Tech Explainers

HOW CHATGPT UNDERSTANDS CONTEXT: THE POWER OF SELF-ATTENTION

BUSINESS SUMMARY

Anyone who has used ChatGPT will notice that the quality of responses of this next-generation chatbot are superior to that of older chatbots. ChatGPT and other similar apps are somehow able to produce responses that are more coherent and well-organized and that go to the heart of the users' instructions. This leap in quality can be in large part explained by the innovation of "self-attention", a mechanism by which a machine learning model can use the context of inputs to extract and apply more information about language, and thus produce higher-quality outputs. In this technology explainer, we describe how the self-attention mechanism works, at a technical level, and highlight its legal implications. With the rise of legal disputes implicating generative AI apps and their outputs, it is imperative that legal practitioners understand the underlying technology in order to make informed assertions and adequately guide clients.

INTRODUCTION

Large language models ("LLMs") are machine learning models designed for natural language processing tasks. Generative LLMs focus on generating new text. One of the most influential and well-known generative LLMs is "GPT", a model developed by OpenAI. GPT utilizes a "transformer" model architecture (the "T" in "GPT"), first defined by researchers at Google in a 2017 paper titled "Attention Is All You Need."[1] As this title implies, the lodestar of the transformer architecture is the concept of "attention", specifically, "self-attention": a way for each element in a sequence to focus on other elements in the sequence and consider their importance adaptively. This mechanism captures contextual relationships between elements of natural language to produce a level of human-like coherence that makes it appear as if the model "understands" natural language.

In this technology explainer, we begin with the assumption that our model has already been trained, and we are examining how the attention mechanism works at "inference time", that is, operating on users' inputs it has not seen before. We will describe how the transformer model, specifically the subtype used by GPT,[2] uses self-attention to produce a mathematical representation of "context" to generate new text.

1 *See generally* Ashish Vaswani, et al., *Attention Is All You Need*, NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems 6000 (Dec. 4, 2017), https://arxiv.org/pdf/1706.03762.pdf.

2 The original transformer architecture consists of an "encoder" and a "decoder". GPT is often described as a "decoder-only" transformer because it does not utilize the encoder part of the transformer. For simplicity, we only refer to GPT as a transformer in this paper and do not otherwise explicitly distinguish between the decoder and encoder in the transformer.

NATURAL LANGUAGE: COMPLEXITY WE TAKE FOR GRANTED

Let's take a closer look at what seems like a simple sentence: "I picked up a book from the library before it closed." Despite being easily understandable to most English-speaking human readers, there's actually a lot of complexity baked into it.

What the sentence is trying to convey is that a person went to a place to get an object before that place closed. What do we make of the word "it" as used in the sentence? In a vacuum, "it" could have many possible meanings. For example, "it" could be referring to "book" or "it" could be referring to "library" (or neither, if "it" was being used as in the phrase, ". . . before it rained"). To further complicate the matter, both libraries and books can "close". Yet, despite all of these obstacles, our brains understand both the general meaning of the sentence, and specifically that "it", as used in our sample sentence, refers to "library", with little friction. How?

The answer is *context*. For example, there are a couple of key words that make it clear to a human being that "it" is referring to "library", including the use of the word "before", which conveys time and impacts the likelihood that "closed" means "not open for business" instead of "shut". The result is that "it" has some sense of place, perhaps imbued in part by the word "library" more strongly than the word "book". And while it would be grammatically correct for "it" to refer to the book, on a logical level, it falters—why would one need to go to the library before a book closed?

The understanding and production of natural language requires using the context of other words in a sequence to inform the meaning of each word. We as of yet have no exact science or clear path to explain how our brains use syntactic context to understand meaning—these are just some possible connections that could be drawn. When it comes to machine learning models, though, "self-attention" is the math-based mechanism by which they selectively weigh each element in a sequence and incorporate information from those elements.

