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:	5931
Page Count:	23

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

Abstract

The purpose of this study is to determine whether large language models (LLMs) can accurately determine which of two semantically comparable programs has a more energy-efficient implementation. We build pairwise comparisons and extract ground-truth labels from execution time, memory use, and code size using a filtered subset of Project CodeNet. A controlled A/B prompt is used to assess the performance of four code-oriented LLMs using accuracy, ranking metrics, and Energy Gap (EGAP). The findings demonstrate that while pairwise accuracy is close to chance for all models, agreement behavior and sensitivity to efficiency variations vary significantly. The results point out current shortcomings in the efficiency reasoning of LLMs and suggest ways to enhance sustainability-aware code analysis.

1 Introduction

The efficiency and sustainability of software systems have drawn more attention due to the expanding scope of contemporary computing. The energy used during program execution is a cost element as well as an environmental concern as applications grow more complicated and data-intensive. Large language models (LLMs) have created new opportunities for automated code generation and reasoning, but traditional optimization techniques—based on algorithmic advances or compiler-level enhancements—continue to play a crucial role. Much less is known about LLMs' capacity to make energy-conscious choices when confronted with multiple implementations of the same job, despite their considerable ability to produce functionally accurate programs.

This poses an important question: Can LLMs consistently determine which of two valid solutions is more energy-efficient? Unlike accuracy, efficiency is comparative in nature; identifying which code variant is more sustainable requires reasoning about execution time, memory usage, and program size. To evaluate LLMs' potential contribution to sustainable software engineering, it is essential to comprehend whether they are capable of making such comparable judgments.

Thus, the goal of this research is:

To evaluate whether large language models (LLMs) can reliably choose the more energy-efficient code variant between two implementations through pairwise comparison, using execution-based sustainability metrics (execution time, memory usage, and code size), and to construct a ranking framework that assesses LLM performance in sustainability-aware decision-making.

In order to achieve this goal, the study queries several cutting-edge LLMs using a controlled prompt, creates a sizable set of pairwise comparisons from real-world code submissions, and assesses their predictions against ground-truth efficiency labels obtained from execution metrics. After that, a multi-level evaluation pipeline is used, which includes pairwise accuracy measurements as well as an energy-penalty measure (EGAP) that measures the consequences of making the wrong choice. Lastly, a framework for aggregated ranking is created to compare models based on how well they conduct sustainability-aware reasoning.

This paper makes three contributions:

- An analysis of LLM decision-making on energy-efficiency comparisons using a scalable pairwise evaluation methodology.
- a sustainability-based evaluation approach that quantifies model flaws by integrating execution indicators with an energy-gap measure.
- a scoring system that combines energy-awareness and efficacy into a single model comparison.

This report's remaining sections are organized as follows.

Section 2 summarizes relevant research in LLM-based reasoning, pairwise code evaluation, energy-efficient software analysis, and sustainable computing. The approach, which includes dataset creation, ground-truth generation, the LLM assessment process, and statistical analysis techniques, is presented in Section 3. The architectural requirements and system design are described in depth in Section 4. The tools and implementation environment are described in Section 5. The evaluation results are presented and discussed in Section 6. Section 7 wraps up the paper and outlines potential avenues for future research.

2 Related Work

Research on energy-aware computing, sustainable software, and large language model (LLM) behavior encompasses a number of related fields, each of which offers some understanding of how efficiency might be quantified, anticipated, or enhanced. However, the main topic investigated in this study—whether LLMs can consistently select the more energy-efficient implementation when given two competing code variants—is not directly addressed by any body of prior research. This section is divided into two sections to place the suggested methodology in the larger scientific context. The research gap that drives our approach is shown by a comparative table Table 1 that summarizes how previous work matches with the particular criteria pertinent to this study.

2.1 Foundational Definitions, Taxonomy, and Nomenclature

The design of computer systems that reduce resource consumption, mainly electrical energy, while preserving functional correctness and performance is known as energy-efficient computing. According to recent evaluations, energy efficiency needs to be addressed at the algorithmic, software, and hardware layers in order to limit the impact on the environment Pazienza et al. (2024). This presents energy efficiency as a more general sustainability criterion as well as a technical optimization objective.

A key component of this viewpoint is software-level energy consumption. According to fundamental measurement studies, algorithmic complexity, execution time, and memory access behavior all influence runtime energy consumption Noureddine et al. (2013). These results demonstrate that software design choices have a significant impact on energy expenditures, highlighting the necessity of accurate and reliable measuring techniques.

By including efficiency and environmental factors into the development lifecycle, sustainable software engineering expands on these issues. Energy-related characteristics, such as resource consumption, model complexity, and inference cost, should be considered first-class quality factors in addition to accuracy and performance, according to work on green artificial intelligence Verdecchia et al. (2023). This brings sustainability-focused decision-making into line with conventional engineering practice.

Metrics like execution time, memory use, and code size are commonly used in empirical investigations to operationalize software efficiency. Code size is linked to maintainability and possible overhead, memory usage indicates resource intensity, and execution time records computational workload. According to Noureddine et al. (2013), these measures are still essential for comparing implementations and describing program behavior.

Recently, large language models (LLMs) have become effective tools for automated reasoning and program synthesis. Studies looking at the environmental features of LLM-generated code, however, reveal that while models frequently yield functionally correct solutions, they do not always produce energy-efficient implementations, especially when deeper algorithmic optimization is needed Mehta et al. (2023) Wang et al. (2018). This encourages research that assesses LLM decision-making by considering efficiency rather than just accuracy.

