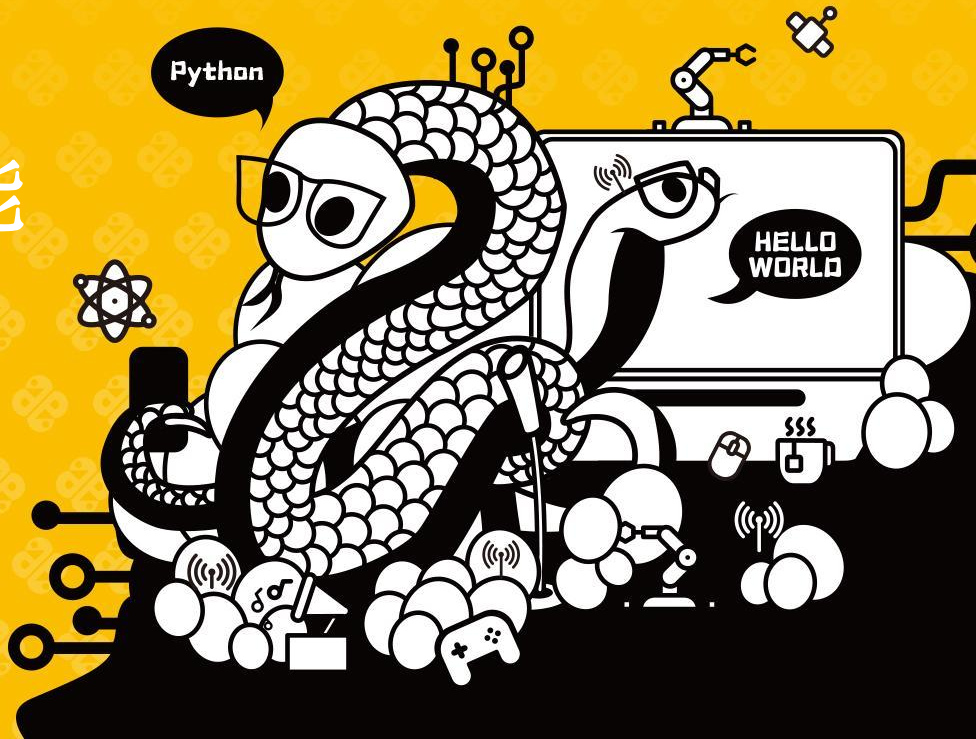
python × PYCHINA Python 中国社区

Python for Good

PyCon China 2022

多模态场景下的高性能Embedding服务

主讲人： 付杰 @ Jina AI， 高级算法工程师

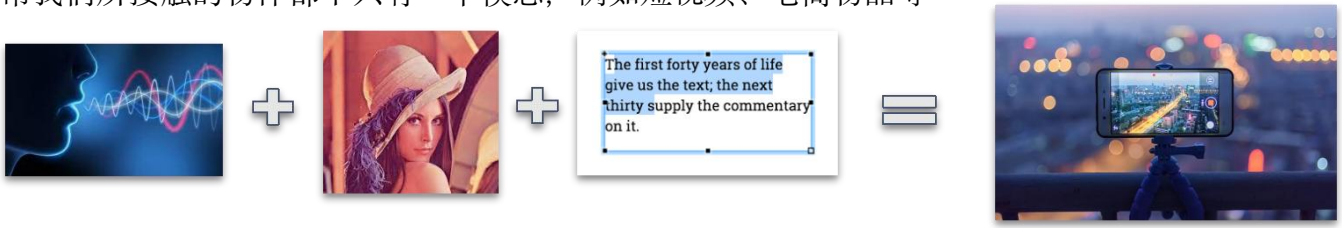


目录

为什么要研究多模态？

- **什么是多模态数据？**

我们生活的世界是一个多模态的世界，包含丰富的文本、视觉、听觉、嗅觉信息，通常我们所接触的物体都不只有一个模态，例如短视频、电商物品等



- **多模态数据在哪里领域有运用？**
  - 消费电商领域：商品的多模态，商品价格，文本描述，图片视频描述
  - 短视频娱乐领域：视频的多模态，图像，声音，字幕，视频标签
  - 自动化/机器人领域：感知信息的多模态，外界采集的图像，声音，环境光线...

为什么要研究多模态？

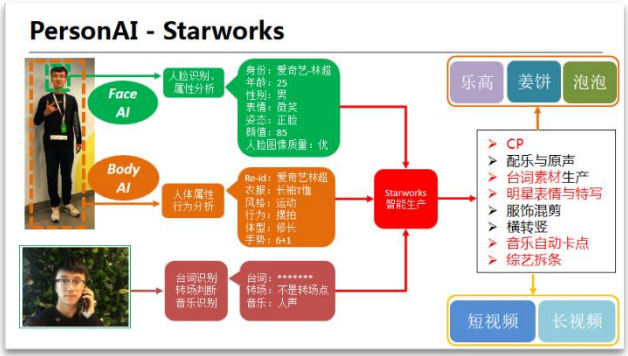
- **短视频电商/广告领域**

哪些行业涉及到多模态数据？

1. 短视频数据：
   - 如何理解用户上传的视频
   - 如何实现内容的精准/多样性分发/推荐
1. 电商搜索：
   - 如何更好的理解商品
   - 针对用户搜索历史如何更好地推荐商品

