```
#!/usr/bin/env python3

"""
This is built to be a general purpose configuration system for
supervised/unsupervised learning tasks.
Process:
    (1) Read configuration file
    (2) Create batch of experiments based on default configuration and options
    (3) Spit out each experiment as a separate process
    (4) Within each experiment subprocess, run train and logs results
    (5) Evaluate functions available after training is complete
"""

import pickle
import os
import sys
import json
import subprocess
import itertools

import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.optimizers import Adam

from config.config_handler import config_handler
from exceptions import MissingConfigArgException

def main():
    """Spits out training jobs for each configuration"""

    # Get cfg name
    cfg_name = get_cfg_name()

    # Create configuration handler
    cfg_handler = config_handler(cfg_name)

    if cfg_handler.get_mode() == "train":
        # Create index file
        create_index_log(cfg_handler)

    # Start a job for each hyperparameter
    print("Number of jobs to start: {}".format(cfg_handler.get_num_experiments()))
    for i in range(cfg_handler.get_num_experiments()):
        start_job(cfg_name, i, cfg_handler.get_mode())

def get_cfg_name():
    """Get cfg name from the command line args"""

    for arg in sys.argv:
        if "-cfg_name=" in arg:
            return arg.replace("-cfg_name=", "")

    raise MissingConfigArgException()

def get_exp_num():
    """Get experiment number from the command line args"""

    for arg in sys.argv:
        if "-exp_num=" in arg:
            return int(arg.replace("-exp_num=", ""))

def get_results_from_file(fbase):

    filename = fbase + "results_dict.pkl"
    with open(filename, "rb") as fp:
        return pickle.load(fp)

def create_index_log(cfg_handler):
    """Write index to file that describes experiment hyperparameters"""
```

```
    # Create directories
    fbase = "results/"
    if not os.path.exists(fbase):
        os.mkdir(fbase)
    batch_name = cfg_handler.get_experiment(0).save.experiment_batch_name
    fbase = "{}{}/".format(fbase, batch_name)
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    # Write index log to file
    with open('{}index.txt'.format(fbase), 'w') as f:
        f.write("Number of experiments: {}\n".format(cfg_handler.get_num_experiments()))
        json.dump(cfg_handler.get_options(), f)
        f.write("\n\n")

        for i in range(cfg_handler.get_num_experiments()):
            individual_option = cfg_handler.get_option(i)
            f.write("\tExperiment {}: ".format(i))
            json.dump(individual_option, f)
            f.write("\n")

def start_job(config_name, experiment_num, mode):
    """
    Starts a job for the fed arguments. This takes the form of a subprocess,
    whether on a normal computer or supercomputer
    """

    print("Starting job: {}".format(experiment_num))

    # Decide which script to run
    if "-s" in sys.argv:
        script_to_run = ["sbatch", "supercomputer_job.sh", "-s"]
    else:
        script_to_run = ["./standard_job.sh"]

    # Build script with hyperparameters
    full_command = [
        *script_to_run,
        "-job",
        "-cfg_name={}".format(config_name),
        "-exp_num={}".format(experiment_num)
    ]

    if mode == "train":
        full_command.append("-train")
    elif mode == "eval":
        full_command.append("-eval")
    else:
        raise Exception("Error: configuration mode must be either train or eval")


    # Run chosen script with correct arguments
    process = subprocess.Popen(full_command)

    # Wait if not parallel
    if "-p" not in sys.argv:
        process.wait()

def evaluate(model=None):
    """
    Evaluation only works on existing models and cached results. The model can be fed as
an argument,
    or reloaded through the configuration file. If reloaded, it will save results into
    the original experiment's directory.
    """

    # Get configuration values
    cfg_name = get_cfg_name()
    experiment_num = get_exp_num()
    cfg_handler = config_handler(cfg_name)
```

```
    # Might want to return evaluation_cfg, then load previous exp_cfg from file. This wor
ks for now though.
    filename, exp_cfg = cfg_handler.get_experiment(experiment_num)

    # Define fbase and create tree
    fbase = "results/"
    if not os.path.exists(fbase):
        os.mkdir(fbase)
    fbase += "{}/".format(exp_cfg.save.experiment_batch_name)
    if not os.path.exists(fbase):
        os.mkdir(fbase)
    fbase += "experiment_{}/".format(experiment_num)
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    # Create print mechanisms
    reset_log_files(fbase, exp_cfg.mode)
    redirect_stdout(fbase, exp_cfg.mode)
    redirect_stderr(fbase, exp_cfg.mode)

    # Load original configuration for this experiment and the associated results director
y
    revived_cfg = cfg_handler.get_cfg_from_file(fbase)
    results = get_results_from_file(fbase)

    # Load dataset
    dataset = cfg_handler.get_dataset(exp_cfg)

    # Load model
    if model is None:
        model = cfg_handler.get_model(input_size=dataset.get_input_size(), exp_cfg=exp_cf
g, filename=filename)
    model.summary()

    # Load evaluation functions
    evaluation_functions = cfg_handler.get_evaluation_functions(exp_cfg)

    # Use evaluation functions
    for evaluation_function in evaluation_functions:
        evaluation_function(dataset, model, exp_cfg, revived_cfg, results, filename)

def train():
    """Trains one ML model"""

    # Get configuration values
    cfg_name = get_cfg_name()
    experiment_num = get_exp_num()
    cfg_handler = config_handler(cfg_name)
    exp_cfg = cfg_handler.get_experiment(experiment_num)

    # Define fbase and create tree