It is closely connected to pairwise code evaluation, which examines two semantically comparable code pieces to see which is more efficient. These pairwise frameworks are used to analyze subtle variations between implementations in clone identification and code-quality evaluation. The difficulty of comparing programs with similar semantics but significant syntactic difference is highlighted by research on cross-lingual clone identification, which supports the usefulness of structured comparison techniques Wang et al. (2018).

The taxonomy for characterizing similarity links between code fragments is provided by the more general field of code clone detection. Four types of clones are distinguished in the literature:

- Type 1: identical copies with the exception of formatting or comments.
- Type 2: identifiers that have been renamed in syntactically comparable pieces.
- Type 3: copied passages that have statements added, deleted, or changed.
- Type 4: implementations having distinct structural forms that are semantically similar.

Lastly, the methodological foundation for comparing LLM performance across several aspects is provided by ranking and decision-making models. In order to evaluate reasoning quality, accuracy, and efficiency, multidimensional LLM benchmarks are increasingly using ranking-based techniques Yan et al. (2024). This provides a theoretical basis for assessing LLM behavior in sustainability-oriented paired code selection.

2.2 Review of Existing Studies and Identification of Research Gaps

Although research in the nexus of sustainable software engineering, energy-efficient computing, and large language models (LLMs) has expanded quickly, it is still dispersed in ways that directly support the goal of this study. The current body of work may be broadly categorized into three threads: (i) computational energy consumption measurement and optimization; (ii) LLM-driven code production and its sustainability implications; and (iii) frameworks for code comparison, clone identification, and model evaluation. None directly assesses whether LLMs can consistently choose the most energy-efficient code variant in a paired decision-making setting, nor do they provide a unified ranking framework for sustainability-aware reasoning, despite the fact that each provides insightful information.

2.2.1 Energy and Sustainability in Software Systems

The significance of measuring energy usage across hardware, workloads, and software behaviors has been established by foundational research. In their systematic assessment of energy measurement methods for software systems, Noureddine et al. (2013) found significant differences between measurement instruments as well as methodological flaws. Pazienza et al. (2024) expanded on this perspective by putting out a comprehensive paradigm that integrates computational, architectural, and behavioral elements to place software energy consumption within the broader objectives of ecologically sustainable computing. These studies highlight the multifaceted nature of energy efficiency, which is influenced by memory access patterns, algorithmic structure, code-level features, and raw execution cost.

Energy concerns in AI systems itself are the subject of more recent research. In their analysis of energy consumption during large language model (LLM) inference, Argerich and Patiño-Martínez (2024) emphasize inefficiencies resulting from input design, hardware choices, and model size. Complementary studies by Mehta et al. (2023) and Castellanos-Nieves and García-Forte (2023) summarize new "Green AI" approaches and promote the use of energy-conscious training and inference pipelines, lightweight model architectures, and efficiency-oriented metrics. Although these contributions aid in defining the sustainability problem space, neither code-level sustainability decisions nor models' capacity to distinguish between more and less efficient program variations are examined.

2.2.2 LLM-Based Code Generation and Sustainable Code

The growing usage of LLMs in software development workflows has recently drawn attention to the sustainability of generated code. In their thorough systematic assessment of Green AI techniques, Verdecchia et al. (2023) found that efficiency issues with LLM-generated software are still mostly unexplored. In their assessment of LLM-driven code generation and evaluation methods, Wang and Chen (2023) observe that efficiency-oriented elements are largely overlooked in favor of performance and accuracy as evaluation criteria.

The effect of LLMs on computing efficiency has started to be investigated empirically. When analyzed using runtime-based criteria, the study by Shi et al. (2024) shows that LLM-generated solutions frequently show inferior efficiency than human-produced

code. This line of research is strengthened by two recent works: Tuttle et al. (2024) and Solovyeva et al. (2025) both investigate prospects for energy optimization through guided code synthesis. While the latter suggests learning-based techniques to encourage LLMs toward more effective implementations, the former demonstrates that contemporary LLMs frequently favor syntactically straightforward but computationally costly structures. Nevertheless, these experiments do not assess whether an LLM can identify which of two supplied code snippets is more efficient; instead, they concentrate on creation rather than decision-making.

The current study revolves around this gap. The literature currently in publication examines "Can LLMs generate efficient code?" but not "Can LLMs identify the energy-efficient solution among alternatives?" This distinction is important because sustainable automated programming assistants require energy-aware choosing.

2.2.3 Code Comparison, Clone Detection, and Pairwise Analysis

While code comparison research offers methodological support, sustainability rationale is not directly addressed. A number of studies, such as Wang et al. (2018) and the current study by Moumoula et al. (2025), concentrate on cross-language code clone detection. In order to determine if various implementations carry out the same operation, these studies investigate token-level, syntactic, and semantic similarity among programs. They do not use execution-based efficiency metrics or assess the models' capacity to distinguish between efficient and inefficient clones, despite being theoretically related because clone detection also entails pairwise relational reasoning.

Pairwise assessment has been used in automated code repair and program synthesis, but mostly for accuracy rather than sustainability. No previous study attempts to align such pairs decisions with LLM predictions, nor does it create a systematic pairwise efficiency evaluation benchmark incorporating runtime, memory utilization, or code size.

