DocArray:
为机器学习而生的数据结构

 felix.wang@jina.ai

Github jina-ai/docarray

About Me

2021-now,	Senior AI Engineer, *Jina AI*
2020-21,	Senior Researcher, *Huya AI*
2018-19,	Senior Researcher, *Tencent AI*
2011-18,	Ph. D., *Hong Kong Baptist University*

王峰, 开源神经搜索框架 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。

About Jina AI

- 👼 Founded in 2020/2
- 👫 50 members
- 🌎 4 offices
- 💵 Raised $38M+
- 🏆 Top-tier AI company

Fundamental

What is DocArray?

jina-ai / docarray Public Fork 66 Starred 729

README.md

docarray

The data structure for unstructured data

Release v0.13.23 coverage 87% downloads 81k/month Slack 3.1k

DocArray is a library for nested, unstructured data in transit, including text, image, audio, video, 3D mesh, etc. It allows deep-learning engineers to efficiently process, embed, search, recommend, store, and transfer the data with a Pythonic API.

Rich data types: super-expressive data structure for representing complicated/mixed/nested text, image, video, audio, 3D mesh data.

Pythonic experience: designed to be as easy as a Python list. If you know how to Python, you know how to DocArray. Intuitive idioms and type annotation simplify the code you write.

Data science powerhouse: greatly accelerate data scientists' work on embedding, matching, visualizing, evaluating via Torch/TensorFlow/ONNX/PaddlePaddle on CPU/GPU.

Data in transit: optimized for network communication, ready-to-wire at anytime with fast and compressed serialization in Protobuf, bytes, base64, JSON, CSV, DataFrame.

Scale to big data: handle out-of-memory data via on-disk document store while staying with exact same API experience. Supporting classic databases and vector databases to enable faster nearest neighbour search.

For modern apps: GraphQL support makes your server versatile on request and response; built-in data validation and JSON Schema (OpenAPI) help you build reliable webservices.

Integrate with IDE: pretty-print and visualization on Jupyter notebook & Google Colab; comprehensive auto-complete and type hint in PyCharm & VS Code.

What is DocArray?

Bridging the gap between prototyping and production

TensorFlow

PyTorch

scikit learn

VS

Flask

FastAPI

gRPC

prototyping

production

What is DocArray?

Design Objectives

- DocArray is designed to be extremely intuitive for Python users, no new syntax to learn. If you know how to Python, you know how to DocArray.
- DocArray is designed to maximize the local experience, with the requirement of cloud readiness at anytime.
- DocArray is designed to represent multimodal data intuitively to face the ever-increasing development of multi/cross-modal applications.

Who will benefit?

- If you are a *data scientist* who works with image, text, video, audio data in Python all day, you should use DocArray: it can greatly accelerate the work on representing, embedding, matching, visualizing, evaluating, sharing data; while stay close with your favorite toolkits, e.g. Torch, Tensorflow, ONNX, PaddlePaddle, JupyterLab, Google Colab.
- If you are a *deep learning engineer* who works on scalable deep learning service, you should use DocArray: it can be the basic building block of your system. Its portable data structure can be wired in Protobuf, compressed bytes, JSON; allowing your engineer friends to happily integrate it into the production system.

Fundamental

Three Concepts

Document: a data structure for easily representing nested, unstructured data.

DocumentArray: a container for efficiently accessing, processing, and understanding multiple Documents.

Dataclass: a high-level API for intuitively representing multimodal data.

What is Document?

Simple Document

- **Content related**: uri, text, tensor, blob;
- **Common side information or metadata**: id, modality, mime_type, offset, location, weight;
  - Further information: tags;

```
from docarray import Document
import numpy

d1 = Document(text='hello')
d2 = Document(blob=b'\f1')
d3 = Document(tensor=numpy.array([1, 2, 3]))
d4 = Document(uri='https://jina.ai',
              mime_type='text/plain',
              granularity=1,
              adjacency=3,
              tags={'foo': 'bar'})
```

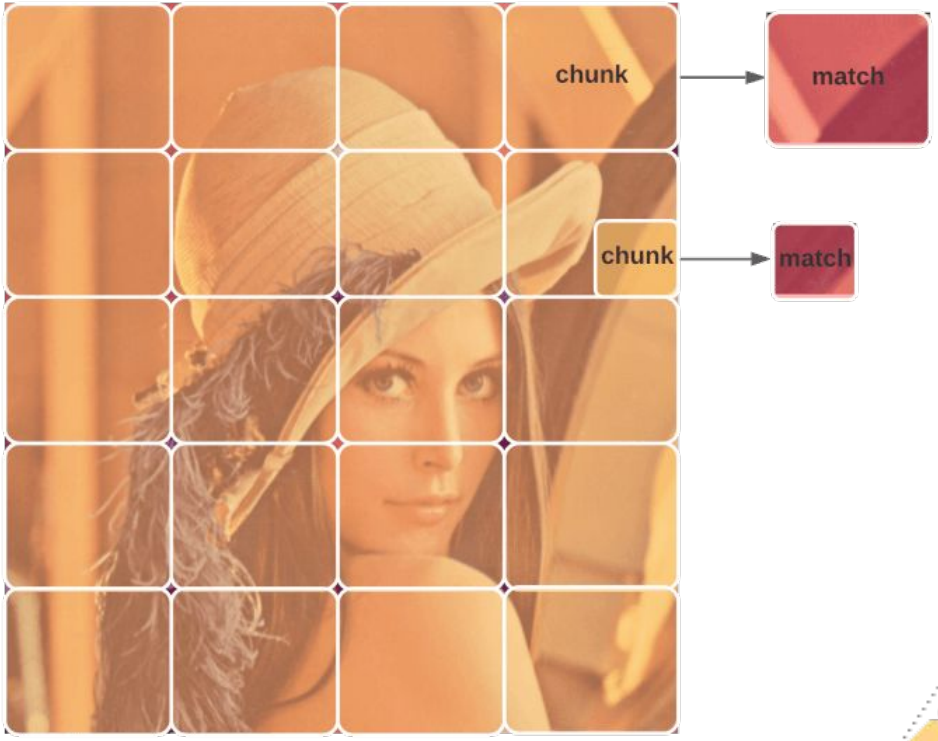
What is Document?

