ALL LLMS are FICCIONES

In 1940, Jorge Luis Borges wrote about a society that became so obsessed with a work of fiction that they abandoned their own culture to adopt the fictional one as reality. The work itself was an extraordinary hoax - a collaboration between scholars and writers of every discipline across generations to produce a false encyclopedia of a non-existent planet called Tlön. Once released, something strange happened: people began living its fantasies. They found its contents intoxicating, slowly losing touch with their own customs and traditions in favor of the manufactured lore.

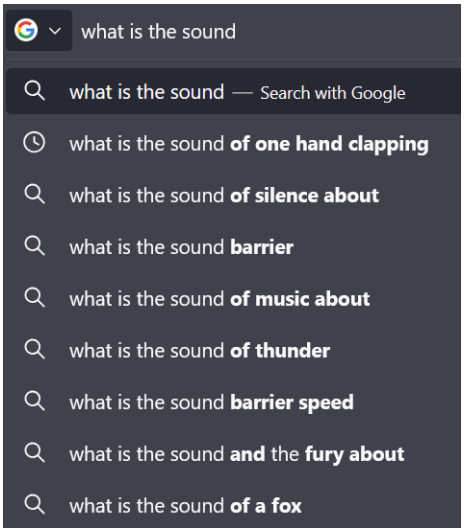
As Large Language Models (LLMs) become more ubiquitous, we too now run the risk of losing our cultures to that of a constructed design. I'd like to provide you with an explanation as to why we must consider the output of all such models as fictitious - that is: why can their output *seem* truthful without actually being true? To answer this question, we need to get a better sense of both a) how LLMs operate and b) what information they contain.

LLMs do not contain facts. They do not even contain words or text. They merely contain relationships.

These relationships, however, are not between words or subjects or ideas, but rather numbers - specifically numerical tokens. The concept of tokenization is the first fundamental abstraction we need to understand before we can understand why all generative AI is fictitious. These models are not exposed to text, but rather numerical IDs that represent groups of characters. For example, in a common tokenization scheme, the phrase "babbling brook" is broken down into the tokens "bab," "bling," and " brook." These in turn are associated with the numerical IDs, [86485, 11037, 198955], respectively (note that the space ' ' before brook is *part* of the token here - this will be important later).

Tokens are strings of at least 1 character, but can sometimes be full words.
Example generated from platform.openai.com/tokenizer

The motivation for tokenization is partly that of compression. To demonstrate this, let's place ourselves as the designers of a Large Language Model. It is not unfair to say that LLMs are sophisticated versions of auto-complete - the feature found in search engines and word processors where if one starts typing a phrase such as "what is the sound of," the auto-complete may suggest "one hand clapping" or "thunder" or "silence about." In the same way that auto-complete finishes a phrase, LLMs scale to complete an arbitrary corpus of text, be it a conversation, a recipe, HTML code, or anything else you can imagine.



An example of Google's auto-complete functionality.

We can think about the problem this way - given a word or phrase, how do we predict which word or phrase would logically follow in sequence?

Something particular to all digital machines, not just generative AI, is that, in order to produce text there must be, at some point, a mapping of numbers to an alphabet or dictionary - this is merely a property for these types of machines. We will call these various ways to map text to numbers as tokenization schemes.[1]

Let's consider two possibilities for prediction - mapping numbers to letters versus mapping numbers to words. Focusing on words, if I were to give you the word "babbling," and ask you to tell me what word follows, I could guess with high confidence that you would respond with either "brook," or "baby." If we were doing a token mapping of words to numbers, this means that with the single token for "babbling" we have a pretty high confidence of guessing the next token correctly.

Now imagine if we were using the token mapping of numbers to letters rather than words. If I gave you a single token, it would be for the letter "b." A single token in this scheme means I need to predict each letter at a time. It would be difficult for a model to accurately predict not just "brook" or "baby", but also "-abbling" from the single token representing "b." You would need 8 times the tokens, one for each letter in babbling, to provide the machine with the same amount of information. In turn, rather than making the single prediction for "baby" or "brook," the machine now needs to make a prediction for each and every letter in those words - a much more daunting task.

