Brains Without Brawn: Evaluating CPU Performance for Code Generation with Large Language Models

Miren Illarramendi[1], **Joseba Andoni Agirre**[1], **Aitor Picatoste**[2], **Juan Ignacio Igartua**[2]
[1]Software and Systems Engineering Research Group
Engineering Faculty of Mondragon University
Arrasate-Mondragon, Spain
e-mails: millarramendi@mondragon.edu, jaagirre@mondragon.edu,
[2]Circular economy and industrial sustainability
Engineering Faculty of Mondragon University
Arrasate-Mondragon, Spain
e-mails: apicatoste@mondragon.edu, jigartua@mondragon.edu

***Abstract*—This research presents a comparative analysis of the performance of various Large Language Models (LLMs) for code generation tasks executed on Central Processing Units (CPUs) without the use of dedicated Graphics Processing Units (GPUs). The study evaluates key metrics including inference time, code generation accuracy, CPU and memory usage, and energy consumption. By conducting repeated experiments, we assess the impact of model size and optimization on efficiency in environments lacking GPU resources. Energy consumption is measured using tools like CodeCarbon, focusing on the environmental impact of running these models on CPU-based systems. The findings offer insights into the trade-offs between model precision, resource usage, and energy efficiency, providing valuable guidance for developers and researchers aiming to balance performance and sustainability in low-resource computing environments.**

Keywords-LLMs; GenIA; GreenComputing; Code Generation; Energy Consumption; Sustainability.

I. Introduction

The rapid advancement of LLMs has revolutionized various fields, including Natural Language Processing (NLP), code generation, and even machine translation. Models like GPT-3, DeepSeek, and Llama have shown remarkable capabilities in tasks ranging from text generation to understanding and generating code. However, these models are computationally expensive and require significant resources, particularly during the training and inference stages [1] [2].

While GPUs are typically the hardware of choice for running large-scale machine learning models due to their high parallel processing capabilities, not all environments have access to dedicated GPUs. Many users, particularly those in resource-constrained settings or utilizing cloud computing, must rely on CPUs for model inference. CPUs, though less powerful than GPUs in terms of parallel processing, are widely available and more energy-efficient in certain use cases, especially for smaller models or lightweight tasks [3].

Despite the growing use of LLMs in production environments, there is a lack of comprehensive analysis comparing the performance and sustainability of these models when executed on CPUs versus GPUs. The existing literature focuses mainly on GPU-based performance, leaving a gap in understanding how LLMs perform in real-world scenarios where only CPU resources are available. Some research has pointed out that the energy consumption of LLMs is often underestimated in most studies, with the environmental impact becoming a significant concern when deploying models at scale.

This study aims to address this gap by conducting a comparative analysis of the performance of various LLMs for code generation tasks when executed in GPU-based environments remotely. Specifically, we will focus on several key metrics, including inference time, code generation accuracy, energy consumption, and computational cost. The initial phase will involve measuring the cost of running inference from our local CPU to understand the energy and computational efficiency of remote execution. The next step will be to extend this analysis by deploying the LLMs directly on our CPU for inference, allowing us to compare performance and resource usage when running these models in resource-constrained environments. Models to be evaluated include "gpt-4o", "gpt-4-turbo", "gpt-3.5-turbo", "gpt-4o-mini", "mistralai/Mistral-7B-Instruct-v0.3", "meta-llama/Meta-Llama-3-8B-Instruct", "alpindale/WizardLM-2-8x22B", and "Qwen/Qwen3-235B-A22B-Instruct-25072". These models have been selected due to their variety in size and diverse platforms (OpenAI, HuggingFace), providing a comprehensive comparison of different model architectures and inference performance across various levels of complexity and computational demands.

The contribution of this research is to remotely measure and monitor code generation tasks in inference across different LLMs, in terms of accuracy and CO2 footprint. By analyzing the efficiency of these models in resource-constrained environments, we provide insights into optimizing the use of LLMs and balancing cost with environmental impact.

In the following sections, we will explore the methodology used to measure these performance metrics and present the results of the experiments to offer a comprehensive comparison of model efficiency across different hardware configurations. By doing so, we aim to provide practical guidance for researchers, developers, and organizations seeking to optimize the use of LLMs in resource-constrained environments while considering cost-effective and sustainable deployment strategies.