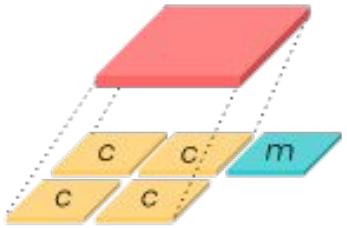
Nested Document

- **Nest structure related**: chunks, matches, granularity, adjacency, parent_id;

```python
from docarray import Document

d = Document(
    id='d0',
    chunks=[Document(id='d1', chunks=Document(id='d2'))],
    matches=[Document(id='d3')],
)
```

Attribute	Description
`doc.chunks`	The list of sub-Documents of this Document. They have `granularity + 1` but same `adjacency`
`doc.matches`	The list of matched Documents of this Document. They have `adjacency + 1` but same `granularity`
`doc.granularity`	The "depth" of the nested chunks structure
`doc.adjacency`	The "width" of the nested match structure


chunk
match
chunk
match


c
c
m
c
c

What is DocumentArray?

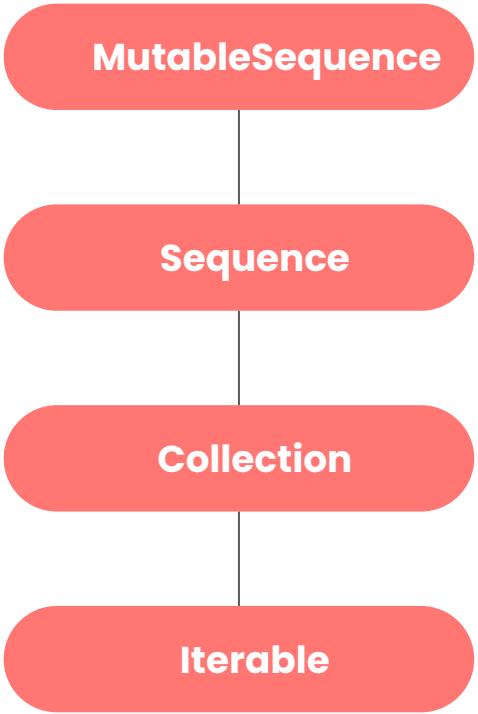
DocumentArray

DocumentArray is a list-like container of Document objects.

- a Python list, as it implements **all** list interfaces.
- It is also powerful as Numpy ndarray and Pandas DataFrame, allowing you to efficiently access elements and attributes of contained Documents.
- greatly accelerate data scientists work on accessing nested elements, evaluating, visualizing, parallel computing, serializing, matching etc.

```
from docarray import DocumentArray

da = DocumentArray()

da.append(Document(text='hello world!'))
da.extend([Document(text='hello'), Document(text='world!')])
```

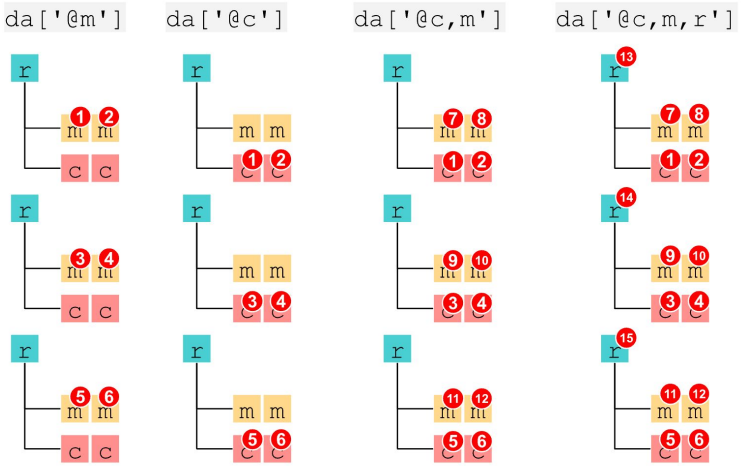
What is DocumentArray?




```python
from jina import Document, DocumentArray

da = DocumentArray([Document() for _ in range(6)])

for j in range(6):
    da[j].scores['sim'] = j

for d in filter(lambda d: d.scores['sim'].value > 2, da):
    print(d)
```

What is DocArray?

Index by nested structure




```
print(da['@m'])
print(da['@c'])
print(da['@c,m'])
print(da['@c,m,r'])
```

```
<DocumentArray (length=6) at 4912623312>
<DocumentArray (length=6) at 4905929552>
<DocumentArray (length=12) at 4913359824>
<DocumentArray (length=15) at 4912623312>
```

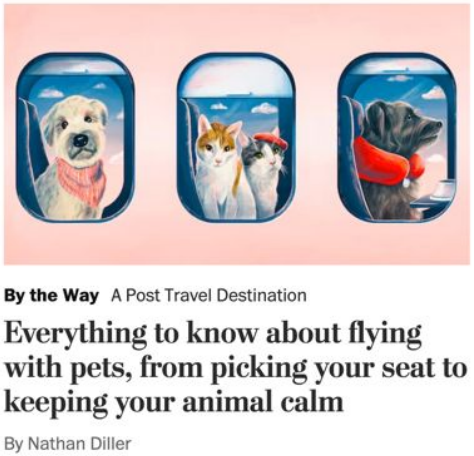
```
from docarray import DocumentArray

da = DocumentArray().empty(3)
for d in da:
    d.chunks = DocumentArray.empty(2)
    d.matches = DocumentArray.empty(2)
```

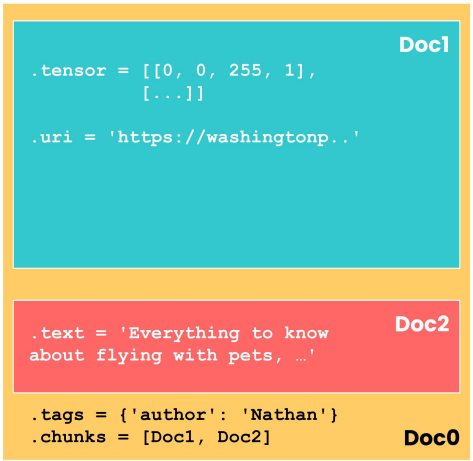
What is Dataclass?

Dataclass

DocArray's dataclass is a high-level API for representing a multimodal document using nested Document structure.

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


@dataclass
class WPArticle:
    banner: Image
    headline: Text
    meta: JSON


a = WPArticle(
    banner='dog-cat-flight.png',
    headline='Everything to know about flying with pets, from picking your seats to keeping your animal calm',
    meta={
        'author': 'Nathan Diller',
        'column': 'By the Way - A Post Travel Destination',
    },
)
```





Integrations

Jupyter Notebook/Colab

Many data scientists work with Jupyter Notebook or Google Colab especially at the early prototyping stage.