TEXT GENERATION WITH SELF-ATTENTION

Let's begin considering a hypothetical individual user who wants to use ChatGPT, an app that utilizes the GPT model, to write a new blog post for her website. The user opens up the app and types in: "Write me a blog post about going to the library." ChatGPT produces a response that begins: "I picked up a book from the library before it closed . . . ." While it may not be obvious, the attention mechanism drove the generation of this output.

How did ChatGPT come up with this response? GPT is an "autoregressive language model", which means it generates outputs *one "token"* (generally, a word or part of a word) *at a time*, repeatedly using the "context" of the prompt and the iterative sequence of tokens generated thus far in its response, to produce the best next token.[3] For example, when GPT goes to generate the token "picked", it does so using the context of the user's prompt and the earlier generated token, "I". This continues on repeat until the model stops producing new tokens (by producing a special token telling it to stop).

3 In addition to the user's prompt and current output, ChatGPT also uses other tokens as context in its input, such as a user's prior prompts, ChatGPT's prior answers to such user, as well as "background tokens" hidden from users that help the model return more chatbot-like answers, among others. We exclude these other tokens from our discussion for simplicity.

INPUT	NEXT TOKEN
"Write me a blog post about going to the library"	"I"
"Write me a blog post about going to the library I"	"picked"
"Write me a blog post about going to the library I picked"	"up"
"Write me a blog post about going to the library I picked up"	"a"
"Write me a blog post about going to the library I picked up a"	"book"
. . . and so on, until a special token tells it to stop	

FIGURE 1: Illustration of GPT's autoregressive text generation, showing inputs and outputs for the generation of the first five tokens of the example. The input, or context, for the generation of the first output token is the prompt alone.

To simplify our discussion of the attention mechanism, we will imagine the model has already produced "I picked up a" and is using this outputted text (and not the prompt) to generate the next token.

First, GPT will "tokenize" the context, converting it into atomic units that the model can numerically represent and track:

```
# Tokenized sample sentence


[“I”, “picked”, “up”, “a”]
```

FIGURE 2: Tokenized sequence corresponding to "I picked up a".

Next, each token will be converted into a unique number, called a "token id", resulting in a list of integers such as the following:

```
# Token ids


[40, 6497, 510, 257]
```

FIGURE 3: Token id sequence corresponding to "I picked up a".

In the above encoded sequence, "I" is represented by the number 40, "picked" is 6497, "up" is 510, and "a" is 257.[4] If the same word appears in the sequence more than once, the same token id is encoded for each appearance of that word.

Each of these token ids is associated with what is called a "token embedding", in the form of a multi-dimensional vector (*e.g.*, a set of hundreds or thousands of component numbers). These components are sometimes also referred to as "weights". We can think of these weights as representing attributes of a given word. Although not discussed in this paper, these weights are among the parameters adjusted throughout the training process.

For our simplified example, we will imagine that the token embedding for each token has only five components (*i.e.*, a five-dimensional vector), and we will use hypothetical random embedding values.

TOKEN	TOKEN ID	TOKEN EMBEDDING
"I"	40	[0.72, -0.76, -0.66, -0.33, 0.57]
"picked"	6497	[0.20, 0.38, 0.14, 0.15, 0.76]
"up"	510	[0.04, -0.66, -0.40, 0.06, 0.45]
"a"	257	[0.65, 0.97, -0.46, -0.15, -0.51]

FIGURE 4: The tokens "I" "picked" "up" "a", corresponding token ids and token embeddings.

For example, the first component (0.72) may capture whether the given token represents a word that is a noun—perhaps a positive value (*e.g.*, 1.0) means something is a noun and a negative value (*e.g.*, -1.0) means it is not. In reality, these components are somewhat inscrutable, as they are derived through training, and often do not map so neatly to particular identifiable attributes of the word.

While the above table (*Figure 4*) is helpful to the human eye, it is not exactly how the model will ingest the inputs. Instead, the model ingests a matrix (a mathematical analog of a table) with four rows (corresponding to four tokens in our context) and five columns (corresponding to the five components of each token embedding in our example).