It's easy to see, from a prediction standpoint, that it would be a much easier task if we could stick to tokenizing words over letters. Still, if this machine is to be robust, we need a way to keep track of every *possible* word. One solution may be to fetch a dictionary and just number every word in sequence starting with 1 for "a" until you reach some six-digit number when you arrive at the final word, "zyzzyva" (a South-African weevil, by the way). In this token scheme, our example of

“babbling brook” may correspond to something like [20101, 0, 29132], for "babbling," “ ,“ (an empty space) and “brook.” This strategy may not be so bad, but then what happens when you encounter a word not in the dictionary, such as novel slang, a proper noun, or even just an uncommon conjugation?

The mapping of words to numbers suffers from extendability. Anytime we come across a new word, we need to change our token scheme, and because this necessitates constant growth, it is fundamentally unstable. A logical alternative may be to use individual letters as atomic building blocks for words. In this scheme, we would just number every letter from 1-26 (note: this is a much simplified example that is not accounting for capitalization or punctuation). Unlike our word-to-number scheme, the letter-to-number scheme is perfectly extendable, and can easily account for any novel word.

One problem with mapping letters-to-numbers is that, while we are able to capture every possible word, our representation system is now incredibly large. To represent “babbling brook,” our tokenization might look something like [2, 1, 2, 2, 12, 9, 14, 7, 0, 2, 18, 16, 16, 11] corresponding to “b,” “a,” “b,” “b,” “l,” “i,” “n,” “g,” “ ,” “b” “r,” “o,” “o,” and “k.” As previously discussed, this increases the complexity of the prediction task immensely, as it now requires more input (more tokens) to make an accurate prediction.

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

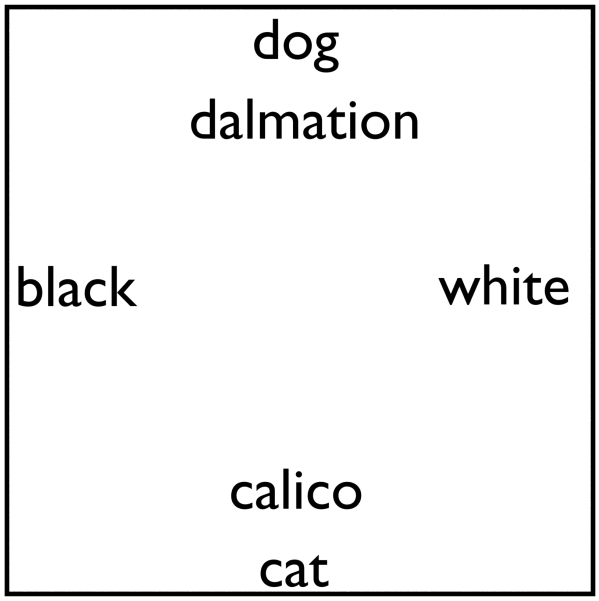
0123456789

Counting with tally marks is similar to the per-letter token system. We can represent any number, but we lose the value of compression - it could take up a great deal of space to represent a number like 1000. Roman numerals compress large numbers quite nicely, but do not extend so easily. New symbols need to be created to represent new, larger numbers. Roman-numerals with a base-10 number system can be seen as a kind of compromise between the two - offering the advantages of extendability and compression we see in tokenization.

Tokenization as an abstraction resolves the tension between predictive power of word-based representation and the extensive properties of letter-based schemes. The exact methods of tokenization may differ slightly, but they generally contain both individual letters and whole words as tokens, *as well as* strings of common letters (such as “bab” and “bling” being their own tokens). Individual letters as tokens provides a worst-case scenario for compression, but allows for maximum extendability. The exact procedures differ, but in general, some form of analysis is done over large amounts of text to reveal common sequences of characters, which then become associated with a token ID, such as 86485 for “bab,” 11037 for “bling,” and 198955 for “ brook.” Uncommon sequences, such as “xqz” are represented as individual tokens for “x,” “q,” and “z,” while a more common sequence such as “xyz” is represented by a single token.

This numerical representation of text still doesn't explain any predictive properties of the LLM. If "bab" and "bling" are separate tokens, how does the model learn to associate them together, and how does it learn that those two tokens are then associated with "brook" or "baby?" To understand how these models learn associations between tokens, we must now turn our attention to the second fundamental abstraction of generative AI - embeddings.

