AI for Developers

—

Treating Open Source AI as a Function

IBM **Developer**

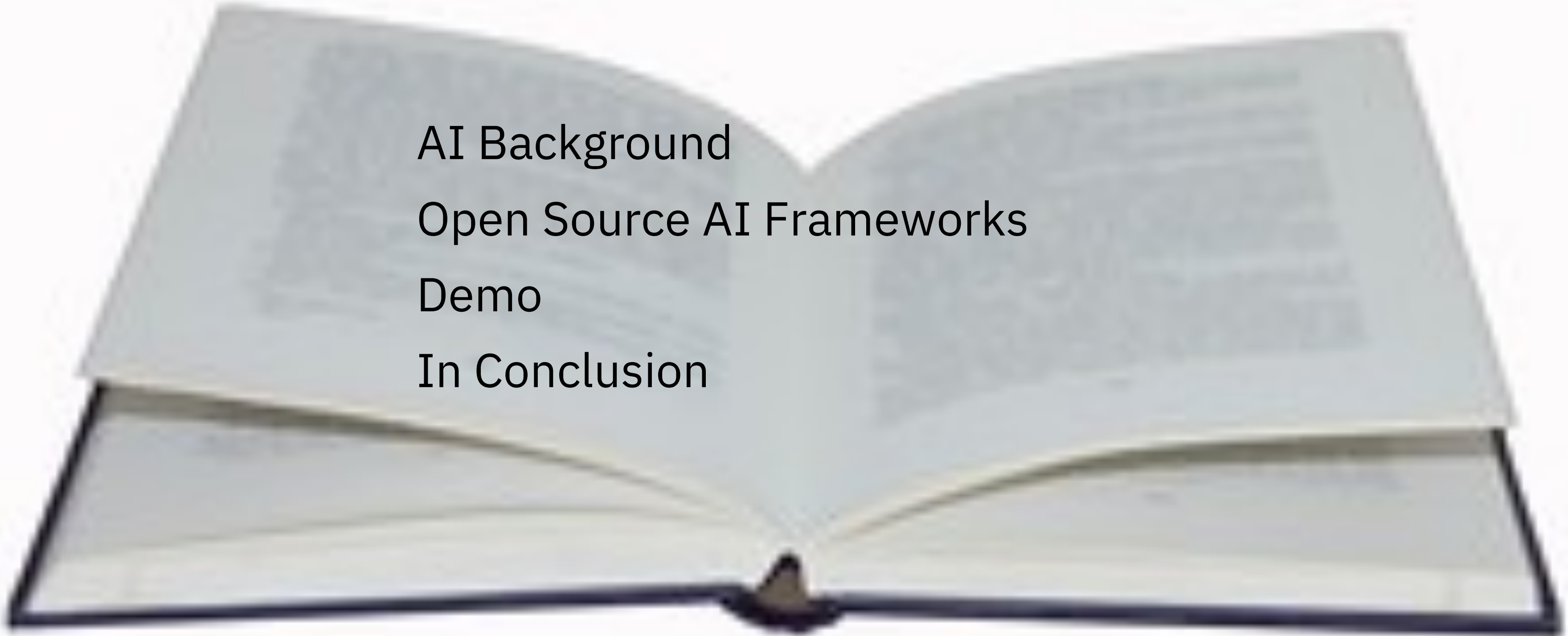
Martin Hickey
Senior Technical Staff Member (STSM)

IBM

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

Photo by pongo – CC BY-NC 2.0

AI Background

What is an AI Model?

