EcoCode CLI: A Lightweight Approach to Sustainable Model Selection in Software Engineering

Miguel El Khal, Aiman Abdul Wahab, Carolyn Alcaraz, Ceylin Ece, Gonenc Turanli
TU Delft
Sustainable Software Engineering

***Abstract*—This paper presents the idea, implementation, and evaluation of EcoCode CLI, a tool designed to encourage more sustainable model choices in AI-assisted software engineering.**

CONTENTS

I. INTRODUCTION

Large language models (LLMs) are now widely used in software engineering. Developers use AI assisted tools for tasks such as writing code, debugging, explaining code, and improving documentation. Although these tools can make development faster and more convenient, their increasing use also raises sustainability concerns. In many cases, large and resource intensive models are used even when the task is simple and could be handled by a smaller model. This can result in unnecessary computational cost and higher energy use during everyday development work.

This issue is especially relevant in AI-assisted software engineering, where model choice is often not visible to the user or is treated mainly as a performance decision. In practice, developers are rarely given clear support for choosing the most suitable model for a task. As a result, larger models may be used more often than necessary. From a sustainability perspective, this matters because even small inefficiencies can build up when AI tools are used repeatedly across many software engineering tasks.

This paper focuses on that problem and proposes *EcoCode CLI*, a lightweight command line tool that helps developers choose a more suitable model tier for a given coding task. Instead of always relying on the largest available model, the tool aims to guide users toward lighter alternatives when possible. The goal is to support more sustainable use of AI in software engineering without reducing practicality or requiring major changes to the developer workflow.

A. Background

AI-assisted tools are becoming a regular part of software development practice. Large language models are now used for a range of tasks, including code generation, debugging, explanation, and documentation support [1], [4]. However, not all of these tasks require the same level of model capability. Some tasks may be handled effectively by smaller models, while more complex tasks may benefit from larger ones. Because larger models generally require more computational resources, using them by default for all tasks may lead to unnecessary energy use [14]. For this reason, model selection is an important issue when considering sustainability in AI-assisted software engineering [2], [6].

B. Related Work

Related work on this topic comes from sustainable software engineering, AI-assisted software development, and developer support tools. In sustainable software engineering, researchers have argued that sustainability should be treated as an important concern throughout software engineering practice, rather than only as a hardware or infrastructure issue [2], [3]. At the same time, AI coding tools such as GitHub Copilot have made large language models more visible in everyday development workflows, supporting tasks such as code generation, explanation, and editing [4], [5].

This idea of lightweight developer guidance is also reflected in static code analysis tools such as SonarQube, ESLint, and Pylint [13], [15], [16]. These tools help developers identify issues related to code quality, correctness, security, and maintainability during development. In that sense, they are similar to EcoCode CLI because both aim to support better decisions within the developer workflow. However, the focus of EcoCode CLI is different. Static analysis tools evaluate the source code itself, whereas EcoCode CLI evaluates the prompt and recommends a more suitable AI model tier before code is generated. Rather than improving the quality of the written code directly, EcoCode CLI focuses on improving the sustainability of the AI-assisted development process.

Recent work has also started to examine the energy efficiency of LLM based code generation. Studies have shown that although LLMs can generate functionally correct code, the produced solutions are not always as energy efficient as human written alternatives [6]. Other work has explored ways to reduce the energy cost of code generation models themselves, for example through energy aware inference methods [7]. Together, these areas of work show growing interest in both sustainable software engineering and responsible AI use, but they also suggest that sustainability oriented model selection remains relatively underexplored in everyday developer tools.

C. Motivation

The motivation for this project comes from the increasing use of AI tools in software development and the limited support for using them more sustainably. In many cases, developers may choose larger models by default, even when a smaller model could provide a sufficient response. This can lead to unnecessary resource use, especially when such tools are used frequently. Our project is based on the idea that sustainability can also be improved through better developer support rather than only through changes to infrastructure or model design.

