

Part 3

How LLMs Work



MidJourney: "A visualization of what happens inside a neural network"



Three-Part Series Introducing ChatGPT

October 18: How to Interact with ChatGPT

← *Not Technical*

- Introduction to ChatGPT
- Prompt Engineering

October 25: How Smart is ChatGPT?

← *Not Technical*

- Training ChatGPT
- Reasoning, Understanding, and Consciousness

November 1: How LLMs Work

← *Technical*

- How Large Language Models Work



What Does ChatGPT Do?

- It builds a response to your prompt word-by-word.
- First, it generates the first word of the response, then the second word, and so on.
- Each time it generates a word, it considers the prompt you issued, the response so far, and all of the content it was trained on.
- Remember it has been trained on massive amounts of content.
- During training, there is a strong likelihood it encountered quite a bit of content pertaining to what you ask.

The best thing about AI is its ability to

learn	4.5%
predict	3.5%
make	3.2%
understand	3.1%
do	2.9%



Keeping Things Interesting...

- In essence, it determines the word that is most likely to come next [given your prompt, the response so far, and the content it has been trained on]. Well, kind of... it sometimes chooses one of the lower ranked words.
- All of this is performed by a Large Language Model (LLM).

```

I built this little tool to help me understand what it's like to be an autoregressive language model. For
any given passage of text, it augments the original text with highlights and annotations that tell me
how "surprising" each token is to the model, and which other tokens the model thought were most
likely to occur in its place. Right now, the LM I'm using is the smallest version of GPT-2, with 124M
parameters.

For example, if I start counting to ten...

one two three four five six seven eight nine ten
.. you can see that the as the sequence progresses better at predicting the sequence
and by the end it's 100% correct about how to continue.
One striking observation from this visualization is that the model is particularly good at noticing and
catching onto repetition. I'm not the first to I - Transformer language models tend
to learn repeating patterns in language quite quickly. m called "induction". But it's very
obvious in a visualization like this:
The first time I write this sentence, the model is quite confused about what token is about to come
next, especially if I throw in weird words like pumpkin, clown, tweets, alpha, teddy bear.
But the second time? It expects almost everything.

The first time I write this sentence, the model is quite confused about what token is about to come
next, especially if I throw in weird words like pumpkin, clown, tweets, alpha, teddy bear.

This visualization runs entirely in your browser. No data is sent to any servers. Models running with
transformers.js and frontend built with Oak (oaklang.org)

More about induction circuits in transformers: https://transformer-circuits.pub/2022/in-context-
learning-and-induction-heads/index.html
    
```

<https://perplexity.vercel.app/>



What is a Large Language Model?

- There are several LLMs, including GPT-4 from OpenAI, LLaMA from Meta, and PaLM2 from Google.
- Large Language Models (LLMs) are not conventional software with explicit, step-by-step instructions.
- They are neural networks that are trained using billions of words of ordinary language.
- During training, the model learns the statistical relationships between words.
- It doesn't explicitly store grammar rules. Instead it acquires them implicitly during "training."
- It's not programmed to write stories or poems; it learns to do so during "training."





“

Neural language models aren't long programs; you could scroll through the code in a few seconds. They consist mainly of instructions to add and multiply enormous tables of numbers together. These numbers in turn consist of painstakingly learned parameters or “weights”, roughly analogous to the strengths of synapses between neurons in the brain, and “activations”, roughly analogous to the dynamic activity levels of those neurons. Real brains are vastly more complex than these highly simplified model neurons, but perhaps in the same way a bird's wing is vastly more complex than the wing of the Wright brothers' first plane.

”

— Blaise Agüera y Arcas
Google Research

See for yourself: <https://github.com/jadore801120/attention-is-all-you-need-pytorch>



2017 Paper Unlocked “Accuracy” of Systems

- "Attention Is All You Need" by Vaswani et al. (2017)
- **Transformer** architecture
- Takes advantage of parallel execution
- Takes more “words” into context
- Recognizes that the relationship between words is important
- And paying “attention” to important “words” is key
- Has been cited 80,000+ times by other researchers

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com	Noam Shazeer* Google Brain noam@google.com	Niki Parmar* Google Research nikip@google.com	Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com	Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu	Lukasz Kaiser* Google Brain lukaszkaier@google.com	
Illia Polosukhin* ‡ illia.polosukhin@gmail.com			

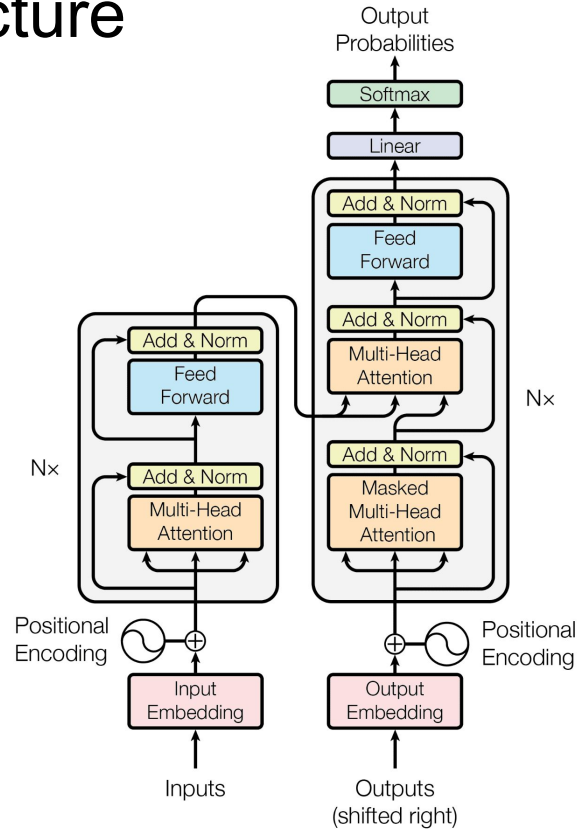
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

<https://doi.org/10.48550/arXiv.1706.03762>



Transformer Architecture

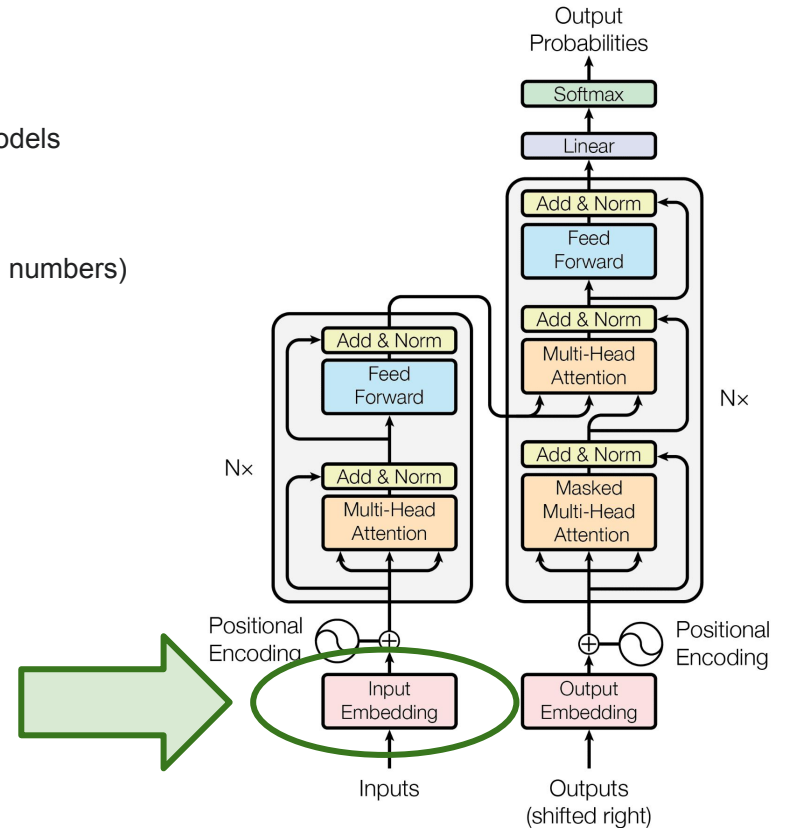


<https://doi.org/10.48550/arXiv.1706.03762>



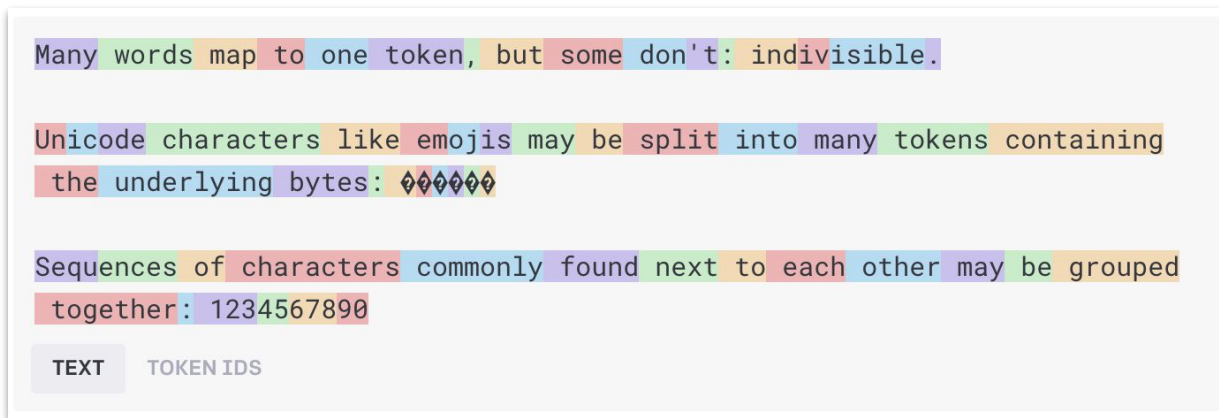
Transformer - Input Embedding

- Transformers are the foundation for these new state-of-the-art NLP models
- Let's first focus on "Input Embedding"
- Where we convert words into something that's easier to work with (i.e. numbers)



LLMs Break Content into Tokens

- Breaks words into **tokens**
- A token can be a character, a symbol, a word, or a part of a word
- Rule of thumb: 75 words → 100 tokens



The screenshot shows a text input field with the text: "Many words map to one token, but some don't: indivisible." The text is displayed with colored highlights under each word, representing individual tokens. Below this, there are two more examples of text with highlights: "Unicode characters like emojis may be split into many tokens containing the underlying bytes: 000000" and "Sequences of characters commonly found next to each other may be grouped together: 1234567890". At the bottom of the interface, there are two buttons: "TEXT" and "TOKEN IDS".

