Tool Invocation – Demonstrating the Marvel of GPT's Flexibility

30 Jan 2024

I know it has to be true – The magic of human-level cognition isn't the result of a brain in which every single piece is perfectly tuned for its very special and isolated task. Instead, there has to be some simple "principle of computation" that is repeated throughout the brain; some component that, when replicated throughout the brain, gives rise to this emergent property we call *intelligence*. Sure, we have a motor cortex, specialized to control movement. We have a visual cortex, specialized to process information from our eyes. And we have a pre-frontal cortex that assimilates all this information together and plans what to do next – "That's a snake! Don't step on it!" But there is evidence that the base circuitry that makes up all these modules is actually quite general. At every point on the neocortex, you see basically the same pattern - an interwoven computational “component” composed of 6 layers of stacked neurons. At every portion of our cortex this pattern is, with very little modification, repeated. Besides the similar morphology, there is other evidence that the computational components are general. In several *really weird* brain rewiring studies, they have redirected visual input to the auditory pathway and shown that animals can compensate quite well - effectively “seeing” with the part of their brain that was rightfully intended to hear stuff! (Mice (Lyckman et al., 2001), ferrets (Sur et al., 1988; Roe et al., 1990; Roe et al., 1992; Roe et al., 1993; Sharma et al., 2000), and hamsters (Schneider, 1973; Kalil & Schneider, 1975; Frost, 1982; Frost & Metin, 1985).)

I think that the transformer architecture present in OpenAI’s GPTs – and in particular, the attention mechanism – is our first step toward identifying a digital analog of the basic, general computational component that I think must exist in our brains.

Tool Invocation in OpenAI’s GPT models.

To demonstrate the generality, let’s look at how the OpenAI models are implementing prompt crafting. Let’s say that we have a function that determines the temperature of a specified city:

```
{
    "type": "function",
    "function": {
        "name": "get_temperature",
        "description": "returns the current temperature of a specifi
ed city",
        "parameters": {
            "type": "object",
            "properties": {
                "city": {
                    "type": "string",
                    "description": "get the temperature for this cit
y",
                },
                "celsius": {
                    "type": "boolean",
                    "description": "the system of measurement to use - defaults to `false` (Fahrenheit)",
                },
            },
            "required": ["city"],
        },
    },
}
```

Well after using *various persuasive means* (😈), I was able to get the model to tell me how tool invocations are presented in the prompt. Let's say that the user is looking for the current temperature in Berlin. In this case, the conversation and the subsequent tool invocation would be represented in ChatML like so:

```
<|im_start|>user
You know what? I love the metric system. All other units of measurement are trash.<|im_end|>
<|im_start|>assistant
Uhhh... ok. So?<|im_end|>
<|im_start|>user
I wish I knew what temperature it was now in Berlin<|im_end|>
<|im_start|>assistant to=function.get_temperature
{
  "city": "Berlin, Germany",
  "celsius": true
}
```

Let's take a look at OpenAI's clever approach here. And in particular, look at how general purpose the model must be to do what it just did there.

Let's start by looking at the start of the assistant response `<|im_end|>\n<|im_start|>assistant`. OpenAI always sticks this at the end of their prompt so that the model knows to begin the assistant response rather than, say, hallucinating more text from the user. This, while clever in its own right, doesn't illustrate the generality of GPT, but it gives us a good starting point. It is at this point that the GPT model will act as a Swiss army knife of classification and prediction models. In the next 20-30 tokens the model *is* at least 5 different inference models packed into one. We'll cover this in order.

The first tokens predicted after `<|im_start|>assistant` are either `\n` *or* `to=function.` . If the model predicts `\n` (as in the first assistant message above) then this conditions the model to continue to completion as a natural language response that is consistent with the user's message. But, if the model predicts `to=function.` then this conditions the model to proceed with all of the subsequent predictions necessary to make a tool invocation. **Here, the model acts as a classification algorithm predicting whether or not a tool should be invoked.**

The next set of tokens corresponds to the name of the tool being evoked – in this case `get_temperature` . **Here, the model acts as a classification algorithm predicting which tool is appropriate based upon the context of the messages.** All the more interesting, this is not a model that has been fine-tuned with a particular set of messages in mind. Rather the model infers from the tool descriptions that `get_temperature` is appropriate in this case. We could have added several other tools and the model would have probably still done a good job of picking this particular tool for the task at hand.