多模态技术在工业界的应用

- 爱奇艺/优酷——视频搜索，视频内容理解

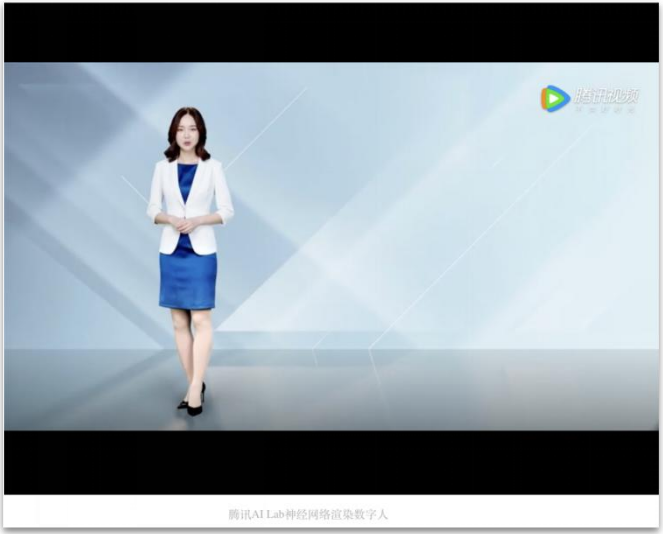


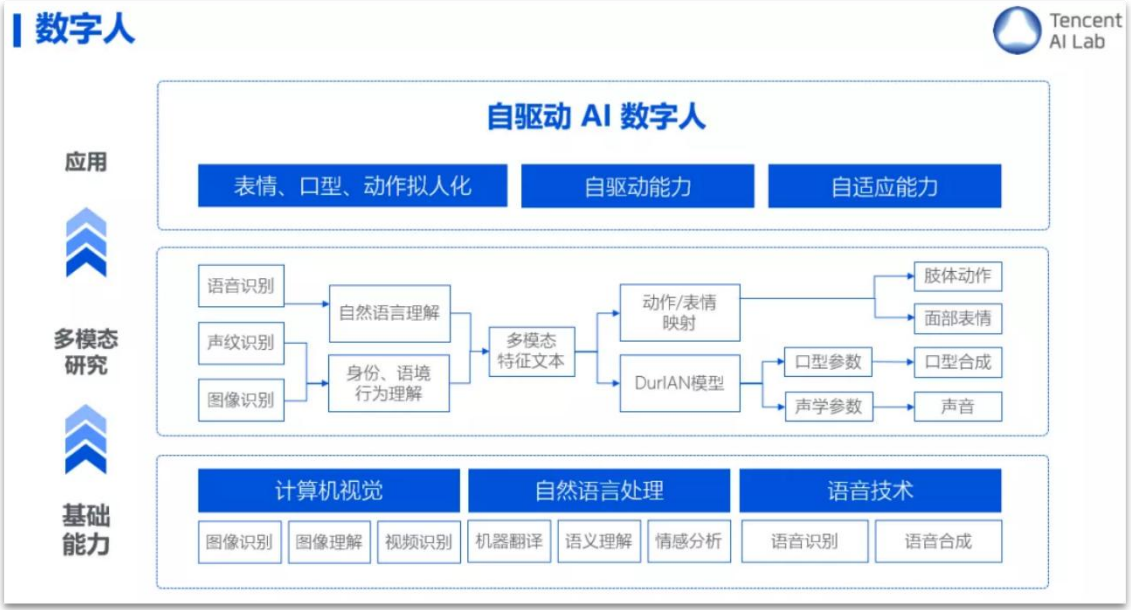
- 多模态召回辅助文本召回
- 只看TA
- 明星视频自动化生成

…

多模态技术在工业界的应用

- 腾讯AILab——数字人





目录

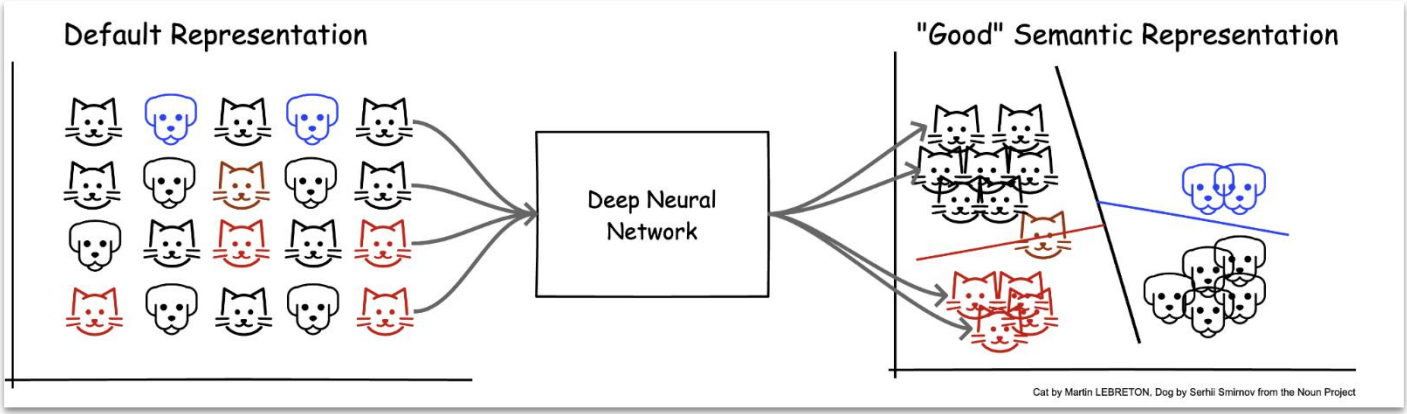
多模态场景下如何表征数据——向量表征

- **表征多模态数据有哪些难点？**

表征单模态：如何针对不同的模态数据选取不同的表征模型

对齐不同模态：如何确定来自两种或两种以上不同模式的(分)要素之间的直接关系

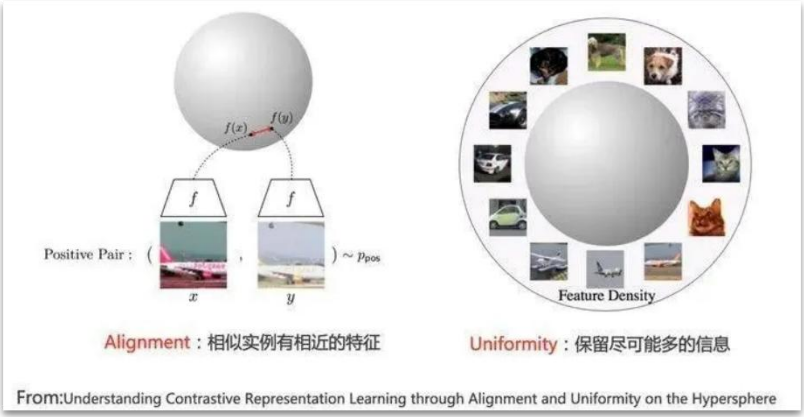
融合各个单模态：如何利用多模态的互补性和冗余性的特质来表示和总结多模态数据

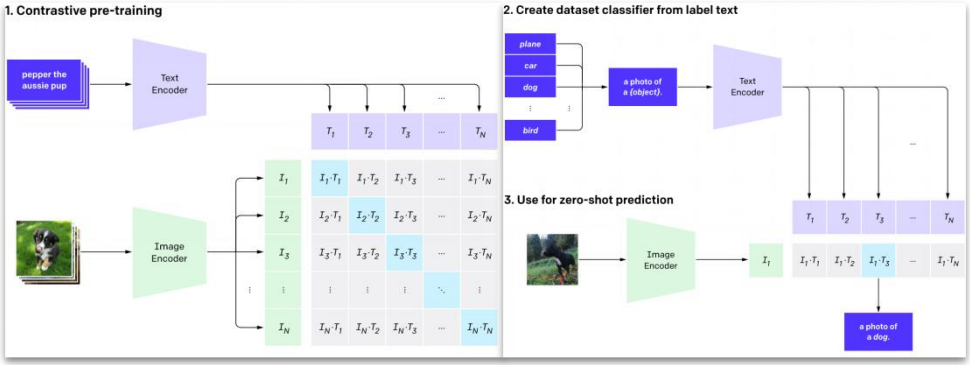


多模态场景下如何表征数据——CLIP

对比学习：自监督学习的一种，从无标注的数据中学习数据的分布

OpenAI 在 2021 年 1 月发布的 CLIP (Contrastive Language-Image Pre-training) 模型，它可以基于文本对图像进行分类，打破了自然语言处理和计算机视觉两大门派「泾渭分明」的界限，实现了多模态 AI 系统

