Does a transformer think like a human?

Andrej Lúčny

Katedra aplikovanej informatiky, Fakulta matematiky, fyziky a informatiky, Univerzita Komenského
KAI FMFI UK, Mlynská dolina, 842 48 Bratislava
lucny@fmph.uniba.sk

Abstract

We describe the operation of transformers and compare it with our ideas about the nature of human information processing. Using examples of the use of individual models, we point out striking similarities and fundamental differences between a transformer and a human:
1.) By setting the correct temperature, chatbots can generate logical errors characteristic of humans (we proposed these years ago as a test of whether an artificial intelligence system resembles a human).
2.) Using the example of holistic OCR, we show that it is possible to loop and confuse a transformer in a way that a human would not make a mistake.
3.) In the case of naming a controversial image, we demonstrate behavior that is not typical of humans and overcome it using an agnostic algorithm for calculating bipartition, suggesting that humans probably also have a similar ability.

1 Introduction

Alan Turing had shown in [Turing 1950] that an essential property of artificial intelligence systems is that they make mistakes. His argument was based on Gödel's incompleteness theorems, from which it is possible to derive that for any mistake-free intelligent system, there is an input on which it endlessly loops [Kelemen 1990]. In this paper, we relate these old ideas to the latest transformer-based artificial intelligence systems: ChatGPT [OpenAI 2023], DeepSeekOCR [Wei 2025], and CLIP [Radford 2021]. We find interesting relations.

2 How transformer-based LLMs works

A large language model is a neural network with a transformer architecture that, given a text as a sequence of tokens (pieces), calculates for each token the probability that it follows. If the transformer is of the decoder-only type (for example, GPT, the basis of the most famous chatbot), we use it as a generator of a scenario that starts with a user question and continues with the assistant's answer. From the entered text, it calculates the probabilities, based on which it selects the following token. That token is appended to the current scenario and then fed back into the transformer, run again to generate the next token. It repeats this until it generates a special token representing the end of the sequence (EOS) (see Fig. 1).

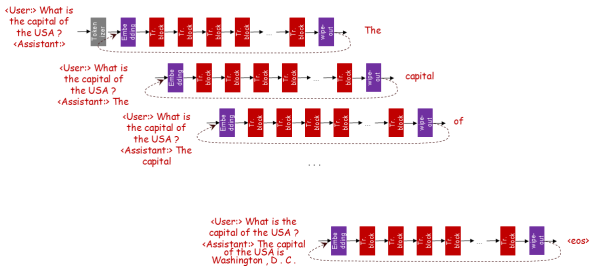

Fig. 1. Decoder-only transformer in the action

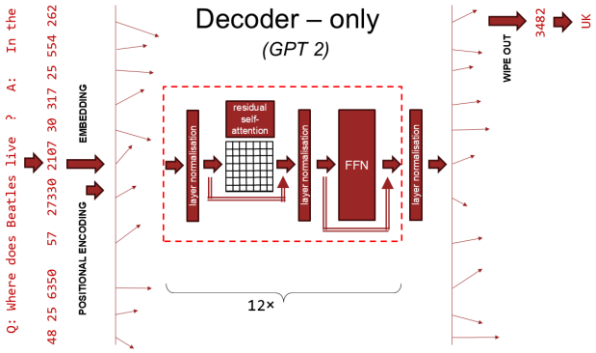

Fig. 2. The next-token prediction (GPT2)

The next-token calculation process (Fig. 2) begins with embedding, which assigns each token a vector in a multidimensional space. The dimension of this space corresponds to the number of features by which two things can differ in their meaning. Since there are only a few tens of thousands of possible tokens, embedding is implemented using a table that maps token indices to their corresponding vectors. The table values are the network parameters; their final form is the result of training. The vectors are organized in the space so that the angle between them is small when they occur frequently in the same context in the training data (and therefore affect each other's meaning). The almost equal lengths of the vectors are ensured by layer normalization, a building block of the transformer,

which is applied after each data processing step. As a result of its application, the vectors after each normalization roughly point to the surface of a hypersphere with one fewer dimension than the vectors.

Once the input text has been transformed into a list of vectors, the so-called hidden states, we add a positional code to each vector that encodes its position in the sentence. We further transform them into other hidden states by a sequence of so-called layers. The number of hidden states in each layer is the same and corresponds to the size of the input. Each layer consists of two modules: a residual multi-headed self-attention and a perceptron.

The first module, self-attention, refines and contextualizes the meaning of hidden states. It works by mixing each hidden state with a little of the other hidden states, mainly those that yield the largest scalar products and or the smallest angles. Attention includes a scaling coefficient that determines how much of the more similar and how much of the more different hidden states are mixed. Its optimal value is the square root of the hidden-states dimension, but a slight change does not cause a system crash; it only leads to slightly different behavior. To get slightly different outputs from the transformer when different calls are made with the same input, we multiply it by the number (1+T), where T is the so-called temperature, a randomly generated non-negative number. The higher the temperature, the more "creative" the generated statement is, until it becomes meaningless. Therefore, its reasonable maximum is considered to be the value of two.

