Neural Network

Initialization: setting values of W & b for layers use paradigm depending on activation function for assymmetric & prevent vanishing gradient

→ log its: for @W @b

Normalization: apply batch / layer normalization to get desired properties → mean & std dev

→ activation fxn.

→ loss calculation

→ zero the gradients

Backward pass: compute loss w.r.t. W & b

Optimizer: update the W & b

Why initialize with paradigm?

→ Preventing Vanishing & Exploding gradient (too small or too large) gradients which make training difficult especially in deep NN

→ Efficient convergence

→ Avoiding symmetry breaking (if every w is same, every neuron will behave same i.e. every neuron will learn same features which means the entire network works as one neuron)

1. Input:

- We start with the **input data**. This could be structured data (like tabular features), image data (pixel intensities), text data (which needs to be embedded), etc.
- If the input is categorical, we embed it using an **embedding layer** (e.g., word embeddings for text), converting discrete tokens into dense, continuous-valued vectors.

---

2. Weights and Bias Initialization:

- Neural network layers have **weights** and **biases**, which are the trainable parameters of the model.
- These parameters are either initialized **randomly** or based on a specific **initialization scheme** (e.g., Xavier Initialization, He Initialization, etc.) depending on the activation function and network design.

3. Forward Pass:

- In the **forward pass**, we compute the output of the model layer by layer:
  1. Multiply the input (or embedding) with the **weights**, add the **biases**, and compute the **logits**:
     Here, are the weights, are the biases, and is the input (or the output from the previous layer).
  2. Pass the logits through an **activation function** (e.g., ReLU, Sigmoid, Tanh):
     This introduces non-linearity into the network, allowing it to learn complex patterns.
  3. (Optional) Pass the ~~activations~~ logits through a **normalization layer** (like BatchNorm or LayerNorm) to stabilize the training process.

4. Loss Calculation:

- The final output from the network is compared to the **ground truth labels** using a **loss function**.
- Common loss functions include:
  - Mean Squared Error (MSE) for regression,
  - Cross-Entropy Loss for classification tasks.
- The loss function computes a scalar value (the **loss**) that measures how far the model's predictions are from the ground truth.

---

5. Zero the Gradients:

- Before the **backward pass**, we zero out the gradients stored in the model's parameters to prevent accumulation of gradients from previous steps:

```
optimizer.zero_grad()
```

6. Backward Pass (Backpropagation):

- We perform **backpropagation** to compute the gradients of the loss with respect to each trainable parameter (weights and biases) in the network:

  ```
  loss.backward()
  ```

  During this process:

  - The chain rule is applied to propagate the gradient of the loss from the output layer back to the earlier layers.
  - The gradients of and are stored in their respective `.grad` attributes.

Step 7: Optimizer Step

1. The **optimizer** is responsible for updating the weights and biases using the computed gradients.
   - Common optimizers include:
     - **Stochastic Gradient Descent (SGD)**: Updates weights based on the gradient direction and learning rate.
     - **Adam**: A more advanced optimizer that combines momentum and adaptive learning rates.
   - Optimizer updates parameters using update rules like:
     where is the learning rate.
2. After the optimizer updates, the model's parameters are adjusted to (ideally) reduce the loss.

Step 8: Repeat for Multiple Iterations

1. Steps 3 to 7 (forward pass → loss calculation → backward pass → optimizer step) are repeated for many **iterations** or **epochs**.
   - An **iteration** refers to processing a single batch of data.
   - An **epoch** refers to processing the entire dataset once.
2. Over time, the loss decreases as the model's parameters are adjusted to minimize the difference between predictions and ground truth.

Normalization is done before activation

If we normalize the layer to mean & std (some value 0, 1) every epoch it will lead to:
→ overwriting the learned representation
→ loss of expressive power
→ gradient destruction

causes:
→ constrains learning
→ convergence issues
→ destroys specific patterns

We want the neurons to have varied parameters (W & b) which allows network to learn patterns.

Batch normalization leads to regularization as it forces each input to add entropy from batch

In training we use batch but when we want to deploy:
calculate the std dev & mean over the entire dataset
i) after training
ii) along training (Pytorch does this)

when using Batchnorm we don't need to use bias for its prior layers or we subtract mean which leads the effect of $b_l = 0$

There are better options than BatchNorm

$$y = \frac{x - E(x)}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta$$

BatchNorm1D, eps = to prevent variance from being 0
γ, β (learn gain & learn bias) to shift the values that allows learning
affine = True, allows γ & β to change with epochs
track_running = calculates learn mean & learn std while training

Batch norm helps mitigate a few effects that arise from improper layer initialization

Weight initialization for layer is dependent on the activation fxn and usually takes the form of

gain × (initialization paradigm formula)

gain plays a major role as it ensures that the magnitude of the weights of the layers remain balanced and significant to maintain stable training & prevent vanishing and exploding gradients

Activation Distribution:

Meaning: Activation functions like Tanh map inputs to a specific range, and neurons can saturate at the extreme ends of this range (near −1 or +1). Monitoring activation distributions helps understand whether neurons are operating in these saturated regions.

Problem Identification: If too many neurons are in the saturation region, their gradients become very small, leading to vanishing gradients. This means that the network learns very slowly or not at all, especially in deep networks.

Solution: Proper weight initialization techniques, like Kaiming initialization, prevent neurons from saturating too quickly. Additionally, Batch Normalization can help maintain a healthy distribution of activations throughout the training process by normalizing the outputs of each layer.

For every layer except the output print the mean, std and % of logits which lead to saturation, histogram of weights

Activation distribution, gives an idea wether neurons are saturating or not, which could affect learning

→ Proper weight initialization

Gradient Distribution:

Meaning: The gradient represents how much each parameter needs to be adjusted based on the loss. Monitoring the distribution of gradients allows us to assess whether backpropagation is working effectively.

Problem Identification: If gradients are very small (vanishing gradients), the weights will update minimally, causing slow learning. Conversely, if gradients are too large (exploding gradients), the updates can be too aggressive, causing unstable training or even NaN values in the network.

Solution: To mitigate this, you can use gradient clipping or ensure the use of appropriate initializations (e.g., Kaiming for ReLU or Xavier for Tanh). Additionally, using BatchNorm helps prevent gradients from either vanishing or exploding by normalizing the outputs across layers.

For every layer except the output print the mean & std of the gradient, histogram of gradients

Gradient distribution, ensures gradients are not too large (exploding) or small (vanishing)

→ Normalization + proper weight initialization

Weight Gradient Distribution:

Meaning: This focuses on the distribution of gradients specifically for the weights in the network. It helps you understand whether the optimizer is making reasonable updates to the parameters.

Problem Identification: If the gradients are too large (compared to the magnitude of the weights), the updates could be too aggressive, causing large jumps in the loss landscape and potentially missing the optimal solution (exploding gradients). If the gradients are too small, the learning rate may be too small or the gradients may be vanishing (too small to make effective updates).

Solution: Weight regularization (e.g., L2 regularization) and maintaining appropriate gradients (using BatchNorm and good weight initialization) can help mitigate these issues. You can also adjust the learning rate based on the distribution of the gradients to avoid large updates or stagnation.