    fbase = "results/"
    if not os.path.exists(fbase):
        os.mkdir(fbase)
    fbase += "{}/".format(exp_cfg.save.experiment_batch_name)
    if not os.path.exists(fbase):
        os.mkdir(fbase)
    fbase += "experiment_{}/".format(experiment_num)
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    # Create print mechanisms
    reset_log_files(fbase, exp_cfg.mode)
    redirect_stdout(fbase, exp_cfg.mode)
    redirect_stderr(fbase, exp_cfg.mode)

    # Cache configuration for future reload
    save_cfg(exp_cfg, fbase)

    # Print info
```

```
    print("Config name: {}".format(cfg_name))
    print("Experiment num: {}".format(experiment_num))
    print()

    # Load data
    dataset = cfg_handler.get_dataset(exp_cfg)
    data_dict = dataset.load_data()

    print("Train ins shape: {}".format(data_dict["train"]["ins"].shape))
    if type(data_dict["train"]["outs"]) == list:
        print("Train outs shape: {}".format(data_dict["train"]["outs"][0].shape))
        print("Train outs shape: {}".format(data_dict["train"]["outs"][1].shape))
    else:
        print("Train outs shape: {}".format(data_dict["train"]["outs"].shape))

    # Build model
    model = cfg_handler.get_model(
        input_size=dataset.get_input_size(),
        exp_cfg=exp_cfg)

    # Compile model
    model.compile(
        optimizer=Adam(learning_rate=exp_cfg.train.learning_rate),
        loss=cfg_handler.get_loss(exp_cfg.train.loss),
        metrics=exp_cfg.train.metrics)
    model.summary()

    # Callbacks
    callbacks = cfg_handler.get_callbacks(fbase, exp_cfg)

    # Future work
    # Check if train key is a dict. If so, do non-generator training. Else, do generator
training.
    # Check if val key exists. If not, don't use.
    #   Otherwise, check if val is a dict or generator and proceed accordingly
    # In total, 6 options.

    # Train model. Requirements for model.fit are liable to expand in the future, and wou
ld
    #   require further development. e.g. using sample or class weights, validation steps
, or validation frequency
    if "val" in data_dict.keys():
        history = model.fit(
                x=data_dict["train"]["ins"],
                y=data_dict["train"]["outs"],
                validation_data = (data_dict["val"]["ins"], data_dict["val"]["outs"]),
                epochs=exp_cfg.train.epochs,
                batch_size=exp_cfg.train.batch_size,
                callbacks=callbacks,
                verbose=exp_cfg.train.verbose
                )
    else:
        history = model.fit(
                x=data_dict["train"]["ins"],
                y=data_dict["train"]["outs"],
                epochs=exp_cfg.train.epochs,
                batch_size=exp_cfg.train.batch_size,
                callbacks=callbacks,
                verbose=exp_cfg.train.verbose
                )

    # Log results
    log_results(data_dict, model, exp_cfg, fbase)

def log_results(data, model, exp_cfg, fbase):
    """Log results to file"""

    print("Logging results")

    # Generate results
```

```
    results = {}
    results_brief = {}
    results["history"] = model.history.history
    if "train" in data.keys():
        results['predict_train'] = model.predict(data["train"]["ins"])
        results['eval_train'] = model.evaluate(data["train"]["ins"], data["train"]["outs"
])
        results_brief["eval_train"] = str(results["eval_train"])

    if "val" in data.keys():
        results['predict_val'] = model.predict(data["val"]["ins"])
        results['eval_val'] = model.evaluate(data["val"]["ins"], data["val"]["outs"])
        results_brief["eval_val"] = str(results["eval_val"])

    if "test" in data.keys():
        results['predict_test'] = model.predict(data["test"]["ins"])
        results['eval_test'] = model.evaluate(data["test"]["ins"], data["test"]["outs"])
        results_brief["eval_test"] = str(results["eval_test"])

    # Create model directory
    if not os.path.exists("{}/model_and_cfg/".format(fbase)):
        os.mkdir("{}/model_and_cfg/".format(fbase))

    # Save model
    model.save("{}/model_and_cfg/saved_weights.h5".format(fbase), save_format="tf")

    # Save brief results for human readability
    with open("{}results_brief.txt".format(fbase), "w") as f:
        f.write(json.dumps(results_brief))

    # Save full results binary
    with open("{}results_dict.pkl".format(fbase), "wb") as f:
        pickle.dump(results, f)

def save_cfg(exp_cfg, fbase):
    # File path
    filename = fbase + "model_and_cfg/"
    if not os.path.exists(filename):
        os.mkdir(filename)

    filename += "exp_cfg"
    with open(filename, "wt") as f:
        f.write(exp_cfg.dump())

def reset_log_files(fbase, mode):
    """Clears all logs files"""

    fbase += "logs/"
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    # File path
    with open("{}{}_out_log.txt".format(fbase, mode), "w") as f:
        pass

    with open("{}{}_err_log.txt".format(fbase, mode), "w") as f:
        pass

def redirect_stdout(fbase, mode):
    """Redirect stdout to file"""

    fbase += "logs/"
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    sys.stdout = open("{}{}_out_log.txt".format(fbase, mode), mode="a")

    print("--------Stdout Start--------")

def redirect_stderr(fbase, mode):
```

```
    """Redirect stderr to file"""

    fbase += "logs/"
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    sys.stderr = open("{}{}_err_log.txt".format(fbase, mode), mode="a")

    print("--------Stderr Start--------", file=sys.stderr)

if __name__ == "__main__":

    # If this is a subprocess, run the training program
    if "-job" in sys.argv:
        if "-train" in sys.argv:
            train()
        elif "-eval" in sys.argv:
            evaluate()
    else:
        main()
```