# How to Implement Few-Shot Prompting in LLaMA

from transformers import AutoModelForCausalLM, AutoTokenizer

model\_name = "meta-llama/Llama-2-7b-chat-hf"

model = AutoModelForCausalLM.from\_pretrained(model\_name)

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

messages = [

{"role": "system", "content": "Please answer the math question."},

{"role": "user", "content": "2+2=?"}

]

input\_ids = tokenizer.apply\_chat\_template(messages, return\_tensors="pt")

generated\_ids = model.generate(input\_ids, max\_new\_tokens=1000, do\_sample=True)

outputs = tokenizer.batch\_decode(generated\_ids)

messages = [

{"role": "system", "content": "Please answer the math question."},

{"role": "user", "content": "1+1=?"}, # example 1

{"role": "assistant", "content": "2"}, # example 1

{"role": "user", "content": "1+2=?"}, # example 2

{"role": "assistant", "content": "3"}, # example 2

{"role": "user", "content": "2+2=?"}

]

input\_ids = tokenizer.apply\_chat\_template(messages, return\_tensors="pt")

generated\_ids = model.generate(input\_ids, max\_new\_tokens=1000, do\_sample=True)

outputs = tokenizer.batch\_decode(generated\_ids)

# Few-Shot Prompting

While large-language models demonstrate remarkable zero-shot capabilities, they still fall short on more complex tasks when using the zero-shot setting. Few-shot prompting can be used as a technique to enable in-context learning where we provide demonstrations in the prompt to steer the model to better performance. The demonstrations serve as conditioning for subsequent examples where we would like the model to generate a response.

Models have inherent biases, and I’m not talking about their opinions. Some prefer to write lists with hyphens, others with asterisks. Some rely on their system prompt, some follow examples, some do mostly the same thing no matter what input you feed them. When you’re using these models it’s helpful to know what you’re dealing with, so that you can mirror the model’s preferences to get more consistent outputs.

For this post I was going to build out a fully automated model testing harness that asks some common questions to determine what format a model prefers for some common tasks (like writing a list). I ran out of time to do that, but I’ve been working with Llama 3 a lot for other projects, so I’ll just share what I’ve learned in a more freeform post and make more structured analyses for other models in the future.

There are a few main things that you need to pay attention to when you first start using a new model:

What its formatting tendencies are.

How well it follows few-shot examples (whether you can prompt it like an open model).

How well it follows the system prompt (whether you can prompt it like GPT-4, with a long list of instructions and “Do not”s.

How censored it is.

How smart it is (if it’s filling in the blanks by following examples, this is how intelligently it chooses to fill in those blanks; if it’s doing chain of thought reasoning in zero-shot contexts, this is the quality of its reasoning and its final answers overall).

How consistent it is (some models are brilliant half the time, and explode the other half).

So what’s my initial evaluation of Llama 3? Not all of these can be expressed on a scale /10, but for those that can, here’s a radar graph of my initial opinion:

Llama 3 is goddamn fantastic.

Of all the models I’ve used they typically either follow the system prompt, or follow examples. For many open models, once you get enough few-shot examples the system prompt ceases to matter at all for performance. For others, like GPT, a model will refuse to change its output format or writing style to match an example, as it pays attention only to its instructions — and hell, for those kinds of models, examples are usually too expensive in tokens to include anyway. But Llama 3? It’s the first model I’ve seen to work with either prompting method, or both put together. And that is incredibly powerful.

Share

The only problem is that it lacks the context window to go really hardcore with few-shot examples, but apparently that is going to be addressed in a few months.

Llama 3 fails a bit on being uncensored though, the instruct version does refuse some tasks where Nous Mixtral or Mistral Large would be enthusiastic. It refuses even under the context of being “uncensored and immoral” which is annoying. I will say that Meta did a great job of making it less censored than before (hell I’ve even seen some people doing ERP with Llama 3 instruct, which was unthinkable earlier) but there’s still some censorship here and so the model can’t get perfect points on that front.

