Modern, Typed Python for (Multimodal) ML

From Training to Deployment

Johannes Messner

PyCon US 2023

CONTENTS

01 Part 1: Typing in Python

02 Part 2: Python Type Hints in ML

About Jina AI

- 👼 Founded Feb 2020
- 👭 60+ members
- 🌎 4 offices
- 💵 Raised $38M+
- 🏆 Top-tier AI company

Is Python a typed programming language?

.jina

Python is a duck typed programming language

Python is a duck typed programming language

What is Duck Typing?

Static Typing (Rust / C++ / Java ...)

```rust
fn add(x: usize, y: usize) -> usize {
    x+y
}
```

Function signature includes types

Duck Typing (Python)

```python
def add(x, y):
    return x + y
```

Anything goes!

What is Duck Typing?

Static Typing (Rust / C++ / Java ...)

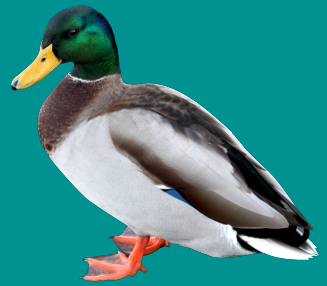
```rust
fn add(x: usize, y: usize) -> usize {
    x+y
}
```

Function signature includes types

Duck Typing (Python)

```python
def add(x, y):
    return x + y
```

Anything goes?

The  Test

*"If it **walks like a duck** and it **quacks like a duck**, then it **must be a duck**"*

The Duck Test in Python

```python
def add(x, y):
    return x + y
```

What is Duck Typing ?

It’s what makes Python great!

PROS:

- Simple
- No need to think about abstractions
- Concise

What is Duck Typing ?

It’s what makes Python great!

PROS:

- Simple
- No need to think about abstractions
- Concise

CONS:

- Hard to maintain
- Fail at runtime, not compile time

DYNAMIC TYPING
STATIC TYPING
DONE!

So what about Type Hints?

What is Duck Typing?

Static Typing (Rust / C++ / Java ...)

```rust
fn add(x: usize, y: usize) -> usize {
    x+y
}
```

Python with Type Hins

```python
def add(x: int, y: int) -> int:
  return x + y
```

Python with type hints

```python
from typing import Optional, Union


class MyClass:
    def my_func(self, x: Union[str, int]) -> str:
        ...

def f(obj: Optional[MyClass] = None) -> None:
    if obj is not None:
        obj.my_func(0)
```

- **Communicate** with
  - Coworkers
  - Future self
  - **IDEs**
- Find bugs early
- *Compatible with duck typing*

History of Python and Types

History of Python and type hints

Python 1.0

1991 : First release of Python
by **Guido van Rossum**

No type hints :/

History of Python and type hints

Python 2.0

2000 : First release of **Python 2.0** by **PythonLab**

Python Logo at this time (on windows)

No type hints :/

History of Python and type hints

Python 3.0

2006 : Development of **Python 3.0** by **PythonLab**

Python Logo at this time (on windows)

No type hints :/

History of Python and type hints

But the people wanted type information!

```python
def add(x, y):
    """
    :param x: int
    :param y: int
    :return: int
    """
    return x + y
```

History of Python and type hints

2006: Python3 PEP 3107

PEP3107 introduced function annotation (!= Type hint)

```python
def add(x, y):
    """
    :param x: int
    :param y: int
    :return: int
    """
    return x + y
```

→

```python
def add(x: 'int', y: 'int') -> 'int':
    return x + y
```

History of Python and type hints

2006: Python3 PEP 3107

PEP3107 introduced **function annotations**

```
def add(x: 'int', y: 'int') -> 'int':
    return x + y

 print(add.__annotation__)
>>> {'a': 'int',
 'B': 'int',
 'return': 'int'}
```

1. Completely **optional**
2. Associate **arbitrary Python expressions** with various parts of a function

But still no type hinting :/

8 years later

History of Python and type hints

2014: Python 3.5 PEP 484

Type hints are introduced in Python 🍾 🚀

History of Python and type hints

2014: Python 3.5 PEP 484

Type hints are introduced in Python 🍾 🚀

A 10 year long story …

All Things Pythonic

Adding Optional Static Typing to Python

by Guido van van Rossum

December 23, 2004

Summary

Optional static typing has long been requested as a Python feature. It's been studied in depth before (e.g. on the type-sig) but has proven too hard for even a PEP to appear. In this post I'm putting together my latest thoughts on some issues, without necessarily hoping to solve all problems.