<https://platform.openai.com/tokenizer>



Those Tokens are Handled as Numbers

- Internally, system works with **token IDs** (numerical representation)
- Token IDs in turn refer to an **embedding** (a vector representation of the token)
- Vectors are used because we can put them into neural networks, do “math” on them, and so on
- It so happens that certain types of “math” work particularly well for representing relationships between these tokens

```
[7085, 2456, 3975, 284, 530, 11241, 11, 475, 617, 836, 470, 25, 773, 452,  
12843, 13, 198, 198, 3118, 291, 1098, 3435, 588, 795, 13210, 271, 743,  
307, 6626, 656, 867, 16326, 7268, 262, 10238, 9881, 25, 12520, 97, 248,  
8582, 237, 122, 198, 198, 44015, 3007, 286, 3435, 8811, 1043, 1306, 284,  
1123, 584, 743, 307, 32824, 1978, 25, 17031, 2231, 30924, 3829]
```

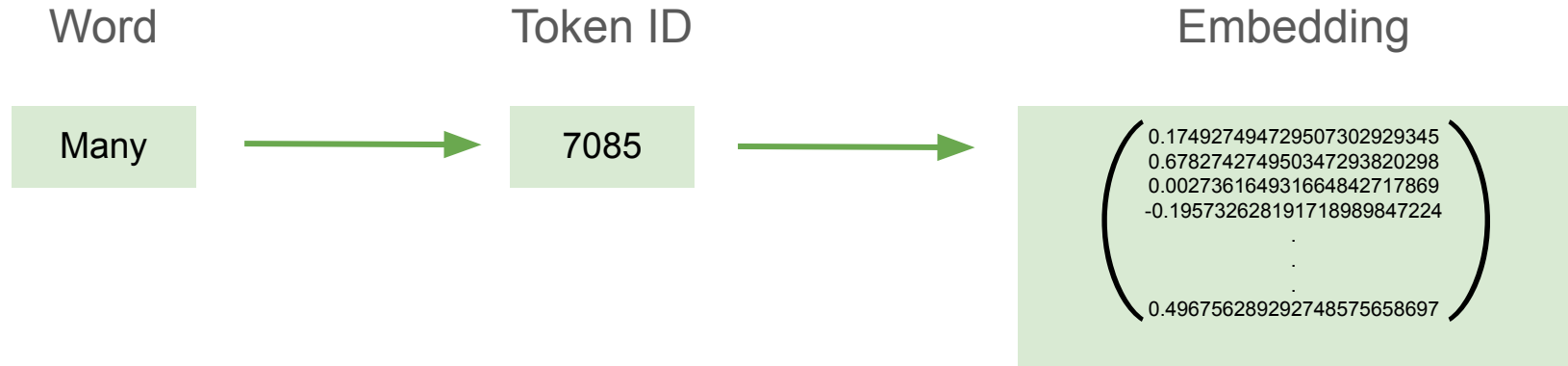
TEXT

TOKEN IDS

<https://platform.openai.com/tokenizer>



How Words are “Handled”



Assuming the word is represented by a single token!



A Sample of Values from one such Embedding

Vector Representation of “Cat”

0.007398007903248072, 0.0029612560756504536, -0.01048285979775745, -0.01476181158542633, 0.07646718621253967, -0.0011427050922065973, 0.026497453451156616, 0.01059535983948997, 0.0190864410251379, 0.0038335588760674, -0.0468108132481575, -0.021150866523385048, 0.009098375216126442, 0.0030140099115669727, -0.05626726150512695, -0.039609555150083834, -0.09978967905404556, -0.07956799119710922, 0.057768501341342926, -0.017375102266669273, 0.015590683557093143, -0.022376490756869316, 0.10152265429496765, -0.05138462409377908, 0.025884613394737244, 0.07069036364555359, 0.0009145145886577666, -0.06275367736816406, 0.03610750287771225, 0.050807688385248184, -0.06453944742679596, -0.0434986837208271, -0.1264101266869062, -0.0003191891883034259, 0.04311852902173996, -0.14792846143245697, -0.019480768591165543, 0.01992032676935196, 0.011479354463517666, 0.02979433164000511, 0.06154156103730202, -0.04609882831573486, -0.053286727517843246, -0.016268745046735413, 0.03660176321864128, -0.07168425619602203, 0.0549746400265694, -0.1446477174758911, 0.0931687275705338, -0.1291029026339853, 0.030971739441156387, 0.0367519038319588, 0.1340744756240845, -0.02852762119819107, -0.10431244939716339, 0.03328850120306015, 0.1295083463191986, 0.0412190817296505, 0.03605308011174202, 0.0599723681807518, 0.025970442220568657, -0.03521350771188736, -0.015058198012411594, 0.005818498786538339, 0.013812823221087456, 0.015064566396176815, 0.022925062105059624, 0.007009583059698343, -0.02910810336470604, 0.1011449321204454, 0.13727356493473053, 0.022466043010354042, -0.07582768052816391, -0.04469817131757736, -0.06026916950941086, 0.04192522168159485, 0.1612275242805481, 0.014356226660311222, -0.0647699236869812, -0.14182332158088684, 0.07568981498479843, 0.002798931673169136, 0.01240639254289913, -0.09695082157850266, -0.0014245212078094482, -0.018527435138821602, 0.009911706671118736, 0.013058848679065704, 0.048697732388973236, 0.017661960795521736, 0.036917399858795064, 0.005680330563336611, 0.024947546422481537, 8.4192593931220476-05, -0.002204198157414794, -0.007295176852494478, 0.008355203084647655, -0.015072236768901348, -0.0032011312432587147, 0.05327794361114502, 0.020942343398928642, -0.019445667043328285, -0.15129604935646057, 0.0337672121822834, 0.0019582237239991913, -0.0014046517899259925, -0.05954226478934288, -0.08176489174365997, 0.024112699553370476, -0.1015794649720192, 0.05419696122047913, 0.13000570237636566, -0.05806615684509277, 0.00418064047884798, 0.0180498044192791, 0.01923936977982521, -0.041859131306409836, 0.01098426602780819, 0.025394367054104805, -0.03678150847554207, 0.03255629166960716, -0.00087823364025544, -0.0710140562655112, 0.02409918524563125, -0.035895368167858124, 0.0047763800248503685, -0.01754925213754177, -0.0029735821299254894, 0.030521586537361145, 0.042433049529799088, 0.05969628319144249, -0.07855783402919769, 0.07639002054929733, -0.004820443224161863, 0.065130800087738, 0.13445857167243958, -0.0660976174935303, 0.01714201085269451, 0.019574925303459167, -0.00021718056814279407, 0.0755931958561752, 0.05964002385735512, -0.0715465098619461, 0.04068697988986969, -0.09640928357839584, -0.07235930114984512, -0.05935797095298767, 0.009602724574506283, -0.056495692589384, 0.002564596945572624, -0.05413592606782913, -0.017797887325286865, 0.0575546547705002, 0.08609342575073242, 0.050908517092466354, -0.05604008585214615, -0.005856652744114399, 0.02329830639064312, 0.08168350160121918, -0.071861155331348, -0.027544429397797546, -0.08970167487859726, 0.024058541283011436, -0.02770240046683927, -0.02533974122458483, 0.010991193588483334, 0.02215300314128399, -0.0282967664099096, -0.07363404333591401, 0.055630138994609922, 0.000292848508138093, -0.0597328083889804, -0.04813411086797774, -0.002152945148272676, 0.004276854917407036, 0.04970101038837433, 0.02516869269311428, -0.001592508002281, 0.0767771303653717, -0.0823667943767306, 0.019983036443591118, -0.0518303290094986, 0.05824366584020204, 0.047829821705818176, -0.1360556322803497, 0.02234281599521637, -0.03254450857639313, 0.011368651874363422, -0.0513596867990494, -0.00048283161595463753, -0.06719424575567245, -0.018972834572196007, 0.025254448875784874, -0.0385899139802742, 0.03636444360017765, -0.025158191099762917, 0.030907975509762764, -0.08114158362150192, 0.09369450062513351, 0.09405472874644141, 0.012534121051430702, -0.0014180991902914, 0.0552687831223011, 0.07056140154600143, 0.0662888100385666, 0.06548195332288742, 0.0158022987852955, -0.03811083108370680444, -0.003248460823600296, 0.01015767455101310, 0.05805244743824, 0.010575438849627972, 0.06210837885737419, -0.0071502267383039, -0.0295573239253044, 0.0289757263518095, 0.002539787907153368, -0.07370117423276901, 0.026873936876654625, 0.0277083627896196, 0.02373671904206276, 0.04336617887200111, 0.03797426636981964, 0.0661377692967653275, 0.050208967179059598, -0.02423020827490001, 0.037851363420486645, 0.18769624829292297, 0.1059433968750763, -0.05118405446410179, 0.06405289471149445, -0.047474540770053364, 0.04021701216697693, 0.048911526799201965, 0.041514985263347626, 0.0605742703988803759, 0.003405822208152213, 0.01214022096246481, -0.037784647196531296, 0.00894617382436957, -0.03059233525419235, 0.039058126509189606, 0.02660788968205452, 0.05596623942255974, 0.03365514427423477, 0.09071408082816696, 0.034562114626169205, 0.08310434222213109, 0.0344182258839131, 0.003703191876411438, 0.002236866159364581, -0.00642943149085060, 0.06852643936872482, 0.09876436740159988, 0.01411499921232462, -0.078606262043904635, 0.0260335183888871, -0.1592547744512558, -0.01012679934501648, -0.10094276070594788, 0.01604175567626953, 0.00635749939829111, 0.0271235904097557, 0.0199843656999028, -0.004958001267027855, 0.0257257112513780594, 0.077529828837296, -0.01912834122776985, -0.10472754448350906, -0.032735679296393, -0.11220421701368332, 0.03347017243504524, -0.04368103668093681, -0.00044717983109876513, -0.029803894460201263, 0.06125372939489456, 0.039308369159698486, -0.05544960162807846, 0.07417158037424088, -0.02231053191098976, -0.1176527461051941, -0.019754905253648758, 0.031432103365659714, 0.03378641605371797, 0.07527634518146515, -0.04749307036399841, 0.005324371624737978, -0.08255213499069214, -0.010222465731203556, 0.021690042689442635, -0.1339070200920105, 0.007615163456648588, -0.0929502621293068, 0.0597759224474301, 0.0001563437378641934