It is important to note that this study is limited to inference-time evaluation, does not include model training or fine-tuning, and relies partially on remote execution data, which may be affected by variables such as network latency, backend optimizations, and limited visibility into the energy consumption of proprietary systems.

The remainder of the paper is organized as follows: Section II and Section III cover the background and related work, respectively, providing the foundational context for this study. In Section IV, we describe the experimental setup and methodologies employed. Section V presents the experimental results, focusing on performance, throughput, code generation accuracy, and energy efficiency. Section VI offers an in-depth discussion of the evaluation, interpreting the results and their implications. Finally, Section VII concludes the paper and proposes directions for future work.

II. Background

The field of LLMs has seen significant advancements in recent years, driven by the rapid development of deep learning techniques and the availability of large-scale datasets. These models, such as GPT-3 [4] and BERT [5], have achieved impressive results across a wide range of NLP tasks, including text generation, translation, and question answering. More recently, specialized models, such as Codex [6] have been developed for tasks related to code generation and software development.

While LLMs have demonstrated remarkable capabilities, they come with substantial computational requirements. Training these models involves large-scale distributed computing on specialized hardware, often utilizing GPUs to speed up the process. However, inference—the process of using pre-trained models to generate outputs—can also be computationally demanding, particularly when deployed in real-time applications. Typically, GPUs are used for inference due to their ability to handle parallel processing, but not all environments have access to GPUs, especially in resource-constrained settings such as edge devices, mobile platforms, or smaller cloud infrastructures.

The challenge of resource efficiency has become increasingly important as the size of LLMs continues to grow. Models like GPT-3, with over 175 billion parameters [4], consume significant amounts of energy during inference. Studies have highlighted the environmental impact of training and running large-scale models, particularly with respect to their carbon footprint and energy consumption [1]. Energy-efficient models and the optimization of inference processes on CPUs have therefore become crucial areas of research, especially when considering the global push toward sustainable AI [2].

In parallel, the demand for code generation has increased, driven by the need to automate repetitive programming tasks, assist with code completion, and enhance software development processes. Models, such as Codex [6] have shown that LLMs can generate syntactically correct and semantically meaningful code from natural language descriptions. These models have the potential to reduce development time and improve software quality by generating boilerplate code, automating refactoring, and even suggesting optimizations.

However, the deployment of LLMs for code generation in environments with limited hardware resources, such as those relying on CPUs instead of GPUs, raises concerns about the trade-offs between performance and energy efficiency. There is limited research comparing the inference performance and energy consumption of different LLMs in CPU-based environments, which is critical for determining their practical use in everyday software development tasks. Moreover, little to no studies address the cost of inference when utilizing remote models, such as ChatGPT provided by OpenAI, which runs on cloud-based infrastructures. Understanding the energy consumption and computational costs when querying remote models from local CPU environments is crucial for optimizing resources, especially when these models are not deployed locally. This gap in the literature underscores the need for comprehensive analyses that consider both local and remote execution scenarios for LLMs.

III. Related work

The growing reliance on LLMs for tasks, such as code generation, text generation, and question answering has significantly advanced the field of artificial intelligence. However, these models, especially large-scale ones like GPT-3 [4], Codex [6], and BERT [5], have raised concerns regarding their environmental impact due to their substantial energy consumption and carbon footprint. As these models become larger, the need for energy-efficient deployment methods becomes critical, particularly when leveraging resources such as CPUs instead of GPUs, which are commonly used in research environments.

A. Energy Consumption and Sustainability in AI

The environmental impact of LLMs has been a topic of growing concern in recent research. Strubell et al. [1] highlighted the significant energy consumption required to train and run models like GPT-3, estimating that the carbon emissions of training such models can rival those of several cars over their lifetimes. This study emphasizes the need for developing models that are not only accurate but also energy-efficient, promoting the idea of Green AI. However, this research focuses primarily on the training phase and the larger-scale infrastructures typically used for training these models, rather than on their inference phase or CPU-based execution.