screenshot

History of Python and type hints

2014: Python 3.5 PEP 484

```python
from typing import Any, Optional

def stringify(x: Any, upper: Optional[bool] = None) -> str:
    if upper:
        return str(x).upper()
    else:
        return str(x)
```

History of Python and type hints

2014: Advanced typing features

```python
from typing import TypeVar

T = TypeVar('T')

def add(x: T, y: T) -> T:
    return x + y
```

TypeVar

```python
from typing import TypeVar, Generic

T = TypeVar('T')

class MyClass(Generic[T]):
    def add(self, x: T, y: T) -> T:
        return x + y

def f(obj: MyClass[int]):
    obj.add(1, 1)
```

Generic

- Overloading
- Covariance and contravariance
- Forward reference

History of Python and type hints

2014+ : more and more new type PEPs

2016 -> 2022 , Python 3.5 -> 3.12

PEP 526, PEP 544, PEP 585, PEP 586, PEP 589, PEP 591, PEP 593, PEP 604, PEP 612, PEP 613, PEP 646, PEP 647, PEP 655, PEP 673, PEP 675, PEP 681

16 typing PEPs in 6 years!

Summing up

- **Python is not** a **statically typed** programming language
- … But a **duck typed** one
- **Type hints slowly entered the Python world** (over 10 years of incubation)
- … But Python is **adding type hint** related features at an **exponential rate**

The Type Hint Ecosystem

Ecosystem around type hints

Mypy

- **Static type checking** for Python
- Official Python project
- 10 years old
- 15k ⭐

: my[py]

Ecosystem around type hints

Beartype

- **Runtime type-checker** written in pure Python
- 2 years old
- 1.5k ⭐

Ecosystem around type hints

Pydantic

- **Data validation** using type hints
- 5 years old
- 13.3k

Ecosystem around type hints

Pydantic

```python
from datetime import datetime
from typing import List, Optional
from pydantic import BaseModel

class User(BaseModel):
    id: int
    name = 'John Doe'
    signup_ts: Optional[datetime] = None
    friends: List[int] = []
```

Ecosystem around type hints

Pydantic

```python
from datetime import datetime
from typing import List, Optional
from pydantic import BaseModel

class User(BaseModel):
    id: int
    name = 'John Doe'
    signup_ts: Optional[datetime] = None
    friends: List[int] = []
```

```python
u = User(id='3', name='Johannes')

print(u)
#> id=3 signup_ts=None friends=[]
name='Johannes'
```

