Comparative Analysis of Carbon Footprint in Manual vs. LLM-Assisted Code Development

Kuen Sum Cheung
King's College London
London, UK
kuen.cheung@kcl.ac.uk

Mayuri Kaul
King's College London
London, UK
mayuri.kaul@kcl.ac.uk

Gunel Jahangirova
King's College London
London, UK
gunel.jahangirova@kcl.ac.uk

Mohammad Reza Mousavi
King's College London
London, UK
mohammad.mousavi@kcl.ac.uk

Eric Zie
Charsfield Research & Advisory
London, UK
eric.zie@cranda.digital

Abstract

Large Language Models (LLM) have significantly transformed various domains, including software development. These models assist programmers in generating code, potentially increasing productivity and efficiency. However, the environmental impact of utilising these AI models is substantial, given their high energy consumption during both training and inference stages. This research aims to compare the energy consumption of manual software development versus an LLM-assisted approach, using Codeforces as a simulation platform for software development. The goal is to quantify the environmental impact and propose strategies for minimising the carbon footprint of using LLM in software development. Our results show that the LLM-assisted code generation leads on average to 32.72 higher carbon footprint than the manual one. Moreover, there is a significant correlation between task complexity and the difference in the carbon footprint of the two approaches.

CCS Concepts

• **Software and its engineering → Software development methods**;

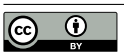
Keywords

software engineering, sustainability, carbon footprint, manual code generation, LLM code generation

ACM Reference Format:
Kuen Sum Cheung, Mayuri Kaul, Gunel Jahangirova, Mohammad Reza Mousavi, and Eric Zie. 2025. Comparative Analysis of Carbon Footprint in Manual vs. LLM-Assisted Code Development. In *1st International Workshop on Responsible Software Engineering (ResponsibleSE '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3711919.3728678

1 Introduction

Large Language Models (LLM) are disrupting many sectors [20] and providing novel ways of producing different types of content [11].



ResponsibleSE '25, Trondheim, Norway

ACM ISBN 979-8-4007-1461-0/2025/06
https://doi.org/10.1145/3711919.3728678

The impact of LLM has expanded to software development and a myriad of approaches have emerged to assist and automate various software development tasks, such as design, implementation, and testing [14][30]. Since their inception, there has been a genuine concern about the environmental impact of LLM, including their carbon footprint [15]. Several studies have raised ethical concerns and showed that LLM use a massive amount of energy. For instance, the end-to-end carbon footprint of GPT3 is estimated to be 554 tons of CO_2 equivalent [15]. This study aims to provide a quantitative analysis of the environmental impact of LLM in programming tasks, when used in an assistive mode and compare it to the traditional manual process of coding.

To make this feasible, we focus on programming tasks for which there are large datasets of programming effort available from public platforms. We then run the same tasks in the LLM-based process and compare the efficiency, correctness, and end-to-end estimated energy consumption of the two approaches. Using these metrics, we answer the following research questions.

(RQ1) Does LLM-based software development lead to less carbon emissions than manual software development?

(RQ2) Is there a correlation between the complexity of the requirements and the difference in carbon footprint between the LLM-based and manual approach? Here, by 'complexity' we mean the difficulty level assigned by Codeforces to a given problem and 'requirement' refers to the set of instructions that define the task in a Codeforces problem statement.

The overall objectives of our study are: 1) to thoroughly analyse the sustainability of LLM-based software development in comparison to manual-based software development, and 2) to propose strategies for the best-practice use of LLM in software development.

The findings of this paper indicate that the end-to-end energy consumption of the LLM-based process is an order of magnitude larger than that of the manual process in all our sampled tasks. Moreover, the gap between the energy consumption of the two approaches increases significantly with the complexity of the requirements. Our results indicate the necessity of a broader study in order to find processes in which the LLM-based approaches may have a comparable or lower energy consumption to justify their replacement for the manual process. We provide the code we have used to calculate the metrics as well as our full experimental data as part of our replication package [9].

2 Literature Review

In this section, we review the literature on the carbon footprint of software development, with a focus on the emerging role of LLM. We provide an overview of the key relevant studies and identify any gaps in the current body of research.

Carbon Footprint of Software Development. There is a substantial and growing interest in assessing the carbon footprint of traditional software development; we refer to surveys [16, 28], scoping- [10, 21, 25], and mapping studies [12]. Recent work indicates a lack of awareness and evaluation methods as major difficulties in adopting sustainable practices in software development [16, 18, 29]. We address these by providing a basic methodology to evaluate and compare LLM-based software development with manual development. Moreover, we raise awareness by demonstrating significant, and to our knowledge hitherto unknown, differences in the carbon footprints of the two approaches.