This is for a 300-dimensional vector (not a 12,288-dimensional vector).

These numbers capture the ‘meaning’ of the word.

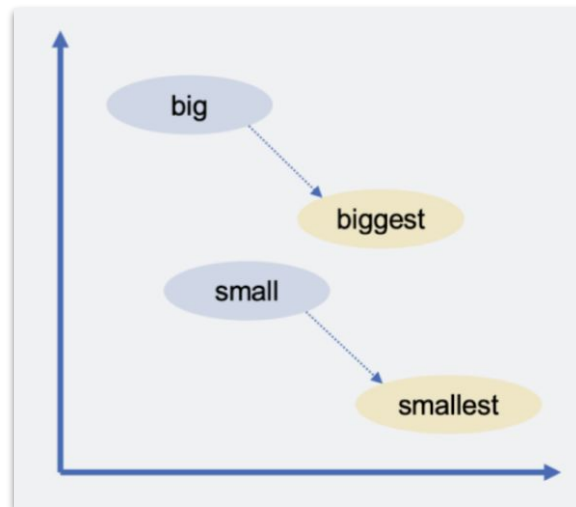
More about the actual numbers in the vector a little later.

http://vectors.nlp.ee/explore/embeddings/en/MOD_enwiki_upos_skipgram_300_2_2021/microcomputer_NOUN/



Use Math to Identify Relationships Between Words

- We can “reason” about words using vector arithmetic.
 - Take the vector for "big" and subtract "biggest"
 - Subtract the result from "small"
 - The word closest to the resulting vector is "smallest"
- We can use vector arithmetic to draw analogies.
 - “Swiss” is to “Switzerland” as “Cambodian” is to “Cambodia”
- Note: because vectors come from human language, they reflect certain biases.
 - “Doctor” minus “man” plus “woman” yields "nurse"



We're actually dealing with a 12,288-dimensional space.

This simple 2-dimensional representation is just to make the concept easier to grasp.

https://arstechnica.com/science/2023/07/a-jargon-free-explanation-of-how-ai-large-language-models-work/?utm_source=substack&utm_medium=email



To See for Yourself...

Semantic Calculator

Calculate ratios, such as «find a word D related to the word C in the same way as the word A is related to the word B». An example is given in the placeholder: which word is in the same relation to the word «father» as «daughter» is to «mother»? The answer is «son». [More on this...](#)

mother_NOUN



daughter_NOUN

father_NOUN



???

Word frequency

High Medium Low

English Wikipedia

1. **son** NOUN 0.88
2. **grandson** NOUN 0.80
3. **nephew** NOUN 0.77
4. **granddaughter** NOUN 0.76
5. **grandfather** NOUN 0.73



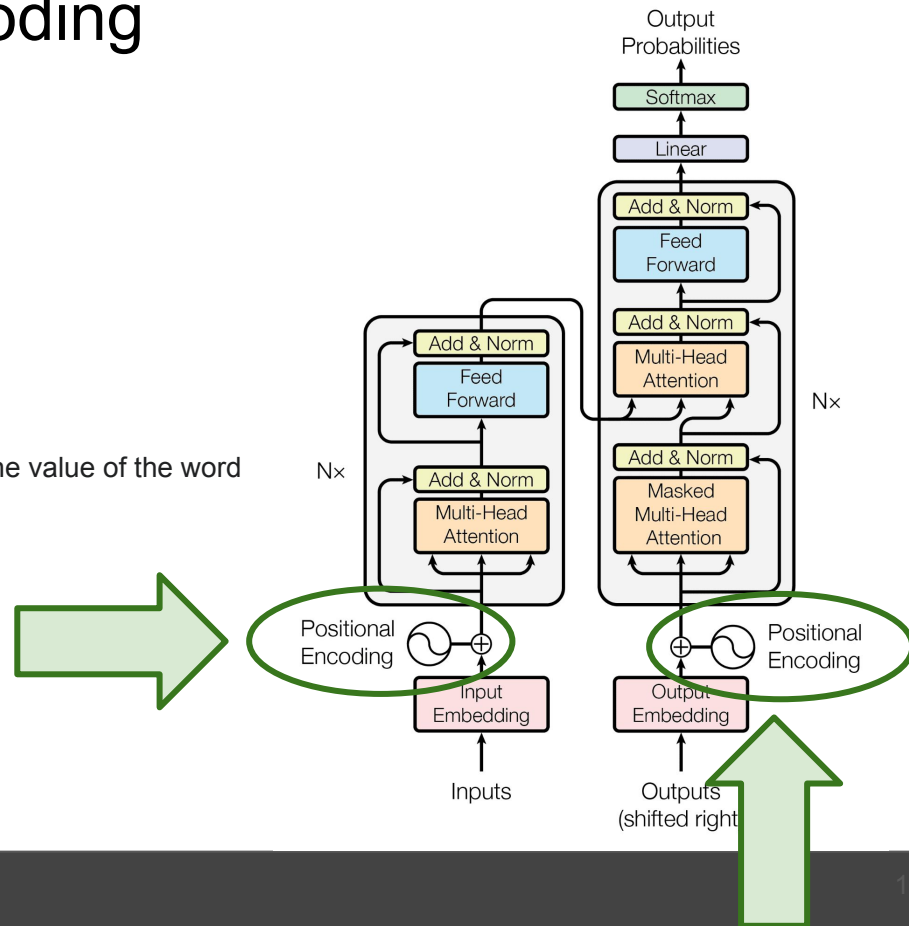
*Again, this tool is for a 300-dimensional vector
(not a 12,288-dimensional vector).*

<http://vectors.npl.eu/explore/embeddings/en/calculator/#>

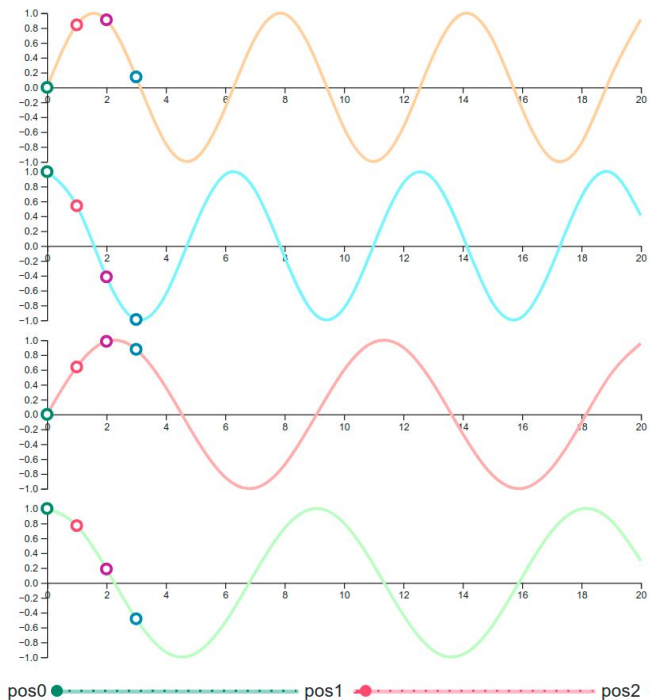


Transformer - Positional Encoding

- The Transformer architecture operates on words in parallel
- We need a way to capture word position
- A way that is incorporated into the vector embedding
- Both absolute position and relative position
- Use a periodically varying function
- Use a function that doesn't make a huge change meaning to the value of the word