Ecosystem around type hints

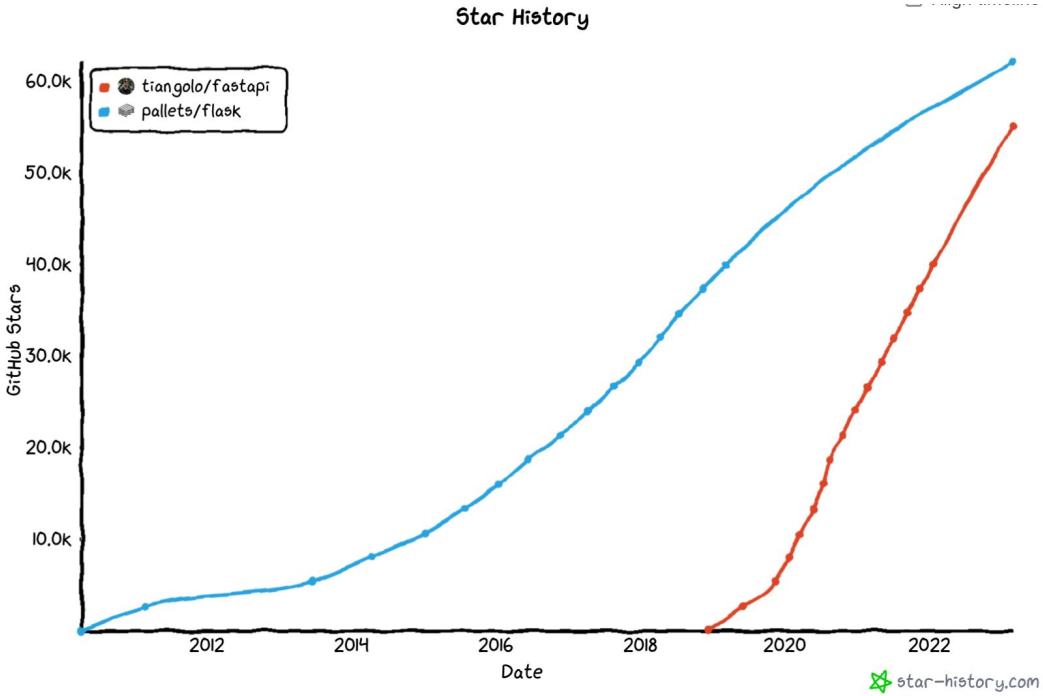
FastAPI

- **Web framework** for building APIs **based on type hints**
- **Based on Pydantic**
- 5 years old
- 55k ⭐

```python
from typing import Union

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    price: float
    is_offer: Union[bool, None] = None

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_name": item.name, "item_id": item_id}

```

Ecosystem around type hints

FastAPI



SQLModel

Ecosystem around type hints

SQLModel

- Interact with **SQL databases** based on type hints
- **Based on Pydantic**
- 10k ⭐

```python
from typing import Optional

from sqlmodel import Field, SQLModel


class Hero(SQLModel, table=True):
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str
    secret_name: str
    age: Optional[int] = None

```

Ecosystem around type hints

Typer

- Build **CLI applications** based on type hints
- Intended to be the FastAPI of CLIs
- 10k ⭐

What about ML frameworks?

Summing up

- Python's **type ecosystem** is **growing**
- … and growing **fast**
- Type based libraries overthrowing long-standing libraries
- ML ecosystem is still missing some type hint based library

DocArray

Type hints for the ML world

From Training to Deployment

```python
def forward(in_1: torch.Tensor, in_2: torch.Tensor) -> torch.Tensor:
  """Compute matrix product of tensors"""
   ...
```

From Training to Deployment

```python
def forward(in_1: torch.Tensor, in_2: torch.Tensor) → torch.Tensor:
  """Compute matrix product of tensors"""
  ...


def forward(
  in_1: TorchTensor[1, 4, 3], in_2: TorchTensor[1, 3, 4]
) → TorchTensor[1, 4, 4]:
  """Compute matrix product of tensors"""
  ...
```

From Training to Deployment

```python
def forward(in_1: torch.Tensor, in_2: torch.Tensor) → torch.Tensor:
  """Compute matrix product of tensors"""
  ...


def forward(
  in_1: TorchTensor[1, 4, 3], in_2: TorchTensor[1, 3, 4]
) → TorchTensor[1, 4, 4]:
  """Compute matrix product of tensors"""
  ...


def forward(
  in_1: TorchTensor['batch', 'n', 'k'], in_2: TorchTensor['batch', 'k', 'm']
) → TorchTensor['batch', 'n', 'm']:
  """Compute matrix product of tensors"""
  ...
```

Type hints for ML

Jaxtyping

- Type annotations and runtime checking for **shape and dtype of Arrays**; (Numpy, Pytorch, Jax)
- 412 ⭐

```python
from jaxtyping import Array, Float, PyTree

def matrix_multiply(x: Float[Array, "X Y"],
                    y: Float[Array, "Y Z"]
                  ) -> Float[Array, "X Z"]:
    return x @ y