Schwartz et al. [2] further expanded on the concept of sustainable AI, advocating for a shift toward models that prioritize resource efficiency. They call for reducing the carbon footprint of deep learning models and propose that energy-efficient algorithms should be a focus in model design. However, their work lacks a focus on real-world inference scenarios, particularly in environments where GPUs are unavailable or impractical for deployment.

Xu et al. [7] provided a comprehensive survey on strategies to improve energy efficiency in deep learning models, addressing

the growing need to reduce the environmental impact of AI systems. They reviewed a variety of techniques, including model compression, pruning, quantization, and efficient data usage, all aimed at optimizing the energy consumption of machine learning models. These methods can significantly reduce the computational load during inference, particularly for large-scale models. While this work offers valuable insights into improving the energy efficiency of deep learning systems, it does not specifically focus on the trade-offs involved in running LLMs, such as GPT-3 or Codex, on CPUs for code generation tasks—an area central to our research.

B. Inference Performance and Resource Allocation

Recent studies have explored optimizing inference efficiency by balancing the load between CPU and GPU. Patterson et al. [8] analyzed the energy consumption and carbon footprint of large-scale deep learning models and discussed strategies for improving energy efficiency during model training and inference. They highlighted the cost and energy-efficiency trade-offs between using GPUs and CPUs, with a focus on reducing the environmental impact of large models like GPT-3. However, their work primarily focuses on general model training and does not specifically address LLMs or code generation tasks. Furthermore, it does not consider the role of automated tools like MLFlow and CodeCarbon, which adjust resources dynamically based on real-time performance and energy consumption metrics.

Furthermore, a more recent study by Patterson et al. [3] provided insight into carbon-efficient machine learning practices, offering actionable strategies to reduce energy consumption in inference tasks, especially when models are run on CPUs in resource-constrained environments. This study is highly relevant to our research, as it provides an essential framework for making inference more sustainable, though it still lacks specific analysis on LLMs for code generation and their optimization on CPUs.

Incorporating energy monitoring into DevOps pipelines has recently been explored by some researchers. For example, CodeCarbon [9] provides a simple framework to measure the carbon footprint of machine learning models during training and inference. By integrating CodeCarbon into the DevOps workflow, practitioners can track the energy consumption and CO2 emissions of models in real time, making it easier to evaluate the environmental impact of model deployment. This approach has been integrated into workflows for smaller, less resource-intensive models but is rarely used for large-scale models like GPT-3 or Codex, especially in CPU-based environments.

A recent study by Rangineeni et al. [10] explored the integration of MLFlow within DevOps pipelines for continuous monitoring and optimization of machine learning models in production environments. Their work highlighted how MLFlow can be utilized to track performance metrics, log experiments, and manage model versions, enabling efficient deployment and resource allocation during inference. They also emphasized the importance of adaptive resource management, which can ensure cost efficiency and sustainability in cloud-based environments. While their research provides valuable insights into optimizing resource allocation using MLFlow, it does not specifically address the application of these practices to LLMs, such as GPT-3 or Codex, for tasks like code generation, nor does it consider the role of energy consumption metrics in the optimization process.

C. Code Generation with LLMs

The application of LLMs for code generation has gained significant attention, particularly with models such as Codex [6], which are specifically designed to generate programming code from natural language prompts. Codex has shown great potential in automating code completion, bug fixing, and refactoring tasks, but there is a lack of research on how these models perform when executed on CPU-based systems as opposed to GPUs.

A recent study by Arora et al. [11] introduced SetupBench, a benchmark designed to evaluate the ability of LLM agents to bootstrap development environments autonomously. The benchmark comprises 93 tasks spanning various programming languages, database engines, and multi-service orchestration scenarios. The evaluation of OpenHands, a state-of-the-art coding agent, revealed low success rates across task categories, particularly in repository setup and local database configuration. The study identified substantial inefficiencies in agent exploration strategies, with a significant percentage of actions being unnecessary compared to optimal human behavior. These findings highlight gaps in current agents' practical environment-bootstrap capabilities.

However, the research does not investigate the inference efficiency of these models when deployed in resource-constrained environments or on CPUs, which is a critical gap in the current body of literature. Furthermore, their focus was mainly on cloud-based models and did not consider the potential environmental impact of running these models in cloud infrastructures, where energy consumption and carbon footprint can vary significantly depending on the hardware used.

