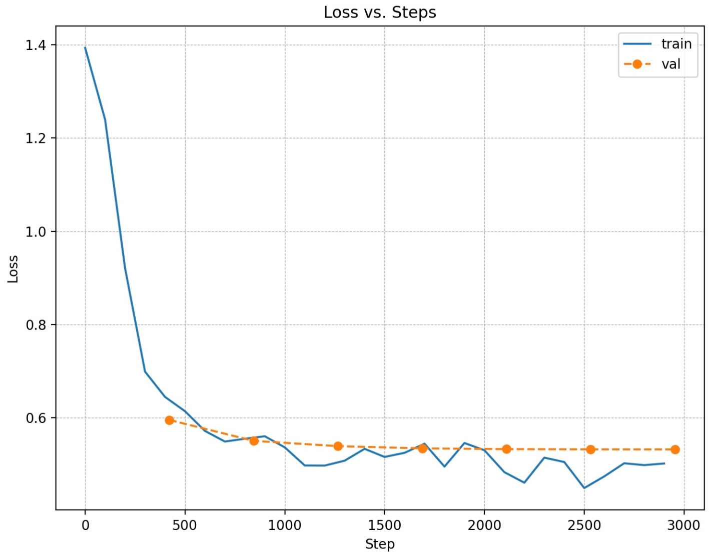
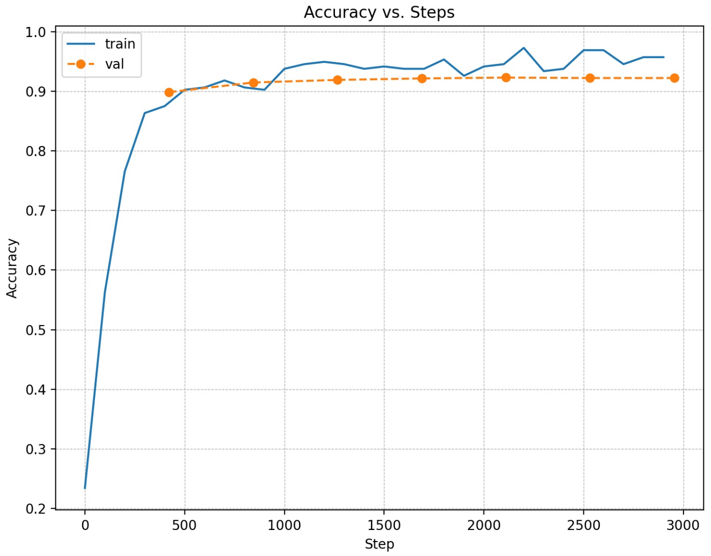
```
{"settings": {"debug": false, "random_seed": 31415, "model": {"num_classes": 4,
"embed_dim": 128, "conv_features": 64, "kernel_size": 5, "dropout_rate": 0.3}, "
data": {"validation_percentage": 0.1, "batch_size": 256, "max_length": 128}, "tr
aining": {"num_epochs": 7, "learning_rate": 0.0005, "weight_decay": 0.0001, "log
_interval": 100, "final_test": true, "checkpoint_dir": "PosixPath('hw05/checkpoi
nts')"}, "plotting": {"output_dir": "PosixPath('hw05/artifacts')", "figsize": [5
, 3], "dpi": 200}}, "event": "Settings loaded", "logger": "hw05", "level": "info
", "timestamp": "2025-10-16T00:29:50.742484Z"}
{"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": 3948100, "event": "Model created", "logger": "hw05", "level": "in
fo", "timestamp": "2025-10-16T00:30:09.960559Z"}
{"event": "Starting final testing...", "logger": "hw05", "level": "info", "times
tamp": "2025-10-16T00:30:10.044023Z"}
{"event": "orbax-checkpoint version: 0.11.25"}
{"event": "Created BasePyTreeCheckpointHandler: use_ocdbt=True, use_zarr3=False,
 pytree_metadata_options=PyTreeMetadataOptions(support_rich_types=False), array_
metadata_store=<orbax.checkpoint._src.metadata.array_metadata_store.Store object
 at 0x7f34c9d01790>, enable_pinned_host_transfer=False, save_concurrent_bytes: 9
6000000000 (89.4 GiB), restore_concurrent_bytes: 96000000000 (89.4 GiB)"}
{"event": "[thread=MainThread] Failed to get flag value for EXPERIMENTAL_ORBAX_U
SE_DISTRIBUTED_PROCESS_ID."}
{"event": "[process=0][thread=MainThread] Using barrier_sync_fn: <function get_b
arrier_sync_fn.<locals>.<lambda> at 0x7f345c30f4c0> timeout: 600 secs and primar
y_host=0 for async checkpoint writes"}
{"event": "Restoring checkpoint from /work/hw05/checkpoints/best_state_epoch_5."
}
{"event": "[process=0] /jax/checkpoint/read/gbytes_per_sec: 0 Bytes/s (total gby
tes: 15.1 MiB) (time elapsed: 185 milliseconds) (per-host)"}
{"event": "Finished restoring checkpoint in 0.20 seconds from /work/hw05/checkpo
ints/best_state_epoch_5."}
{"test_loss": 0.5386621763308843, "test_acc": 0.9197916666666667, "event": "test
_results", "logger": "hw05.training", "level": "info", "timestamp": "2025-10-16T
00:30:11.412115Z"}
{"test_loss": 0.5386621763308843, "test_acc": 0.9197916666666667, "event": "Fina
l Test Results", "logger": "hw05", "level": "info", "timestamp": "2025-10-16T00:
30:11.412709Z"}
```