```
In [1]: import numpy as np

        from docarray import Document

        d = Document(embedding=np.random.random(100))

        d.chunks.append(Document(text='children', embedding=np.random.random(100)))

        d
```

Document: 47aa2d19b629b69897206071760fa99d

Attribute	Value
embedding	[redacted]

└─ **Chunks**
  └─ **Document:** 9e8bab23105d1aac48a54e9dec85253e

Attribute	Value
parent_id	47aa2d19b629b69897206071760fa99d
granularity	1
text	children
embedding	[redacted]

```
In [2]: from docarray import Document

        d = Document(uri='left/00001.jpg').load_uri_to_image_tensor()
        d.display()
```

Integrations

Deep Learning Frameworks

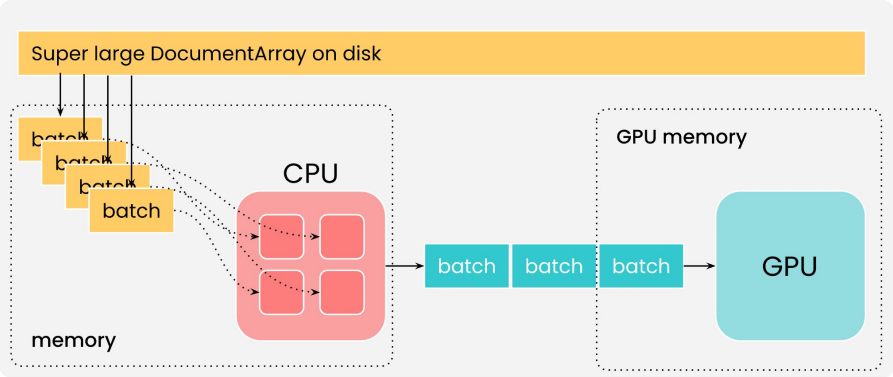
DocArray can be easily integrated into PyTorch, Tensorflow, PaddlePaddle frameworks.

```python
import numpy as np
import paddle
import torch

from docarray import Document, DocumentArray

emb = np.random.random([10, 3])

da = DocumentArray(
    [
        Document(embedding=emb),
        Document(embedding=torch.tensor(emb).to_sparse()),
        Document(embedding=torch.tensor(emb)),
        Document(embedding=paddle.to_tensor(emb)),
    ]
)

da.save_binary('test.protobuf.gz')
```

Integrations

Deep Learning Frameworks

dataloader: Load, map, batch in one-shot

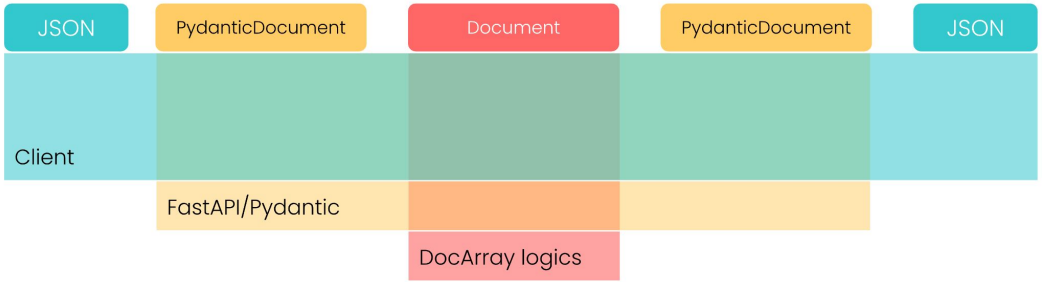



```python
import time

from docarray import DocumentArray


def cpu_job(da):
    time.sleep(2)
    print('cpu job done')
    return da


def gpu_job(da):
    time.sleep(1)
    print('gpu job done')


for da in DocumentArray.dataloader(
    'da.protobuf.gz', func=cpu_job, batch_size=64, num_worker=4
):
    gpu_job(da)
```

Integrations

FastAPI/Pydantic

DocArray supports **pydantic data model** via PydanticDocument and PydanticDocumentArray.




```python
from docarray import Document, DocumentArray

from fastapi import FastAPI

app = FastAPI()


@app.post('/single')
async def create_item(item: PydanticDocument):
    d = Document.from_pydantic_model(item)
    # now `d` is a Document object
    ...  # process `d` how ever you want
    return d.to_pydantic_model()


@app.post('/multi')
async def create_array(items: PydanticDocumentArray):
    da = DocumentArray.from_pydantic_model(items)
    # now `da` is a DocumentArray object
    ...  # process `da` how ever you want
    return da.to_pydantic_model()
```

Example

Visual Search

Use the Totally Looks Like dataset to build a simple image search.

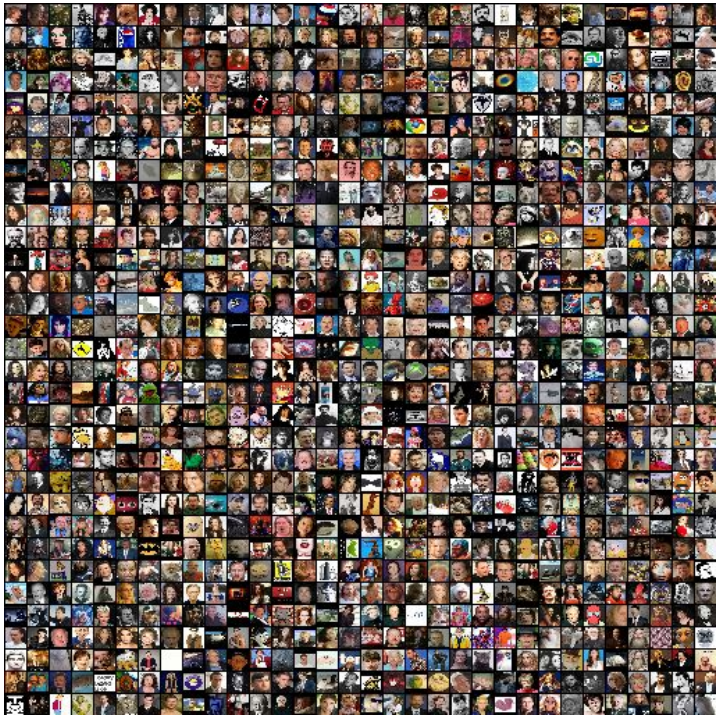
left/00018.jpg	right/00018.jpg	left/00131.jpg	right/00131.jpg
			

Example

Visual Search

Load images