D. Gap in Literature

While the literature provides a strong foundation for understanding the energy consumption and performance of deep learning models, particularly in large-scale environments using GPUs, there is limited research specifically addressing the trade-offs and performance of LLMs for code generation when executed in CPU-only environments. Most of the existing studies focus on training and GPU-based inference, overlooking the operational efficiency and sustainability of running LLMs in low-resource environments where only CPUs are available.

Energy-efficient deployment strategies using tools like MLFlow and CodeCarbon remain underexplored for LLMs, particularly in real-time inference tasks like code generation. This paper addresses this gap by comparing the CPU-based performance and energy efficiency of various LLMs, focusing on code generation and incorporating energy monitoring

through MLFlow and CodeCarbon to evaluate inference performance and environmental impact in resource-constrained environments.

IV. Experimental Setup

This section describes the setup for evaluating the performance, energy efficiency, and cost of various LLMs for code generation tasks. The models selected for the experiments are from OpenAI and Hugging Face repositories, with performance monitoring conducted using MLFlow and energy consumption tracking via CodeCarbon.

A. LLMs Selected for the Experiments

The models shown in Table I will be evaluated for code generation in C programming tasks from OpenAI and HuggingFace (via Novita as inference provider):

TABLE I: SELECTED LLMs FOR THE EXPERIMENTS.

From OpenAI	From Hugging Face
gpt-4o	mistralai/Mistral-7B-Instruct-v0.3
gpt-4-turbo	meta-llama/Meta-Llama-3-8B-Instruct
gpt-3.5-turbo	alpindale/WizardLM-2-8x22B
gpt-4o-mini	Qwen/Qwen3-235B-A22B-Instruct-2507

These models represent a range of architectures, including large-scale models like gpt-4o and optimized models like gpt-4o-mini.

B. Hardware and Operating System

- CPU: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz (2.42 GHz)
- Operating System: Windows 11 Pro

The experiments will be conducted on this CPU-based system, a typical environment for users without access to high-performance hardware like GPUs for inference tasks.

C. Performance Monitoring and Energy Consumption Measurement

To evaluate the energy consumption and carbon emissions, the following tools will be employed:

- MLFlow will be integrated to monitor and track the inference time, accuracy, and computational cost of each model.
- CodeCarbon will be used to track CO emissions and energy consumption for each inference task, helping assess the environmental impact of running LLMs.

D. Inference Tasks for the LLMs

The models will generate C source code for a set of problems, which are as follows:

- Prime Number Check: *Prompt: Write a C program that checks if a number is prime. The program validation returns 1 if it is prime and 0 if not. The number to check is 11.*
- Finding the Greatest of Three Integers: *Prompt: Write a C program that defines three integer variables and prints the greatest of them. The numbers to check are 11, 22, and 33.*
- Even/Odd Check: *Prompt: Write a C program that returns 1 if the input number is even, and 0 if it is odd. The input number for testing will be 122.*
- Absolute Difference Calculation: *Prompt: Write a C program that calculates the absolute difference between two numbers. The input numbers are -122 and 11.*
- Sum of Digits: *Prompt: Write a C program that calculates the sum of the digits of the input number. The input number is 123.*

Each task was repeated 10 times for each model to ensure that the results are statistically significant and to account for potential variations in model performance across multiple runs. The generated code will be validated using gcc to ensure correctness, and the inference time, accuracy, energy consumption, and computational cost will be tracked.

E. Evaluation Criteria

- Inference Time: Time taken by the model to generate the required C code.
- Code Generation Accuracy: Correctness of the generated code and whether it can be compiled and executed without errors.
- Energy Consumption: Measured using CodeCarbon to assess the energy used during inference.
- Computational Cost: The cost of running inference on remote models (via APIs for models like gpt-4o).

F. Experimental Phases

- Phase 1: Remote Inference via APIs: The first phase will focus on querying the models remotely using API calls (for models like GPT-4 and others from Hugging Face) and measuring inference time, accuracy, and energy consumption.
- Future Work - Phase 2: Local Inference on CPU: The second phase, which will be explored in future work, will involve deploying the LLMs locally on the CPU to assess their performance and energy efficiency in resource-constrained environments. This phase will compare the local CPU performance against the remote inference to evaluate trade-offs in energy efficiency and computational cost when running on CPUs.