To address this, we propose EcoCode CLI, a tool that helps developers choose more appropriate model tiers for software engineering tasks. The goal is to encourage the use of lighter models when possible, while still supporting useful AI-assisted development. In this paper, we describe the sustainability problem addressed by the project, present the proposed solution and its implementation, and discuss how the tool can be evaluated in practice.

II. Problem Description

The main problem addressed in this project is the unnecessary use of large language models for routine software engineering tasks. In practice, developers often use powerful AI models by default, even for simple requests such as explaining code, making small edits, or generating basic documentation. While this may be convenient, it can also lead to unnecessary computational cost and higher energy use. Because developers are usually not given clear guidance on which model is most suitable for a task, they may rely on larger models even when smaller and more efficient alternatives would be sufficient.

One reason this problem continues is that model choice is often invisible in everyday development workflows. Many AI tools are designed for convenience, but they do not help users understand the trade off between model capability and resource use. As a result, sustainability is rarely considered when a developer submits a prompt. This creates a gap between the increasing use of AI in software engineering and the need to use these systems in a more efficient and responsible way.

A. Context

This problem appears in a software engineering environment where AI tools are becoming part of everyday development practice. Developers use large language models for many different tasks, including generating code, explaining existing code, debugging errors, and revising small sections of text or documentation. These tasks can vary greatly in complexity, yet the same model may still be used for all of them by default. In this context, choosing a model is not only a technical decision but also a sustainability related one, since different model sizes can require different amounts of computational resources.

B. Challenges

A key challenge in this project is estimating the complexity of a developer task in a way that is both simple and practical. Software engineering tasks can vary widely, so it is not always easy to decide whether a task requires a larger model or whether a smaller model would be sufficient. In this project, one practical indicator is the total input length, which combines the length of the prompt with the length of any relevant file context. This is used as a simple proxy for task complexity and computational demand.

Recent work suggests that input length can influence LLM performance and reasoning behaviour, and that shorter prompts may be more effective for simpler tasks, while longer inputs can sometimes introduce unnecessary complexity or noise [8]–[10], [12]. However, total length is not a direct indicator of task difficulty. For example, a simple request may refer to a large source file, while a short prompt may still describe a difficult problem. For this reason, our project uses total input length as one factor for estimating complexity, while recognizing that it does not fully determine how difficult a task really is.

III. Proposed Solution

This section introduces the proposed solution, *EcoCode CLI*, and outlines how it guides developers toward more sustainable AI model choices.

EcoCode CLI addresses this challenge by introducing a lightweight command-line tool designed to assist developers in selecting the most energy-efficient AI model for code generation tasks. The tool adapts model recommendations based on the estimated complexity of a given task prompt, thereby balancing performance and energy consumption.

Given a developer prompt, the tool performs a lightweight analysis to estimate task complexity. Based on this analysis, the tool will then output the following:

- Task complexity classification
- Recommended model tier
- Estimated energy consumption
- Baseline energy usage of the large model
- Estimated energy savings
- Token usage

To evaluate the effectiveness of *EcoCode CLI*, two primary dimensions are considered: the quality of the generated code changes and the estimated energy consumption. Code quality is assessed by comparing the generated file output against the reference patch from SWE-bench dataset using file matching metrics. Energy consumption is estimated from model selection and token usage, allowing the system to be compared against a baseline that always uses the largest model.

IV. Methodology

A. Approach

This work adopts a design-oriented approach to develop a lightweight decision system for sustainable model selection. The key idea is that different software engineering tasks, depending on how difficult they may be, could make use of smaller, more energy-efficient models when appropriate. This could reduce unnecessary energy consumption.

A heuristic-based routing strategy, instead of a learned model, is chosen to ensure interpretability, low computational overhead, and ease of integration into existing workflows. This strategy also allows the tool to make fast decisions without adding to the computational cost and potentially even undermining its sustainability goals.