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**

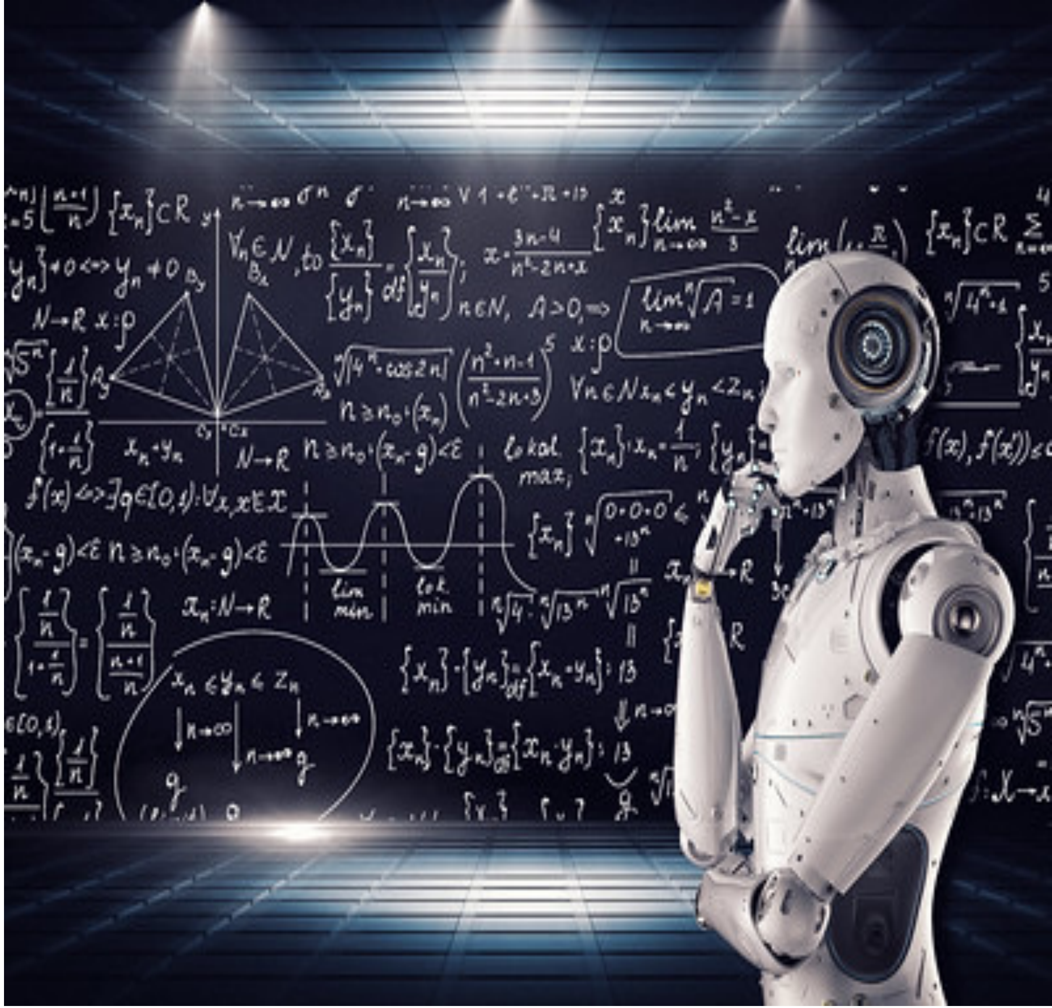


Image by mikemacmarketing – CC BY 2.0

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

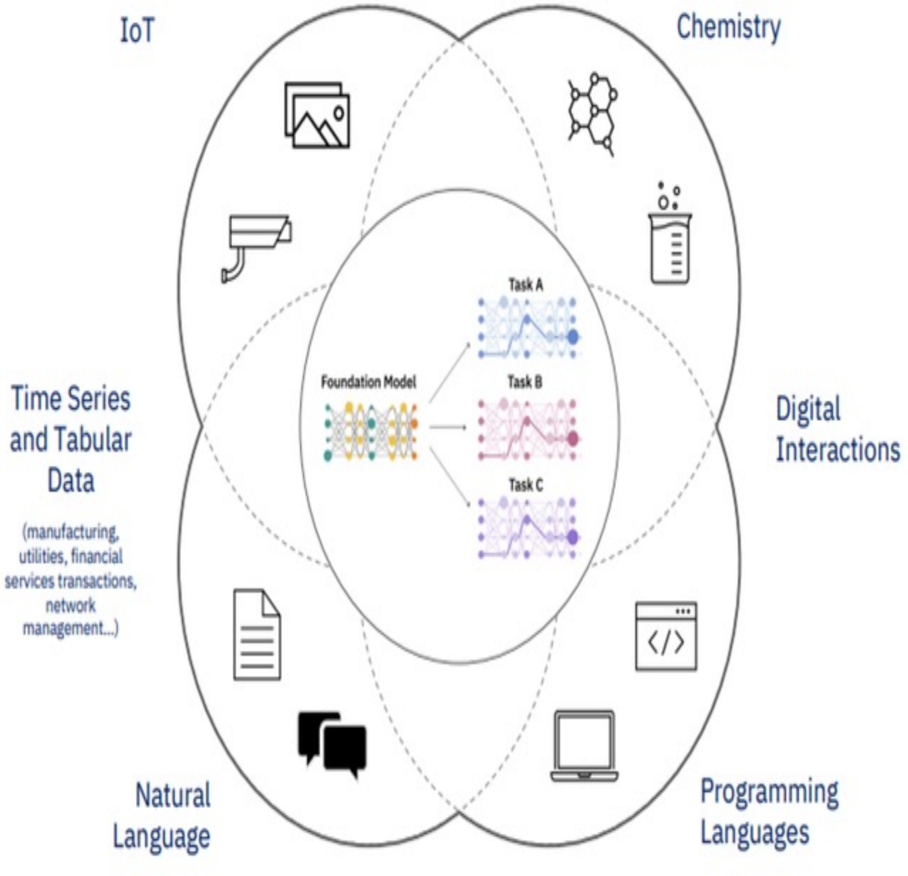

IoT
Chemistry
Time Series and Tabular Data