Next, the model starts generating JSON. After the opening braces, the model generates the names of the arguments. This was a particularly clever trick on the part of OpenAI. In my own earlier experimentation with building tool usage from scratch using the completion API, I made everything look like Python functions with positional arguments. But then either 1) the model would complete the functions using positional arguments or 2) the model would attempt to use keyword arguments. In the first case, the model doesn't get the extra token or two worth of time to write the argument name and "contemplate" what a good argument value should be. In the second case, I would have had trouble parsing the response because I was only anticipating positional arguments! But for the actual OpenAI tools implementation, because of the way the tool definition is represented in the system message (a post I'll write at another time) and because of OpenAI's fine-tuning, we will *always* get the name of the arguments. Further, notice that in some cases, not all of the names need to be specified. In this case, the `celsius` argument is not required. However, the model has chosen to include it because of the user's aforementioned love of the metric system. **Therefore, in correctly generating the names of required arguments, the model acts as a classification algorithm that predicts tool argument names.**

Finally, after each argument name, the model generates tokens representing the type and value corresponding to those arguments. **Argument value prediction is yet another form of inference.** And it's perhaps the most general inference of all the above – it is inferring a very specific value to solve this problem: "based on whatever is in this conversation, identify the best value for some argument X used in some function Y as described in the functions list at the start of the conversation".

Multi-Tool Invocation

The point I wish to make is born out in the details already listed above. But for the sake of completeness, there is one more morsel to discuss regarding tool completion. What does the model do if it intends to invoke multiple tools simultaneously? Consider a user who wants to know the temperature of two different cities:

```
<|im_start|>user
So for some dumb reason, I want to know the temperature in London an
d in Madrid.<|im_end|>
<|im_start|>assistant to=multi_tool_use.parallel
{
  "tool_uses": [{
    "recipient_name": "functions.get_temperature",
    "parameters": {"city":"London, England"}
  }, {
    "recipient_name": "functions.get_temperature",
    "parameters": {"city":"Madrid, Spain"}
  }]
}<|im_end|>
```

This extends our conversation above with another split point in the decision pipeline. After the model has decided that some tool needs to be invoked `<|im_start|>assistant to=` It needs to either determine whether we just need a single tool, conditioned by the tokens `function.` or multiple tools conditioned by the tokens `multi_tool_use.parallel` . (If you're interested I can write a follow-up post showing how this hidden `parallel` function is represented in the system message. I got that out of the model too! 😈 😈!) **So here, the model acts as a classifier determining whether or not multi-tool invocation is required.**

In Summary

In the conversation above, I have demonstrated that in the span of 20 or 30 tokens, the GPT is general enough to function as at least 5 different inference algorithms. There are still more interesting observations to make. For one, look at the order of these inference problems. Without exception they go from most general to most specific: "Need a tool?" ➡ "Several?" ➡ "Which tool?" ➡ "Which args?" ➡ "What values?". If this was unintentional then it was fortunate; if it was intentional, then it was clever to say the least, because this progression conditions the model to think convergently and narrow down to the best answer using a broad-strokes to narrow-details pattern of thinking. Moreover, the model has only been fine-tuned on the first couple of inference problems – determining whether or not a tool should be invoked at all, and determining how whether more than one can be invoked at once. The remainder of the classification and inference problems are conditioned by the description of the tools in the system message and by the text in the conversation. The model is therefore acting like a just-in-time inference algorithm rather than a model that was made for that specific purpose. Consider what we would have done in the olden days (2 years ago) to achieve the same effect. We would have had to build different bespoke models for each point in this pipeline, and the models would have had to be fine-tuned for the specific set of tools that were required for that task. Instead, we get a "just-in-time" inference algorithm *and* the specification for that just-in-time algorithm is defined in the same medium as the algorithms themselves are implemented in – a mechanism that can attend to previous tokens and predict the likelihood of subsequent tokens.

Attention and token prediction. It's amazing that such a simple mechanism has been able to accomplish so much. But for that very reason I think that this mechanism is approaching the simple, general computational component that I've looked for and dreamed about all of my career. We're just one tweak away from *something big*.

Related Posts

Julia ... a short story 15 Jul 2024

The Primary Objective of Software Design: Minimizing Total Cognitive Load 20 May 2023

Similarity Search for Grouped Content (Teaser) 30 Apr 2023