image recognition with jax

brettkoonce.com/talks

february 25, 2023

outline

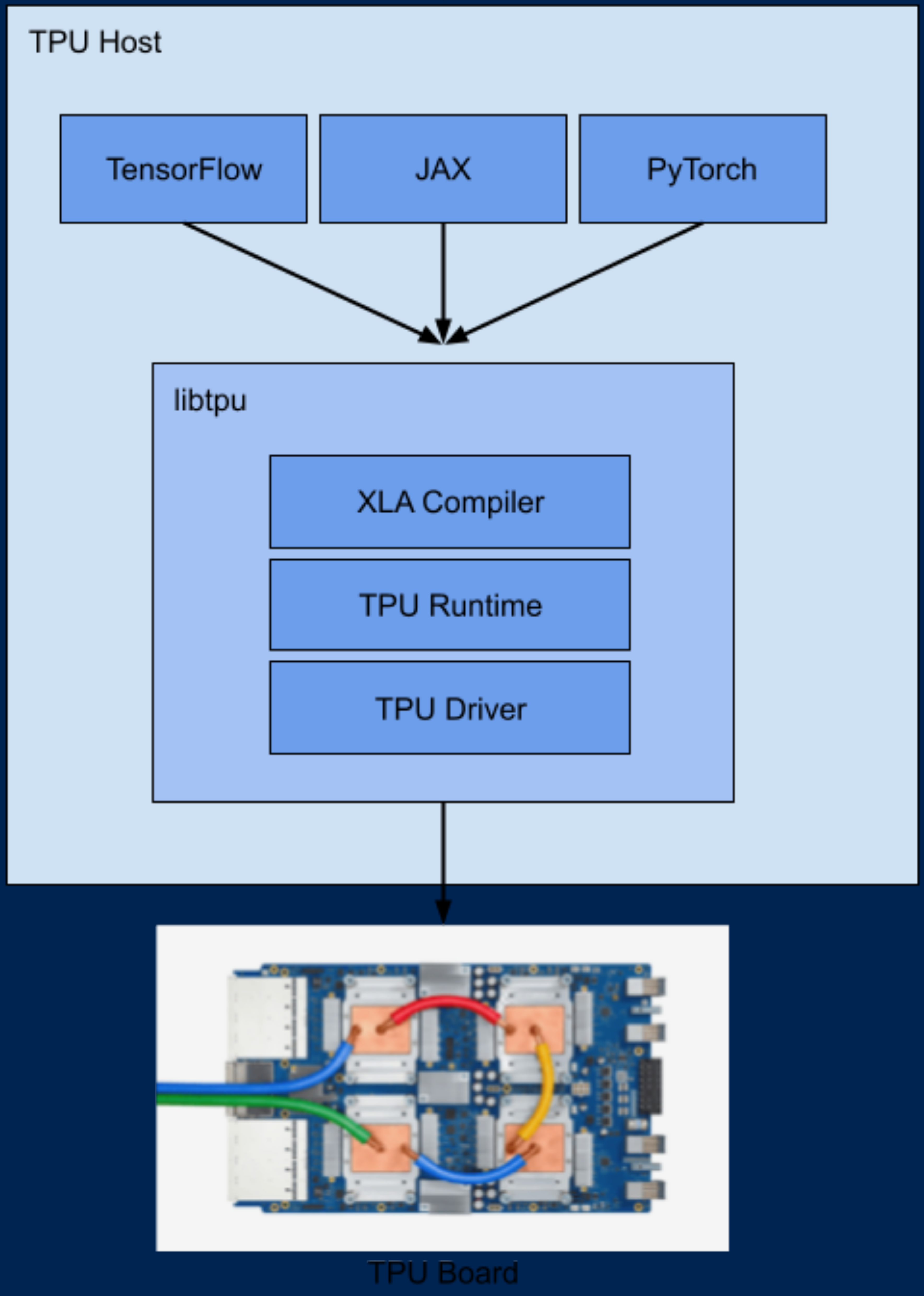
- image recognition, jax
- mnist, cifar: 2d cnn
- imagenet: resnet 50
- transformers --> vit
- scaling using jax, future

image recognition

- cats vs. dogs --> binary classification
- mnist, cifar, imagenet
- retraining, foundational models
- object detection/semantic segmentation
- other fields (stable diffusion, audio, RL, ...)

