UNIVERSITÉ CÔTE D'AZUR

Lecture 8: Large models (LLMs, VLMs, Generators)

Advanced deep learning

Rémy Sun
remy.sun@inria.fr

Inria

cnrs

i3s sophia antipolis

Course organization

Advanced Deep Learning

UNIVERSITÉ CÔTE D'AZUR

- Goal: In-depth understanding of important Deep Learning staples
  - Reinforce what you have already seen
  - Introduce state of the art models

- This is a hands-on course in pytorch
  - Minimal math
    - Enough to understand
  - Quite a bit of coding
  - Get comfortable with the standard pipeline

Course organization: 10 Lectures

UNIVERSITÉ CÔTE D'AZUR

- L1-2: Overview of Deep Learning (F. Precioso)
- L3-4: Fundamentals of Deep Learning (R. Sun)
- L5-6: Transformers (R. Sun)

- ***L8: Large models (LLMs, VLMs, Generators) (R. Sun)***

- L7: Tricks of the trade (R. Sun)

- L9: Ethics of AI (F. Precioso)

- L10: Intro to generative models (P-A. Mattei)

Today!

UNIVERSITÉ CÔTE D'AZUR

- Talk about big models
  - Refresher on Transformers
  - General overview
  - Zoom in on how LLMs are built
  - Prompt engineering
  - Reasoning in the age of LLMs

- Lab session
  - Load up a trained LLM
  - Play a bit with it

Before we start

How does pytorch work?

UNIVERSITÉ CÔTE D'AZUR

- Create neural network
  - Use the torch.nn modules
    - Can use torch.nn.Modules
    - Or create a new class inheriting from torch.nn modules

- The training loop is

How does pytorch work?

UNIVERSITÉ CÔTE D'AZUR

- Create neural network
  - Use the torch.nn modules
    - Can use torch.nn.Modules
    - Or create a new class inheriting from torch.nn modules

- The training loop is

```
yhat = model(X)
L,acc = loss_accuracy(loss,yhat,Y)
optim.zero_grad()
L.backward()
optim.step()
```

Data efficient Image T. (Touvron et al. 2021)

UNIVERSITÉ CÔTE D'AZUR

Ablation on ↓	Pre-training	Fine-tuning	Rand-Augment	AutoAug	Mixup	CutMix	Erasing	Stoch. Depth	Repeated Aug.	Dropout	Exp. Moving Avg.	top-1 accuracy: pre-trained 224^2	top-1 accuracy: fine-tuned 384^2
none: DeiT-B	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8 ±0.2	83.1 ±0.1
optimizer	SGD	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	74.5	77.3
	adamw	SGD	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8	83.1
data augmentation	adamw	adamw	✗	✗	✓	✓	✓	✓	✓	✗	✗	79.6	80.4
	adamw	adamw	✗	✓	✓	✓	✓	✓	✓	✗	✗	81.2	81.9
	adamw	adamw	✓	✗	✗	✓	✓	✓	✓	✗	✗	78.7	79.8
	adamw	adamw	✓	✗	✓	✗	✓	✓	✓	✗	✗	80.0	80.6
	adamw	adamw	✓	✗	✗	✗	✓	✓	✓	✗	✗	75.8	76.7
regularization	adamw	adamw	✓	✗	✓	✓	✗	✓	✓	✗	✗	4.3*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✗	✓	✗	✗	3.4*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✗	✗	✗	76.5	77.4
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✓	✗	81.3	83.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✓	81.9	83.1

CNNs also benefit (Wightmann et al. 2021)

UNIVERSITÉ CÔTE D'AZUR

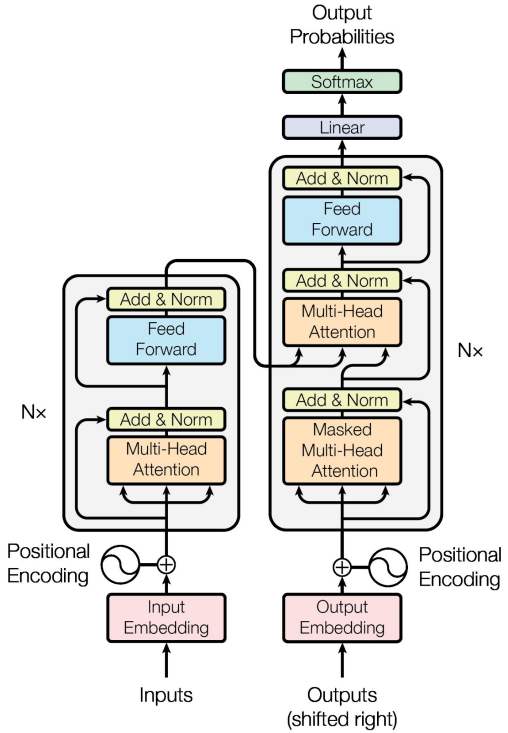
	Previous approaches					Ours		
Procedure →	ResNet	PyTorch	FixRes	DeiT	FAMS (×4)	A1	A2	A3
Reference	[13]	[1]	[48]	[45]	[10]			
Train Res	224	224	224	224	224	224	224	160
Test Res	224	224	224	224	224	224	224	224
Epochs	90	90	120	300	400	600	300	100
# of forward pass	450k	450k	300k	375k	500k	375k	188k	63k
Batch size	256	256	512	1024	1024	2048	2048	2048
Optimizer	SGD-M	SGD-M	SGD-M	AdamW	SGD-M	LAMB	LAMB	LAMB
LR	0.1	0.1	0.2	1×10^{-3}	2.0	5×10^{-3}	5×10^{-3}	8×10^{-3}
LR decay	step	step	step	cosine	step	cosine	cosine	cosine
decay rate	0.1	0.1	0.1	-	$0.02^{t/400}$	-	-	-
decay epochs	30	30	30	-	1	-	-	-
Weight decay	10^{-4}	10^{-4}	10^{-4}	0.05	10^{-4}	0.01	0.02	0.02
Warmup epochs	✗	✗	✗	5	5	5	5	5
Label smoothing ε	✗	✗	✗	0.1	0.1	0.1	✗	✗
Dropout	✗	✗	✗	✗	✗	✗	✗	✗
Stoch. Depth	✗	✗	✗	0.1	✗	0.05	0.05	✗
Repeated Aug	✗	✗	✓	✓	✗	✓	✓	✗
Gradient Clip.	✗	✗	✗	✗	✗	✗	✗	✗
H. flip	✓	✓	✓	✓	✓	✓	✓	✓
RRC	✗	✓	✓	✓	✓	✓	✓	✓
Rand Augment	✗	✗	✗	9/0.5	✗	7/0.5	7/0.5	6/0.5
Auto Augment	✗	✗	✗	✗	✓	✗	✗	✗
Mixup alpha	✗	✗	✗	0.8	0.2	0.2	0.1	0.1
Cutmix alpha	✗	✗	✗	1.0	✗	1.0	1.0	1.0
Erasing prob.	✗	✗	✗	0.25	✗	✗	✗	✗
ColorJitter	✗	✓	✓	✗	✗	✗	✗	✗
PCA lighting	✓	✗	✗	✗	✗	✗	✗	✗
SWA	✗	✗	✗	✗	✓	✗	✗	✗
EMA	✗	✗	✗	✗	✗	✗	✗	✗
Test crop ratio	0.875	0.875	0.875	0.875	0.875	0.95	0.95	0.95
CE loss	✓	✓	✓	✓	✓	✗	✗	✗
BCE loss	✗	✗	✗	✗	✗	✓	✓	✓
Mixed precision	✗	✗	✗	✓	✓	✓	✓	✓
Top-1 acc.	75.3%	76.1%	77.0%	78.4%	79.5%	80.4%	79.8%	78.1%

Refresher on Transformers

A new type of neural layer for everying

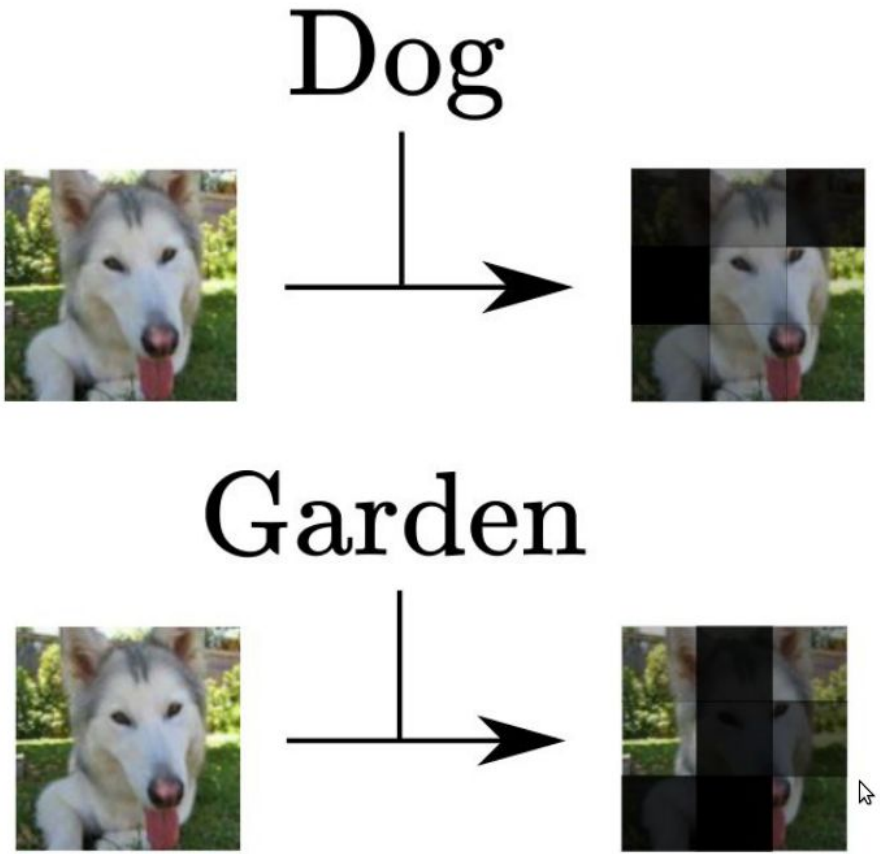
UNIVERSITÉ CÔTE D'AZUR

- What is the state of the art in
  - Computer vision?
    - ~~CNNs~~ -> **Transformers**
  - Natural language processing?
    - ~~RNNs~~ -> **Transformers**
  - Time series?
    - ~~RNNs or TCNs~~ -> **Transformers**
  - Multimodal problems?
    - ~~Hybrid?~~ -> **Transformers**



- Transformers use keeps increasing over time

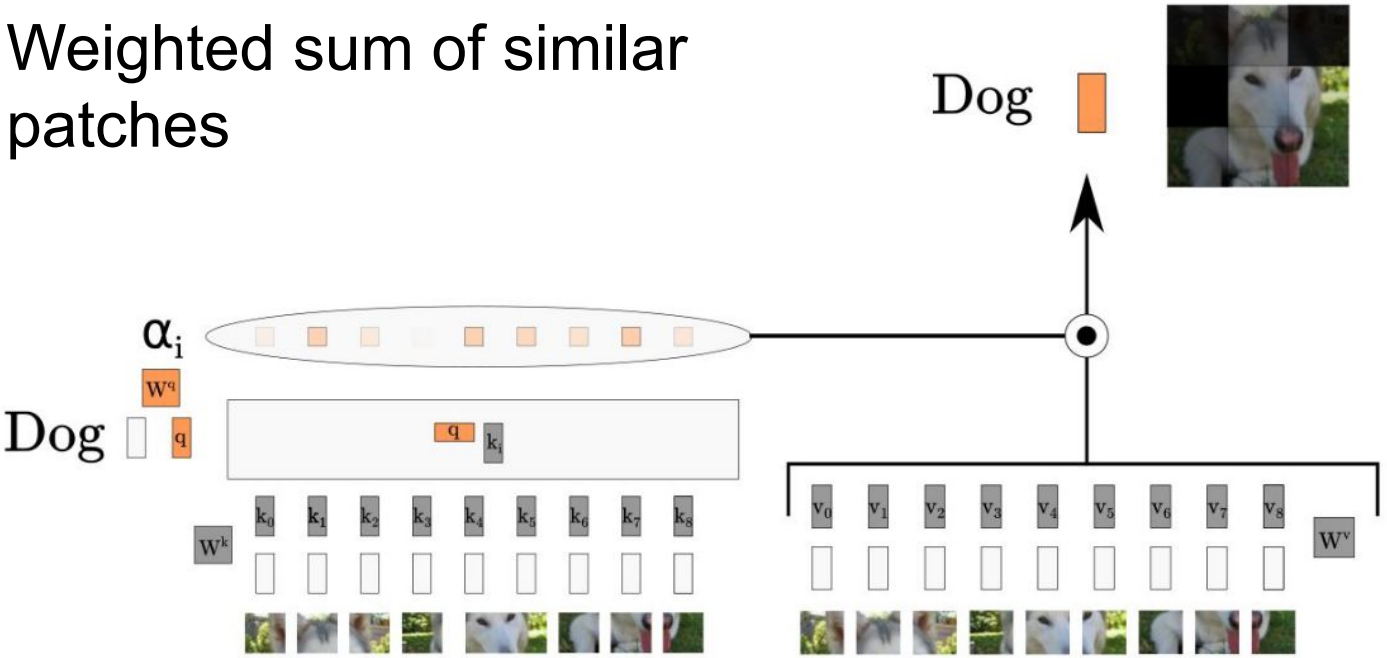
Main idea



- What is a Dog?
  - It is a 4 legged animal with fur and ears and eyes and a head and …
  - It is on this part of that picture.

