```
# pyproject.toml.jinja

[project]
name = "hw03"
version = "0.1.0"
description = "MNIST CNN"
readme = "README.md"
authors = [
    { name = "Wongee (Freddy) Hong", email = "wongee.hong@cooper.edu" }
]
requires-python = ">=3.11"
dependencies = [
    "structlog",
    "numpy",
    "tensorflow",
    "tensorflow_datasets",
    "pydantic-settings",
    "matplotlib",
    "tqdm",
    "jax",
    "jaxlib",
    "flax",
    "optax",
    "scikit-learn",
]

[project.scripts]
hw03 = "hw03:main"

[build-system]
requires = ["uv_build>=0.8.3,<0.9.0"]
build-backend = "uv_build"
```

```python
import logging
import os
import sys
from pathlib import Path

import jax
import numpy as np
import structlog


class FormattedFloat(float):
    def __repr__(self) -> str:
        return f"{self:.4g}"


def custom_serializer_processor(logger, method_name, event_dict):
    for key, value in event_dict.items():
        if hasattr(value, "numpy"):
            value = value.numpy()
        if isinstance(value, jax.Array):
            value = np.array(value)
        if isinstance(value, (np.generic, np.ndarray)):
            value = value.item() if value.size == 1 else value.tolist()
        if isinstance(value, float):
            value = FormattedFloat(value)
        if isinstance(value, Path):
            value = str(value)
        event_dict[key] = value
    return event_dict

def configure_logging(log_dir: Path = Path("artifacts"), log_name: str = "train.json"):
    log_dir.mkdir(parents=True, exist_ok=True)
    log_file = log_dir / log_name
    log_level = os.environ.get("LOG_LEVEL", "INFO").upper()

    shared_processors = [
        structlog.stdlib.add_logger_name,
        structlog.stdlib.add_log_level,
        structlog.stdlib.PositionalArgumentsFormatter(),
        structlog.processors.TimeStamper(fmt="iso"),
        structlog.processors.StackInfoRenderer(),
        structlog.processors.format_exc_info,
        structlog.processors.UnicodeDecoder(),
        custom_serializer_processor,
    ]


    structlog.configure(
        processors=shared_processors + [

            structlog.stdlib.ProcessorFormatter.wrap_for_formatter,
        ],
        logger_factory=structlog.stdlib.LoggerFactory(),
        wrapper_class=structlog.stdlib.BoundLogger,
        cache_logger_on_first_use=True,
    )

    console_renderer = structlog.dev.ConsoleRenderer(
        colors=sys.stdout.isatty(),
        exception_formatter=structlog.dev.RichTracebackFormatter(),
    )
    console_handler = logging.StreamHandler(sys.stdout)
    console_handler.setFormatter(
        structlog.stdlib.ProcessorFormatter(

            processors=[
                structlog.stdlib.ProcessorFormatter.remove_processors_meta,
                console_renderer
            ],
```

```python
        )
    )

    file_renderer = structlog.processors.JSONRenderer()
    file_handler = logging.FileHandler(log_file, mode="w")
    file_handler.setFormatter(
        structlog.stdlib.ProcessorFormatter(
            processors=[
                structlog.stdlib.ProcessorFormatter.remove_processors_meta,
                file_renderer
            ],
        )
    )

    root_logger = logging.getLogger()
    root_logger.addHandler(console_handler)
    root_logger.addHandler(file_handler)
    root_logger.setLevel(log_level)

    logging.getLogger("matplotlib").setLevel(logging.WARNING)
    logging.getLogger("PIL").setLevel(logging.WARNING)

    return log_file
```

```python
from pathlib import Path
from typing import Tuple
from pydantic import BaseModel
from pydantic_settings import BaseSettings


class ModelSettings(BaseModel):
    input_depth: int = 1
    layer_depths: list[int] = [16,32]
    blocks_per_stage: list[int] = [1,1]
    num_classes: int = 10
    dropout: float = 0.2


class DataSettings(BaseModel):
    batch_size: int = 32
    validation_size: int = 10000
    shuffle_buffer: int = 10000


class TrainingSettings(BaseModel):
    """Settings for model training."""
    num_epochs: int = 3
    learning_rate: float = 0.005
    weight_decay: float = 1e-4                   # L2 penalty
    log_interval: int = 100


class PlottingSettings(BaseModel):
    """Settings for logging and saving artifacts."""
    output_dir: Path = Path("artifacts")
    figsize: Tuple[int, int] = (5, 3)
    dpi: int = 200


class AppSettings(BaseSettings):
    """Main application settings."""
    debug: bool = False
    random_seed: int = 31415

    model: ModelSettings = ModelSettings()
    data: DataSettings = DataSettings()
    training: TrainingSettings = TrainingSettings()
    plotting: PlottingSettings = PlottingSettings()


def load_settings() -> AppSettings:
    """Load application settings."""