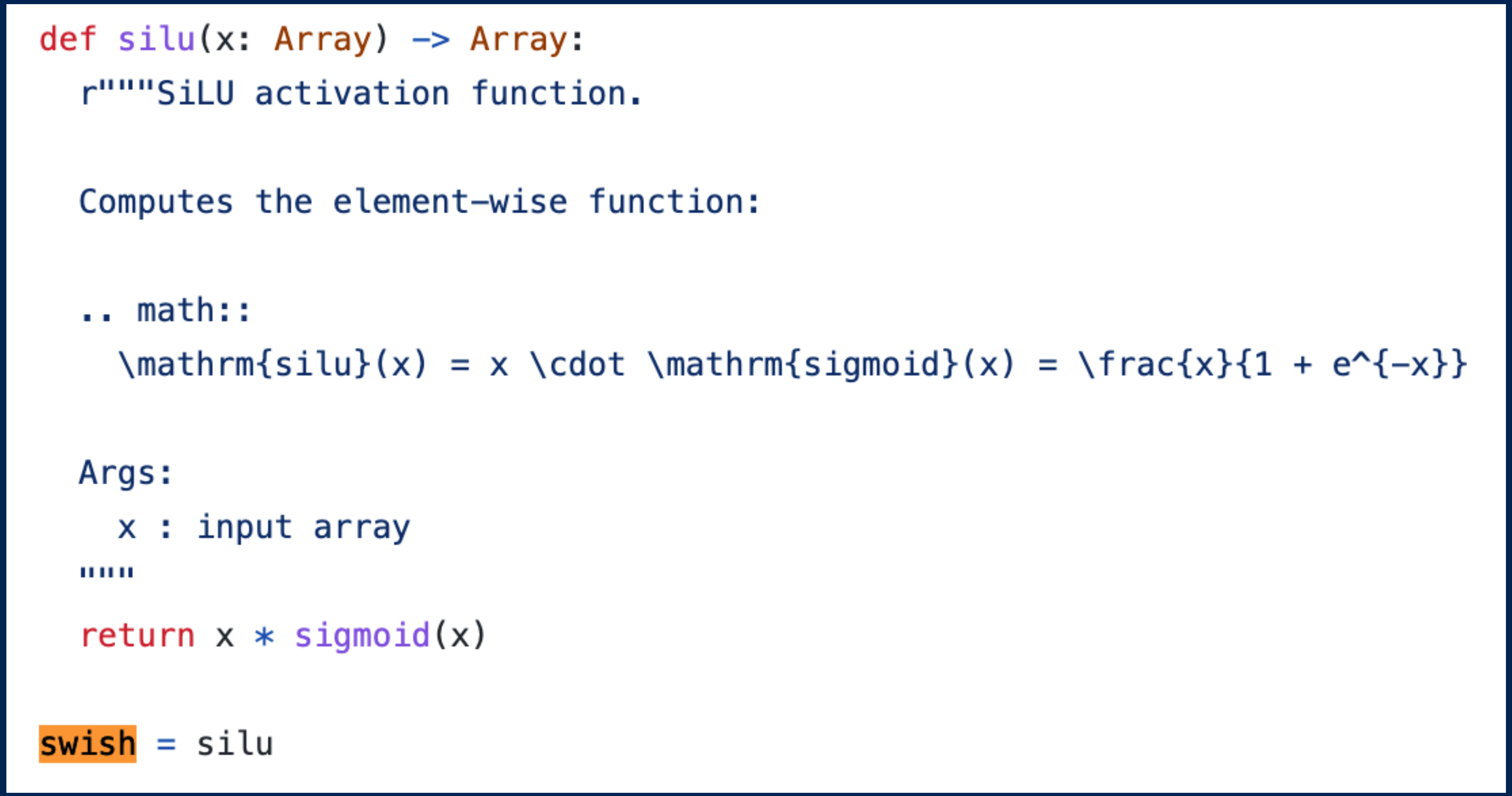
jax

- numpy api --> xla converter
- ecosystem - deepmind/google
- session demo



autodiff

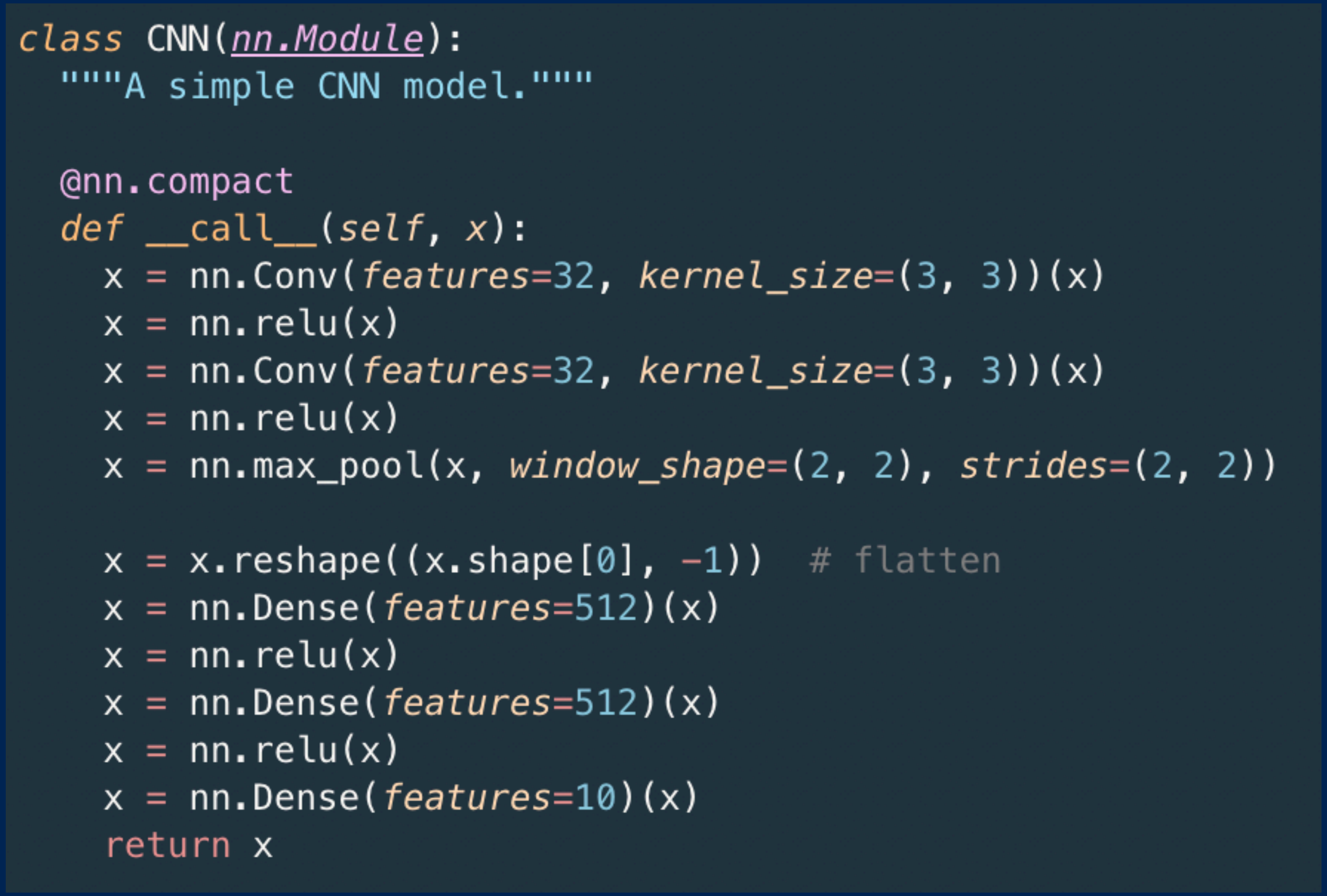
- pmap + vmap
- p•j•v
- custom gradients
- roger grosse



```python
def silu(x: Array) -> Array:
  r"""SiLU activation function.

  Computes the element-wise function:

  .. math::
    \mathrm{silu}(x) = x \cdot \mathrm{sigmoid}(x) = \frac{x}{1 + e^{-x}}

  Args:
    x : input array
  """
  return x * sigmoid(x)

swish = silu
```