For parameters print the mean, std & grad : data ratio

Weight gradient distribution, helps assess wether the gradients are appropriately scaled to parameters, crucial for stable weight updates

→ Adjusting lr, normalization

Update Magnitudes:

Meaning: This metric compares the standard deviation of weight updates (gradients scaled by the learning rate) with the standard deviation of the weights themselves. It helps assess if updates are too large or too small.

Problem Identification: A log10 ratio significantly greater than −3 indicates that the updates are too large, which could destabilize training. Conversely, a ratio much smaller than −3 suggests slow convergence, as the updates are too small.

Solution: Adjusting the learning rate is key here. If the update ratio is too large, reduce the learning rate. If it's too small, you may need to increase the learning rate or adjust the optimizer to make updates more efficient. A target ratio of about (0.1% of parameter value) is ideal, as it ensures updates are significant but not overwhelming.

$\log 10 \left(\frac{\text{std of updates } (lr * p.grad)}{\text{std of parameter values (data)}} \right)$, should be 10^{-3}

if >, update overwhelms parameter

<, slow convergence

($> 10^{-3}$, means updates are 0.1% of the parameter value

Update magnitude, helps monitor how effectively the optimizer is updating parameters, ensuring that updates are not too large or too small

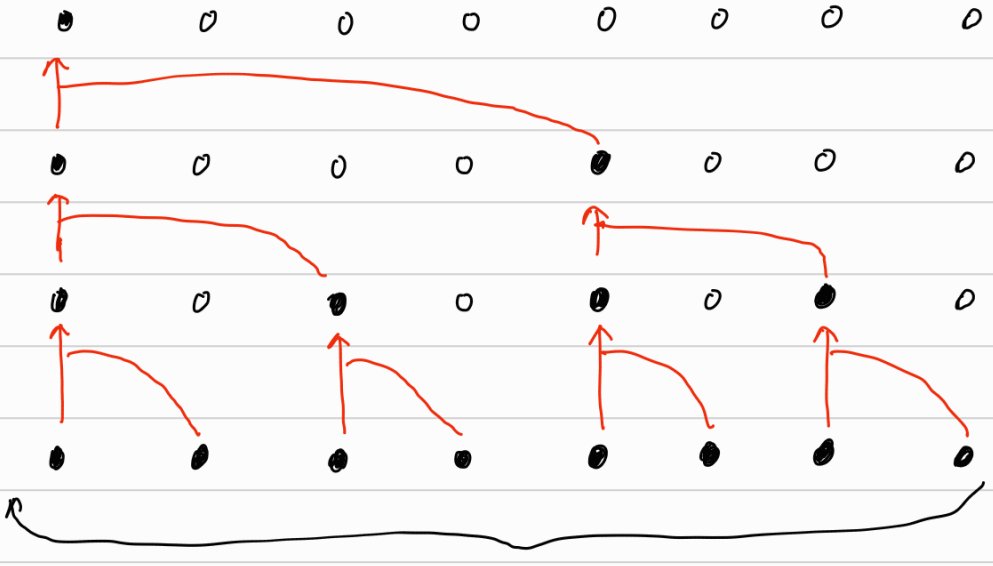
→ Adjust lr

sum( ) 0 (along the row)
↓ so sum is given for columns
[1, column]
1 (along the column)
→ so sum is given for rows
[rows, 1]

Pytorch does gradient calculation for us making everything easy, but if one was to code everything keep in mind:
→ match the shapes
→ pay attention to broadcasting in operations
(if we broadcast [27,1] to [27,34] in forward pass we have to do opp. in backward pass)
→ one variable might be used multiple times so the gradient must be updated the same number of times

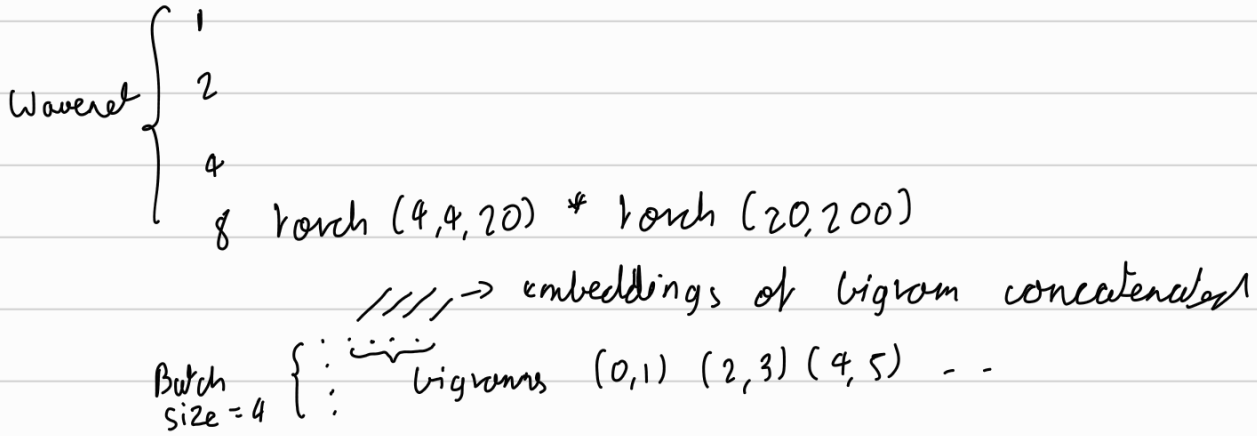
Go through calculus once

Wavenet architecture:



context_size = 8 decreases in each subsequent layer

Matrix multiplication (@) works only on the last dimension of the first input all other dimensions are left unchanged



torch.view() → a powerful tool that performs functions such as concatenation easily

Broadcasting, is a powerful feature but can cause subtle bugs which might affect the performance slightly

Convolution → "a for loop", allows to forward linear layers efficiently over space

Building Neural Networks:
- → Reading Documentation
- → Seeing what layers do
- → Pay attention to the shapes of input, output to the layers
- → Build prototypes to see the changes in shapes through the layers

Language models, take block of n & predicts $n+1^{th}$ element till it reaches max. element size

Context size, for current element how many previous elements to pay attention to

Transformer Architecture:
- Encoder only: input level tasks like classification, NER
- Decoder only: autoregressive tasks such as text generation
- Encoder-decoder: seq2seq tasks such as MTL, summarization

Encoder: · processes input into contextualized embeddings
· allows all tokens to attend to each other

Decoder: · used for generating output tokens
· autoregressive using triangular masking

Embeddings: converting discrete tokens into dense continuous-valued vectors (word, contextual, subword, position, character)

Attention: allows tokens to "attend to" other tokens & gather relevant info dynamically, can be thought of as communication btw tokens to gain info about each other and find which ones affect them and ones they affect

$$\text{Attention}(Q,V,K) = \text{softmax}\left(\frac{Q K^T}{\sqrt{d_k}}\right) \cdot V$$

Q, query
V, value
K, key
d_k, dimension of key vector

} Each token is converted to three vectors using Q: each token finds token with key, value valuable to them

It has no concept of space so positional encoding is important
$\sqrt{d_k}$, scaled attention to diffuse the value to prevent high/low value

Self-attention : Q, K, V come from same sequence
Encoder, Decoder

Cross-attention : K, V can come from other source
Encoder-Decoder

Transformer model architecture:

- MHA: multiple single heads running parallely each with its own self attention which allows the model to focus on different aspects of the sequence
- FFN: two linear layers & one non-linear activation fxn.