```

FastAPI

Pydantic

SQLModel

Type-based

web dev

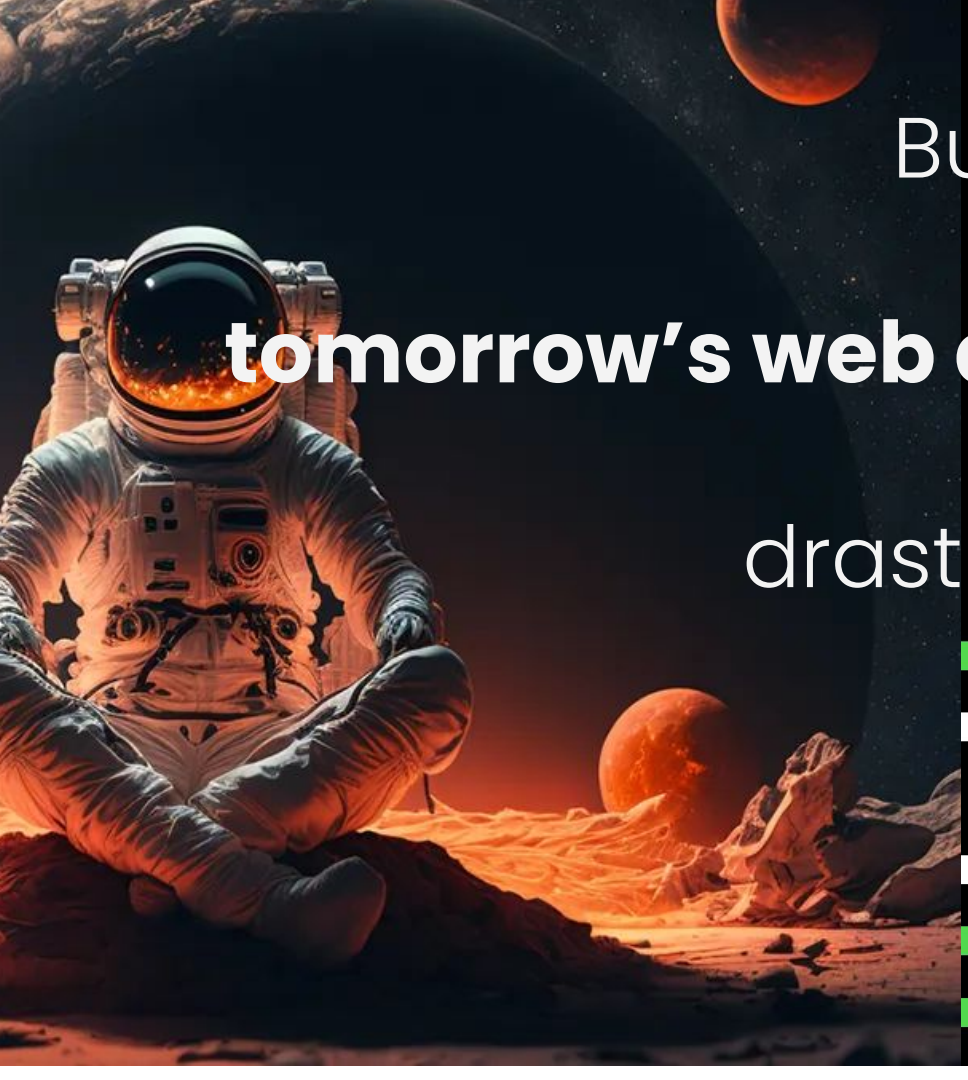
ecosystem

From Training to Deployment

```
class MyImage(BaseModel):
  url: Url
  image_tensor: TorchTensor[224, 224]
  embedding: TorchTensor[768]
```

From Training to Deployment

```
class MyImage(
  url: Url
  image_tensor
  embedding:
```



But

tomorrow's web apps will change

drastically

DocArray

Multimodal AI enters the game

Just in 2022:

- **ChatGPT:** text to text
- **StableDIffusion, Midjourney, DALL-E:** text to image
- **Whisper:** voice to text
- **MusicLM:** test to music

DocArray

Multimodal AI enters the game

Required data types:

- Image
- Audio
- Video
- 3D mesh
- **Tensor**

DocArray

Multimodal AI enters the game

Required data types:

- Image
- Audio
- Video
- 3D mesh
- **Tensor**

But Pydantic's types are :

- Datetime
- Url
- PaymentCardNumber
- ...

From Training to Deployment

```
class MyImage(
  url: Url
  image_tensor
  embedding:
```

From Training to Deployment

```
class MultiModalDoc(BaseModel):
  image_url: Url
  text_url: Url
  mesh_url: Url
  video_url: Url

doc = MultiModalDoc( ... )
```

From Training to Deployment

```
class MultiModalDoc(BaseModel):
  image_url: Url
  text_url: Url
  mesh_url: Url
  video_url: Url

doc = MultiModalDoc( ... )

image_tensor = doc.image_url.???
text_tensor = doc.text_url.???
mesh_tensor = doc.mesh_url.???
video_tensor = doc.video_url.???
```

From Training to Deployment

How do you store your data?

How to serve your app?

From Training to Deployment

How do you store your data?

SQLModel

How to serve your app?

FastAPI

From Training to Deployment

How do you store your data?

SQLModel ?

How to serve your app?

FastAPI ?

