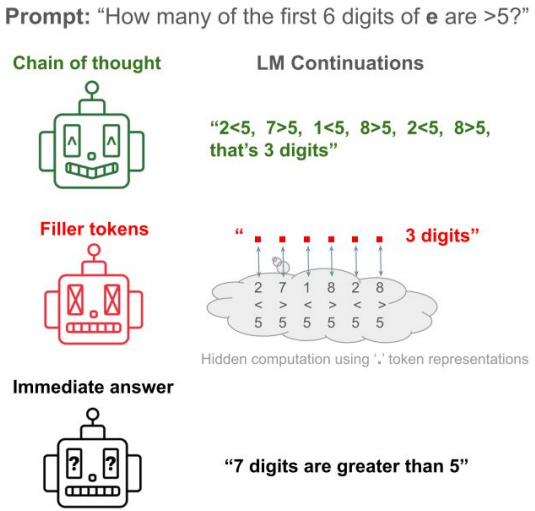
image source

Chain-of-thought Prompting

https://arxiv.org/abs/2309.15402

Why does it work?

- The “emergent” ability can be learned from **pretraining on code data** (Ma et al., 2024; Puerto et al., 2024).
- The model may be just **using extended inference time** to perform more computation (Pfau et al., 2024).



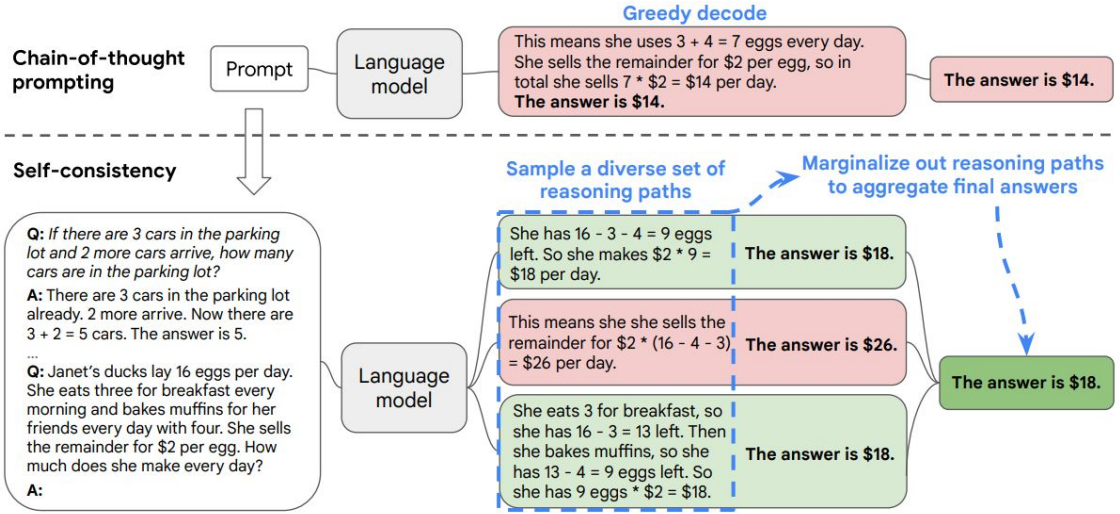
Test-time scaling

https://arxiv.org/pdf/2203.11171

CoT prompting has shown a new path: test-time scaling.

The simplest example of test-time scaling:

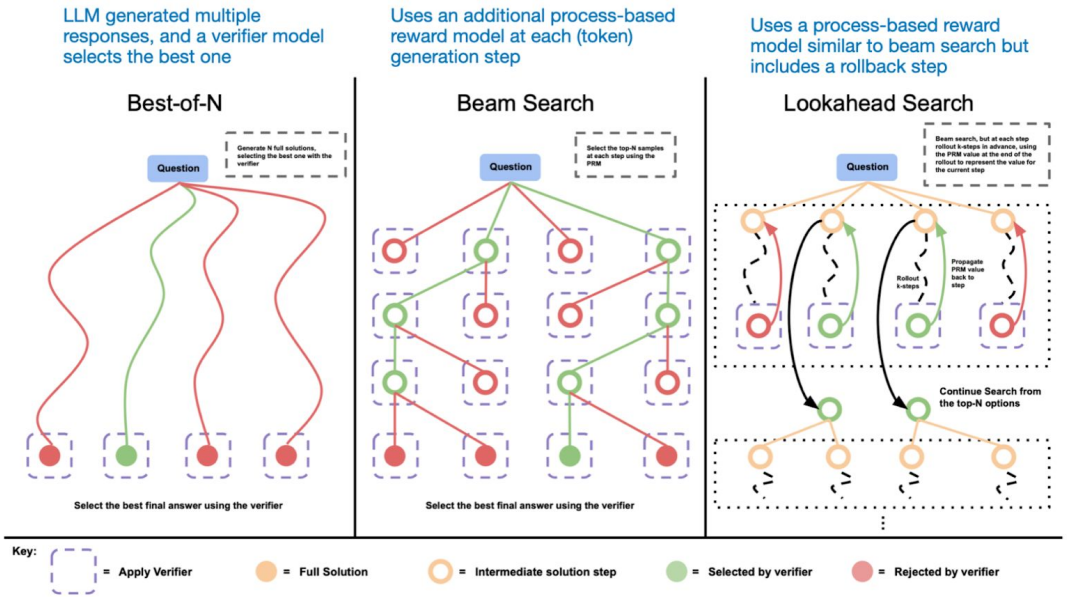
1. Generate multiple CoT paths for the given problem.
2. Use **majority voting** to select the final answer.