Attended representation

- Weighted sum of similar patches



Attended representation

```python
def scaled_dot_product(q, k, v, mask=None):
    d_k = q.size()[-1]

    ########################
    ### YOUR CODE HERE! ###
    ########################

    # Compute attn_logits
    attn_logits = torch.matmul(q, k.transpose(-2, -1))
    attn_logits /= math.sqrt(d_k)

    # Apply mask if not None
    if mask is not None:
        attn_logits = attn_logits.masked_fill(mask == 0, - 1e14)

    # Pass through softmax
    attention = F.softmax(attn_logits, dim=-1)

    # Weight values accordingly
    output_values = torch.matmul(attention, v)

    ########################
    ###       END        ###
    ########################

    return output_values, attention
```

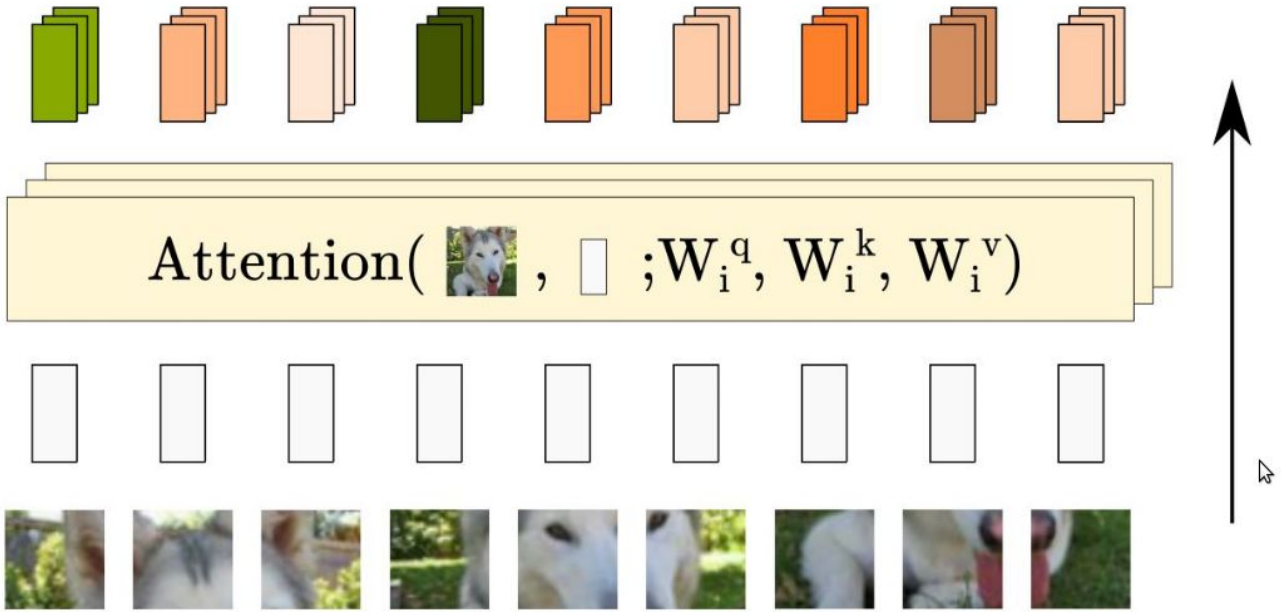
```python
def forward(self, x, mask=None, return_attention=False):

    ########################
    ### YOUR CODE HERE! ###
    ########################

    batch_dim, seq_length, input_dim = x.shape

    # Compute linear projection for qkv and separate heads
    # QKV: [Batch, Head, SeqLen, Dims]
    qkv = self.qkv_proj(x) # Batch x SeqLen x Hidden_dim * 3
    qkv = qkv.reshape(batch_dim, seq_length, self.num_heads, 3* self.head_dim)
    qkv = qkv.permute(0, 2, 1, 3)
    q, k, v = qkv.chunk(3, dim=-1)


    # Apply Dot Product Attention to qkv ()
    attention_values, attention = scaled_dot_product(q, k, v)

    # Concatenate heads to [Batch, SeqLen, Embed Dim]
    attention_values = attention_values.reshape(batch_dim, seq_length, self.embed_dim)

    # Output projection
    o = self.o_proj(attention_values)

    ########################
    ###       END        ###
    ########################

    if return_attention:
        return o, attention
    else:
        return o
```

Self-attention

- You can compare a sequence to itself!

MultiHead

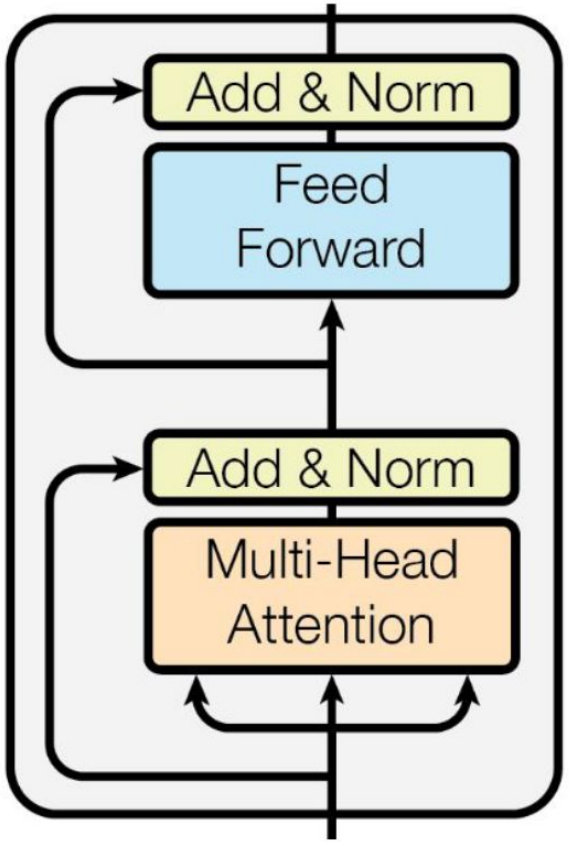
- And even have multiple interpretations



Transformer (Encoder) Block

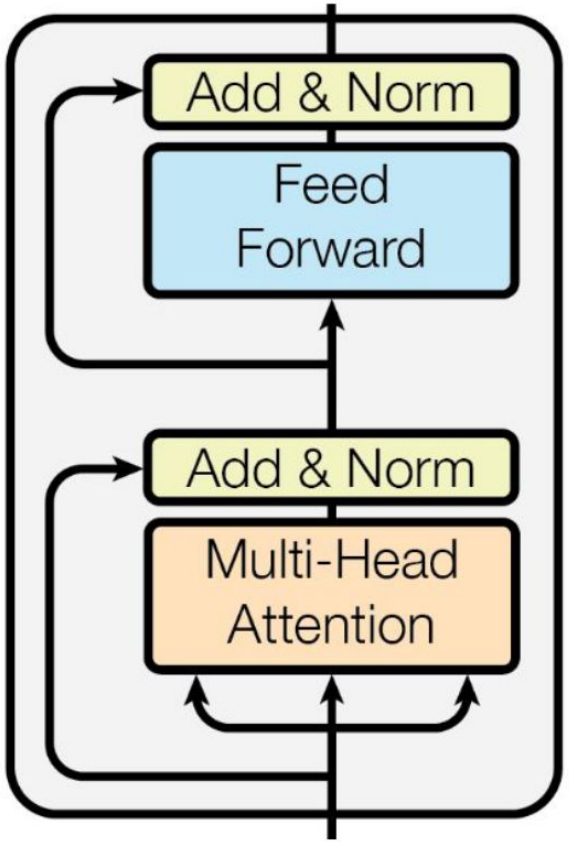
UNIVERSITÉ CÔTE D'AZUR

- Basic encoder block
  - MultiHead attention
  - Skip connection
  - Small MLP applied to each token
  - Skip connection
- Stacked in a transformer



Transformer (Encoder) Block

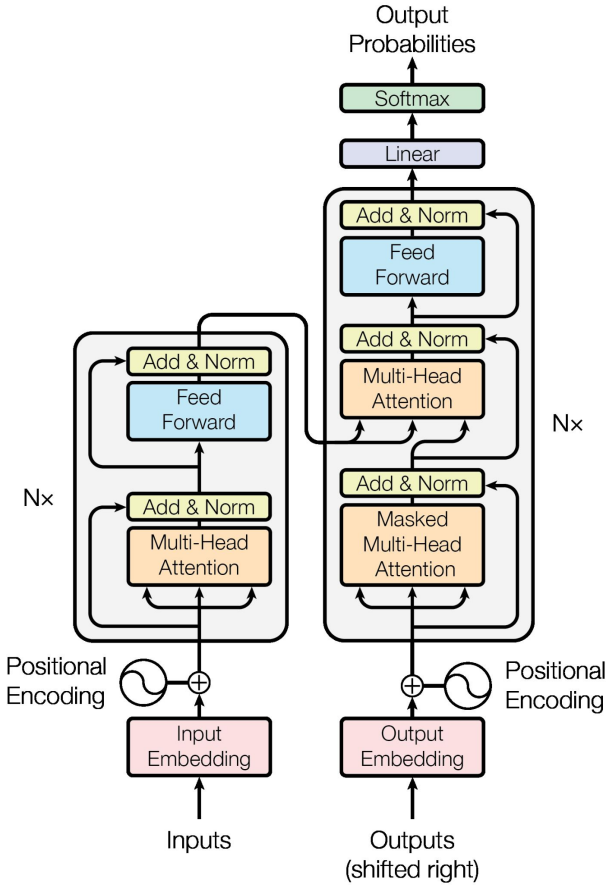
```python
class EncoderBlock(nn.Module):
    def __init__(self, input_dim, num_heads, dim_feedforward, dropout=0.0):
        """
        Args:
            input_dim: Dimensionality of the input
            num_heads: Number of heads to use in the attention block
            dim_feedforward: Dimensionality of the hidden layer in the MLP
            dropout: Dropout probability to use in the dropout layers
        """
        super().__init__()

        # Create Attention layer
        self.self_attn = MultiheadAttention(input_dim, input_dim, num_heads)

        # Create Two-layer MLP with droput
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, input_dim*2),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(2*input_dim, input_dim)
        )
        # Layers to apply in between the main layers (Layer Norm and Dropout)
        self.norm = nn.Sequential(
            nn.LayerNorm(input_dim),
            nn.Dropout(dropout)
        )

    def forward(self, x, mask=None):
        # Compute Attention part
        attn=self.self_attn(x)
        x=self.norm(attn+x)

        # Compute MLP part
        x = self.norm(x+self.mlp(x))

        return x
```



Seminal success: Vaswani et al. 2017

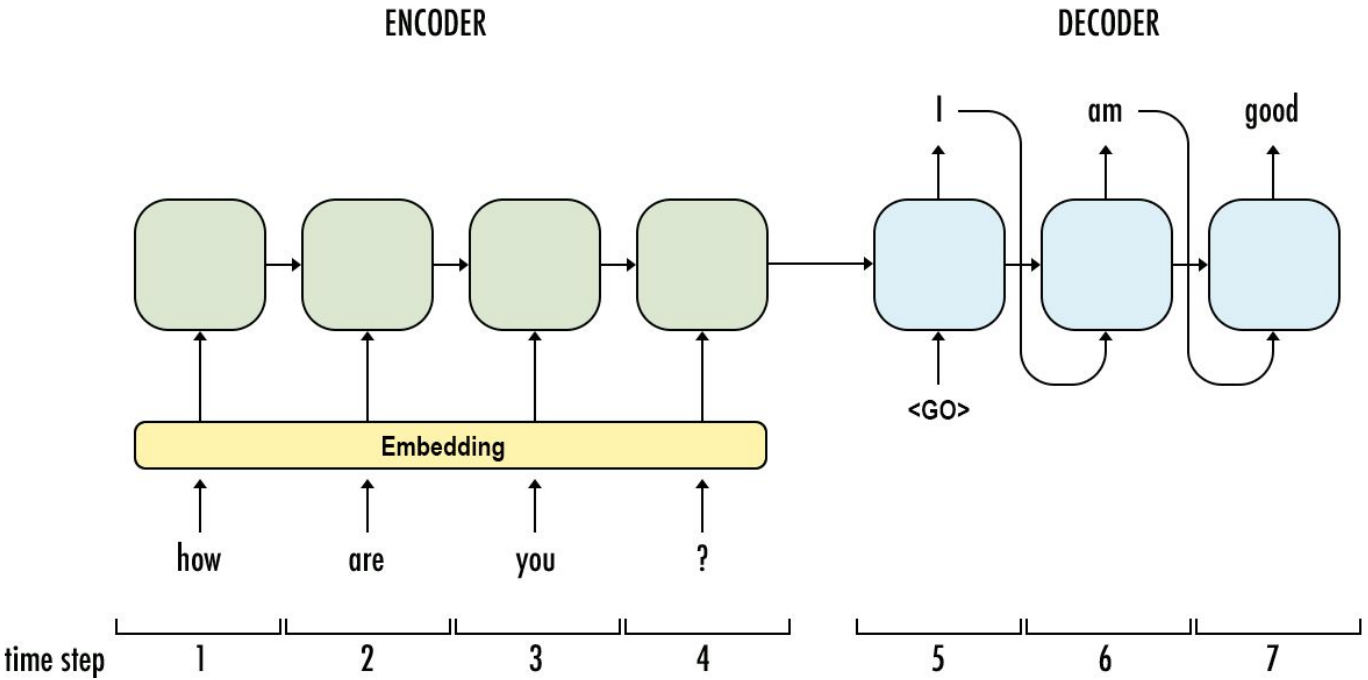
- Completely does away with recurrent units
  - Attention as a first class citizen!
  - Introduces element wise MLP for transform