    return AppSettings()
```

```
{"settings": {"debug": false, "random_seed": 31415, "model": {"input_depth": 1,
"layer_depths": [16, 32], "blocks_per_stage": [1, 1], "num_classes": 10, "dropou
t": 0.2}, "data": {"batch_size": 32, "validation_size": 10000, "shuffle_buffer":
 10000}, "training": {"num_epochs": 3, "learning_rate": 0.005, "weight_decay": 0
.0001, "log_interval": 100}, "plotting": {"output_dir": "PosixPath('artifacts')"
, "figsize": [5, 3], "dpi": 200}}, "event": "Settings loaded", "logger": "hw03",
 "level": "info", "timestamp": "2025-09-25T02:07:48.522129Z"}
{"event": "Variant folder /root/tensorflow_datasets/mnist/3.0.1 has no dataset_i
nfo.json"}
{"event": "Generating dataset mnist (/root/tensorflow_datasets/mnist/3.0.1)"}
{"event": "Downloading https://storage.googleapis.com/cvdf-datasets/mnist/t10k-i
mages-idx3-ubyte.gz into /root/tensorflow_datasets/downloads/mnist/cvdf-datasets
_mnist_t10k-images-idx3-ubytedDnaEPiC58ZczHNOp6ks9L4_JLids_rpvUj38kJNGMc.gz.tmp.
585aa212f6fa475f9f15dc5b68fa15cb..."}
{"event": "Downloading https://storage.googleapis.com/cvdf-datasets/mnist/t10k-l
abels-idx1-ubyte.gz into /root/tensorflow_datasets/downloads/mnist/cvdf-datasets
_mnist_t10k-labels-idx1-ubyte4Mqf5UL1fRrpd5pIeeAh8c8ZzsY2gbIPBuKwiyfSD_I.gz.tmp.
b7d62e358adf4378987a4a03fb02f668..."}
{"event": "Downloading https://storage.googleapis.com/cvdf-datasets/mnist/train-
images-idx3-ubyte.gz into /root/tensorflow_datasets/downloads/mnist/cvdf-dataset
s_mnist_train-images-idx3-ubyteJAsxAi0QnOBEygBw_XW2X7zp-LBZAIqqYSHN8ru4ZO4.gz.tm
p.38fc58d802414ab482306728efbe75d0..."}
{"event": "Downloading https://storage.googleapis.com/cvdf-datasets/mnist/train-
labels-idx1-ubyte.gz into /root/tensorflow_datasets/downloads/mnist/cvdf-dataset
s_mnist_train-labels-idx1-ubytedcDWkl3FO9T-WMEH1f1Xt51eIRmePRIMAk6X147Qw8w.gz.tm
p.1a7f3a45749a460b86aaddd869c7c5a3..."}
{"event": "Done writing /root/tensorflow_datasets/mnist/incomplete.WZ3BFY_3.0.1/
mnist-train.tfrecord*. Number of examples: 60000 (shards: [60000])"}
{"event": "Done writing /root/tensorflow_datasets/mnist/incomplete.WZ3BFY_3.0.1/
mnist-test.tfrecord*. Number of examples: 10000 (shards: [10000])"}
{"event": "Creating a tf.data.Dataset reading 1 files located in folders: /root/
tensorflow_datasets/mnist/3.0.1."}
{"event": "Constructing tf.data.Dataset mnist for split train, from /root/tensor
flow_datasets/mnist/3.0.1"}
{"event": "Load dataset info from /root/tensorflow_datasets/mnist/3.0.1"}
{"event": "Creating a tf.data.Dataset reading 1 files located in folders: /root/
tensorflow_datasets/mnist/3.0.1."}
{"event": "Constructing tf.data.Dataset mnist for split test, from /root/tensorf
low_datasets/mnist/3.0.1"}
{"event": "Unable to initialize backend 'tpu': INTERNAL: Failed to open libtpu.s
o: libtpu.so: cannot open shared object file: No such file or directory"}
{"num_params": 19722, "event": "Model created", "logger": "hw03", "level": "info
", "timestamp": "2025-09-25T02:08:09.157200Z"}
{"epoch": 1, "step": 0, "loss": 2.538978099822998, "acc": 0.09375, "event": "tra
in_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T0
2:08:12.446197Z"}
{"epoch": 1, "step": 100, "loss": 2.028336524963379, "acc": 0.125, "event": "tra
in_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T0
2:08:15.838477Z"}
{"epoch": 1, "step": 200, "loss": 1.4762825965881348, "acc": 0.46875, "event": "
train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-2
5T02:08:18.981887Z"}
{"epoch": 1, "step": 300, "loss": 1.838819980621338, "acc": 0.3125, "event": "tr
ain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T
02:08:21.913127Z"}
{"epoch": 1, "step": 400, "loss": 1.0535175800323486, "acc": 0.65625, "event": "
train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-2
5T02:08:25.291893Z"}
{"epoch": 1, "step": 500, "loss": 1.158921480178833, "acc": 0.53125, "event": "t
rain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25
T02:08:29.203533Z"}
{"epoch": 1, "step": 600, "loss": 0.8958538770675659, "acc": 0.5625, "event": "t
rain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25
T02:08:33.340111Z"}
{"epoch": 1, "step": 700, "loss": 1.1473503112792969, "acc": 0.6875, "event": "t
rain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25
T02:08:37.707516Z"}
{"epoch": 1, "step": 800, "loss": 0.5245754718780518, "acc": 0.78125, "event": "
train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-2
```

```
5T02:08:42.157514Z"}
{"epoch": 1, "step": 900, "loss": 0.4066757261753082, "acc": 0.90625, "event": "
train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-2
5T02:08:46.768054Z"}
{"epoch": 1, "step": 1000, "loss": 0.4078470468521118, "acc": 0.84375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:08:51.460418Z"}