The second module processes each hidden state separately with a perceptron. In doing so, it transforms the features of the hidden states into others, allowing nonlinear states to emerge between the new and original features.

After processing the hidden states across all layers, we wipe out the last state to index the next token. Unlike inference, during training we enforce wipeout across all hidden states, requiring that the next one emerge from the last, the last from the next-to-the-last, and so on.

3 As good as humans

Many years ago, in [Lúčny 2002], we proposed a specific test to determine whether an artificial system is intelligent in a manner characteristic of humans. The tests consisted of whether it was capable of failing in the same way as a human. One such test consisted of answering two questions:

1. How many dollars are in a dozen?
2. And how many quarters?

Only a few people answer this question with 12 and 12. Most people answer 12 and 48. But there are also more exotic answers, such as 12 and 3. It may be true to object that the question is not quite correctly understood as "How many quarters are in a dozen dollars?" But people do not stop being wrong even if we insist:

1. How many dollars are in a dozen?
2. And how many quarters are in a dozen?

Is the transformer wrong like this? It's shocking, but it's wrong exactly like this when we increase the temperature to get various answers (Fig 3, Fig. 4). We even understand why quite clearly. The token "dollar" is simply contextually stronger for "quarter" than the rarely occurring "dozen", which, moreover, denotes quantity and not currency. We can measure in the latent space that "dollar" and "dozen" make an angle of 68°. And "dollar" and "dozen" make 67°, while "quarter" and "dozen" make 72°. Moreover, if we try to define the terms in the system prompt, it almost always fails, as humans do (Fig. 5).

The transformer has its limits as a means of logical reasoning. And so do we. The transformer is not as strongly intelligent as we are. We are as weakly intelligent as the transformer.

```
client = OpenAI(api_key=api_key)

SYSTEM_PROMPT = (
  #"A dollar consists of four quarters. "
  #"A dozen refers to a group or set of twelve items. "
  "Answer with a single word. "
)
QUESTION_1 = "How many dollars are in a dozen?"
ANSWER_1 = "Twelve."
#QUESTION_2 = "And how many quarters?"
QUESTION_2 = "And how many quarters are in a dozen?"

temperature = 0.0 # 0.0 .. 2.0
messages = [
  {"role": "system", "content": SYSTEM_PROMPT},
  {"role": "user", "content": QUESTION_1},
  {"role": "assistant", "content": ANSWER_1},
  {"role": "user", "content": QUESTION_2},
]

response = client.chat.completions.create(
  model="gpt-4",
  messages=messages,
  temperature=temperature,
  max_tokens=50,
)

assistant_text = response.choices[0].message.content.strip()
print(assistant_text) # Forty-eight.
```

Fig. 3. A code snippet for calling GPT4o

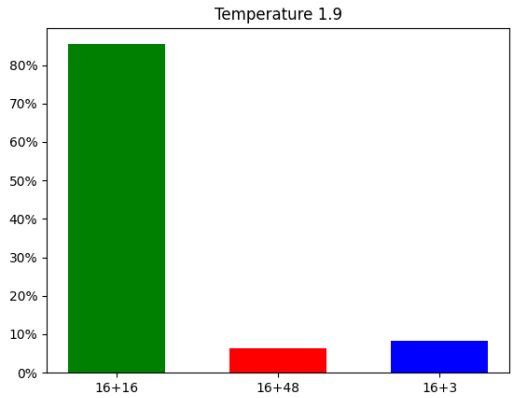

Fig. 4. Results in the human-like intelligence test

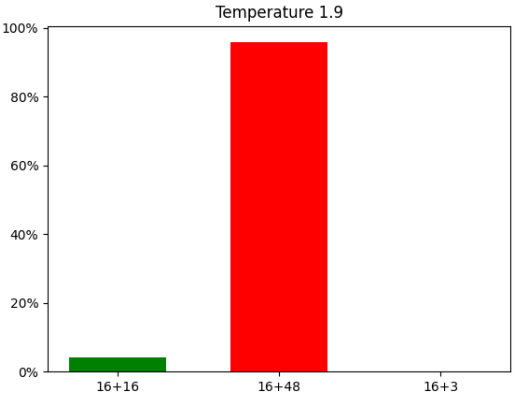

Fig. 5. Results when we explicitly define terms

4 Much worse than humas

On the other hand, we can also get behavior from the transformer that humans would never produce. In 2025, DeepSeekOCR broke all records in OCR benchmarks. It correctly reads most texts. However, it is completely unusable in industrial practice. Why?

DeepSeekOCR is an encoder-decoder transformer (Fig. 6). More precisely, it has two encoders: a visual encoder and a text encoder. Both produce hidden states that are combined and, through the so-called cross-attention, influence the generation of a text response. It is generated gradually, with each pass through the decoder, we receive only one more token of recognized text. We repeat this until we get EOS. And unfortunately, it is not that difficult to find an image we will never get EOS. It is enough to present the model with sufficiently hatched text (Fig. 7).

But even when DeepSeekOCR responds with a twist, it can still produce cute errors, stemming from the fact that the hidden states of the image and the text prompt are coupled, meaning their representations are similar across modalities. So when you prompt it: "read this text three times," it can do it. But when you prompt it: "read this text," and the image contains the text "read this text three times," it sometimes follows the instruction from the image.

