National College of Ireland

Evaluating Large Language Models for Sustainability-Aware Code Selection Using Pairwise Efficiency Comparison

MSc Research Project
Data Analytics

Cedric John Savio Pereira
Student ID: x23415975

School of Computing
National College of Ireland

Supervisor: Dr. Abdul Razzaq

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Cedric John Savio Pereira
Student ID:	x23415975
Programme:	Data Analytics
Year:	2025
Module:	MSc Research Project
Supervisor:	Dr. Abdul Razzaq
Submission Due Date:	11/12/2025
Project Title:	Evaluating Large Language Models for Sustainability-Aware Code Selection Using Pairwise Efficiency Comparison
Word Count:	567
Page Count:	8

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:	Cedric John Savio Pereira
Date:	11th 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):	

Evaluating Large Language Models for Sustainability-Aware Code Selection Using Pairwise Efficiency Comparison

Cedric John Savio Pereira
x23415975

1 Overview

This configuration manual provides the technical setup required to run the complete analytical workflow for the research project titled "Evaluating Large Language Models for Sustainability-Aware Code Selection Using Pairwise Efficiency Comparison". It detail's the hardware specifications, software installations, Python environment, dependencies, and the step-by-step execution of data preprocessing, downloading the models, prompt construction and evaluation.

2 Hardware and Software Requirements

2.1 Hardware Specifications

Every experiment was carried out with the following setup in a Google Colab Pro environment which provides Linux virtual machines running Ubuntu:

- Runtime Type: Python 3
- Hardware Accelerator: NVIDIA A100 GPU (80 GB VRAM) – High-RAM runtime
- System RAM: 167.1 GB, typical usage 14–30 GB depending on batch size
- GPU RAM: 80 GB, usage between 60–75 GB during peak inference
- Disk Allocation: 235.7 GB, with 76 GB utilised during processing
- Google Drive Storage: 100 GB, of which 85 GB was used for datasets, intermediate files, model weights, and evaluation outputs

2.2 Required Python Libraries

The following libraries were used in the final implementation:

- Data Processing
  - pandas, numpy — dataset loading, cleaning, and transformation

  - tqdm — progress monitoring
  - openpyxl — Excel file creation and multi-sheet writing
- Pair Generation and Metrics
  - scikit-learn — standardisation, classification metrics
  - scipy — Kendall- correlation, hierarchical clustering utilities
  - statsmodels — Fleiss' Kappa for inter-model agreement
  - matplotlib, seaborn — plotting evaluation heatmaps and histograms
- Large Language Models
  - transformers — HuggingFace model loading tokenizer execution
  - accelerate — device placement and inference optimisation
  - torch — tensor manipulation and GPU execution

2.3 Dataset

Project CodeNet, a massive benchmark with millions of competitive programming submissions in several programming languages, is the source of the dataset used in this study Puri et al. (2021). For every submission, CodeNet comprises:

- Source code
- Execution time (ms)
- Memory usage (KB)
- Code size (bytes)
- Task identifier
- Status (e.g., Accepted, TLE, MLE)
- Accuracy score

2.4 LLMs Configuration

Four open-sourced models were run locally within the GPU environment after being downloaded once from HuggingFace. No additional adjustment or fine-tuning was done.

- StarCoder2
- Qwen2.5-Coder
- DeepSeek-Coder-6.7B
- DeepSeek-Coder-1.3B

3 Implementation

3.1 Dataset Preparation

The dataset, after pre-processing has 2,989 faults with multiple legitimate contributions after cleaning. The vast range of submissions for each problem reflects the inherent diversity of CodeNet users. More than 50 languages are submitted; the most popular ones are C++, C, Python, and Java.

The pre-processing of the dataset can be seen in Figure 1

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
import pandas as pd
import glob

folder_path = '/content/drive/MyDrive/codet/'

excel_files = glob.glob(os.path.join(folder_path, '*.xlsx'))

print(f"Found {len(excel_files)} Excel files.")
```

```
Found 2989 Excel files.
```

```
import pandas as pd

for file_path in excel_files:
    try:

        df = pd.read_excel(file_path)
        print(f"Processing file: {os.path.basename(file_path)}")

        valid_statuses = [
            'Accepted',
            'Time Limit Exceeded',
            'Memory Limit Exceeded',
            'Output Limit Exceeded'
        ]

        df_filtered = df[df['Status'].isin(valid_statuses)]

        df_filtered = df_filtered[
            df_filtered['Accuracy'].notna() &
            (df_filtered['Accuracy'] != '0/1')
        ]

        df_clean = df_filtered.dropna(subset=['Memory', 'Exec Time'])

        df_clean = df_clean.drop(['SubmissionID', 'User'], axis=1)


        df_clean.to_excel(file_path, index=False)
        print(f"Saved processed data to: {file_path}")

    except Exception as e:
        print(f"Error processing {file_path}: {e}")
```

Figure 1: Pre-Processing CodeNet dataset

3.2 LLMs Processing

3.2.1 Download Models

As said in subsection 2.4, the 4 open-sourced models are to be downloaded to run them locally. This can be done by executing the code as shown in Figure 2.

