Replication Instructions for Configuration Performance Learning

This document provides step-by-step instructions to reproduce the experiments, metrics, and visualizations presented in the Configuration Performance Learning project.

🗀 Project Structure & File Descriptions

```
ConfigurationPerformanceLearning/
├── code/
│   ├── ann.py                  # Implements standard Artificial Neural Network model
│   ├── ann_tuned.py            # ANN with feature engineering and hyperparameter tuning
│   ├── feature_engineering.py    # Modular preprocessing: scaling, encoding, imputing
│   ├── linear_regression.py      # Baseline linear regression model
│   ├── random_forest.py          # Random Forest model without tuning
│   ├── random_forest_tuned.py    # Random Forest with feature engineering and hyperparameter tuning
│   ├── voting_regressor.py       # Voting Regressor using RF, XGBoost, and LR (basic)
│   ├── voting_regressor_tuned.py # Voting Regressor with feature engineering + tuning
│   ├── xgb_regressor.py          # Standard XGBoost model
│   └── xgb_regressor_tuned.py    # XGBoost with feature engineering and hyperparameter tuning
│
├── datasets/                   # Input datasets for each configurable system
│   ├── batlik/
│   ├── dconvert/
│   ├── h2/
│   ├── jump3r/
│   ├── kanzi/
│   ├── lrzip/
│   ├── x264/
│   ├── xz/
│   └── z3/
│
├── heatmaps/                   # Visual comparison of model performance
│   ├── average_mae_heatmap.png
│   ├── average_mape_heatmap.png
│   ├── average_rmse_heatmap.png
│   └── ttest_significance_heatmap.png
│
├── results/                    # CSV outputs for each model and statistical comparison
│   ├── ann.csv
```

```
│   ├── ann_tuned.csv
│   ├── linear_regression.csv
│   ├── random_forest.csv
│   ├── random_forest_tuned.csv
│   ├── voting_regressor.csv
│   ├── voting_regressor_tuned.csv
│   ├── xgb_regressor.csv
│   ├── xgb_regressor_tuned.csv
│   ├── Average MAE Comparison.csv
│   ├── Average MAPE Comparison.csv
│   ├── Average RMSE Comparison.csv
│   ├── ttest_results.csv         # Paired t-test results
│   └── stat_test.py              # Paired t-test statistical analysis across
models
│
├── requirements.txt              # Python library dependencies for setting up the
environment
├── manual.pdf                    # User guide explaining how to run and use the
system
├── requirements.pdf              # System requirements and specifications
(functional + non-functional)
├── replication.pdf               # Step-by-step instructions to reproduce results
and evaluations
```

⚙ Environment Setup

```
git clone https://github.com/your-username/ConfigurationPerformanceLearning.git
cd ConfigurationPerformanceLearning
pip install -r requirements.txt
```

🚀 Running Experiments

Run any model with:

```
python code/<script_name>.py
```

For example:

```
python code/xgb_regressor.py
python code/ann_tuned.py
```

📊 Outputs & Metrics

Each script runs 33 train-test splits and computes:

- MAPE (Mean Absolute Percentage Error)
- MAE (Mean Absolute Error)
- RMSE (Root Mean Squared Error)

My Test Results are saved in `results/` as:

- `xgb_regressor.csv`
- `ann_tuned.csv`
- `voting_regressor_tuned.csv`
- ...

📈 Visualizations & Comparisons

Pre-generated heatmaps are in the `heatmaps/` folder for:

- Average MAE, MAPE, RMSE per system
- Statistical significance (`ttest_significance_heatmap.png`)

You can re-run `stat_test.py` to regenerate `ttest_results.csv` or perform custom comparisons.

✅ Reproducibility Notes

- All scripts use fixed `random_state` for reproducibility.
- Tuning uses GridSearchCV on the first repeat.
- Feature engineering is modular and reusable via `feature_engineering.py`.

⚡ Optional Quick Test

To reduce runtime:

- Edit any script to use just 1 system:

  ```
  systems = ['h2']
  ```

- Reduce `num_repeats` from 33 to 5 for faster testing.

🧪 Running Statistical Comparison (Paired t-tests)

The file `stat_test.py` is used to **compare MAE scores** across models using paired t-tests.
It checks whether the difference in performance between models is statistically significant.

◇ Output

The script generates a file:

```
ttest_results.csv
```

This file includes:

- Model A
- Model B
- T-statistic
- P-value
- Whether the difference is statistically significant (`Yes` / `No`)

⚠ Important Setup Instructions (If Using Your Own Outputs)

By default, `stat_test.py` expects result CSV files to be located in the `results/` folder.

If your outputs are stored elsewhere (e.g. in `code/output/` after running model scripts), follow these steps:

☑ Step 1: Update File Paths

In `stat_test.py`, modify the `file_paths` dictionary to reflect your file locations.

Change this:

```
"XGBoost": "xgboost.csv"
```

To this:

```
"XGBoost": "../code/output/xgboost.csv"
```

Repeat this for all model entries.

☑ Step 2: Run the Script

From the `results/` directory:

```
python stat_test.py
```

This will save the t-test results in `ttest_results.csv` within the same folder.

This document ensures full traceability and repeatability of all models, outputs, and results in the project.