National College of Ireland

Configuration Manual

Research Practicum
Msc Data Analytics

Valesca Frey Pereira
Student ID: 24190284

School of Computing
National College of Ireland

Supervisor: Dr. Anderson Simiscuka

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Valesca Frey Pereira
Student ID:	24190284
Programme:	Msc Data Analytics
Year:	2025
Module:	Research Practicum
Supervisor:	Dr. Anderson Simiscuka
Submission Due Date:	20/12/2025
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	10th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Valesca Frey Pereira
24190284

1 Introduction

This configuration manual provides the full setup required to replicate the experiments conducted in the research project Classifying Autism Spectrum Disorder via brain connectivity and Graph Neural Networks.

The project uses fMRI data from the ABIDE dataset to construct functional brain graphs, process node features, and train Graph Attention Networks (GAT) and Graph Convolutional Networks (GCN) implemented in PyTorch Geometric.

The following aspects of the project setup are covered, with each section building on the previous one from the initial setup to the final deployment:

- System and software requirements;
- Environment setup;
- Dataset download and processing;
- Model Execution;
- Results Interpretations;

2 System Requirements

The systems requirements are minimal and allow easy replication.

2.1 Hardware Requirements

Table 1 sensitizes the minimum hardware requirements.

Component	Minimum Requirement
Processor	Apple M3 or intel
RAM	8 GB
Storage	20 GB free

Table 1: Minimum system specifications

2.2 Software Requirements

Table 2 sensitizes the software requirements.

Software / Tool	Version	Function
Python	3.10+	Core programming environment
PyTorch	2.0+	Deep learning framework
PyTorch Geometric	Latest	GNN layers (GAT, GCN)
NetworkX	Latest	Graph manipulation
NumPy / Pandas	Latest	Data processing
Nilearn	Latest	fMRI handling
Scikit-learn	Latest	Splits + metrics
Matplotlib / Seaborn	Latest	Visualisations
Jupyter Notebook	Latest	Running the code

Table 2: Software and tools required

3 Environment Setup

3.1 Python and Jupyter Notebook Installation

Below are the instructions for installing Python and the required software tool:

- Download and install the latest version of Python from the official website;
- Follow the installation instructions for your operating system;
- Once installed, install Jupyter Notebook via pip: pip install notebook

Once setup and installed, the Jupyter notebook can be launched by opening a terminal or command prompt and typing "Jupyter Notebook".

3.2 Libraries and Packages

Ensure the following Python libraries and packages are installed. Figure 1 shows the packages installing piece of code.

```
[ ]: pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu
     pip install torch-geometric
     pip install numpy pandas scikit-learn matplotlib seaborn
     pip install nilearn nibabel
```

Figure 1: install packages

```
[ ]: import os
     from dataclasses import dataclass

     import numpy as np
     import pandas as pd

     import torch
     import torch.nn as nn
     import torch.nn.functional as F

     from torch_geometric.data import Data
     from torch_geometric.loader import DataLoader
     from torch_geometric.nn import GATConv, global_mean_pool

     from sklearn.model_selection import StratifiedShuffleSplit, StratifiedKFold
     from sklearn.utils.class_weight import compute_class_weight
     from sklearn.metrics import (
         accuracy_score, f1_score, precision_score, recall_score,
         roc_auc_score
     )

     import matplotlib.pyplot as plt


     from nilearn import datasets
     import seaborn as sns
```

Figure 2: import libraries

4 Project Structure

The project follows a modular structure. The main folder "autism_research" contains all the essential notebooks to be run. The second file "other-tests" includes adjacents experiments for historical information. Figure 3 shows the project structure.

```
10 autism_research
11 |
12 |-- 00_download_data.ipynb
13 |-- 01_exploratory.ipynb
14 |-- 02_manifest.ipynb
15 |-- 03_build_graphs.ipynb
16 |-- 04_GENERAL-TRAIN_GAT.ipynb
17 |-- 05_GENERAL-TRAIN-GCN.ipynb
18 |-- 06_train_gat_men_only_STRUCTURED_160epochs.ipynb
19 |-- 07_train_gat_men_only_STRUCTURED_300epochs.ipynb
20 |-- 08_train_women.ipynb
21 |-- 09_train_men_downsized.ipynb
22
23 other-tests
24 | ...
```

Figure 3: Project Structure

5 Data Preparation

5.1 Data Source

The dataset was sourced from ABIDE I. The module for downloading the data includes:

- downloads fMRI data via the Nilearn fetcher;
- stores metadata in csv;
- filters subjects with valid scans;

- prepares paths used in later steps

```
[2]:
import os

class Config:
    data_root = "./abide_data"
    pipeline  = "cpac"   # processing pipeline
    atlas     = "aal"    # atlas used for time-series extraction
    out_root  = "./runs"

cfg = Config()
os.makedirs(cfg.data_root, exist_ok=True)
os.makedirs(cfg.out_root, exist_ok=True)

print("Data root folder:", cfg.data_root)
print("Outputs folder:", cfg.out_root)
```