- Transformer
  - Transforms the input throughout the layers
  - Also to blame for BERT, ELMO, DALL-E, …

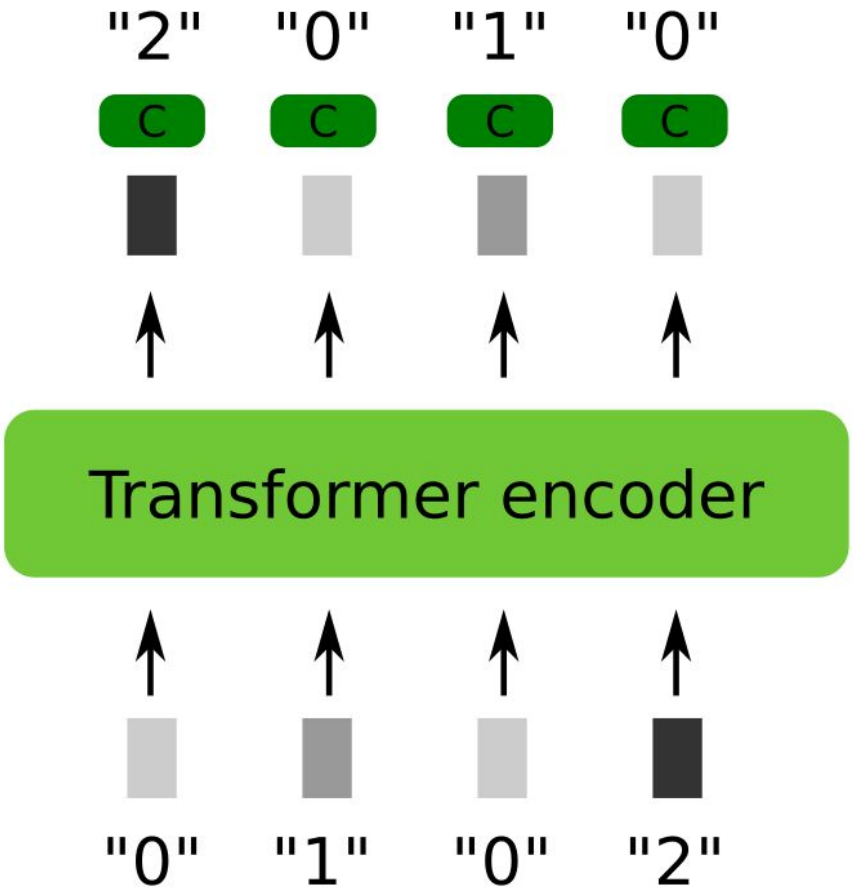


Autoregressive prediction

- Predict next token, feed it back into the decoder

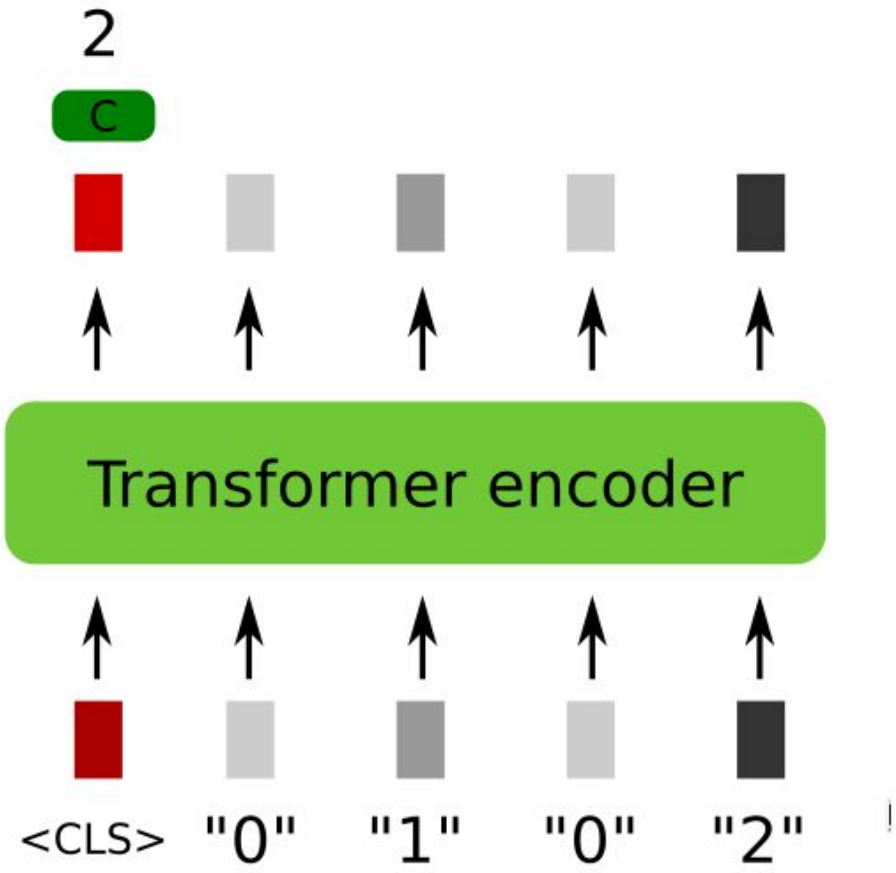


Token level predictions



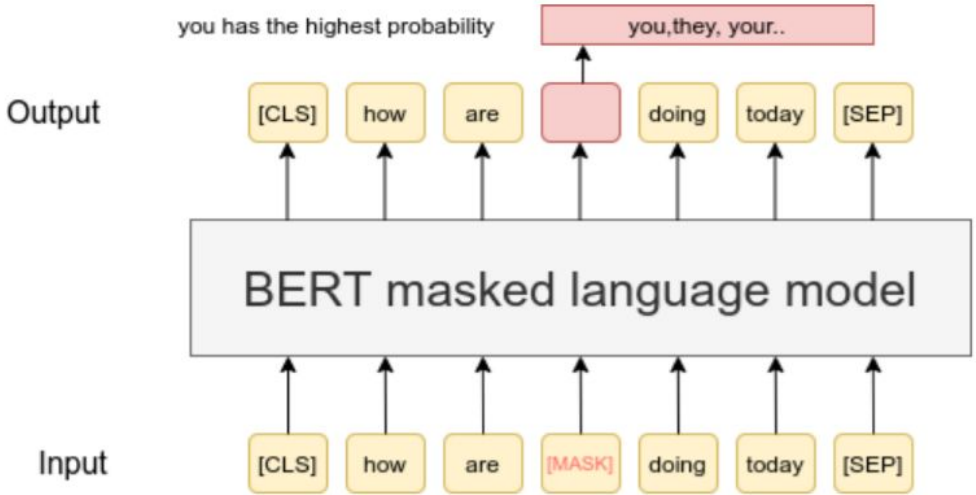
- Rich token features
  - Due to attention!
- Predictions for each token
  - Invert sequence here
- Predict with linear layers

Classification token



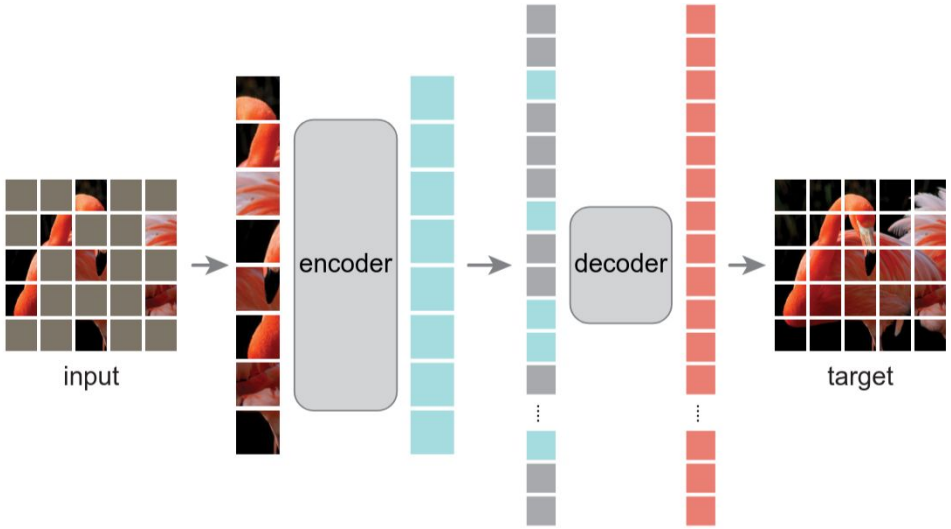
- Rich token features
  - Due to attention!
- Question/Classification token
  - “What is the class”
- Accumulate relevant info from other tokens

BERT pretraining



- Pseudo-objective
  - Mask part of the sentence
  - Try to predict the masked part

Masked auto-encoders



- Also works for images!
- Originally introduced in BeIT paper

Vision Transformers as backbones

UNIVERSITÉ CÔTE D'AZUR

- Extract rich features from images
  - Including long range dependencies
- Masked token prediction for pretraining
  - Only works with transformers
  - Much better than traditional contrastive training
- Replace CNNs as backbones for downstream tasks
  - Segmentation
  - Detection

Large models (LLMs, VLMs, and image generators)

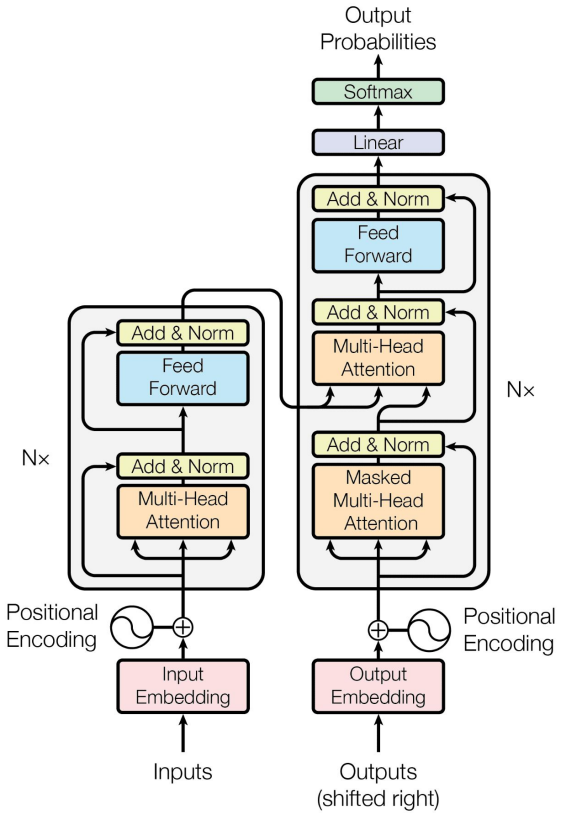
LLMs

… And text to image models!

Large Language Models

UNIVERSITÉ CÔTE D'AZUR

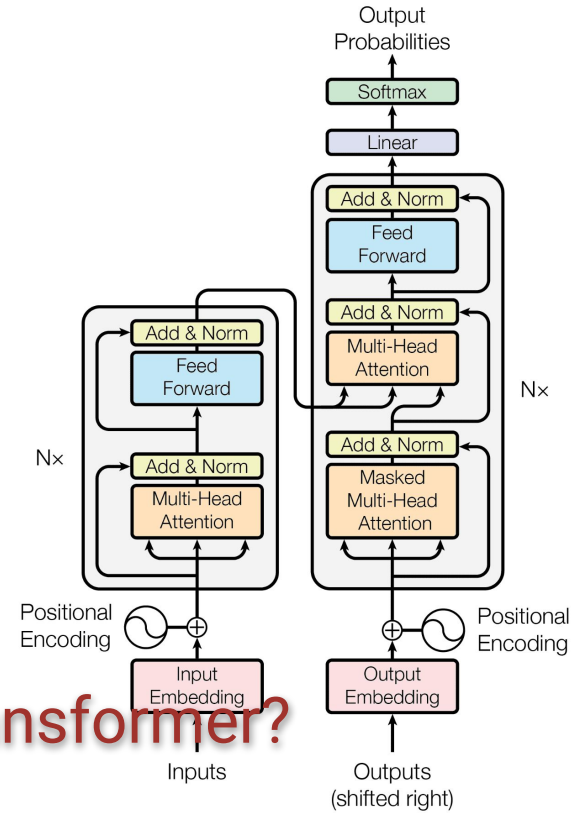
- General architecture
  - Encoder
  - Decoder
- Very impressive performance
  - ChatGPT
  - Bard
  - Copilot



Large Language Models

UNIVERSITÉ CÔTE D'AZUR

- General architecture
  - Encoder
  - Decoder
- Very impressive performance
  - ChatGPT
  - Bard
  - Copilot

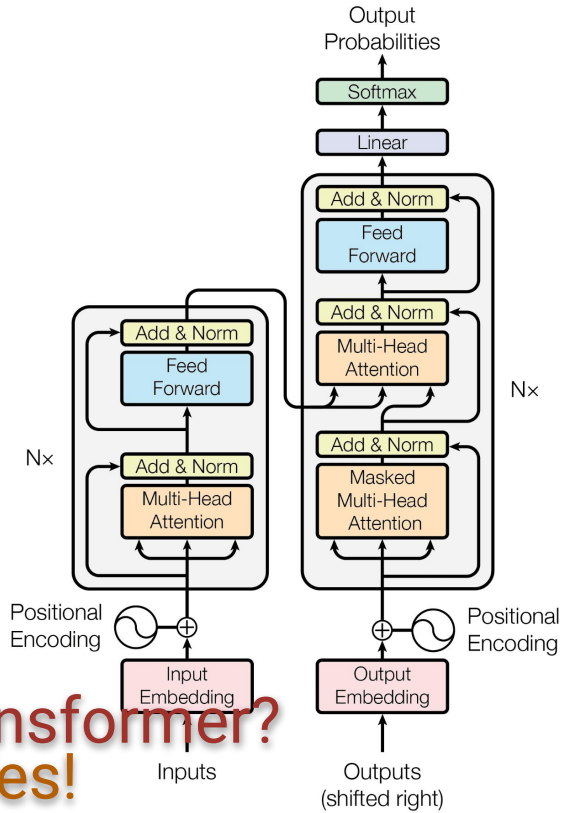


That is just a transformer?