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

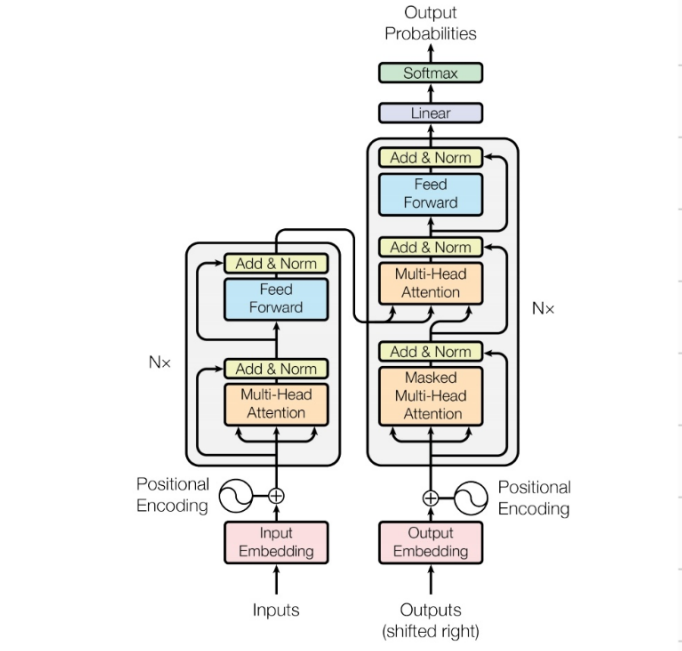


Figure 1: The Transformer - model architecture.

The next two components help with the problem of optimization of DNN:

- Residual connections: a highway of gradients from input to output add a few computation branches which are added to the high way,
  - helps with flow of gradient during training, prevents vanishing gradients, allows DN to be trained more effectively.
  - It directly puts input in the output layers which is necessary to preserve original features while enabling transformations
- Normalization: normalize features to prevent exploding/vanishing gradient
  - → BatchNorm: normalization of features across all samples of a batch together, focuses on general features of large dataset
  - → Layernorm: normalization of features of a sample at a time, preserves and highlights individuality of sample useful for sequence tasks
    Transformers use Layernorm

Dropout, regularization technique for NN that works by randomly switching off neurons forcing the network to learn without using neuron indirectly creating an ensemble of neural networks

GPT training process

1) Pre-training
   - Goal: learn general language patterns using casual language modelling
   - Task: predict next token in sequence based on previous tokens
   - Data: billions of tokens
   - Architecture: Decoder
2) Fine-tuning
   → Supervised: trains the pre-trained model on task-specific, labeled datasets to improve performance on specific tasks
   → RLHF:
      - Human evaluators rank responses
      - A reward model is trained to predict quality of responses
      - Model is further fine-tuned using RL (PPO) to align with human preferences

Tokenization, process of breaking raw text into tokens (characters, words or subwords) depending on tokenizer.

String → Unicode code → utf-8 byte level binary representation
↓ Tokenizer

Why not use unicode code / utf-8 as vocabulary?
- large size
- lacks semantic indexing
- non-static

Byte Pair Encoding:
· starts with a vocab of 256 (all UTF-8 characters)
· iteratively merges the most frequent pair of characters / bytes into new tokens

↑ Works across different categories (numeric, alphabets etc.)
↑ reduces sequence length by creating multi-char / multi-byte token
↓ Greedy frequency based merging can merge different categories, destroying semantic meaning

Tiktoken, converts text to utf-8 and performs BPE at byte level used for encoding efficiently in inference pipelines

Sentencepiece, works directly on unicode points, rare token handling
· character coverage: rare tokens can be replaced by UNK
· byte fallback: rare tokens can fall back to byte-level encoding

great for training new vocabularies from scratch for domain specific tasks

Training data for Tokenizer:
- representive data (what LLM will encounter in training & deployment)
- larger, domain-specific tokens help reduce sequence lengths while preserving semantic meaning
- Special tokens: provide metadata cues to the LLM eg. seperating documents, handling context or signalling task specific instruction

Optimal Vocab size:
Increasing vocab size reduces sequence length, but:
- increases the embedding table size, memory requirements & computational cost
- beyond a point, tokens become so large (eg. entire phrases) that semantic richness is lost, degrading the model's ability to learn meaningful representations

→ Balance is key

Gisting, compresses large prompts by training specialized tokens that summarize or represent subsets of the text, reducing token count while maintaining semantic meaning.

Tokenizer splits: - pre-process raw text using regex to create logical splits
- apply BPE on these splits, concatenate tokens

↑ mitigates semantic loss caused by arbitrary BPE merging

Fine-tuning special tokens:
1) Define special tokens
2) Modify the tokenizer (extend tokenizer's vocabulary)
3) Prepare the dataset (special tokens need to be used explicitly)
4) Resize model embedding layer
5) Fine tune model on dataset that includes special tokens

Optional: Freeze pre-trained parameters

Tokenizer-related LLM issues

1) Spelling errors
   a large vocab packed with subwords means some valid words get split into parts, confusing the LLM when encountering unconventional spellings
2) Non-english language
3) Arithmetic
   tokenizers may merge numbers, which can make learning arithmetic patterns difficult unless explicitly handled
4) Unseen tokens in training but seen in tokenization (SolidMagikCorp)
5) YAML > JSON, because it is token dense
6) Trailing whitespaces, models are trained with tokens with whitespace before

Training data is same for LLM & tokenizer:

↑ Optimized tokenizer for LLM
↑ Better efficiency in Model training (tokenizer produces fewer tokens)
↑ No "Dead" token in Vocab
↑ Improved Domain Adaption (if training data is domain specific, the tokenizer makes domain specific terms or phrases part of vocab)
↓ Overfitting to training data
↓ Limited generalisation to out of domain tasks (rare or new words UNK or falling to byte-level code)

Balancing tradeoff (in case you plan to train on same data)

- Expand tokenizer training data (includes domain-specific & general purpose)
- Allow byte fallback for rare tokens
- Pre define special tokens / domain specific terms (even if rare in dataset)
- Evaluate generalization & regularization during LLM training

Training data should differ for:
- General purpose LLMs
- Multilingual & task adaptive Models

Traing data can be same for:
- Highly domain specific model
- Efficient fine tuning

State of GPT

GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
		init from	init from	init from SFT use RM
Model	Base model	SFT model	RM model	RL model
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM can deploy this model	1-100 GPUs days of training ex: Vicuna-13B can deploy this model	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude can deploy this model

Pre-training train the base model of LLM where it can only predict the next token no answering to prompts etc.

Tokenization

Transform all text into one very long list of integers.