Carbon Footprint of LLM-Assisted Software Development. To date, there have been few studies addressing the carbon footprint of LLM in software development. Faiz et al. explore the substantial carbon footprint generated by LLM during training and inference [15]. The end-to-end carbon footprint of GPT3 is estimated to be 554 tons of CO_2 equivalent which indicates the high environmental cost of LLM-assisted programming. Additionally, a recent line of work indicates that the code generated by the LLM has a larger carbon footprint than the human-written code [26, 27]. This line of work differs from our work in that it focuses on the sustainability metrics of LLM-generated code rather than evaluating the carbon footprint of the development effort. The work by Belchev [13] is one of the first works that investigates this direction with a focus on the LLM's energy consumption and efficiency in software development tasks such as code generation, bug fixing, documentation, and testing. However, this work does not provide a comparison to the carbon footprint of the traditional, human-led development efforts.

Identified Research Gaps. While there is increasing interest in studying both the carbon footprint of traditional software development and the environmental impact of LLM, there is a notable gap in research focused on the carbon footprint of LLM-assisted software development and how it compares to the manual software development. The broader context of using LLM throughout the software development lifecycle remains underexplored. Our research aims to fill this gap.

3 Programming Tasks Dataset

In this section, we describe the dataset selection process and criteria for task inclusion.

3.1 Choice of Programming Tasks Dataset

To answer our research questions, we include programming task datasets satisfying the following two essential criteria:

(1) **Well-Defined and Diverse Tasks:** We required tasks that are clearly defined and cover a range of complexities. In particular, every task is accompanied by clearly defined test cases and explicit instructions, which provide concrete examples of expected functionality and performance benchmarks. Well-defined tasks ensure that the LLM focuses on solving the task without performance loss due to ambiguity. Additionally, incorporating tasks of varying complexity allows us to measure how the carbon footprint scales with the difficulty of the tasks, providing insights into the energy efficiency of the LLM across different types of challenges.

(2) **Comprehensive Submission Data:** Beyond just the tasks, we needed a substantial amount of data, including detailed submission records from at least 1,000 participants who successfully solved each task to avoid bias, account for variability in human performance, and gain statistical confidence. This data should include crucial metrics such as code runtime, memory usage, and the time spent on each task. We would prefer a platform that provides an API to facilitate the automation of our process.

When choosing a dataset for programming tasks, multiple platforms were considered, including LeetCode [6], HackerRank [4], and Codeforces [1]. Each of these platforms hosts competitive programming contests and provides a range of task complexities, making them potential candidates for our research. However, the final selection was narrowed down to Codeforces because none of the other platforms, e.g., LeetCode and HackerRank, provide the data needed for our study, particularly regarding the time spent on each task. This data is essential for estimating the task complexity and calculating the associated carbon footprint. Codeforces, however, provides the data through an API facilitating the automation of our measurement and comparison process.

Codeforces is one of the most popular platforms for practicing competitive programming, attracting both novice and experienced programmers from around the world. Below we provide some key concepts used in the remainder of our methodology:

- **Contests**: Users participate in timed contests, which feature a suite of well-defined programming tasks of varying complexity. These contests are numbered as rounds; in our experiment, we selected rounds 1983, 1984, and 1994 for analysis, which have sufficiently many tasks of varying complexity with sufficiently many participants (12 tasks in total).
- **Rating System**: After participating in a contest, a user's rating will increase or decrease based on their performance relative to other contestants. With a current range of ratings from 4009 to -53, this rating system confirms that both novice and experienced programmers actively use Codeforces.
- **Editorials**: After a contest has ended, the organizers release code solutions and detailed editorials explaining how to solve each task. These editorials were used as a part of the prompt to the LLM.

3.2 Task Selection

Within Codeforces not all tasks had the desired number of submissions or included the required data, such as code runtime or memory usage. We focused on tasks that met the following conditions:

(1) Each included task should be solved by at least 1,000 participants to ensure diversity and statistical significance. In our study, we assume that the programming experience held by the participants is normalized across participants based on the variety of user ratings in the contest.

(2) Each included task should provide code runtime and memory usage.

3.3 Justification for Python-Specific Focus

Python was chosen as the primary language for this study for two main reasons. First, GPT-4 outputs Python by default when generating code, making it critical to assess LLM performance in the language it most frequently uses. Second, Python is the second most popular language used on Codeforces, with a wide range of participants consistently submitting solutions in Python during contests. This popularity ensures that we have access to a large dataset of Python submissions, providing the necessary metrics to evaluate both runtime performance and carbon footprint.

4 LLM-Assisted Code Generation

To assess the carbon footprint of the LLM-assisted approach, we need to simulate software development using LLMs. For this, we have designed the following process:

(1) We provide the problem to the LLM and ask the LLM to write the Python solution for it.
(2) We submit the code generated by the LLM to Codeforces, and record the number of tests passed.
(3) If any of the test cases fail, we provide the LLM with the error trace of the failing test cases and ask it to fix the code. We repeat this step up to 4 times. This number is chosen as a practical balance: it gives the LLM several chances to correct its mistakes while keeping the overall process efficient, given that improvements tend to taper off after a few iterations.
(4) If the LLM still cannot generate code that passes all test cases after 5 queries, we assume that it is not capable of solving this task without human support. To provide such support, we pass the LLM the *human insight* provided by Codeforces for this specific task. This insight is a short textual guide for the programmer on how to approach the task.
(5) If the LLM still fails, we feed it the test result, keep the human insight in the prompt and ask the LLM to fix the code. We repeat this step for up to 3 times. Similarly, the limit of 3 iterations in this phase is set to offer the LLM enough opportunities to integrate the additional human guidance while maintaining a feasible and controlled experimental framework.