Large Language Models

UNIVERSITÉ CÔTE D'AZUR

- General architecture
  - Encoder
  - Decoder
- Very impressive performance
  - ChatGPT
  - Bard
  - Copilot

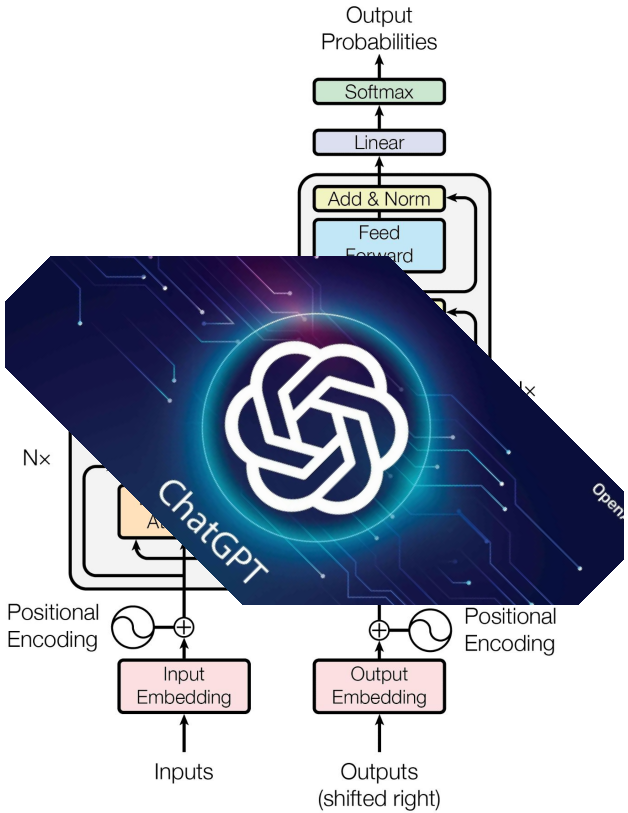


That is just a transformer?
Yes!

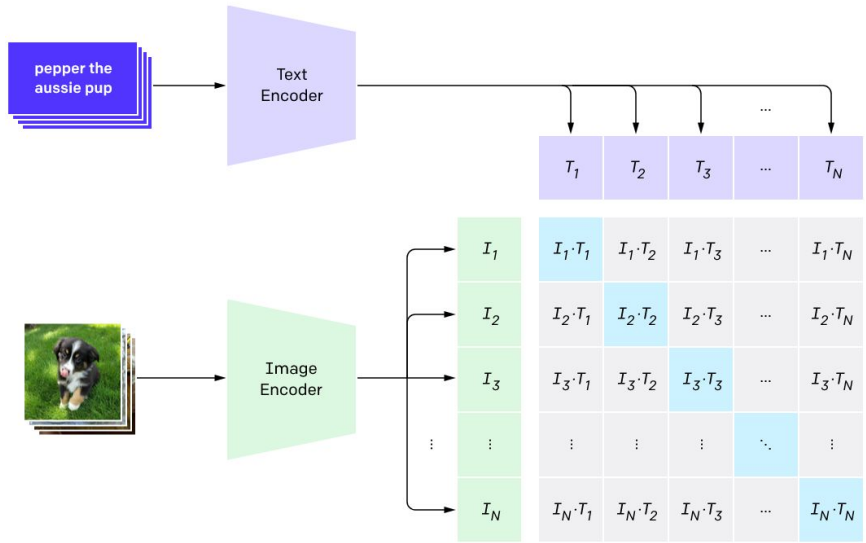
Large Language Models

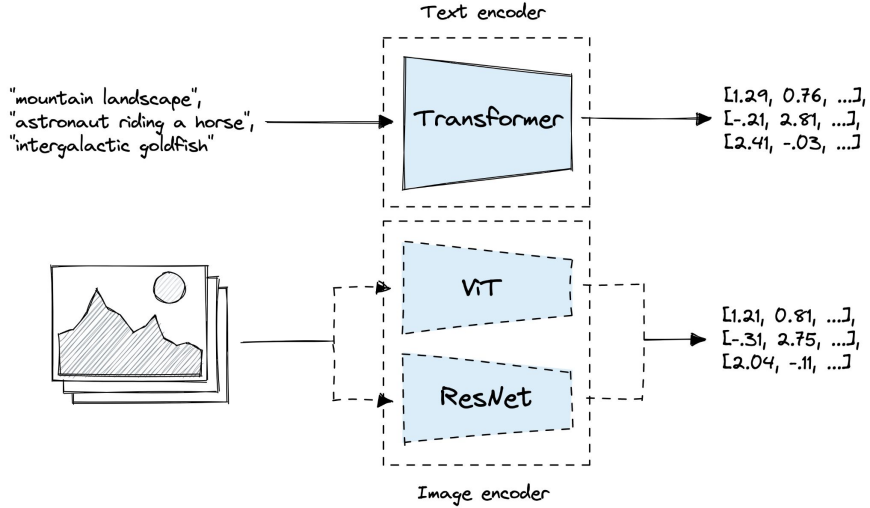
UNIVERSITÉ CÔTE D'AZUR

- GPT: Generative Pretrained Transformer
  - Pretrain as a language model (encoder/decoder)
  - Keep encoder for task specific fine-tuning
  - Decoder-only now
- Train on lots and lots of data!
  - Along with a few neat tricks

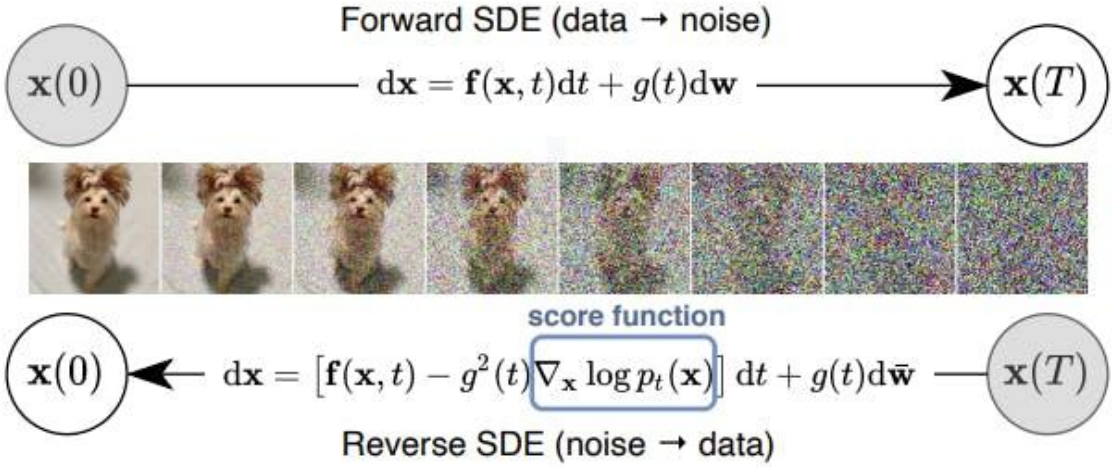


Vision Language Models





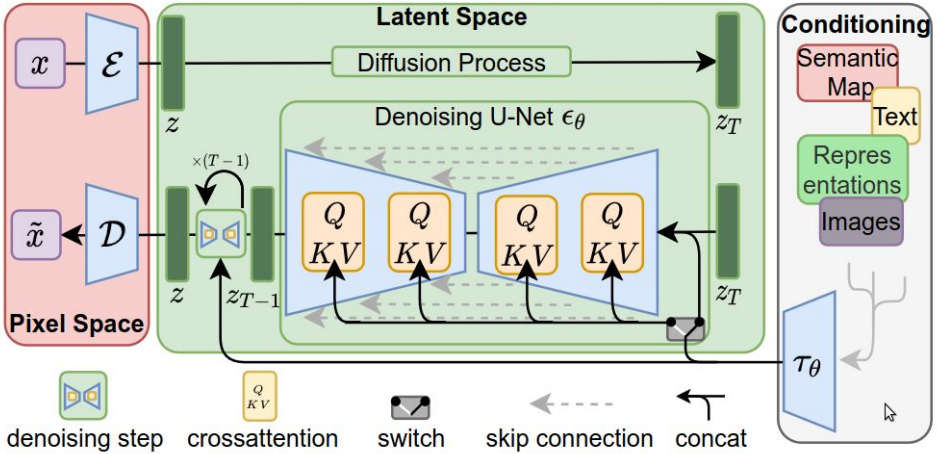
Generating images (with Diffusion)



- Generator
  - Image space
  - Latent space
- Learn to link the two
  - With network
  - Iteratively (diffusion)

Text-to-Image models

- Generator
  - Random z
  - Map z to some image x
  - Gradual process called Diffusion
- Inject condition tokens
  - Cross attention in generative process

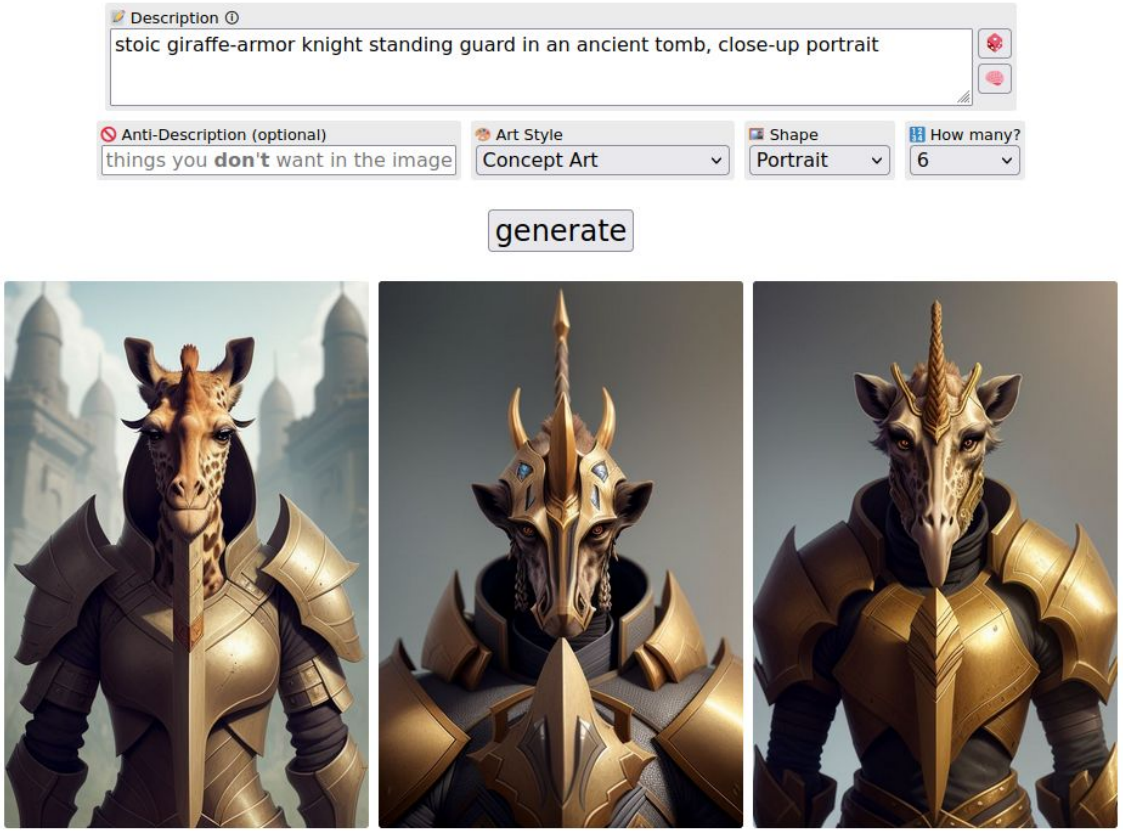


Takeaway

- LLM: Standard transformer encoder/decoder
  - Trained with lots and lots of data!
- Text-to-Image: Generative model contextualized by conditioning information
  - Trained with lots and lots of data!
- Very impressive results
  - Facilitated by the expressive power of attention
  - As long as you have lots and lots of data!