```
from docarray import DocumentArray

left_da = DocumentArray.from_files('left/*.jpg')


left_da.plot_image_sprites()
```



Example

Visual Search

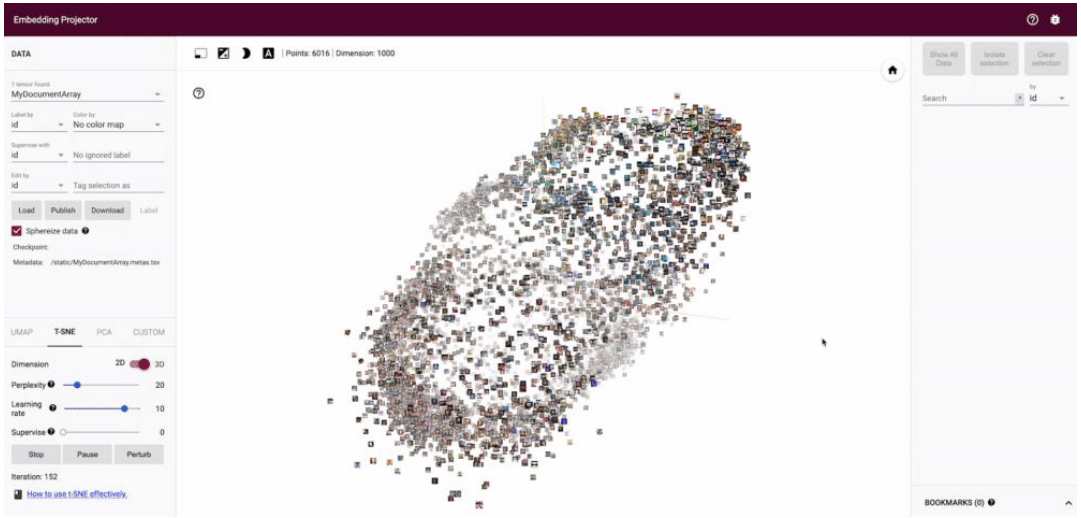
Apply preprocessing

```python
from docarray import Document


def preproc(d: Document):
    return (
        d.load_uri_to_image_tensor()  # load
        .set_image_tensor_normalization()  # normalize color
        .set_image_tensor_channel_axis(-1, 0)
    )  # switch color axis for the PyTorch model later


left_da.apply(preproc)
```

Example

Visual Search

Embedded images



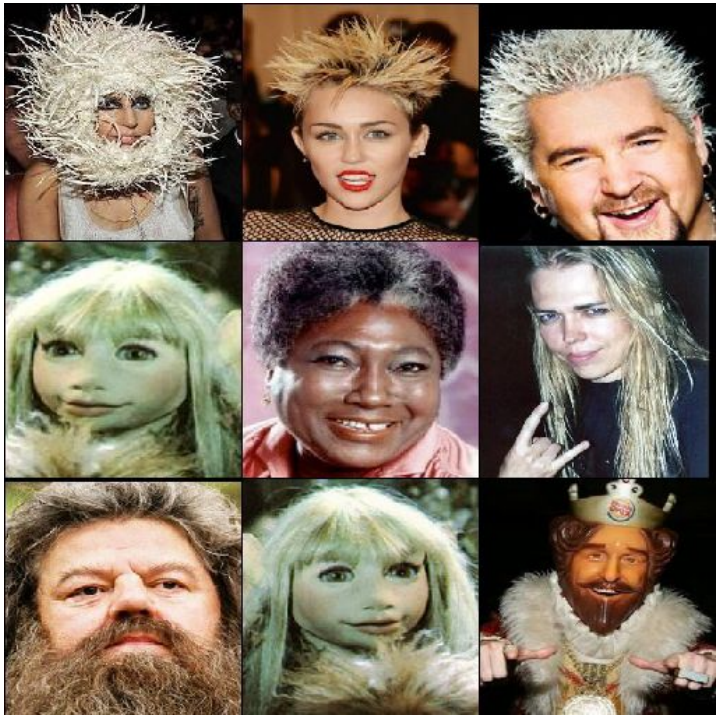

```
import torchvision

model = torchvision.models.resnet50(pretrained=True)  # load ResNet50
left_da.embed(model, device='cuda')  # embed via GPU to speed up

left_da.plot_embeddings()
```

Example

Visual Search

Match nearest neighbours

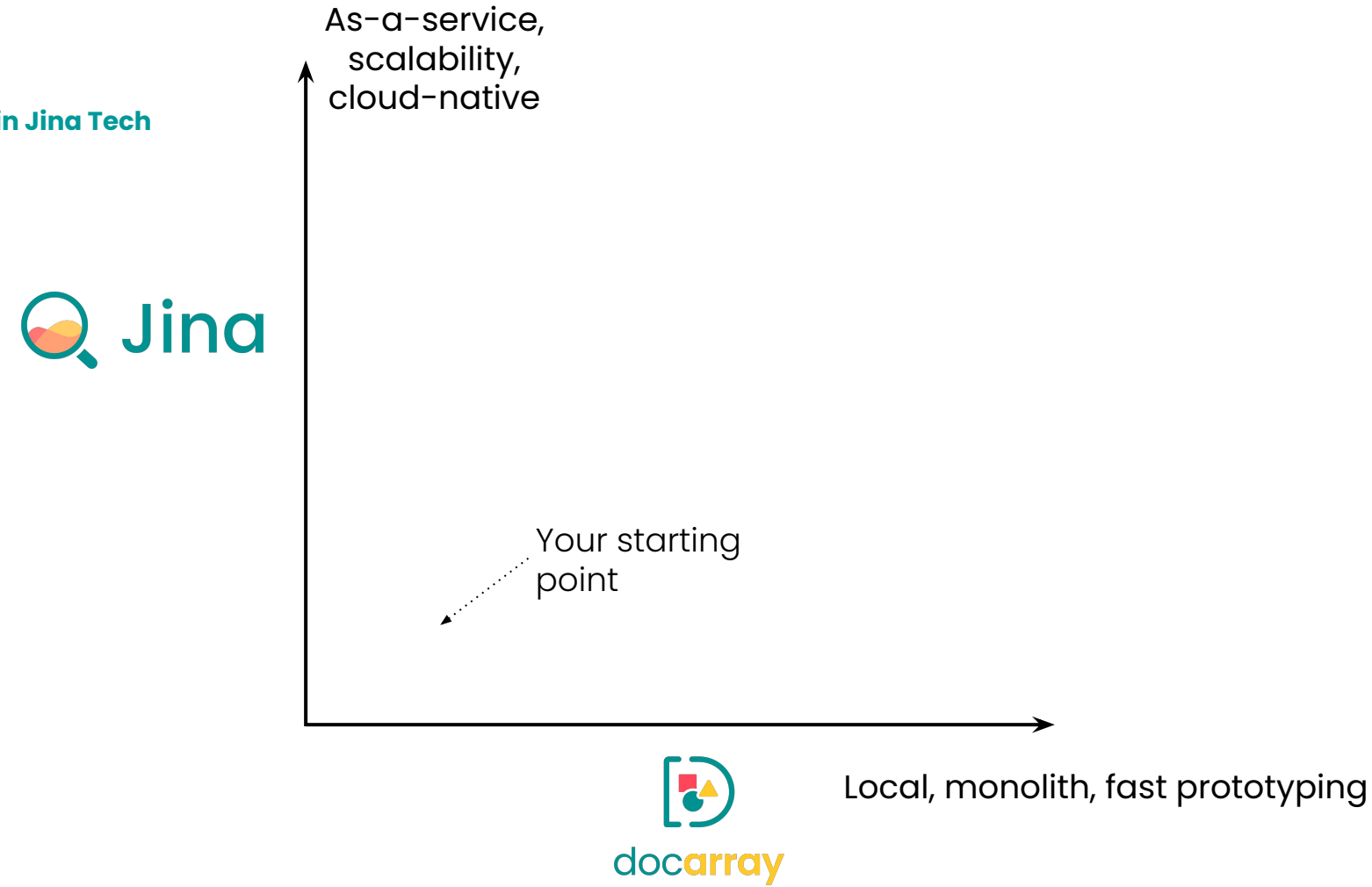
```
left_da.match(right_da, limit=9)