For each task, we repeat the process starting from Step 1 three times and record the mean number for each metric. Our study uses 3 contests which have 12 valid tasks. Figure 1 demonstrates one repetition of the experiment for one task.

The LLM we choose in this study is GPT-4. We chose this model because we found sufficient data on training and query costs, which are relevant to our energy estimation. For most other popular (e.g., programming-focused) LLM, such data is not available.

5 Carbon Footprint Estimation

In this section we introduce the key terms and formulas required for carbon footprint estimation and discuss the values that we used for various metrics. We then describe how we estimated the carbon footprint for manual and LLM-assisted code generation.

5.1 Key Terminology, Formulas and Metric Values

To provide details on carbon footprint estimation that we have performed as part of this study, it's essential to define key terms. The **carbon footprint (CF)** represents the total greenhouse gas emissions caused directly or indirectly by a system, typically measured in kilograms of CO_2 equivalent ($kgCO_2e$). In the context of software development, this footprint is tied to the total energy consumed (TTEC) during execution. The **consumed energy (*E*)** is the product of the power of a device (p) and the time (t) it runs and is typically measured in kilowatt-hours (kWh):

$$E = p \times t \tag{1}$$

Carbon intensity (CI) refers to the amount of carbon dioxide (CO_2) emissions produced per unit of energy consumed, typically measured in kilograms of CO_2 equivalent per kilowatt-hour ($kgCO_2e/kWh$). The carbon intensity of an energy source is a key factor in determining the environmental impact of energy consumption because different energy sources emit different amounts of CO_2 when generating electricity. For instance, fossil fuels have high carbon intensities because burning them releases a significant amount of carbon dioxide, while renewable energy sources such as wind, solar, or hydro have much lower carbon intensities, as they do not produce CO_2 when generating electricity. It should also be noted that the carbon intensity of energy sources is location-sensitive and can further be refined if information about the geographical location of energy sources is provided.

The carbon footprint (CF) of software can be calculated using the following formula:

$$CF = E \times CI \tag{2}$$

where E is the consumed energy and CI is the carbon intensity of that energy.

As the approximation for carbon intensity, in this study we use the data provided by the 2023 report from Nowtricity [2]. This report estimates emissions based on the life cycle CO_2 equivalent for each energy source. Nowtricity uses methods and data defined in UNECE [7] and IPCC [3] reports, including not just direct emissions but also those from infrastructure and the supply chain. Based on all this data, Nowtricity reports the average emissions of 217g CO_2 per kWh which is the number we use in all of our calculations.

As Formula 1 indicates, to compute energy consumption, we need to know the values for power (p). In our study, we assume that the code generation is performed using a standard laptop. We distinguish between the power the laptop consumes during regular use for coding activities (p_{laptop}) and the power it draws when running the code (p_{runtime}). As the value for p_{laptop} we use the 4.075W, the average between the values of 3.59W (reported by Asus [8]) and of 4.62W (reported by Dell [5]).

We define p_{runtime} as the combined power of the CPU (p_{cpu}) and RAM (p_{ram}), adjusted by the memory usage percentage (u):

$$p_{\text{runtime}} = p_{\text{cpu}} + p_{\text{ram}} \times u \tag{3}$$

In our study, we assume that a RAM of 16 GB is being used, and therefore estimate u as the used RAM divided by 16. We measure p_{cpu} and p_{ram} using the Python library CodeCarbon [23] by running

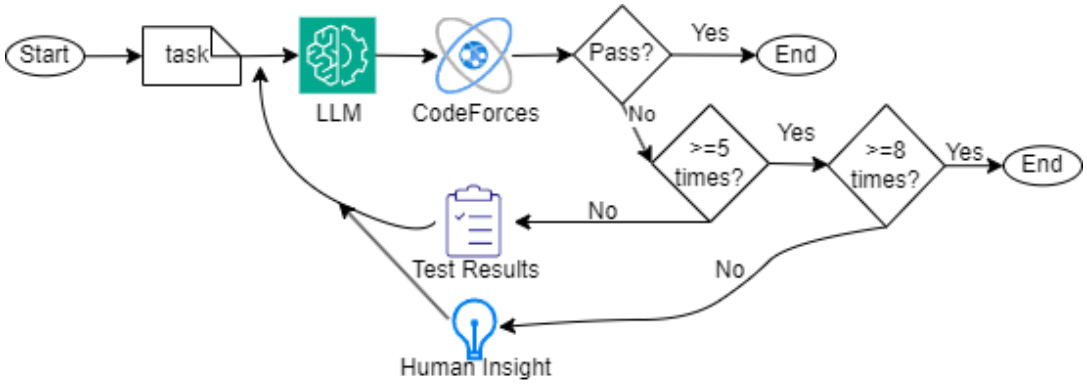


Figure 1: One Iteration of the LLM Experiment

the code locally. However, it was observed that the CPU power usage remained consistent and was capped at the Thermal Design Power (TDP) of the processor. Similarly, runtime RAM power is also capped to a certain value.