{"epoch": 1, "step": 1100, "loss": 0.24065490067005157, "acc": 0.90625, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:08:56.205368Z"}
{"epoch": 1, "step": 1200, "loss": 0.4178001284599304, "acc": 0.90625, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:09:01.311952Z"}
{"epoch": 1, "step": 1300, "loss": 0.31115055084228516, "acc": 0.875, "event": "
train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-2
5T02:09:06.309855Z"}
{"epoch": 1, "step": 1400, "loss": 0.29164016246795654, "acc": 0.90625, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:09:11.258503Z"}
{"epoch": 1, "step": 1500, "loss": 0.29139769077301025, "acc": 0.9375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:09:16.215563Z"}
{"epoch": 1, "step": 1563, "val_loss": 0.30389410909562825, "val_acc": 0.9019568
690095847, "event": "epoch_summary", "logger": "hw03.training", "level": "info",
 "timestamp": "2025-09-25T02:09:34.996285Z"}
{"epoch": 2, "step": 1600, "loss": 0.3914353549480438, "acc": 0.84375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:09:37.786850Z"}
{"epoch": 2, "step": 1700, "loss": 0.24718624353408813, "acc": 0.90625, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:09:42.108987Z"}
{"epoch": 2, "step": 1800, "loss": 0.4984695613384247, "acc": 0.84375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:09:46.611778Z"}
{"epoch": 2, "step": 1900, "loss": 0.2861315608024597, "acc": 0.90625, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:09:51.110826Z"}
{"epoch": 2, "step": 2000, "loss": 0.14241105318069458, "acc": 0.90625, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:09:55.552855Z"}
{"epoch": 2, "step": 2100, "loss": 0.2858550548553467, "acc": 0.90625, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:10:00.053447Z"}
{"epoch": 2, "step": 2200, "loss": 0.06083556264638901, "acc": 1.0, "event": "tr
ain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T
02:10:04.499250Z"}
{"epoch": 2, "step": 2300, "loss": 0.19406363368034363, "acc": 0.9375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:10:08.978488Z"}
{"epoch": 2, "step": 2400, "loss": 0.08005142211914062, "acc": 1.0, "event": "tr
ain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T
02:10:13.494265Z"}
{"epoch": 2, "step": 2500, "loss": 0.18751657009124756, "acc": 0.9375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:10:17.805327Z"}
{"epoch": 2, "step": 2600, "loss": 0.26830390095710754, "acc": 0.90625, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:10:22.037843Z"}
{"epoch": 2, "step": 2700, "loss": 0.21286556124687195, "acc": 0.90625, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:10:26.305835Z"}
{"epoch": 2, "step": 2800, "loss": 0.3422226309776306, "acc": 0.84375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:10:30.701704Z"}
{"epoch": 2, "step": 2900, "loss": 0.2789686918258667, "acc": 0.875, "event": "t
rain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25
T02:10:35.306584Z"}
{"epoch": 2, "step": 3000, "loss": 0.4412480592727661, "acc": 0.875, "event": "t
rain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25
```

```
T02:10:40.052853Z"}
{"epoch": 2, "step": 3100, "loss": 0.1342550665140152, "acc": 0.96875, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:10:44.749327Z"}
{"epoch": 2, "step": 3126, "val_loss": 0.16497566046330114, "val_acc": 0.9504792
33226837, "event": "epoch_summary", "logger": "hw03.training", "level": "info",
"timestamp": "2025-09-25T02:10:56.240764Z"}
{"epoch": 3, "step": 3200, "loss": 0.2555471956729889, "acc": 0.875, "event": "t
rain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25
T02:11:00.780476Z"}
{"epoch": 3, "step": 3300, "loss": 0.39086252450942993, "acc": 0.90625, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:11:05.014910Z"}
{"epoch": 3, "step": 3400, "loss": 0.06019824743270874, "acc": 1.0, "event": "tr
ain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T
02:11:09.379806Z"}
{"epoch": 3, "step": 3500, "loss": 0.16334864497184753, "acc": 0.9375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:11:13.633909Z"}
{"epoch": 3, "step": 3600, "loss": 0.060306839644908905, "acc": 1.0, "event": "t
rain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25
T02:11:17.993204Z"}
{"epoch": 3, "step": 3700, "loss": 0.1764403134584427, "acc": 0.96875, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:11:22.038726Z"}
{"epoch": 3, "step": 3800, "loss": 0.10806598514318466, "acc": 1.0, "event": "tr
ain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T
02:11:26.221545Z"}
{"epoch": 3, "step": 3900, "loss": 0.18277491629123688, "acc": 0.90625, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:11:31.095782Z"}
{"epoch": 3, "step": 4000, "loss": 0.274816632270813, "acc": 0.9375, "event": "t
