AI for Developers

Treating Open Source AI as a Function

Martin Hickey
Senior Technical Staff Member (STSM)
whoami

- 25+ yr tech career in enterprise and open source software
- Helm core maintainer and TOC member
- Contributor to Kubernetes and Open Telemetry
- Open source developer at IBM
Agenda

AI Background
Open Source AI Frameworks
Demo
In Conclusion
An AI model is a program that has been trained on a set of data to recognize certain patterns or make certain decisions without further human intervention. Models apply different algorithms to relevant data inputs to achieve the tasks, or output, they’ve been programmed for.

Source: https://www.ibm.com/topics/ai-model
Model Journey

– Model Building/ Prototyping:
  • Data:
    – Loading
    – Preparing
  • Algorithm
  • Training
  • Validation
– Hosting/ Serving
– Inference

Image by mikemacmarketing – CC BY 2.0
Generative AI

- Foundation models:
  - Trained on large unlabeled datasets
  - Tuned for different tasks
- Large Language Models (LLMs):
  - General-purpose language understanding and generation
- Generative AI:
  - Uses deep-learning models
  - Generates high-quality text, images, and other content
  - Based on the data the model was trained on
Open Source AI Frameworks
HuggingFace

- AI community based around open source:
  - Libraries
  - Models
  - Data sets
- Expansive catalog of open source AI models
- Hosted AI model service

Image: https://huggingface.co
HuggingFace: Example

- Function:
  - Translate English to French

- Uses:
  - T5 Small model for translation
  - HuggingFace transformers API for loading and inference of the model

- Load and inference the model:
  - $ python model.py

```python
# File name: model.py
from transformers import pipeline

class Translator:
    def __init__(self):
        # Load model
        self.model = pipeline("translation_en_to_fr", model="t5-small")

    def translate(self, text: str) -> str:
        # Run inference
        model_output = self.model(text)

        # Post-process output to return only the translation text
        translation = model_output[0]["translation_text"]

        return translation

translator = Translator()
translation = translator.translate("Hello world!")
print(translation)
```

Source:
https://docs.ray.io/en/latest/serve/getting_started.html @mhickeybot
Ray

- Unified framework for scaling AI and running distributed Python workloads
- Consists of three layers:
  - Runtime: Python, domain-specific set of libraries that provide a scalable and unified toolkit for ML applications
  - Core: Python library to scale Python applications and accelerate machine learning workloads
  - Cluster: Set of worker nodes connected to a common Ray head node for running applications

Image: https://docs.ray.io/en/latest/ray-overview/index.html
Ray: Example

– Wrap model code as follows:
  • Python decorator `serve.deployment`:
    – Converts Python class to a Ray Serve Deployment object
  • Pass parameters in the `serve.deployment` decorator
  • `__call__` method is called on a HTTP request
– Serve the model as HTTP sever using `serve run` command:
  • `$ serve run serve_quickstart:translator_app`
– Inference the model using following Python client code:

```python
# File name: serve_quickstart.py
from starlette.requests import Request

import ray
from ray import serve

from transformers import pipeline

@serve.deployment(num_replicas=2, ray_actor_options={"num_cpus": 0.2, "num_gpus": 0})
class Translator:
    def __init__(self):
        # Load model
        self.model = pipeline("translation_en_to_fr", model="t5-small")

    def translate(self, text: str) -> str:
        # Run inference
        model_output = self.model(text)

        # Post-process output to return only the translation text
        translation = model_output[0]["translation_text"]

        return translation

    async def __call__(self, http_request: Request) -> str:
        english_text = http_request.json()
        return self.translate(english_text)

translator_app = Translator.bind()
```

Source: https://docs.ray.io/en/latest/serve/getting_started.html

```python
import requests
english_text = "Hello world!"

response = requests.post("http://127.0.0.1:8000/", json=english_text)

french_text = response.text

print(french_text)
```
Triton Inference Server

- Streamlines AI inferencing
- Multiple deep learning and machine learning frameworks supported like:
  - TensorRT, TensorFlow, PyTorch, ONNX, Python, etc.
- Supports inference across:
  - cloud, data center, edge and embedded devices
- Processor support:
  - NVIDIA GPUs, x86 and ARM CPU, or AWS Inferentia

Image:
Triton IS: Example (1)

- Wrap model as follows:
  - `TritonPythonModel` class name is mandatory
  - `execute` function is called on HTTP request

```python
import triton_python_backend_utils as pb_utils
import numpy as np
from transformers import pipeline

class TritonPythonModel:
    def initialize(self, args):
        self.model = pipeline("translation_en_to_fr", model="t5-small")

    def execute(self, self, requests):
        responses = []
        for request in requests:
            # Decode the Byte Tensor into Text
            input = pb_utils.get_input_tensor_by_name(request, "text")
            input_string = input.as_numpy()[0].decode()

            # Call the Model pipeline
            model_output = self.model(input_string)
            translation = model_output[0]["translation_text"]

            # Encode the text to byte tensor to send back
            translation_response = pb_utils.InferenceResponse()
            output_tensors= [
                pb_utils.Tensor("translation_text",
                np.array([translation.encode()]),
            )
            responses.append(translation_response)
        return responses

    def finalize(self, self, args):
        self.model = None
```

Source: https://www.inferless.com/learn/nvidia-triton-inference-inferless

@mhickeybot
Bootstrap for model serving:

- Config file named `config.pbtxt`:
  - Describes the model input/output, name, backend
- Model directory as follows:

```python
name: "translate"
backend: "python"
input [  
  
  
  
  
  
  name: "text"
data_type: TYPE_STRING
dims: [-1]
  
  
  
  
  
  ]
output [  
  
  
  
  
  
  name: "translation_text"
data_type: TYPE_STRING
dims: [-1]
  
  
  
  
  
  ]
instance_group [  
  
  
  
  
  
  {
    kind: KIND_GPU
  }
  
  
  
  
  
  ]
```

Source: https://www.inferless.com/learn/nvidia-triton-inference-inferless
Serve the model as follows:

- Run the Triton Inference Server container:

- Add the model directory to the Triton Inference Server container

- Start the Triton HTTP server (in the container):
  - `tritonserver --model-repository `pwd`/models`

- Inference the model with HTTP request to the server

```bash
curl --location --request POST 'http://<<IP-Address>>/v2/models/translate/infer' \
   --header 'Content-Type: application/json' \
   --data-raw '{
     "inputs": [
       {
         "name": "text",
         "shape": [1],
         "datatype": "BYTES",
         "data": ["Hello world!"]
       }
     ]
   }
'
```
Demo
In Conclusion
Final Thoughts

Models are programs which achieve a task or generate an output.

Model interfacing is becoming more programmer centric and can be called like any other API.

There are multiple open source AI frameworks available to help managing models.
Thank you.

Martin Hickey
Developer

Twitter: @mhickeybot