2d cnn

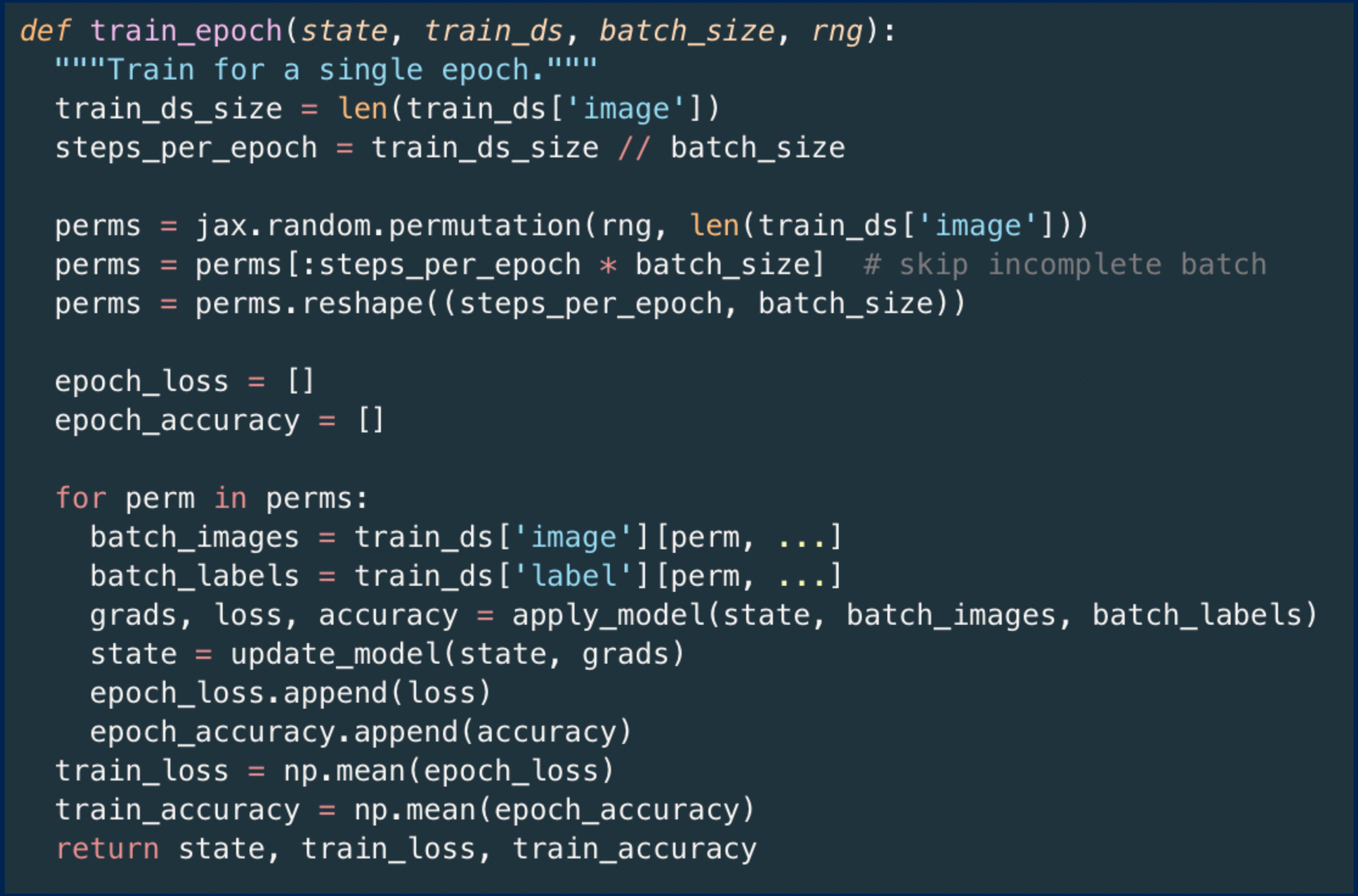
- mlp
- cnn
- jax model
- [3x3] * 2 + mnist



```python
class CNN(nn.Module):
  """A simple CNN model."""

  @nn.compact
  def __call__(self, x):
    x = nn.Conv(features=32, kernel_size=(3, 3))(x)
    x = nn.relu(x)
    x = nn.Conv(features=32, kernel_size=(3, 3))(x)
    x = nn.relu(x)
    x = nn.max_pool(x, window_shape=(2, 2), strides=(2, 2))

    x = x.reshape((x.shape[0], -1))  # flatten
    x = nn.Dense(features=512)(x)
    x = nn.relu(x)
    x = nn.Dense(features=512)(x)
    x = nn.relu(x)
    x = nn.Dense(features=10)(x)
    return x
```

mnist demo (fp32)

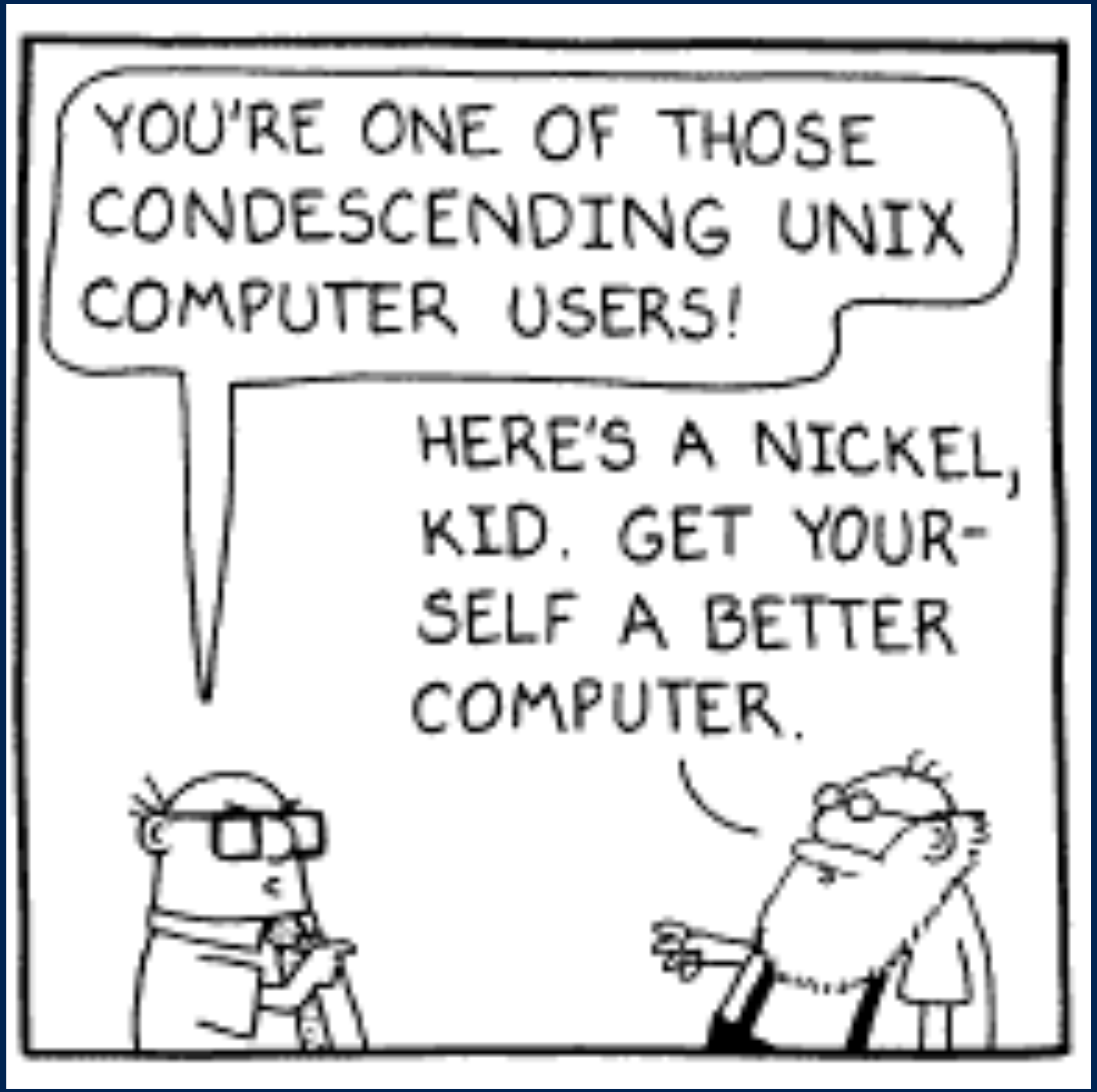
- jax hello world
- run demo



```python
def train_epoch(state, train_ds, batch_size, rng):
  """Train for a single epoch."""
  train_ds_size = len(train_ds['image'])
  steps_per_epoch = train_ds_size // batch_size

  perms = jax.random.permutation(rng, len(train_ds['image']))
  perms = perms[:steps_per_epoch * batch_size]  # skip incomplete batch
  perms = perms.reshape((steps_per_epoch, batch_size))

  epoch_loss = []
  epoch_accuracy = []

  for perm in perms:
    batch_images = train_ds['image'][perm, ...]
    batch_labels = train_ds['label'][perm, ...]
    grads, loss, accuracy = apply_model(state, batch_images, batch_labels)
    state = update_model(state, grads)
    epoch_loss.append(loss)
    epoch_accuracy.append(accuracy)
  train_loss = np.mean(epoch_loss)
  train_accuracy = np.mean(epoch_accuracy)
  return state, train_loss, train_accuracy
```

ssh tpu

- tmux -->
- htop
- free -h


YOU'RE ONE OF THOSE CONDESCENDING UNIX COMPUTER USERS!
HERE'S A NICKEL, KID. GET YOURSELF A BETTER COMPUTER.