Figure 4: Download phenotypes

```
[6]: import os, glob

     # 1. Download phenotypics
     pheno_file = os.path.join(cfg.data_root, "Phenotypic_V1_0b_preprocessed1.csv")
     if not os.path.exists(pheno_file):
         try:
             from urllib.request import urlretrieve
             url = "https://raw.githubusercontent.com/preprocessed-connectomes-project/abide/master/Phenotypi
             print("Downloading phenotypics file...")
             urlretrieve(url, pheno_file)
             print("Downloaded:", pheno_file)
         except Exception as e:
             print("Phenotypics download failed:", e)
     else:
         print("Phenotypics file already exists.")

     # 2. Download ABIDE preprocessed data (CPAC + AAL)
     data_path = os.path.join(cfg.data_root, "ABIDE_pcp", "cpac", "nofilt_noglobal")
     pattern = os.path.join(data_path, "*_rois_aal.1D")
     found = glob.glob(pattern)
     if len(found) > 10:
         print(f"Found {len(found)} AAL .1D files under {data_path}")
     else:
         print("Few or no AAL .1D files found. Attempting to fetch using Nilearn...")
         try:
             from nilearn.datasets import fetch_abide_pcp
             ds = fetch_abide_pcp(
                 data_dir=cfg.data_root,
                 pipeline=cfg.pipeline,
                 derivatives=['rois_aal'],
                 quality_checked=True,
                 verbose=1
             )
             print("Downloaded / verified ABIDE CPAC AAL data.")
         except Exception as e:
             print("FAILED to fetch ABIDE data automatically:", e)
```

Figure 5: Download preprocessed data

5.2 Data handling

Figure 5 shows the piece of code which handles the null and not interesting values. Subjectd missing sex information and with excessive head motion were disconsiderated.

5.3 Graph Construction

The graph construction module includes:

- loads each fMRI timeseries;
- applies Fisher-z transform;
- normalises each region time series;

```python
#  remove subjects missing sex, diagnosis or MeanFD > 0.2 ( excessive head motion)
reasons = []
keep = []
for _, row in manifest.iterrows():
    r = []
    if pd.isna(row.get("sex")) or pd.isna(row.get("dx")):
        r.append("missing_phenos")
    try:
        if row.get("fd") and float(row["fd"]) > cfg.fd_threshold:
            r.append("high_motion")
    except:
        pass
    reasons.append(";".join(r))
    keep.append(len(r) == 0)

manifest["qc_exclude_reason"] = reasons
manifest["keep"] = keep
```

Figure 6: handle missing data

- computes Pearson correlations (functional connectivity);
- keeps top-k strongest edges per node;
- builds PyTorch Geometric Data objects;
- saves.

```python
    # Torch tensors
    x_t = torch.tensor(X, dtype=torch.float32)
    ei_t = torch.tensor(edge_index, dtype=torch.long)
    ew_t = torch.tensor(edge_w, dtype=torch.float32)
    y_t  = torch.tensor([int(dx) if pd.notna(dx) else -1], dtype=torch.long)
```

Figure 7: Graph construction

6 Model Execution

6.1 SimpleGCN and SmallGATClasses

Figure 8 and 9 represents the GCN and GAT architectures, respectivelly.

6.2 Running experiments

Each experiment follows the same pipeline:

- Import graphs;
- Split into train/test;
- Build DataLoaders;
- Train model;
- Save metrics and bar charts.

```python
class SimpleGCN(nn.Module):
    """
    Simple 2-layer GCN:
    - conv1: in_dim -> hidden
    - conv2: hidden -> hidden
    - global mean pooling
    - linear layer to 2 classes (ASD vs Control)
    """
    def __init__(self, in_dim, hidden, dropout):
        super().__init__()
        self.conv1 = GCNConv(in_dim, hidden)
        self.conv2 = GCNConv(hidden, hidden)
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden, 2)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        batch = getattr(
            data, "batch",
            torch.zeros(x.size(0), dtype=torch.long, device=x.device)
        )

        # 1st GCN layer
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.dropout(x)

        # 2nd GCN layer
        x = self.conv2(x, edge_index)
        x = F.relu(x)
        x = self.dropout(x)

        # Global pooling over nodes → one vector per subject
        x = global_mean_pool(x, batch)

        # Final classification
        out = self.fc(x)
        return out
```

Figure 8: GCN Model

```python
class SmallGAT(nn.Module):
    def __init__(self, in_dim, hidden, heads, dropout):
        super().__init__()
        self.gat1 = GATConv(in_dim, hidden, heads=heads, concat=True, dropout=dropout)
        self.gat2 = GATConv(hidden * heads, hidden, heads=heads, concat=True, dropout=dropout)
        self.gat3 = GATConv(hidden * heads, hidden, heads=1, concat=False, dropout=dropout)
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden, 2)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        batch = getattr(
            data, "batch",
            torch.zeros(x.size(0), dtype=torch.long, device=x.device)
        )

        x = F.elu(self.gat1(x, edge_index))
        x = self.dropout(x)
        x = F.elu(self.gat2(x, edge_index))
        x = self.dropout(x)
        x = F.elu(self.gat3(x, edge_index))
        x = self.dropout(x)

        x = global_mean_pool(x, batch)
        out = self.fc(x)
        return out
```

Figure 9: GAT model

Five main setups for experiments were run.

- GCN full data split in 80% for training and 20% for test.
- GCN full data split in 80% for training and 20% for test.
- GAT men trained.
- GAT women trained.
- GAT balanced subset.