rain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25
T02:11:35.564553Z"}
{"epoch": 3, "step": 4100, "loss": 0.1901731938123703, "acc": 0.9375, "event": "
train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-2
5T02:11:40.041957Z"}
{"epoch": 3, "step": 4200, "loss": 0.11744438856840134, "acc": 0.96875, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:11:44.686833Z"}
{"epoch": 3, "step": 4300, "loss": 0.07564717531204224, "acc": 0.96875, "event":
 "train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09
-25T02:11:49.304296Z"}
{"epoch": 3, "step": 4400, "loss": 0.1251562535762787, "acc": 0.9375, "event": "
train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-2
5T02:11:54.006251Z"}
{"epoch": 3, "step": 4500, "loss": 0.07822797447443008, "acc": 1.0, "event": "tr
ain_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T
02:11:58.744469Z"}
{"epoch": 3, "step": 4600, "loss": 0.11893424391746521, "acc": 0.9375, "event":
"train_iter", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-
25T02:12:03.439134Z"}
{"epoch": 3, "step": 4689, "val_loss": 0.1339081363460888, "val_acc": 0.95956469
6485623, "event": "epoch_summary", "logger": "hw03.training", "level": "info", "
timestamp": "2025-09-25T02:12:16.610756Z"}
{"event": "Training finished. Generating final plots...", "logger": "hw03.traini
ng", "level": "info", "timestamp": "2025-09-25T02:12:16.611829Z"}
{"event": "Starting final testing...", "logger": "hw03", "level": "info", "times
tamp": "2025-09-25T02:12:17.183976Z"}
{"test_loss": 0.1314426691482623, "test_acc": 0.959564696485623, "event": "test_
results", "logger": "hw03.training", "level": "info", "timestamp": "2025-09-25T0
2:12:26.280264Z"}
{"test_loss": 0.1314426691482623, "test_acc": 0.959564696485623, "event": "Final
 Test Results", "logger": "hw03", "level": "info", "timestamp": "2025-09-25T02:1
2:27.797058Z"}
```

```python
import matplotlib
import matplotlib.pyplot as plt
import structlog

log = structlog.get_logger()

font = {
    # "family": "Adobe Caslon Pro",
    "size": 10,
}

matplotlib.style.use("classic")
matplotlib.rc("font", **font)

def plot_losses(train_losses, val_losses, settings):
    plt.figure(figsize=(8, 5))
    plt.plot(train_losses, label="Train Loss", alpha=0.7)
    steps_per_epoch = 50000 // settings.training.batch_size
    val_x = [i * steps_per_epoch for i in range(1, len(val_losses)+1)]
    plt.plot(val_x, val_losses, label="Validation Loss", marker="o")
    plt.xlabel("Step")
    plt.ylabel("Loss")
    plt.title("Training & Validation Loss")
    plt.legend()
    plt.grid(True)

    settings.plotting.output_dir.mkdir(parents=True, exist_ok=True)
    output_path = settings.plotting.output_dir / "Loss Graph.pdf"
    plt.savefig(output_path)
    log.info("Saved Loss Graph plot", path=str(output_path))
```

```
import optax
import structlog
from flax import nnx
import jax
import pathlib as Path

from .config import load_settings
from .logging import configure_logging
from .model import Classifier
from .training import train, test_evaluation
from .data import load_mnist

def main() -> None:
    settings = load_settings()
    log_file = configure_logging()
    log = structlog.get_logger()
    log.info("Settings loaded", settings=settings.model_dump())
    print(f"Logs are being saved to {log_file}")

    ds_train, ds_val, ds_test = load_mnist(
        batch_size=settings.data.batch_size,
        validation_size=settings.data.validation_size,
    )
    rngs = nnx.Rngs(params=settings.random_seed, dropout=settings.random_seed +
1)
    model = Classifier(
        rngs=rngs,
        input_depth=settings.model.input_depth,
        layer_depths=settings.model.layer_depths,
        blocks_per_stage=settings.model.blocks_per_stage,
        layer_kernel_sizes=[(3,3)] * len(settings.model.layer_depths),
        num_classes=settings.model.num_classes,
        dropout=settings.model.dropout,
    )

    params = nnx.state(model, nnx.Param)
    num_params = sum(p.size for p in jax.tree_util.tree_leaves(params))
    log.info("Model created", num_params=num_params)

    # LR schedule and optimizer
    train_size      = 50000
    steps_per_epoch = train_size // settings.data.batch_size
    decay_steps     = settings.training.num_epochs * steps_per_epoch
    lr_schedular = optax.schedules.cosine_decay_schedule(
        init_value=settings.training.learning_rate,
        decay_steps=decay_steps,
    )
    optimizer = nnx.Optimizer(
        model,
        optax.adamw(
            learning_rate=lr_schedular,
            weight_decay=settings.training.weight_decay,
        ),
        wrt=nnx.Param,
    )

    _ = train(model, optimizer, ds_train, ds_val, rngs)

    # Test
    log.info("Starting final testing...")