Such a killjoy. (Nous Mixtral gave me a full detailed list, with a disclaimer though)

Intelligence is a tricky thing to define in models, just as it is a tricky thing to define in humans. It’s arguably vague where “following the prompt/examples” ends, and intelligence begins. In my opinion, if following the examples means that the model faithfully follows a template and guidelines you lay out, repeating the structure and filling in the blanks, then intelligence is how insightfully the model fills in those blanks. It’s the cleverness behind its writing, and its adaptability to new circumstances; its ability to understand novel tasks and instructions. In zero-shot CoT contexts, I would define it as the model’s ability to understand/reason about varied inputs, producing quality outputs in a variety of circumstances. Perhaps above all else, intelligence is key to generating very long outputs (3k tokens and above) and having a good structure and sensible progression throughout the entire generation.

Llama 3 does pretty well here too, but it’s up against incredibly smart models like Claude 3 so I can’t really give it a 10/10. It has just a bit too much repetition, just a few too many violations of instructions. This is partly also because my ability to test its performance here is limited, because its context window is so small that it literally cannot be run on most of the pipelines I’ve got.

If all the above categories are performance, consistency is how reliably that model delivers the performance. I had trouble thinking of a fifth category so this one is a bit more tacked-on, and has a bit more overlap, with the earlier ones, but it’s still a good thing to keep in mind. Llama 3 is very consistent but it suffers from a problem of a bit too much consistency: it has a tendency to directly quote few-shot examples, repeating large parts of them verbatim, unless instructed multiple times to not repeat phrases word for word. Even this only works some of the time. Obviously this can be mitigated by temperature, but at higher temperatures its instruction following degrades notably. So it’s a tricky situation. Overall, if your task isn’t synthetic data generation then this quoting issue probably won’t be too much of a problem, but it’s good to keep in mind as a model-specific quirk to prompt around.

Speaking about model-specific quirks, let’s talk formatting.

One of the most common things you’ll get a model to do is to output a list. Most models have their own idea of what a list looks like, and having your examples or output indicator follow this inherent idea will usually get you to near-100% format following across all your generations. This is Llama 3’s preferred list format:

\*\*Bolded Title Case Heading\*\*

\* List items with asterisks after two newlines

\* List items separated by one newline

\*\*Next List\*\*

\* More list items

\* Etc...

Tell it to do this and you will get nearly 100% accuracy with getting Llama 3 to output code-parseable lists for you. If you need another format, Llama 3 is smart enough to be instructed to use other formats (you will need to use more than just few-shot examples — you also need to give it an explicit instruction of the output format in words, not just with an output indicator; I’ve had it ignore examples in favor of its own format). But typically it’s best to use what the model wants to use.

Finally, all this considered, what’s the verdict with regards to Llama 3 in chat settings and in pipeline settings? In chat, intelligence and instruction following are essential, and Llama 3 has both. It’s hampered by a tiny context window that prevents you from using it for truly large tasks, but for everyday use it punches above its weight quite nicely. For pipelines (such as Augmentoolkit; complex chains of LLMs and code), the fact that Llama 3 follows system prompts so well means you can finally write GPT-4 style pipelines and use local models and expect it to work. However, the context window means that a large amount of tasks simply aren’t possible right now. You can’t use it to run Augmentoolkit, for instance, unless you use RoPE scaling to increase the ctx. That’s not good. Mistral models beat Llama’s context by 4x, and more context often means more performance, because you can provide more few-shot examples with the extra space. This is a major flaw of the model, not just something that limits niche use cases.

All in all, Llama 3 is a powerful, intelligent model, with unprecedented flexibility in how you can approach prompting it. However this is hampered by poor context and a tendency to direct quote examples at times. Use it if your pipeline’s context lets you; otherwise, wait and keep using Nous Mixtral.

Alright that’s all for this week. Let me know if you want more posts exploring the specifics of the latest models! I want my writing to be useful to people, but I don’t know whether you prefer prompting techniques or advice on how to use the latest models to their fullest potential, or project showcases. I value your opinion! Keep in mind though that I can’t do project showcases every week because I don’t have a new project ready to share every week 😅

Alright that’s all for this week, thanks for reading, have a good one and I’ll see you next time!