GOTC 2023

全球开源技术峰会

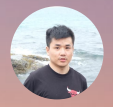
THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE, INTO THE FUTURE

「AI is Everywhere」专场

OpenGPT：多模态大模型推理框架 🚀

Taking LMM app into production with OpengGPT



王峰 @ Jina AI

2023年05月28日

About Me

2021 - now, Engineering manager, Jina AI

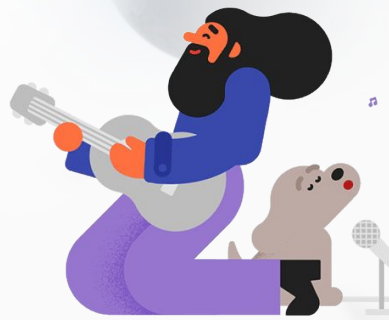
2020- 21, Senior Researcher, Huya AI

2018 - 19, Senior Researcher, Tencent AI

2011 - 18, Ph. D., Hong Kong Baptist University

numb3r3 felix.wang@jina.ai

王峰，开源 MLOPs 框架 Jina 的核心贡献者，专注机器学习与深度学习算法在 NLP，多模态表征学习和信息检索领域的落地与应用。



AI的未来范式

多模态大模型的兴起

GPT-4 的一大亮点就是不仅能理解文字，还能识别图片内容

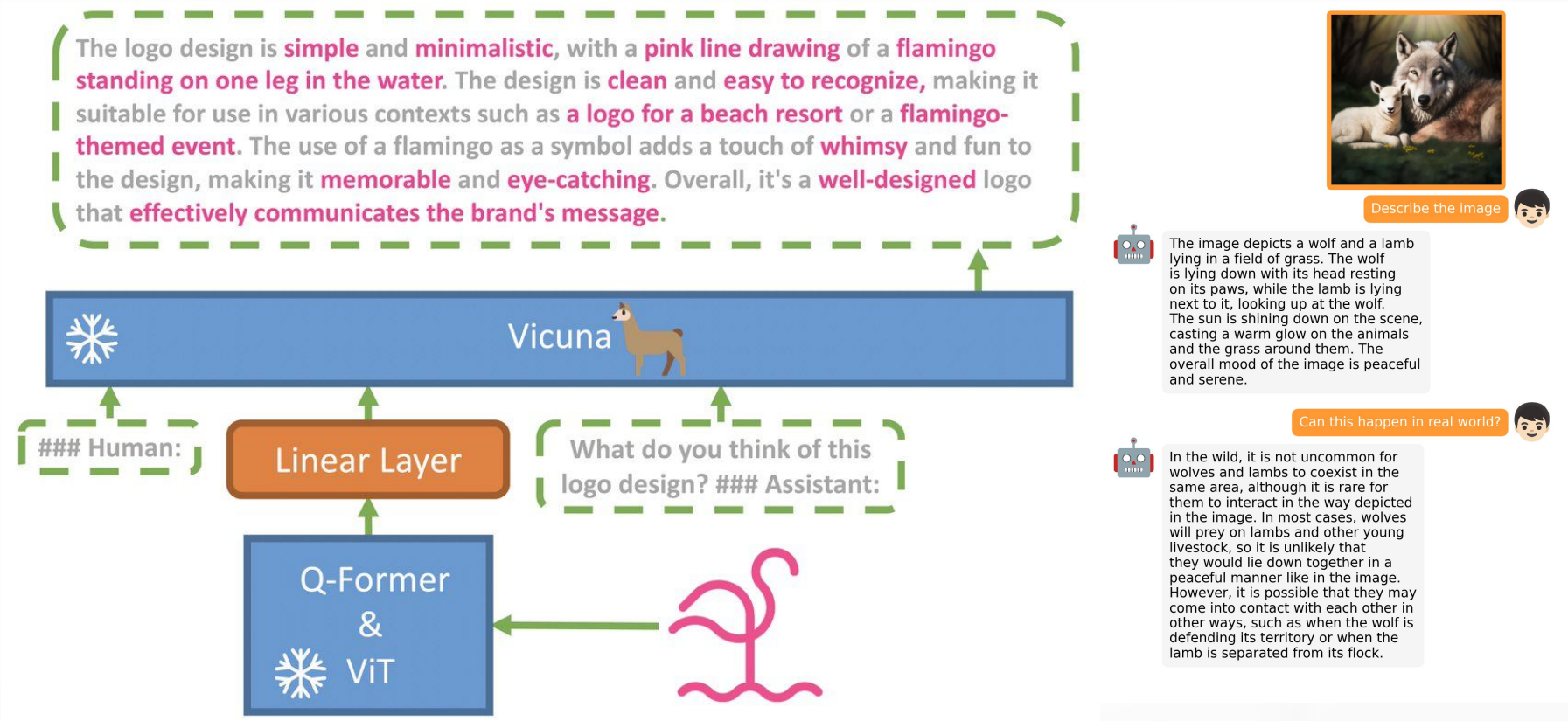
GPT-4 visual input example, Extreme Ironing:	
User	What is unusual about this image?
	 Source: https://www.barnorama.com/wp-content/uploads/2016/12/03-Confusing-Pictures.jpg
GPT-4	The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

Table 16. Example prompt demonstrating GPT-4's visual input capability. The prompt requires image understanding.