imagenet demo

- scenic + resnet50

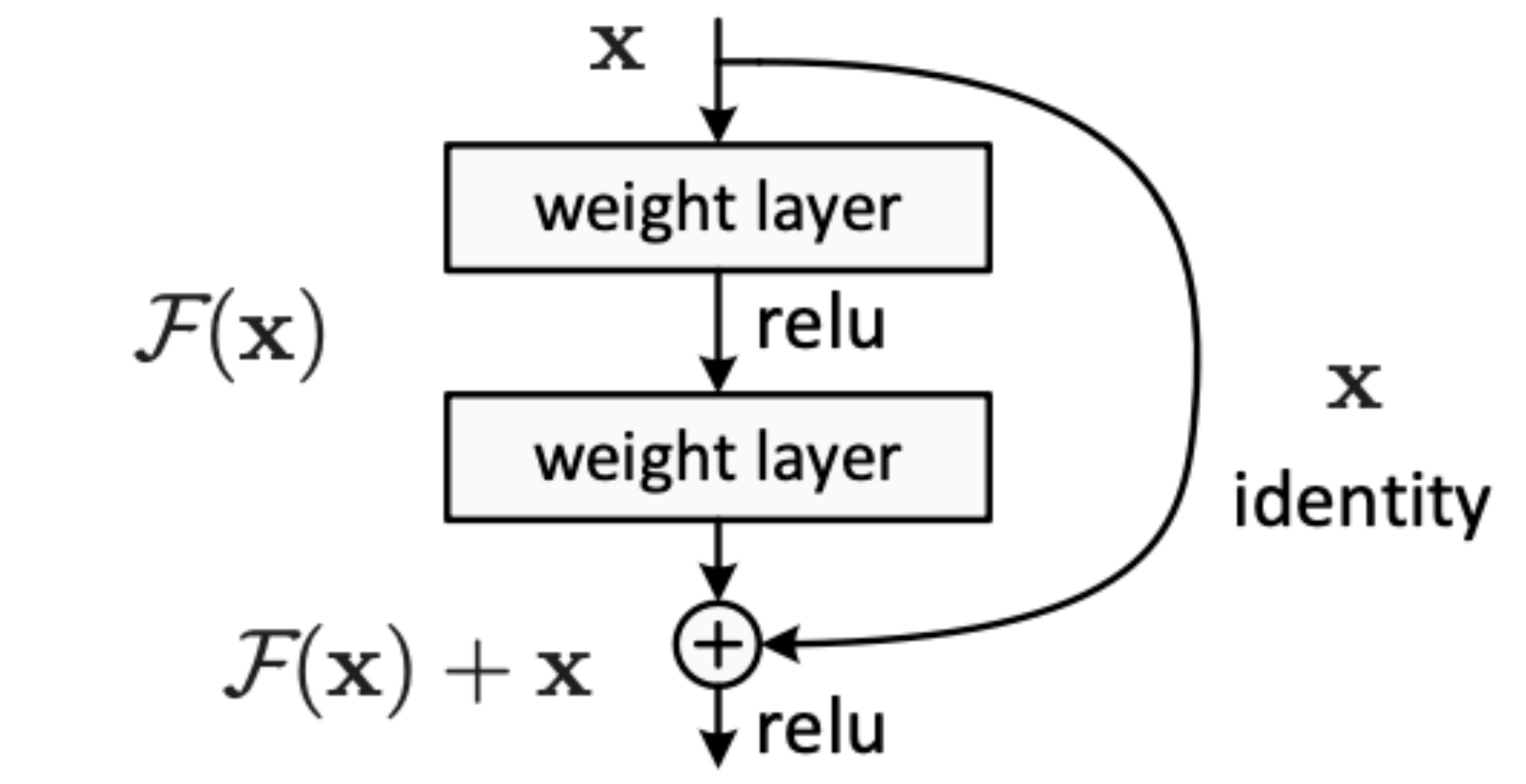
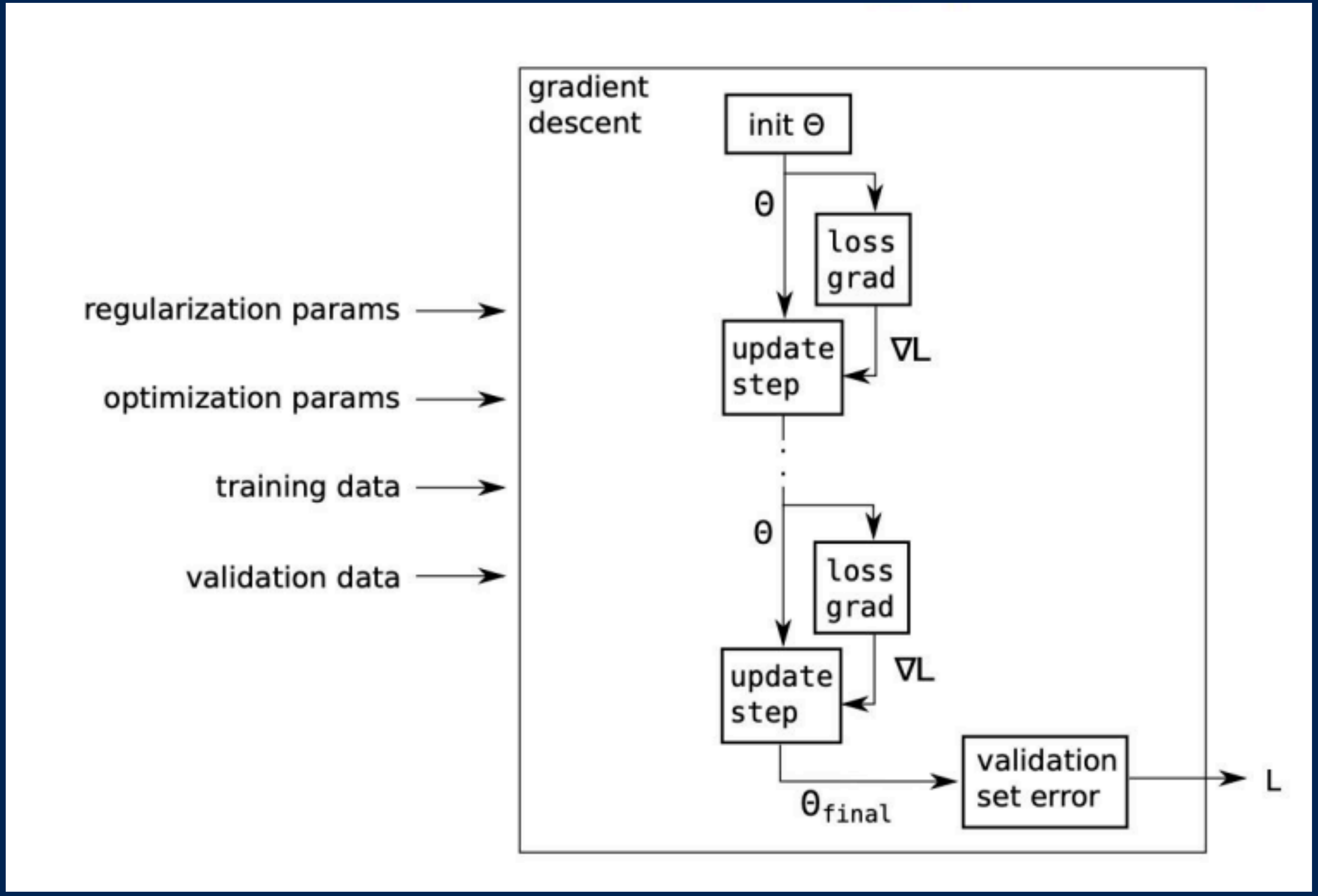