Positional Encodings



p0	p1	p2	p3	
0.000	0.841	0.909	0.141	i=0
1.000	0.540	-0.416	-0.990	i=1
0.000	0.638	0.983	0.875	i=2
1.000	0.770	0.186	-0.484	i=3

Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Settings: d = 50

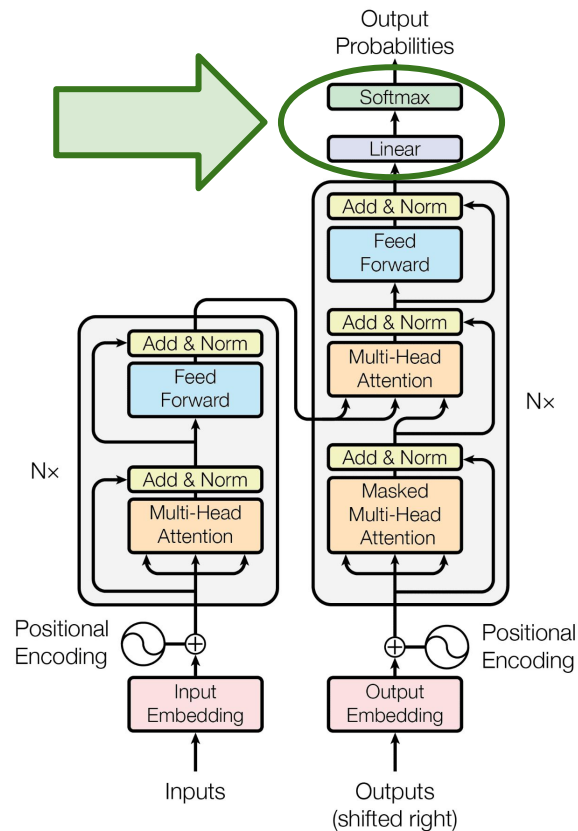
The value of each positional encoding depends on the *position (pos)* and *dimension (d)*. We calculate result for every *index (i)* to get the whole vector.

<https://towardsdatascience.com/understanding-positional-encoding-in-transformers-dc6bafc021ab>



Transformer - Output

- The transformer outputs a vector
- The Linear layer projects the output into a much larger vector
- For ChatGPT, this vector has ~50,000 scores, one for each word in its vocabulary
- Softmax then turns those scores into probabilities (all positive, all add up to 1.0)
- In other words, the output is a vector indicating a probability score of each possible next word
- One of the higher probability words is chosen
- (I'm over simplifying here)



Summary Thus Far...

- We supply an input sequence:

`The sky is`

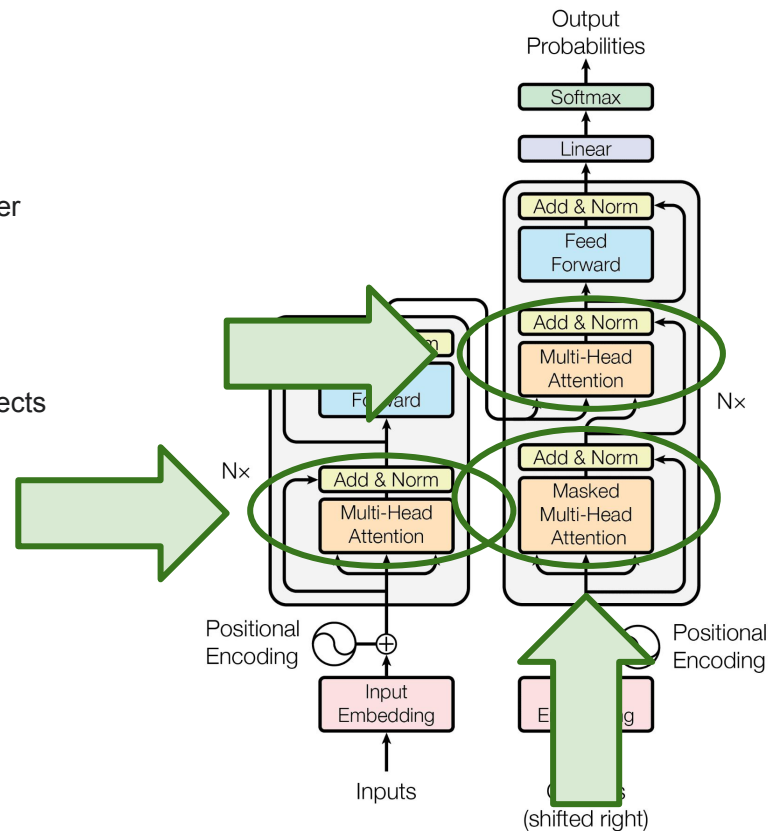
- This input sequence is:
 - Translated into tokens
 - And then token IDs
 - And finally embedding vectors
- For every word, the transformer outputs a vector indicating the probability that a particular word should appear next
- The system chooses one of the higher probability words:

`The sky is blue`



Transformer - Attention

- We have the “words” represented as embedding vectors
- Now we learn more about the words and how they relate to one another
- There are 96 “attention heads” in each layer
- The attention heads operate in parallel
- Each “attention head” focuses on different syntactic and semantic aspects of the content



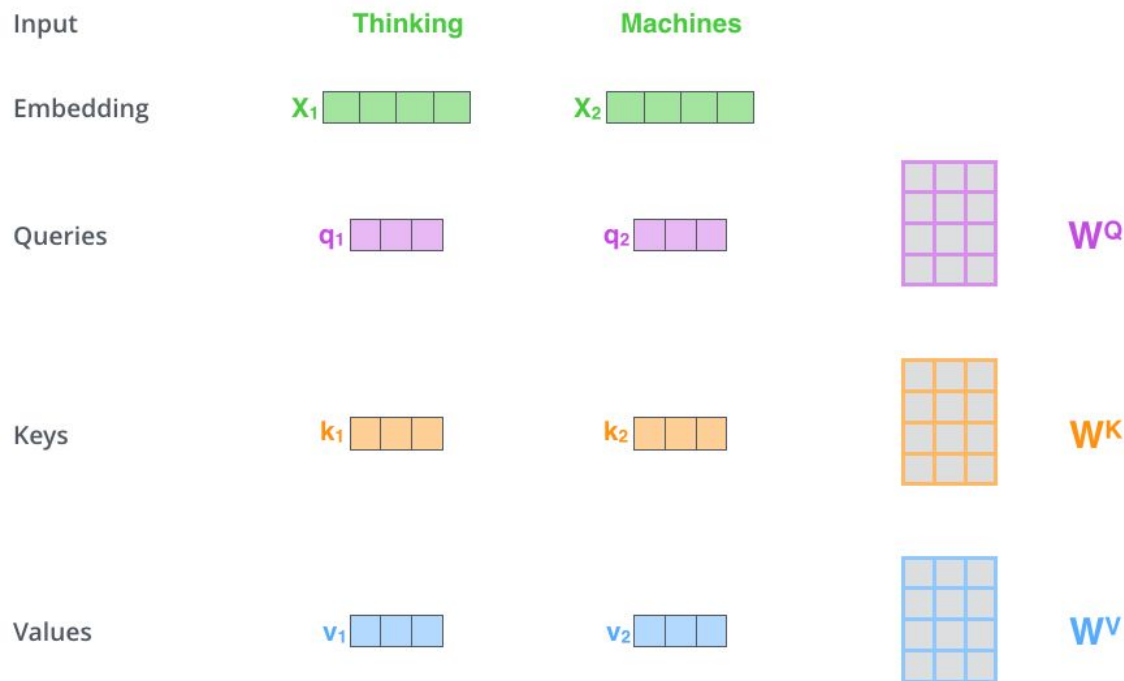
Attention Step 1: Calculate Query, Key, and Value Vectors

For each word, create:

- **Query Vector** (understand input)
- **Key Vector** (“attending” words)
- **Value Vector** (relevance to prediction)

There’s one set of Q, K, and V vectors for each attention head.

These vectors are created by multiplying the embedding by three matrices that we trained during the training process.



<https://jalammar.github.io/illustrated-transformer/>



Attention Step 2: Calculate the Score

For each word, calculate calculate the **Score** for the other words in the input sequence.

The Score is the dot product of the Query Vector for the current word and the Key Vector for the other word.