    test_loss, test_acc = test_evaluation(model, ds_test, rngs)
    log.info("Final Test Results", test_loss=test_loss, test_acc=test_acc)
```

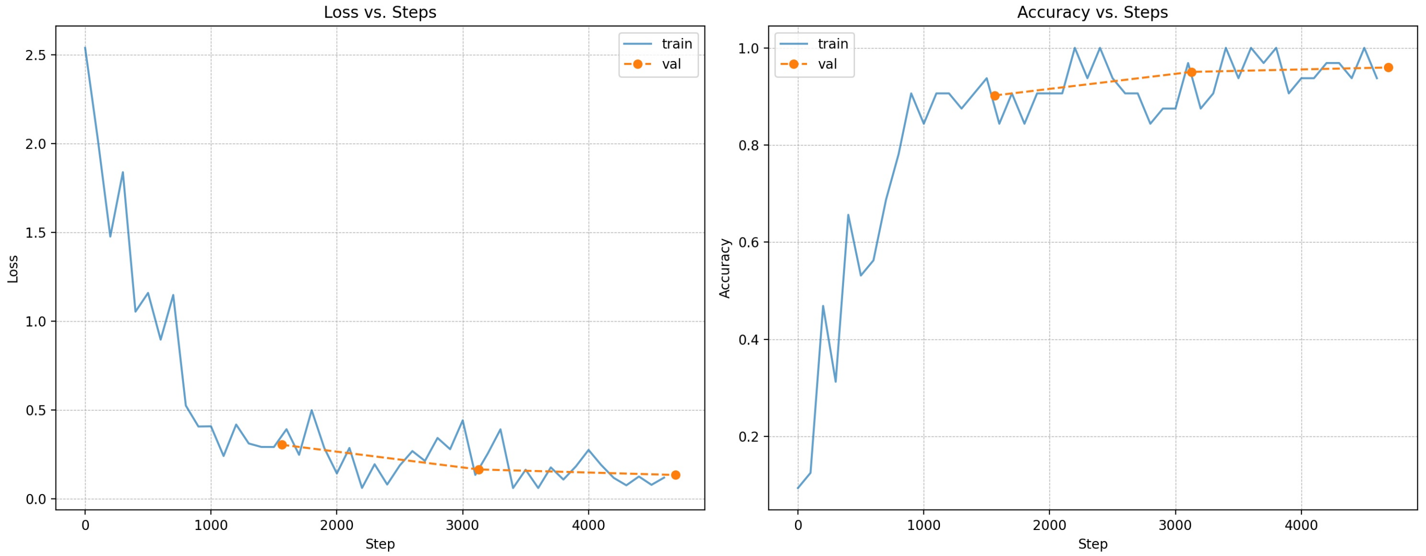
```
import jax
import jax.numpy as jnp
from flax import nnx

class Conv2d(nnx.Module):
    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        kernel_size,
        *,
        strides=(1, 1),
        padding="SAME",
        use_bias=True,
        dropout=0.0,
        rngs: nnx.Rngs,
    ):
        self.conv = nnx.Conv(
            in_channels, out_channels, kernel_size,
            strides=strides, padding=padding, use_bias=use_bias, rngs=rngs
        )
        self.dropout = nnx.Dropout(dropout, rngs=rngs) if dropout > 0.0 else Non
e

    def __call__(self, x, *, rngs: nnx.Rngs):
        x = self.conv(x)
        if self.dropout:
            x = self.dropout(x, rngs=rngs)
        return x


class ResidualBlock(nnx.Module):
    """Basic residual block."""