```python
from transformers import AutoTokenizer, AutoModelForCausalLM
import os
from google.colab import drive

drive.mount('/content/drive')

HF_HOME = "/content/drive/MyDrive/hf_models"
os.makedirs(HF_HOME, exist_ok=True)

MODELS = [
    "bigcode/starcoder2-7b",
    "deepseek-ai/DeepSeek-Coder-6.7B-base"
    "Qwen/Qwen2.5-Coder-7B-Instruct"
    "deepseek-ai/deepseek-coder-1.3B-instruct"
]

for model_name in MODELS:
    print(f"\nDownloading {model_name} ...")
    local_path = os.path.join(HF_HOME, model_name.replace("/", "_"))
    os.makedirs(local_path, exist_ok=True)

    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name)

    tokenizer.save_pretrained(local_path)
    model.save_pretrained(local_path)

    print("Saved to:", local_path)

print("\nAll models downloaded & saved to Drive")
```

Figure 2: Code to download the models

We can confirm the models are downloaded and accessible to the notebook by executing this code as seen in Figure 3

3.2.2 Code Pair Generation

Code pairs (forward and reverse) are generated.

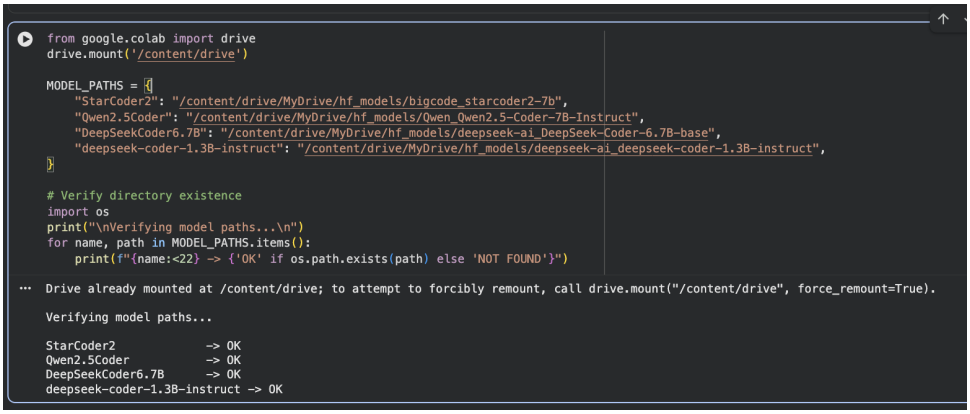
- Forward pair: code_i, code_j
- Reverse pair: code_j, code_i

The following implementation generates the pairs Figure 4

3.2.3 Prompt Construction

The generated pairs are then sent to the "clone_pair_evaluation" to build the prompt as seen in Figure 5 and Figure 6

The results obtained from the model are initially stored in an excel file.



```
from google.colab import drive
drive.mount('/content/drive')

MODEL_PATHS = {
    "StarCoder2": "/content/drive/MyDrive/hf_models/bigcode_starcoder2-7b",
    "Qwen2.5Coder": "/content/drive/MyDrive/hf_models/Qwen_Qwen2.5-Coder-7B-Instruct",
    "DeepSeekCoder6.7B": "/content/drive/MyDrive/hf_models/deepseek-ai_DeepSeek-Coder-6.7B-base",
    "deepseek-coder-1.3B-instruct": "/content/drive/MyDrive/hf_models/deepseek-ai_deepseek-coder-1.3B-instruct",
}

# Verify directory existence
import os
print("\nVerifying model paths...\n")
for name, path in MODEL_PATHS.items():
    print(f"{name:<22} -> {'OK' if os.path.exists(path) else 'NOT FOUND'}")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

Verifying model paths...

StarCoder2             -> OK
Qwen2.5Coder           -> OK
DeepSeekCoder6.7B      -> OK
deepseek-coder-1.3B-instruct -> OK
```

Figure 3: Downloaded models verification

```
for (i, j) in pairs:
    print("i and j.    ======.     ", i,j)
    code_i = df.loc[i]["Source code"]
    code_j = df.loc[j]["Source code"]

    # ---------------- Forward pair ----------------
    forward_row = clone_pair_evaluation(i, j, code_i, code_j)
    forward_row.update({
        "doc": excel_name,
        "row_i_exec_time": df.loc[i]["Exec Time"],
        "row_i_memory": df.loc[i]["Memory"],
        "row_i_code_size": df.loc[i]["Code Size"],
        "row_j_exec_time": df.loc[j]["Exec Time"],
        "row_j_memory": df.loc[j]["Memory"],
        "row_j_code_size": df.loc[j]["Code Size"]
    })
    master_df = pd.concat([master_df, pd.DataFrame([forward_row])],
                          ignore_index=True)
    # ---------------- Reverse pair ----------------
    reverse_row = clone_pair_evaluation(j, i, code_j, code_i)
    reverse_row.update({
        "doc": excel_name,
        "row_i_exec_time": df.loc[j]["Exec Time"],
        "row_i_memory": df.loc[j]["Memory"],
        "row_i_code_size": df.loc[j]["Code Size"],
        "row_j_exec_time": df.loc[i]["Exec Time"],
        "row_j_memory": df.loc[i]["Memory"],
        "row_j_code_size": df.loc[i]["Code Size"]
    })

    master_df = pd.concat([master_df, pd.DataFrame([reverse_row])],
                      ignore_index=True)
```

Figure 4: Pair Generation