2.2.4 LLM Evaluation, Ranking, and Decision-Making Frameworks

LLM evaluation frameworks have made significant progress, especially in the areas of multi-dimensional benchmarking, reasoning, and code comprehension. Yan et al. (2024) present CodeScope, a multilingual benchmark that evaluates code generation, test-case reasoning, and execution-based code understanding. An organized examination of model performance across generating tasks is also provided by Wang and Chen (2023). Although these works provide advanced evaluation techniques, the assessment does not include sustainability metrics.

Actually, no benchmarking or rating system now in use assesses LLMs' capacity to select the more energy-efficient program when given a choice. In order to close this gap, the current study introduces:

- A formulation of paired decisions based on quantifiable sustainability measures (execution time, memory use, code size).
- a methodical process that uses actual execution data from numerous problem situations to determine ground truth.
- LLM sustainability-aware reasoning performance is measured using a ranking system that combines accuracy, pairwise correctness, and EGAP-based efficiency penalties.

2.2.5 Synthesis and Identified Research Gap

The literature shows that awareness of sustainability in software systems and LLM-generated artifacts is growing throughout various streams. But no earlier work:

- assesses the capacity of LLMs to choose the more effective code variation in a controlled pairwise-comparison environment,
- integrates execution-derived efficiency gaps (EGAP) into a framework for evaluating models, or
- creates a comparative score of LLMs according to how well they make sustainable decisions.

The main contribution of the current study is an unexplored intersection—energy-aware paired reasoning.

Paper	Code efficiency metrics	Software energy measurement	Energy-efficient code generation	Pairwise code comparison	Code clone detection / similarity	LLM-based code reasoning	LLM sustainability evaluation	LLM ranking / benchmarking	Direct sustainability-aware pairwise decision-making
A Review of Energy Measurement Approaches	∼	✓	✗	✗	✗	✗	✗	✗	✗
Measuring & Improving Energy Efficiency of LLM Inference	∼	✓	✗	✗	✗	✗	✓	✗	✗
Review for Green Energy ML and AI Services	∼	✓	✗	✗	✗	✗	✓	✗	✗
Green Computing	✗	✓	✗	✗	✗	✗	✗	✗	✗
Can LLMs Generate Green Code?	✓	∼	✓	∼	✗	✓	✓	∼	✗
AI-Powered but Power-Hungry	✓	∼	✓	✗	✗	✓	✓	✗	✗
Green-Code	✓	∼	✓	✗	✗	✓	✓	✗	✗
The Struggles of LLMs in Cross-Lingual Code Clone Detection	✗	✗	✗	∼	✓	∼	✗	✗	✗
CodeScope Benchmark	✓	✗	✗	∼	∼	✓	✗	✓	✗

Table 1: Paper-category mapping table.

3 Methodology

The methodological approach for answering the research question, "Can large language models (LLMs) reliably identify the more energy-efficient implementation between two functionally equivalent solutions?" is described in this chapter. The study employs a controlled experimental methodology, progressing methodically from the creation of the dataset to paired evaluation and ranking that takes sustainability into account.

3.1 Overview and Experimental Tasks

This study's goal is:

To evaluate whether large language models (LLMs) can reliably choose the more energy-efficient implementation between two solutions through pairwise comparison, using execution-based sustainability metrics, and to rank models based on sustainability-aware decision-making.

Inspired by pairwise preference-learning frameworks like RankNet Burges et al. (2005), the methodological approach Figure 1 adheres to an organized sequence. Every stage is planned for reproducibility, bias control, and openness.

Figure 1

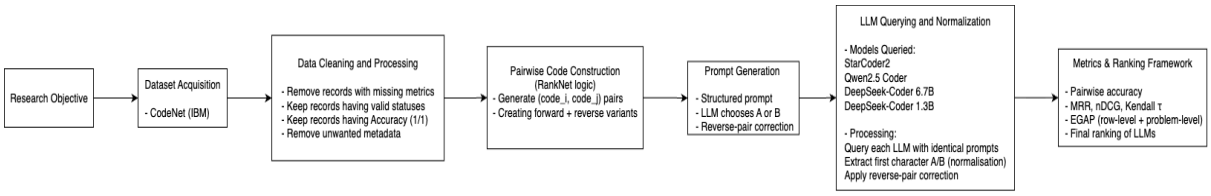

Figure 1: Methodology Pipeline

The procedure consists of five experimental tasks to accomplish the research goal:

- **Dataset Acquisition:** The main source of data is Project CodeNet Puri et al. (2021). It offers the comprehensive execution metadata needed for sustainability research in addition to millions of competitive programming submissions.
- **Data Cleaning and Preprocessing:** Only valid statuses are kept, submissions without execution metrics are eliminated, and accuracy is maintained by retaining only samples that achieve Accuracy 1/1.
- **Pairwise Code Construction:** Ordered comparison pairs are created from submissions for the same problem. In line with pairwise comparison practice, both forward and backward versions are generated to avoid positioning bias in LLM reasoning Burges et al. (2005)
- **Prompt Generation and LLM Querying:** The LLM is instructed by a controlled prompt to select only "A" or "B." To guarantee uniform interpretation, predictions are normalized and corrected for reversal.
- **Metrics and Ranking framework:** Correctness criteria, ranking metrics, and energy-gap measures are used to score LLM outputs Noureddine et al. (2013) and Mehta et al. (2023). The sustainability-aware model rating is made up of all of these.