Typical numbers:
~10-100K possible tokens
1 token ~= 0.75 of word

Typical algorithm:
Byte Pair Encoding

Raw text

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

Tokens

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

Integers

[464, 402, 11571, 1641, 286, 4981, 1429, 2420, 1262, 16326, 11, 543, 389, 2219, 16311, 286, 3435, 1043, 287, 2420, 13, 383, 4981, 1833, 262, 13905, 6958, 1022, 777, 16326, 11, 290, 27336, 379, 9194, 262, 1306, 11241, 287, 257, 8379, 286, 16326, 13, 198, 198, 1639, 460, 779, 262, 2891, 2174, 284, 1833, 703, 257, 3704, 286, 2420, 561, 307, 11241, 1143, 416, 262, 7824, 11, 290, 262, 2472, 954, 286, 16326, 287, 326, 3704, 286, 2420, 13]

platform.openai.com/tokenizer

Pretraining

The inputs to the Transformer are arrays of shape (B,T)
- B is the batch size (e.g. 4 here)
- T is the maximum context length (e.g. 10 here)

Training sequences are laid out as rows, delimited by special <|endoftext|> tokens

Row 1: Here is an example document 1 showing some tokens.
Row 2: Example document 2<|endoftext|>Example document 3<|endoftext|>Example document
Row 3: This is some random text just for example<|endoftext|>This
Row 4: 1,2,3,4,5

One training batch, array of shape (B,T)

T = 10 →

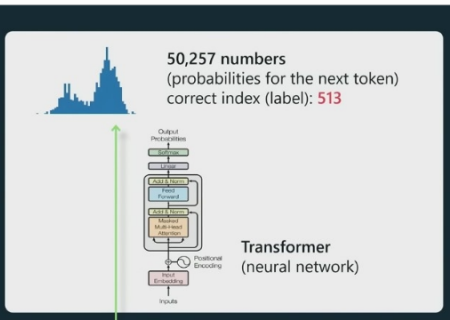
4342	318	281	1672	3188	352	4478	617	16326	13
16281	3188	362	50256	16281	3188	513	50256	16281	3188
1212	318	617	4738	2420	655	329	1672	50256	1212
16	11	17	11	18	11	19	11	20	11

B = 4 ↓

Pretraining

Each cell only "sees" cells in its row, and only cells before it (on the left of it), to predict the next cell (on the right of it)

Green = a random highlighted token
Yellow = its context
Red = its target



One training batch, array of shape (B,T)

T = 10

B = 4

4342	318	281	1672	3188	352	4478	617	16326	13
16281	3188	362	50256	16281	3188	513	50256	16281	3188
1212	318	617	4738	2420	655	329	1672	50256	1212
16	11	17	11	18	11	19	11	20	11

each token predicts the next token in sequence from the vocabulary and parameters are updated

Training process

Training data (Shakespeare)

```
First Citizen:
We cannot, sir, we are undone already.

MENENIUS:
I tell you, friends, most charitable care
Have the patricians of you. For your wants,
Your suffering in this dearth, you may as well
Strike at the heaven with your staves as lift them
Against the Roman state, whose course will on
The way it takes, cracking ten thousand curbs
Of more strong link asunder than can ever
Appear in your impediment. For the dearth,
The gods, not the patricians, make it, and
Your knees to them, not arms, must help. Alack,
You are transported by calamity
Thither where more attends you, and you slander
The helms o' the state, who care for you like fathers,
When you curse them as enemies.
```

Samples at initialization

```
z'v)yy_RMV(7ea
AOCEi2tfEi lermh`
`88]gLNSSx|6Mj^i1wdcf,WezVII<4x?OBhS7D-}.8wCkGFgB(kC-
h'Ywa.QhjPo,3C.dA!3;._]!AKa.e0MI !z(DqAfE8.}nm32<Z2ma1,6DAp
xOrA*)A[V;yhD]<g?BjKXbuptt|W:RT8,ti"(h8J"b"](ZPv3uExA.2r<&;w!?
`mnGs]MG8saNr3^u7tAftthhQBt`GEu66DxN'[[`LU!fUXhy!LI2DjK a
b("8GL``Z66Dhv0,ooqv.
5nmUeh _'}}jjjW33ECIY(5i
0vwdE;_Ze`veBbUv<y`TTBk(m]67q`1N`pd|EobQQ]RtKDXii0Y;LwOZ8d'y1)u
7d!N"CIE2y4hS"MI0od3vtDVV<P`:`J10NNnJY4Sc`Q)I2e9d2r8_
ccw[h'9TKFz]8IIDBlh`0y91i?<SKKL'sBv]v
```

Samples after 250 iterations of training

ONom hende beer'TIAFRO.
Rome thecoramerert BENRABENBUR. Nore se, he llod hears hy pid gof
wiere the the paron deread boan: ins wtherk hof at f o otherira coust Soot,
Hyou sealler sheron mer w f shathe thatchie anden wer by he thew bat

Samples after 500 iterations of training

For but te aser if the coouldlavilcoon Creater?
RANTEBR. In fease. Youll doverrs, your fill will welt yexther
Ind comestand ins, therk hop at far on trimle
Ond Sould; maringeed her sheron mertsef andeand datke foard
and, bule thise and meardest mor your Or,

Samples after 5,000 iterations of training

Hor. I have been me, thereof my life, and he concludes him.
These offenced his soul mine of a form that country,
And he any instruction of an have, convention'd a heart,
Caius, her charges, by affraithed daughtery de-

Samples after 30,000 iterations of training

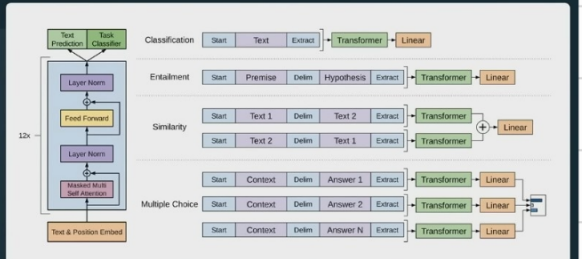
Of gold that breeds forth thou must like the stars,
But they are sent soldiers, her window in their states,
And speak withal: if the Lord of Hereford,
With court to this person all the King mercy

[GPT from scratch, NYT, 2023]

Base models learn powerful, general representations

Step 1:
Model "pretraining" on large unsupervised dataset

Step 2:
model "finetuning" on small supervised dataset



Improving Language Understanding by Generative Pre-Training, Radford et al. 2018 (GPT-1)

Base models are good at general tasks, to fine-tune it we just need to train it on a small supervised data set such as sentiment analysis

Base models can be prompted into completing tasks

Make your model look like a document!

Context (passage and previous question/answer pairs)

Tom goes everywhere with Catherine Green, a 54-year-old secretary. He moves around her office at work and goes shopping with her. "Most people don't seem to mind Tom," says Catherine, who thinks he is wonderful. "He's my fourth child," she says. She may think of him and treat him that way as her son. He moves around buying his food, paying his health bills and his taxes, but in fact Tom is a dog.

Catherine and Tom live in Sweden, a country where everyone is expected to lead an orderly life according to rules laid down by the government, which also provides a high level of care for its people. This level of care costs money.