Figure 2. Residual learning: a building block.

(*) parallel

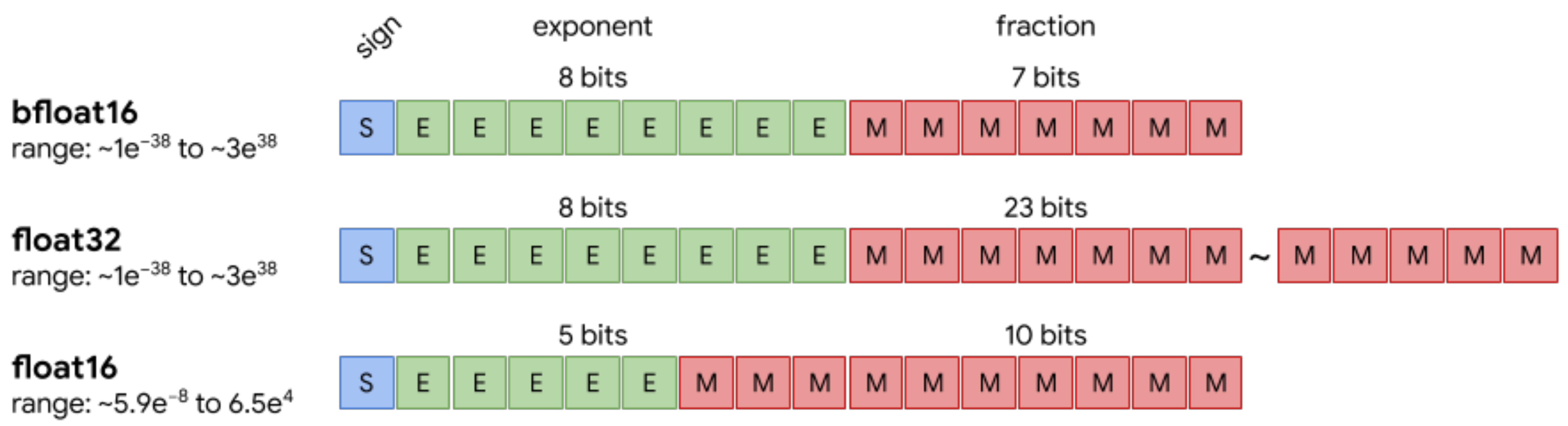
- data (resnet)
- model (transformers)
- pipeline (deepspeed)



fast.ai + dawnbench

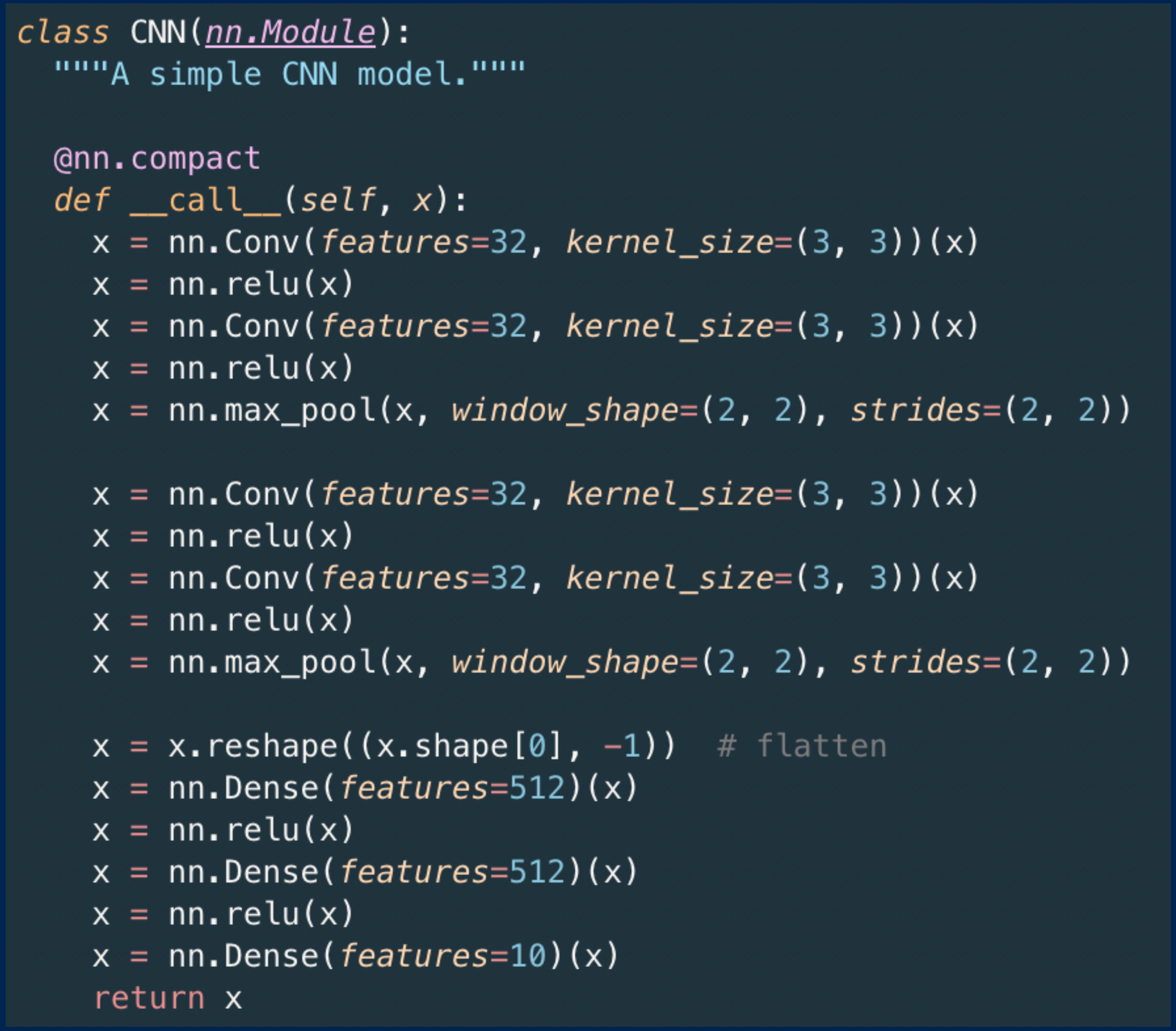
- fp16 (--> bf16 today)
- cosine annealing
- progressive resizing
- tpu results
- resnet34, resnet18 --> page: resnet9