Strong methodological validity is ensured by this organized pipeline while preserving the results' interpretability and generalizability.

3.2 Dataset Construction and Characteristics

3.2.1 Dataset Source – Project CodeNet

Project CodeNet, a benchmark of more than 14 million contributions written in more than 50 programming languages, is used in the study (Puri et al., 2021). Every contribution consists of:

- Execution time (ms)
- Memory usage (KB)
- Code size (byte)
- Program status
- Source code

These attributes provide direct proxies for computational resource consumption, a central consideration in sustainable software evaluation Noureddine et al. (2013) and Mehta et al. (2023).

3.2.2 Data Cleaning and Filtering Procedure

The dataset is standardized by a multi-stage process:

- **Elimination of unfinished records:** Submissions with insufficient memory, code size, or execution time are rejected.
- **Maintaining valid execution statuses:** Accepted, Time Limit Exceeded, Memory Limit Exceeded, and Output Limit Exceeded are the only four statuses that are retained. Under CodeNet limitations, these indicate meaningful or successful executions Puri et al. (2021)
- **Requirement for semantic correctness:** To guarantee that every variation correctly completes the underlying goal, only solutions with Accuracy = 1/1 are kept.
- **Preserving key fields:** Task ID, source code, execution time, memory, code size, status, and language are the only fields that are kept for future study.

Crucially, every programming language is kept. This preserves ecological validity in cross-linguistic comparison while reflecting actual LLM usage.

3.2.3 Dataset Characteristics

The dataset 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.

Because issues vary in difficulty, implementations vary in strategy, and languages differ in execution behavior, this linguistic and structural diversity enhances generalizability. A strong foundation for assessing sustainability-aware reasoning across various programming paradigms is provided by the heterogeneity.

3.3 Pairwise Code Comparison and Ground-Truth Derivation

The following step creates the ground-truth efficiency labels and transforms submissions into pairwise comparison instances after the dataset has been prepared.

3.3.1 Pairwise Instance Generation

There are usually several legitimate entries for each problem, each with a different algorithmic design and resource behavior. These are converted into pairs that are ordered:

- `code_i` — the first implementation
- `code_j` — the second implementation

This structure is similar to pairwise-preference setups that are frequently used for ranking and comparative assessment Burges et al. (2005)

To remove positional bias, forward and reverse versions are generated for every unordered pair. Consistent labeling across these variants is ensured by later correction rules.

3.3.2 Execution-Based Efficiency Metrics

Three execution-based dimensions are used to assess each submission Noureddine et al. (2013) and Mehta et al. (2023):

- Execution time (ms) – CPU workload
- Memory usage (KB) – RAM consumped
- Code size (bytes) – associated wih maintainability and loading overhead

The definition of a composite efficiency score is:

$$E = \text{Exec Time} + \text{Memory} + \text{Code Size}.$$

A more energy-efficient approach is indicated by lower ratings.

This straightforward additive score is consistent with accepted sustainability-analysis practices and avoids model-dependent weighting assumptions.

3.3.3 Ground-Truth Label Assignment

Efficiency scores Ei and Ej identify the actual preference for each pair:

- A, if $E_i < E_j$
- B, if $E_j < E_i$
- A, if tied (deterministic tie-break)

The label is automatically inverted if the pair appears in reverse order (for example, the stored pair is j,i rather than i,j). This guarantees the semantic equivalency of forward and reverse variations.

3.3.4 Energy Gap (EGAP) Computation

The Energy Gap is calculated to determine how significant each choice is:

$$\text{EGAP} = |E_i - E_j|$$

Decisions with significant efficiency differences are shown by large EGAP values, while near-equivalent implementations are represented by tiny values. According to Manning et al. (2008), this supports weighted evaluation, which is comparable to graded relevance metrics in information retrieval.

3.4 LLM Evaluation Procedure

This step investigates whether LLMs can consistently identify the most effective implementation. Procedures eliminate positional bias and guarantee uniformity.

3.4.1 Models Evaluated

Four cutting-edge code-oriented LLMs are evaluated:

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

A comparative evaluation of sustainability-aware reasoning is made possible by the differences between these models' architectures and training sets.

3.4.2 Prompt Construction

Every model uses the same, neutral prompt. It is

- identifies implementations as A and B,
- displays both in the same format, and
- gives the model instructions to just respond "A" or "B," without providing an explanation.

Metadata and execution metrics are not shared. This separates the capacity of each model to decipher efficiency cues based only on code structure Yan et al. (2024).

3.4.3 Prediction Normalisation

LLM answers differ (e.g., "Answer: B," "I choose A"). A straightforward rule extracts:

- extract the first alphabetical character,
- convert to uppercase,
- map to A, B, or UNKNOWN.

3.4.4 Reverse-Pair Correction

To preserve semantic consistency across variations, the model's prediction is inverted (A $\leftrightarrow$ B) if a pair is found to be reversed.

3.4.5 Output Preparation for Evaluation

Every model generates the following for every pair:

- a final label (A/B),
- a sign of correctness,
- an an accuracy score weighted by EGAP.

The evaluation measures that follow are derived from these results.

3.5 Evaluation Metrics and Statistical Framework

The last phase evaluates sustainability awareness and forecast accuracy among models.

3.5.1 Effectiveness Metrics

- Accuracy is the percentage of accurate forecasts.
- F1, Precision, and Recall treat A as a positive class.
- Kendall's τ—the relationship between ground-truth and model orderings

These show pair-level accuracy.