AI的未来范式

多模态大模型的兴起

MiniGPT-4：简单整合了BLIP-2和Vicuna，通过BLIP-2将图片转为文字描述，再用Vicuna基于这个文字描述进行内容创作。



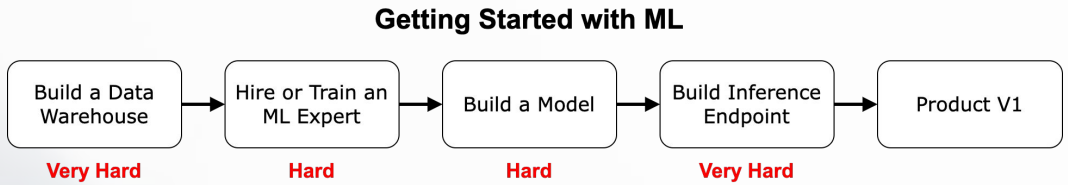
Zhu, Deyao et al. “MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models.” Arixv, 2023

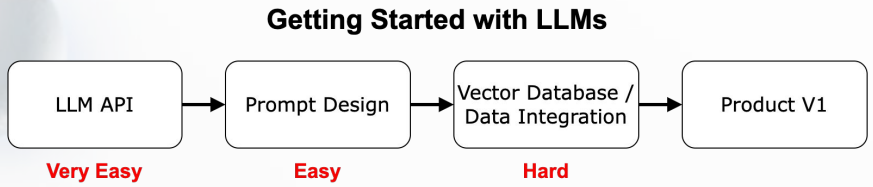
AI的未来范式

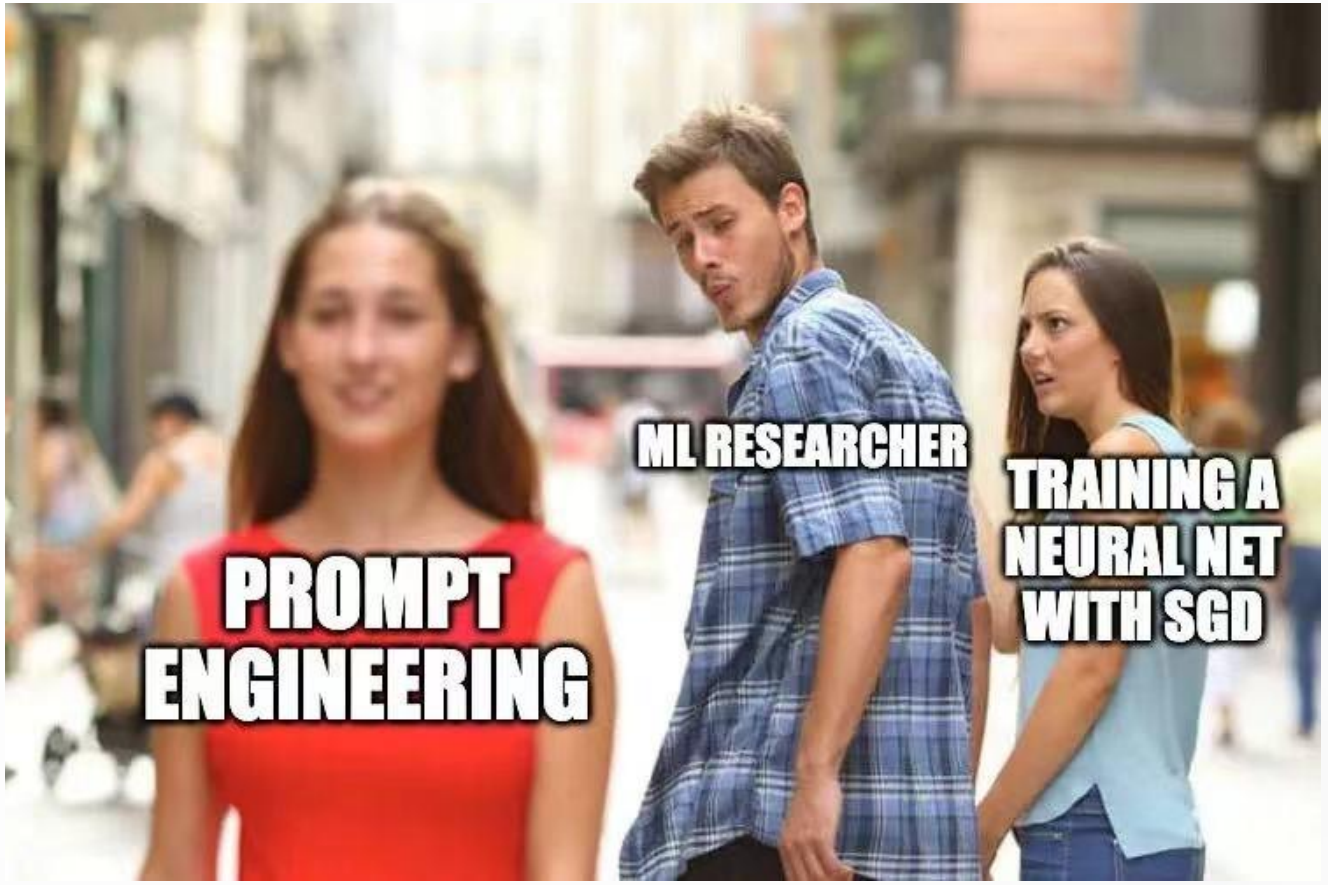
大模型的“能力涌现”

ChatGPT/GPT-4的横空出世，已经彻底改变了NLP领域的研究态势，模型能力越来越强

- 使用提示词 prompt 来完成特定意图；
- 由于“**涌现**”能力，借助 Incontext Learning 方法可以处理未见过的任务；
- 改变了模型使用的范式