This matrix, called the "Input Embedding Matrix", is reproduced below:

```
# Input Embedding matrix

[
       [ 0.72, -0.76, -0.66, -0.33,  0.57], # "I" embedding
       [ 0.20,  0.38,  0.14,  0.15,  0.76], # "picked" embedding
       [ 0.04, -0.66, -0.40,  0.06,  0.45], # "up" embedding
       [ 0.65,  0.97, -0.46, -0.15, -0.51]  # "a" embedding
]
```

FIGURE 5: The data in the token embedding from the table in *Figure 4*, reproduced in matrix notation.

4 These token ids are predefined by whichever tokenizer the model uses; we have used token ids from the GPT-3 (legacy) tokenizer to demonstrate our example. See Tokenizer, OPENAI, https://platform.openai.com/tokenizer (last visited Jan. 17, 2024).

The first step in the attention mechanism is to pass the Input Embedding matrix into the model so that it can perform a sequence of operations with it. Using matrix multiplication, the model will combine the Input Embedding matrix with three different matrices: the Query Weight, the Key Weight and the Value Weight matrices (each, a "Weight matrix").[5] Just as the Input Embedding matrix contains components adjusted during the training process, the components in the Weight matrices are likewise adjusted during the training process, and are similarly somewhat inscrutable. Whereas the Input Embedding matrix is meant to directly represent the relevant tokens, the Weight matrices will be used to capture contextualized information among tokens.

The Input Embedding matrix is first multiplied against each of the Query Weight, Key Weight and Value Weight matrices. The result is three new matrices: the Query, Key and Value matrices. The Query matrix and Key matrix are then combined, again using matrix multiplication, to produce the Attention Score matrix. (We will return to the Value matrix at a later step.)

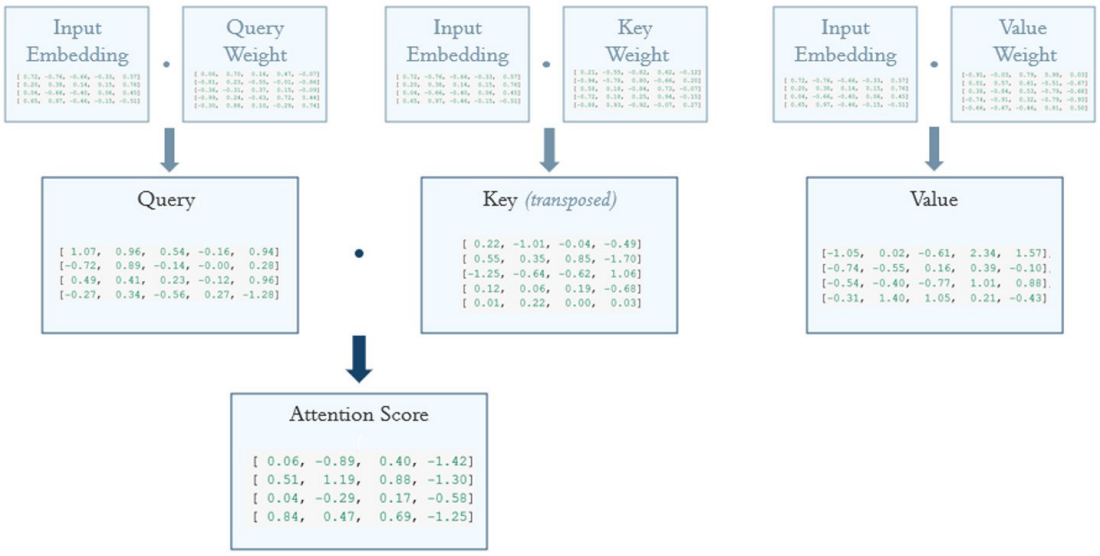

FIGURE 6: Visual representation of matrix multiplication steps to combine the Input Embedding, Query Weight and Key Weight matrices to produce the Attention Score matrix.[6]

5 In this article, our narrow goal is to demystify how GPT can extract contextual information through self-attention. In reality, GPT has *many* other trained weights than those we describe in this article (including the other sets of query, key and value matrices). And even within this small piece of the transformer model we focus on, we are simplifying *significantly* for the purposes of this discussion. For example, in addition to the fact that there are considerably more weights for each input in the Weight matrices, we also do not discuss how the model accounts for token order (addressed by "positional embeddings").