3.5.2 Ranking Metrics

Taking a cue from Manning et al. (2008) evaluation:

- Correct top-rank picks are rewarded by MRR (Mean Reciprocal Rank).
- nDCG (Normalised Discounted Cumulative Gain) rewards accurate choices on high-impact pairs by using EGAP as graded relevance.
- ERR (Expected Reciprocal Rank) represents predicted utility under uncertainty.

3.5.3 Energy-Based Metrics

Row-Level EGAP — absolute efficiency difference per pair. Problem-Level EGAP - the difference between the optimal implementation selected by the model and the actual most efficient one. These quantify how model flaws affect sustainability.

3.5.4 Final LLM Ranking Framework

All evaluation dimensions are combined into a weighted score:

$$\text{Final Score} = 0.30 \cdot \text{Accuracy} + 0.20 \cdot \tau + 0.20 \cdot \text{MRR} + 0.20 \cdot \text{nDCG} + 0.10 \cdot \text{Adjusted EGAP}$$

4 Design Specification

The architectural design, processing workflow, and methodological decisions supporting the sustainability-aware evaluation system are presented in this part. Pairwise efficiency comparison, direction-corrected model choice analysis, and multi-level sustainability rating are all supported by the design's modular, extendable framework. The overall architecture of the system is shown in Figure 2.

Figure 2

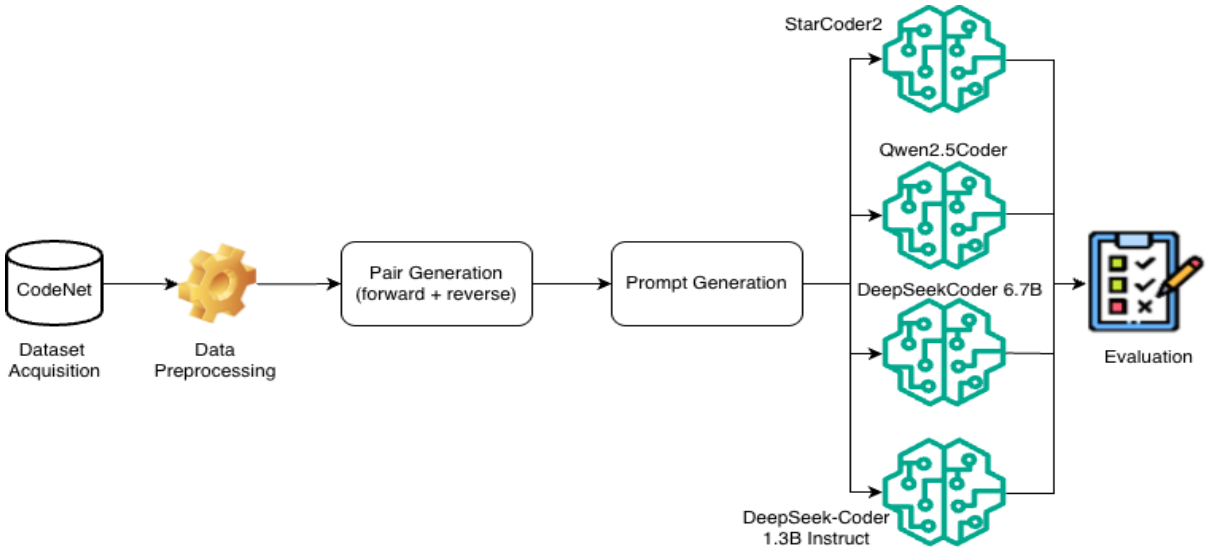

Figure 2: System Architecture

4.1 System Architecture Overview

An end-to-end pipeline is implemented by the system architecture (Figure 2), which starts with dataset acquisition and moves on to structured pair generation, unified prompt construction, LLM inference, output normalization, evaluation, and final ranking. Four architectural principles are followed in the design:

- **Ground-truth sustainability derived from CodeNet metrics**
  The Project CodeNet dataset's execution time, memory utilization, and code size are objective measures of efficiency (Puri et al. 2021). The system does not execute code contributions locally because these metrics are provided by the dataset.

- **Pairwise comparison as a controlled evaluation setting**
  Two semantically comparable code implementations make up each task instance. To prevent positional ordering from skewing LLM predictions, both forward and reverse versions of each pair are produced. In pairwise decision-making investigations, this guarantees methodological validity.

- **Separation of concerns through modular components**
  Scalability and controlled model experimentation are made possible by specialized preprocessing, pair creation, rapid construction, inference, normalization, and evaluation modules. Because the system is modular, it can be expanded in the future

to include more LLMs, different efficiency metrics (such energy-per-instruction), or automated feature extraction without requiring modifications to the main pipeline.

- **Multi-level evaluation for sustainability-aware ranking**
  A comprehensive evaluation of an LLM's capacity to choose energy-efficient code is provided by pair-level accuracy, problem-level EGAP, and ranking metrics.