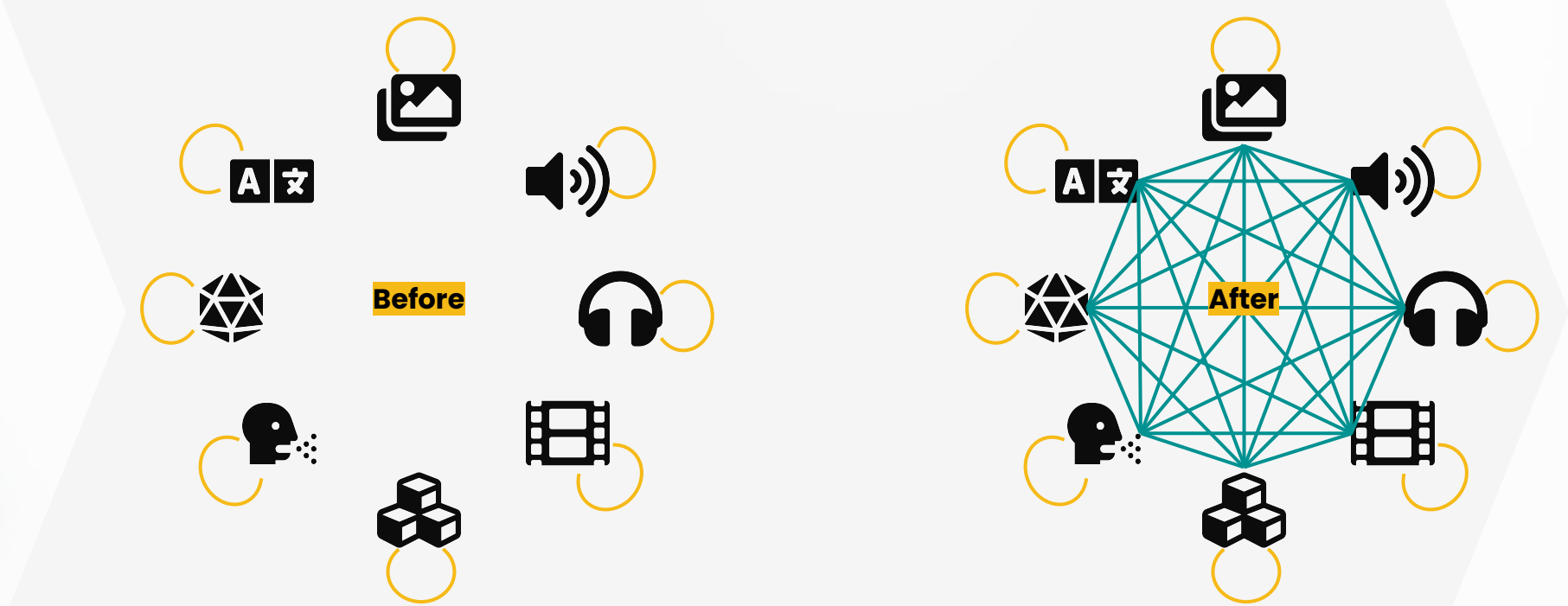




AI的未来范式

多模态 AI 的崛起

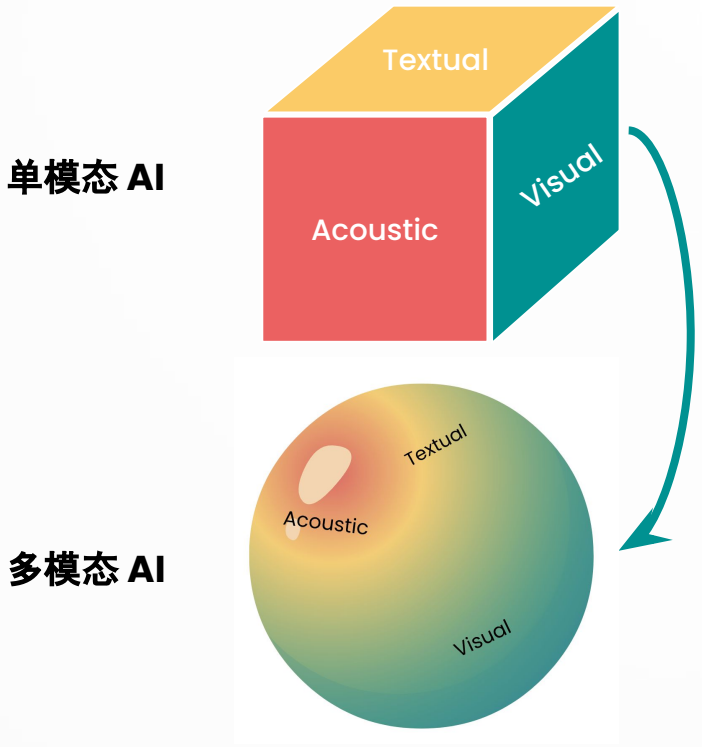
人工智能正在从文本、语音、视觉等单模态智能，向着多种模态融合的通用人工智能方向发展。



AI的未来范式：多模态大模型

多模态模型

简单来说，就是指模型可以处理多种结构/类型的数据，可接收多种类型的数据源，例如 GPT-4，它既可以处理你输入的文本，也可以处理你上传的图片。



表征学习

让模型为所有模态创建通用的特征表示

迁移学习

让模型首先学习基础知识，然后在特定领域进行微调。

基于多模态的大模型将实现图文音统一知识表示，
成为人工智能基础设施。

多模态大模型应用挑战

多模态 AI 的崛起

- 计算资源需求：大模型由于其参数众多，因此在训练和推理阶段都需要大量的计算资源。这可能需要昂贵的硬件设备，并可能导致能源消耗问题。
- 模型推理效率低：大模型的高效推理是工程应用的关键技术。推理环节在计算精度（FP16/INT8）、算力消耗量等方面的要求较低，但 GPU 显存不足的问题同样会出现在推理环节。此外，模型推理速度受限于通信延迟和硬件内存带宽。如何保持低延迟的前提下，还尽可能节省计算资源和使现有显存满足推理的要求，是我们依然面临的问题。
- 数据处理：多模态数据需要复杂的预处理和后处理步骤。
- 模型优化：大模型的训练通常需要精细的超参数调整和优化策略，以确保模型的性能和稳定性。
- 模型部署：部署相对复杂，在生产环境中部署大模型可能会面临技术和运营挑战。

以上就是一些大模型和多模态技术在实际应用中可能会遇到的技术挑战。解决这些挑战可能需要跨领域的合作和持续的技术创新。

延迟(lateny)
+
成本(cost)

jina-ai / opengpt Public

Edit Pins | Watch 20 | Fork 3 | Starred 18

README.md

OpenGPT

OpenGPT

多模态大模型服务框架

> "A playful and whimsical vector art of a Stochastic Tigger, wearing a t-shirt with a "GPT" text printed logo, surrounded by colorful geometric shapes. –ar 1:1 –upbeta"
>
> — Prompts and logo art was produced with PromptPerfect & Stable Diffusion X

Made with JinaAI | pypi v0.0.3 | license Apache-2.0

OpenGPT is an open-source *cloud-native* large-scale ***multimodal models*** (LMMs) serving framework. It is designed to simplify the deployment and management of large language models, on a distributed cluster of GPUs. We aim to make it a one-stop solution for a centralized and accessible place to gather techniques for optimizing large-scale multimodal models and make them easy to use for everyone.