6 Matrix multiplication combines two matrices to produce a third matrix, where each number in the resulting matrix is calculated by the sum of products of corresponding row and column elements. In matrix multiplication, the number of columns in the first matrix must equal the number of rows in the second matrix. The resulting third matrix will have dimensions corresponding with the number of rows in the first matrix and the number of columns in the second matrix. See *Figure 10* for an example.

The numbers in this resulting Attention Score matrix represent a combination of the information captured in the Input Embedding, Query Weight and Key Weight matrices, or put another way, the Attention Score matrix constitutes a set of numbers representing the relationship *between* each of the tokens. This can be represented in a table as follows:

	"I"	"PICKED"	"UP"	"A"
"I" ATTENTION TO	0.06	-0.89	0.40	-1.42
"PICKED" ATTENTION TO	0.51	1.19	0.88	-1.30
"UP" ATTENTION TO	0.04	-0.29	0.17	-0.58
"A" ATTENTION TO	0.84	0.47	0.69	-1.25

FIGURE 7: Attention Score values resulting from calculations described and represented in *Figure 6*.

These values are then scaled to control the impact that outlier small and large attention scores can have on later calculations, which can destabilize the model. Each row of these scaled scores is then "masked", which means that the attention weights associated with tokens occurring after the token of interest are set to 0.00 (as seen below, in *Figure 8*) to prevent tokens from attending to subsequent tokens in the sequence. Then, the masked scores are normalized through a function called "Softmax", which converts the scores into a probability distribution, with the components in each row adding up to one. The resulting matrix is called the "Attention Weight" matrix:

```
# Attention Weight matrix

[
       [ 1.00, 0.00, 0.00, 0.00], # "I" attention weights
       [ 0.42, 0.58, 0.00, 0.00], # "picked" attention weights
       [ 0.34, 0.30, 0.36, 0.00], # "up" attention weights
       [ 0.31, 0.27, 0.29, 0.13]  # "a" attention weights
]
```

FIGURE 8: Attention Weight matrix.

How does one read this matrix? Let's break it down using a table:

	"I"	"PICKED"	"UP"	"A"
ATTENTION THAT "I" SHOULD PAY TO	1.00	0.00	0.00	0.00
ATTENTION THAT "PICKED" SHOULD PAY TO	0.42	0.58	0.00	0.00
ATTENTION THAT "UP" SHOULD PAY TO	0.34	0.30	0.36	0.00
ATTENTION THAT "A" SHOULD PAY TO	0.31	0.27	0.29	0.13

FIGURE 9: Attention Weight matrix reproduced as a table showing the attention that each token should pay to each other token in the sequence.

We can now see what these numbers represent: a "weight" of how much each token should "pay attention" to each other element in the sequence (including itself) as context for the purposes of generating the next token. That is, an "attention weight". Tokens with lower attention weight values have less influence on contextualized representation of the surrounding tokens. For example, we can crudely state that the attention weights in *Figure 9* suggest that "a" should pay most attention to "I" because 0.31 is the highest value in the row containing "a". On the flip side, "a" should pay least attention to itself, "a", because 0.13 is the lowest value in that same row.

The model then uses these attention weights to produce a new *contextualized* token embedding for each token—a representation of the token that includes information about the relationships between it and other tokens, and thus goes beyond the token's stand-alone attributes—by multiplying the Attention Weight matrix by the Value matrix (which has not yet been used in our calculations, recall from above) to produce the Attention Output matrix.