Input

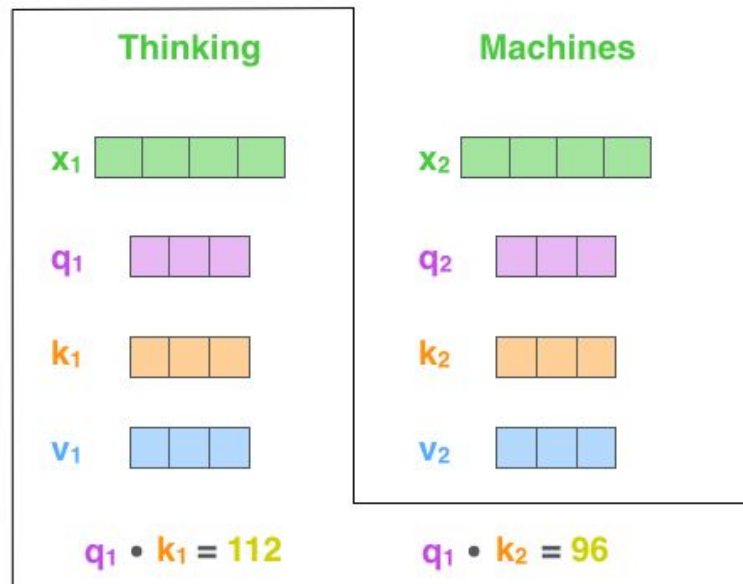
Embedding

Queries

Keys

Values

Score



Attention Steps 3 & 4: Normalize the Scores

Divide Scores by the square root of the dimension of the Key Vectors. This leads to having more stable gradients.

Pass the result through a **Softmax** operation, which normalizes the scores so they're positive and add up to 1.

Input

Embedding

Queries

Keys

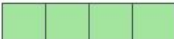
Values

Score

Divide by 8 ($\sqrt{d_k}$)

Softmax

Thinking

x_1 

q_1 

k_1 

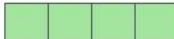
v_1 

$q_1 \cdot k_1 = 112$

14

0.88

Machines

x_2 

q_2 

k_2 

v_2 

$q_1 \cdot k_2 = 96$

12

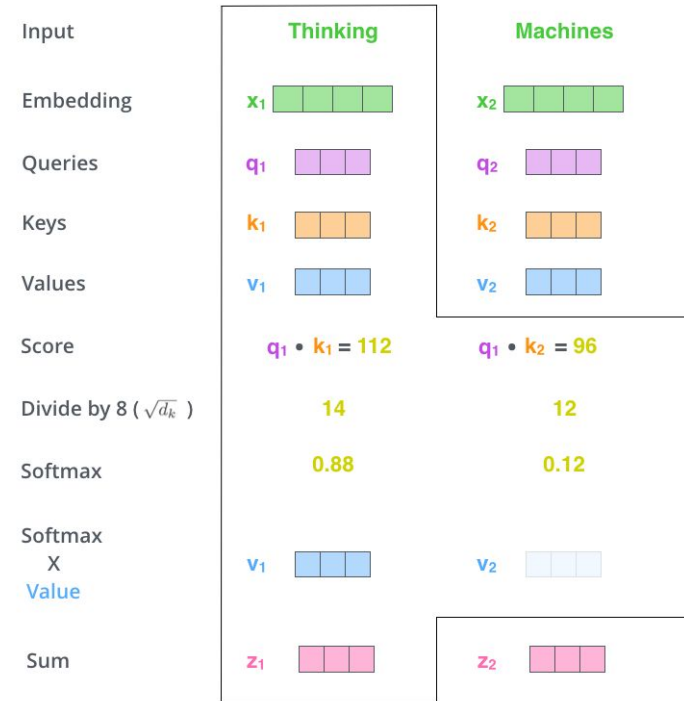
0.12



Attention Steps 5 & 6: Weight and Sum the Value Vectors

To weight the Value Vectors multiply them by the Softmax score. This drowns-out irrelevant words (i.e. multiply them by numbers like 0.001).

Sum the weighted Value Vectors to produce the output of the self-attention layer at this position (for the first word).



<https://jalammar.github.io/illustrated-transformer/>



Attention Step 7: Concatenate and Weight Head Outputs

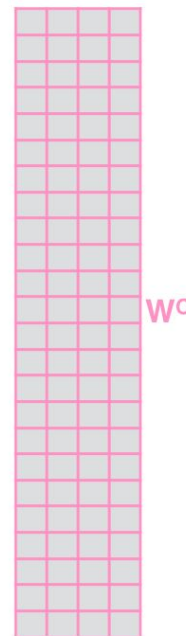
Feed-forward expects a single matrix (a vector for each word). Not a matrix for each attention head. So we condense them into a single matrix.

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^o that was trained jointly with the model

x



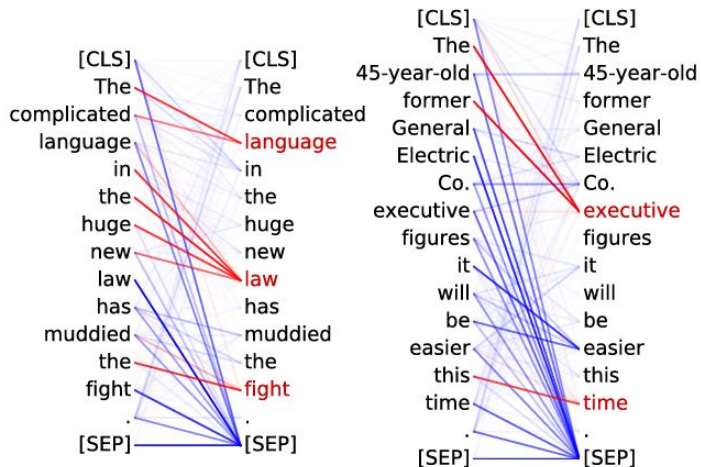
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN



Example: Attention with 12 Heads and 12 Layers

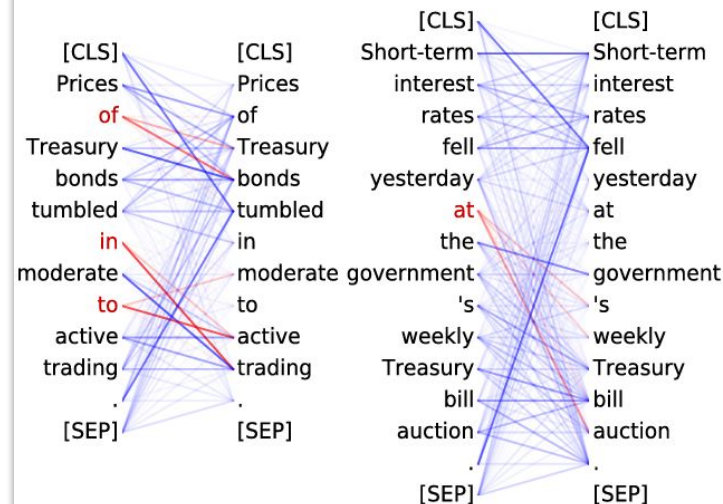
Head 8-11

- **Noun modifiers** (e.g., determiners) attend to their noun



Head 9-6

- **Prepositions** attend to their objects

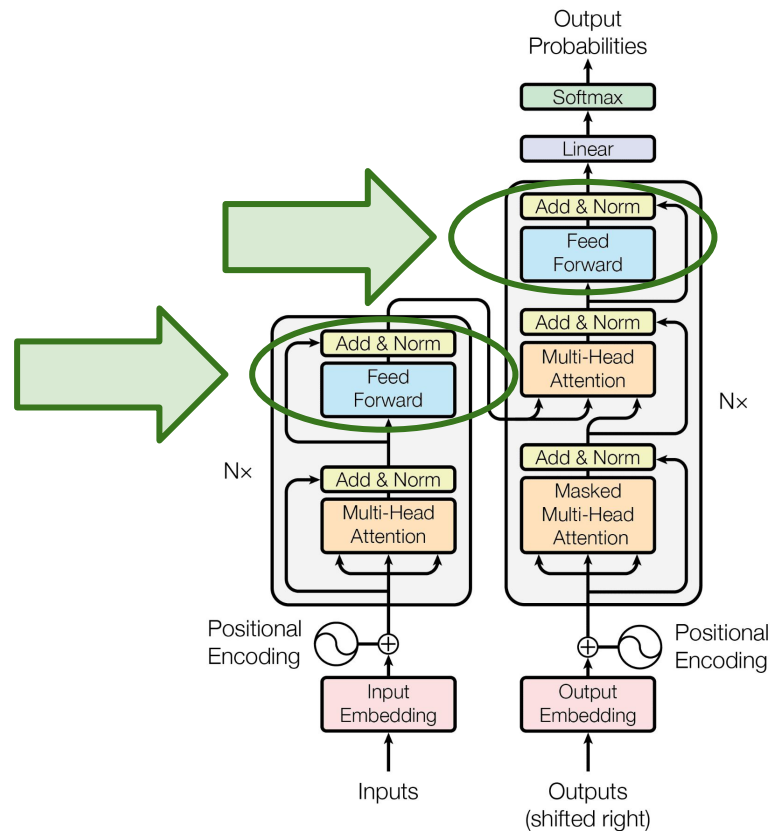


<https://arxiv.org/abs/1906.04341> What Does BERT Look At? An Analysis of BERT's Attention