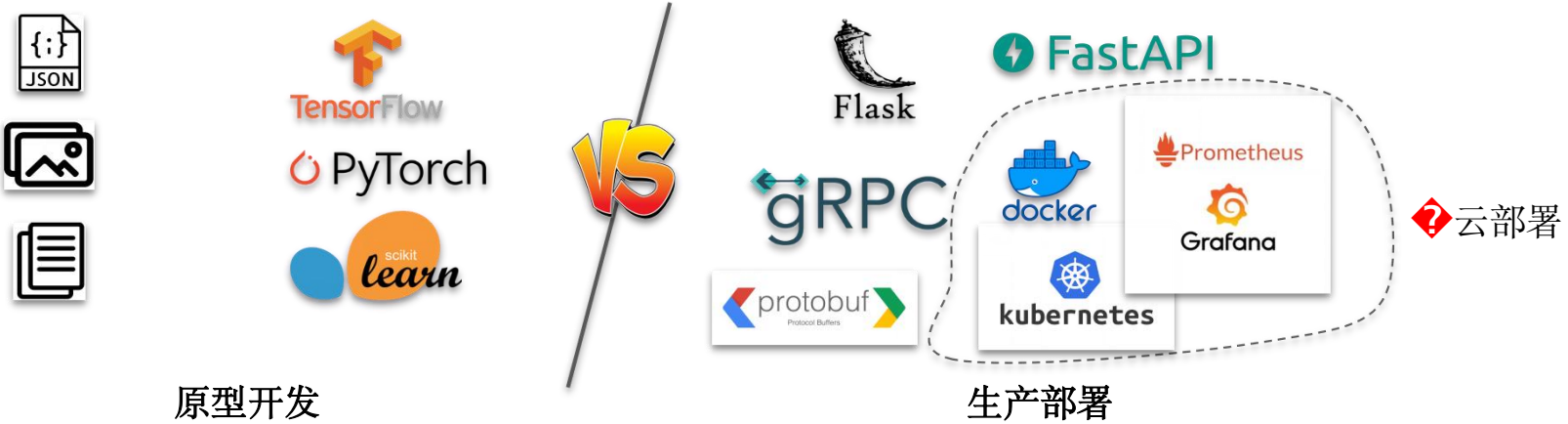




目录

模型推理服务的现状和挑战

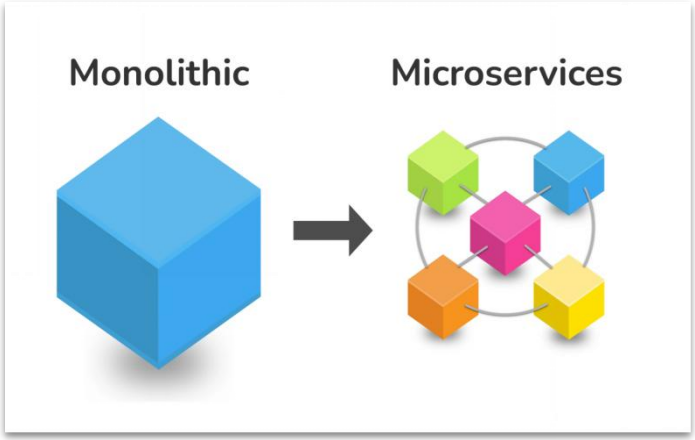
- **复杂技术栈**：模型推理、分布式、服务协议、日志收集、服务监控等
- **部署多模型**：越来越多的应用需要多个模型模型一起工作
- **云原生部署**：容器化，微服务，服务编排，异构计算等
- **业务集成难**：推理服务能够快速即成到业务系统中，并且需要满足不同场景个性化需求

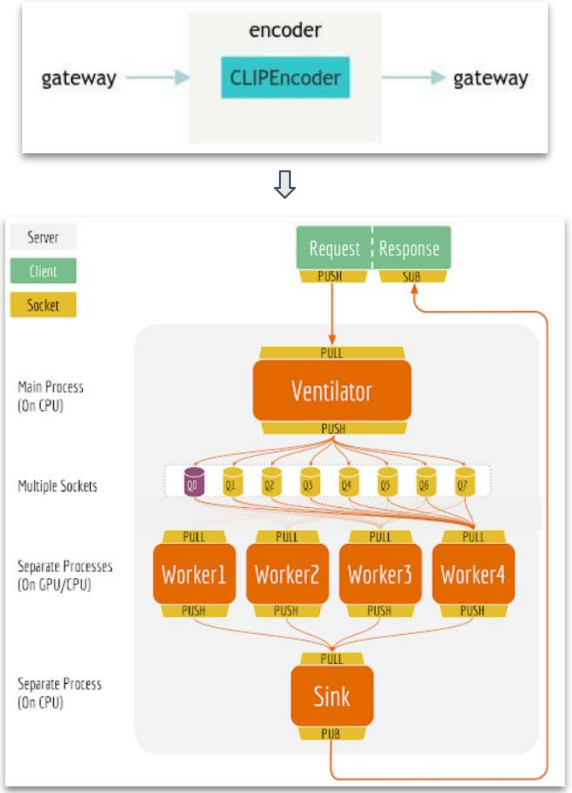


模型推理服务的现状和挑战

- **服务协议**：HTTP，gRPC
- **数据格式**：JSON， Protobuf
- **开发语言**：Python, C++, Java, Golang, Rust
- **模型格式**：Tensorflow, Pytorch, ONNX, PaddlePaddle
- **请求模式**：批处理，实时推理，流式推理；异步 vs 同步；
- **推理性能**：对推理加速引擎的支持， 例如 Nvidia TensorRT, OpenAI Triton, ONNX-Runtime 等；
- **服务监控**：Promethesus, OpenTelemetry, Grafana