```python
def clone_pair_evaluation(i, j, code_i, code_j):
    code_i = "" if code_i is None else str(code_i)
    code_j = "" if code_j is None else str(code_j)

    if len(code_i.strip()) == 0 or len(code_j.strip()) == 0:
        return {
            "row_i": i,
            "row_j": j,
            "code_i": code_i,
            "code_j": code_j,
            "StarCoder2": "",
            "Qwen2.5Coder": "",
            "DeepSeekCoder6.7B": "",
            "deepseek-coder-1.3B-instruct": "",
        }

    prompt = build_prompt(code_i, code_j)

    results = {
        "row_i": i,
        "row_j": j,
        "code_i": code_i,
        "code_j": code_j
    }

    for entry in MODELS_TO_RUN:
        alias = entry["alias"]
        path = entry["path"]

        # Load model once
        if alias not in MODEL_GENERATORS:
            MODEL_GENERATORS[alias] = load_model(path)

        gen = MODEL_GENERATORS[alias]

        # Evaluate pair
        result_letter = run_model_on_pair(gen, code_i, code_j)

        # Store in dict
        results[alias] = result_letter

    return results
```

Figure 5: clone Pair Evaluation

```python
def build_prompt(codeA, codeB):
    return f"""
You are given two code snippets, Code A and Code B, that both solve the same problem. Your task is to determine which code is more environmentally sustainable overall based on green computing principles.

Instructions

- Evaluate both codes independently and impartially.
- Do not favor one code based on formatting, style, syntax, language, or order (A or B).
- Provide only a single letter — A or B — as your answer, with no explanation or symbols.

 Code A

{codeA}

Code B

{codeB}

Answer:
"""
```

Figure 6: Prompt Builder

3.2.4 Normalize Predictions

In situations wherein the model returns, additional text, such as "Answer: A," "I choose A," and "Option A is better". The responses are normalized to guarantee comparability as seen in Figure 7

```python
def normalize_prediction(pred):
    """Return 'A' or 'B'. If invalid, mark as UNKNOWN."""
    if isinstance(pred, str) and len(pred) > 0:
        c = pred.strip()[0].upper()
        return c if c in ["A", "B"] else "UNKNOWN"
    return "UNKNOWN"
```

Figure 7: Normalize the Model's Responses

3.3 EGAP calculation

```python
def efficiency_score(exec_t, mem, size):
    """Compute total efficiency = lower is better."""
    return exec_t + mem + size

true_labels = []
egap_exec_time = []
egap_memory = []
egap_code_size = []
egap_total = []
# Storage for later problem-level EGAP
results["eff_i"] = np.nan
results["eff_j"] = np.nan

for idx, row in results.iterrows():

    #  Column-level EGAP (absolute difference)
    et = abs(row["row_i_exec_time"] - row["row_j_exec_time"])
    mem = abs(row["row_i_memory"] - row["row_j_memory"])
    size = abs(row["row_i_code_size"] - row["row_j_code_size"])

    egap_exec_time.append(et)
    egap_memory.append(mem)
    egap_code_size.append(size)

    # Compute efficiency per row (sum of metrics)
    Ei = efficiency_score(row["row_i_exec_time"], row["row_i_memory"], row["row_i_code_size"])
    Ej = efficiency_score(row["row_j_exec_time"], row["row_j_memory"], row["row_j_code_size"])

    # Store for problem-level EGAP use
    results.at[idx, "eff_i"] = Ei
    results.at[idx, "eff_j"] = Ej

    #  True label
    if Ei < Ej:
        forward_label = "A"
    elif Ej < Ei:
        forward_label = "B"
    else:
        forward_label = "A"  # tie-break default

    # Detect reversed pair: if row_i > row_j
    if row["row_i"] > row["row_j"]:
        # Flip the label (A ↔ B)
        forward_label = "B" if forward_label == "A" else "A"

    true_labels.append(forward_label)

    # Combined EGAP (absolute difference)
    egap_total.append(abs(Ei - Ej))
```

Figure 8: Calculating EGAP

4 Replication Steps

- Download the datasets and place in project directory.
- Start the python environment, connect to the hardware requirements subsection 2.1.
- Run the cells in notebook x23415975 - Code.ipynb sequentially.
- All plots, tables, and models will auto-generate.

References

Puri, R. et al. (2021). Project codenet: A large-scale ai dataset for code.