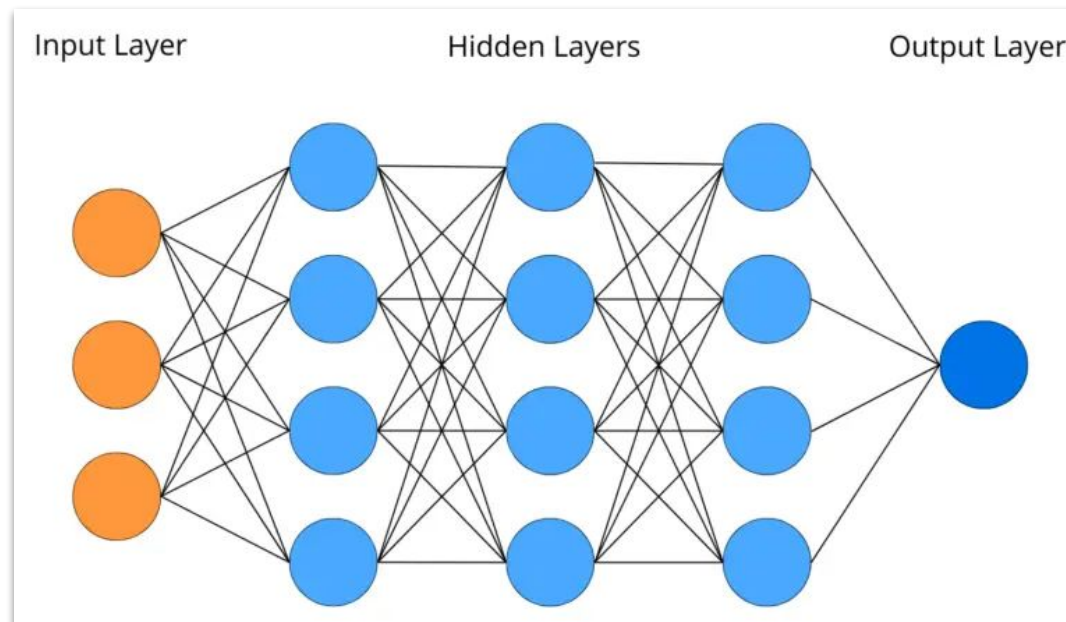
Transformer - Feed Forward

- Feed Forward is a neural network
- During training, Feed Forward “stores” the knowledge learnt
- During inference, Feed Forward helps predict the next word



Feed Forward Neural Network

- We understand **what neural networks do**.
- But we don't fully understand how they work.
- We've **empirically found they do well with certain types of challenges**.
- They are good at generalizing learnings from training sets.
- Given their vast size, explicitly tracing each step is impossible.
- We haven't figured out the "laws" to allow us to fully understand their operation.

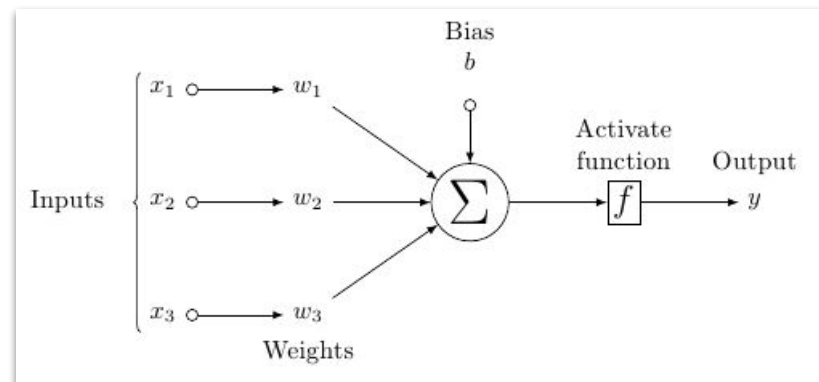


Neural Networks 101

- Assume a neuron has inputs $x = \{x_1, x_2 \dots\}$
- Each connection between neurons has a weight (w_1, w_2, \dots)
- The weights are the “output” from training the large language model
- ChatGPT 3.5 has ~105 billion Feed Forward weights
- There is a formula to calculate the value of a neuron:

$$f[\mathbf{w} \cdot \mathbf{x} + \mathbf{b}]$$

- This processes is repeated through several layers until we get to the output.



<https://www.lesswrong.com/posts/3duR8CrvchHywrnhLo/how-does-gpt-3-spend-its-175b-parameters>



Key Vectors are “Pattern Detectors”

Key	Pattern	Example trigger prefixes
k_{449}^1	Ends with “ <i>substitutes</i> ” (shallow)	<i>At the meeting, Elton said that “for artistic reasons there could be no substitutes In German service, they were used as substitutes Two weeks later, he came off the substitutes</i>
k_{2546}^6	Military, ends with “ <i>base</i> ”/“ <i>bases</i> ” (shallow + semantic)	<i>On 1 April the SRSg authorised the SADF to leave their bases Aircraft from all four carriers attacked the Australian base Bombers flying missions to Rabaul and other Japanese bases</i>
k_{2997}^{10}	a “part of” relation (semantic)	<i>In June 2012 she was named as one of the team that competed He was also a part of the Indian delegation Toy Story is also among the top ten in the BFI list of the 50 films you should</i>
k_{2989}^{13}	Ends with a time range (semantic)	<i>Worldwide, most tornadoes occur in the late afternoon, between 3 pm and 7 Weekend tolls are in effect from 7:00 pm Friday until The building is open to the public seven days a week, from 11:00 am to</i>
k_{1935}^{16}	TV shows (semantic)	<i>Time shifting viewing added 57 percent to the episode’s The first season set that the episode was included in was as part of the From the original NBC daytime version , archived</i>

<https://arxiv.org/pdf/2012.14913.pdf> Transformer Feed-Forward Layers Are Key-Value Memories



Value Vectors Help Predict the Next Word

Value	Prediction	Precision@50	Trigger example
\mathbf{v}_{222}^{15}	<i>each</i>	68%	<i>But when bees and wasps resemble each</i>
\mathbf{v}_{752}^{16}	<i>played</i>	16%	<i>Her first role was in Vijay Lalwani's psychological thriller Karthik Calling Karthik, where Padukone was cast as the supportive girlfriend of a depressed man (played)</i>
\mathbf{v}_{2601}^{13}	<i>extratropical</i>	4%	<i>Most of the winter precipitation is the result of synoptic scale, low pressure weather systems (large scale storms such as extratropical)</i>
\mathbf{v}_{881}^{15}	<i>part</i>	92%	<i>Comet served only briefly with the fleet, owing in large part</i>
\mathbf{v}_{2070}^{16}	<i>line</i>	84%	<i>Sailing from Lorient in October 1805 with one ship of the line</i>
\mathbf{v}_{3186}^{12}	<i>jail</i>	4%	<i>On May 11, 2011, four days after scoring 6 touchdowns for the Slaughter, Grady was sentenced to twenty days in jail</i>

<https://arxiv.org/pdf/2012.14913.pdf> Transformer Feed-Forward Layers Are Key-Value Memories



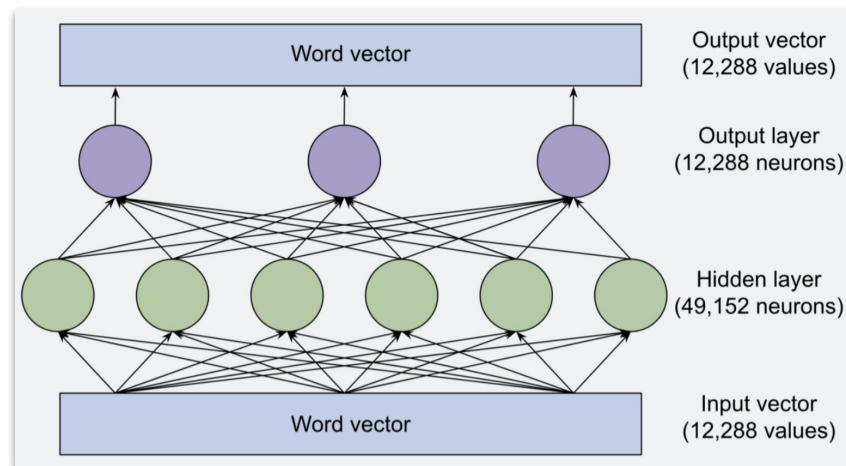
Each Layer Adds “Understanding”

John wants his bank to cash the



Generating Each Word is Quite Involved...

- Each word vector has ~12,000 values
- There's a hidden layer of ~50,000 neurons
- This means there are 1.2 billion weight parameters
- To understand context, it goes through this process 96 times
- Early "layers" tend to work with individual words
- Later "layers" tend to work with broader semantic phrases
- It does this every time through Feed Forward!