For demonstration purposes, this is what the calculations of the attention scores would look like for the "a" token, which will ultimately be used to predict the next token:

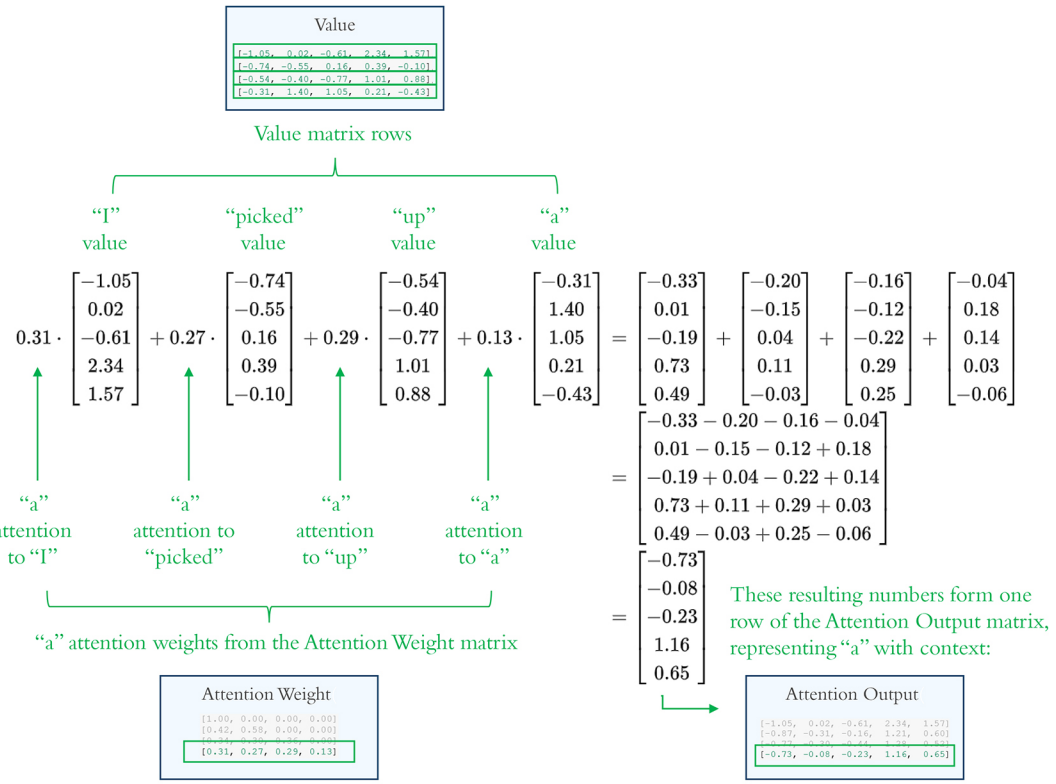

FIGURE 10: Portion of algebraic steps multiplying the Attention Weight matrix with the Value matrix, calculating values corresponding to "a" with context, forming Row 4 of the Attention Output matrix.

Figure 10 shows the calculations combining the attention that "a" pays to each token in the sequence (numbers from Row 4 of the Attention Weight matrix) with the value embedding of each token in the sequence (each row of the Value matrix). The resulting numbers represent "a" with context, forming Row 4 of the Attention Output matrix. Carrying out this process completely (for each input token), the resulting Attention Output matrix is as follows:

```
# Attention Output matrix

[
        [-1.05,  0.02, -0.61,  2.34,  1.57], # "I" with context
        [-0.87, -0.31, -0.16,  1.21,  0.60], # "picked" with context
        [-0.77, -0.30, -0.44,  1.28,  0.52], # "up" with context
        [-0.73, -0.08, -0.23,  1.16,  0.65]  # "a" with context
]
```

FIGURE 11: Attention Output matrix with Row 4 highlighted, corresponding with calculations shown in *Figure 10*.

We started with the Input Embedding matrix, which contained weights representing standalone attributes of each token. After undertaking each of the preceding calculations, now each row in the Attention Output matrix represents *both* the attributes of each standalone token *and* contextual information from each preceding token. In other words, we end up with a representation of the tokens that is context-aware.