OpenGPT

多模态大模型推理服务框架

An open-source *cloud-native* of **large multi-modal models** (LMMs) serving framework.

- Support for multi-modal models on top of large language models
- Scalable architecture for handling high traffic loads
- Optimized for low-latency inference
- Automatic model partitioning and distribution across multiple GPUs
- Centralized model management and monitoring
- REST API for easy integration with existing applications

Jina 🤝 Huggingface

Get Started

Installation

Install the package with `pip`:

```
pip install open_gpt_torch
```

Quickstart

```
import open_gpt

model = open_gpt.create_model(
    'stabilityai/stablelm-tuned-alpha-3b', device='cuda', precision='fp16'
)

prompt = "The quick brown fox jumps over the lazy dog."

output = model.generate(
    prompt,
    max_length=100,
    temperature=0.9,
    top_k=50,
    top_p=0.95,
    repetition_penalty=1.2,
    do_sample=True,
    num_return_sequences=1,
)
```

模型本地推理

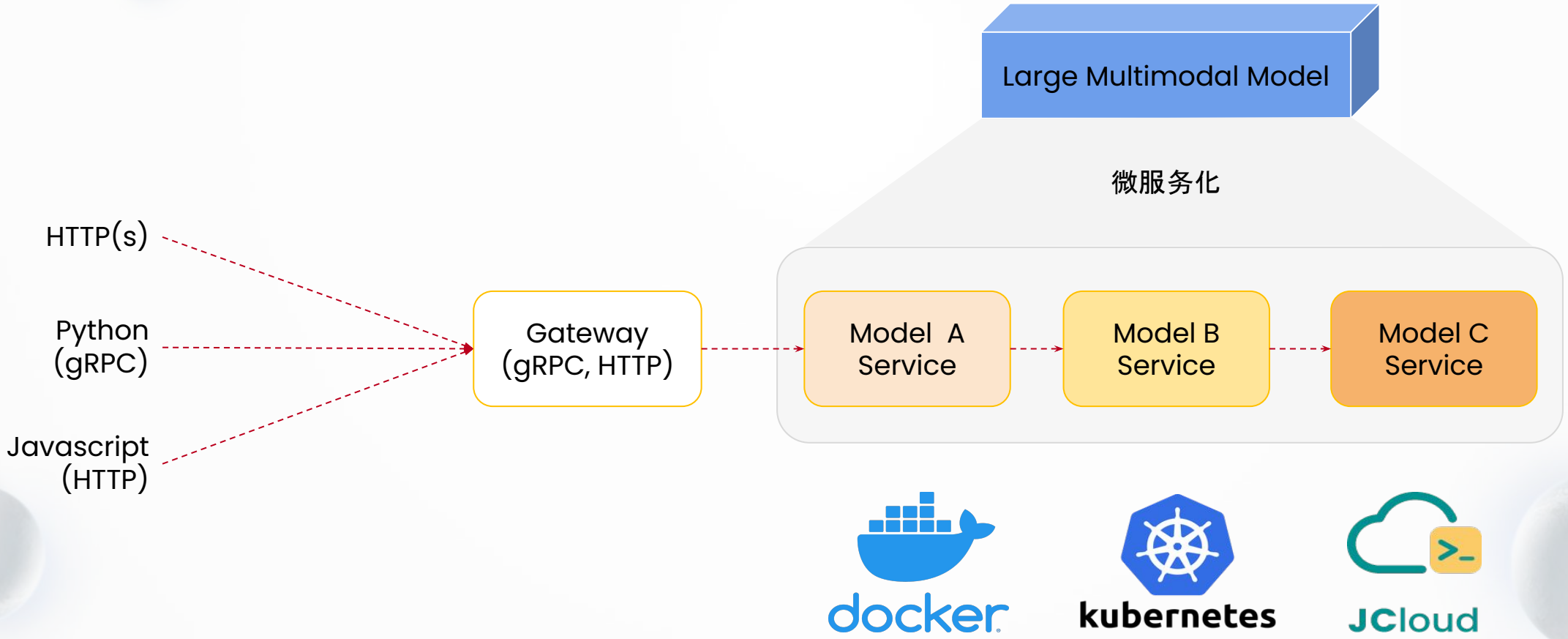
主要内容

- 模型服务架构
- 模型微服务化
- 模型推理优化
- 模型推理 API

OpenGPT： 架构图

大模型推理 **“微服务”** 框架

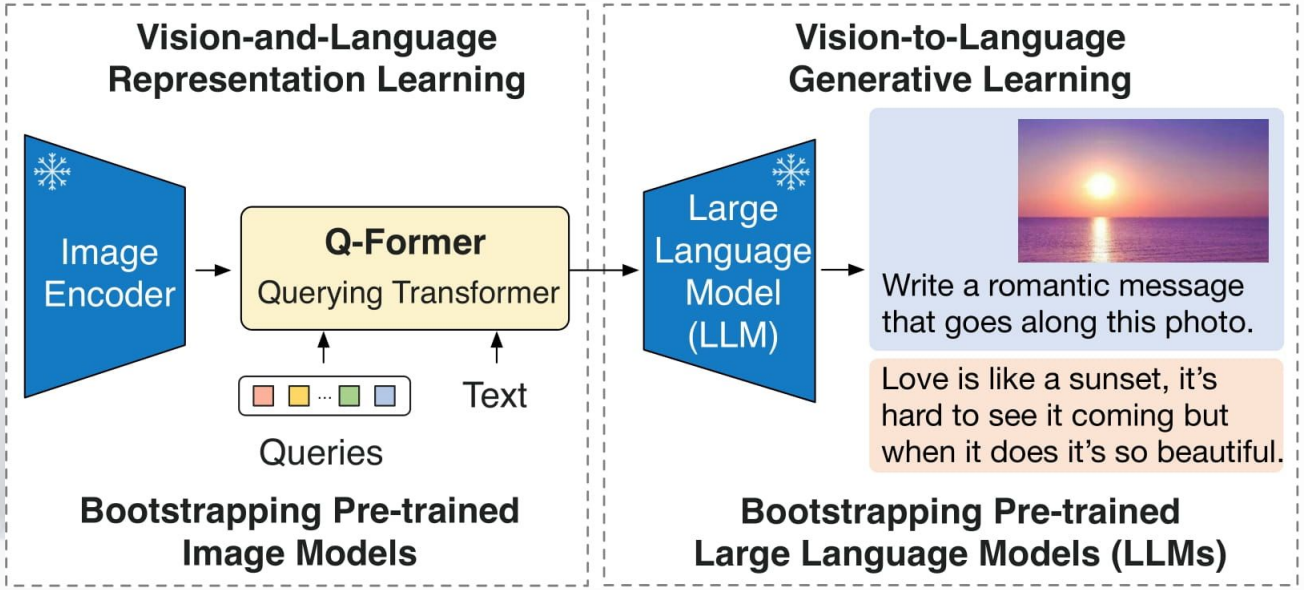
...

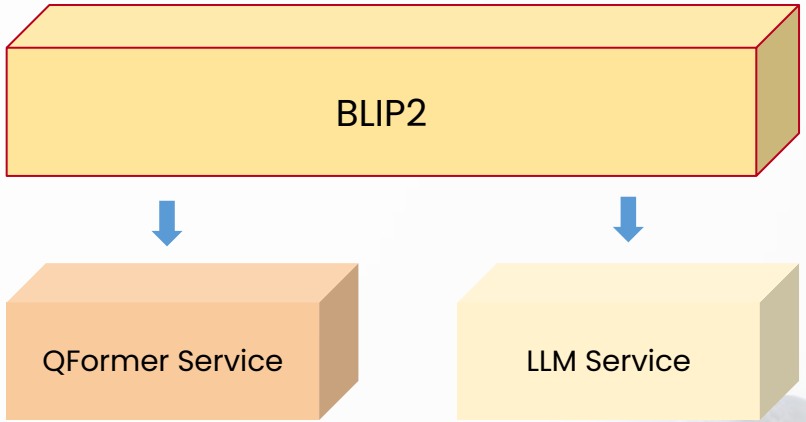


OpenGPT：模型微服务化

BLIP-2 一种新的视觉语言模型范式

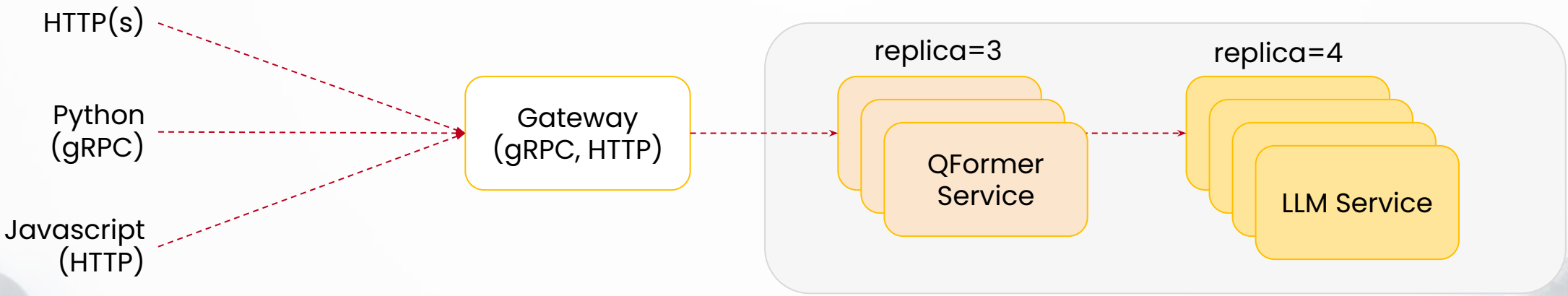
模块化 （or Mixture-Experts）：可以任意组合并充分利用两个预训练好的视觉编码器和 LLM，而无须端到端地预训练整个架构。这使得我们可以在多个视觉语言任务上实现最先进的结果，同时显著减少训练参数量和预训练成本





OpenGPT：模型微服务化

分布式推理服务

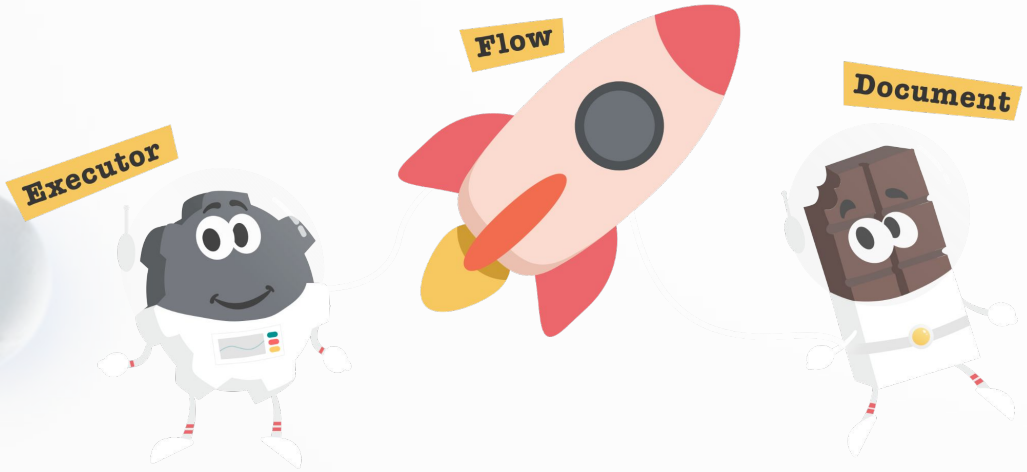


OpenGPT：模型微服务化

Jina 适用于多模态AI应用开发的微服务框架

设计 Document, Executor 和 Flow这 3 个基本概念来抽象问题

- Document：基本数据结构，原生支持 protobuf 序列化方案，从而具备高效的网络数据传输效率；
- Executor：微服务逻辑处理基本单元；
- Flow：微服务编排管理