Loss vs. Steps
train
val
Loss
Step
1.4
1.2
1.0
0.8
0.6
0
500
1000
1500
2000
2500
3000


Accuracy vs. Steps
train
val
Accuracy
Step
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
500
1000
1500
2000
2500
3000

```
# pyproject.toml.jinja

[project]
name = "hw05"
version = "0.1.0"
description = "AG News Classification"
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",
    "transformers",
    "datasets",
]

[project.scripts]
hw05 = "hw05: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("hw05/artifacts"), log_name: str = "train.jso
n"):
    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
```

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


class ModelSettings(BaseModel):
    num_classes: int = 4
    embed_dim: int = 128
    conv_features: int = 64
    kernel_size: int = 5
    dropout_rate: float = 0.3


class DataSettings(BaseModel):
    validation_percentage: float = 0.1
    batch_size: int = 256
    max_length: int = 128


class TrainingSettings(BaseModel):
    """Settings for model training."""
    num_epochs: int = 7
    learning_rate: float = 0.0005
    weight_decay: float = 1e-4                   # L2 penalty
    log_interval: int = 100
    final_test: bool = True
    checkpoint_dir: Path = Path("hw05/checkpoints")


class PlottingSettings(BaseModel):
    """Settings for logging and saving artifacts."""
    output_dir: Path = Path("hw05/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()
```

```
import optax
import structlog
from flax import nnx
import jax
import jax.numpy as jnp
import pathlib as Path
import orbax.checkpoint as ocp

from .config import load_settings
from .logging import configure_logging
from .model import Conv1DTextClassifier
from .training import train, test_evaluation, load_checkpoint
from .data import load_AG_News, tokenize_datasets

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}")


    raw_ds = load_AG_News(
        val_per=settings.data.validation_percentage
    )

    rngs = nnx.Rngs(params=settings.random_seed, dropout=settings.random_seed +
1)
    tokenized, tokenizer = tokenize_datasets(
        raw_ds,
        model_name="distilbert-base-uncased",
        max_length=settings.data.max_length,
    )
    def make_batches(ds_split, batch_size):
        for i in range(0, len(ds_split), batch_size):
            batch = ds_split[i : i + batch_size]
            yield {
                "input_ids": jnp.array(batch["input_ids"]),
                "label": jnp.array(batch["label"]),
            }

    train_batches = list(make_batches(tokenized["train"], settings.data.batch_siz
e))
    val_batches   = list(make_batches(tokenized["val"], settings.data.batch_size
))
    test_batches  = list(make_batches(tokenized["test"], settings.data.batch_size
))

    model = Conv1DTextClassifier(
        vocab_size=tokenizer.vocab_size,
        num_classes=settings.model.num_classes,
        embed_dim=settings.model.embed_dim,
        conv_features=settings.model.conv_features,
        kernel_size=settings.model.kernel_size,
        dropout_rate=settings.model.dropout_rate,
        rngs=rngs,
    )