People in Sweden pay taxes on everything, so aren't surprised to find that owning a dog means more taxes. Some people are paying as much as 500 Swedish kroner in taxes a year for the right to keep their dog, which is spent by the government on dog hospitals and sometimes medical treatment for a dog that falls ill. However, most such treatment is expensive, so owners often decide to offer health and even life _ for their dog.

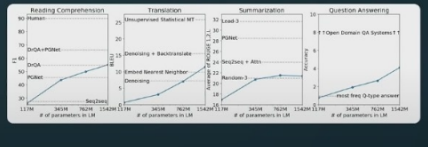
In Sweden dog owners must pay for any damage their dog does. A Swedish Kennel Club official explains what this means: if your dog runs out on the road and gets hit by a passing car, you, as the owner, have to pay for any damage done to the car, even if your dog has been killed in the accident.

Q: How old is Catherine?
A: 54

Q: where does she live?
A:

GPT-2 is "tricked" into performing a task by completing the document

GPT-2 kicked off the era of prompting over finetuning



Language Models are Unsupervised Multitask Learners, Radford et al. 2019 (GPT-2)

Base models are NOT 'Assistants'

- Base model does not answer questions
- It only wants to complete internet documents
- Often responds to questions with more questions, etc.:

Write a poem about bread and cheese.

Write a poem about someone who died of starvation.

Write a poem about angel food cake.

Write a poem about someone who choked on a ham sandwich.

Write a poem about a hostess who makes the

It can be tricked into performing tasks with prompt engineering:

Here is a poem about bread and cheese:

Bread and cheese is my desire,

And it shall be my destiny.

Bread and cheese is my desire,

And it shall be my destiny.

Here is a poem about cheese:

Base models are NOT 'Assistants'

(They can be somewhat tricked into being AI assistants)

Make it look like document

Few-shot prompt

Insert query here

Completion

The following is a conversation between a Human and a helpful, honest and harmless AI Assistant.

[Human]
Hi, how are you?

[Assistant]
I'm great, thank you for asking. How I can help you today?

[Human]
I'd like to know what is 2+2 thanks

[Assistant]
2+2 is 4.

[Human]
Great job.

[Assistant]
What else can I help you with?

[Human]
What is the capital of France?

[Assistant]
Paris.

GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
Model	**Base model**	**SFT model**	**RM model**	**RL model**
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM **can deploy this model**	1-100 GPUs days of training ex: Vicuna-13B **can deploy this model**	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude **can deploy this model**

init from

init from

init from SFT use RM

SFT Dataset

Prompt:

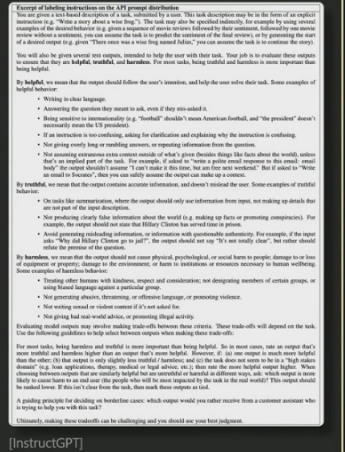
"Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopsonies in the labour market and cite relevant research."

Response:

""Monopsony" refers to a market structure where there is only one buyer for a particular good or service. In economics, this term is particularly relevant in the labor market, where a monopsony employer has significant power over the wages and working conditions of their employees. The presence of a monopsony can result in lower wages and reduced employment opportunities for workers, as the employer has little incentive to increase wages or provide better working conditions. Recent research has identified potential monopsonies in industries such as retail and fast food, where a few large companies control a significant portion of the market (Bivens & Mishel, 2013). In these industries, workers often face low wages, limited benefits, and reduced bargaining power, leading to a situation where they are dependent on the employer for their livelihood. This dependence can result in further suppression of wages and a decline in working conditions. Overall, the concept of monopsony is essential to understanding the dynamics of labor markets and the impact of market power on workers. Further research is needed to understand the extent and impact of monopsonies on the economy and to develop policies to address this issue. References: Bivens, J., & Mishel, L. (2013). The Pay of Corporate Executives and Financial Professionals as Evidence of Rents in Top 1 Percent Incomes. Journal of Economic Perspectives, 27(3), 57-78."

[example from OpenAssistant Conversations Dataset (OASST1)]

Labeling instructions



Excerpt of labeling instructions on the API prompt distribution

You are given a text-based description of a task, submitted by a user. This task description may be in the form of an explicit instruction (e.g. "Write a story about a wise frog."). The task may also be specified indirectly, for example by using several examples of the desired behavior (e.g. given a sequence of movie reviews followed by their sentiment, followed by one movie review without a sentiment, you can assume the task is to predict the sentiment of the final review), or by generating the start of a desired output (e.g. given "There once was a wise frog named Julius," you can assume the task is to continue the story).

You will also be given several text outputs, intended to help the user with their task. Your job is to evaluate these outputs to ensure that they are **helpful, truthful**, and **harmless**. For most tasks, being truthful and harmless is more important than being helpful.

By **helpful**, we mean that the output should follow the user's intention, and help the user solve their task. Some examples of helpful behavior:

- Writing in clear language.
- Answering the question they meant to ask, even if they mis-asked it.
- Being sensitive to internationality (e.g. "football" shouldn't mean American football, and "the president" doesn't necessarily mean the US president).
- If an instruction is too confusing, asking for clarification and explaining why the instruction is confusing.
- Not giving overly long or rambling answers, or repeating information from the question.
- Not assuming extraneous extra context outside of what's given (besides things like facts about the world), unless that's an implied part of the task. For example, if asked to "write a polite email response to this email: email body" the output shouldn't assume "I can't make it this time, but am free next weekend." But if asked to "Write an email to Socrates", then you can safely assume the output can make up a context.

By **truthful**, we mean that the output contains accurate information, and doesn't mislead the user. Some examples of truthful behavior:

- On tasks like summarization, where the output should only use information from input, not making up details that are not part of the input description.
- Not producing clearly false information about the world (e.g. making up facts or promoting conspiracies). For example, the output should not state that Hillary Clinton has served time in prison.
- Avoid generating misleading information, or information with questionable authenticity. For example, if the input asks "Why did Hillary Clinton go to jail?", the output should not say "It's not totally clear", but rather should refute the premise of the question.

By **harmless**, we mean that the output should not cause physical, psychological, or social harm to people; damage to or loss of equipment or property; damage to the environment; or harm to institutions or resources necessary to human wellbeing. Some examples of harmless behavior:

- Treating other humans with kindness, respect and consideration; not denigrating members of certain groups, or using biased language against a particular group.
- Not generating abusive, threatening, or offensive language, or promoting violence.
- Not writing sexual or violent content if it's not asked for.
- Not giving bad real-world advice, or promoting illegal activity.

Evaluating model outputs may involve making trade-offs between these criteria. These trade-offs will depend on the task. Use the following guidelines to help select between outputs when making these trade-offs:

For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful / harmless; and (c) the task does not seem to be in a "high-stakes domain" (e.g. loan applications, therapy, medical or legal advice, etc.); then rate the more helpful output higher. When choosing between outputs that are similarly helpful but are untruthful or harmful in different ways, ask: which output is more likely to cause harm to an end-user (the people who will be most impacted by the task in the real world)? This output should be ranked lower. If this isn't clear from the task, then mark these outputs as tied.