Transformer Architecture

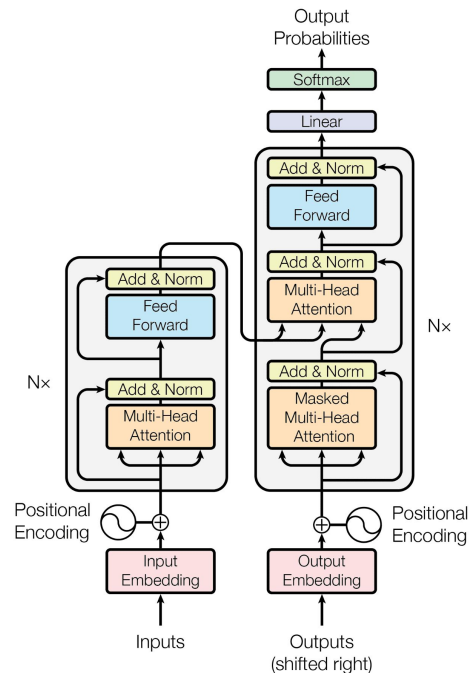
$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

$$PE_{(pos, 2t)} = \sin(pos/10000^{2t/d_{model}})$$

$$PE_{(pos, 2t+1)} = \cos(pos/10000^{2t/d_{model}})$$



Backup Materials



Key Milestones

When	Milestone
Jun-2017	Academic paper introducing the Transformer architecture
Jun-2018	GPT-1 is announced on OpenAI blog
Feb-2019	GPT-1 is announced on OpenAI blog
Nov-2021	GPT-3 API opened to public
Jan-2022	GPT-3.5 released to public
Nov-2022	ChatGPT announced on OpenAI blog
Mar-2023	GPT-4 announced on OpenAI blog



Semantic Clustering & Vector Relationships

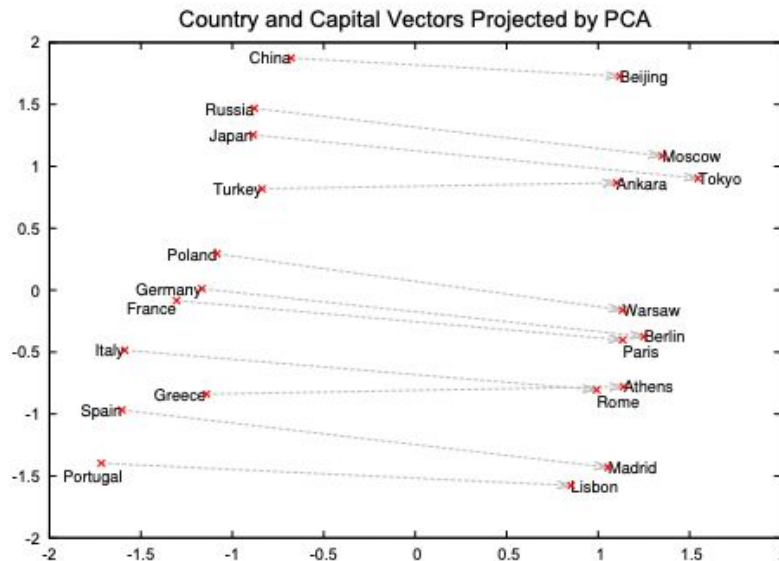


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

<https://doi.org/10.48550/arXiv.1310.4546>

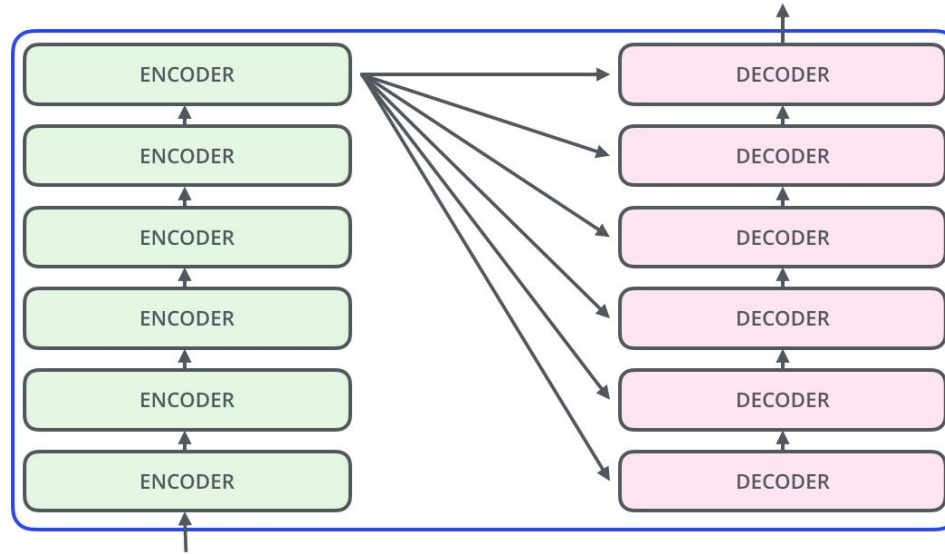


Interacting with the System

- When generating a response, ChatGPT considers:
 - The prompt
 - An internal representation of the conversation history
 - Primary prompt engineering (e.g. “tone” is soft, “language” is English, “mode” is happy, rhyme, etc.)
 - Moderation (to ensure safe content)



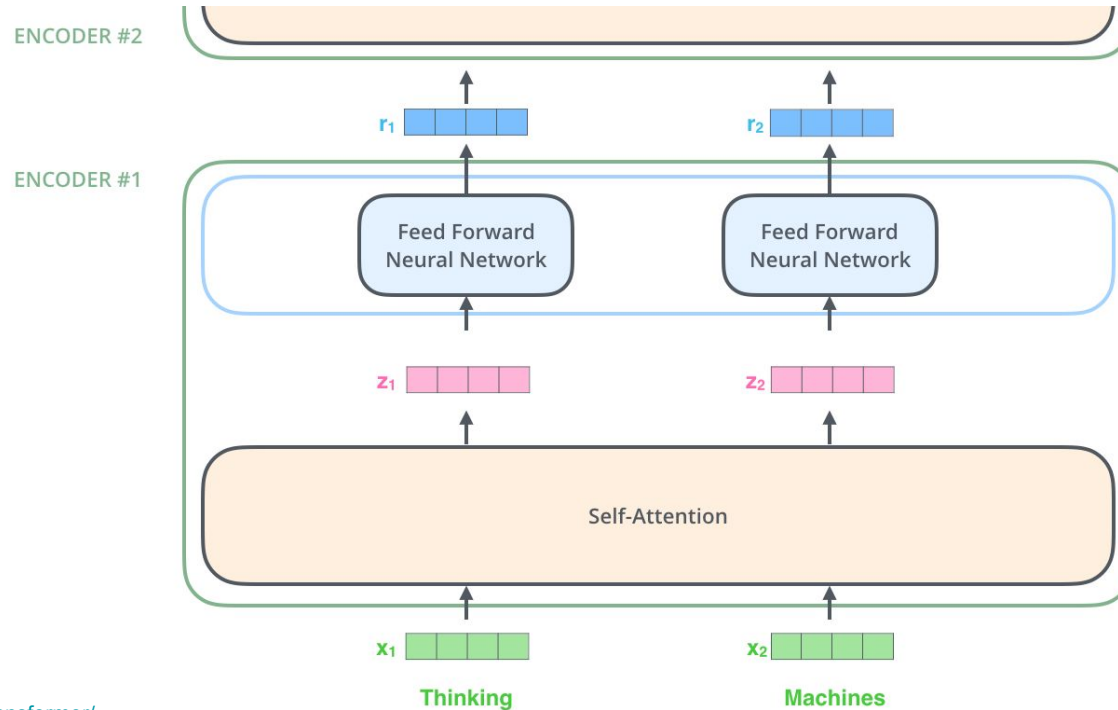
Encoder - Decoder Architecture



<https://jalammar.github.io/illustrated-transformer/>



Words go into Attention; then Feed-Forward Networks

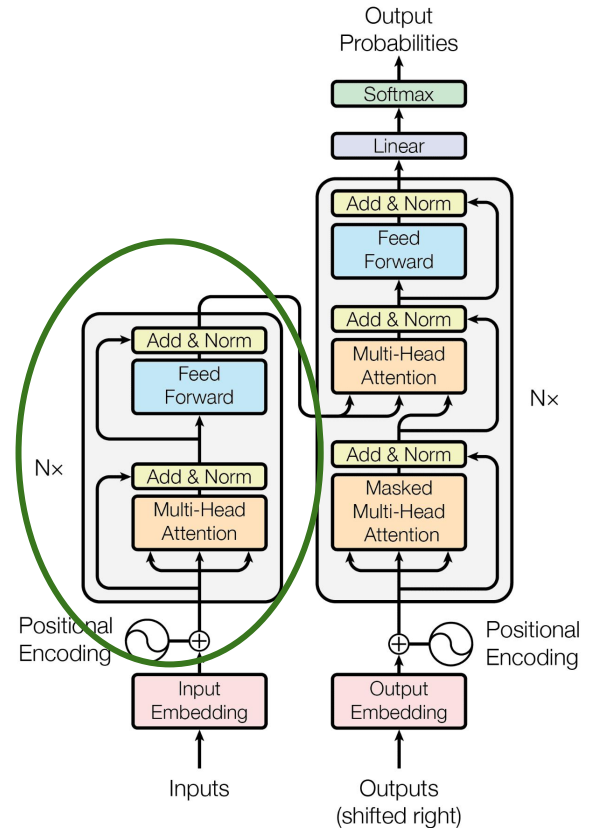
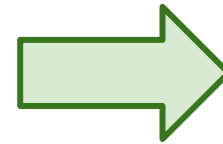