5.2 Carbon Footprint for Manual Approach

In this subsection, our goal is to estimate the *carbon footprint* of the manual code generation for each task. We calculate the *total energy consumption* as the sum of three distinct parts: *Coding Energy Consumption* (CEC), *Debugging Energy Consumption* (DEC) and *Testing Energy Consumption* (TEC).

Coding Energy Consumption (CEC): This metric refers to the energy consumed by an average laptop during the coding process, excluding energy used for testing and debugging. We calculated it using the Formula 1 and p_{laptop} value for the power.

To calculate the mean time spent (MTS) on coding for each task denoted by t, we used the relative submission time provided by Codeforces. The relative submission time is the timestamp of when a participant submitted a solution for a task, measured from the start of the contest. Since Codeforces only provides this relative submission time (rather than the actual time spent on each task), we had to take a number of measures explained below to ensure accuracy.

First, to ensure that we could accurately estimate the time spent on each task, we filtered out participants who did not complete the tasks sequentially. This is because, for participants who completed tasks in a different order, it is unclear how much time they spent on each specific task. Sequential completion means the participant worked on the tasks in the order they were presented in the contest, allowing us to infer more accurate task-level time estimates.

Second, we applied standard methods to exclude outliers. Given that we assume the skills of participants are normally distributed, we used the common statistical technique of excluding data points that are more than two standard deviations from the mean.

Debugging Energy Consumption (DEC): This metric quantifies the energy consumed during the time spent running code in the debugging phase.

First, we calculate the additional power consumed during debugging by calculating the runtime power (3) and excluding p_{laptop} (which is already accounted for in the Coding Energy Consumption metric). The formula becomes:

$$p_{\text{debug}} = p_{\text{runtime}} - p_{\text{laptop}}$$

To estimate how long the code runs during debugging, we rely on existing studies. According to Stripe [24], developers spend about 42% of their total development time on debugging. Ko et al. [17] found that, on average, only 10% of the debugging time is spent actually running the code. Therefore, the time spent running code during debugging is calculated as:

$$t_{\text{debug}} = \text{Mean Time Spent} \times 0.42 \times 0.1$$

Testing Energy Consumption (TEC): We estimated the energy consumption during testing based on the code's runtime. Similarly to CEC, we use Formula 1 to calculate the testing energy consumption. However, as the value of power we use p_{runtime}, as during the testing process we need to actually run the code.

To estimate the time the testing process takes, we need to know how much time one run of the tests takes and how many times the tests were run. For the former, we use the run time data and calculate the average across all participants for each task. For the latter, we use the mean of the number of times a user submits their code. The values for both these metrics are provided by Codeforces and we accessed them using Codeforces API.

5.3 Carbon Footprint for LLM-Assisted Approach

To calculate the total energy consumption (TTEC) for the LLM-assisted approach we need to account for multiple components. The first one is the energy consumption associated with sending queries to an LLM (E_{query}). The second component is related to the fact that a human insight is part of our LLM-assisted approach (Step 4). Therefore, we need to calculate the energy consumption associated with this step by estimating the time it takes to produce the human insight (t_{insight}). Lastly, the process we have explained in the previous subsection does not always produce a solution that passes all test cases. Therefore, there is a need for the developer to understand the solution generated by the LLM and add the missing functionalities. This makes for our third component to estimate

which we need to know the time spent on adding missing functionalities ($t_{\text{add\_functionalities}}$).

When accounting for all the mentioned components, our formula for total energy consumption becomes:

$$E_{\text{total}} = E_{\text{query}} + (t_{\text{insight}} + t_{\text{add\_functionalities}}) \times p_{\text{laptop}}$$

Query Energy Consumption (QEC). To estimate the overall query energy consumption we need to account for all queries we pass to the LLM and estimate the energy consumption of a single query. For the latter, we use the finding by Ludvigsen [19] that the energy consumption for each inference (query) is estimated to be 0.0022 kWh, based on the findings. In addition to the inference energy cost, we also account for the energy consumed during the training phase of the Large Language Model (LLM). To estimate the training energy cost per query, we divided the total training energy consumption (50 GWh) by the estimated number of queries that the LLM will handle over its lifecycle (5.68 Giga queries). Based on available data, we estimate the energy consumption per query for training to be 0.0088 kWh. Therefore, the total energy consumption per query, including both inference and training, is 0.011 (0.0022 + 0.0088) kwh.

It should be noted that these numbers are estimated using known hardware specifications and general data. No actual energy profiling tools or power meters were used to measure the precise energy consumption. We will discuss this limitation in more detail in the limitations section.

Estimated Time Spent on Producing the Insight (ETHI): This metric accounts for the time a human spends understanding the task and providing insight to the LLM. According to a study by Minelli et al. [22], developers typically spend 38% of their total time understanding a task.