print(left_da['@m', ('uri', 'scores__cosine__value')])


left/02262.jpg right/03459.jpg 0.21102
left/02262.jpg right/02964.jpg 0.13871843
left/02262.jpg right/02103.jpg 0.18265384
left/02262.jpg right/04520.jpg 0.16477376
...
```



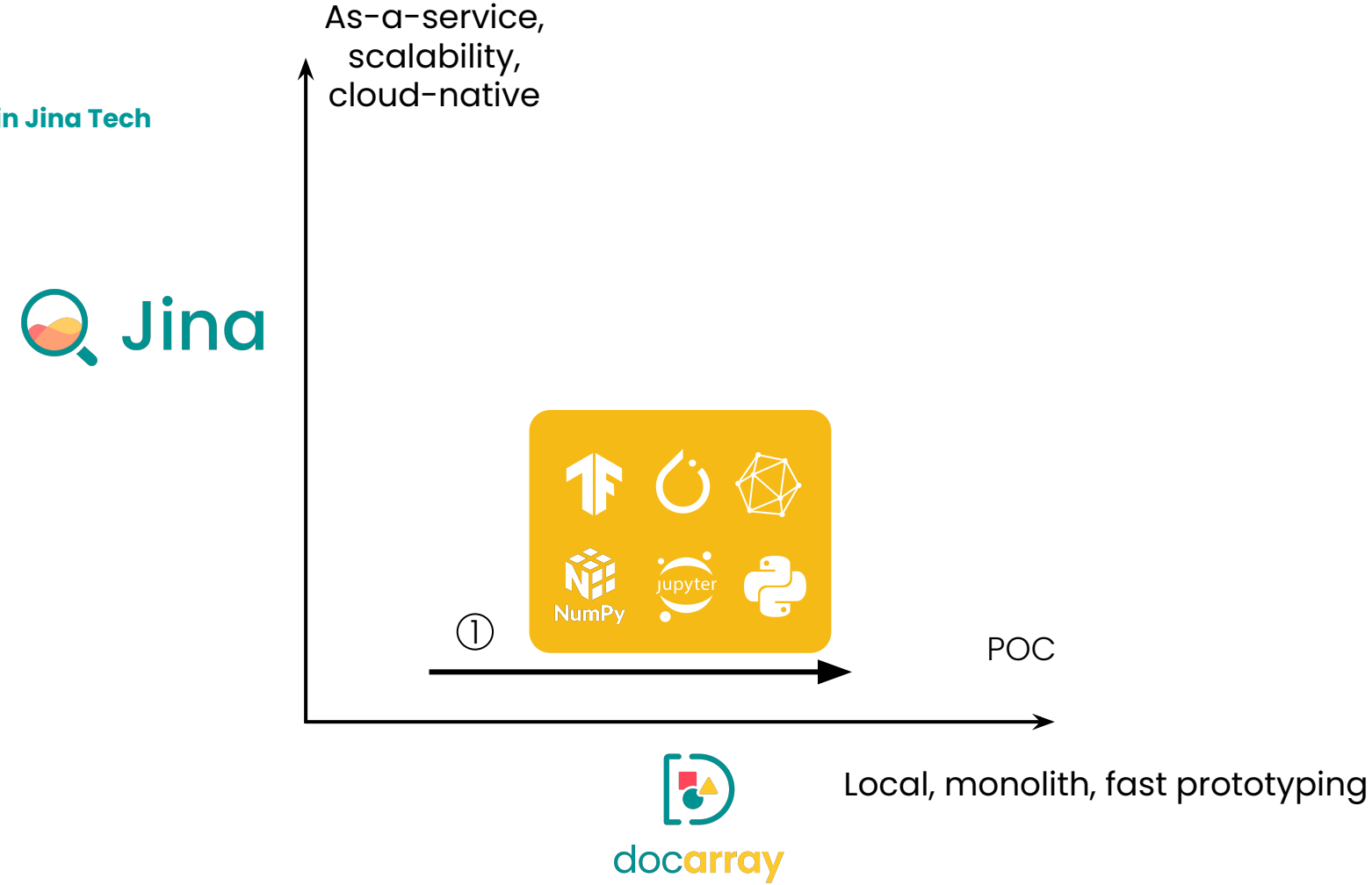
Jina Tech Stacks

DocArray	Data structure for unstructured data
Jina	Cloud-Native Neural Search Framework for Any Kind of Data
Jina Hub	Share and discover building blocks for neural search applications
Finetuner	Finetune deep neural networks for better embedding on neural search tasks