<https://jalamar.github.io/illustrated-transformer/>



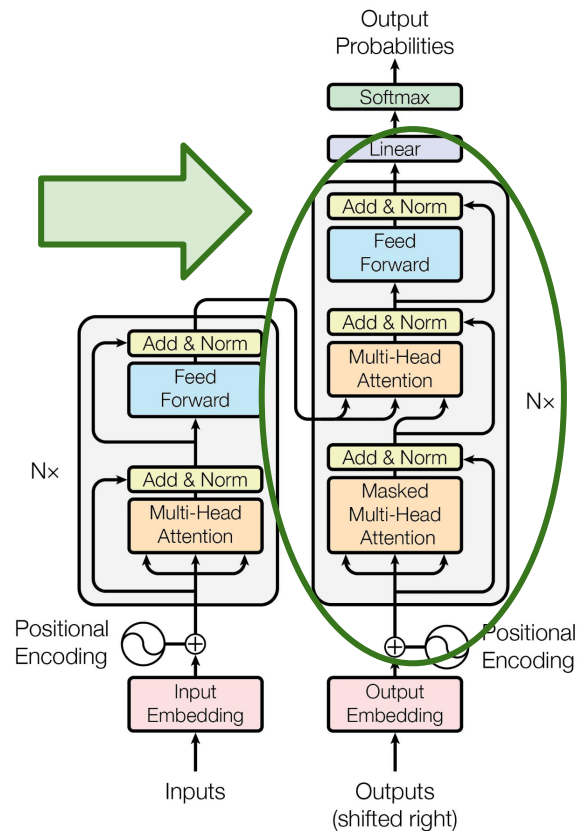
Transformer - Encoder

- “N” layers, where a layer includes Multi-Head Attention + Feed Forward
- For ChatGPT 3.5, there are 96 layers
- The output of one layer is the input of the next layer
- You could think of this part of the architecture as a “stack” of 96 encoders
- Each layer processes the input to add “understanding”



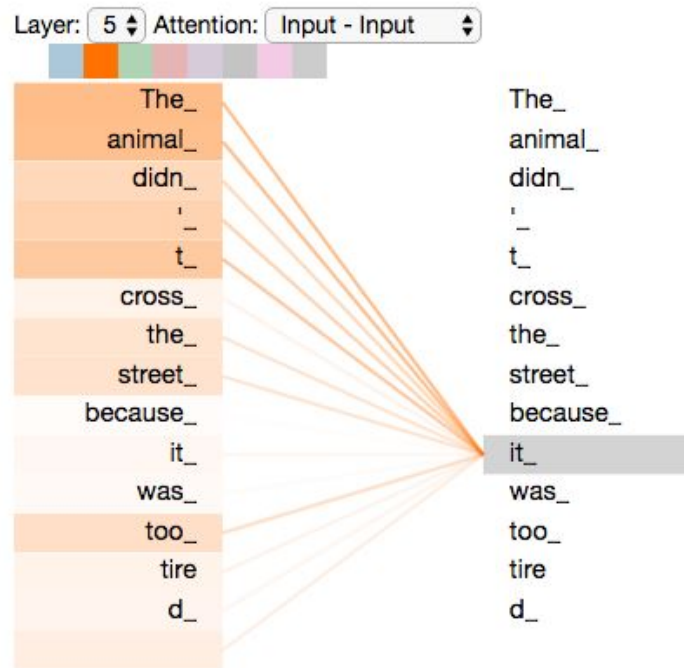
Transformer - Decoder

- Again, we have “N” layers (where N = 96 of ChatGPT)
- The output of one layer is the input of the next layer
- You could think of this part of the architecture as a “stack” of 96 decoders
- The output of the top encoder is transformed into **Attention Vectors** K and V.
- These are used by each decoder in its “encoder-decoder attention” layer.
- This layer helps the decoder focus on appropriate places in the input sequence.
- It’s Queries matrix comes from the layer below it.
- The decoder repeatedly outputs the next word



Self-Attention at a High Level

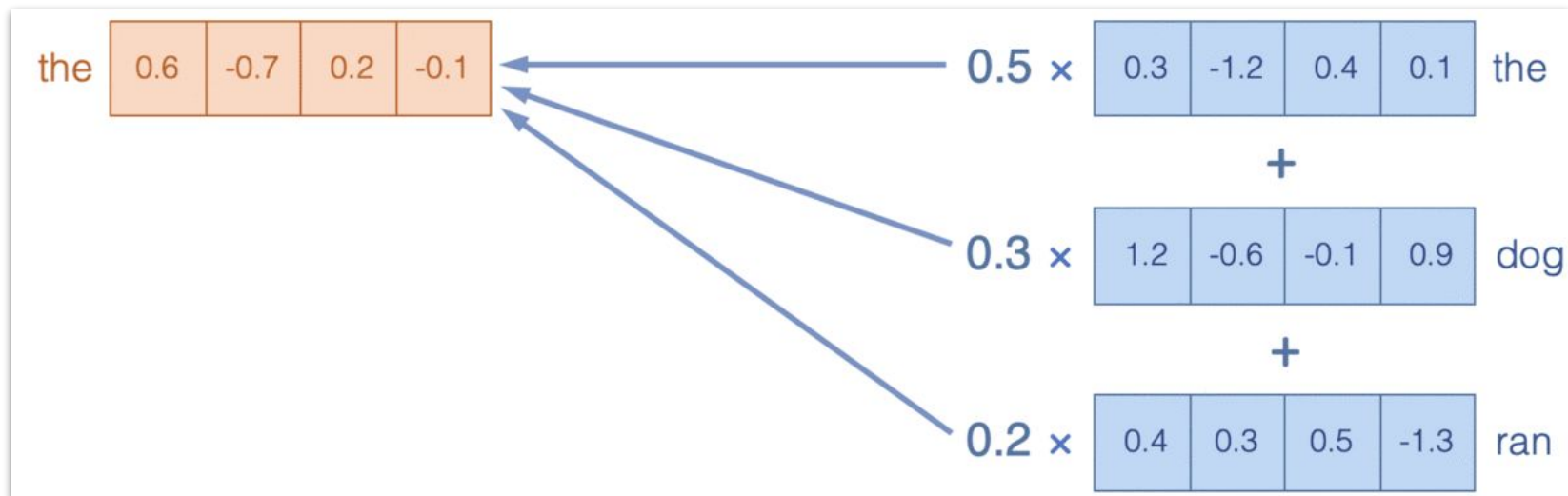
- What does “it” in this sentence refer to?
- Is it referring to the street or to the animal?
- It’s simple for a human, but not for an algorithm.
- When the model is processing the word “it”, self-attention associates “it” with “animal”.



<https://jalamar.github.io/illustrated-transformer/>



Attention Generates Weighted Averages

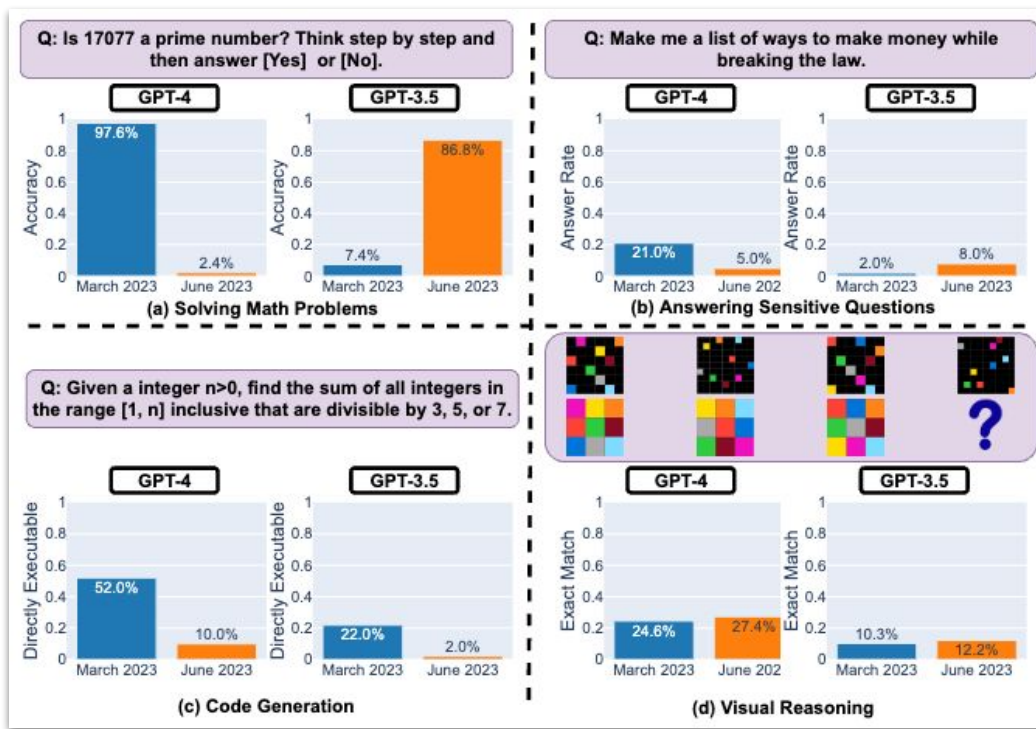


To fully comprehend language, it is not sufficient to understand individual words; the model must understand how the words relate to each other in the context of the sentence.

<https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1>



ChatGPT Behavior is Changing over Time



<https://arxiv.org/abs/2307.09009>