    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        *,
        stride=(1, 1),
        dropout=0.0,
        rngs: nnx.Rngs,
    ):
        self.norm1 = nnx.GroupNorm(num_features=in_channels, num_groups=8, rngs=
rngs)
        self.conv1 = Conv2d(
            in_channels, out_channels, (3, 3),
            strides=stride, dropout=dropout, rngs=rngs,
        )
        self.norm2 = nnx.GroupNorm(num_features=out_channels, num_groups=8, rngs
=rngs)
        self.conv2 = Conv2d(
            out_channels, out_channels, (3, 3),
            strides=(1, 1), dropout=0.0, rngs=rngs,
        )
        self.shortcut = (
            Conv2d(
                in_channels, out_channels, (1, 1),
                strides=stride, dropout=0.0, rngs=rngs,
            )
            if in_channels != out_channels or stride != (1, 1)
            else None
        )

    def __call__(self, x, *, rngs: nnx.Rngs):
        out = self.norm1(x)
        out = nnx.relu(out)
        out = self.conv1(out, rngs=rngs)

        out = self.norm2(out)
```

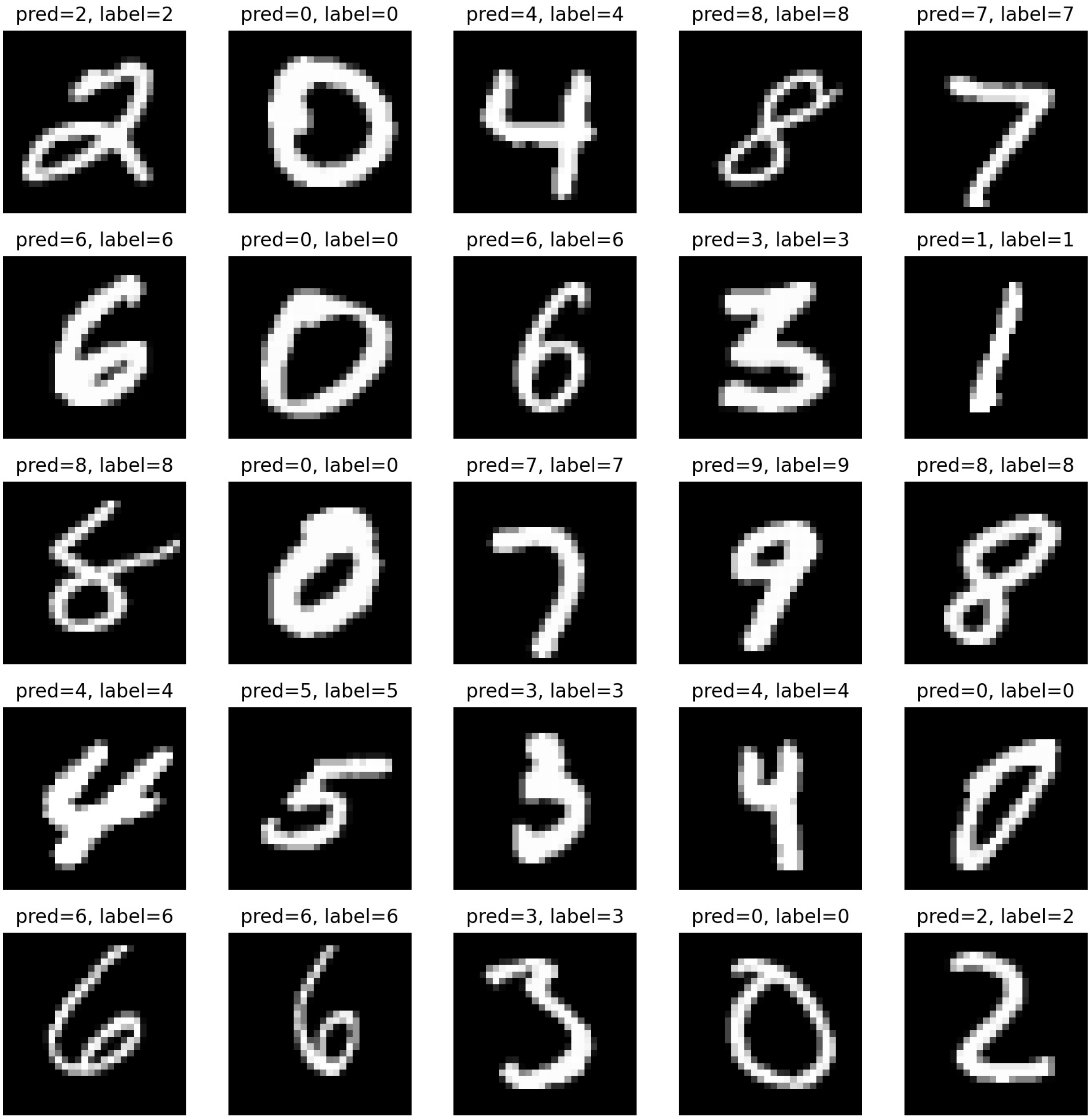
```
        out = nnx.relu(out)
        out = self.conv2(out, rngs=rngs)

        identity = x if self.shortcut is None else self.shortcut(x, rngs=rngs)
        return out + identity

class ResidualStage(nnx.Module):
    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        num_blocks: int,
        dropout: float,
        stride=(1, 1),
        *,
        rngs: nnx.Rngs,
    ):
        self.blocks = []
        self.blocks.append(
            ResidualBlock(
                in_channels,
                out_channels,
                stride=stride,
                dropout=dropout,
                rngs=rngs,
            )
        )
        # Rest keep stride = (1,1)
        for _ in range(1, num_blocks):
            self.blocks.append(
                ResidualBlock(
                    out_channels,
                    out_channels,
                    stride=(1, 1),
                    dropout=dropout,
                    rngs=rngs,
                )
            )

    def __call__(self, x, *, rngs: nnx.Rngs):
        for block in self.blocks:
            x = block(x, rngs=rngs)
        return x

class Classifier(nnx.Module):
    """Residual network classifier for MNIST."""