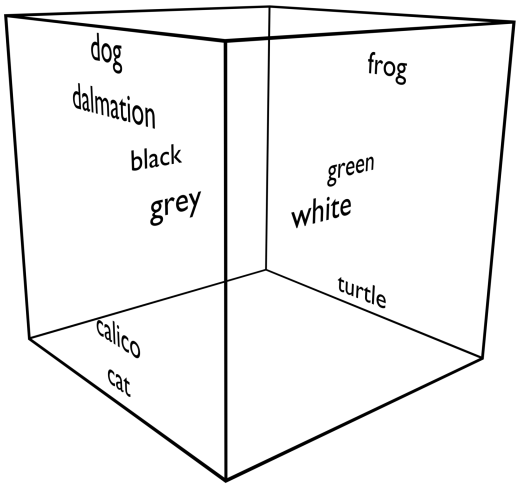
As an exercise, imagine a completely empty square. We are now going to start placing concepts in this square while keeping it as organized as possible. Our first concepts are "black" and "white". Since they are opposite enough, we will put "black," on the far left side and "white" on the far right side. Next we are given "cat" and "dog." We will place "dog" at the top of the square and "cat" at the bottom. If we were to step back now, we can see that our square may represent different colors of common household pets. If we now receive "dalmation" and "calico," we would likely place them somewhere towards the center of the square.



When organizing concepts in a square, we might place "dalmatian" close to "dog", but also somewhere in between "black" and "white". You could imagine if we were given "doberman," it would go somewhere to the left of dalmatian, closer to black than white.

Suppose we now receive "green." We may first want to place it somewhere between black and white, but then we realize that it doesn't quite fit appropriately between dog and cat. The spectrum of "black and white" has some fundamental logical relation to "cats and dogs" and "green." However, "cats and dogs" and "green" are not as symbolically linked. The solution is actually quite elegant - instead of giving you a square to organize these concepts in, we'll extend the space by a dimension and turn it into a cube. Now we can visualize organizing these concepts

in a room. Imagining a single door that opens to a perfectly cube room. We still keep black and white on opposite left and right sides. We will also keep “dog and cat" towards the top and bottom of the room, however, they will be immediately as we walk in.



Once we extend the dimensions, we can now retain the association of “dalmation” and “calico” to “black” and “white” while creating a distance between those animals and “green.” Note this analogy is just for illustrative purposes. The takeaway is that adding dimensions can create new pathways for concepts to be related or unrelated.

We can then place green in the center of the room, between black and white, but not directly between dog and cat. If we then receive “frog” and “turtle” we could place these on the far opposite wall, flanking green, but with some distance from dog and cat.

The point here is not the exact categorization of terms, but rather a demonstration that, given enough space, we can come up with a specific way of organizing concepts. Increasing the number of dimensions we have access to increases the number of ways our concepts can be associated.

In this example, we came up with our own associations and organizations of concepts based on our own knowledge and intuition. LLMs develop associations between various tokens via repeated exposure to extensive training data. Rather than a simple bi-direction (left/right, top/bottom, or x/y) or even 3D (position in a room, or x/y/z) association, the number of dimensions for these relationships, called embeddings, can be in the thousands.

If tokens map text to numbers, we can think of embeddings as organizing these tokens in some high-dimensional space. On one dimension “bab” and “bling” might be

closer together, in another, “bab” and “ylon” might be, and in yet another “bling,” and “shiny” could share a strong association.

If Generative AI models contain any knowledge (read: know anything), it is the positioning and arrangement of these embeddings. Recall that these embeddings are not arrangements of concepts as in our example (black, white, dog, cat) but of tokens.

How the exact embeddings are found is the result of repeated exposure to training data. For an LLM, incomprehensible amounts of text (more than a single human could read in a lifetime) are “tokenized” and exposed repeatedly so that the model can learn to approximate the relationships in the existing text. This is important, as a model can only learn connections from that which it is exposed to.