We can now see the impact of this Query, Key and Value system. The Query and Key matrices are used to ascribe context by assigning attention weights between the input tokens. The attention weights are then applied to representations of the input tokens in the Value matrix. And the end result is an Attention Output matrix that represents the inputs with some sense of context. This is why it is called "self-attention"; it is the mechanism where each element in a sequence attends to all other elements, including itself.[7] This self-attention mechanism is the core feature used by GPT to understand context when making next token predictions.

Now going back to the initial hypothetical, our user prompted ChatGPT to "Write me a blog post about going to the library." ChatGPT responded using its underlying GPT model to generate "I", "picked", "up" and "a" one token at a time, and would next be most likely to select "book" as the next best token. While our simplified example focused only on how attention works for the generated output, in reality, the attention mechanism would also be applied to the prompt as well—after all, the context of tokens from the prompt is how it knows to generate "a blog post about going to the library".

The types of calculations described here occur every time a new token is generated in response to a user's ChatGPT prompt. The ability to attend to thousands of preceding tokens is what makes GPT's outputs more sophisticated and human-sounding than that of older chatbots. How, precisely, GPT uses this context to produce the next token is a complicated process that is outside the scope of this technology explainer.

LEGAL SIGNIFICANCE OF SELF-ATTENTION TRANSFORMERS

Given its widespread use and importance in many modern generative AI models (even far beyond LLMs), understanding the technical underpinnings of self-attention is critical. The legal issues presented by generative AI are complex and evolving. In particular, potential liability for copyright infringement and ownership regimes are areas of outsized focus in the current legal landscape.

This technology explainer takes one step in the direction of demystifying how generative AI apps such as ChatGPT are able to "understand" users' prompts and return such intelligible answers. By using the self-attention mechanism, the model underlying ChatGPT uses context—generally the user's prompt and its own previous output—to produce new tokens, one token at a time, through a series of mathematical calculations. While these calculations are highly complex in aggregate, the key takeaway of the self-attention mechanism is that the model uses the relationships between each token in a sequence to inform the meaning of each token within the sequence. With this more nuanced information, it is able to produce better, more human-like outputs.

7 The "self-attention" process we describe is only one small step in a transformer model. The Attention Output matrix is the product of only one "attention head", and there are several others. This process also includes a series of feed-forward neural networks and is repeated multiple times. A significant portion of the learning occurs in these other layers, which contain more parameters that the model will be able to toggle as part of its design to reduce the loss function in training. Yet, the process we described remains the core mechanism explicitly designed to extract context from the inputs.

It is no longer adequate for legal practitioners to speak about generative AI in broad-strokes analogies and assumptions, which may be misleading and imbue erroneous legal conclusions. From legal briefs, to policymaking, to conversation, it can be tempting to analogize between the function of LLMs and human thought processing and creation. However, a more precise understanding of how and why modern generative AI models are capable of producing increasingly coherent and human-sounding outputs is now a prerequisite to applying the law to this technology.

David J. Kappos
+1-212-474-1168
dkappos@cravath.com

Sasha Rosenthal-Larrea
+1-212-474-1967
srosenthal-larrea@cravath.com

Carys J. Webb, *CIPP/US, CIPP/E, CIPM*
+1-212-474-1249
cwebb@cravath.com

Daniel M. Barabander
+1-212-474-1284
dbarabander@cravath.com

Lucille Dai-He
+1-212-474-1860
ldaihe@cravath.com

Cravath, Swaine & Moore LLP

NEW YORK

Worldwide Plaza
825 Eighth Avenue
New York, NY 10019-7475
T+1-212-474-1000
F+1-212-474-3700

LONDON

CityPoint
One Ropemaker Street
London EC2Y 9HR
T+44-20-7453-1000
F+44-20-7860-1150

WASHINGTON, D.C.

1601 K Street NW
Washington, D.C. 20006-1682
T+1-202-869-7700
F+1-202-869-7600