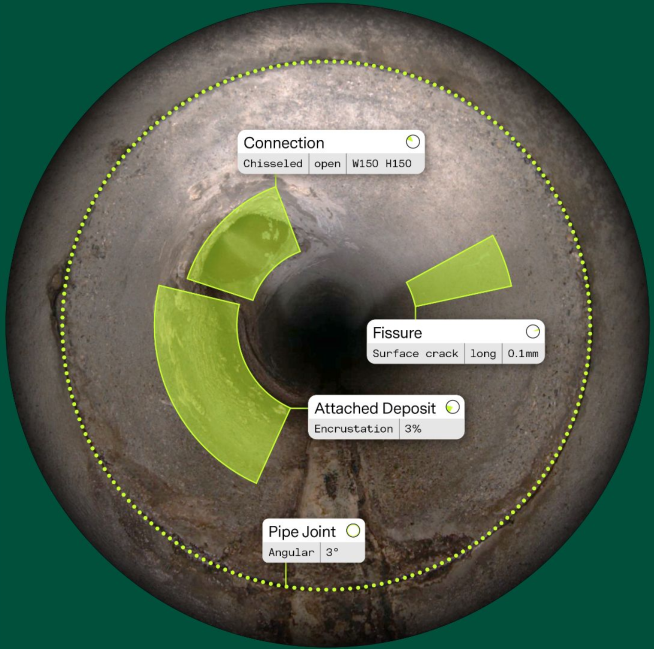
Eric Wolf
Co-Founder at Pallon
eric.wolf@pallon.com

```
def squeeze(
  input: Tensor[DType, Shape],
  dim: Dim)
-> Tensor[DType, 🤔]
```

Can We Squeeze( ) More Out of Python’s Type System?

The Challenge of Tensor Shape Annotations

Machine Learning for the Underworld

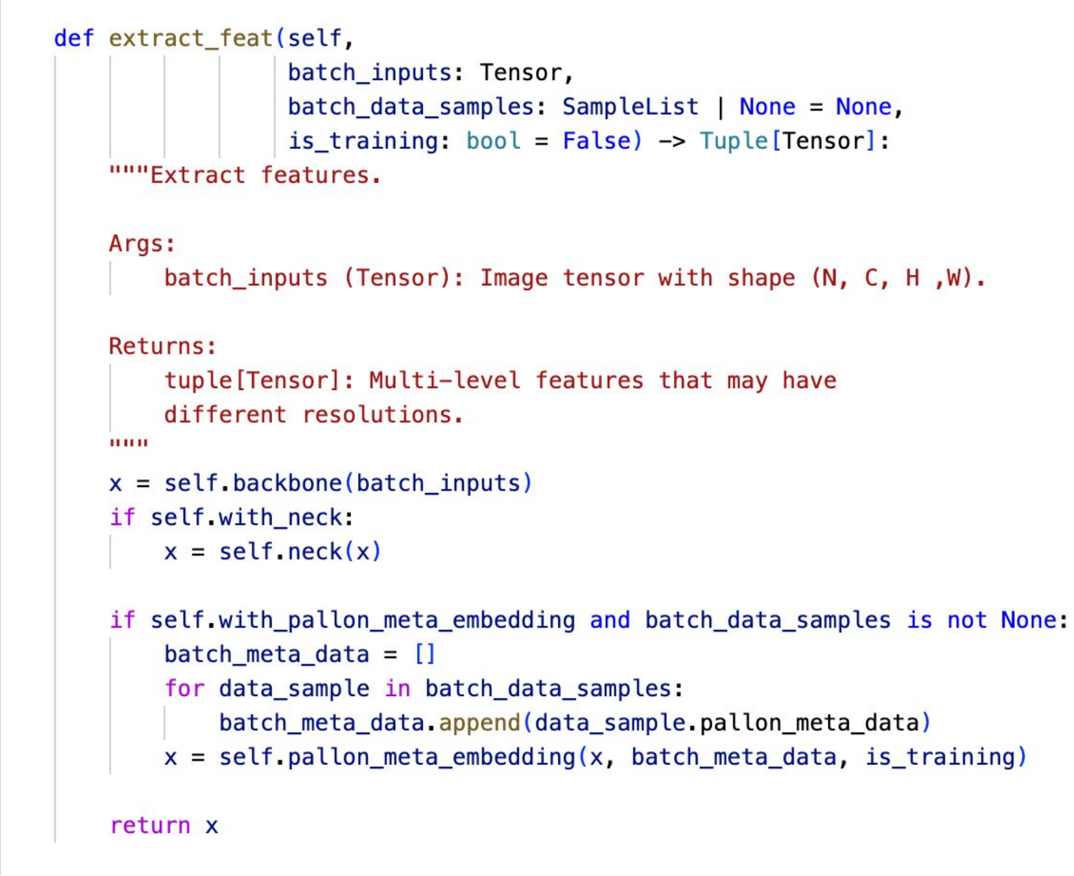


Eric Wolf
Co-Founder at Pallon
eric.wolf@pallon.com

```
def squeeze(
  input: Tensor[DType, Shape],
  dim: Dim)
-> Tensor[DType, 🤔]
```

Can We Squeeze( ) More Out of Python’s Type System?