(manufacturing, utilities, financial services transactions, network management...)
Foundation Model
Task A
Task B
Task C
Digital Interactions
Natural Language
Programming Languages

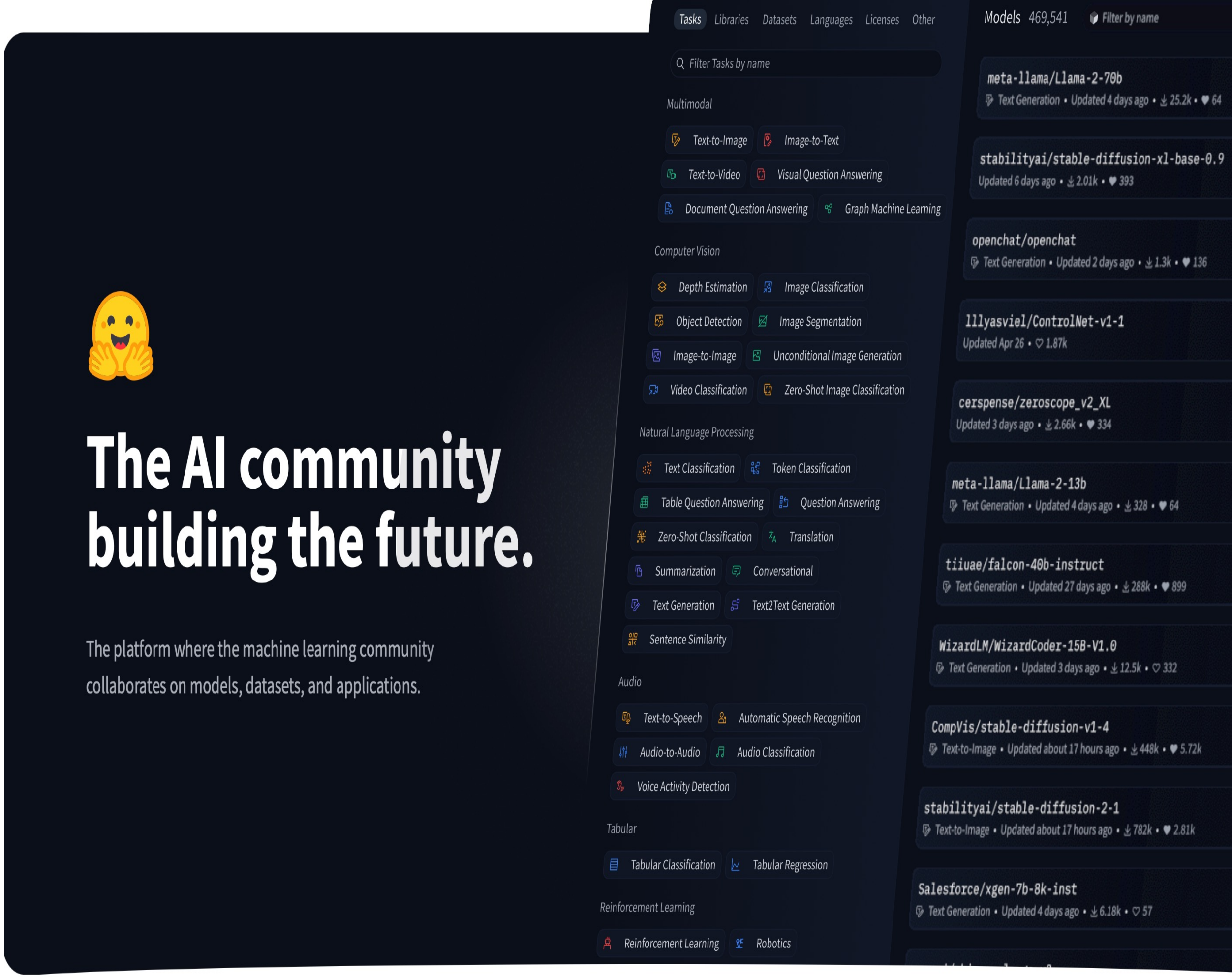
Image: IBM

@mhickeybot

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



🤗

The AI community building the future.