    def __init__(
        self,
        input_depth: int,
        layer_depths: list[int],
        blocks_per_stage,
        layer_kernel_sizes: list[tuple[int, int]],
        num_classes: int,
        *,
        rngs: nnx.Rngs,
        dropout: float = 0.1,
    ):
        assert len(layer_depths) == len(layer_kernel_sizes), \
            "layer_depths and layer_kernel_sizes must match in length"
        self.init_conv = Conv2d(input_depth, layer_depths[0],
                                kernel_size=layer_kernel_sizes[0],
                                strides=(1, 1), dropout=dropout, rngs=rngs)

        # Build stages
        self.stages = []
        self.blocks_per_stage = blocks_per_stage
        in_ch = layer_depths[0]
        for i, out_ch in enumerate(layer_depths):
            stride = (2, 2) if i > 0 else (1, 1)  # downsample at stage boundari
es
            num_blocks = (self.blocks_per_stage[i]
                          if self.blocks_per_stage is not None
                          else 1)
            stage = ResidualStage(
                in_channels=in_ch,
                out_channels=out_ch,
                num_blocks=num_blocks,
                stride=stride,
                dropout=dropout,
                rngs=rngs,
            )
            self.stages.append(stage)
            in_ch = out_ch

        # Final classifier head
        self.fc = nnx.Linear(in_ch, num_classes, rngs=rngs)

    def __call__(self, x, *, rngs: nnx.Rngs):
        x = self.init_conv(x, rngs=rngs)
        for stage in self.stages:
            x = stage(x, rngs=rngs)

        x = jnp.mean(x, axis=(1, 2))

        # Final logits
        return self.fc(x)
```

```python
import jax
import jax.numpy as jnp
import optax
from flax import nnx
import matplotlib.pyplot as plt
import structlog

from .config import load_settings

log = structlog.get_logger()

def cross_entropy_loss(logits, labels):
    return optax.softmax_cross_entropy_with_integer_labels(
        logits, labels
    ).mean()


def compute_accuracy(logits, labels):
    preds = jnp.argmax(logits, axis=-1)
    return jnp.mean(preds == labels)


def _loss_fn(model, batch, rngs):
    images, labels = batch
    labels = labels.ravel()
    logits = model(images, rngs=rngs)
    loss = cross_entropy_loss(logits, labels)
    return loss, logits


@nnx.jit
def train_step(model, optimizer, batch, rngs):
    grad_fn = nnx.value_and_grad(_loss_fn, has_aux=True)
    (loss, logits), grads = grad_fn(model, batch, rngs)
    optimizer.update(model, grads)
    return loss, logits


@nnx.jit
def eval_step(model, batch, rngs):
    loss, logits = _loss_fn(model, batch, rngs)
    return loss, logits

def evaluate(model, dataset, rngs):
    total_loss, total_acc, n_batches = 0.0, 0.0, 0
    for batch in dataset.as_numpy_iterator():
        loss, logits = eval_step(model, batch, rngs)
        _, labels = batch
        labels = labels.ravel()
        acc = compute_accuracy(logits, labels)
        loss = float(loss)
        acc = float(acc)
        total_loss += loss
        total_acc += float(acc)
        n_batches += 1
    return total_loss / max(n_batches, 1), total_acc / max(n_batches, 1)

def train(model, optimizer, train_ds, val_ds, rngs):
    settings = load_settings()
    output_dir = settings.plotting.output_dir
    output_dir.mkdir(parents=True, exist_ok=True)

    log_interval = settings.training.log_interval
    num_epochs = settings.training.num_epochs

    metrics_history = {
    "train_steps": [],
    "train_loss": [],
    "train_accuracy": [],
```

```python
    "val_steps": [],
    "val_loss": [],
    "val_accuracy": [],
}

    global_step = 0

    for epoch in range(1, num_epochs + 1):

        for batch in train_ds.as_numpy_iterator():
            loss, logits = train_step(model, optimizer, batch, rngs)

            if global_step % log_interval == 0:
                images, labels = batch
                labels = labels.ravel()
                acc = compute_accuracy(logits, labels)

                loss_val = float(loss)
                acc_val = float(acc)

                metrics_history["train_steps"].append(global_step)
                metrics_history["train_loss"].append(loss_val)
                metrics_history["train_accuracy"].append(acc_val)

                log.info(
                    "train_iter",
                    epoch=epoch,
                    step=global_step,
                    loss=loss_val,
                    acc=acc_val,
                )
            global_step += 1


        val_loss, val_acc = evaluate(model, val_ds, rngs)
        metrics_history["val_steps"].append(global_step)
        metrics_history["val_loss"].append(val_loss)
        metrics_history["val_accuracy"].append(val_acc)

        log.info(
            "epoch_summary",
            epoch=epoch,
            step=global_step,
            val_loss=val_loss,
            val_acc=val_acc,
        )

    log.info("Training finished. Generating final plots...")
