Deploy Llama 2 7B/13B/70B on Amazon SageMaker

#GENERATIVEAI #HUGGINGFACE #SAGEMAKER #LLM

August 7, 2023
10 min read
View Code

LLaMA 2 is the next version of the LLaMA. It is trained on more data - 2T tokens and supports context length window upto 4K tokens. Meta fine-tuned conversational models with Reinforcement Learning from Human Feedback on over 1 million human annotations.

In this blog you will learn how to deploy Llama 2 model to Amazon SageMaker. Your are going to use the Hugging Face LLM DLC is a new purpose-built Inference Container to easily deploy LLMs in a secure and managed environment. The DLC is powered by Text Generation Inference (TGI) a scalelable, optimized solution for deploying and serving Large Language Models (LLMs). The Blog post also includes Hardware requirements for the different model sizes.

In the blog will cover how to:

1. Setup development environment
2. Retrieve the new Hugging Face LLM DLC
3. Hardware requirements
4. Deploy Llama 2 to Amazon SageMaker
5. Run inference and chat with the model
6. Clean up

Lets get started!

1. Setup development environment

You are going to use the `sagemaker` python SDK to deploy Llama 2 to Amazon SageMaker. You need to make sure to have an AWS account configured and the `sagemaker` python SDK installed.

```
!pip install "sagemaker>=2.175.0" --upgrade --quiet
```

If you are going to use Sagemaker in a local environment. You need access to an IAM Role with the required permissions for Sagemaker. You can find here more about it.

```
import sagemaker
import boto3
sess = sagemaker.Session()
# sagemaker session bucket -> used for uploading data, models and logs
# sagemaker will automatically create this bucket if it not exists
sagemaker_session_bucket=None
if sagemaker_session_bucket is None and sess is not None:
    # set to default bucket if a bucket name is not given
    sagemaker_session_bucket = sess.default_bucket()

try:
    role = sagemaker.get_execution_role()
except ValueError:
    iam = boto3.client('iam')
    role = iam.get_role(RoleName='sagemaker_execution_role')['Role']['Arn']
```

```
sess = sagemaker.Session(default_bucket=sagemaker_session_bucket)

print(f"sagemaker role arn: {role}")
print(f"sagemaker session region: {sess.boto_region_name}")
```

2. Retrieve the new Hugging Face LLM DLC

Compared to deploying regular Hugging Face models you first need to retrieve the container uri and provide it to our `HuggingFaceModel` model class with a `image_uri` pointing to the image. To retrieve the new Hugging Face LLM DLC in Amazon SageMaker, you can use the `get_huggingface_llm_image_uri` method provided by the `sagemaker` SDK. This method allows us to retrieve the URI for the desired Hugging Face LLM DLC based on the specified `backend`, `session`, `region`, and `version`. You can find the available versions here

```
from sagemaker.huggingface import get_huggingface_llm_image_uri

# retrieve the llm image uri
llm_image = get_huggingface_llm_image_uri(
  "huggingface",
  version="0.9.3"
)

# print ecr image uri
print(f"llm image uri: {llm_image}")
```

3. Hardware requirements

Llama 2 comes in 3 different sizes - 7B, 13B & 70B parameters. The hardware requirements will vary based on the model size deployed to SageMaker. Below is a set up minimum requirements for each model size we tested.

Note: We haven't tested GPTQ models yet.

Model	Instance Type	Quantization	# of GPUs per replica
Llama 7B	`(ml.)g5.2xlarge`	`-`	1
Llama 13B	`(ml.)g5.12xlarge`	`-`	4
Llama 70B	`(ml.)g5.48xlarge`	`bitsandbytes`	8
Llama 70B	`(ml.)p4d.24xlarge`	`-`	8

Note: Amazon SageMaker currently doesn't support instance slicing meaning, e.g. for Llama 70B you cannot run multiple replica on a single instance.

These are the minimum setups we have validated for 7B, 13B and 70B LLaMA 2 models to work on SageMaker. In the coming weeks, we plan to run detailed benchmarking covering latency and throughput numbers across different hardware configurations. We are currently not recommending deploying Llama 70B to g5.48xlarge instances, since long request can timeout due to the 60s request timeout limit for SageMaker. Use `p4d` instances for deploying Llama 70B it.

It might be possible to run Llama 70B on `g5.48xlarge` instances without quantization by reducing the `MAX_TOTAL_TOKENS` and `MAX_BATCH_TOTAL_TOKENS` parameters. We haven't tested this yet.

4. Deploy Llama 2 to Amazon SageMaker

To deploy meta-llama/Llama-2-13b-chat-hf to Amazon SageMaker you create a `HuggingFaceModel` model class and define our endpoint configuration including the `hf_model_id`, `instance_type` etc. You will use a `g5.12xlarge` instance type, which has 4 NVIDIA A10G GPUs and 96GB of GPU memory.