    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)

    steps_per_epoch = len(train_batches)
    decay_steps = settings.training.num_epochs * steps_per_epoch

    lr_schedule = optax.schedules.cosine_decay_schedule(
        init_value=settings.training.learning_rate,
        decay_steps=decay_steps,
    )
```

```

    optimizer = nnx.Optimizer(
        model,
        optax.adamw(
            learning_rate=lr_schedule,
            weight_decay=settings.training.weight_decay,
        ),
        wrt=nnx.Param,
    )

    if settings.training.final_test:
        log.info("Starting final testing...")

        model_kwargs = dict(
            vocab_size=tokenizer.vocab_size,
            num_classes=settings.model.num_classes,
            embed_dim=settings.model.embed_dim,
            conv_features=settings.model.conv_features,
            kernel_size=settings.model.kernel_size,
            dropout_rate=settings.model.dropout_rate,
        )

        model = load_checkpoint(Conv1DTextClassifier, model_kwargs)
        test_loss, test_acc = test_evaluation(model, test_batches, rngs)
        log.info("Final Test Results", test_loss=test_loss, test_acc=test_acc)
    else:
        _ = train(model, optimizer, train_batches, val_batches, rngs)
```

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

class Conv1DTextClassifier(nnx.Module):
    def __init__(
        self,
        vocab_size: int,
        num_classes: int = 4,
        embed_dim: int = 128,
        conv_features: int = 64,
        kernel_size: int = 5,
        dropout_rate: float = 0.2,
        rngs: nnx.Rngs = None,
    ):
        self.embedding = nnx.Embed(
            num_embeddings=vocab_size,
            features=embed_dim,
            rngs=rngs,
        )
        self.conv1 = nnx.Conv(
            in_features=embed_dim,
            out_features=conv_features,
            kernel_size=(kernel_size,),
            rngs=rngs,
        )
        self.dropout = nnx.Dropout(rate=dropout_rate, rngs=rngs)
        self.linear = nnx.Linear(conv_features, num_classes, rngs=rngs)

    def __call__(self, input_ids, *, train=True):
        x = self.embedding(input_ids)
        x = jax.nn.relu(self.conv1(x))
        x = jnp.max(x, axis=1)
        x = self.dropout(x, deterministic=bool(not train))
        logits = self.linear(x)
        return logits
```

```python
# src/hw05/training.py
import jax
import jax.numpy as jnp
import optax
from flax import nnx
import matplotlib.pyplot as plt
import structlog
import orbax.checkpoint as ocp
from pathlib import Path
import os

from .config import load_settings

log = structlog.get_logger()

# Loss & accuracy

def cross_entropy_loss(logits, labels, num_classes, smoothing=0.1):
    one_hot = jax.nn.one_hot(labels, num_classes)
    smoothed = one_hot * (1 - smoothing) + smoothing / num_classes
    loss = optax.softmax_cross_entropy(logits, smoothed).mean()
    return loss

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


# Core loss_fn
def _loss_fn(model, batch, rngs, train):
    input_ids = batch["input_ids"]
    labels = batch["label"]
    settings = load_settings()
    logits = model(input_ids, train=train)
    loss = cross_entropy_loss(logits, labels, settings.model.num_classes)
    return loss, logits


# Train & Eval Steps
@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, True)
    optimizer.update(model, grads)
    return loss, logits


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


# Epoch Evaluation

def evaluate(model, dataset, rngs):
    total_loss, total_acc, n_batches = 0.0, 0.0, 0
    for batch in dataset:
        loss, logits = eval_step(model, batch, rngs, train=False)
        labels = batch["label"]
        acc = compute_accuracy(logits, labels)
        total_loss += float(loss)
        total_acc += float(acc)
        n_batches += 1
    return total_loss / max(n_batches, 1), total_acc / max(n_batches, 1)


# Main Training Loop
def train(model, optimizer, train_ds, val_ds, rngs):
```

```python
    settings = load_settings()
    output_dir = settings.plotting.output_dir
    output_dir.mkdir(parents=True, exist_ok=True)
    checkpoint_dir = settings.training.checkpoint_dir.resolve()
    checkpoint_dir.mkdir(parents=True, exist_ok=True)
    checkpoint_dir = ocp.test_utils.erase_and_create_empty(checkpoint_dir)

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

    metrics_history = {
        "train_steps": [],
        "train_loss": [],
        "train_accuracy": [],