The platform where the machine learning community collaborates on models, datasets, and applications.

Multimodal

- Text-to-Image
- Image-to-Text
- Text-to-Video
- Visual Question Answering
- Document Question Answering
- Graph Machine Learning

Computer Vision

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation
- Image-to-Image
- Unconditional Image Generation
- Video Classification
- Zero-Shot Image Classification

Natural Language Processing

- Text Classification
- Token Classification
- Table Question Answering
- Question Answering
- Zero-Shot Classification
- Translation
- Summarization
- Conversational
- Text Generation
- Text2Text Generation
- Sentence Similarity

Audio

- Text-to-Speech
- Automatic Speech Recognition
- Audio-to-Audio
- Audio Classification
- Voice Activity Detection

Tabular

- Tabular Classification
- Tabular Regression

Reinforcement Learning

- Reinforcement Learning
- Robotics

meta-llama/Llama-2-70b
Text Generation • Updated 4 days ago • 25.2k • ♥ 64

stabilityai/stable-diffusion-xl-base-0.9
Updated 6 days ago • 2.01k • ♥ 393

openchat/openchat
Text Generation • Updated 2 days ago • 1.3k • ♥ 136

lllyasviel/ControlNet-v1-1
Updated Apr 26 • ♡ 1.87k

cerspense/zeroscope_v2_XL
Updated 3 days ago • 2.66k • ♥ 334

meta-llama/Llama-2-13b
Text Generation • Updated 4 days ago • 328 • ♥ 64

tiiuae/falcon-40b-instruct
Text Generation • Updated 27 days ago • 288k • ♥ 899

WizardLM/WizardCoder-15B-V1.0
Text Generation • Updated 3 days ago • 12.5k • ♡ 332

CompVis/stable-diffusion-v1-4
Text-to-Image • Updated about 17 hours ago • 448k • ♥ 5.72k

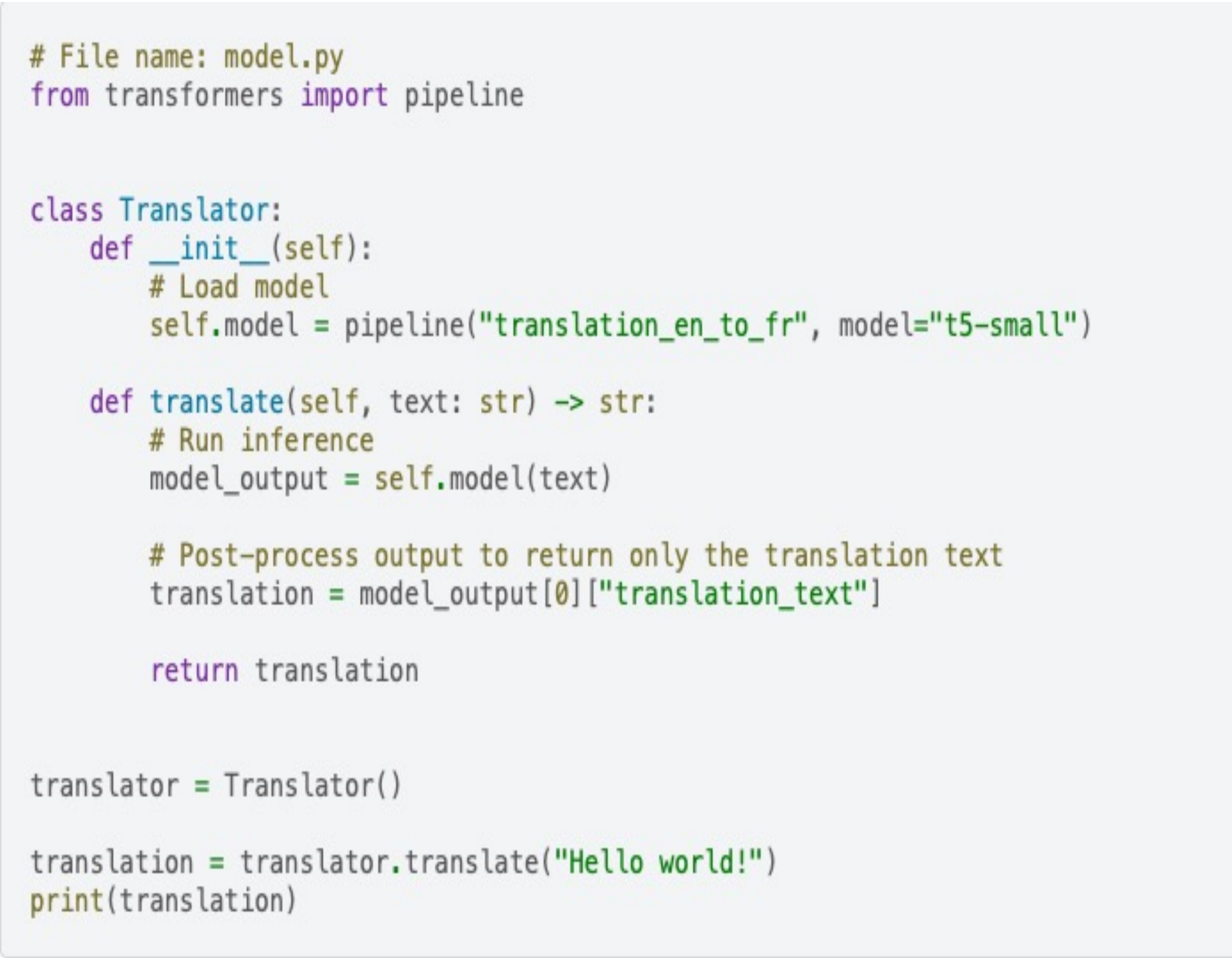
stabilityai/stable-diffusion-2-1
Text-to-Image • Updated about 17 hours ago • 782k • ♥ 2.81k