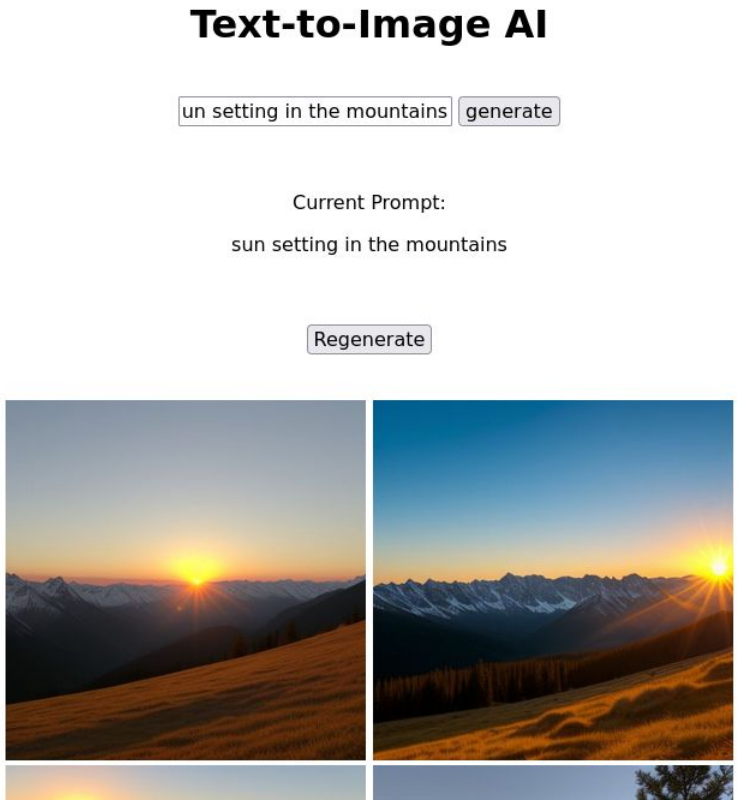
Character generator

https://perchance.org/ai-character-generator



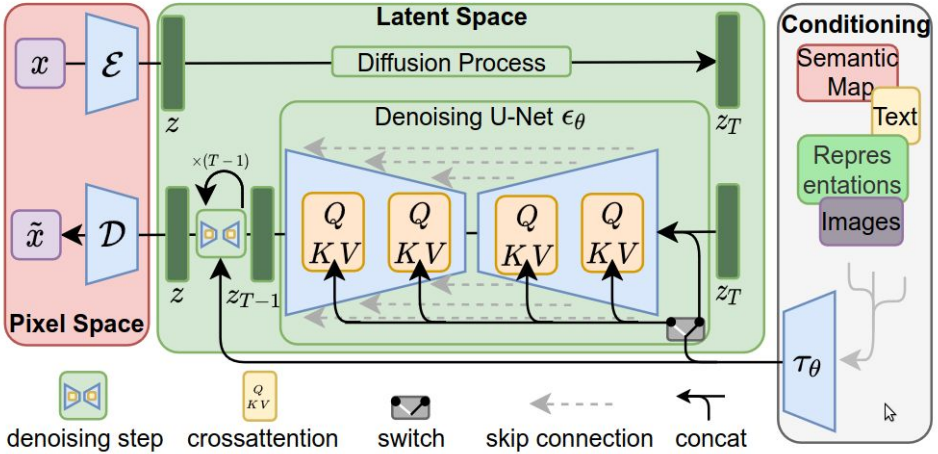
Text-to-Image

https://perchance.org/aiimageaccess



It is an issue of how you present it

- Same underlying model
  - Stable Diffusion
- Text-to-image
  - “Pure” Stable Diffusion
- Character generator
  - Stable Diffusion w/ Custom text prompts

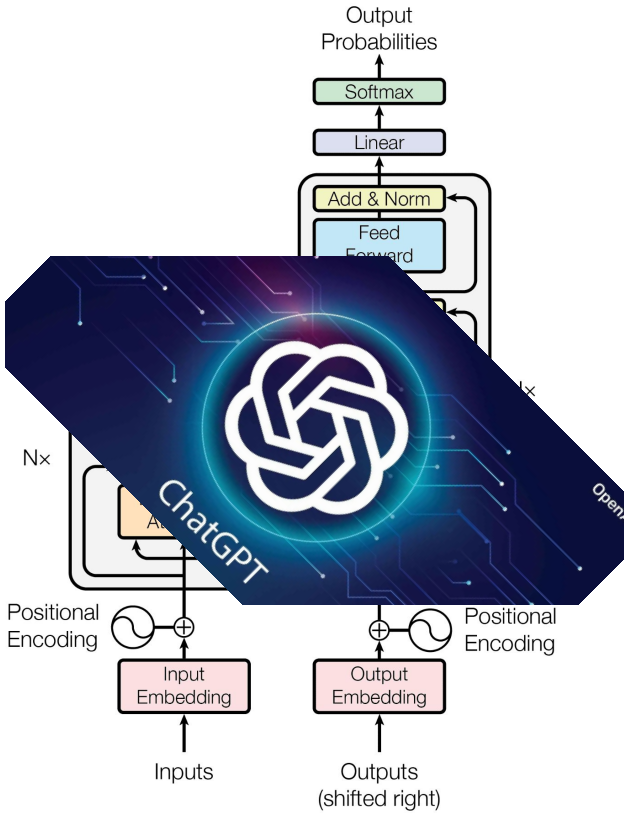


What goes into training a Large (Language) Model?

Road to ChatGPT

UNIVERSITÉ CÔTE D'AZUR

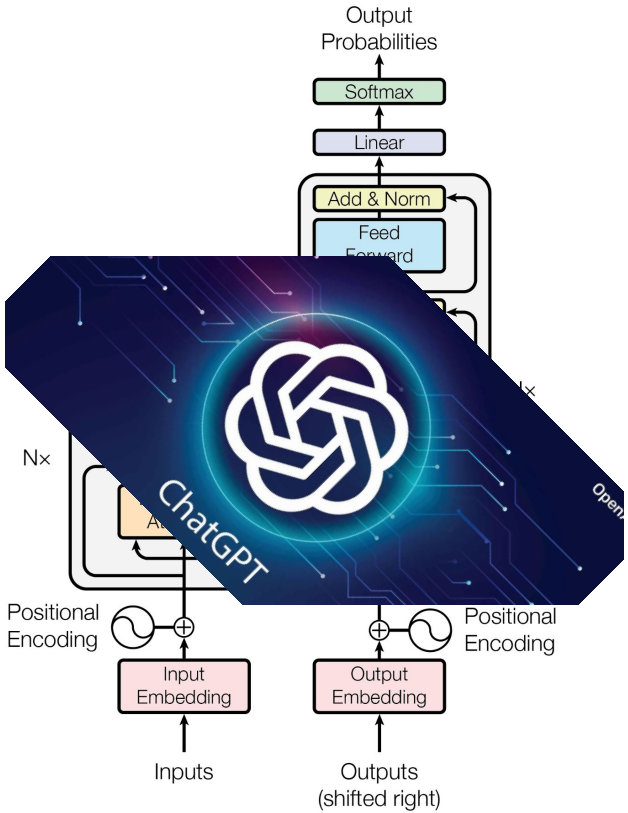
- What do we need to do?
  - Get (lots of) data
  - Prepare data
  - Get a model
  - Make the model work
  - Align it to our preferences
  - Deploy!



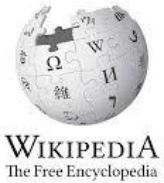
Road to ChatGPT

UNIVERSITÉ CÔTE D'AZUR

- What do we need to do?
  - Get (lots of) data
  - Prepare data
  - Get a model
  - **Make the model work**
    - Takes some work
  - Align it to our preferences
  - Deploy!



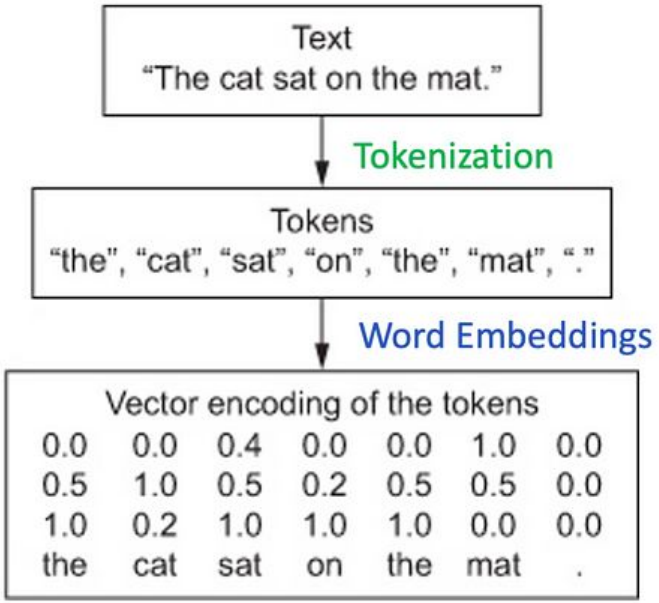
Get data and define your tasks



- Grab data that fits your general case
  - From public data
    - Running out of new public data…
  - From (your?) private data

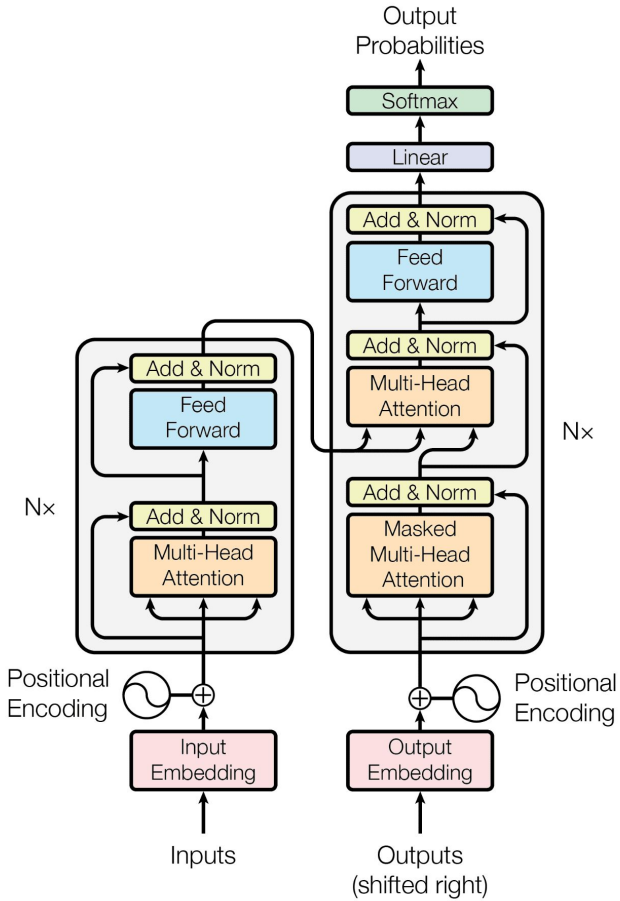
Prepare the data!

- Format text for your problem
- Tokenize into atomic elements
  - Letters, n-grams, words
- Embed in vector space
  - Index -> vector
  - Can be learned



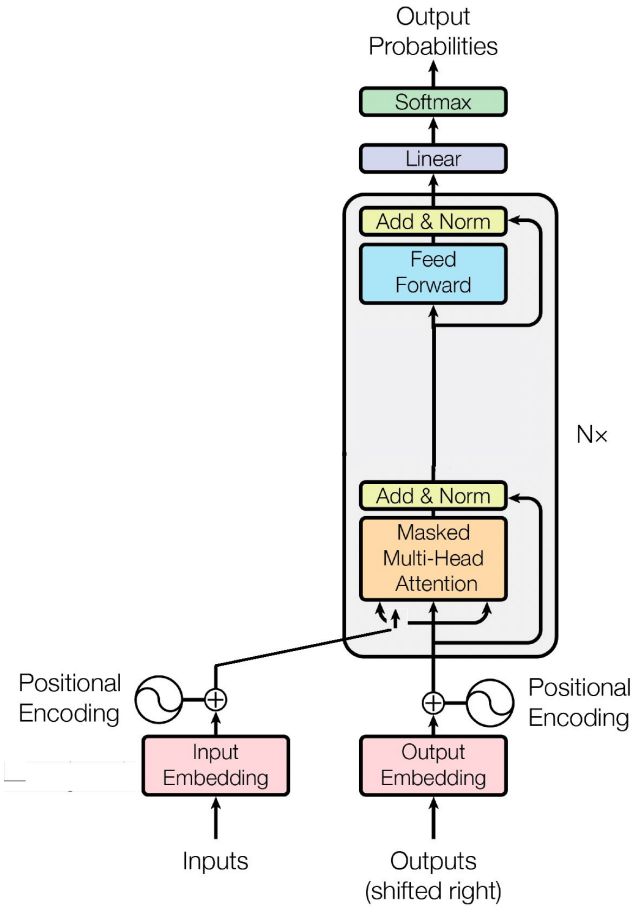
Build a model

- Transformer architecture
  - Suitable for your problem
    - Big enough for data
  - Studied in class!
- Could be something else
  - RNN, SSM, Mixer, …
- Fairly straightforward



Build a model

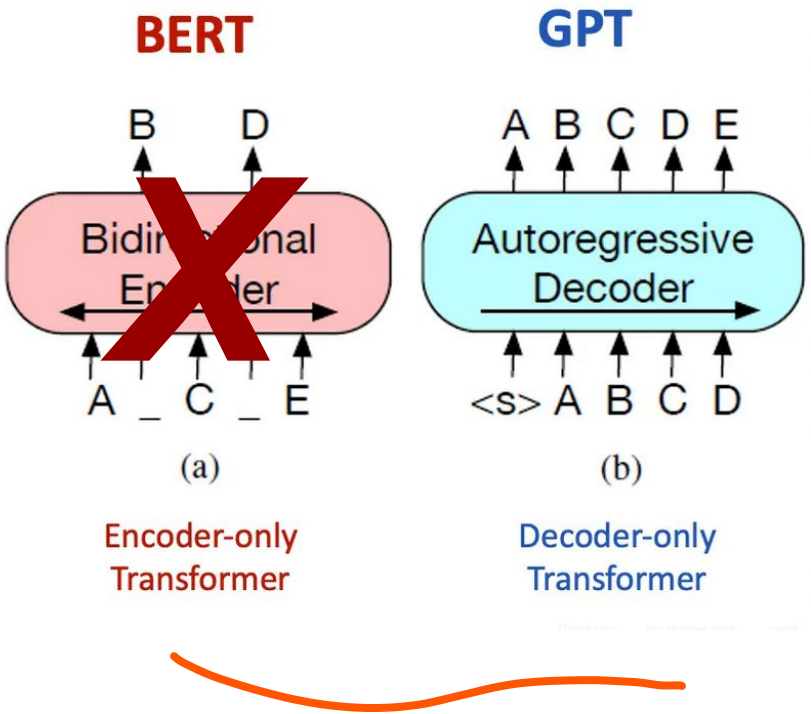
- Transformer architecture
  - Suitable for your problem
    - Big enough for data
  - Studied in class!
- Could be something else
  - RNN, SSM, Mixer, …
- Fairly straightforward
  - Decoder-only nowadays