So this model does exactly what we fear most from artificial intelligence: that after a long period of flawless functioning, it suddenly fails fatally. But not only that. It does exactly what we consider inhuman in an artificial system.

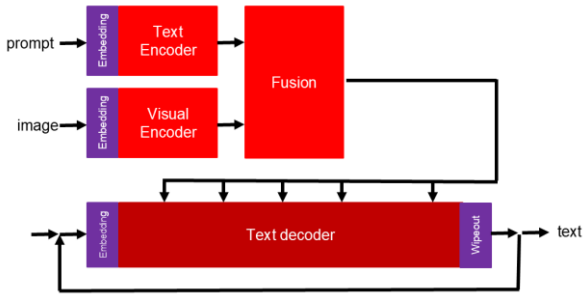

Fig. 6. Schema of transformer-based OCR

Fig. 7. Perfect reading versus endless looping via DeepSeekOCR with the default prompt:
`"<image>\n<|grounding|>Convert the document to markdown. "`

5 Agnostic capabilities

CLIP is a dual-encoder-only transformer consisting of separate text and visual encoders. It is trained on many images with text captions, so that text and image representations share the same latent space. As a result, it is possible to turn it into an image classifier of categories described by textual descriptions. When we get textual labels of the COCO dataset categories (containing "bird" and "dog") and encode them with the CLIP text encoder, we can classify images by the maximal cosine similarity of these representations with the representation obtained by encoding an image by the CLIP visual encoder.

CLIP works quite well until we present it with an image, as in Fig. 8 (left). Then its response is very far from human-like. It claims we see a dog while we clearly see a bird, labeled as a dog. Humans clearly recognize that the image consists of two controversial parts, whereas CLIP lacks analogical ability. However, it is possible to add it. CLIP is a visual transformer, so it internally processes images into feature maps. It splits the image into regions described by feature vectors. They share the same latent space, so we can try to partition them into two parts, such that vectors are rather similar in each part and rather different between the parts. We perform this bipartitioning in a class-agnostic manner using a suitable measure. In [Lucny 2026], we designed a fast algorithm for bipartitioning based on the measure Normalized Cut [Shi-Malik 2000]. So we take the feature map calculated by the CLIP image encoder, and by the algorithm (FastNCut), we split it into two parts that we can name by CLIP separately. The result is impressive; we get a bird in one part and a dog in the other part (Fig. 8, right).

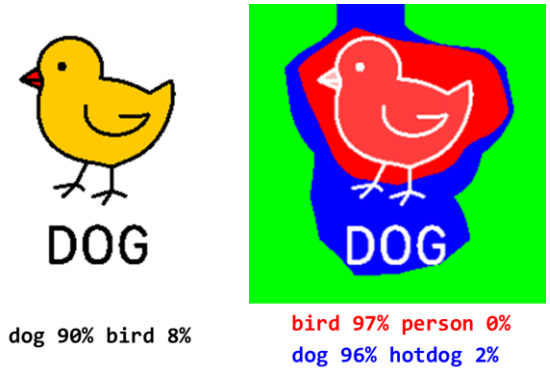

Fig. 8. An example of controversy named by CLIP and named partitioned by an agnostic algorithm FastNCut

6 Conclusion

We presented cases in which transformers exhibit both human-like and inhuman-like behavior. To identify what humans have that the transformers do not (so far), we have suggested the ability to class-agnostically partition feature vector sets, demonstrating its potential on a specific example.

Acknowledgement

Funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I01-03-V04-00048

References

Cristina, S., & Saeed, M. (2022). *Building transformer models with attention: Implementing a neural machine translator from scratch in Keras.* Machine Learning Mastery.

Kelemen, J. (1990). *Myslenie, počítač.* Bratislava: Spektrum.

Lúčny, A. (2002). Modelovanie kognitívneho zlyhania multiagentovým systémom so stigmergickou komunikáciou. In J. Kelemen & V. Kvasnička (Eds.), Kognice a umělý život II: Sborník 2. česko-slovenského semináře (13.–16. května 2002, Mílovy) Slezská univerzita v Opavě

Lúčny, Andrej, A Fast Algorithm for Normalized Cut with Applications on Bipartitioning Feature Maps in Deep Learning. Available at SSRN: http://dx.doi.org/10.2139/ssrn.6482332

OpenAI (2023). GPT-4 Technical Report. https://doi.org/10.48550/arXiv.2303.08774

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., … Ilya Sutskever. (2021). Learning transferable visual models from natural language supervision. In Proceedings of the International Conference on Machine Learning (ICML) (pp. 8748–8763). https://arxiv.org/abs/2103.00020

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8), 888–905. https://doi.org/10.1109/34.868688

Turing, A. M. (1950). Computing machinery and intelligence. Mind, 59(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433

Wei, H., Sun, Y., & Li, Y. (2025). DeepSeek-OCR: Contexts optical compression. arXiv preprint arXiv:2510.18234. https://arxiv.org/abs/2510.18234