4.2 Dataset Acquisition and Preprocessing

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. 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

4.2.1 Dataset Filtering and Cleaning

The following preprocessing procedures were used to guarantee the accuracy and comparability of ground-truth labels:

- **Removal of incomplete records:** Submissions that lacked code size, memory consumption, or execution time were rejected.
- **Filtering by status:** The following statuses were the only ones kept: Accepted, Time Limit Exceeded (TLE), Memory Limit Exceeded (MLE), and Output Limit Exceeded (OLE). These classifications stand for legitimate CodeNet execution outputs.
- **Correctness constraint:** To guarantee that all code variations accurately carry out the underlying algorithmic objective, only submissions with accuracy "1/1" were retained.
- **Excess metadata removal:** Task, language, status, execution time, memory, code size, accuracy, and source code were the only fields left in the dataset that were necessary for this investigation.

A clean, multilingual dataset appropriate for pairwise comparison was generated by this procedure.

4.3 Pairwise Code Construction

For every problem, a pairwise comparison set was created in order to assess if LLMs are capable of differentiating between more effective implementations of the same solution:

4.3.1 Pair Generation Strategy

For each task:

- unique pairs (code_i, code_j) were generated.
- Two variations were made for each pair:
  - Forward pair: A = code_i, B = code_j
  - Reverse pair: A = code_j, B = code_i

In LLM decision making, this guarantees robustness against positional prejudice.

4.3.2 Ground-Truth Efficiency Calculation

A sustainability score is assigned to each submission:

$$E = \text{Exec Time} + \text{Memory} + \text{Code Size}.$$

Higher efficiency is indicated by lower values. For every pair, the true label is:

- A, if $E_i < E_j$
- B, if $E_j < E_i$

The label is switched appropriately for couples that are reversed.

4.4 Prompt Generation and LLM Querying

For controlled comparison, prompt construction must be of the highest caliber. A consistent, fixed prompt template is used in the study:

> 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:

Four open-sourced LLMs designed for code or reasoning were used:

- StarCoder2

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

Local Inference Execution

To guarantee reproducibility and control over model settings, all inference was handled on local GPU hardware rather than using API-based model execution. Every pair was supplied as an independent inference question. To enforce deterministic behavior, sampling variance and temperature were turned off.

4.5 Prediction Normalisation and Reversal Correction

Additional text, such as "Answer: A," "I choose A," and "Option A is better," is frequently included in LLM results. Every response is normalized to guarantee comparability:

```
normalize_prediction(pred):
    return first character in {A, B} else UNKNOWN
```

4.5.1 Reversal Correction

The model prediction is flipped if a pair is inverted (row_i > row_j in the original dataset):

- If LLM predicts A → becomes B
- If LLM predicts B → becomes A

This guarantees that judgments made in forward and reverse pairings make sense.

4.6 Evaluation Metrics

4.6.1 Pairwise Metrics

- Accuracy
- Precision, Recall, F1
- Kendall-τ correlation

These assess the accuracy and coherence of pairwise judgments.

4.6.2 Problem-Level EGAP

The efficiency penalty that results from a model choosing a less-than-ideal implementation is measured by EGAP:

$$\text{EGAP} = E_{\text{model-best}} - E_{\text{true-best}}$$

Better sustainability-aware judgment is indicated by lower EGAP.

4.6.3 Ranking Framework

A final ranking score combines the following:

- Pairwise accuracy
- Kendall-τ
- Average EGAP across all problems
- Normalised weighted scoring

This results in a ranking of all four LLMs with an emphasis on sustainability.

4.7 Summary

A thorough, expandable, and empirically supported pipeline for determining whether LLMs can consistently select the more energy-efficient implementation is established by the design specification mentioned above. The design guarantees:

- Clean and reproducible dataset preparation
- Controlled pairwise evaluation
- Deterministic prompting and inference
- Strong direction correction and normalization
- Evaluation of sustainability using multiple metrics
- Correctness and energy-efficiency impact are the two factors that determine model ranking.

The goal of the study is fully supported by this architecture, which also serves as a foundation for potential future extensions like more models, automated explainability, or more sustainability metrics.

5 Implementation

The practical application of the sustainability-aware LLM evaluation system is discussed in this section. The system architecture is directly followed by the implementation, which uses a modular, repeatable, and computationally efficient workflow to realize each component, from dataset preparation and pair formation to LLM querying, assessment, and ranking. The system uses API-based model interactions and structured data transformation pipelines, and it was fully implemented in Python and run in the Google Colab environment. Transformed pairwise datasets, LLM decision logs, assessment metrics, and ranking tables based on sustainability are among the final products.

5.1 System Configuration

5.1.1 Hardware Specifications

Every experiment was carried out with the following setup in a Google Colab Pro environment:

- 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

The A100 arrangement greatly lowers latency, particularly when assessing tens of thousands of prompt-response pairs, even though GPU acceleration is not necessarily necessary for LLM inference in this study. Large dataframes (0.5M+ rows) are handled smoothly during pair formation and assessment because of the high RAM configuration.

5.1.2 Operating System

Every component was run on Linux virtual machines running Ubuntu that were made available by Google's Colab environment. This offers a controlled environment for installing dependencies, consistent package versions, and CUDA support for GPU execution.

5.2 Development Configuration

5.2.1 Programming Languages and Tools

It makes use of

- Python 3.10 — primary development language
- Google Colab Notebook environment — interactive execution, dataset processing, LLM querying, and result visualisation
- HuggingFace Transformers Model Hub — only used for downloading the LLM checkpoints, not for training
- Local inference runtime — once downloaded, all models (StarCoder2, Qwen2.5Coder, DeepSeekCoder-6.7B, DeepSeek-1.3B-Instruct) were executed within the Colab environment without external API dependence

By doing this, variability that can result from dynamic model updates, rate limitations, or remote API latency is eliminated and reproducibility is guaranteed.