Linguistic Emily M. Bender coined the phrase “stochastic parrot.” To refer to how LLMs operate. The parrotting portion refers to how these models can only learn to parody the data they are exposed to. “Stochastic” refers to the fact that there is a bit of probability and randomness involved. Recall that LLMs work like auto-complete at scale. How they determine the next token is going to be dependent on the prior. For illustration, if I start typing “bab,” how would the LLM know which token “ylon” or “bling” follows. Maybe it recommends “ylon” 60% of the time. But what if I were discussing a river prior to typing “bab?” Because this changes the context of the conversation, it may not recommend “bling” 90% of the time. Perhaps if I were discussing Iraq, the probability of the model selecting “ylon” would increase to 95% instead.

We have established that LLMs work based off 3 fundamental abstractions:

1. Tokenization - LLMs operate off of tokens - numerical references to strings of text.
2. Their “knowledge” is merely the multi-dimensional arrangement of these tokens, aka the embeddings, determined by the tokens exposed to it during training.
3. Their output is a probabilistic sampling of these tokens from this multi-dimensional space.

In this way, the output of LLMs is more fictitious by nature than it is factual. It contains no notion of “facts” or what a ground-truth might be. It may contain a relative sense of however the text “facts” gets tokenized, but this is not the same as knowing the truth. To demonstrate this further, I want to turn to the well-known example of the “strawberry” problem.

Around the release of GPT-3, it was a common ploy to test the knowledge of LLMs by asking, “How many r’s are in strawberry?” A literate human could tell you 3, but the LLMs would consistently return the response, “2.” To understand why it would respond with such an illogical answer, we need to understand it through the 3 abstractions previously mentioned.

The tokenization of “strawberry” returns the tokens “str,” “aw” and “berry.” Interestingly enough, “ strawberry” (with a space before the ‘s’) is its own single token. This is actually quite common in tokenization schemes. If a full word is tokenized, it is most likely to be with a single space in front of it, partially to designate it as its own word, but also because there is unlikely to be any word that has “strawberry” in the middle of it. This is unlike “berry,” which is its own token with *and* without a space in front of it, as “berry” is a common enough string of text to appear on its own within a word.

Even though “str,” “aw,” and “berry,” are separate tokens, they are going to occupy some similar space in some embedding dimension to “ strawberry.” The individual tokens will become linked to one another. “Str” and “ strawberry” may be arranged together in a space of words that start similarly, while “berry” and “ strawberry” be linked in yet another. But remember, these are just tokens. The LLM does not know that “berry”(token: 19772) is a substring of “ strawberry,” (token: 101830). but it has formed, through repeated exposure, that token 19772 is associated with token 101830.

So when we ask “how many r’s are in strawberry,” the LLM is not considering its response in any manner similar to how humans would. First it receives [8923, 1991, 428, 802, 553, 306, 101830] corresponding to:

Our text is actually received as [8923, 1991, 428, 802, 553, 306, 101830] to the LLM, each token corresponding to the strings above.

If, during training, the LLM never is exposed to the string “there are 3 r’s in strawberry,” the LLM may not be able to respond with the factual answer.

LLMs create text that seems factual in the same way that historical fiction depicts true events. The context and appearance may seem appropriate, but there is no strict delineation from true events and fictional narratives. Trying to derive truth from LLMs is like trying to understand history by only reading historical fiction - you may, incidentally, arrive at certain truths, but there is nothing than can inherently corroborate the truth.

This does not remove utility from generative AI. There are a number of ways to bootstrap these models with capabilities outside of the generative models themselves, such as the use of search-engines or other third-party documents that can essentially bias certain connections to a ground-truth or preference as determined by humans. This is what a good deal of Reinforcement Learning for Human Feedback (RLHF) and some other modern “extended-thinking” mechanisms operate.

It is important, though, to properly understand the output of these machines as something that *can be designed*. In Borge's world, Tlön was the master project of an atheist slave-owner from the early 19th century. Much like with that fictional planet, on ours there are human actors deciding what tokenized text these models are exposed to. We can bias them towards something that seems like the truth, they can even stumble upon things we can prove to be true, but they never inherently know the truth. But just as in Borge's world, ours can adopt these fictitious outputs as our lore henceforth, adopting the outputs while slowly forgetting and erasing the sources from which they come from.

1. In reality, if one were focusing on just mapping numbers to characters, they are more likely in the realm of character encoders, such as ASCII and UNICODE (UTF-8). These examples are more for illustrative purposes, and for that purpose, we can call them tokenization schemes.