(Pre-)train the model on large dataset

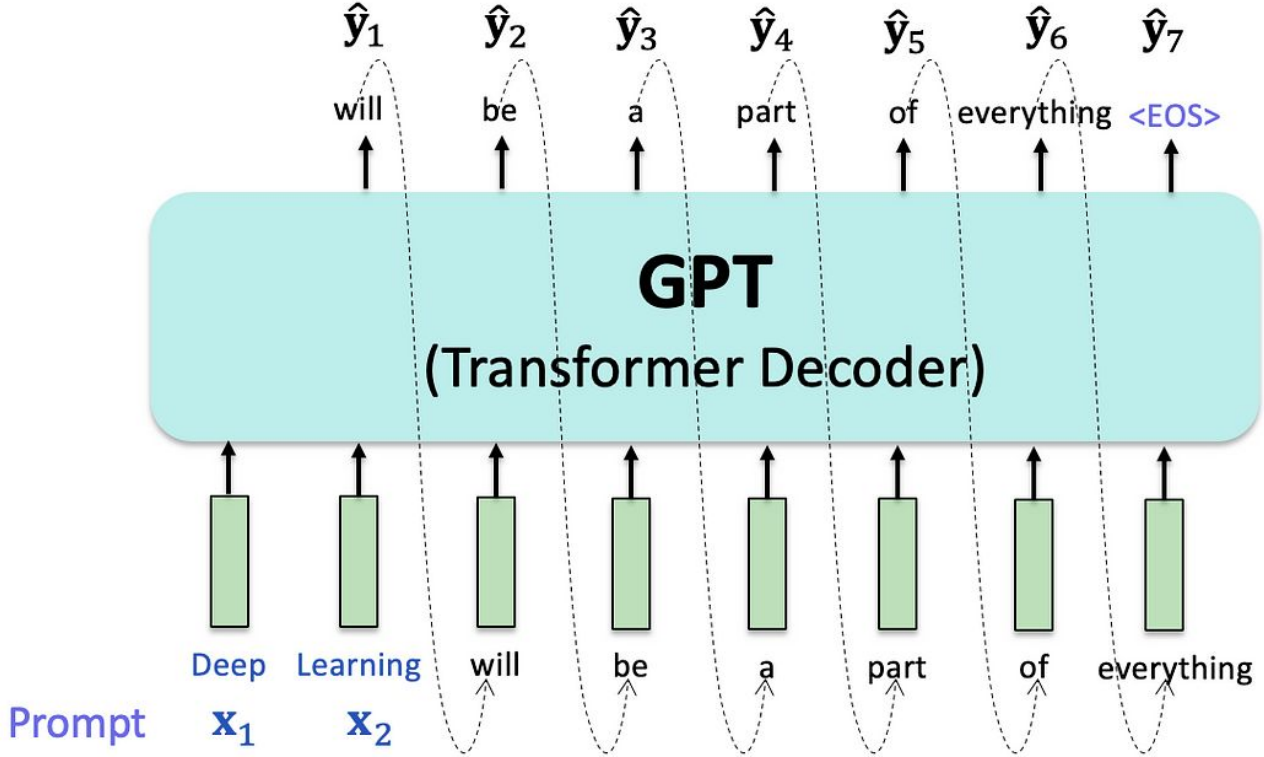
- Train on large amounts of generalist data with no label
  - Encoder: Masked pre-training
  - **Decoder: Next token prediction**

- What we use our large dataset for

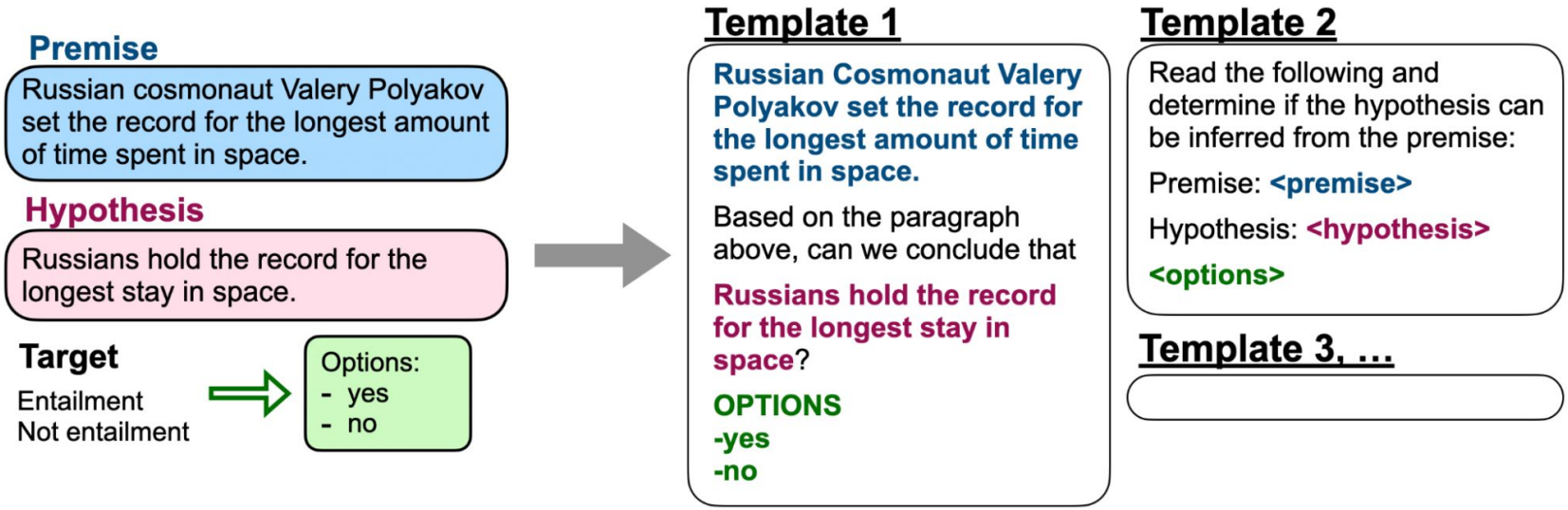


Autoregressive prediction

- Predict next token, feed it back into the decoder

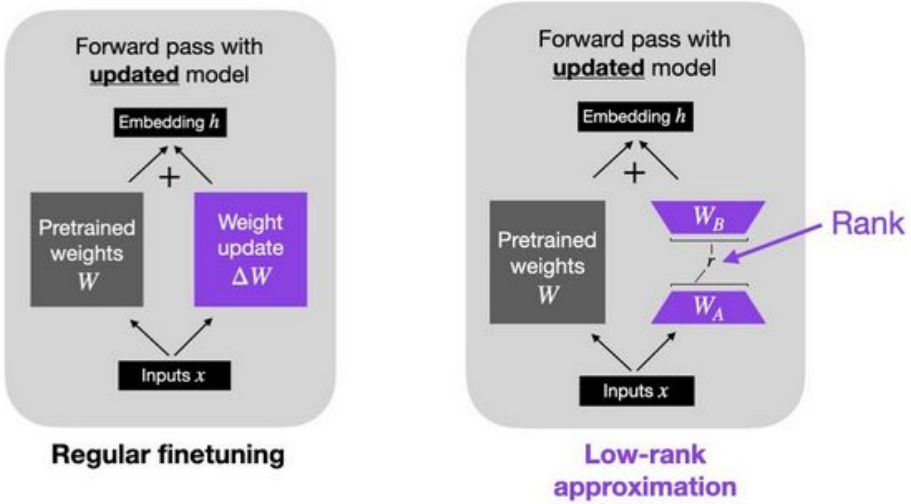


Fine-tune to our purposes



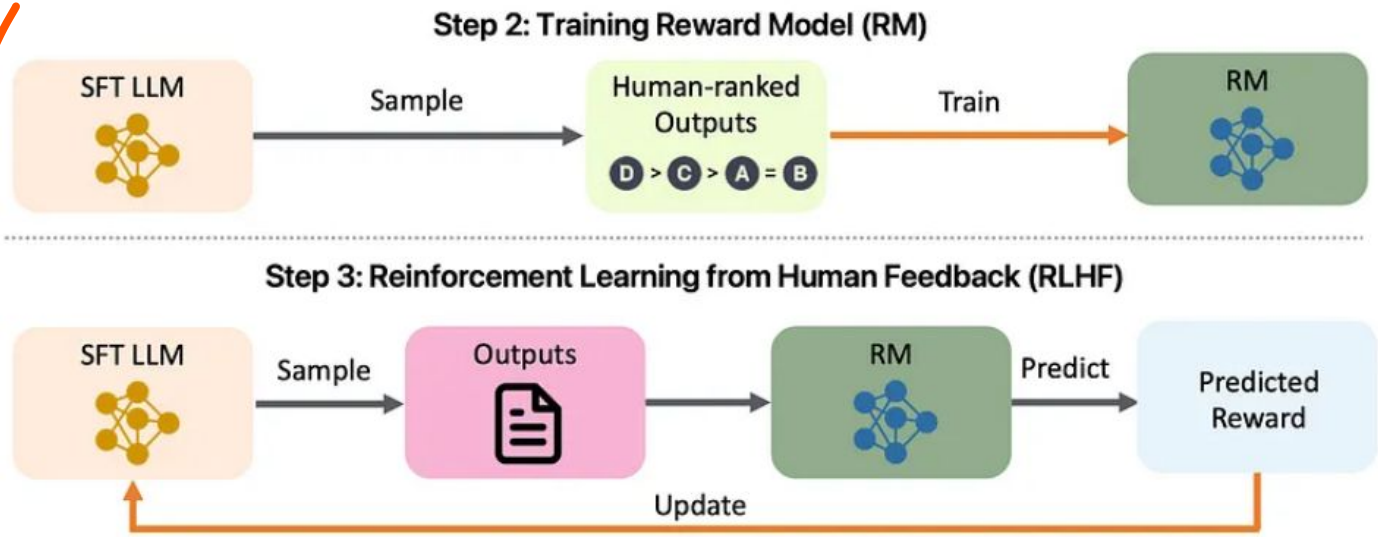
- Large dataset is unspecialized
  - You do not want wikipedia completion…
  - Make smaller specialized dataset
    - Fine tune on it

Fine-tune to our purposes (efficiently)



- Fine tuning is expensive for LLMs
  - Can fine tune “smaller” weights
  - Fine tune a low rank adapter on weights

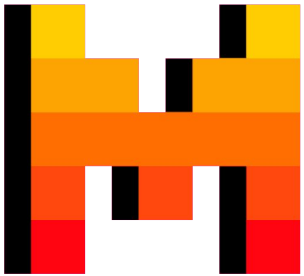
Alignment to preferences



- Need to align to human preference
  - Train proxy model to emulate preferences
  - Use proxy model to fine-tune LLM (PPO, DPO)

(Sell access to your model)

- What you have
  - A really big model
    - Not given to client
  - Code to run the model with inputs
    - See Lab session
- What you sell
  - API access
    - The right to send inputs and get answers



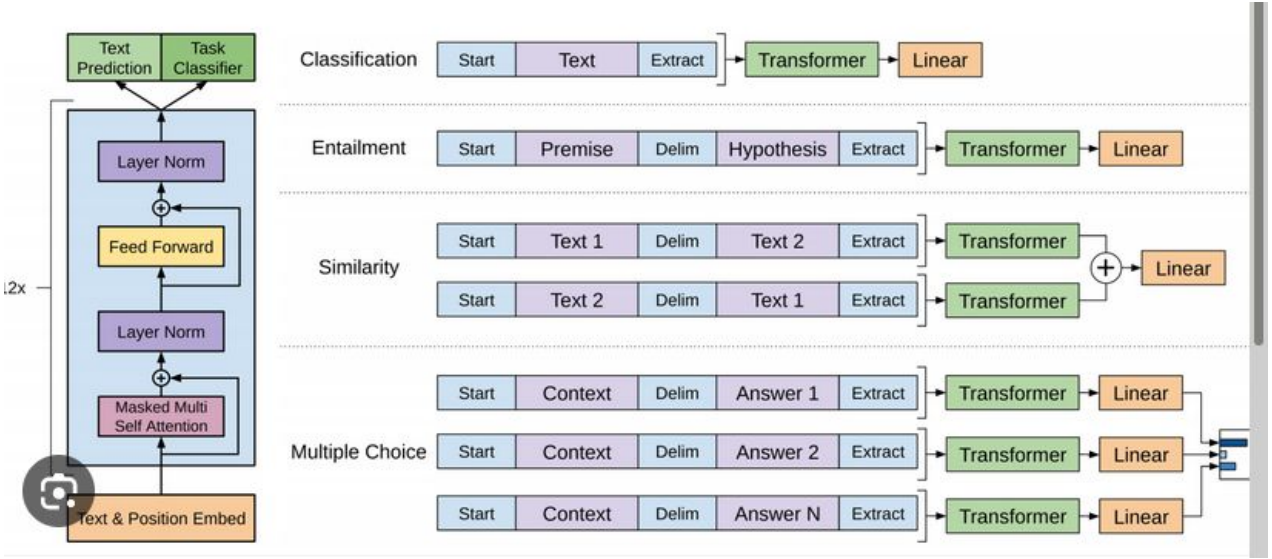
Takeaway

- LLMs are trained with
  - Huge amounts of generalist data
  - Some specialized date
    - For a specialized downstream task
  - Proxies for human feedback
  - Enormous computational resources
  - Transformer blocks
    - Decoder-only nowadays