模型推理服务的现状和挑战——云原生





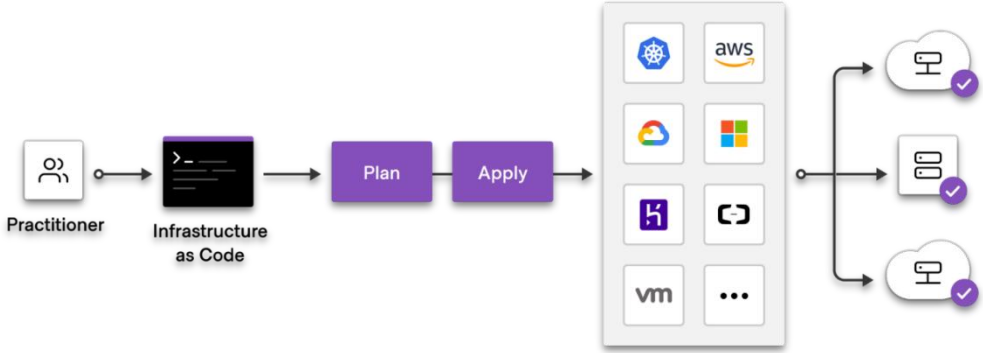
云原生的特点：

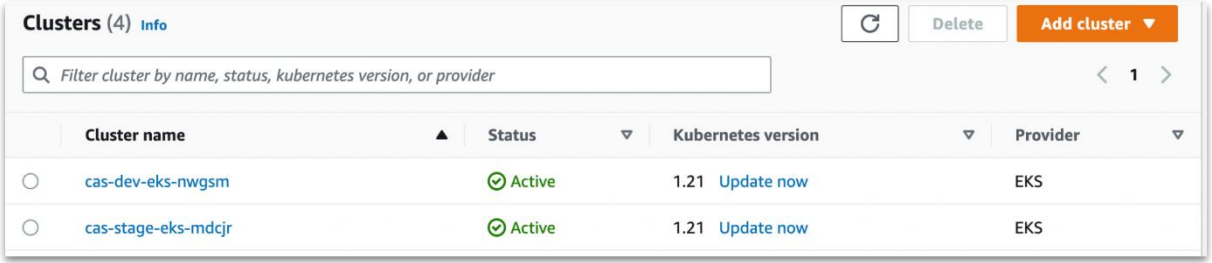
- **容器化技术**：保证每个模块都有独立的运行环境；
- **微服务技术**：保证每个模块在以分布式形式独立运行；
- **微服务编排**：负责协调流水线的持续运行和弹性伸缩；
- **指标观测性**：及时捕获性能指标，收集日志，快速定位故障原因；

模型推理服务的现状和挑战——服务部署

Terraform是一种安全有效地构建、更改和版本控制基础设施的工具

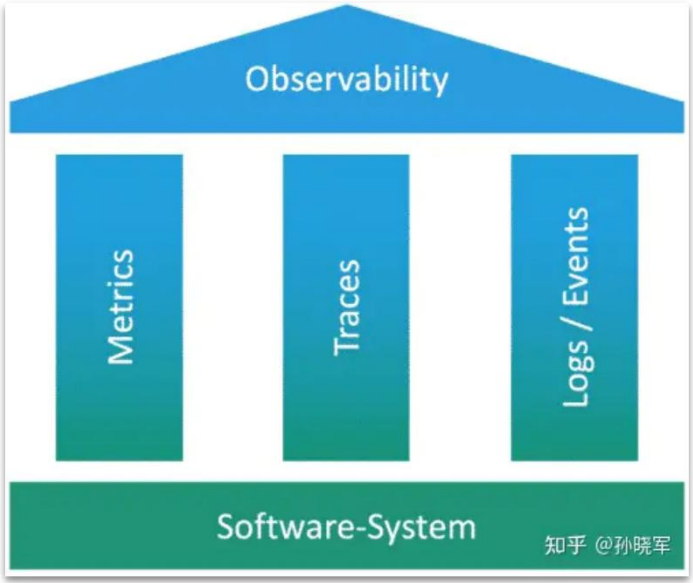
线上服务部署在Amazon EKS(Elastic Kubernetes Service)

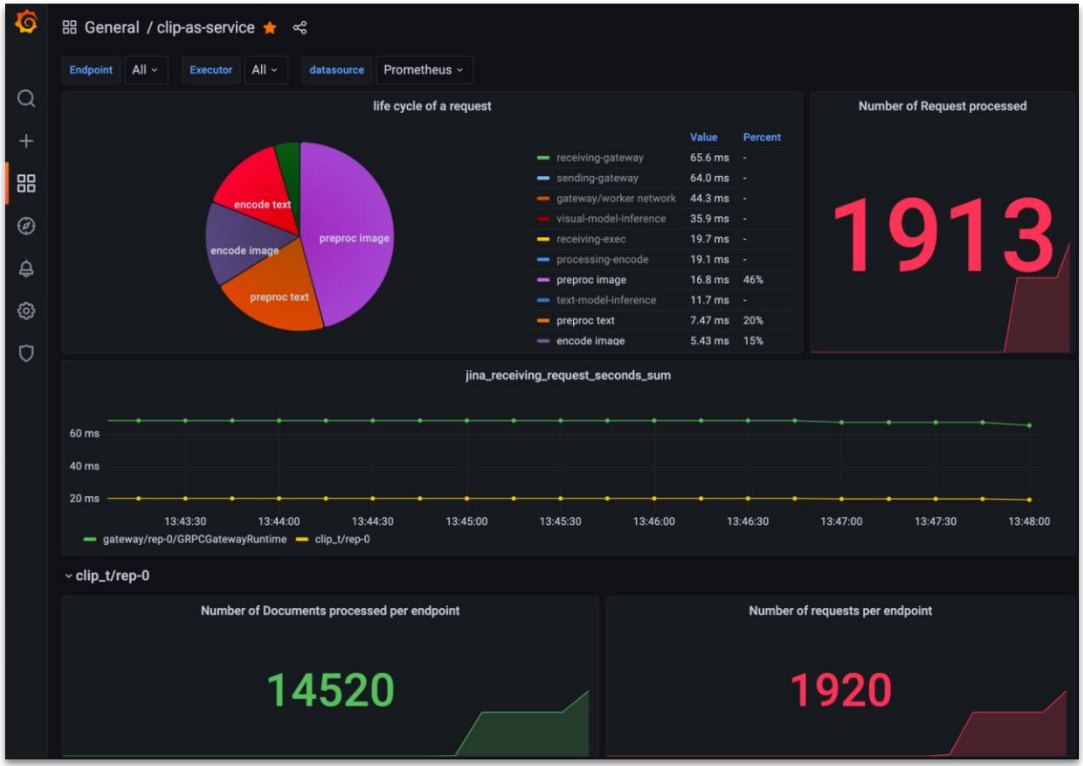





```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }

  required_version = ">= 1.2.0"
}

provider "aws" {
  region  = "us-west-2"
}

resource "aws_instance" "app_server" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleAppServerInstance"
  }
}
```