CLIP-as-service	Embed images and sentences into fixed-length vectors with CLIP
Now	One line to host them all. Bootstrap your image search case in minutes.
JCloud	Deploy and manage Jina projects on the cloud

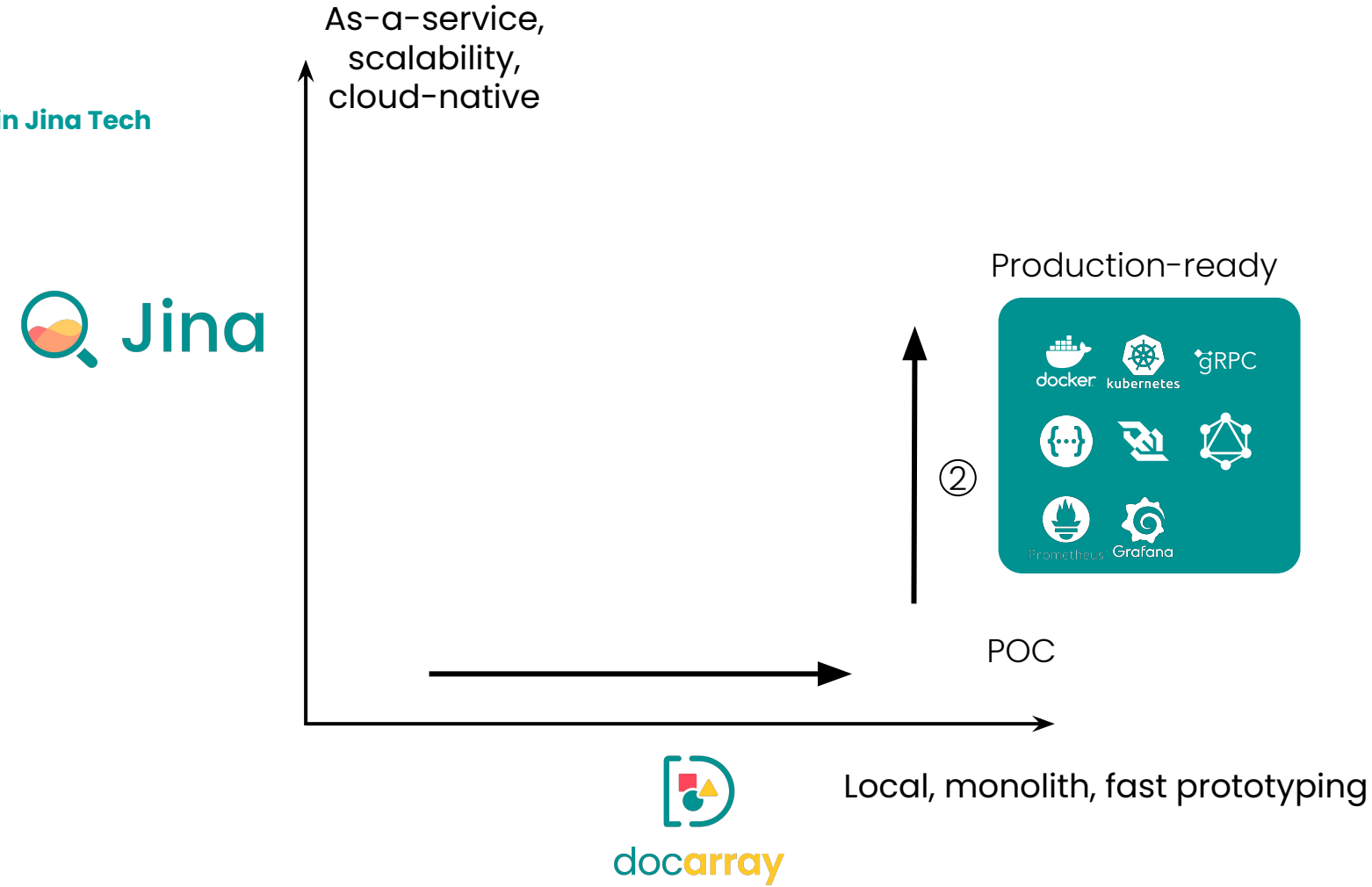
Docarray in Jina Tech Stack



Docarray in Jina Tech Stack

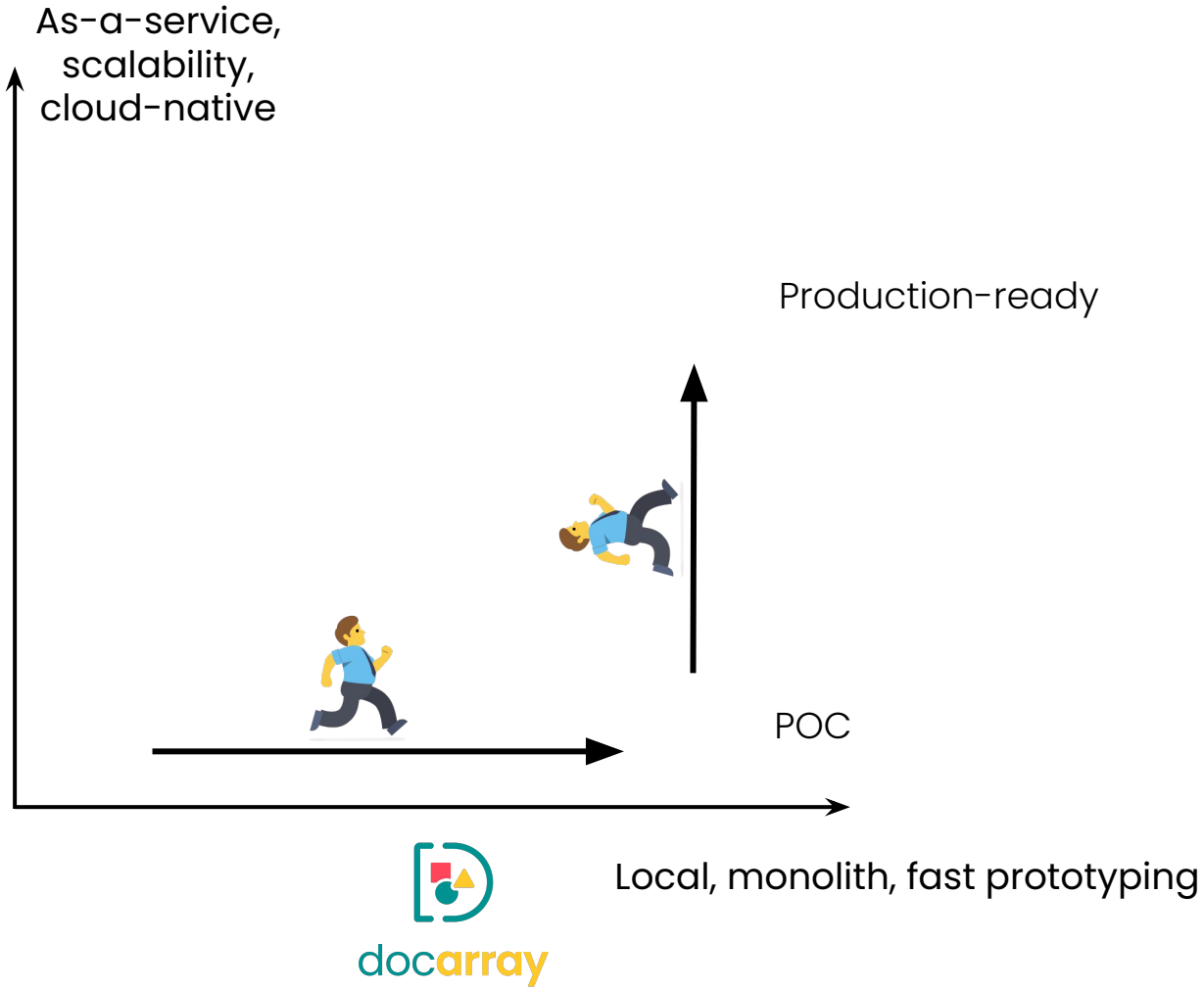


Docarray in Jina Tech Stack



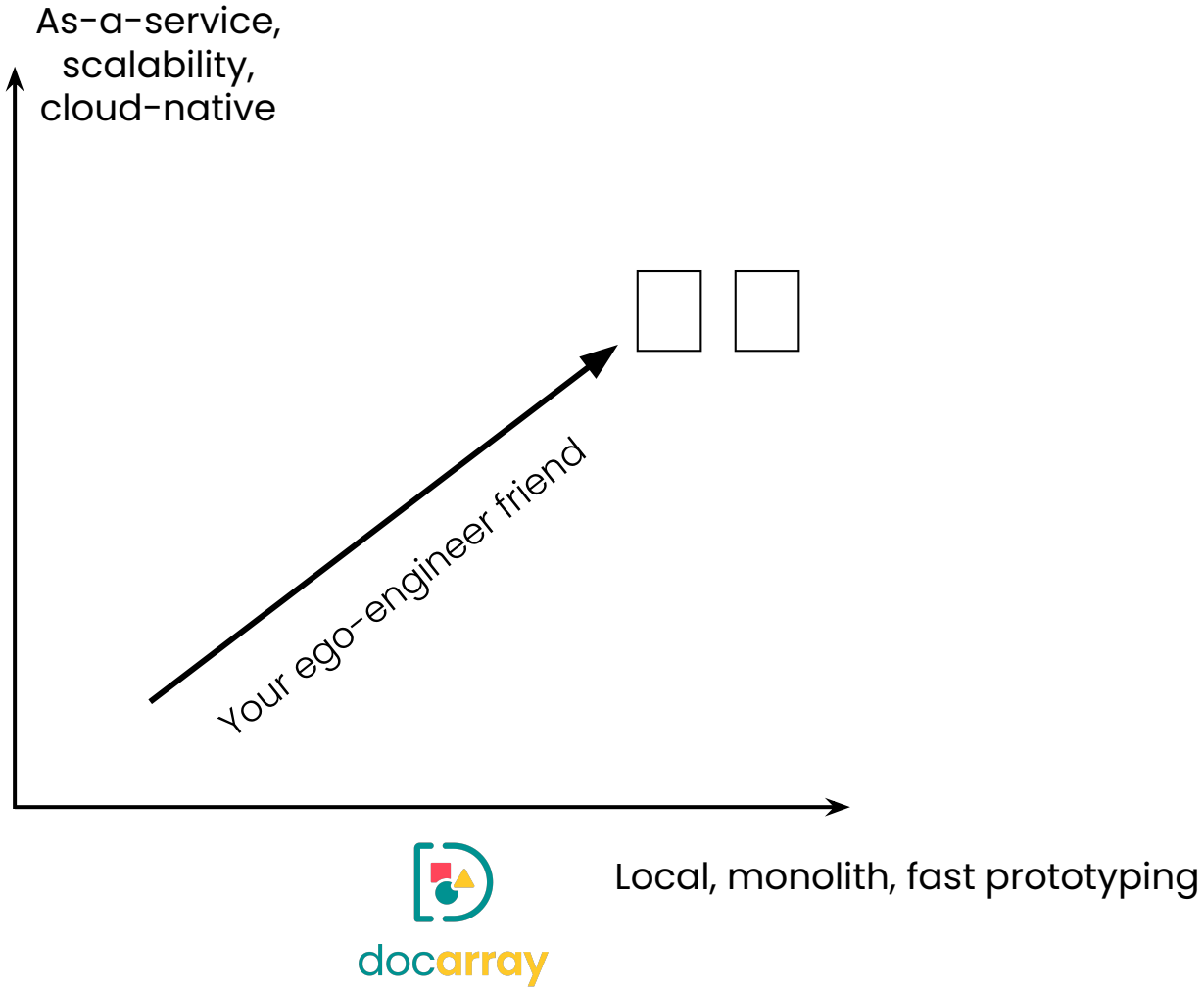
Docarray in Jina Tech Stack

Jina



Docarray in Jina Tech Stack

Jina



Fundamental

3 Lines for *what?*