Since the LLM already generates a partial solution, we adjust this time by considering the percentage of test cases passed by the LLM before receiving human insight. The idea is that the more the LLM succeeds initially, the less time the human needs to spend understanding and solving the remaining parts of the task.

The formula for estimating the time spent on producing the insight is:

$$t_{\text{insight}} = t \times 0.38 \times (1 - TC_{\text{passed}})$$

Where:

- t is the total average time a developer would typically spend on the task,
- 0.38 is the fraction of time developers typically spend on task understanding [22],
- TC_{passed} is the percentage of test cases passed before human insight, and $(1 - TC_{\text{passed}})$ represents the remaining portion of the task that needs human intervention.

Estimated Time to Add Missing Functionalities (ETAF): This metric estimates the time required to manually add any missing functionalities when the LLM fails to fully complete the task, even after human insight is provided. The total time is a combination of the time spent on reading and extending the code, plus the time required to implement the remaining unsolved parts of the task. We exclude the time producing the insight as it means the developer already understands the task.

The formula for estimating the time is:

$$t_{\text{add}} = t_{\text{read\_extend}} + (1 - TC_{\text{after\_insight}}) \times t - t_{\text{insight}}$$

Where:

- $t_{\text{read\_extend}}$ is the estimated time spent on reading and extending the code,
- $TC_{\text{after\_insight}}$ is the percentage of test cases passed after human insight has been given, $(1-TC_{\text{after\_insight}})$ represents the missing percentage of functionalities,
- t is the total average time a developer spends on the task,
- t_{insight} is the time spent on producing the human insight.

To calculate $t_{\text{read\_extend}}$ we rely on the study by Ko et al. [17], which reports that developers typically spend about 20% of their debugging time reading code and another 20% editing (extending) code. Thus, we estimate the time spent on reading and extending as a portion of the total debugging time.

The formula for estimating the time is:

$$t_{\text{read\_extend}} = t \times 0.42 \times (0.2 + 0.2)$$

Where:

- t is the total average time spent on the task,
- 0.42 is the proportion of the total time spent on debugging [24]
- 0.2 is the fraction of debugging time spent on reading code,
- Another 0.2 represents the fraction of debugging time spent on editing (extending) the code.

6 Results and Discussion

In this section, we present and analyse the results obtained from our study. The analysis is structured around our two main research questions.

6.1 RQ1: LLM-based vs. manual software development

To answer RQ1, we need to compare the carbon footprint of the manual approach (based on the Codeforces contest data) to the carbon footprint of the LLM-based approach. Table 1 presents the data for the manual process for the selected 12 tasks from 3 contests. Table 2 presents the same data for the LLM-assisted approach. In each table we report data for three different contest that consist of 5, 4 and 3 tasks correspondingly. In the first column we list metrics associated with the calculation of the carbon footprint with their units indicated in the brackets. As Table 1 shows, the mean time spent (row "MTS") on each task varies on average between 444 to 2487 seconds, indicating the different level of effort the tasks take.

In Table 2 the row "NQBH" reports the number of queries sent to LLM before the human insight during the LLM-assisted software development. As the values in this row indicate, for only 3 tasks out of 12, the maximum number of 5 queries was not reached, indicating that LLMs were not able to solve the task (such that all provided test cases pass) without the support from human. The row "NHIQ" reports the number of human insight queries passed to LLM. We can see that for 4 tasks out of 12, this number also reached its maximum value of 3. The row "TPAH" reports the percentage of the test cases that passed after the human insight. This number is

Table 1: Manual Carbon Footprint Estimation for the selected contest rounds: each column represents a distinct task within a specific contest. For each task, the table reports Mean Time Spent (MTS), Coding Energy Consumption (CEC), Testing Energy Consumption (TEC), Debugging Energy Consumption (DEC), Total Energy Consumption (TTEC), and the resulting Carbon Footprint (CF). The measurement unit for each reported metric is indicated in brackets.

	Contest 1983					Contest 1984				Contest 1994		
	A	B	C	D	E	A	B	C	D	A	B	C
MTS (s)	444	2216	2361	2018	2487	805	1437	1803	2146	748	1300	1783
CEC (kwh)	5.03E-04	2.51E-03	2.67E-03	2.28E-03	2.82E-03	9.11E-04	1.63E-03	2.04E-03	2.43E-03	8.47E-04	1.47E-03	2.02E-03
TEC (kwh)	3.37E-07	1.46E-06	3.49E-06	3.35E-06	9.64E-06	6.82E-07	1.37E-06	2.59E-06	4.02E-06	8.41E-07	1.03E-06	2.26E-06
DEC (kwh)	5.14E-05	2.57E-04	2.74E-04	2.34E-04	2.88E-04	9.32E-05	1.66E-04	2.09E-04	2.49E-04	8.66E-05	1.51E-04	2.07E-04
TTEC (kwh)	5.54E-04	2.77E-03	2.95E-03	2.52E-03	3.11E-03	1.01E-03	1.79E-03	2.25E-03	2.68E-03	9.34E-04	1.62E-03	2.23E-03
CF (g)	0.120	0.600	0.640	0.547	0.676	0.218	0.389	0.489	0.582	0.203	0.352	0.483

Table 2: LLM-assisted Carbon Footprint Estimation for the selected contest rounds: each column represents a distinct task within a specific contest. For each task, the table reports Number of Queries Before the Human insight (NQBH), Number of Human Insight Queries (NHIQ), Percentage of the Test cases that Passed after the Human insight (TPAH), Query Energy Consumption (QEC), Estimated Time Spent on Producing the Insight (ETHI), Estimated Time to Add Missing Functionalities (ETAF), Total Energy Consumption (TTEC) and the resulting Carbon Footprint (CF). The measurement unit for each reported metric is indicated in brackets.