lower precision



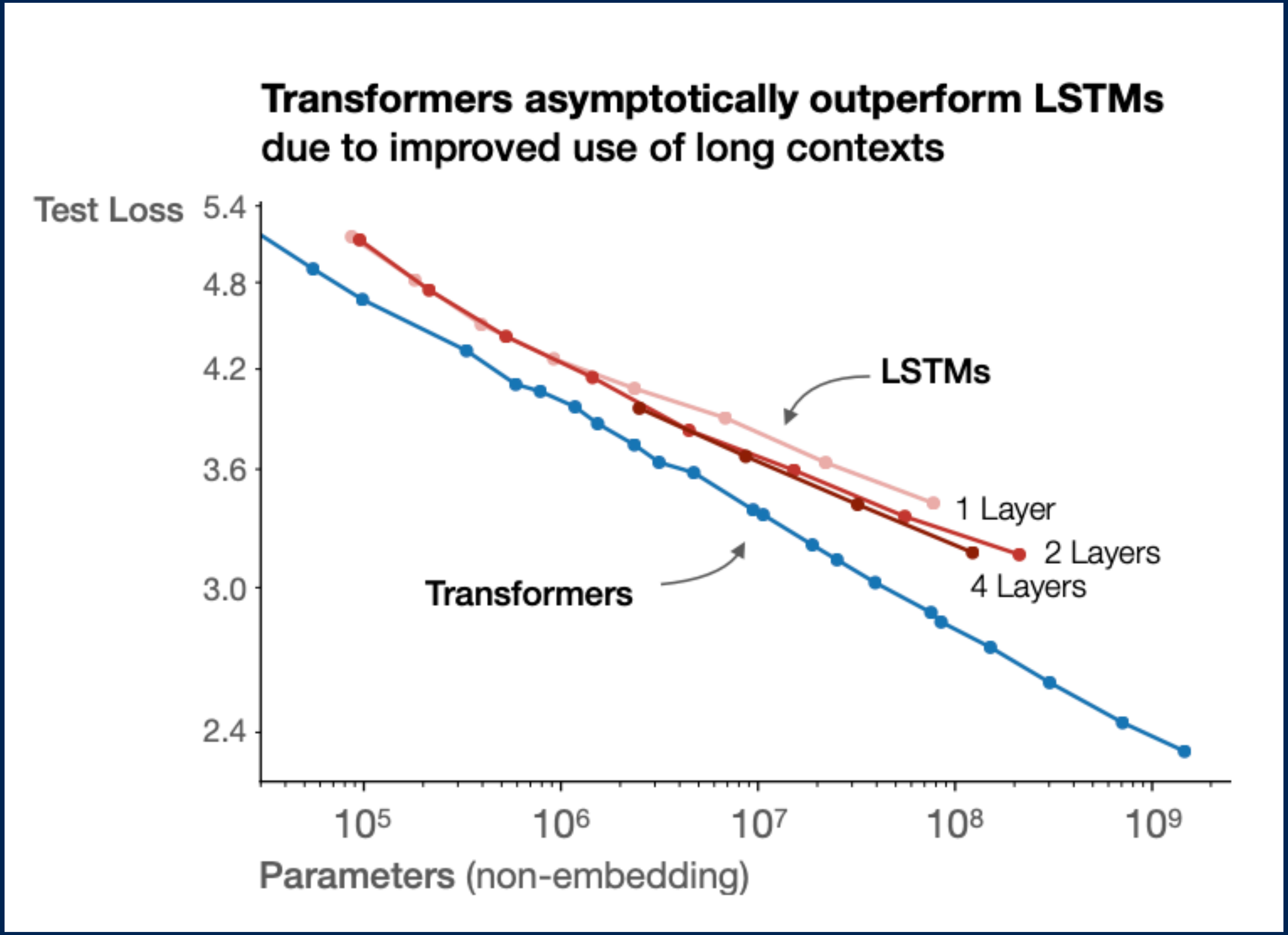
cifar demo (bf16)

- **jax hello world**
- **run demo**

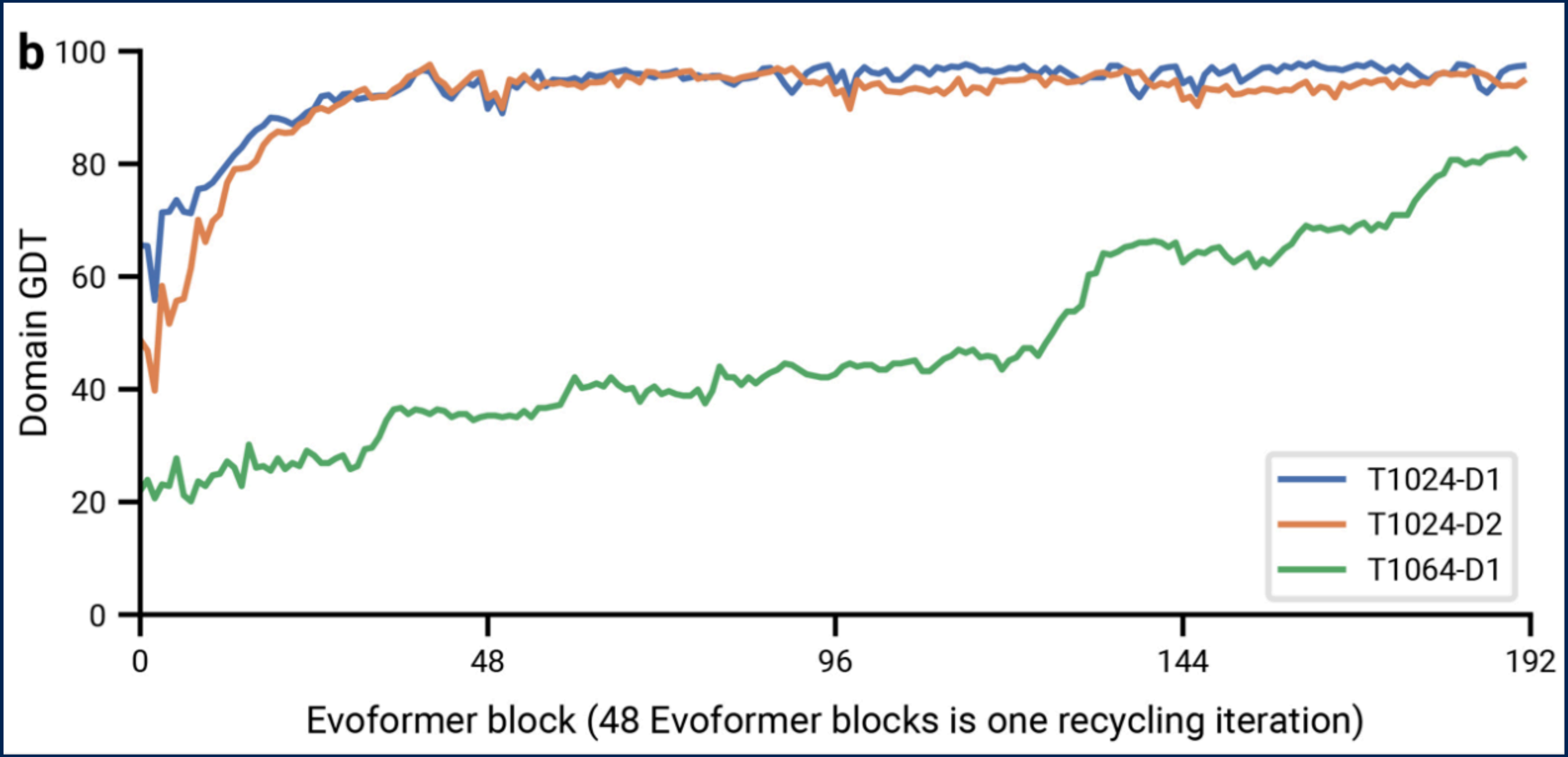


```python
class CNN(nn.Module):
  """A simple CNN model."""

  @nn.compact
  def __call__(self, x):
    x = nn.Conv(features=32, kernel_size=(3, 3))(x)
    x = nn.relu(x)
    x = nn.Conv(features=32, kernel_size=(3, 3))(x)
    x = nn.relu(x)
    x = nn.max_pool(x, window_shape=(2, 2), strides=(2, 2))

    x = nn.Conv(features=32, kernel_size=(3, 3))(x)
    x = nn.relu(x)
    x = nn.Conv(features=32, kernel_size=(3, 3))(x)
    x = nn.relu(x)
    x = nn.max_pool(x, window_shape=(2, 2), strides=(2, 2))

    x = x.reshape((x.shape[0], -1))  # flatten
    x = nn.Dense(features=512)(x)
    x = nn.relu(x)
    x = nn.Dense(features=512)(x)
    x = nn.relu(x)
    x = nn.Dense(features=10)(x)
    return x
```