Salesforce/xgen-7b-8k-inst
Text Generation • Updated 4 days ago • 6.18k • ♡ 57

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

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

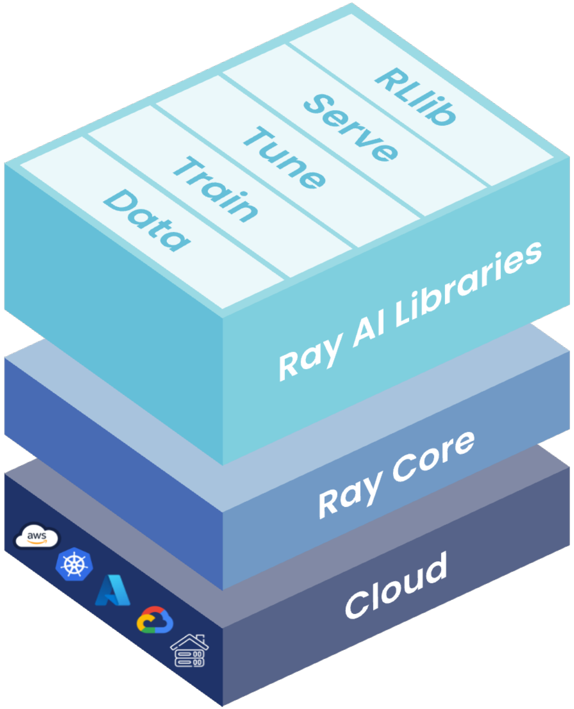
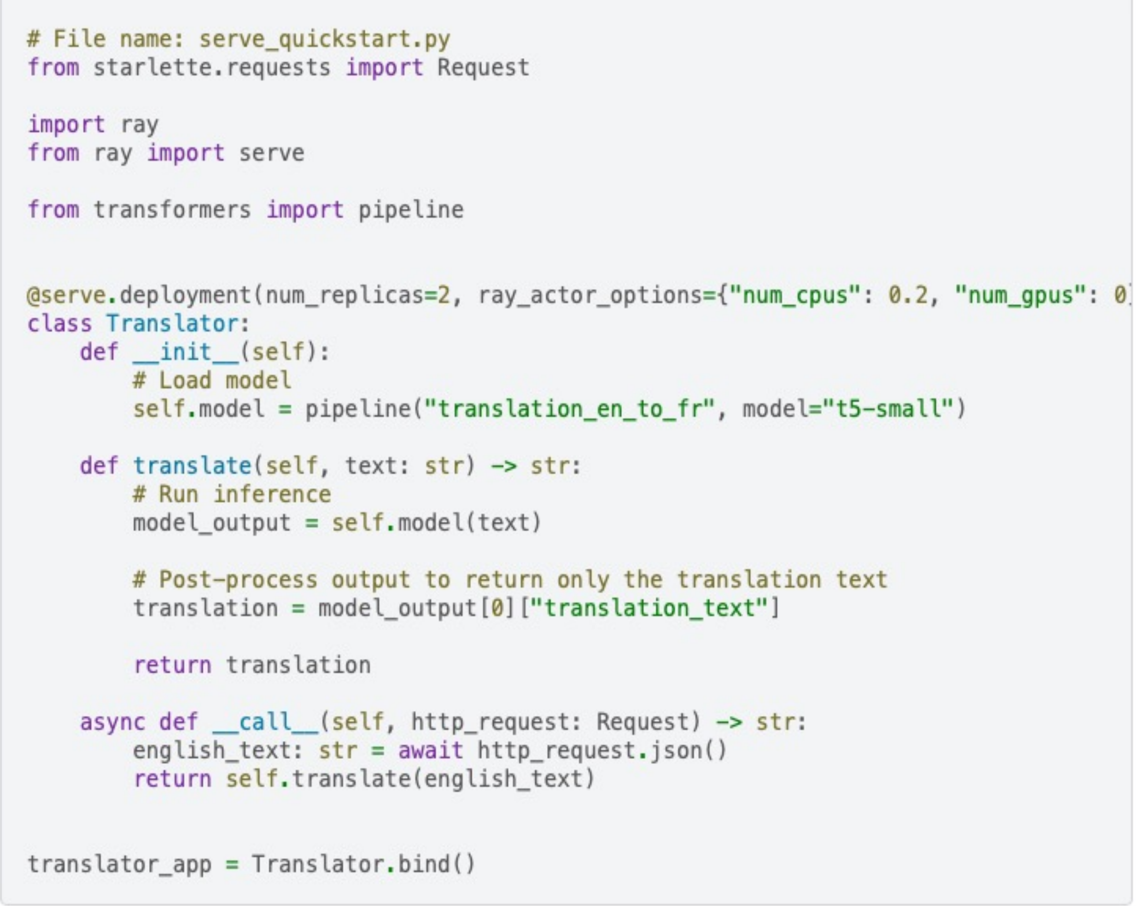