5.2.2 Libraries and Dependencies

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-τ 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

5.3 LLM Inference Configuration

5.3.1 Model Suite

Four models were run locally within the GPU environment after being downloaded once from HuggingFace, subsection 4.4. No additional adjustment or fine-tuning was done, guaranteeing that the outcomes accurately represent each model's capacity for zero-shot reasoning.

5.3.2 Output Generation

The implementation produces several structured outputs:

- results_with_egap.xlsx
  - Contains all per-row metrics
  - Includes normalized predictions, correctness flags, EGAP scores
- evaluation_summary.xlsx
  - Per-model performance sheets
  - Agreement matrices
  - Ranking metrics (MRR, nDCG)
  - Final ranking table
- Plots

  - Model agreement heatmap
  - Disagreement histogram
  - Hierarchical clustering dendrogram
- Problem-Level EGAP Sheet
  - Quantifies sustainability penalty per problem per model
  - Supports ranking based on holistic efficiency-related behaviour

The evaluation and discussion chapters immediately benefit from these outcomes.

6 Evaluation

The empirical findings from the paired comparison trials are shown in this section. Section 6.5 provides complete interpretation; just numerical results are presented here. To assess accuracy, agreement behavior, ranking performance, and sustainability impact across the four LLMs, four experiments were carried out.

6.1 Pairwise Prediction Behaviour

The distribution of model outputs for each of the 126,889 couples is summarized in Table 2. Counts of forecasts for labels A and B as well as unknown outputs are included in the results.

Model	**A**	**B**	**Unknown**	**Total**
StarCoder2	63,838	62,622	429	126,889
Qwen2.5Coder	62,621	64,220	48	126,889
DeepSeekCoder-6.7B	60,973	64,218	1,698	126,889
DeepSeek-Coder-1.3B-Instruct	63,488	63,394	7	126,889

Table 2: Model prediction distribution across labels A, B, and Unknown.

Table 3 displays pairwise accuracy measures.

Model	**Accuracy**	**Precision**	**Recall**	**F1**	**Kendall-τ**
StarCoder2	0.5016	0.4855	0.5067	0.4959	0.0036
Qwen2.5Coder	0.5148	0.4981	0.5087	0.5033	0.0291
DeepSeekCoder-6.7B	0.4917	0.4758	0.4781	0.4769	-0.0174
DeepSeek-Coder-1.3B-Instruct	0.5059	0.4893	0.5065	0.4978	0.0118

Table 3: Model performance comparison across standard evaluation metrics.

6.2 Model Agreement and Disagreement

The agreement heatmap between model predictions is shown in Figure 3. The percentage of pairings for which models generated identical outputs is represented by agreement values. The distribution of the number of models that disagree on each pair is shown in Table 4.

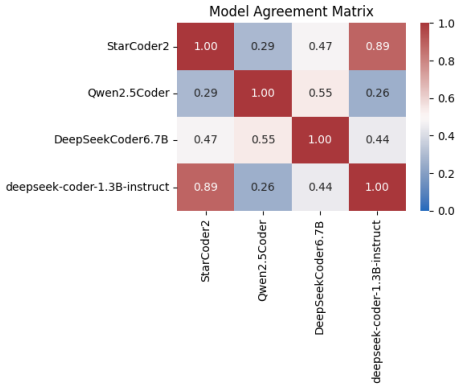

Figure 3: Agreement heatmap across models.

# Disagreeing Models	Count
0	14,318
1	57,517
2	55,054

Table 4: Model disagreement frequency.

6.3 Ranking Metrics

EGAP was used as the relevance signal to calculate ranking-based metrics (MRR, nDCG, and ERR) as seen in Table 5.

Model	MRR	nDCG	ERR
StarCoder2	0.7508	0.8206	0.7508
Qwen2.5Coder	0.7574	0.8213	0.7574
DeepSeekCoder-6.7B	0.7459	0.8155	0.7459
DeepSeek-Coder-1.3B-Instruct	0.7529	0.8206	0.7529

Table 5: Model ranking performance across MRR, nDCG, and ERR.

6.4 Final Ranking Score

Accuracy, Kendall-τ, MRR, nDCG, and modified EGAP were used to calculate a weighted composite as show in Table 6.

6.5 Discussion

The findings show quantifiable variations in the four LLMs' performance in paired comparisons that take sustainability into account. The highest accuracy and ranking scores were obtained by Qwen2.5Coder, closely followed by DeepSeek-Coder-1.3B-Instruct and StarCoder2. Some model pairs, such as StarCoder2 and DeepSeek-Coder-1.3B-Instruct, showed excellent alignment according to agreement analysis, while others showed low agreement, indicating diverging decision tendencies.

While ranking measurements show systematic variance in how consistently each model chooses the more efficient implementation, energy-aware results (EGAP-driven metrics)

Model	Final Score
Qwen2.5Coder	0.4760
DeepSeek-Coder-1.3B-Instruct	0.4688
StarCoder2	0.4655
DeepSeekCoder-6.7B	0.4563

Table 6: Final composite score for each model.

show that all models perform moderately when the efficiency difference between submissions is significant. When these trends are combined, Qwen2.5Coder is ranked first overall in the composite rating.

Together, these results indicate that although existing LLMs show some ability to identify more efficient code variations, performance is still inconsistent among models and measurements. Section 7 offers more information on methodological constraints and implications.