- LLMs learn to predict words in sentences

About prompts

Fine-tuning



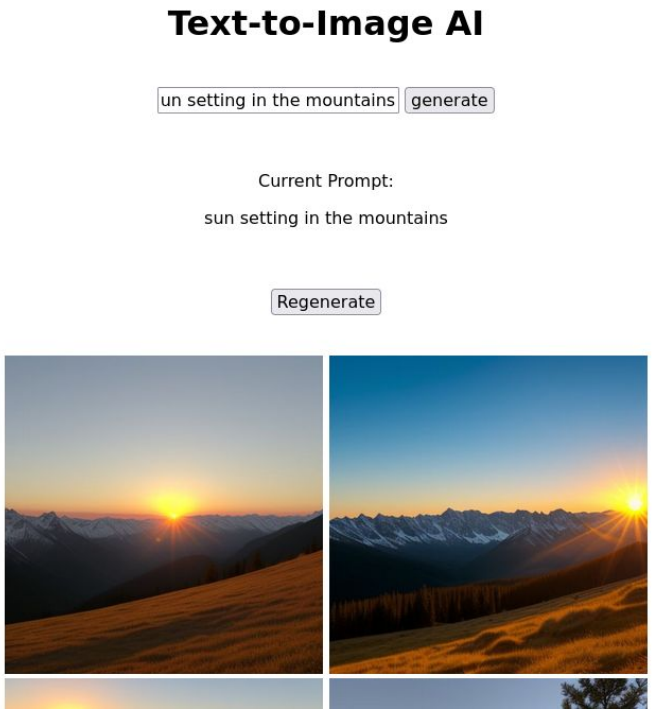
- Fine tune encoder on downstream tasks

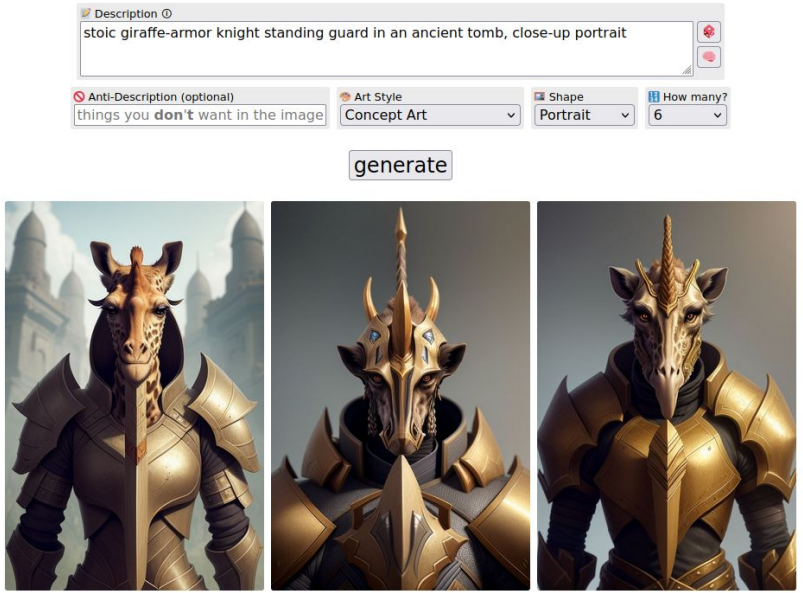
- So expensive (huge models!)
  - And maybe we already finetuned

Let’s not train

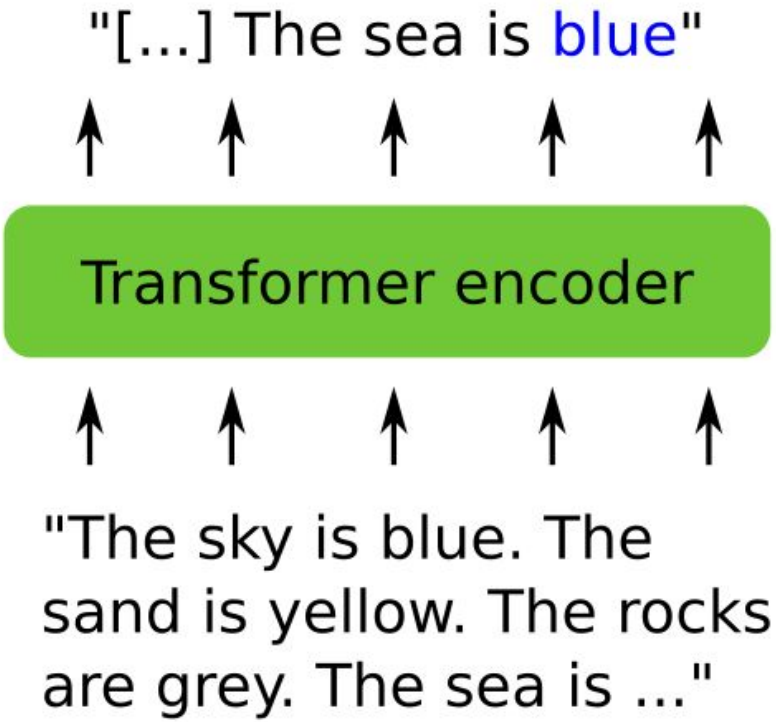
Training is hard, let’s not do it
But how ?

Remember this ?



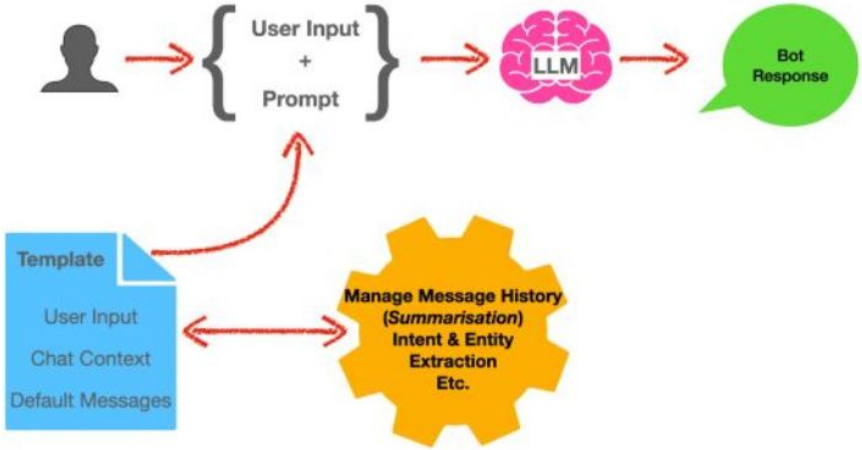


Prompt engineering



- E.g. Make a logical sequence to condition the model
- Need to find the right conditioning
  - Prompt engineering
  - Templating, Chain-of-thought, …
- No training!

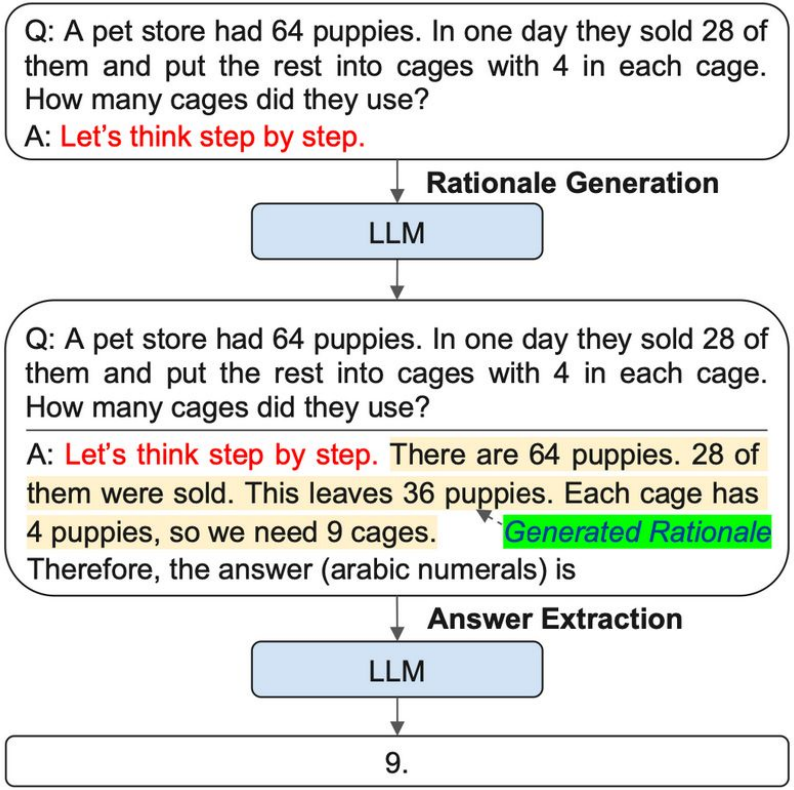
General context and framing



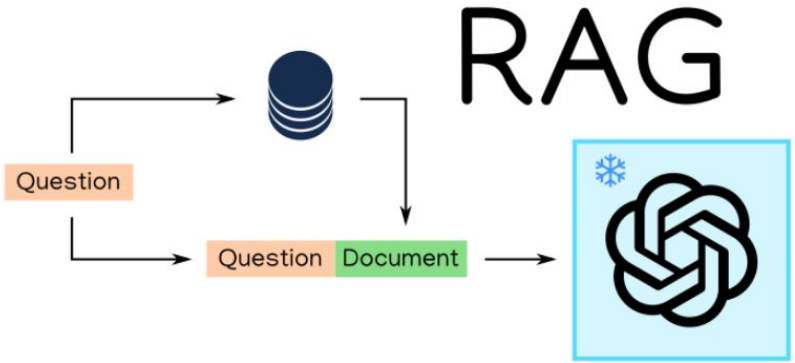
- The answer is learned behavior + input context
  - Put as much as possible in input
    - Template, behavior conditioning, history, …

Chain-of-Thought prompting

- LLMs output likely chains of words
  - Can lead to large leaps of logic
- Chain-of-Thought
  - Make network output smaller steps
  - Each smaller step more reliable

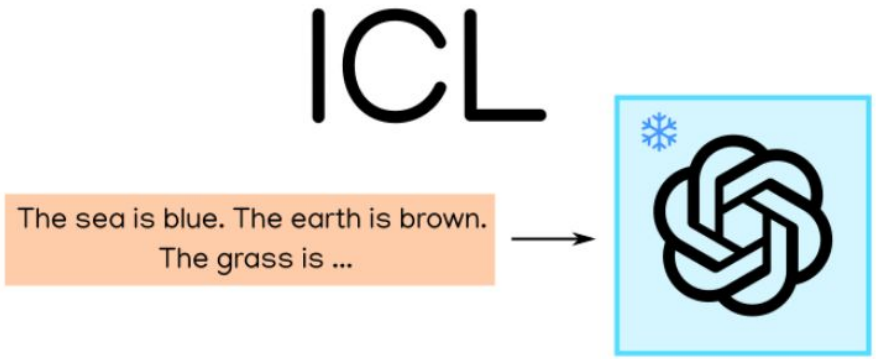


Retrieval Augmented Generation (RAG)



- Sometimes learned behavior + context input is not enough
  - Finetune learned behavior? Expensive
  - Add to context input by adding relevant documents from database

In-context learning (ICL)



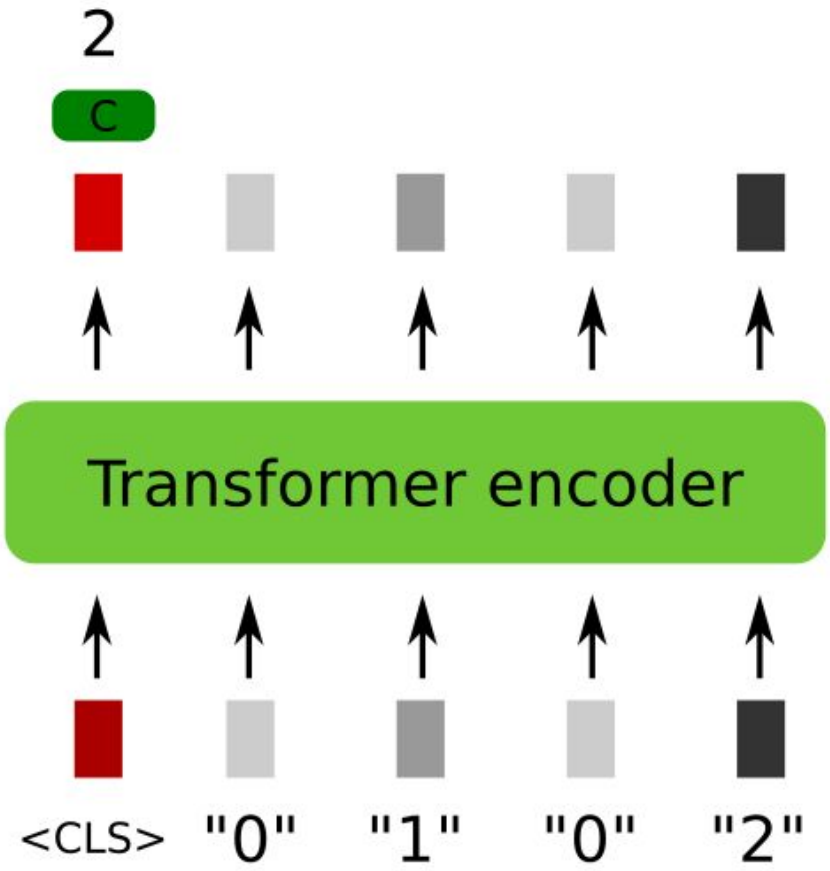
- Can give example to neural network to specialize the task done by network
  - Few-shot learning
    - Good results for tabular data!
  - Conditioning the network behavior

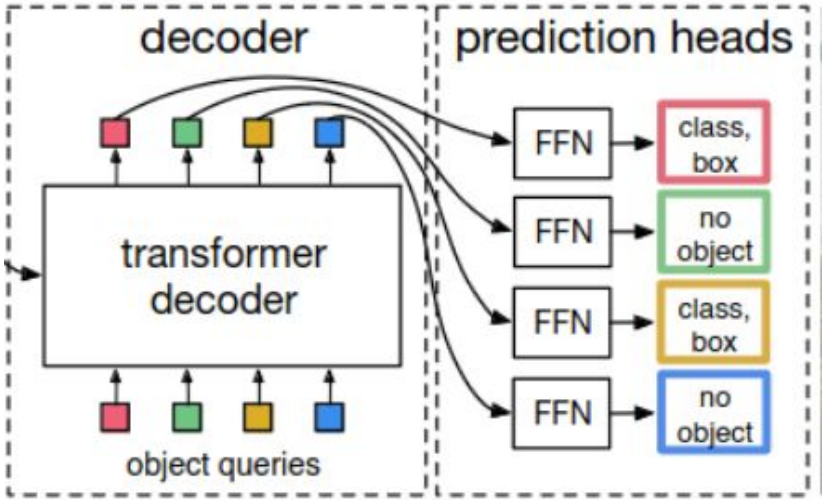
Let’s train a little

Ok, but surely we cannot do everything like that.
Maybe we could train a little?