模型推理服务的现状和挑战——服务监控





OpenTelemetry: Tracing + Metrics数据收集

Jaeger + Promethesus： 数据接收并统计

Grafana： 前端数据展示

模型推理服务的现状和挑战——服务监控


```python
# for image
if _img_da:
    with self.tracer.start_as_current_span(
        'img_minibatch_encoding'
    ) as img_encode_span:
        img_encode_span.set_attribute(
            'num_pool_workers', self._num_worker_preprocess
        )
        for minibatch, batch_data in _img_da.map_batch(
            partial(
                self._preproc_images,
                drop_image_content=_drop_image_content,
            ),
            batch_size=self._minibatch_size,
            pool=self._pool,
        ):
            with self.monitor(
                name='encode_images_seconds',
                documentation='images encode time in seconds',
            ):
                minibatch.embeddings = (
                    self._model.encode_image(**batch_data)
                    .cpu()
                    .numpy()
                    .astype(np.float32)
                )
```

```python
def monitor(
    self, name: Optional[str] = None, documentation: Optional[str] = None
) -> Optional[MetricsTimer]:
    """
    Get a given prometheus metric, if it does not exist yet, it will create it and
    :param name: the name of the metrics
    :param documentation:  the description of the metrics

    :return: the given prometheus metrics or None if monitoring is not enable.
    """
    _summary = (
        self._metrics_buffer.get(name, None) if self._metrics_buffer else None
    )
    _histogram = (
        self._histogram_buffer.get(name, None) if self._histogram_buffer else None
    )

    if self._metrics_buffer and not _summary:...

    if self._histogram_buffer and not _histogram:...

    if _summary or _histogram:
        return MetricsTimer(
            _summary,
            _histogram,
            histogram_metric_labels={'runtime_name': self.runtime_args.name},
        )

    return contextlib.nullcontext()
```

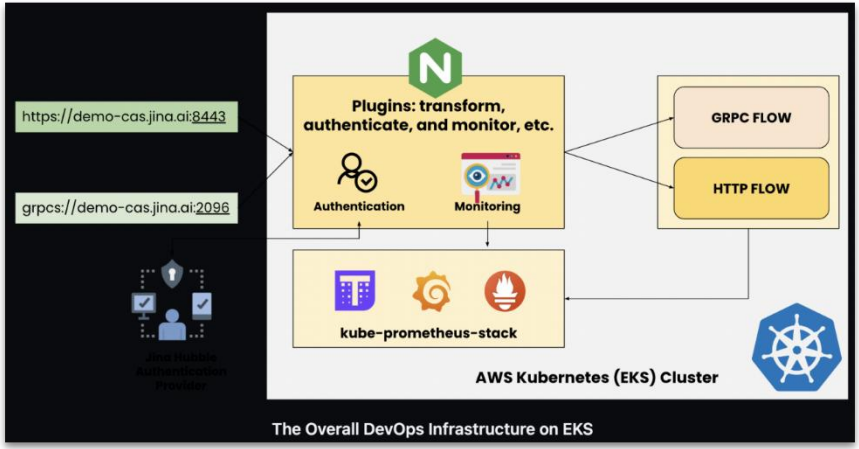
模型推理服务的现状和挑战——服务协议

Version 1：部署两个Flow，通过Nginx来进行转发

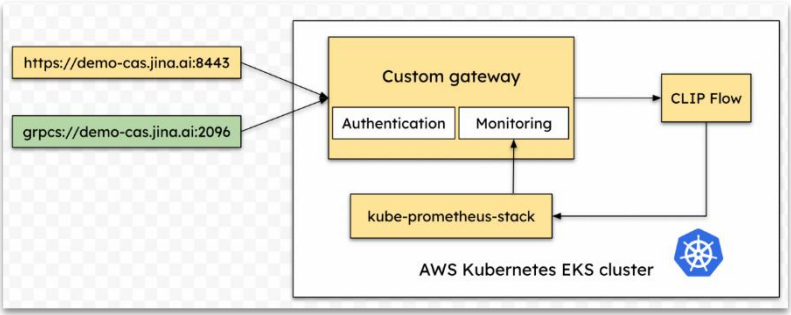
缺点：部署流程复杂，维护成本高，并且存在两套服务，资源消耗大

Version 2：集成Jina custom gateway的能力，只需要部署一套服务

优点：同时支持多种协议，大大降低部署成本和资源消耗



Version 1



Version 2

模型推理服务的现状和挑战——请求模式


```python
async def aencode(self, content, **kwargs):
```

```python
async def arank(
    self, docs: Union['DocumentArray', Iterable['Document']], **kwargs
) -> 'DocumentArray':
```

```python
async for _ in self._async_client.post(
    on=f'/encode/{model_name}'.rstrip('/'),
    **self._get_post_payload(content, results, **kwargs),
    on_done=on_done,
    on_error=on_error,
    on_always=partial(self._update_pbar, func=on_always),
    parameters=parameters,
):
    continue
```

```python
from jina import Executor, requests, dynamic_batching, Flow, DocumentArray, Document
import numpy as np
import torch


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

        # initialize model
        self.model = torch.nn.Linear(in_features=128, out_features=128)

    @requests(on='/bar')
    @dynamic_batching(preferred_batch_size=10, timeout=200)
    def embed(self, docs: DocumentArray, **kwargs):
        docs.embeddings = self.model(torch.Tensor(docs.tensors))

flow = Flow().add(uses=MyExecutor)
```

- client端使用async异步请求
- client端支持用户自定义on_done, on_error, on_always，满足用户的回调需求
- 支持dynamic batching，最大程度利用cpu/gpu资源

模型推理服务的现状和挑战——模型格式

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
```

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:
            tokens = self.tokenizer.encode_plus(doc.text)
            inputs = {name: np.atleast_2d(value) for name, value in tokens.items()}

            output, pooled = self.session.run(None, inputs)
            # assign the encoding results to ``embedding``
            doc.embedding = pooled[0]
```