7 Conclusion and Future Work

The purpose of this study was to determine whether large language models (LLMs) can consistently determine which of two functionally equal code submissions is more energy-efficient. The methodology assessed model behavior across correctness metrics, ranking metrics, model agreement patterns, and sustainability-specific measures like EGAP at both the pair and problem levels using a controlled pairwise comparison framework applied to a sizable, varied subset of the Project CodeNet dataset.

The findings show that existing LLMs have a limited but inconsistent capacity to choose the more effective implementation. Even though Qwen2.5Coder performed best overall in terms of accuracy, Kendall-τ, MRR, nDCG, and adjusted EGAP, there was significant disagreement between models and the absolute accuracy values stayed near random chance. This suggests that in modern LLMs, efficiency-aware reasoning is still an emerging rather than mature skill. With no model exhibiting great consistency across all criteria, the significant performance disparities between models further demonstrate how model-dependent energy-efficiency judgment is. All of the results point to the fact that while LLMs have early indications of comparative efficiency reasoning that could be improved with focused refining, they are not yet trustworthy as stand-alone tools for sustainability-aware code review.

Despite these drawbacks, the work integrates controlled prompting, direction-corrected predictions, composite EGAP analysis, and multi-metric ranking to offer a scalable experimental basis for assessing sustainability-aware decision making. The methodological framework, which may be easily expanded as new models or efficiency indicators become available, is just as important as the empirical results.

There are numerous ways to further this research:

- **Incorporation of more granular sustainability metrics:** Hardware-level metrics like CPU energy-per-instruction, cache misses, or dynamic power measures may be included in future research. Compared to the execution time, memory, and code size provided by CodeNet, these parameters would enable more detailed examination.

- **Evaluation using larger and more diverse LLM families:** It may be possible to determine whether architectural or training-objective variation improves sustainability-aware reasoning by using instruction-tuned, mixture-of-experts, or reinforcement-optimized models.
- **Enhanced Prompting and Explainability for Efficiency Reasoning:** Future research may examine whether allowing LLMs to use tool-assisted analysis, employ enhanced contrastive prompts, or offer brief reasoning improves their performance. By combining these prompting strategies with simple explainability techniques, such as emphasizing expensive loops, recursion depth, or memory-intensive structures, it may be possible to shed light on the reasons behind certain efficiency decisions made by models and differentiate real reasoning from heuristics.
- **Development of training or fine-tuning methods for sustainability:** Future models may be able to actively prioritize sustainability during code generation or selection by employing reinforcement learning with efficiency-based reward signals or fine-tuning LLMs directly on energy-labeled samples.
- **Expansion to multi-variant or full-ranking scenarios:** Pairwise preference is assessed in this study. The model would more closely resemble real-world optimization pipelines if it were extended to rank complete sets of submissions per problem.

References

Argerich, M. F. and Patiño-Martínez, M. (2024). Measuring and improving the energy efficiency of large language models inference, *IEEE Access* **12**: 80194–80207.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N. and Hullender, G. (2005). Learning to rank using gradient descent, *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pp. 89–96.

Castellanos-Nieves, D. and García-Forte, L. (2023). Improving automated machine-learning systems through green ai, *Applied Sciences* **13**(20): 11583.

Manning, C. D., Raghavan, P. and Schütze, H. (2008). *Introduction to Information Retrieval*, Cambridge University Press.

Mehta, Y., Xu, R., Lim, B., Wu, J. and Gao, J. (2023). A review for green energy machine learning and ai services, *Energies* **16**(15): 5718.

Moumoula, M. B., Kaboré, A. K., Klein, J. and Bissyandé, T. F. (2025). The struggles of llms in cross-lingual code clone detection, *Proc. ACM Softw. Eng.* **2**(FSE).

Noureddine, A., Rouvoy, R. and Seinturier, L. (2013). A review of energy measurement approaches, *Proceedings of the 2013 IEEE/ACM International Conference on Green and Sustainable Software (GREENS)*, pp. 21–30.

Pazienza, A., Emanuele, U., Paolo, D. and Livio, T. (2024). A holistic approach to environmentally sustainable computing, *Journal of Sustainable Computing: Informatics and Systems* **41**: 100–118.

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

Shi, J., Yang, Z., Kang, H. J., Xu, B., He, J. and Lo, D. (2024). A review of sustainable computing strategies and architectures.

Solovyeva, L., Weidmann, S. and Castor, F. (2025). Ai-powered, but power-hungry? energy efficiency of llm-generated code, *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pp. 49–60.

Tuttle, J. F., Chen, D., Nasrin, A., Soto, N. and Zong, Z. (2024). Can llms generate green code? a comprehensive study through leetcode, *2024 IEEE 15th International Green and Sustainable Computing Conference (IGSC)*, pp. 39–44.

Verdecchia, R., Sallou, J. and Cruz, L. (2023). A systematic review of green ai, *ACM Computing Surveys* .

Wang, J. and Chen, Y. (2023). A review on code generation with llms: Application and evaluation.

Wang, P., Svajlenko, J., Wu, Y., Xu, Y. and Roy, C. K. (2018). Ccaligner: A token based large-gap clone detector, *Proceedings of the 40th International Conference on Software Engineering*, pp. 1066–1077.

Yan, W. et al. (2024). Codescope: An execution-based benchmark for evaluating llms on code tasks.