Note: This is a form to enable access to Llama 2 on Hugging Face after you have been granted access from Meta. Please visit the Meta website and accept our license terms and acceptable use policy before submitting this form. Requests will be processed in 1-2 days.

```
import json
from sagemaker.huggingface import HuggingFaceModel

# sagemaker config
instance_type = "ml.p4d.24xlarge"
number_of_gpu = 8
health_check_timeout = 300

# Define Model and Endpoint configuration parameter
config = {
  'HF_MODEL_ID': "meta-llama/Llama-2-70b-chat-hf", # model_id from hf.co/models
  'SM_NUM_GPUS': json.dumps(number_of_gpu), # Number of GPU used per replica
  'MAX_INPUT_LENGTH': json.dumps(2048),  # Max length of input text
  'MAX_TOTAL_TOKENS': json.dumps(4096),  # Max length of the generation (including input te
  'MAX_BATCH_TOTAL_TOKENS': json.dumps(8192),  # Limits the number of tokens that can be pr
  'HUGGING_FACE_HUB_TOKEN': "<REPLACE WITH YOUR TOKEN>"
  # ,'HF_MODEL_QUANTIZE': "bitsandbytes", # comment in to quantize
}

# check if token is set
assert config['HUGGING_FACE_HUB_TOKEN'] != "<REPLACE WITH YOUR TOKEN>", "Please set your Hug

# create HuggingFaceModel with the image uri
llm_model = HuggingFaceModel(
  role=role,
  image_uri=llm_image,
  env=config
)
```

After you have created the `HuggingFaceModel` you can deploy it to Amazon SageMaker using the `deploy` method. You will deploy the model with the `ml.g5.12xlarge` instance type. TGI will automatically distribute and shard the model across all GPUs.

```
# Deploy model to an endpoint
# https://sagemaker.readthedocs.io/en/stable/api/inference/model.html#sagemaker.model.Model.
llm = llm_model.deploy(
  initial_instance_count=1,
  instance_type=instance_type,
  container_startup_health_check_timeout=health_check_timeout, # 10 minutes to be able to l
)
```

SageMaker will now create our endpoint and deploy the model to it. This can takes a 10-15 minutes.

5. Run inference and chat with the model

After our endpoint is deployed you can run inference on it. You will use the `predict` method from the `predictor` to run inference on our endpoint. You run inference with different parameters to impact the generation. Parameters can be defined as in the

`parameters` attribute of the payload. As of today the TGI supports the following parameters:

- `temperature`: Controls randomness in the model. Lower values will make the model more deterministic and higher values will make the model more random. Default value is 1.0.
- `max_new_tokens`: The maximum number of tokens to generate. Default value is 20, max value is 512.
- `repetition_penalty`: Controls the likelihood of repetition, defaults to `null`.
- `seed`: The seed to use for random generation, default is `null`.
- `stop`: A list of tokens to stop the generation. The generation will stop when one of the tokens is generated.
- `top_k`: The number of highest probability vocabulary tokens to keep for top-k-filtering. Default value is `null`, which disables top-k-filtering.
- `top_p`: The cumulative probability of parameter highest probability vocabulary tokens to keep for nucleus sampling, default to `null`
- `do_sample`: Whether or not to use sampling ; use greedy decoding otherwise. Default value is `false`.
- `best_of`: Generate best_of sequences and return the one if the highest token logprobs, default to `null`.
- `details`: Whether or not to return details about the generation. Default value is `false`.
- `return_full_text`: Whether or not to return the full text or only the generated part. Default value is `false`.
- `truncate`: Whether or not to truncate the input to the maximum length of the model. Default value is `true`.
- `typical_p`: The typical probability of a token. Default value is `null`.
- `watermark`: The watermark to use for the generation. Default value is `false`.

You can find the open api specification of the TGI in the swagger documentation

The `meta-llama/Llama-2-13b-chat-hf` is a conversational chat model meaning you can chat with it using the following prompt:

```
<s>[INST] <<SYS>>
{{ system_prompt }}
<</SYS>>

{{ user_msg_1 }} [/INST] {{ model_answer_1 }} </s><s>[INST] {{ user_msg_2 }} [/INST]
```

We create a small helper method `build_llama2_prompt`, which converts a List of "messages" into the prompt format. We also define a `system_prompt` which is used to start the conversation. You will use the `system_prompt` to ask the model about some cool ideas to do in the summer.

```
def build_llama2_prompt(messages):
    startPrompt = "<s>[INST] "
    endPrompt = " [/INST]"
    conversation = []
    for index, message in enumerate(messages):
        if message["role"] == "system" and index == 0:
            conversation.append(f"<<SYS>>\n{message['content']}\n<</SYS>>\n\n")
        elif message["role"] == "user":
            conversation.append(message["content"].strip())
        else:
            conversation.append(f" [/INST] {message['content'].strip()}</s><s>[INST] ")

    return startPrompt + "".join(conversation) + endPrompt

messages = [
```

```
  { "role": "system","content": "You are a friendly and knowledgeable vacation planning assi
]
```

Lets see, if Clara can come up with some cool ideas for the summer.

```
# define question and add to messages
instruction = "What are some cool ideas to do in the summer?"
messages.append({"role": "user", "content": instruction})
prompt = build_llama2_prompt(messages)

chat = llm.predict({"inputs":prompt})

print(chat[0]["generated_text"][len(prompt):])
```

Now, run inference with different parameters to impact the generation. Parameters can be defined as in the `parameters` attribute of the payload.

```
# hyperparameters for llm
payload = {
  "inputs":  prompt,
  "parameters": {
    "do_sample": True,
    "top_p": 0.6,
    "temperature": 0.9,
    "top_k": 50,
    "max_new_tokens": 512,
    "repetition_penalty": 1.03,
    "stop": ["</s>"]
  }
}

# send request to endpoint
response = llm.predict(payload)

print(response[0]["generated_text"][len(prompt):])
```

6. Clean up

To clean up, you can delete the model and endpoint.

```
llm.delete_model()
llm.delete_endpoint()
```

Conclusion

Deploying Llama 2 on Amazon SageMaker provides a scalable, secure way to leverage LLMs. With just a few lines of code, the Hugging Face Inference DLC allows everyone to easily integrate powerful LLMs into applications.

Thanks for reading! If you have any questions, feel free to contact me on Twitter or LinkedIn.