This study will provide insights into the cost and environmental impact of deploying LLMs for code generation, particularly in scenarios where access to GPU resources is limited. The analysis will focus on the trade-offs between performance, energy efficiency, and cost, with the remote inference phase being the first step toward a more comprehensive study that will include local deployment on CPUs as future work.

V. Results

The results of our experiments are presented across three key categories: Performance and Throughput, Code Generation Accuracy, and Energy Efficiency. The models considered for this study are from both OpenAI and Hugging Face, with varying parameter sizes ranging from 3.8 billion to 235 billion parameters. Each model was evaluated based on its execution time, accuracy, energy consumption, and CO2 emissions, which are discussed in detail below.

A. Performance Throughput

The execution time of each model varied significantly, primarily due to the differences in model size and complexity. GPT-4o and GPT-4-turbo, the largest models in the study, had execution times of 2.3 minutes and 3.5 minutes, respectively. Despite optimizations in GPT-4-turbo, it required more time to complete the task, suggesting trade-offs between speed and accuracy.

In contrast, GPT-3.5-turbo was the fastest, taking only 1.9 minutes to generate code. However, this speed came at the cost of accuracy, as shown in the next section. The smaller model, GPT-4o-mini, took 2.8 minutes, slightly slower than GPT-4o, but still significant for its reduced size.

The smaller models from Hugging Face, including Mistral-7B and Meta-Llama-3-8B, took between 3.9 and 4.7 minutes for the task, which is relatively long compared to their smaller size. Finally, the largest models like WizardLM-2-8x22B and Qwen/Qwen3-235B took 7.8 minutes and 1.2 hours, respectively, showing the strong correlation between model size and execution time.

B. Code Generation Accuracy

The accuracy of the models in generating correct C code varied significantly, with the larger models performing better in generating valid code. GPT-4o achieved the highest accuracy of 54%, demonstrating its effectiveness in generating correct code for the given tasks. On the other hand, GPT-4-turbo showed a slight decrease in performance, with 36% accuracy, indicating the speed optimizations may have sacrificed some code generation quality.

GPT-3.5-turbo, with its smaller size, performed poorly, with an accuracy of 12%, reflecting the limitations of smaller models for such complex tasks. The smaller models, such as Mistral-7B and Meta-Llama-3-8B had accuracy rates of 6% and 18%, respectively, indicating that smaller parameter models struggle with generating accurate code. Larger models like WizardLM-2-8x22B and Qwen/Qwen3-235B both showed 17% accuracy, suggesting that despite their massive size, they also faced challenges in code generation.

C. Energy Efficiency

The energy consumption per inference varied based on model size, with larger models generally consuming more energy. GPT-4o and GPT-4-turbo consumed between 0.1–0.5 kWh, which is typical for models of their size and complexity. Interestingly, GPT-4o-mini, despite being smaller, consumed 0.003 kWh, slightly more than GPT-4o, likely due to specific optimizations and the inherent inefficiency of smaller models for complex tasks.

In contrast, smaller models like Mistral-7B and Meta-Llama-3-8B consumed 0.0015 kWh and 0.0028 kWh, respectively, indicating their efficiency relative to their size. However, larger models like WizardLM-2-8x22B and Qwen/Qwen3-235B consumed significantly more energy, with values of 0.0047 kWh and 0.0071 kWh, respectively, consistent with their massive size and computational demands.

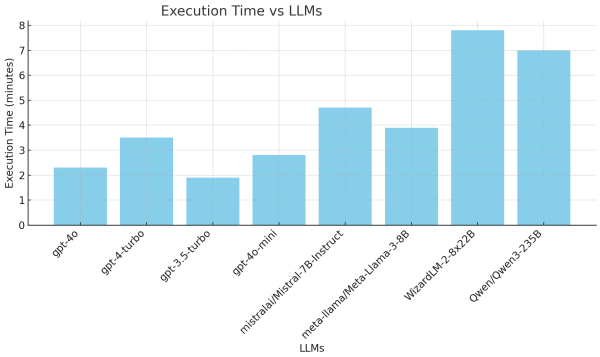

Figure 1: Execution Time vs LLMs.