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()
```

基于Jina的强大能力，可以将各种framework/runtime的模型封装为executor，提供服务接口供外部直接调用

模型推理服务的现状和挑战——推理加速

- 半精度（fp16）推理

半精度推理在分类和检索任务上对性能影响不大，但对显存的需求有明显降低

```
Datasets: CIFAR10   Task: zero-shot classification  acc: fp16(fp32)
-------------------------------------------------------------------------------
|               model            |      acc1      |    acc2/5      |   mean recall |
|--------------------------------|----------------|----------------|---------------|
|          RN50::openai          |  71.55(-0.03) |  98.11(-0.04)  |  71.50(-0.01) |
|    ViT-B-16::laion400m_31      |  91.70(-0.01) |  99.72(0.00)   |  91.69(0.00)  |
|ViT-B-32-quickgelu::laion400m_32|  90.72(+0.03) |  99.79(0.00)   |  90.74(+0.03) |
|    ViT-L-14::laion400m_31      |  94.66(0.00)  |  99.90(0.00)   |  94.67(0.00)  |
|    ViT-H-14::laion400m_31      |  97.45(0.00)  |  99.93(0.00)   |  97.44(+0.01) |
|    ViT-g-14::laion2b_s12b_b42k |  97.06(out of memory)| 99.93   |  97.08        |
-------------------------------------------------------------------------------

Datasets: VOC2007 Task: zero-shot calssification precision: fp16(fp32)
-----------------------------------------------------
|               model            |      precision      |
|--------------------------------|---------------------|
|          RN50::openai          |  74.83750(-0.0052)  |
|    ViT-B-16::laion400m_31      |  78.35778(-0.0043)  |
|ViT-B-32-quickgelu::laion400m_32|  76.27471(+0.0071)  |
|    ViT-L-14::laion400m_31      |  78.53070(+0.0009)  |
|    ViT-H-14::laion2b_s32b_b79k |  80.12784(+0.0039)  |
-----------------------------------------------------
```

```
Datasets: flickr8k   Task: zero-shot retrieval  acc: fp16(fp32)
-------------------------------------------------------------------------------
|               model            |      image_recall@5      |    text_recall@5      |
|--------------------------------|--------------------------|-----------------------|
|          RN50::openai          | 0.5033920886(+0.0006741) |  0.689284384(+-0.00037) gpu: (2477MB/2993MB)
|    ViT-B-16::laion400m_31      | 0.6197503209(0)          |  0.765047609(+0.000247) gpu: (2531MB/3043MB)
|ViT-B-32-quickgelu::laion400m_32| 0.5794092416(+0.0001482) |  0.739710807(+0.000370) gpu: (2691MB/3159MB)
|    ViT-L-14::laion400m_31      | 0.6750710606(+0.0000241) |  0.805833637(0)         gpu: (3607MB/5085MB)
|    ViT-H-14::laion2b-s32b_79k  | 0.7459646463(0)          |  0.856136441(+0.000247) gpu: (5021MB/8591MB)
-------------------------------------------------------------------------------
```

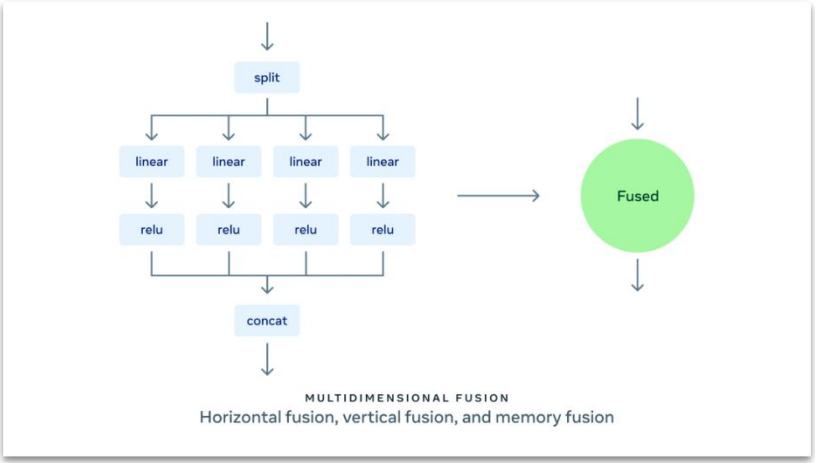
模型推理服务的现状和挑战——推理加速

- AITemplate

把 AI 模型转换成高性能 C++ GPU 模板代码的 Python 框架

该框架在设计上专注于性能和简化系统

AITemplate 系统一共分为两层：前端部分进行图优化，后端部分针对目标 GPU 生成 C++ 模板代码



Tested on RTX3080: Model: `ViT-L-14::laion2b-s32b-b82k`

shape	pt (ms)	ait (ms)	without `flash_attn`	mean idff	max diff
(1, 77)	8.6888	0.8523	1.6269	0.00335	0.01758
(2, 77)	8.7543	0.9854	2.0161	0.00333	0.01782
(4, 77)	8.7231	1.2459	2.8970	0.00358	0.04297
(8, 77)	9.4466	2.0201	4.8552	0.00355	0.03906
(16, 77)	10.0222	3.4399	8.7880	0.00333	0.03906
(1, 224, 224, 3)	18.0799	3.7753	8.4608		
(2, 224, 224, 3)	17.9421		8.4604		

模型推理服务的现状和挑战——推理加速

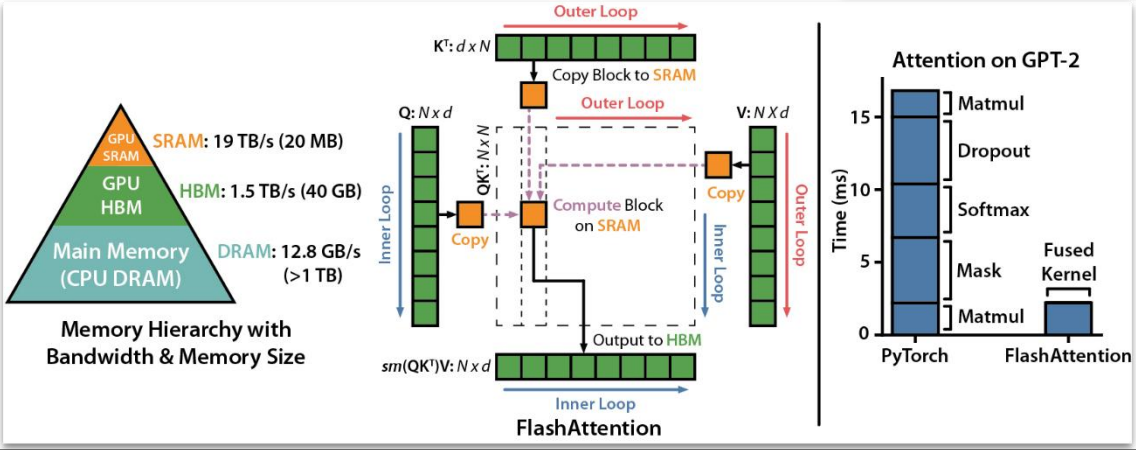
- FlashAttention

传统优化attention策略：优化FLOPS，稀疏近似/低秩近似

FlashAttention：优化GPU memory IO：避免频繁向HBM读写数据，尽量使用SRAM完成计算

Model: `ViT-L-14::laion2b-s32b-b82k`

shape	baseline(s)	flash_attn(s)	speed up (x)	mean diff
(1, 77)	0.81193	0.66437	1.22211	0.00066
(2, 77)	0.8264	0.70035	1.17998	0.0007
(4, 77)	0.81998	0.69887	1.1733	0.00064
(8, 77)	1.05975	0.85742	1.23597	0.00054
(16, 77)	2.12992	1.68367	1.26504	0.00057
(1, 3, 224, 224)	2.00593	1.34507	1.49131	0.00155
(2, 3, 224, 224)	3.74493	2.29818	1.62952	0.00122
(4, 3, 224, 224)	7.35365	4.44447	1.65456	0.00159
(8, 3, 224, 224)	14.63006	9.05604	1.6155	0.00135
(16, 3, 224, 224)	29.63732	18.29142	1.62028	0.00155



在GPT-2上实现了7倍的加速，并且内存的使用与序列长度成线性关系

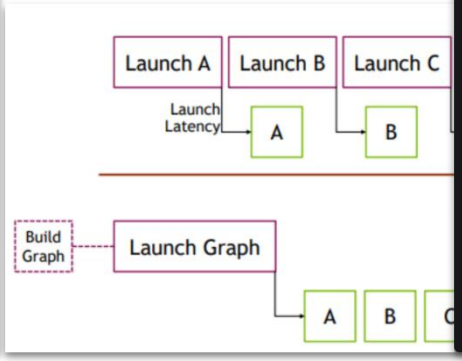
模型推理服务的现状和挑战——推理加速

- torch.fx

用于捕获和转换PyTorch模型，主要有三种用途：

- 修改网络结构
- FX Graph Mode Quantization
- 结合CUDAGraph加速推理