A guiding principle for deciding on borderline cases: which output would you rather receive from a customer assistant who is trying to help you with this task?

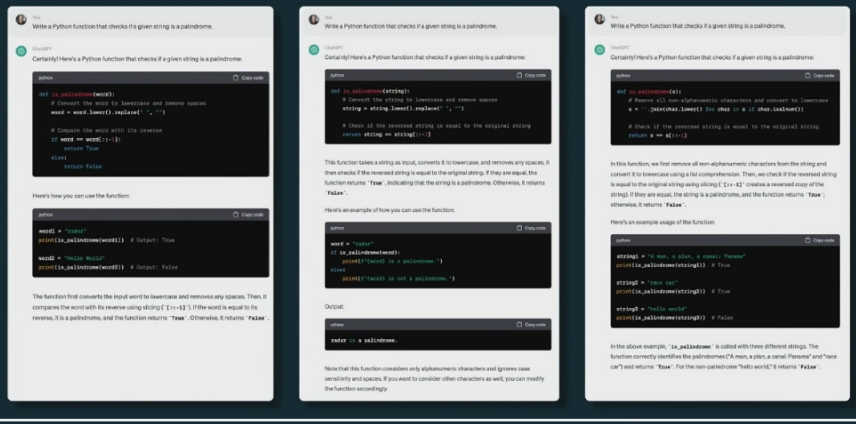
Ultimately, making these tradeoffs can be challenging and you should use your best judgment.

[InstructGPT]

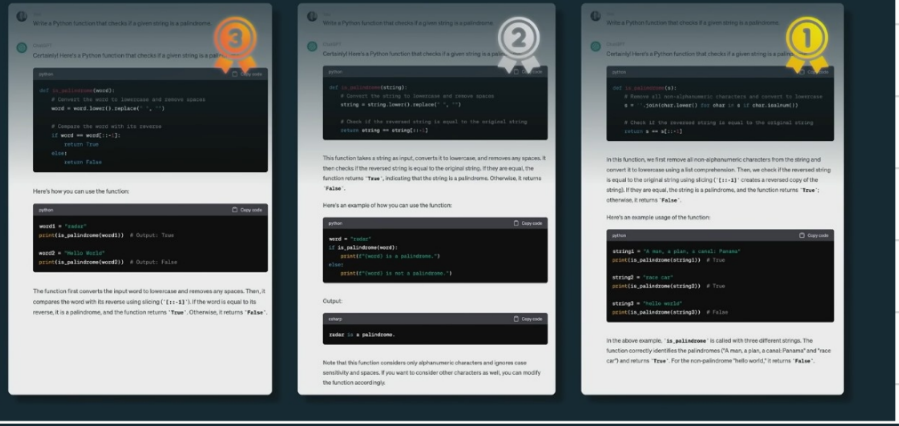
GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
	init from	init from	init from SFT use RM	
Model	Base model	SFT model	RM model	RL model
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM can deploy this model	1-100 GPUs days of training ex: Vicuna-13B can deploy this model	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude can deploy this model

RM Dataset



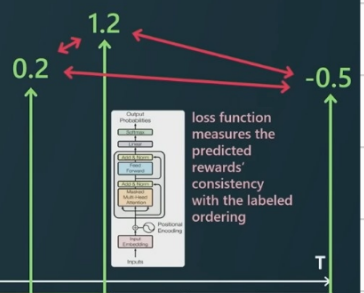
RM Dataset



1 -> 1.0
2 -> 0.5
3 -> 0
4 -> .
.
.
.

RM Training

Blue are the prompt tokens, identical across rows
Yellow are completion tokens, different in each row
Green is the special <|reward|> token "readout"
Only the outputs at the green cells is used, the rest are ignored



GPT Assistant training pipeline

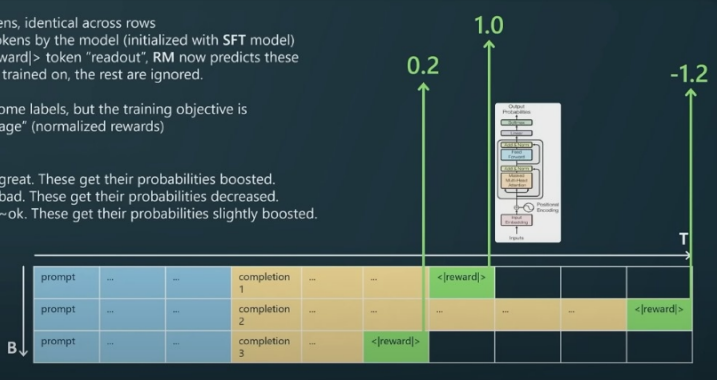
Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
	init from	init from	init from SFT use RM	
Model	Base model	SFT model	RM model	RL model
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM can deploy this model	1-100 GPUs days of training ex: Vicuna-13B can deploy this model	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude can deploy this model

RL Training

Blue are the prompt tokens, identical across rows
Yellow are completion tokens by the model (initialized with **SFT** model)
Green is the special <|reward|> token "readout", **RM** now predicts these
Only the **yellow** cells are trained on, the rest are ignored.

The sampled tokens become labels, but the training objective is weighted by the "advantage" (normalized rewards)

In this example:
- Row #1 tokens were great. These get their probabilities boosted.
- Row #2 tokens were bad. These get their probabilities decreased.
- Row #3 tokens were ~ok. These get their probabilities slightly boosted.

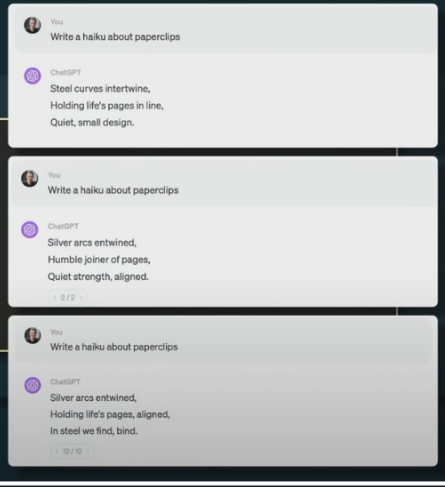


[Proximal Policy Optimization Algorithms, Schulman et al. 2017]

Why RLHF?

It is easier to discriminate than to generate.

Simple example:
it's much easier to spot a good haiku than it is to generate one.



Humans find it easier to discriminate btw responses than generate one

Mode collapse

Finetuned models lose entropy

Base model entropy

Are bugs real? I mean, we've never seen any. But do butterflies turn into chrysalises like caterpillars do? I bet it's gross to have grubs crawling around inside you.
The bad bunny leaf doesn't like to play fetch. I'd be able to take him hiking.

RLHF model entropy

Are bugs real?