DocArray

Enter: DocArray

docarray

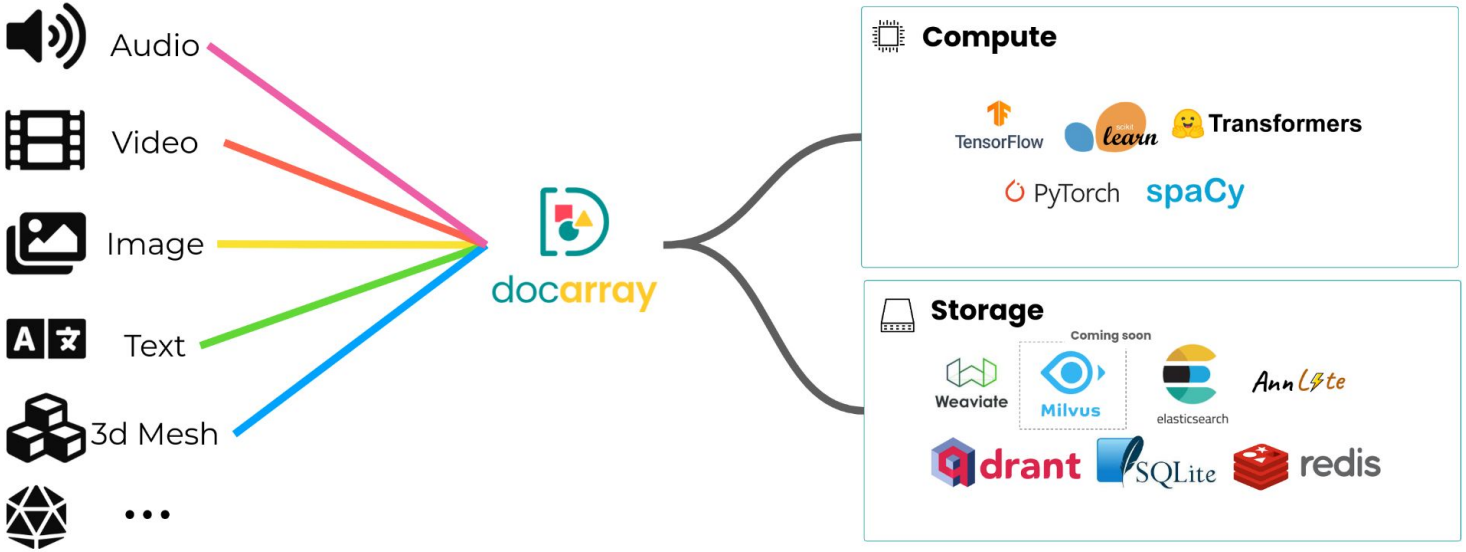
1. Represent
2. Send
3. Store

Multi-modal data

DocArray

A Multimodal ML library

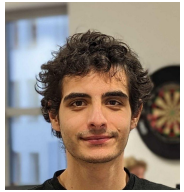
- Provide a common interface for different modalities



DocArray

Enter: DocArray

- **Part of Linux Foundation AI and Data**
- **6 main maintainers + ~50 community contributors**


Sami

Charlotte

Anne

Jackmin

Joan

Johannes (me)

DocArray

Represent:
Pydantic with a hint of ML

Live coding time! 🤞

DocArray

Send:
FastAPI and HTTP

Live coding time! 🤞

DocArray

Store:
ORM for Vector Databases

Live coding time! 🤞

DocArray

Store:
ORM for Vector Databases

weaviate

drant

redis

elasticsearch

HNSWLib

Milvus

DocArray

Store:
ORM for Vector Databases

weaviate

drant

redis

elasticsearch

HNSWLib

Milvus

Live coding time! ✌️

DocArray

Jina is building the ML-typed ecosystem

3 years ago	Type hints	Type hints for AI apps
Manual validation	Pydantic	**DocArray**
Flask	FastAPI	**Jina**
SQLAlchemy	SQLModel	**DocArray**

Summing up

- Type hints are entering the Python codebase
- The type hint ecosystem is growing fast and eating the Python world
- DocArray embraces this wave of types and apply it to the ML/AI world

Thank you !

Follow me (if you want):

@atomicfIndr

Give a star to DocArray:

https://github.com/**docarray**/**docarray**

Join DocArray Discord:

https://discord.com/invite/**WaMp6PVPgR**