	Contest 1983					Contest 1984				Contest 1994		
	A	B	C	D	E	A	B	C	D	A	B	C
NQBH	1	5	5	5	5	4	5	5	5	1.67	5	5
NHIQ	0.00	1.33	3.00	1.67	3.00	0.00	2.00	3.00	3.00	0.00	1.00	2.33
TPAH	100%	100%	0%	69%	0%	100%	100%	73%	69%	100%	100%	33%
QEC (Kwh)	0.011	0.070	0.088	0.073	0.088	0.044	0.077	0.088	0.088	0.018	0.066	0.081
ETHI (s)	0	842	897	767	945	0	546	685	748	0	494	678
ETAF (s)	0	0	1860	1459	1960	0	0	109	282	0	0	817
TTEC (kwh)	0.011	0.071	0.091	0.074	0.091	0.044	0.078	0.089	0.089	0.018	0.067	0.082
CF (g)	2.39	15.32	19.77	16.15	19.81	9.55	16.84	19.29	19.35	3.99	14.44	17.86

Table 3: Ratio Difference between LLM and Manual Approach: each column represents a distinct task within a specific contest. The mean and standard deviation are reported across all tasks.

	Contest 1983					Contest 1984				Contest 1994		
	A	B	C	D	E	A	B	C	D	A	B	C
ratio	19.92	25.53	30.89	29.52	29.30	43.81	43.29	39.45	33.25	19.66	41.02	36.98
Mean: 32.72						Standard deviation: 8.41						

below 100% for 6 tasks, i.e. for half of the tasks the LLM-assisted approach could not fully solve the task.

When it comes to the overall carbon footprint, as it is clear from the results shown in Tables 1 and 2, the LLM-assisted software development leads to a significantly higher carbon footprint compared to manual software development. Focusing on the total energy consumption and carbon footprint (last two rows of Tables 1 and 2), the LLM-based approach has at least 19.92 times and at most 43.81 times more carbon footprint (resp. energy consumption) than the manual approach (mean: 32.72, standard deviation: 8.41) The comparison of the two approaches is summarised in Table 3.

A closer look into the carbon footprint for the manual code generation shows that across 12 tasks the coding process accounts on average for 90.61% of the carbon footprint, while testing and debugging account for 9.28% and 0.11% correspondingly. For LLM-assisted code generation, 98.68% of the carbon footprint is due to querying the LLM. The remaining 0.77% comes from generating human insights that support the LLM, while 0.56% is due to developers adding missing functionalities. This suggests that optimizing LLM

queries or incorporating more efficient models could significantly reduce the environmental impact of LLM-assisted development workflows.

6.2 RQ2: The Impact of Task Complexity

To address RQ2, i.e., to investigate the correlation between task complexity and the difference in carbon footprints between the manual and LLM-assisted approaches, we first present a scatter plot 2a showing the relationship between task complexity (approximated by mean time spent) and the direct difference in carbon footprints between the manual and LLM-assisted approaches.

From our visual inspection of the scatter plot, it appears that as task complexity increases, the direct difference in the carbon footprint between LLM-assisted and manual approaches also increases. The plot 2b illustrates our expected pattern. We anticipate a relationship resembling a tanh function, where the direct difference in carbon footprint has lower and upper limits because of the number of maximum (8) and minimum (1) queries we set in our experiment.

To quantitatively assess the strength and significance of this relationship, we conducted two statistical tests: the Pearson correlation test and the Spearman rank correlation test with significance level (α) at 0.05. We tested the following hypotheses:

- **Null Hypothesis (H_0)**: There is no correlation between task complexity and the direct difference in carbon footprints between manual and LLM-assisted approaches.
- **Alternative Hypothesis (H_A)**: There is a significant correlation between task complexity and the direct difference in carbon footprints between manual and LLM-assisted approaches.

The results of our statistical tests are as follows:

- **Pearson Correlation Test:** The Pearson correlation coefficient was 0.890, with a p-value of 0.00011. Since the p-value is less than the significance level of 0.05, we reject the null hypothesis and conclude that there is a statistically significant positive linear correlation between task complexity and the direct difference in carbon footprints.
- **Spearman Rank Correlation Test:** The Spearman correlation coefficient was 0.840, with a p-value of 0.0006. This result also shows a significant monotonic relationship.

Based on these analyses, we conclude that there is a significant linear correlation between task complexity and the difference in carbon footprint of the two approaches. This finding emphasizes the potential drawbacks of using LLM for more complex tasks.