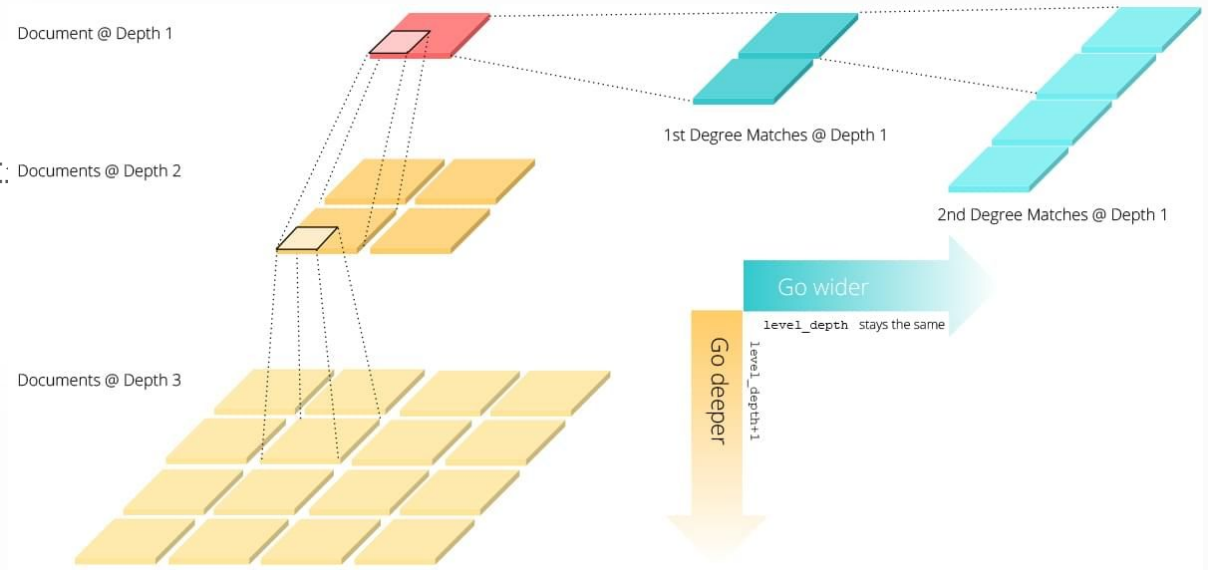


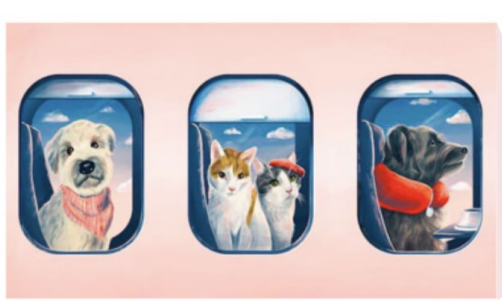

```python
from jina import Executor, Document, Flow, requests

class MyExecutor(Executor):

  @requests
  def foo(self, docs, **kwargs):
    print('recv:', docs)

f = Flow().add(uses=MyExecutor)

with f:
  f.post('/', Document(text='hello, world!'))
```

OpenGPT：模型微服务化

Jina Document

- 支持不同张量数据类型 np.ndarray， torch.tensor 等
- 可以表示不同模态数据，例如文本，图像，音频
- 支持嵌套关系





By the Way A Post Travel Destination

Everything to know about flying with pets, from picking your seat to keeping your animal calm

By Nathan Diller

Dataclass representation

```
from docarray import dataclass, Document
from docarray.typing import Image, Text, JSON

@dataclass
class WPArticle:
    banner: Image = 'cat-dog-flight.png'
    headline: Text = 'Everything to know about flying with
pets, from picking your seats to keeping your animal calm'
    meta: JSON = {
        'author': 'Nathan Diller',
        'Column': 'By the Way - A Post Travel Destination'}

Document(WPArticle())
```

OpenGPT：模型微服务化

Jina Executor 微服务逻辑单元

PyTorch

```python
import torch  # 1.8.1
import torchvision.models as models  # 0.9.1
from jina import Executor, requests


class PytorchMobilNetExecutor(Executor):
    def __init__(self, **kwargs):
        super().__init__()
        self.model = models.quantization.mobilenet_v2(pretrained=True, quantize=True)
        self.model.eval()

    @requests
    def encode(self, docs, **kwargs):
        blobs = torch.Tensor(docs.get_attributes('blob'))
        with torch.no_grad():
            embeds = self.model(blobs).detach().numpy()
            for doc, embed in zip(docs, embeds):
                doc.embedding = embed
```

TensorFlow

```python
import numpy as np
import tensorflow as tf
from keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input
from tensorflow.python.framework.errors_impl import InvalidArgumentError

from jina import Executor, requests


class TfMobileNetEncoder(Executor):
    def __init__(self, **kwargs):
        super().__init__()
        self.image_dim = 224
        self.model = MobileNetV2(pooling='avg', input_shape=(self.image_dim, self.image_dim, 3))

    @requests
    def encode(self, docs, **kwargs):
        buffers, docs = docs.get_attributes_with_docs('buffer')

        tensors = [tf.io.decode_image(contents=b, channels=3) for b in buffers]
        resized_tensors = preprocess_input(np.array(self._resize_images(tensors)))

        embeds = self.model.predict(np.stack(resized_tensors))
        for d, b in zip(docs, embeds):
            d.embedding = b

    def _resize_images(self, tensors):
        resized_tensors = []
        for t in tensors:
            try:
                resized_tensors.append(tf.keras.preprocessing.image.smart_resize(t, (self.image_dim, self.image_dim)))
            except InvalidArgumentError:
                # this can happen if you include empty or other malformed images
                pass
        return resized_tensors
```