Image: https://docs.ray.io/en/latest/ray-overview/index.html

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:

```
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)
```



```
# 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: str = await http_request.json()
        return self.translate(english_text)


translator_app = Translator.bind()
```

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

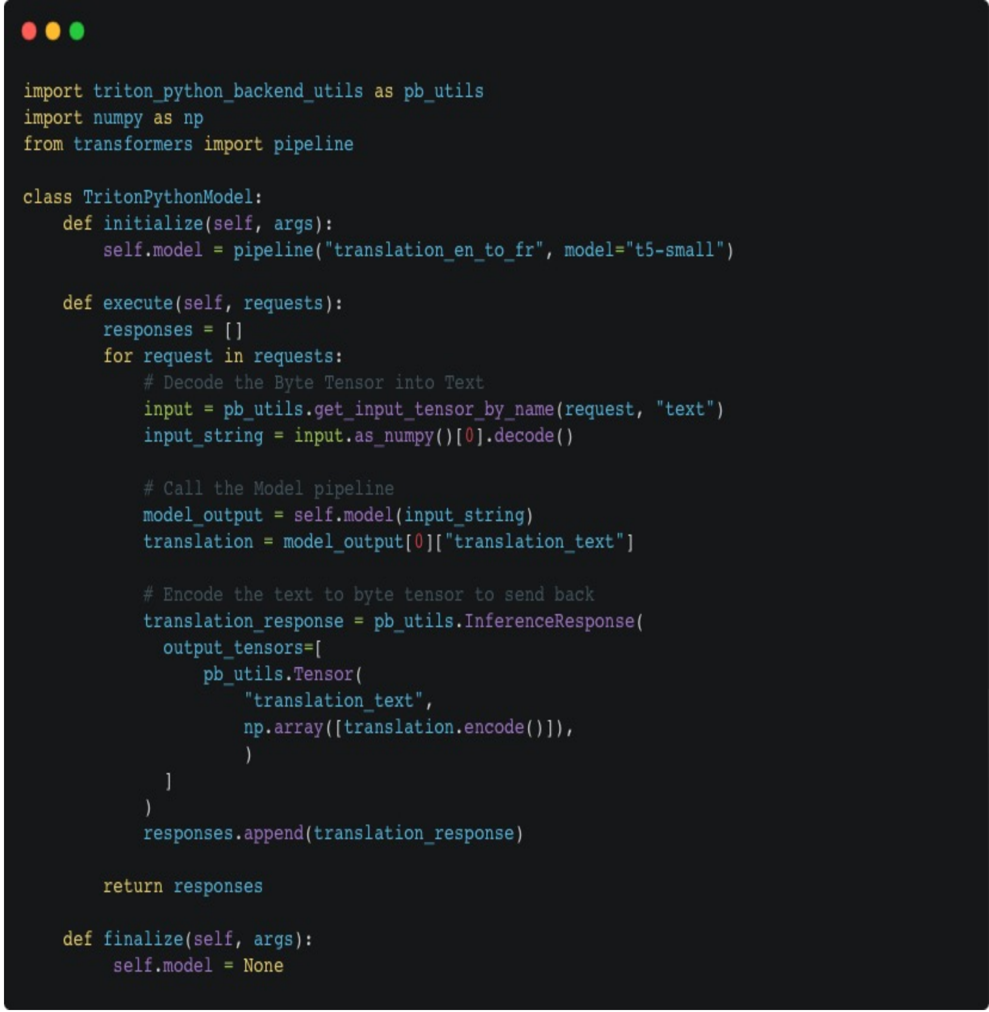
Triton Inference Server

- 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: https://developer.nvidia.com/triton-inference-server