        "val_steps": [],
        "val_loss": [],
        "val_accuracy": [],
    }

    global_step = 0
    best_val_acc = 0.0
    best_ckpt_path = None

    for epoch in range(1, num_epochs + 1):
        for i, batch in enumerate(train_ds):
            loss, logits = train_step(model, optimizer, batch, rngs)

            if global_step % log_interval == 0:
                labels = batch["label"]
                acc = compute_accuracy(logits, labels)
                metrics_history["train_steps"].append(global_step)
                metrics_history["train_loss"].append(float(loss))
                metrics_history["train_accuracy"].append(float(acc))
                log.info(
                    "train_iter",
                    epoch=epoch,
                    step=global_step,
                    loss=float(loss),
                    acc=float(acc),
                )
            global_step += 1

        # Evaluate
        val_loss, val_acc = evaluate(model, val_ds, rngs)
        if epoch % 5 == 0 or epoch == num_epochs:
            checkpointer = ocp.StandardCheckpointer()
            _, state = nnx.split(model)
            ckpt_path = checkpoint_dir / f"state_epoch_{epoch}"
            param_state = nnx.state(model, nnx.Param)
            checkpointer.save(ckpt_path, param_state)
            if val_acc > best_val_acc:
                best_val_acc = val_acc
                best_ckpt_path = checkpoint_dir / f"best_state_epoch_{epoch}"
                checkpointer.save(best_ckpt_path, param_state)

        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,
        )

    # Plotting metrics
    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"
)
    ax2.plot(metrics_history["train_steps"], metrics_history["train_accuracy"], label="t
rain")

    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")
    ax2.set_xlabel("Step")
    ax2.set_ylabel("Accuracy")

    ax1.legend(); ax2.legend()
    ax1.grid(True, linestyle='--', linewidth=0.5)
    ax2.grid(True, linestyle='--', linewidth=0.5)

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

    log.info("Training complete.")
    return metrics_history


# Test Evaluation
def test_evaluation(model, test_ds, rngs):
    test_loss, test_acc = evaluate(model, test_ds, rngs)
    log.info("test_results", test_loss=test_loss, test_acc=test_acc)
    return test_loss, test_acc


def load_checkpoint(model_cls, model_kwargs: dict):
    settings = load_settings()
    checkpoint_dir = settings.training.checkpoint_dir.resolve()
    checkpointer = ocp.StandardCheckpointer()

    ckpts = sorted(checkpoint_dir.glob("best_state_epoch_*"))
    if not ckpts:
        raise FileNotFoundError(f"No checkpoints found in {checkpoint_dir}")
    latest_ckpt = ckpts[-1]
    print(f"Loading checkpoint from: {latest_ckpt}")

    rngs = nnx.Rngs(0)
    model = model_cls(**model_kwargs, rngs=rngs)

    param_ref = nnx.state(model, nnx.Param)
    restored_params = checkpointer.restore(latest_ckpt, param_ref)

    nnx.update(model, restored_params)

    print("Checkpoint successfully restored!")
    return model
```

```
from datasets import load_dataset
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer


def load_AG_News(val_per: float = 0.1, seed = 0):
    raw = load_dataset("ag_news")
    df = raw["train"].to_pandas()
    train_df, val_df = train_test_split(
        df, test_size=val_per, stratify=df["label"], random_state=seed
    )

    # Re-wrap as Hugging Face Dataset objects
    from datasets import Dataset
    train_ds = Dataset.from_pandas(train_df, preserve_index=False)
    val_ds   = Dataset.from_pandas(val_df, preserve_index=False)

    return {
        "train": train_ds,
        "val": val_ds,
        "test": raw["test"],
    }


def tokenize_datasets(ds_dict, model_name="distilbert-base-uncased", max_length=128):
    """
    Tokenize AG News datasets using a pretrained DistilBERT tokenizer.
    Returns tokenized datasets and the tokenizer.
    """
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    def preprocess(examples):
        return tokenizer(
            examples["text"],
            padding="max_length",
            truncation=True,
            max_length=max_length,
        )

    tokenized = {}
    for split, ds in ds_dict.items():
        tokenized[split] = ds.map(preprocess, batched=True)
        tokenized[split].set_format(
            type="np",
            columns=["input_ids", "attention_mask", "label"],
        )

    return tokenized, tokenizer
```