PyTorch Lightning

```python
from pl_bolts.models.autoencoders import AE

from jina import Executor, requests

import torch


class PLMwuAutoEncoder(Executor):
    def __init__(self, **kwargs):
        super().__init__()
        self.ae = AE(input_height=32).from_pretrained('cifar10-resnet18')
        self.ae.freeze()

    @requests
    def encode(self, docs, **kwargs):
        with torch.no_grad():
            for doc in docs:
                input_tensor = torch.from_numpy(doc.blob)
                output_tensor = self.ae(input_tensor)
                doc.embedding = output_tensor.detach().numpy()
```

fast.ai

```python
import torch
from fastai.vision.models import resnet18

from jina import Executor, requests


class ResnetImageEncoder(Executor):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.model = resnet18()
        self.model.eval()

    @requests
    def encode(self, docs, **kwargs):
        batch = torch.Tensor(docs.get_attributes('blob'))
        with torch.no_grad():
            batch_embeddings = self.model(batch).detach().numpy()

        for doc, emb in zip(docs, batch_embeddings):
            doc.embedding = emb
```

ONNX

```python
from pathlib import Path

import numpy as np
import onnxruntime
from jina import Executor, requests
from transformers import BertTokenizerFast, convert_graph_to_onnx


class ONNXBertExecutor(Executor):
    def __init__(self, **kwargs):
        super().__init__()

        # export your huggingface model to onnx
        convert_graph_to_onnx.convert(
            framework="pt",
            model="bert-base-cased",
            output=Path("onnx/bert-base-cased.onnx"),
            opset=11,
        )

        # create the tokenizer
        self.tokenizer = BertTokenizerFast.from_pretrained("bert-base-cased")

        # create the inference session
        options = onnxruntime.SessionOptions()
        options.intra_op_num_threads = 1  # have an impact on performances
        options.graph_optimization_level = (
            onnxruntime.GraphOptimizationLevel.ORT_ENABLE_ALL
        )

        # Load the model as a graph and prepare the CPU backend
        self.session = onnxruntime.InferenceSession(
            "onnx/bert-base-cased.onnx", options
        )
        self.session.disable_fallback()

    @requests
    def encode(self, docs, **kwargs):
        for doc in docs:
```

MindSpore

```python
import numpy as np
from mindspore import Tensor  # mindspore 1.2.0
import mindspore.ops as ops
import mindspore.context as context

from jina import Executor, requests


class MindsporeMwuExecutor(Executor):
    def __init__(self, **kwargs):
        super().__init__()
        context.set_context(mode=context.PYNATIVE_MODE, device_target='CPU')
        self.encoding_mat = Tensor(np.random.rand(5, 5))

    @requests
    def encode(self, docs, **kwargs):
        matmul = ops.MatMul()
        for doc in docs:
            input_tensor = Tensor(doc.blob)  # convert the ``ndarray`` of the doc to ``Tensor``
            output_tensor = matmul(self.encoding_mat, input_tensor)  # multiply the input with the encoding matrix.
            doc.embedding = output_tensor.asnumpy()  # assign the encoding results to ``embedding``
```

PaddlePaddle

```python
import paddle as P # paddle==2.1.0
import numpy as np
from ernie.modeling_ernie import ErnieModel # paddle-ernie 0.2.0.dev1
from ernie.tokenizing_ernie import ErnieTokenizer

from jina import Executor, requests


class PaddleErineExecutor(Executor):
    def __init__(self, **kwargs):
        super().__init__()
        self.tokenizer = ErnieTokenizer.from_pretrained('ernie-1.0')
        self.model = ErnieModel.from_pretrained('ernie-1.0')
        self.model.eval()

    @requests
    def encode(self, docs, **kwargs):
        for doc in docs:
            ids, _ = self.tokenizer.encode(doc.text)
            ids = P.to_tensor(np.expand_dims(ids, 0))
            pooled, encoded = self.model(ids)
            doc.embedding = pooled.numpy()
```

scikit-learn

```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer

from jina import Executor, requests, DocumentArray


class TFIDFTextEncoder(Executor):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        from sklearn import datasets

        dataset = fetch_20newsgroups()
        tfidf_vectorizer = TfidfVectorizer()
        tfidf_vectorizer.fit(dataset.data)
        self.ttfidf_vectorizer = tfidf_vectorizer

    @requests
    def encode(self, docs: DocumentArray, *args, **kwargs):
        iterable_of_texts = docs.get_attributes('text')
        embedding_matrix = self.tfidf_vectorizer.transform(iterable_of_texts)

        for i, doc in enumerate(docs):
            doc.embedding = embedding_matrix[i]
```

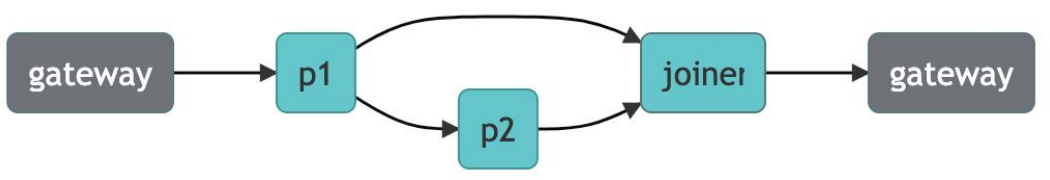
OpenGPT：模型微服务化

Jina Flow 微服务编排管理API

灵活的微服务编排

```python
from jina import Flow

f = Flow().add().add()
```


```python
from jina import Flow

f = (Flow()
        .add(name='p1')
        .add(name='p2')
        .needs(['p1', 'p2']))
```