CO2 emissions follow the same trend as energy consumption. GPT-4o generated 0.000132657 kg of CO2 per inference, while GPT-4-turbo emitted 0.000337222 kg. GPT-3.5-turbo produced 0.00044873 kg, further showing the inefficiency of smaller models in terms of their carbon footprint. Mistral-7B and Meta-Llama-3-8B had lower CO2 emissions of 0.000270392 kg and 0.000498069 kg, respectively, reflecting their lower energy consumption.

The largest models had the highest emissions: WizardLM-2-8x22B generated 0.001331077 kg and Qwen/Qwen3-235B generated 0.005493181 kg.

VI. Discussion And Evaluation

The results of our experiments provide valuable insights into the trade-offs between performance, accuracy, and energy efficiency when evaluating different LLMs for code generation tasks. Based on the execution time, accuracy, energy consumption, and CO2 emissions, we analyze the performance of the selected models and evaluate their practical application for real-world code generation tasks. This discussion will draw comparisons between the models and explore the implications of these results for both developers and environmental concerns.

A. Performance and Throughput

As shown in Figure 1, there is a clear correlation between model size and execution time. Larger models, such as GPT-4o and GPT-4-turbo require more time to generate code. Specifically, GPT-4o took 2.3 minutes per task, while GPT-4-turbo took 3.5 minutes. Although GPT-4-turbo is optimized for faster inference, its performance trade-offs manifest in a longer execution time compared to the base model. On the other hand, GPT-3.5-turbo is the fastest model at 1.9 minutes, but this speed comes at the cost of lower accuracy, as shown in the next section.

The smaller models like Mistral-7B and Meta-Llama-3-8B have execution times of 4.7 minutes and 3.9 minutes, respectively. Despite their smaller parameter sizes, they do not achieve significant speed advantages. In contrast, Qwen/Qwen3-235B and WizardLM-2-8x22B take the longest to execute, with 1.2 hours and 7.8 minutes, respectively, reflecting the computational burden of their massive parameter sizes.

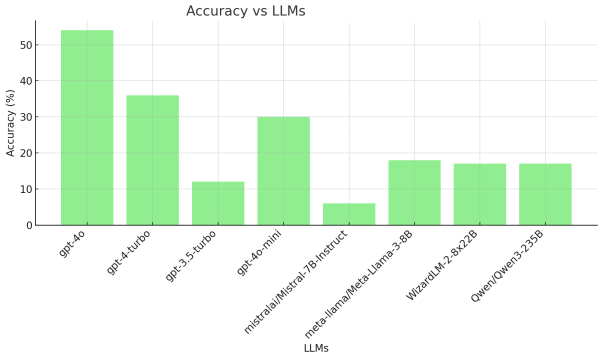

Figure 2: Accuracy vs LLMs.

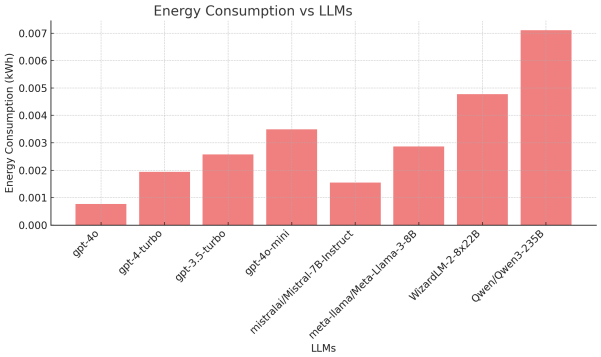

Figure 3: Energy Consumption VS LLMs.

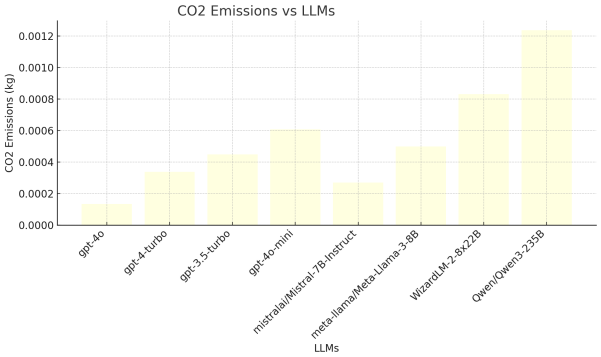

Figure 4: CO2 Emissions vs LLMs.