Triton IS: Example (1)

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

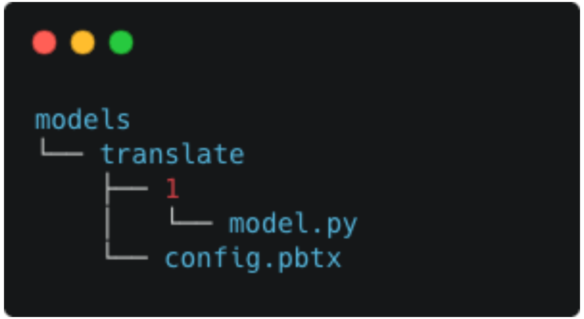


```python
import triton_python_backend_utils as pb_utils
import numpy as np
from transformers import pipeline

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

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

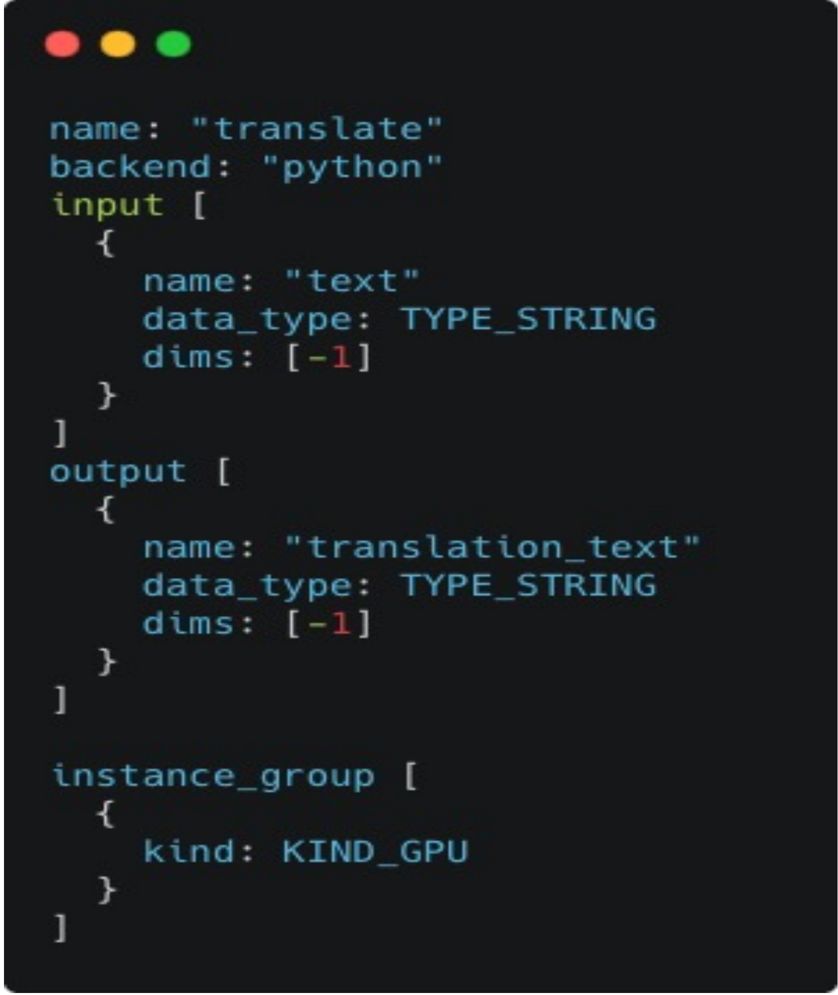
Source: https://www.inferless.com/learn/nvidia-triton-inference-inferless

Triton IS: Example (2)

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