6.3 Best Practices for Green Coding with LLM

In this section, we provide some recommendations that can inform future best practices for using LLM in green coding to achieve efficiency and minimal carbon footprint. To optimize the use of LLM in green software development, we recommend investigating the following approaches:

- For complex tasks, decomposing them into smaller, manageable sub-tasks can mitigate the significant energy use gap (up to 30% based on our observed data). This decomposition could be done manually or automatically, but the carbon footprint of the decomposition itself needs to be further investigated.
- We recommend assessing task complexity and adjusting the development process around it to minimise environmental impact. Depending on the task complexity we can determine the level of human involvement / autonomy in code generation. Other parameters such as the LLM-type and the resources used for training the LLM can further be used in fine-tuning the process once more data is made available about the public and open-source LLM, their training, and their carbon footprint.

These practices can guide developers in making more informed decisions when using LLM in software development, balancing the benefits of LLM with the need to reduce environmental impact.

7 Threats to Validity

We acknowledge several threats to the validity of our results and its generalisation. These limitations primarily arise from the limited data available to date and limited background research and methodologies.

Scope of Analysis: Our study focuses solely on the implementation, testing, and debugging phases of software development, excluding other stages such as planning and design. This limitation is due to the availability of data at the time of our research, and thus, our findings may not fully represent the entire software development life cycle.

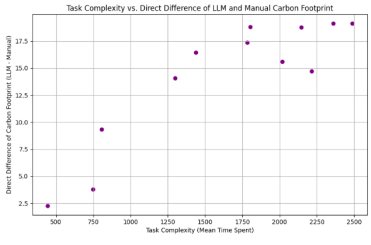
Assumptions about Hardware: We assumed a specific machine configuration for the manual coding side of the study. Variations in the Thermal Design Power (TDP) of different machines could result in different energy consumption metrics. However, our conclusion on the statistical test should remain the same as our metrics in LLM also consider the manual effort.

LLM and Prompting Limitations: Our study is limited to the use of a single LLM, and as such, our findings may not be generalisable to other models with different architectures or training data. Additionally, we employed very simple zero-shot prompts—providing only the task description and asking the model to generate code without further guidance. While this reduces prompt-related bias, it may not reflect the full potential of LLMs under more sophisticated prompting strategies. Due to space constraints, the full set of prompts used is provided in our replication package [9].

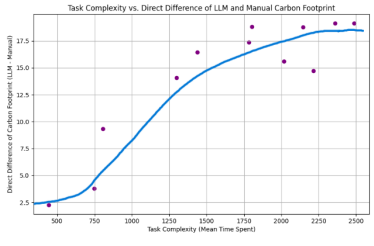
Limited Data Points: While our analysis would benefit from more data points, including a broader range of tasks with different complexity, the limitations in the number of queries allowed with GPT-4 made it time-consuming to conduct the LLM experiments.

Estimation of Energy Consumption: Many of the metrics, such as query energy consumption, are estimated using available hardware specifications of the LLM rather than precise energy profiling tools or power meters. Likewise, the total number of queries GPT-4 will handle over its lifetime is unknown but we estimated it using the lifetime of GPT-3.

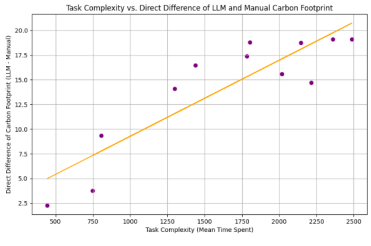
Competitive Programming Environment: The tasks selected for our study are drawn from a competitive programming environment, which may not fully reflect real-world software development processes. As a result, the findings from this study might not directly translate to typical software development scenarios.



(a) Task Complexity vs. Direct Difference of LLM and Manual Carbon Footprint



(b) Task Complexity vs. Direct Difference of LLM and Manual Carbon Footprint with Expected pattern



(c) Task Complexity vs. Direct Difference of LLM and Manual Carbon Footprint with Best-Fit Line

Figure 2: Comparison of Task Complexity and Carbon Footprint

8 Conclusions

In conclusion, we have shown that using an LLM-based approach to software engineering results in higher carbon emissions than a manual approach, with the gap increasing linearly as task complexity grows. Our work presents opportunities for a more in-depth analysis. For instance, we could delve into the reasons behind the difference in carbon emissions by conducting a detailed examination of LLM. This could involve studying the lifetime of an LLM or the training process, and investigating the impact of these variables. Such a comprehensive analysis could pave the way for making AI models, particularly LLM, more sustainable and energy efficient.

9 Acknowledgments

This work has been supported by the ITEA grants GreenCode (project number 23016) and GENIUS (project number 23026).