OpenGPT：模型微服务化

Jina Flow 微服务编排管理API

支持不同的服务网关协议

```python
from jina import Flow

f = Flow()
```

via gRPC

```python
from jina import Flow

f = Flow(protocol='websocket')
```

via Websocket

```python
from jina import Flow

f = Flow(protocol='http')
```

via HTTP

Inference | Effortlessly integrate state-of-the-art ML models into your software | Jina AI

Effortlessly integrate state-of-the-art ML models into your software

Jina AI • Jina AI

GOTC

cloud.jina.ai/user/inference

Update

Explore
Dashboard
Services
Finetuner
Inference
Search BETA
MLOps
Flows
Executors
Storage
Executor Hub
Settings
Settings

Image Caption

Image Captioning is the task of describing the content of an image in words.

Playground

Get Access to API

Input

You can input an image URL, upload your image, or click one of the examples to try.

https://picsum.photos/id/358/100

Number of captions 1

Run

Examples

Captions 624ms

- a black and white photo of a bridge

Cohere	Curated models for - Embedding, Generation, and Classification	No information available publicly	~ US$ 0.002
Replicate	Non-curated list of models for - Embedding, Generation, Classification, Captioning …	2 seconds per image caption with BLIP2	~ US$ 0.005
Jina AI	Curated models for - Embedding, Captioning, Reasoning, Upscaling, (Generation)	0.5 seconds per image caption with BLIP2	US$ 0.0001

OpenGPT：模型推理优化

模型加载内存占用优化

通常模型加载需要把模型权重从文件加载到内存CPU中，然后再copy到CUDA中。这个过程实际上非常占用内存。而且这部分的资源是临时性的，加载完成后会释放。

但是在实际部署中，实际申请的节点资源是根据峰值决定的。如何减少模型对于内存占用的最大峰值是需要非常注意的问题。

```python
from transformers import AutoConfig, AutoModelForCausalLM, AutoTokenizer
from accelerate import init_empty_weights, load_checkpoint_and_dispatch


config = AutoConfig.from_pretrained('stabilityai/stablelm-tuned-alpha-7b')

model_path = huggingface_hub.snapshot_download('stabilityai/stablelm-tuned-alpha-7b')

with init_empty_weights():
    model = AutoModelForCausalLM.from_config(
        config, torch_dtype=dtype, trust_remote_code=True
    )
    # make sure token embedding weights are still tied if needed
    model.tie_weights()

    model = load_checkpoint_and_dispatch(
        model,
        model_path,
        device_map='balanced',
        no_split_module_classes=['GPTNeoXLayer'],
        dtype=torch.float16,
      )
```

Jina 🤝 Huggingface

OpenGPT：模型推理优化

多 GPU 推理优化

大规模模型的推理显存需求过高，通常单个 GPU 的显存是不够的，因此支持多 GPU 并行是大模型推理技术优化的第一步。

- 将推理工作负载拆分到多个 GPU 上，多 GPU 推理还能减少推理延迟（inference latency），以满足实际生产环境当中对延迟到严格要求。
- 过多的 GPU 并行，会增加跨 GPU 通信时间和降低每个 GPU 的计算粒度，从而导致最终结果是增加而不是减少延迟（latency）的问题。
- 因此，为了满足延迟要求同时减少并行开销，**有必要调整并行度，并确定给定模型体系结构和硬件平台的最佳值。**

```python
from transformers import AutoConfig, AutoModelForCausalLM, AutoTokenizer
from accelerate import init_empty_weights, load_checkpoint_and_dispatch


config = AutoConfig.from_pretrained('stabilityai/stablelm-tuned-alpha-7b')

model_path = huggingface_hub.snapshot_download('stabilityai/stablelm-tuned-alpha-7b')

with init_empty_weights():
    model = AutoModelForCausalLM.from_config(
        config, torch_dtype=dtype, trust_remote_code=True
    )
    # make sure token embedding weights are still tied if needed
    model.tie_weights()

    model = load_checkpoint_and_dispatch(
        model,
        model_path,
        device_map='balanced',
        no_split_module_classes=['GPTNeoXLayer'],
        dtype=torch.float16,
      )
```

Jina 🤝 Huggingface

OpenGPT：模型推理优化

模型推理 API

不同的模型推理方式，使用统一的推理API

```python
import open_gpt

model = open_gpt.create_model(
    'stabilityai/stablelm-tuned-alpha-3b', device='cuda', precision='fp16'
)

prompt = "The quick brown fox jumps over the lazy dog."

output = model.generate(
    prompt,
    max_length=100,
    temperature=0.9,
    top_k=50,
    top_p=0.95,
    repetition_penalty=1.2,
    do_sample=True,
    num_return_sequences=1,
)
```

本地模型推理

```python
from open_gpt import Client

client = Client()

# connect to the model server
model = client.get_model(endpoint='grpc://0.0.0.0:51000')

prompt = "The quick brown fox jumps over the lazy dog."

output = model.generate(
    prompt,
    max_length=100,
    temperature=0.9,
    top_k=50,
    top_p=0.95,
    repetition_penalty=1.2,
    do_sample=True,
    num_return_sequences=1,
)
```

客户端模型推理

总结展望

- 多模态大模型的狂飙
- 多模态大模型部署的挑战
- OpenGPT的模型部署方案
- 我们会持续优化opengpt，集成更多模型推理优化技术

大型语言模型已经改变了AI领域的格局，带来了许多新的机遇和挑战。未来，我们期待看到更多有关如何优化和应用这些模型的研究和创新。

THANKS

Jina AI 开源社区致力于促进多模态 AI 技术的应用落地以及传播，通过人工智能和深度学习技术，帮助开发者和企业减少开发学习成本，加快开发部署效率。

查看官网，了解更多：https://jina.ai/

1分钟填写反馈问卷
领取精美周边礼品