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    ax1.set_title("Loss vs. Steps")
    ax2.set_title("Accuracy vs. Steps")

    ax1.plot(metrics_history["train_steps"], metrics_history["train_loss"], label="train"
, alpha=0.7)
    ax2.plot(metrics_history["train_steps"], metrics_history["train_accuracy"], label="t
rain", alpha=0.7)

    ax1.plot(metrics_history["val_steps"], metrics_history["val_loss"], label="val", m
arker='o', linestyle='--')
    ax2.plot(metrics_history["val_steps"], metrics_history["val_accuracy"], label="val
", marker='o', linestyle='--')

    ax1.set_xlabel("Step")
    ax1.set_ylabel("Loss")
    ax1.legend()
    ax1.grid(True, which='both', linestyle='--', linewidth=0.5)

    ax2.set_xlabel("Step")
    ax2.set_ylabel("Accuracy")
    ax2.legend()
    ax2.grid(True, which='both', linestyle='--', linewidth=0.5)

    fig.tight_layout()
    fig.savefig(output_dir / "final_training_metrics.png", dpi=settings.plotting.dpi)
    plt.close(fig)

    return metrics_history

def test_evaluation(model, test_ds, rngs):
    settings = load_settings()
    output_dir = settings.plotting.output_dir
    output_dir.mkdir(parents=True, exist_ok=True)

    test_loss, test_acc = evaluate(model, test_ds, rngs)
    log.info("test_results", test_loss=test_loss, test_acc=test_acc)

    test_batch = next(test_ds.as_numpy_iterator())
    _, logits = eval_step(model, test_batch, rngs)
    images, labels = test_batch
    preds = jnp.argmax(logits, axis=-1)

    fig, axs = plt.subplots(5, 5, figsize=(10, 10))
    for i, ax in enumerate(axs.flatten()):
        ax.imshow(images[i, ..., 0], cmap="gray")
        ax.set_title(f"pred={int(preds[i])}, label={int(labels[i])}")
        ax.axis("off")
    fig.tight_layout()
    fig.savefig(output_dir / "test_predictions.png", dpi=settings.plotting.dpi)
    plt.close(fig)

    return test_loss, test_acc
```

```
import tensorflow_datasets as tfds
import tensorflow as tf

def load_mnist(batch_size=128, validation_size=10000, seed=0):
    ds_train = tfds.load("mnist", split="train", shuffle_files=True, as_supervised=
True)
    ds_test  = tfds.load("mnist", split="test", as_supervised=True)

    def _preprocess(image, label):
        image = tf.cast(image, tf.float32) / 255.0
        label = tf.cast(label, tf.int32)
        return image, label

    ds_train = ds_train.map(_preprocess, num_parallel_calls=tf.data.AUTOTUNE)
    ds_test  = ds_test.map(_preprocess, num_parallel_calls=tf.data.AUTOTUNE)

    ds_val = ds_train.take(validation_size)
    ds_train = ds_train.skip(validation_size)

    ds_train = ds_train.shuffle(10 * batch_size, seed=seed).batch(batch_size).pr
efetch(tf.data.AUTOTUNE)
    ds_val   = ds_val.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    ds_test  = ds_test.batch(batch_size).prefetch(tf.data.AUTOTUNE)

    return ds_train, ds_val, ds_test
```


Loss vs. Steps
Loss
Step
train
val
0.0
0.5
1.0
1.5
2.0
2.5
0
1000
2000
3000
4000
Accuracy vs. Steps
Accuracy
Step
train
val
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000


pred=2, label=2
pred=0, label=0
pred=4, label=4
pred=8, label=8
pred=7, label=7
pred=6, label=6
pred=0, label=0
pred=6, label=6
pred=3, label=3
pred=1, label=1
pred=8, label=8
pred=0, label=0
pred=7, label=7
pred=9, label=9
pred=8, label=8
pred=4, label=4
pred=5, label=5
pred=3, label=3
pred=4, label=4
pred=0, label=0
pred=6, label=6
pred=6, label=6
pred=3, label=3
pred=0, label=0
pred=2, label=2