References

[1] Codeforces. https://codeforces.com/.
[2] Current emissions in united kingdom. https://www.nowtricity.com/country/united-kingdom/.
[3] Fifth Assessment Report (AR5): Climate Change 2014: Mitigation of Climate Change. https://www.ipcc.ch/report/ar5/wg3/.
[4] HackerRank. https://www.hackerrank.com/.
[5] LCA Latitude 7300 Anniversary Edition. https://www.delltechnologies.com/asset/en-us/products/laptops-and-2-in-1s/technical-support/full-lca-latitude7300-anniversary-edition.pdf.
[6] LeetCode. https://leetcode.com/problemset/.
[7] Life Cycle Assessment of electricity generation options. https://unece.org/sites/default/files/2022-04/LCA_3_FINAL%20March%202022.pdf.
[8] Product Carbon Footprint Report. https://esg.asus.com/english/file/PEP_Notebook_C423.pdf.
[9] Replication package. https://github.com/HayatoKanee/Green-LLM, 2025.
[10] Siti Rohana Ahmad Ibrahim, Jamaiah Yahaya, and Hasimi Sallehudin. Green software process factors: A qualitative study. *Sustainability*, 14(18), 2022.
[11] Chris Arkenberg. Generative ai tools in media and entertainment. *Deloitte Insights*, 2023.
[12] Peter Bambazek, Iris Groher, and Norbert Seyff. Requirements engineering for sustainable software systems: a systematic mapping study. *Requirements Engineering*, 28(3):481–505, September 2023.
[13] Boris Belchev. Code green: Evaluating the carbon and energy implications of llm integration in software development. Master's thesis, University of Twente, 2025.
[14] Lenz Belzner, Thomas Gabor, and Martin Wirsing. Large language model assisted software engineering: Prospects, challenges, and case study. In Bernhard Steffen, editor, *Bridging the Gap Between AI and Reality*, pages 355–374, Cham, 2024. Springer Nature Switzerland.
[15] Ahmad Faiz, Sotaro Kaneda, Ruhan Wang, Rita Chukwunyere Osi, Prateek Sharma, Fan Chen, and Lei Jiang. LLMCarbon: Modeling the end-to-end carbon footprint of large language models. In *The Twelfth International Conference on Learning Representations*, 2024.
[16] Leila Karita, Brunna C. Mourão, and Ivan Machado. Software industry awareness on green and sustainable software engineering: a state-of-the-practice survey. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES '19, page 501–510, New York, NY, USA, 2019. Association for Computing Machinery.
[17] Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12):976, 2006.
[18] Christoph König. Sustainable software engineering: Visions and perspectives beyond energy efficiency. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, ICSE-Companion '24, page 231–233, New York, NY, USA, 2024. Association for Computing Machinery.
[19] Kasper Groes Albin Ludvigsen. ChatGPT's Energy Use per Query. https://towardsdatascience.com/chatgpts-energy-use-per-query-9383b8654487, 2023.
[20] Spyros Makridakis, Fotios Petropoulos, and Yanfei Kang. Large language models: Their success and impact. *Forecasting*, 5(3):536–549, 2023.
[21] Sean McGuire, Erin Schultz, Bimpe Ayoola, and Paul Ralph. Sustainability is stratified: Toward a better theory of sustainable software engineering. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1996–2008, 2023.
[22] Roberto Minelli, Andrea Mocci, and Michele Lanza. I know what you did last summer - an investigation of how developers spend their time. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 25–35, 2015.
[23] Victor Schmidt, Karan Goyal, Akhilesh Deepak Joshi, Benjamin Feld, Lyndon Conell, Nikolas Laskaris, Daphne Blank, James Wilson, Sam Friedler, and Sasha Luccioni. CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing. https://doi.org/10.5281/zenodo.4658424, 2021.
[24] Stripe. The developer coefficient. https://stripe.com/files/reports/the-developer-coefficient.pdf, 2018.
[25] Jakub Swacha. Models of sustainable software: A scoping review. *Sustainability*, 14(1), 2022.
[26] Tina Vartziotis, Ippolyti Dellatolas, George Dasoulas, Maximilian Schmidt, Florian Schneider, Tim Hoffmann, Sotirios Kotsopoulos, and Michael Keckeisen. Learn to code sustainably: An empirical study on LLM-based green code generation. *arXiv preprint arXiv:2403.03344*, 2024.
[27] Tina Vartziotis, Maximilian Schmidt, George Dasoulas, Ippolyti Dellatolas, Stefano Attademo, Viet Dung Le, Anke Wiechmann, Tim Hoffmann, Michael Keckeisen, and Sotirios Kotsopoulos. Carbon footprint evaluation of code generation through as a service. In *2024 Stuttgart International Symposium on Automotive and Engine Technology*, pages 230–241. Springer Fachmedien Wiesbaden, 2024.
[28] Colin C. Venters, Rafael Capilla, Elisa Yumi Nakagawa, Stefanie Betz, Birgit Penzenstadler, Tom Crick, and Ian Brooks. Sustainable software engineering: Reflections on advances in research and practice. *Inf. Softw. Technol.*, 164(C), December 2023.
[29] Michael Wahler, Norbert Seyff, and Maria Susana Soriano Ramirez. Exploring assessment criteria for sustainable software engineering processes. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*, ICSE-SEIS'24, page 107–117, New York, NY, USA, 2024. Association for Computing Machinery.
[30] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620v5*, 2023.