Test-time scaling

https://arxiv.org/pdf/2408.03314

More advanced extensions involve tree-like search techniques and a verifying the solution with a verifier model (e.g., code interpreter).



Large Reasoning Models (LRMs)

Goal: instilling the reasoning process into the behavior of the LLM.

→ The model is trained on reasoning traces.

The traces include problem decomposition, intermediate steps, and back-tracking.

Where to get the reasoning traces?

Examples of LRMs:

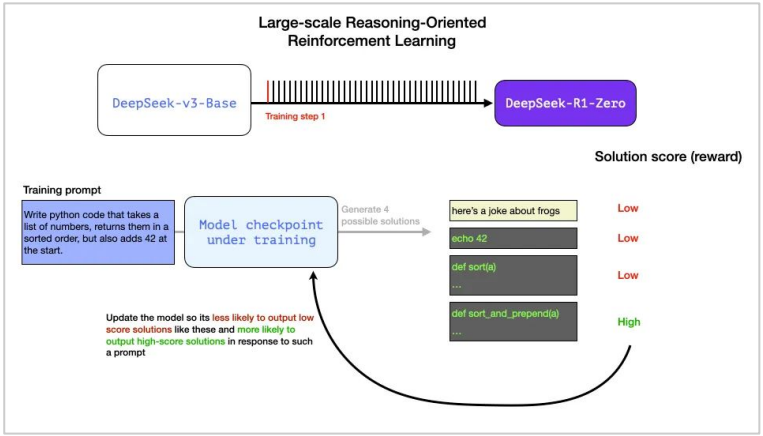
- OpenA1 o1, o3, o4
- DeepSeek-R1
- Gemini 2.5
- Claude 3.7 Thinking

Training techniques for LRMs

https://arxiv.org/abs/2501.12948

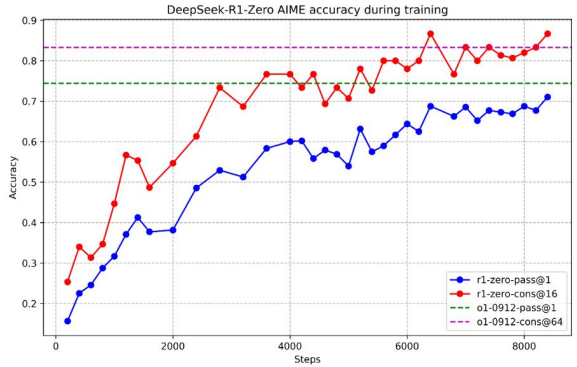
Option #1: Pure reasoning-oriented RL

1. Get a base LLM (no instruction tuning).
2. Run the model on datasets for code generation and math problem solving.
3. Perform RL training with rewards for accuracy (using a verifier) and format (rule-based).



- Natural emergence of test-time computation.
- Early unstable cold-start phase.
- Missing general capabilities.

Use to make DeepSeek-R1-Zero.