The Challenge of Tensor Shape Annotations

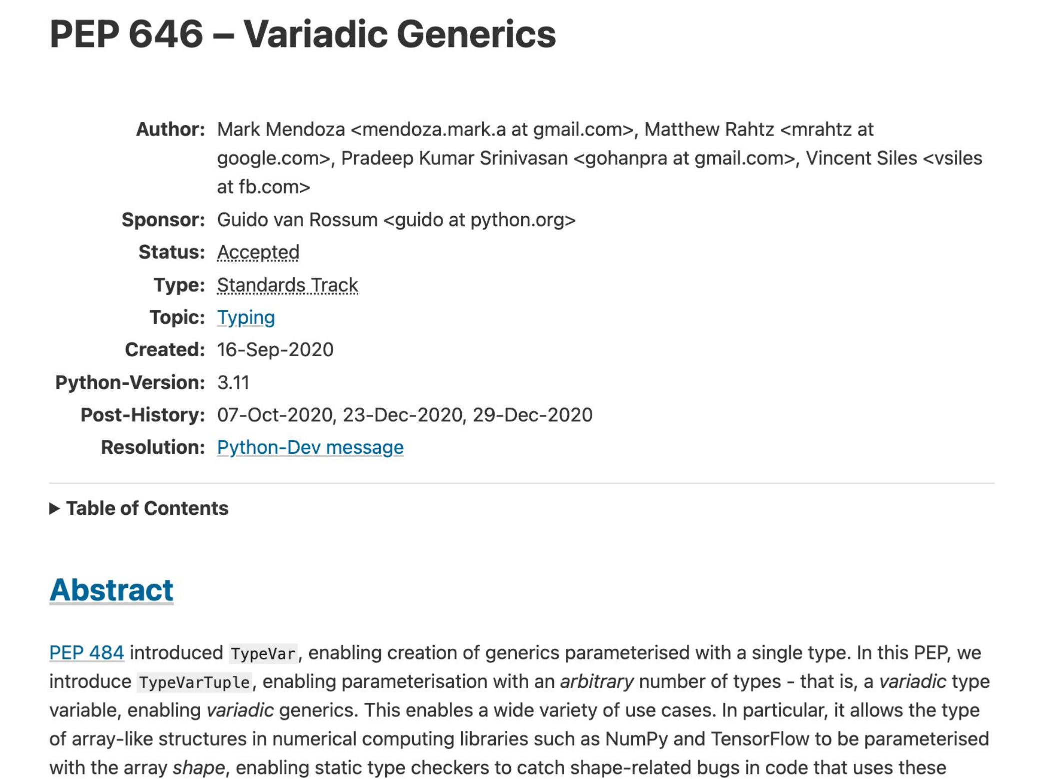


```python
def extract_feat(self,
                 batch_inputs: Tensor,
                 batch_data_samples: SampleList | None = None,
                 is_training: bool = False) -> Tuple[Tensor]:
    """Extract features.

    Args:
        batch_inputs (Tensor): Image tensor with shape (N, C, H ,W).

    Returns:
        tuple[Tensor]: Multi-level features that may have
        different resolutions.
    """
    x = self.backbone(batch_inputs)
    if self.with_neck:
        x = self.neck(x)

    if self.with_pallon_meta_embedding and batch_data_samples is not None:
        batch_meta_data = []
        for data_sample in batch_data_samples:
            batch_meta_data.append(data_sample.pallon_meta_data)
        x = self.pallon_meta_embedding(x, batch_meta_data, is_training)

    return x
```

Source: MMDetection

Let’s try…

Good News 🙌



PEP 646 – Variadic Generics

Author: Mark Mendoza <mendoza.mark.a at gmail.com>, Matthew Rahtz <mrahtz at google.com>, Pradeep Kumar Srinivasan <gohanpra at gmail.com>, Vincent Siles <vsiles at fb.com>
Sponsor: Guido van Rossum <guido at python.org>
Status: Accepted
Type: Standards Track
Topic: Typing
Created: 16-Sep-2020
Python-Version: 3.11
Post-History: 07-Oct-2020, 23-Dec-2020, 29-Dec-2020
Resolution: Python-Dev message

▶ **Table of Contents**

Abstract

PEP 484 introduced `TypeVar`, enabling creation of generics parameterised with a single type. In this PEP, we introduce `TypeVarTuple`, enabling parameterisation with an *arbitrary* number of types - that is, a *variadic* type variable, enabling *variadic* generics. This enables a wide variety of use cases. In particular, it allows the type of array-like structures in numerical computing libraries such as NumPy and TensorFlow to be parameterised with the array *shape*, enabling static type checkers to catch shape-related bugs in code that uses these

How would you type this?

```
x: Array[Batch, Height, Width, 3]

x + [5.2, -1.1, 4.7]          # Array[Batch, Height, Width, 3]

x.transpose((0, 3, 1, 2))     # Array[Batch, 3, Height, Width]

x.reshape((x.shape[0], -1))   # Array[Batch, 3 * Height * Width]

conv2d(
  x,
  out_channels=128,
  kernel_size=3
)                             # Array[Batch, Height - 1, Width - 1, 128]
```

What are we waiting for?