```python
class Document:
    text: str = ''

def foo(docs, **kwargs):
    for d in docs:
        d.text += 'hello, world!'

docs = [Document(), Document()]
foo(docs)
foo(docs)
for d in docs:
    print(d.text)
```

```python
#!pip install jina
from jina import DocumentArray, Executor, Flow, requests

class MyExec(Executor):

    @requests
    async def foo(self, docs: DocumentArray, **kwargs):
        for d in docs:
            d.text += 'hello, world!'

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

with f:
    r = f.post('/', DocumentArray.empty(2))
    print(r.texts)
```

Fundamental

Hello, world

```python
#!pip install jina
from jina import DocumentArray, Executor, Flow, requests

class MyExec(Executor):

    @requests
    async def foo(self, docs: DocumentArray, **kwargs):
        for d in docs:
            d.text += 'hello, world!'

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

with f:
    r = f.post('/', DocumentArray.empty(2))
    print(r.texts)
```

Fundamental

Imports

```
#!pip install jina
from jina import DocumentArray, Executor, Flow, requests

class MyExec(Executor):

    @requests
    async def foo(self, docs: DocumentArray, **kwargs):
        for d in docs:
            d.text += 'hello, world!'

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

with f:
    r = f.post('/', DocumentArray.empty(2))
    print(r.texts)
```

Fundamental

Define Executor

```
#!pip install jina
from jina import DocumentArray, Executor, Flow, requests

class MyExec(Executor):

    @requests
    async def foo(self, docs: DocumentArray, **kwargs):
        for d in docs:
            d.text += 'hello, world!'

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

with f:
    r = f.post('/', DocumentArray.empty(2))
    print(r.texts)
```

Fundamental

Build Flow

```python
#!pip install jina
from jina import DocumentArray, Executor, Flow, requests

class MyExec(Executor):

    @requests
    async def foo(self, docs: DocumentArray, **kwargs):
        for d in docs:
            d.text += 'hello, world!'