```
models
└── translate
    ├── 1
    │   └── model.py
    └── config.pbtx
```

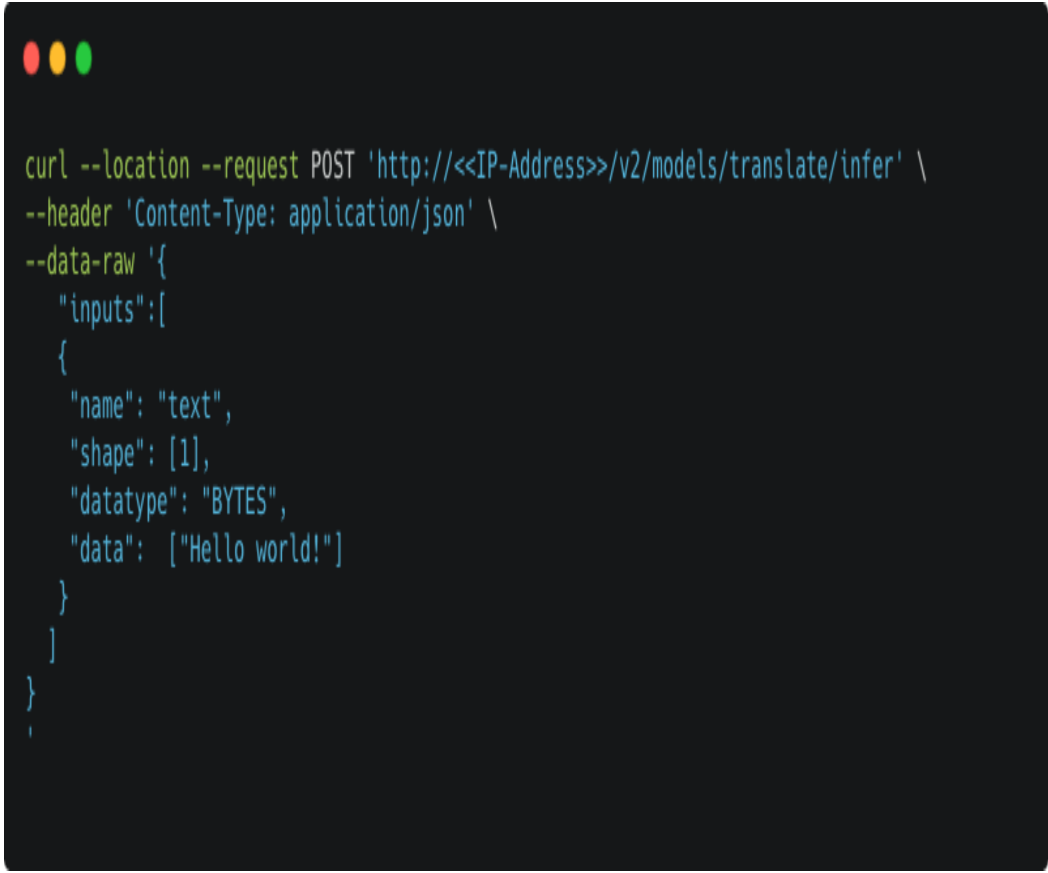


```
name: "translate"
backend: "python"
input [
  {
    name: "text"
    data_type: TYPE_STRING
    dims: [-1]
  }
]
output [
  {
    name: "translation_text"
    data_type: TYPE_STRING
    dims: [-1]
  }
]

instance_group [
  {
    kind: KIND_GPU
  }
]
```

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

Triton IS: Example (3)

- Serve the model as follows:
  - Run the Triton Inference Server container:
    - `docker run --shm-size=1g --ulimit memlock=-1 -p 8000:8000 -p 8001:8001 -p 8002:8002 --ulimit stack=67108864 -ti nvcr.io/nvidia/tritonserver:<xx.yy>-py3`
  - 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



```
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!"]
   }
  ]
}
'
```

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

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.

Photo by HckySo – CC BY-NC 2.0

@mhickeybot

Thank you.

Martin Hickey

Developer

—

Twitter: @mhickeybot