For basic support

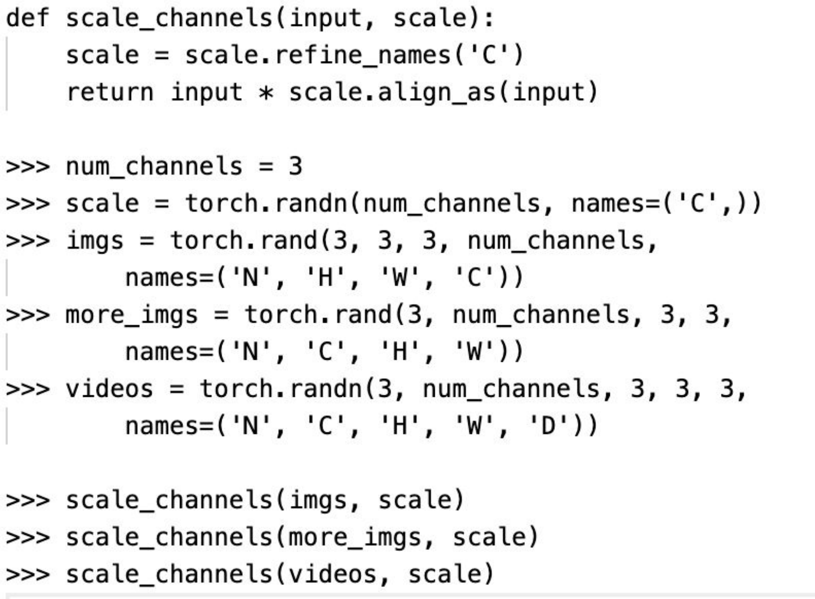
- Mypy support for PEP-646
- Numpy type hints
- Pytorch, TensorFlow, JAX, Scipy, etc. type hints

For maximum typing magic 🧙‍♀️

- Python syntax: unpack multiple type var tuples, slice type var tuples, literal arithmetic, …
- Mypy/Pyright/… support

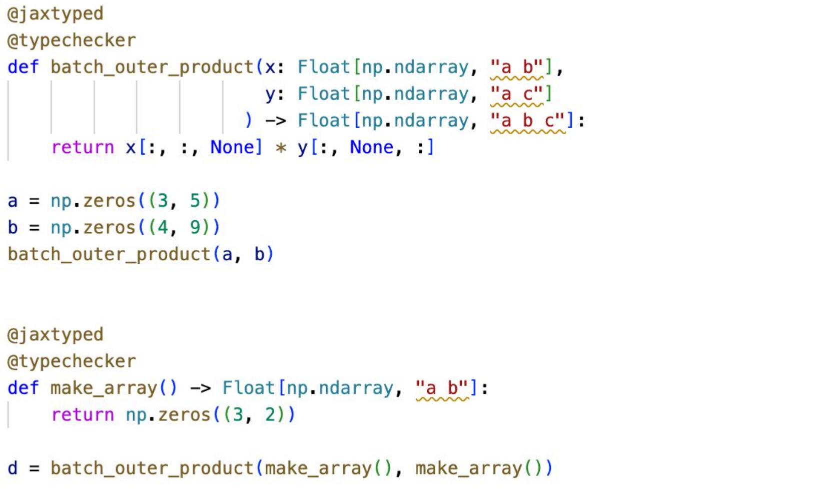
What can we do today? Runtime checks!

Pytorch Named Tensors (prototype)



```
def scale_channels(input, scale):
    scale = scale.refine_names('C')
    return input * scale.align_as(input)

>>> num_channels = 3
>>> scale = torch.randn(num_channels, names=('C',))
>>> imgs = torch.rand(3, 3, 3, num_channels,
        names=('N', 'H', 'W', 'C'))
>>> more_imgs = torch.rand(3, num_channels, 3, 3,
        names=('N', 'C', 'H', 'W'))
>>> videos = torch.randn(3, num_channels, 3, 3, 3,
        names=('N', 'C', 'H', 'W', 'D'))

>>> scale_channels(imgs, scale)
>>> scale_channels(more_imgs, scale)
>>> scale_channels(videos, scale)
```

Jaxtyping



```
@jaxtyped
@typechecker
def batch_outer_product(x: Float[np.ndarray, "a b"],
                        y: Float[np.ndarray, "a c"]
                       ) -> Float[np.ndarray, "a b c"]:
    return x[:, :, None] * y[:, None, :]

a = np.zeros((3, 5))
b = np.zeros((4, 9))
batch_outer_product(a, b)


@jaxtyped
@typechecker
def make_array() -> Float[np.ndarray, "a b"]:
    return np.zeros((3, 2))

d = batch_outer_product(make_array(), make_array())
```

Pallon

In conclusion…

- Can we use shape types today? Not yet 😭
- But Python is getting there, sloooowly
- You can use runtime checks–but they are not as cool

Thank you

Talk to me about type hints and sewers :)

Eric Wolf
Co-Founder at Pallon
eric.wolf@pallon.com