B. Code Generation Accuracy

The accuracy of code generation, as shown in Figure 2, is heavily influenced by model size. GPT-4o outperforms all other models with a 54% accuracy, highlighting its ability to understand and generate correct code. In contrast, GPT-4-turbo achieved a lower accuracy of 36%, which suggests that speed optimizations negatively impacted the model's ability to generate correct code.

The smaller models, such as GPT-3.5-turbo (12% accuracy), Mistral-7B (6% accuracy), and Meta-Llama-3-8B (18% accuracy) perform poorly in generating correct C code, which is expected due to their limited number of parameters and training data. Despite their reduced size, larger models like Qwen/Qwen3-235B and WizardLM-2-8x22B also showed relatively low accuracy (17%), indicating that even large models do not always excel in specialized tasks like code generation, which requires deep understanding of syntax and logic.

C. Energy Efficiency

When evaluating energy consumption (Figure 3) and CO2 emissions (Figure 4), we observe a direct correlation with the model's size and computational requirements. The larger models, such as GPT-4o and GPT-4-turbo consume between 0.1 and 0.5 kWh per inference, with GPT-4o using 0.000762179 kWh and GPT-4-turbo using 0.001937503 kWh. These higher consumption rates reflect the larger energy footprint of running complex models, particularly when deployed in cloud environments that require significant computing resources.

Smaller models, such as Mistral-7B and Meta-Llama-3-8B, use far less energy, consuming 0.001553531 kWh and 0.002861641 kWh, respectively. This demonstrates that smaller models are more energy-efficient, although their reduced size results in lower accuracy for code generation tasks. While the smaller models are more energy-efficient, their performance is not optimal for generating high-quality code, making them less suitable for complex software development tasks.

The larger models like Qwen/Qwen3-235B and WizardLM-2-8x22B consume significantly more energy, with Qwen/Qwen3-235B using 0.007103992 kWh and WizardLM-2-8x22B using 0.004770831 kWh. These models have the highest CO2 emissions per inference, with Qwen/Qwen3-235B producing 0.005493181 kg of CO2 and WizardLM-2-8x22B producing 0.001331077 kg of CO2. The high energy consumption and emissions of these large models suggest that while they may have certain advantages in scale, they are less efficient for deployment in resource-constrained or environmentally conscious environments.

D. Implications for Practical Use

The results indicate that larger models like GPT-4o offer the best performance and accuracy but at the expense of higher energy consumption and environmental impact (if we consider the approximate energy consumption during the training phase). These models are suitable for applications where accuracy is the primary concern, and there are sufficient computational resources. However, their high energy consumption makes them less ideal for environmentally conscious or resource-limited environments.

On the other hand, smaller models like Mistral-7B and Meta-Llama-3-8B are more energy-efficient but suffer from significantly lower accuracy, making them less suitable for complex tasks such as code generation. GPT-4o-mini shows promise with a balance of moderate energy consumption and relatively good accuracy at 30%. These models could be a viable option when moderate performance is acceptable, and energy efficiency is prioritized.

VII. Conclusion and Future Work

This study provides a comprehensive evaluation of various LLMs for code generation tasks, focusing on performance, accuracy, energy efficiency, and environmental impact. The results reveal several key insights:

1) Larger models, such as GPT-4o and GPT-4-turbo offer superior accuracy but require significantly more execution time and consume higher amounts of energy, leading to increased CO2 emissions. These models are optimal for tasks where high accuracy is crucial, but their environmental impact makes them less suitable for resource-constrained environments.
2) Smaller models like Mistral-7B and Meta-Llama-3-8B offer better energy efficiency and lower CO2 emissions, but their accuracy is considerably reduced. These models may be suitable for scenarios where energy consumption is a priority and moderate performance is acceptable.
3) Models like GPT-4o-mini, with a balance of moderate energy consumption and reasonable accuracy, could serve as a compromise between large models and smaller, more efficient models.