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

with f:
    r = f.post('/', DocumentArray.empty(2))
    print(r.texts)
```

Fundamental

Send/Recv Data

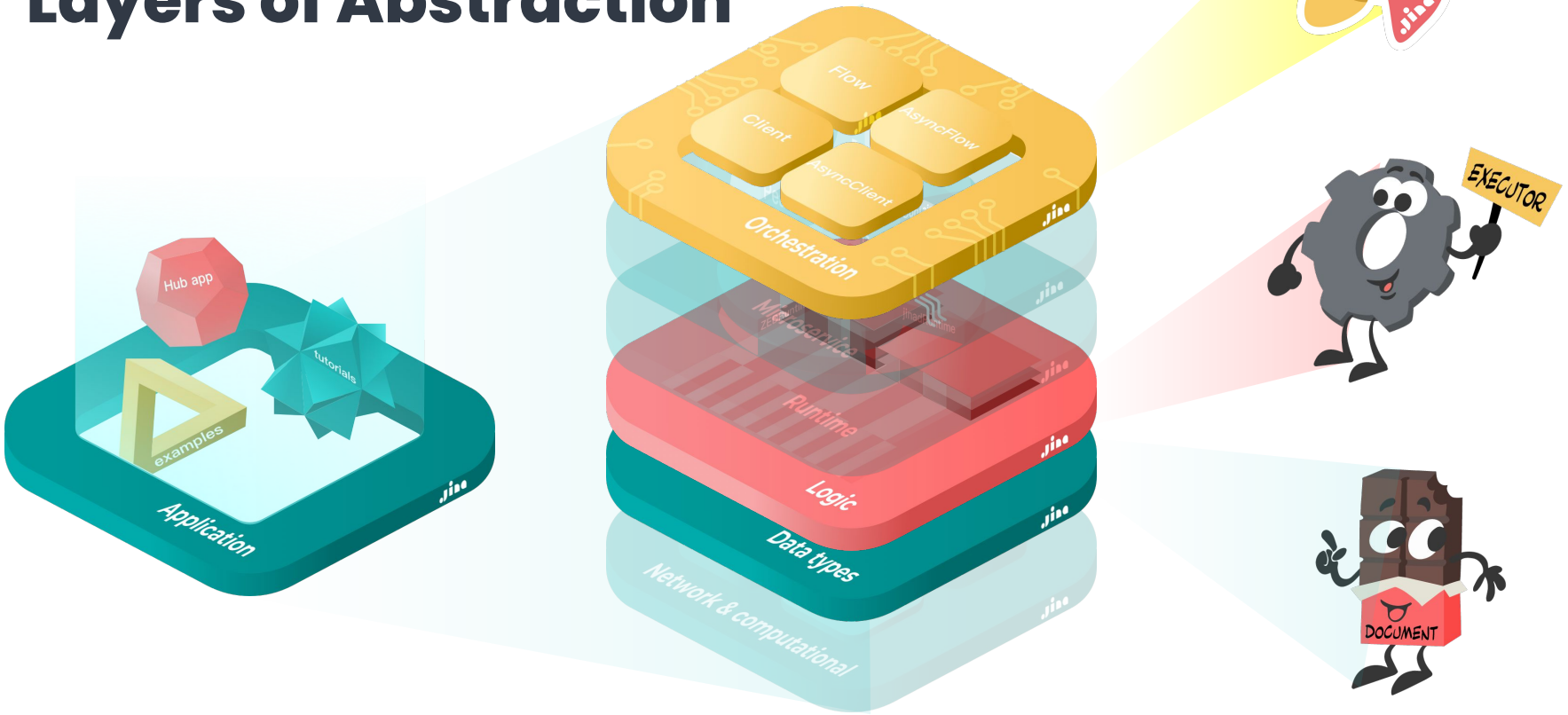
```python
#!pip install jina
from jina import DocumentArray, Executor, Flow, requests

class MyExec(Executor):

    @requests
    async def foo(self, docs: DocumentArray, **kwargs):
        for d in docs:
            d.text += 'hello, world!'

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

with f:
    r = f.post('/', DocumentArray.empty(2))
    print(r.texts)
```

Fundamental

Run it!

Design of Jina

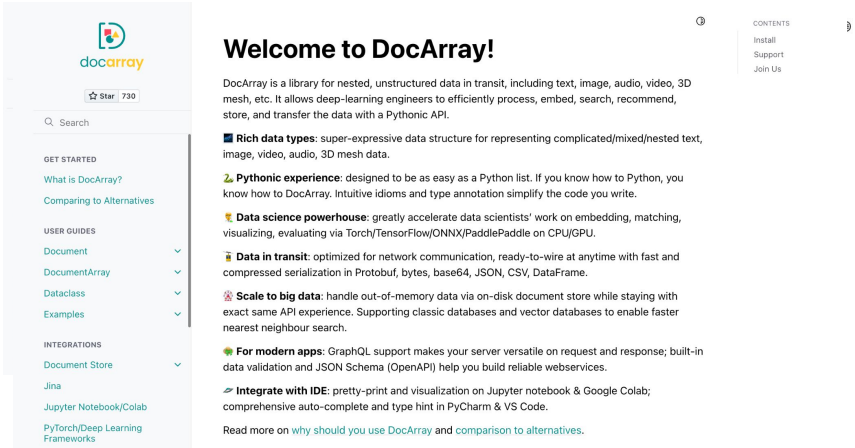
Layers of Abstraction



Next Steps

Learning Bootcamp

learn.jina.ai



Docs

docarray.jina.ai

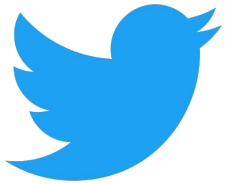
Stay up to date

Get in touch

get.jina.ai

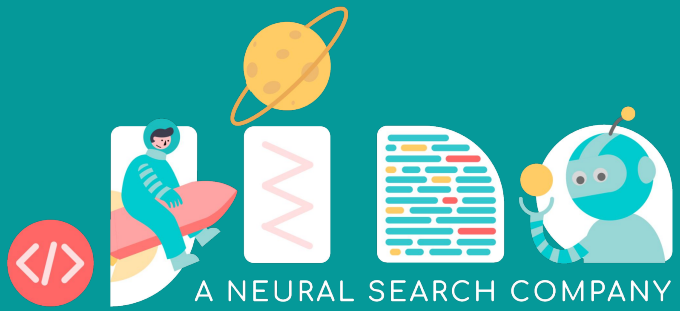
slack.jina.ai

Jina AI



@JinaAI_

Jina AI



Global jina.ai @JinaAI_ get.jina.ai slack.jina.ai