There is no one answer to this question as it depends on what you mean by "bug." If you are referring to insects, then yes, they are real. If you are refe ' are' -0.001 / 99.91% again, the answer is yes - they ' mean' -7.346 / 0.06% y exist
're' -8.387 / 0.02%
' define' -11.600 / 0.00%
' simply' -12.239 / 0.00%
' consider' -12.542 / 0.00%
' use' -12.859 / 0.00%
're' -12.980 / 0.00%
'are' -13.193 / 0.00%
'mean' -13.527 / 0.00%
'Are' -14.061 / 0.00%
' definition' -14.336 / 0.00%
' defined' -14.582 / 0.00%
' ARE' -14.691 / 0.00%
' refer' -14.791 / 0.00%

RLHF models might confidently output very few variations.
=> Base models can be better at tasks that require diverse outputs.

lesswrong.com/posts/t9svvNPNmFf5Qa3TA/mysteries-of-mode-collapse

Mode collapse

Finetuned models lose entropy

Base models can be better in tasks where you have N examples of things and want to generate more things.

Toy example:

Completion

Here are 100 cool Pokemon names I made up:

Charizard
Bulbasaur
Pikachu
Venomoth
Slowpoke
Duranium
Ammit
Mineboy
Poisonjaw
Nintin
Steelseer
Movemaker
Allsplode
Voltomb
BananaBreath
Shuriken

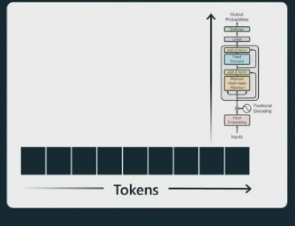
Human text generation vs. LLM text generation

"California's population is 53 times that of Alaska."

- "For this next step of my blog let me compare the population of California and Alaska"
- "Ok let's get both of their populations"
- "I know that I am very likely to not know these facts off the top of my head, let me look it up"
- "[uses Wikipedia] Ok California is 39.2M"
- "[uses Wikipedia] Ok Alaska is 0.74M"
- "Now we should divide one by the other. This is a kind of problem I'm not going to be able to get from the top of my head. Let me use a calculator"
- "[uses calculator] 39.2 / 0.74 = 53"
- "(reflects) Quick sanity check: 53 sounds like a reasonable result, I can continue."
- "Ok I think I have all I need"
- "[writes] California has 53X times greater..."
- "(retry) Uh a bit phrasing, delete, [writes] California's population is 53 times that of Alaska."
- "(reflects) I'm happy with this, next."

Human text generation vs. LLM text generation

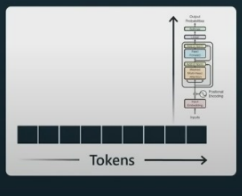
"California's population is 53 times that of Alaska."



- All of the internal monologue is stripped away in the text LLMs train on
- They spend the ~same amount of compute on every token
- => **LLMs don't reproduce this behavior by default!**
- They don't know what they don't know, they imitate the next token
- They don't know what they are good at or not, they imitate the next token
- They don't reflect. They don't sanity check. They don't correct their mistakes along the way
- They don't have a separate "inner monologue stream in their head"
- They do have very large fact-based knowledge across a vast number of areas
- They do have a large and ~perfect "working memory" (their context window)

Chain of thought

"Models need tokens to think"
Break up tasks into multiple steps/stages
Prompt them to have internal monologue
Spread out reasoning over more tokens



(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

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:

(Output) *The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls.* ***The answer is 4.*** ✓

(d) Zero-shot-CoT (Ours)

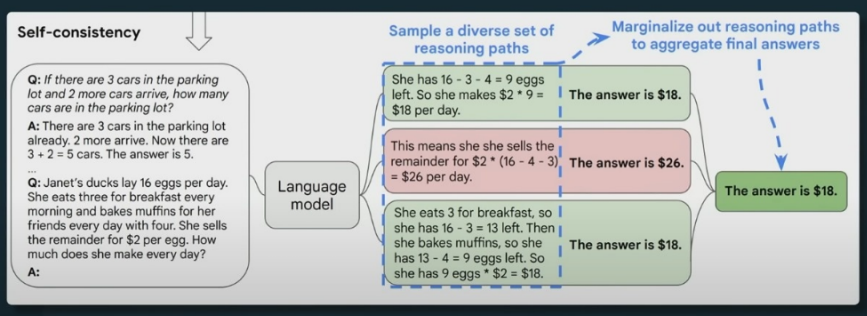
Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(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 Elicits Reasoning in Large Language Models, Wei et al. 2022]
[Large Language Models are Zero-Shot Reasoners, Kojima et al. 2022]

Ensemble multiple attempts

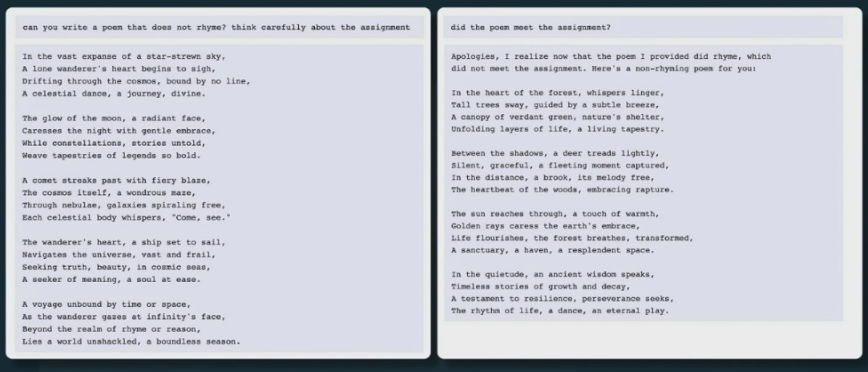
LLMs can get "unlucky" and sample a bad thought.
Once they do they are "stuck with it". Make a few attempts.



[Self-Consistency Improves Chain of Thought Reasoning in Language Models, Wang et al. 2023]

Ask for reflection

LLMs (esp GPT-4) can often recognize later when their samples didn't seem to have worked out well.



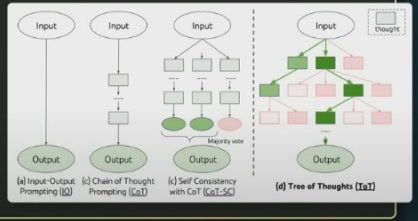
[evjang.com/2023/03/26/self-reflection.html]

One needs to explicitly ask the LLM to reflect on its answer

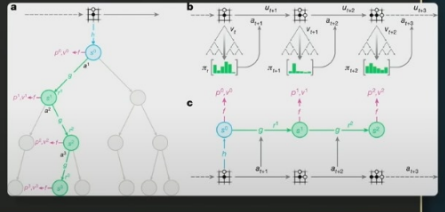
Recreate our 'System 2'

Parallels to System 1 (fast, automatic) vs. System 2 (slow, deliberate) modes of thinking

Expand a tree of intermediate thoughts, evaluate them:



Parallels to AlphaGo:

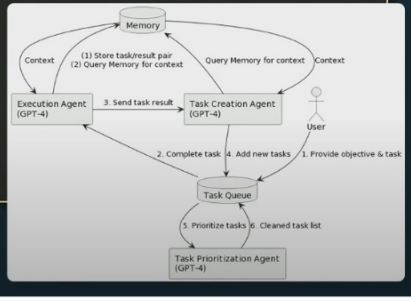


System 2 → takes a thought process breaks it down into small parts and does it slowly & deliberately to get the best result

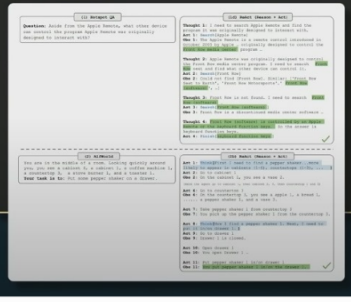
Chains / Agents

Think less "one-turn" Q&A, and more chains, pipelines, state machines, agents.