In conclusion, the performance of LLMs in code generation tasks is a delicate balance between accuracy, energy consumption, and environmental impact. The choice of model should depend on the specific use case, with larger models being preferred for high-accuracy requirements, while smaller models offer better energy efficiency for more environmentally-conscious applications.

Future Work

Building on the insights from this study, several lines of future research are planned:

- Expansion to Other LLMs: The experiments can be extended to other LLMs beyond those evaluated in this study. Newer models or those from different providers may offer improvements in performance, energy efficiency, and environmental impact that could alter the current conclusions.
- Evaluation on Other Software Development Tasks: While the current study focused on code generation tasks in C programming, future experiments will expand to other types of software development tasks such as debugging, code optimization, and automatic code refactoring. This will help assess whether the trade-offs observed in this study hold true for a wider range of software engineering tasks.
- Incorporating Local Deployment: The main future direction involves repeating the experiments but this time using local LLMs. This will involve deploying the models on both local CPUs and local GPUs. By doing this, we can compare the performance and energy consumption when models are running locally, with the aim to identify the most efficient configurations for resource-constrained environments. The availability of local GPUs could offer a significant improvement in execution time and energy consumption, making it a valuable area of exploration.
- Optimization of Inference Efficiency: Future work will also include optimizing inference strategies for energy consumption and accuracy. Investigating different quantization, pruning, and distillation methods could provide potential pathways for improving the efficiency of LLMs without compromising their performance.

By addressing these future directions, the research will not only provide further insights into the trade-offs between performance and efficiency but also contribute to the development of more sustainable AI practices, particularly in the context of code generation and software development.

Acknowledgements

The authors are part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1519-22), supported by the Department of Education, Universities and Research of the Basque Country. This research was supported by the Ikerketa Taldeak funding (IT1519-22) and the GRECO Elkartek project (KK-2024/00090), both funded by Eusko Jaurlaritza.

References

[1] E. Strubell, A. Ganesh, and A. McCallum, *Energy and policy considerations for deep learning in nlp*, (visited on 09/03/2025), 2019. arXiv: 1906.02243 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1906.02243 (visited on 09/03/2025).

[2] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, *Green AI*, (visited on 09/03/2025), 2019. arXiv: 1907.10597 [cs.CY]. [Online]. Available: https://arxiv.org/abs/1907.10597 (visited on 09/03/2025).

[3] D. Patterson *et al.*, *The carbon footprint of machine learning training will plateau, then shrink*, (visited on 09/03/2025), 2022. arXiv: 2204.05149 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2204.05149 (visited on 09/03/2025).

[4] T. B. Brown *et al.*, *Language models are few-shot learners*, (visited on 09/03/2025), 2020. arXiv: 2005.14165 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.14165 (visited on 09/03/2025).

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, (visited on 09/03/2025), 2019. arXiv: 1810.04805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805 (visited on 09/03/2025).

[6] M. Chen *et al.*, *Evaluating large language models trained on code*, (visited on 09/03/2025), 2021. arXiv: 2107.03374 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2107.03374 (visited on 09/03/2025).

[7] J. Xu, W. Zhou, Z. Fu, H. Zhou, and L. Li, *A survey on green deep learning*, (visited on 09/03/2025), 2021. arXiv: 2111.05193 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2111.05193 (visited on 09/03/2025).

[8] D. Patterson et al., *Carbon emissions and large neural network training*, (visited on 09/03/2025), 2021. arXiv: 2104.10350 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2104.10350 (visited on 09/03/2025).

[9] MLCO2, *Codecarbon*, (visited on 09/03/2025), 2025. [Online]. Available: https://github.com/mlco2/codecarbon (visited on 09/03/2025).

[10] Y. Rangineeni and J. Pub, "End-to-end mlops: Automating model training, deployment, and monitoring", *Journal of Recent Trends in Computer Science and Engineering (JRTCSE)*, vol. 7, pp. 60–76, Sep. 2019.

[11] A. Arora, J. Jang, and R. Zilouchian Moghaddam, *Setupbench: Assessing software engineering agents' ability to bootstrap development environments*, (visited on 09/03/2025), 2025. arXiv: 2507.09063 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2507.09063 (visited on 09/03/2025).