```python
import torch
from torch.fx import symbolic_trace, GraphModule

def my_func(x):
  return torch.relu(x).neg()

# Program capture via symbolic tracing
traced : GraphModule = symbolic_trace(my_func)
for n in traced.graph.nodes:
  print(f'{n.name} = {n.op} target={n.target} args={n.args}')
"""
x = placeholder target=x args=()
relu = call_function target=<built-in method relu ...> args=(x,)
neg = call_method target=neg args=(relu,)
output = output target=output args=(neg,)
"""

print(traced.code)
"""
def forward(self, x):
    relu = torch.relu(x);  x = None
    neg = relu.neg();  relu = None
    return neg
"""
```

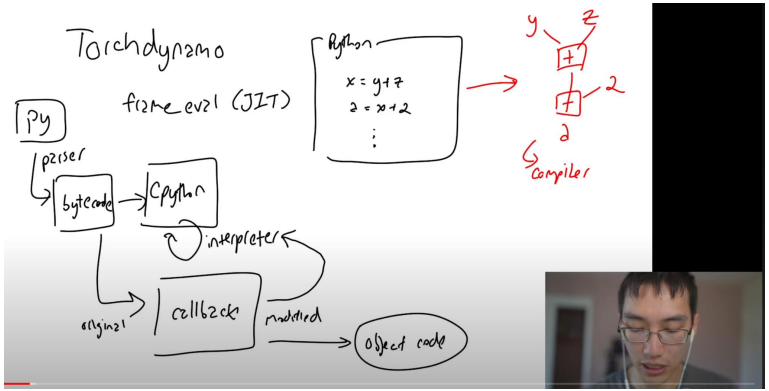



```python
import torch
from torch.fx import symbolic_trace

from clip_server.model.openclip_model import OpenCLIPModel

model = OpenCLIPModel(name='Vit-B-32::openai',
device='cuda')._model_vision

# Symbolic tracing frontend - captures the semantics of the module
graph_module: torch.fx.GraphModule = symbolic_trace(model)

static_inputs = [torch.zeros_like(x, device='cuda') for x in
example_inputs]

stream = torch.cuda.Stream()
stream.wait_stream(torch.cuda.current_stream())

# Build CUDAGraph
graph = torch.cuda.CUDAGraph()
with torch.cuda.graph(graph, stream=stream,
pool=torch.cuda.graph_pool_handle()):
    static_outputs = graph_module(*static_inputs)
if not isinstance(static_outputs, (list, tuple)):
    static_outputs = (static_outputs,)


# forward with CUDAGraph
def run(*new_inputs):
    for dst, src in zip(static_inputs, new_inputs):
        dst.copy_(src)  # cuda graph can only read data from the same
address
    graph.replay()
    return static_outputs
```

模型推理服务的现状和挑战——推理加速

- TorchDynamo

在bytecode层面修改并生成fx.graph



https://www.youtube.com/watch?v=egZB5Uxki0I

```python
def optimize_model_dynamo(
        original_model: Union[CLIPTextTransformer, CLIPVisionTransformer],
        pool = torch.cuda.graph_pool_handle()
    ) -> Callable:

    def compiler(gm: torch.fx.GraphModule, example_inputs: List[torch.Tensor]):
        return cuda_graphs_wrapper(gm, example_inputs, pool=pool)