transformers



alphafold 2


b
100
80
60
40
20
0
Domain GDT
0
48
96
144
192
Evoformer block (48 Evoformer blocks is one recycling iteration)
T1024-D1
T1024-D2
T1064-D1

vit

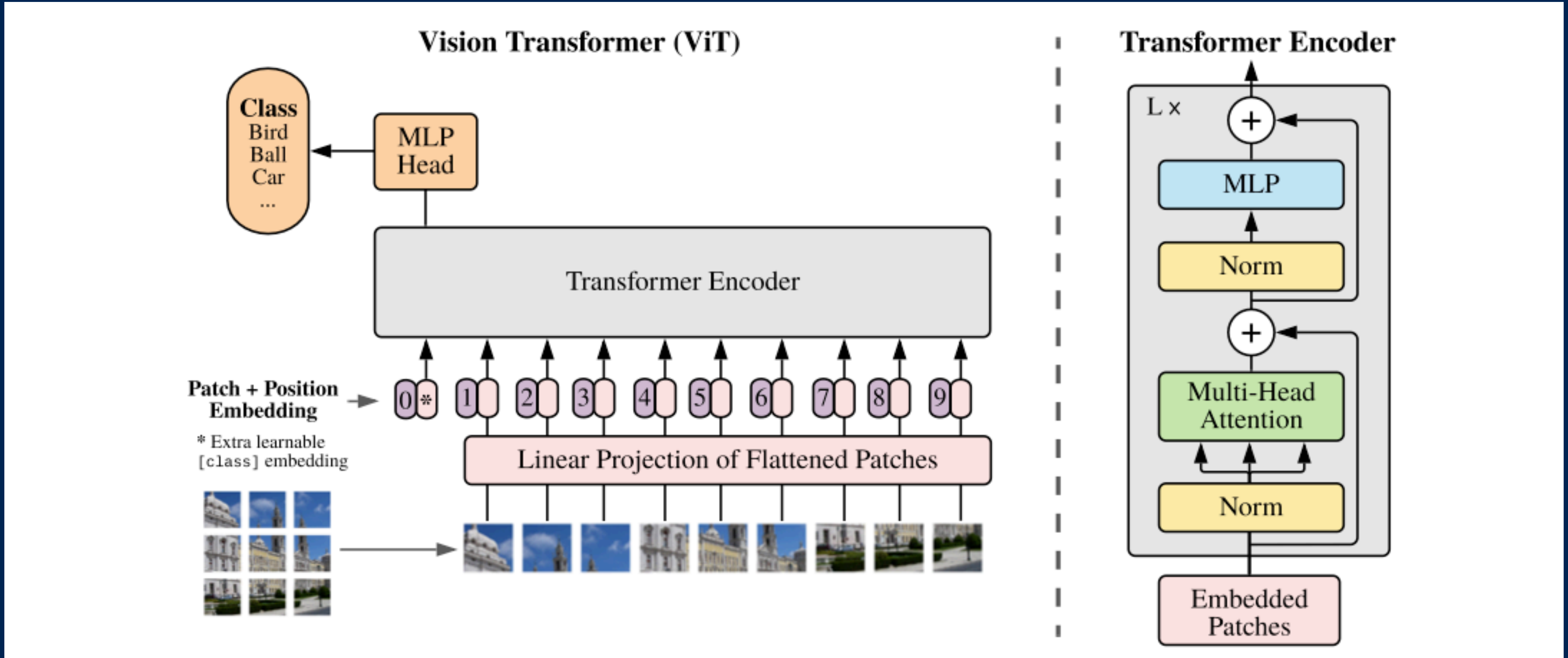


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

vit + scenic

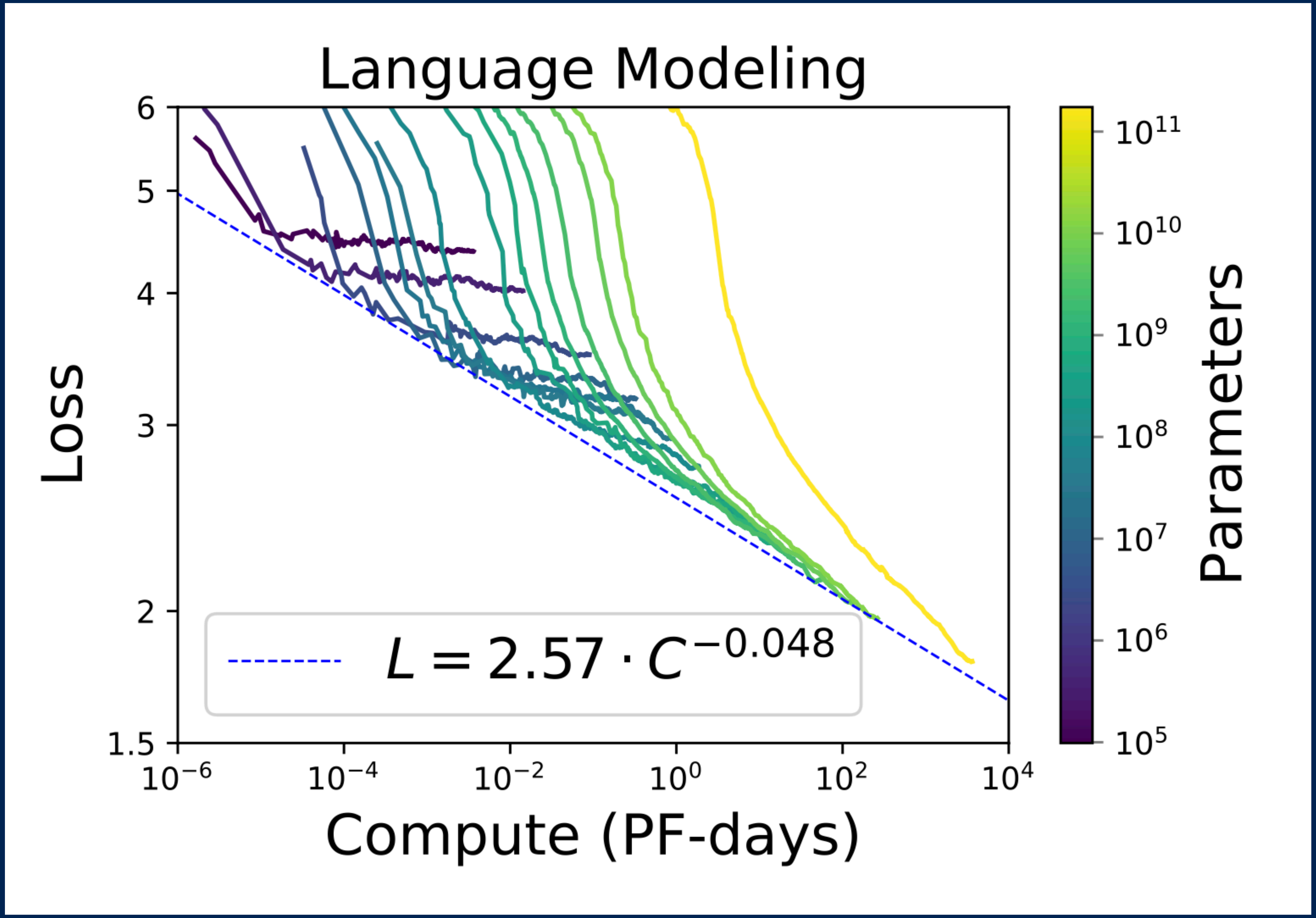
- start demo:
  - 4x gpu (local)
  - tpu-v3-8
  - tpu-v3-32 pod (512gb)

recap

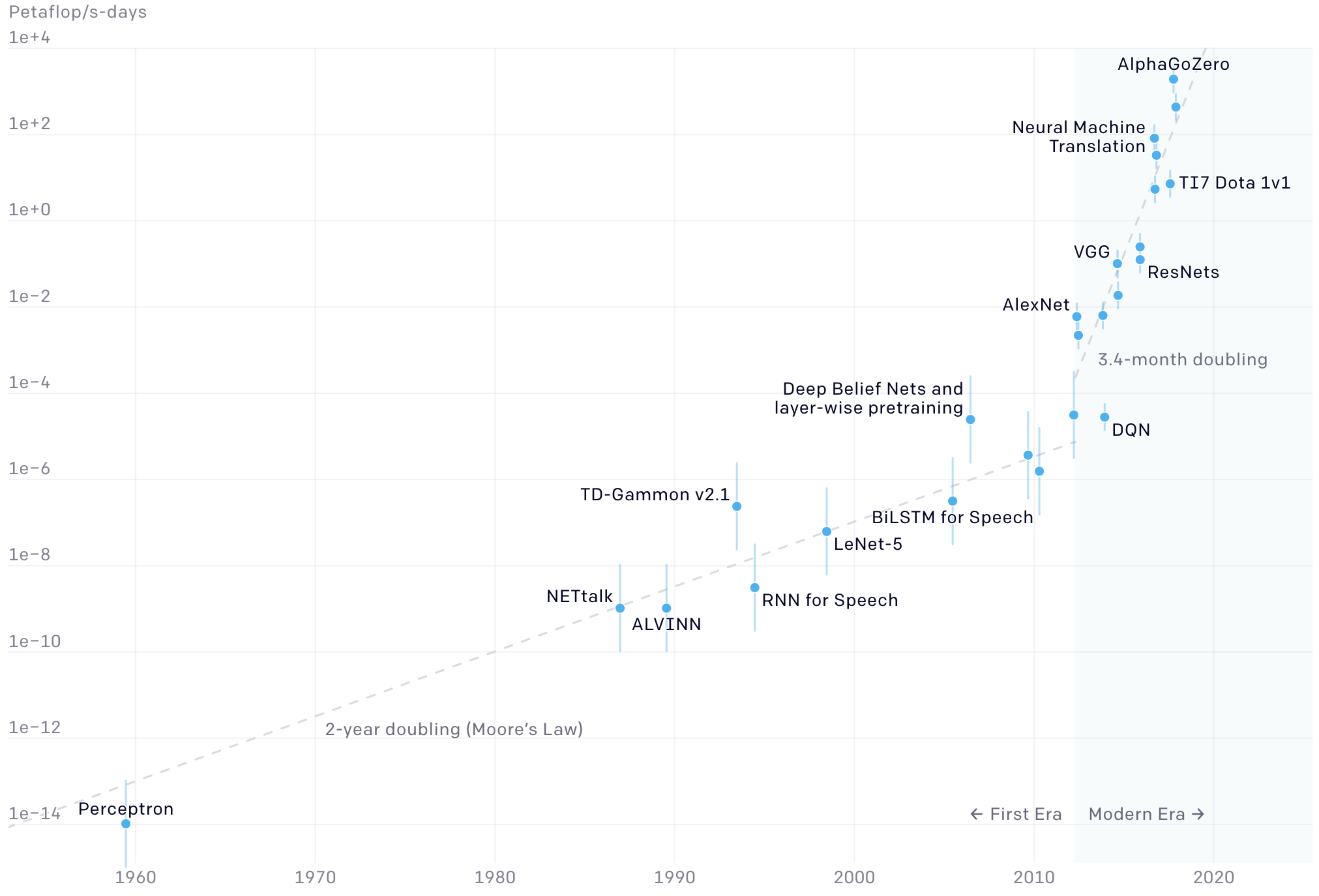
- built simple cnn using flax
- trained resnet50 network
- trained vit network
- used jax + tpu to scale compute

scaling hypothesis

- **simple model + large data >>> complexity**



Two Distinct Eras of Compute Usage in Training AI Systems



trc

- thank you to trc program for the tpu time
- can verify experiments that would take weeks in a few hours, serial vs parallel
- jax general availability, ecosystem
- thank jax team for feedback

next steps

- imagerecognitionwithjax.com
- eleuther.ai
- gwern: bitter lesson
- fast.ai
- pytorch 2