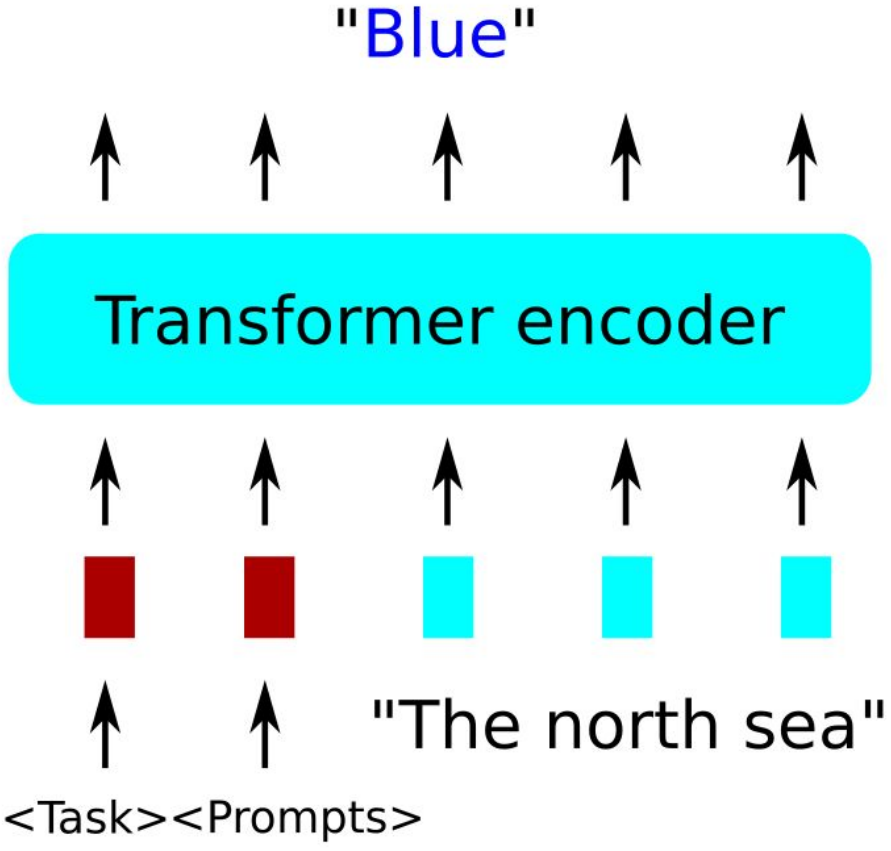
How?

Remember this ?





Prompt tuning/learning



- Add task tokens
  - **Only** optimize these tokens
  - Prompt tuning /learning
- Can be SGD
  - “Heavy” gradients
- Can be done with alternative optimization

Takeaway

- Transformers are very powerful
  - But very data hungry
  - Freeform attention

- Tokens are a very powerful tool in transformers
  - Classification/detection tokens
  - Contextual prompt engineering
  - Prompt learning/tuning

- Departure from classical fine-tuning

Large Language models and reasoning

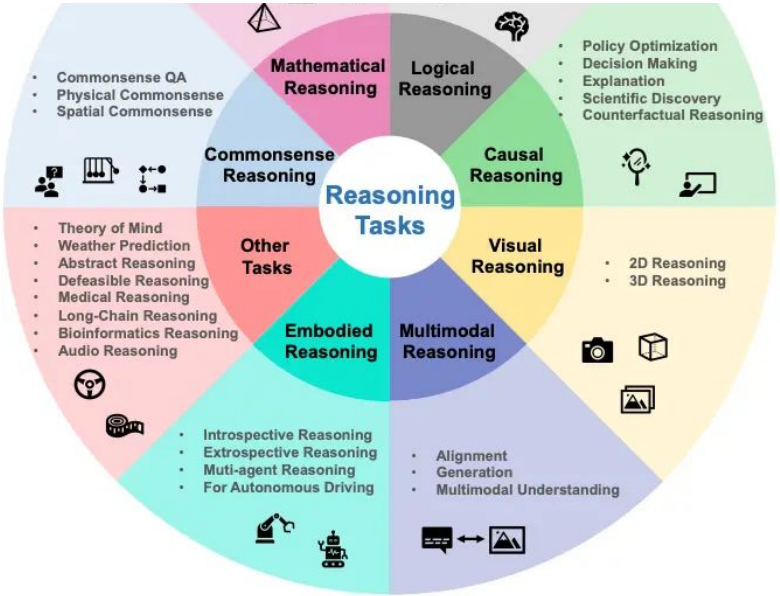
Introduction

Google DeepMind

AI achieves silver-medal standard solving International Mathematical Olympiad problems

Breakthrough models AlphaProof and AlphaGeometry 2 solve advanced reasoning problems in mathematics.

Jul 25, 2024



Support

Reasoning Techniques

- Pre-Training
- Fine-tuning
- Mixture of Experts (MoE)
- Alignment Training
- In-Context Learning
- Autonomous Agent

Introduction

Google DeepMind

AI achieves silver-medal standard solving International Mathematical Olympiad problems

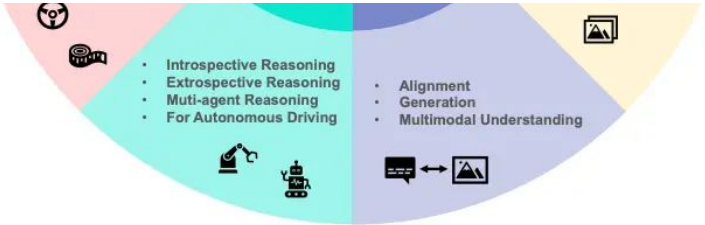
Breakthrough models AlphaProof and AlphaGeometry 2 solve advanced reasoning problems in mathematics

Ars Technica

Apple study exposes deep cracks in LLMs' "reasoning" capabilities

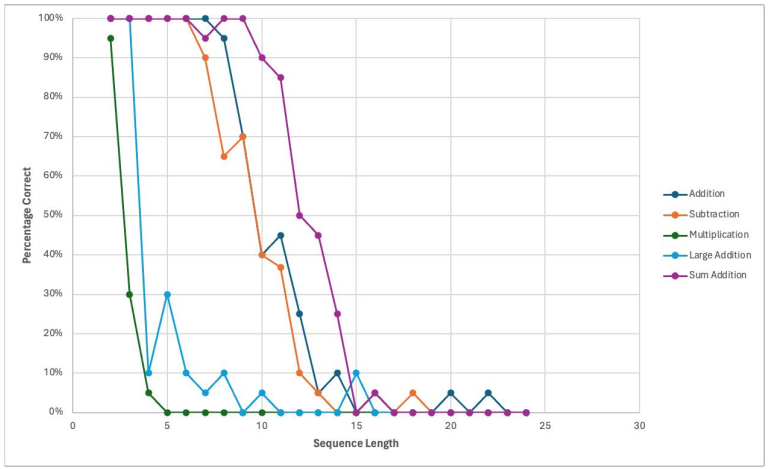
A new study from six Apple engineers shows that the mathematical reasoning displayed by advanced large language models can be extremely brittle and unreliable.

1 month ago



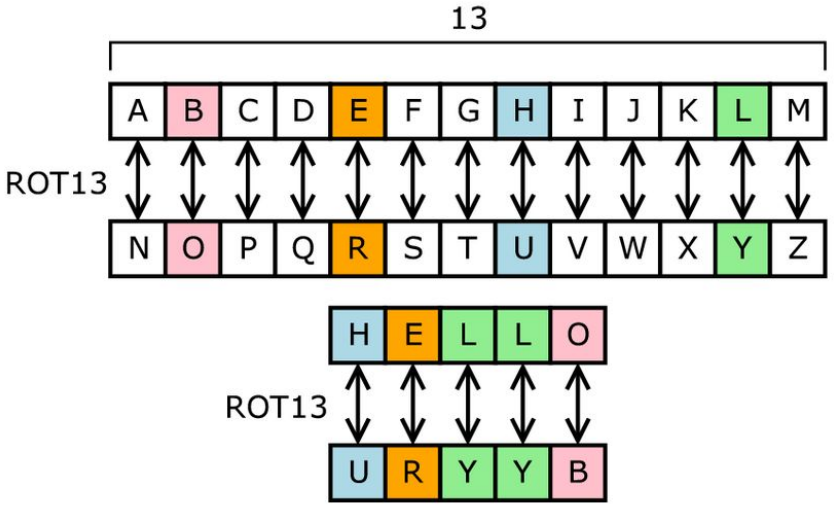
Case study: Simple arithmetic

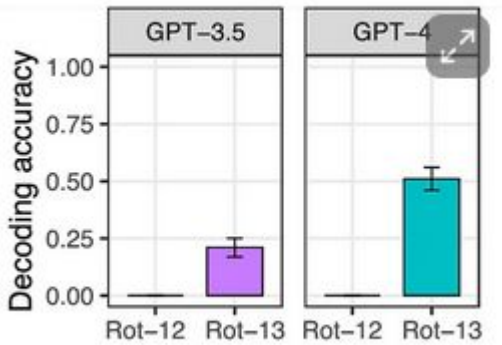
- Ask simple arithmetic problems to LLM
  - No targeted training
- Kind of ok
  - For very simple problems
  - With short sequences



Issues with reasoning benchmarks

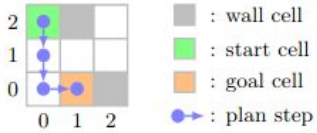
- Models can perform very complex tasks
  - E.g. decoding the ROT-13 cipher
    - Move letters by 13 units
- Tied to common problems
  - Horrible on ROT-12





Outputting reasoning

- Can learn to predict reasoning steps
  - Give reasoning steps during training
  - Will output reasoning steps to approximate solver



(a) Maze navigation task

Prompt			Response		
bos			bos		
start	0	2	plan	0	2
goal	1	0	plan	0	1
wall	1	2	plan	0	0
wall	2	0	plan	1	0
eos			eos		

(b) Tokenization of a planning task and its solution

A* planning algorithm

Require: Start node n_{start} and goal node n_{goal}.
$\mathcal{S}_{\text{closed}} \leftarrow \{\}$
$\mathcal{S}_{\text{frontier}} \leftarrow \{n_{\text{start}}\}$
while $|\mathcal{S}_{\text{frontier}}| > 0$ **do**
  $n_{\text{curr}} = \arg\min_{n \in \mathcal{S}_{\text{frontier}}} \text{cost}(n)$
  $\mathcal{S}_{\text{closed}} \leftarrow \mathcal{S}_{\text{closed}} \cup \{n_{\text{curr}}\}$
  for $n_{\text{child}} \in \text{children}(n_{\text{curr}})$ **do**
    if $\text{pos}(n) = \text{pos}(n_{\text{child}})$ for any $n \in \mathcal{S}_{\text{closed}} \cup \mathcal{S}_{\text{frontier}}$ **then**
      if $c(n) + h(n) \leq c(n_{\text{child}}) + h(n_{\text{child}})$ **then**
        continue
      end if
    end if
    Set $\text{parent}(n_{\text{child}}) \leftarrow n_{\text{curr}}$
    $\mathcal{S}_{\text{frontier}} \leftarrow \mathcal{S}_{\text{frontier}} \cup \{n_{\text{child}}\}$
  end for
end while
Compute and return plan by recursing on parents of n_{curr}.

Tokenization of algorithm execution

Trace:
bos
create 0 2 c0 c3 — Add node to frontier
close 0 2 c0 c3 — Add node to closed set
create 0 1 c1 c2 — Cost from start $c(n)$
close 0 1 c1 c2 — Heuristic value $h(n)$
create 0 0 c2 c1
create 1 1 c2 c1
close 0 0 c2 c1
create 1 0 c3 c0
close 1 0 c3 c0

Plan:
plan 0 2
plan 0 1
plan 0 0
plan 1 0
eos

(c) A^*'s execution when solving a planning task is logged into an execution trace

Emulating reasoning byproducts

UNIVERSITÉ CÔTE D'AZUR

- What does a LLM do?
  - Model language
  - Complete likely ends of sentences

- What is language?
  - A way to communicate what we think
  - What is reasoning?
    - What we think

- **LLMs emulate a byproduct of reasoning**