    @torchdynamo.optimize(compiler)
    def run(*args, **kwargs):
        return original_model.forward(*args, **kwargs)

    return run
```

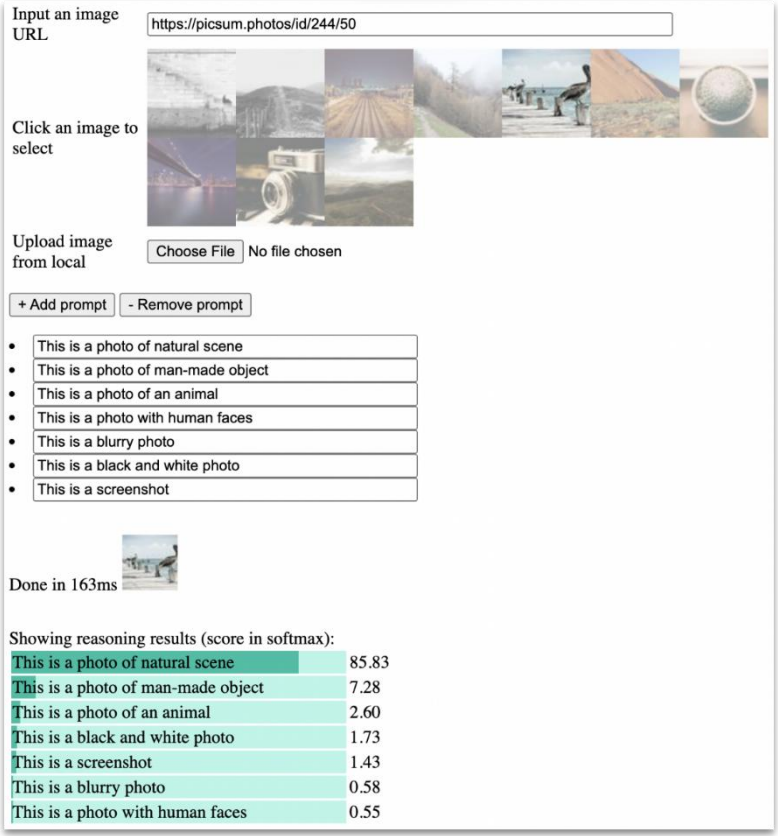
Performance on TextModel:

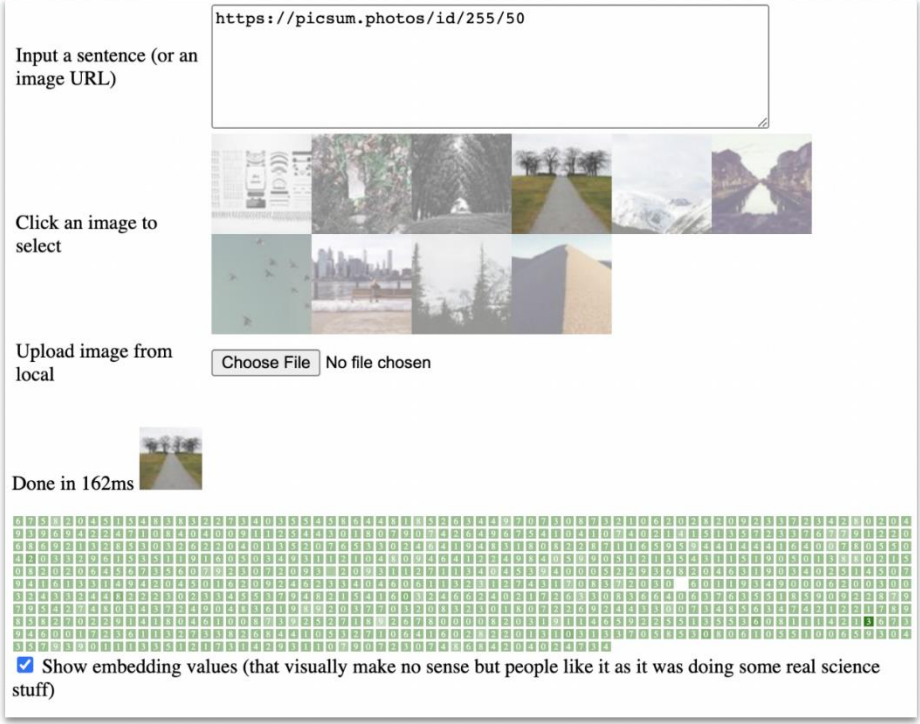
times	shape	pt (s)	graph (s)	speed up	GPU(MB)	RES(MB)
1	(1, 77)	0.0068	0.00182/0.00309	3.6558/2.1231	1857/3011/2993	-
	(2, 77)	0.0073	0.00216/0.00419	3.0058/1.6446	1859/3005/3132	-
	(4, 77)	0.0070	0.00294/0.00753	2.2076/0.9188	1859/3087/3043	-
	(8, 77)	0.0071	0.00450/0.01200	2.3033/0.5882	1863/3369/3315	-
	(16, 77)	0.0087	0.00791/0.02126	1.0675/0.4427	1883/3619/3591	-

将在明年的torch2.0中正式发布，目前已经整合到torch的main branch中，可参考：https://pytorch.org/docs/master/dynamo/get-started.html

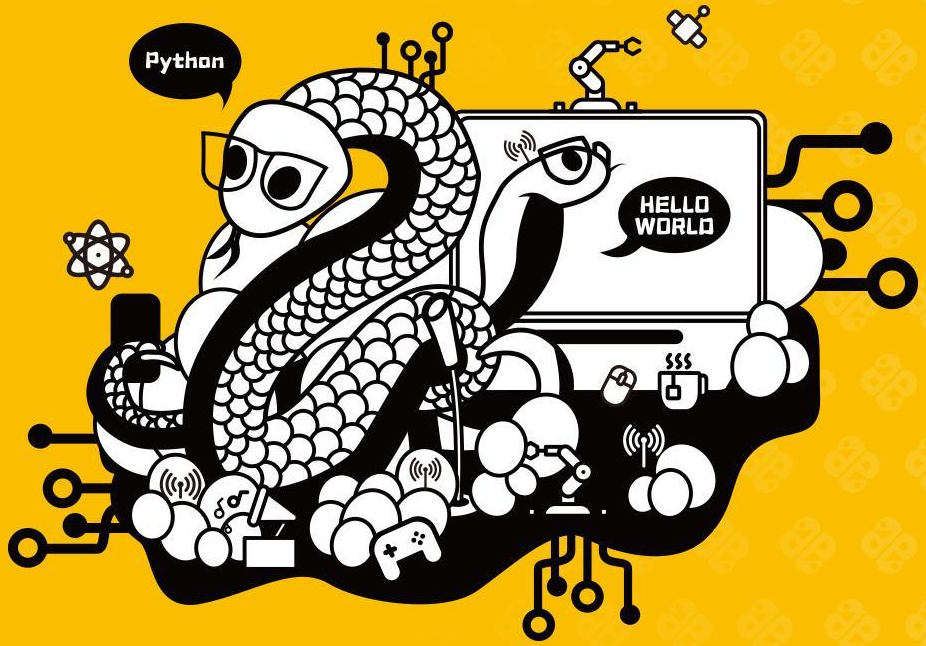
目录

demo: CLIP-as-service





https://clip-as-service.jina.ai/playground/embedding/#



Thanks!

感谢观看