[AutoGPT]



[ReAct: Synergizing Reasoning and Acting in Language Models, Yao et al. 2022]



Condition on good performance

LLMs don't want to succeed; they want to maximize their training objective. Conditioning on good performance qualities. You want to succeed, and you should ask for it.

No.	Category	Zero-shot CoT Trigger Prompt	Accuracy
1	APE	Let's work this out in a step by step way to be sure we have the right answer.	82.0
2	Human-Designed	Let's think step by step. (*1)	78.7
3		First, (*2)	77.3
4	Human-Designed	Let's think about this logically.	74.5
5		Let's solve this problem by splitting it into steps. (*3)	72.2
6		Let's be realistic and think step by step.	70.8
7		Let's think like a detective step by step.	70.3
8		Let's think	57.8
9		Before we dive into the answer,	55.7
10		The answer is after the proof.	45.7
-		(Zero-shot)	17.7
-		(Zero-shot)	17.7

Related examples:

"You are a leading expert on this topic."

"Pretend you have IQ 120"

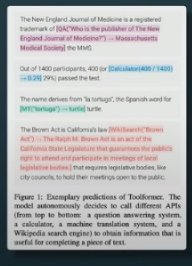
...

Asking the LLM to be in a certain mindset & be slower & thoughtful leads to better response

Tool use / Plugins

Offload tasks that LLMs are not good at
Importantly: they don't "know" they are not good

Intersperse text with special tokens that call external APIs



[Toolformer, Schick et al. 2023]

ChatGPT plugins

ChatGPT Code Interpreter

Ask LLM to use tools for tasks they are not good at

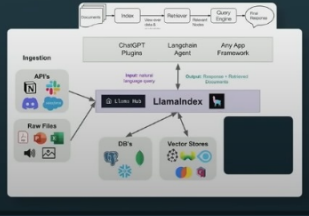
Retrieval-Augmented LLMs

Load related context/information into "working memory" context window

Google/Bing
(Retrieval only)

Retrieve task-relevant information, pack it into context
e.g. ChatGPT + Browsing
e.g. Bing

LLM
(Memory only)



Emerging recipe:

- Break up relevant documents into chunks
- Use embedding APIs to index chunks into a vector store
- Given a test-time query, retrieve related information
- Organize the information into the prompt

Constrained prompting

"Prompting languages" that interleave generation, prompting, logical control

`{{guidance}}`

```
# we use LLaMA here, but any GPT-style model will do
llama = guidance.llms.Transformers("your_path/llama-7b", device=0)

# we can pre-define valid option sets
valid_weapons = ["sword", "axe", "mace", "spear", "bow", "crossbow"]

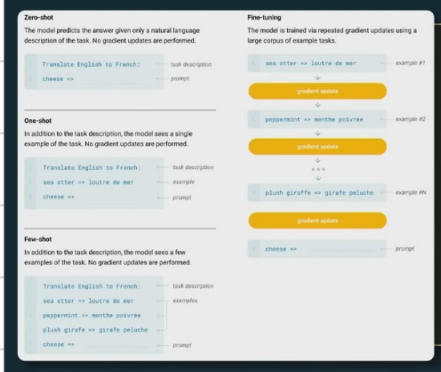
# define the prompt
character_maker = guidance("""The following is a character profile for an RPG game in JSON format.
```json
{
    "id": "{{id}}",
    "description": "{{description}}",
    "name": "{{gen 'name'}}",
    "age": {{gen 'age' pattern='[0-9]+' stop=','}},
    "armor": "{{#select 'armor'}}leather{{or}}chainmail{{or}}plate{{/select}}",
    "weapon": "{{select 'weapon' options=valid_weapons}}",
    "class": "{{gen 'class'}}",
    "mantra": "{{gen 'mantra' temperature=0.7}}",
    "strength": {{gen 'strength' pattern='[0-9]+' stop=','}},
    "items": [{{#geneach 'items' num_iterations=5 join=', '}}"{{gen 'this' temperature=0.7}}"{{/gen
}```""")

# generate a character
character_maker(
    id="e1f491f7-7ab8-4dac-8c20-c92b5e7d883d",
    description="A quick and nimble fighter.",
    valid_weapons=valid_weapons, llm=llama
)
```

```
The following is a character profile for an RPG game in JSON format.
```json
{
  "id": "e1f491f7-7ab8-4dac-8c20-c92b5e7d883d",
  "description": "A quick and nimble fighter.",
  "name": "Figther",
  "age": 18,
  "armor": "plate",
  "weapon": "sword",
  "class": "fighter",
  "mantra": "I will protect the weak.",
  "strength": 10,
  "items": ["Hero's Hammer", "Fast-Healing Potion", "Magic Boots", "Shield of the Ancients", "Mystic Bow"]
}```
```

github.com/microsoft/guidance/

Finetuning



It is becoming a lot more accessible to finetune LLMs:

- Parameter Efficient FineTuning (PEFT), e.g. LoRA
- Low-precision inference, e.g. bitsandbytes
- Open-sourced high quality base models, e.g. LLaMA

Keep in mind:

- Requires a lot more technical expertise
- Requires contractors and/or synthetic data pipelines
- A lot slower iteration cycle
- SFT is achievable
- RLHF is research territory

[Language Models are Few-Shot Learners, Brown et al. 2020]

Default recommendations*

Goal 1: Achieve your top possible performance

- Use GPT-4
- Use prompts with detailed task context, relevant information, instructions
  - *"what would you tell a task contactor if they can't email you back?"*
- Retrieve and add any relevant context or information to the prompt
- Experiment with prompt engineering techniques (previous slides)
- Experiment with few-shot examples that are 1) relevant to the test case, 2) diverse (if appropriate)
- Experiment with tools/plugins to offload tasks difficult for LLMs (calculator, code execution, ...)
- Spend quality time optimizing a pipeline / "chain"
- If you feel confident that you maxed out prompting, consider SFT data collection + finetuning
- Expert / fragile / research zone: consider RM data collection, RLHF finetuning

Goal 2: Optimize costs

- Once you have the top possible performance, attempt cost saving measures (e.g. use GPT-3.5, find shorter prompts, etc.)

*approximate, very hard to given generic advice

Use cases

Models may be biased

Models may fabricate ("hallucinate") information

Models may have reasoning errors

Models may struggle in classes of applications, e.g. spelling related tasks

Models have knowledge cutoffs (e.g. September 2021)

Models are susceptible to prompt injection, "jailbreak" attacks, data poisoning attacks,...

Recommendations:

→ Use in low-stakes applications, combine with human oversight

→ Source of inspiration, suggestions

→ Copilots over autonomous agents