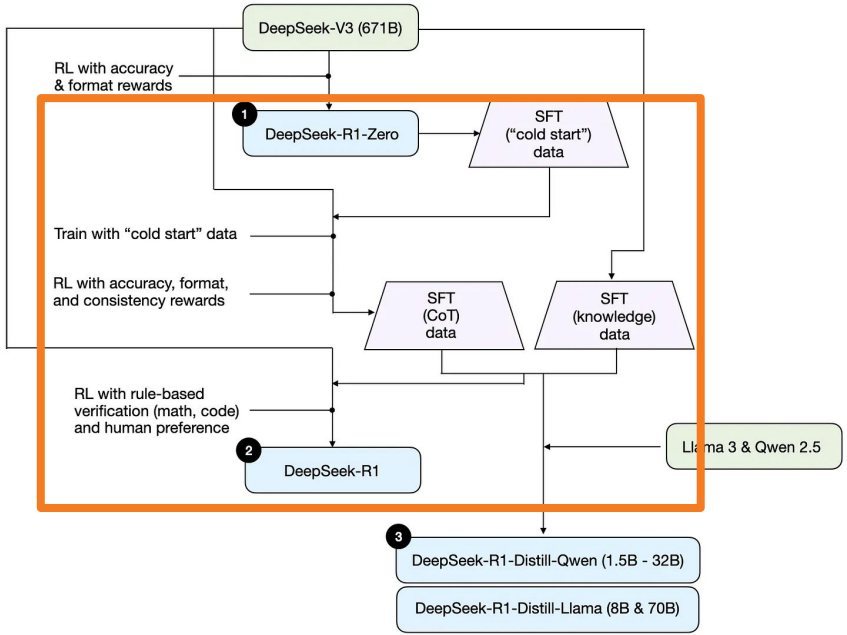


Training techniques for LRMs

from Sebastian Rashka's "Understanding Reasoning LLMs"

Option #2. Supervised fine-tuning (SFT), followed by RL.

1. Get a pure RL-tuned LRM, generate outputs A and post-process them by human annotators.
2. Fine-tune an instruction-tuned LLM on A → generate outputs B.
   - addressing unstable cold start.
3. Fine-tune a base LLM on a mixture of B and non-reasoning training data.
   - addressing missing general capabilities
4. Continue training with reasoning-oriented as well as human preference RL.



The process was used to make DeepSeek-R1 (671B parameters); likely also used in OpenAI-o1.

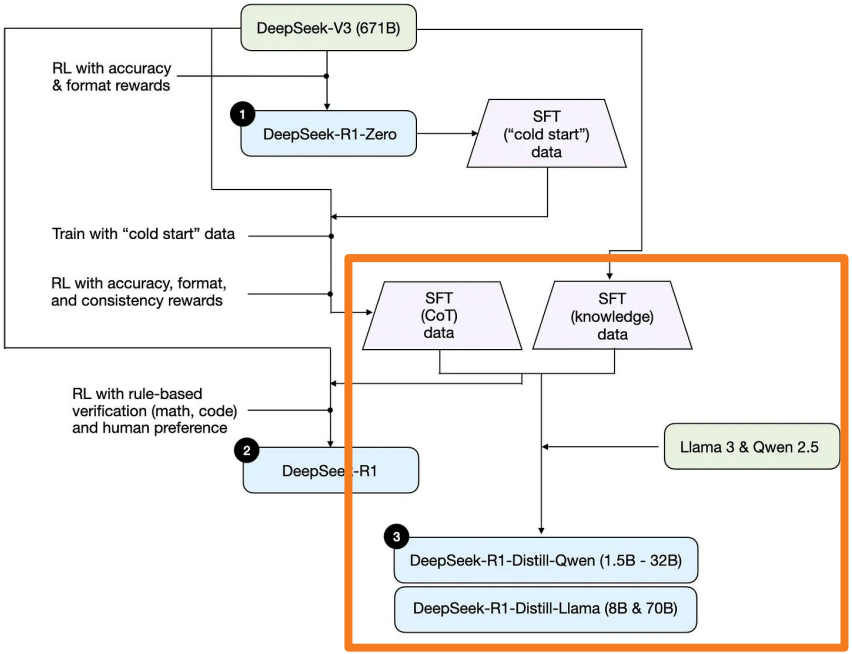
Training techniques for LRMs

from Sebastian Rashka's "Understanding Reasoning LLMs"

Option #3: Pure supervised fine-tuning.

- Fine-tuning a small base LLM (e.g., Llama or Qwen models) on the combination of reasoning and non-reasoning outputs.
- Cheaper to run but still expensive to train (800k SFT samples)
- NovaSky's Sky-T1 used only 17k SFT samples ($450) to perform on par with o1

Used to make smaller ("distilled") DeepSeek models.



Efficiency & Advanced Techniques

- LLMs can be implemented more efficiently:
  - **Lower memory footprint**: parameter quantization, (Q)LoRA
  - **Faster inference**: MoE, linear attention, FlashAttention
- Post-training performance of LLMs can improved at test time:
  - RAG for **extending the model knowledge**
  - CoT and reasoning for **tackling complex problems**

https://ufal.cz/courses/npfl140

Additional links

- Mixture of Experts Explained - HuggingFace blogpost
- Understanding Reasoning LLMs - blogpost.
- awesome-o1 - list of resources related to OpenAI O1.
- Scaling Test-time Compute - HuggingFace blogpost.