Tasks are classified into three levels of complexity (easy, medium, and hard), which are then mapped to their corresponding model tiers (small, medium, and large). This reflects the differences in model capability and computational costs.

This approach balances usability, performance and sustainability while minimizing disruption to developer workflows.

B. Dataset

TABLE I
EcoCode routing: difficulty, model, samples, and energy cost.

Diff.	Model ID	n	Energy (kWh/token)
Easy	gemini/gemini-2.5-flash	13	4.0×10^{-7}
Medium	openrouter/meta-llama/llama-3.3-70b-instruct	21	5.9×10^{-7}
Hard	cerebras/qwen-3-235b-a22b-instruct-2507	5	4.35×10^{-6}

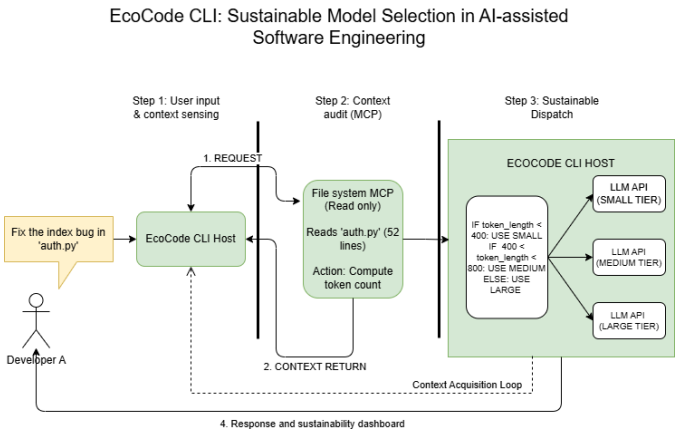

Fig. 1. EcoRoute design overview

To evaluate EcoCode CLI, tasks from the SWE-bench Lite dataset [17] were used. This SWE-bench Lite dataset contains real-world software engineering issues derived from GitHub repositories.

SWE-bench Lite is well-suited for this study because it provides realistic and diverse tasks with varying levels of complexity, allowing for the evaluation of how the system behaves under different practical conditions.

In this project, the dataset is also used as the basis for evaluation by comparing the code produced by the selected model against the reference patch provided in the benchmark. The evaluation focuses on how closely the generated file changes match the expected solution.

V. Implementation

A. Design

EcoCode CLI is implemented as a lightweight command-line tool that analyzes developer prompts and recommends an appropriate AI model tier based on estimated task complexity. The system operationalizes the methodology described in the previous section by implementing a modular pipeline consisting of four main stages: input processing, feature extraction, complexity assessment, and model recommendation.

To support the energy-aware routing strategy introduced in the methodology, the system incorporates a simplified energy estimation model. Each of the three model tiers is assigned a relative energy cost, enabling the system to compute estimated savings when a smaller model is opted for instead of the larger baseline model that would typically be called.

B. Development

The system was implemented in Python and consists of three main components: the core analysis logic, the command-line interface and the execution module for applying code transformations as suggested by the model.

1) Core Analysis Logic: The core analysis logic unit is responsible for analyzing input prompts and generating recommendations. The input includes the problem description and contextual information such as referenced files. From this

input, several features are extracted, including prompt token count, presence of file references, and file context size (if a file is provided).

Task complexity is determined using heuristic thresholds based on the extracted features, such as token length and length of a provided file.

To better approximate real input complexity, the total token count is computed as a weighted sum:

$$T_{total} = T_{prompt} + 0.3 \cdot T_{file} \tag{1}$$

This weighting reflects the assumption that file content contributes less to the complexity than the original prompt.

The model then classifies the task into one of the three categories using fixed thresholds that have taken inspiration from Levy et. al [18]:

- Easy: $T_{total} < 200$
- Medium: $200 \leq T_{total} < 800$
- Hard: $T_{total} \geq 800$

Model selection is then directly mapped from the task complexity following the rules:

- Easy $\rightarrow$ Small model
- Medium $\rightarrow$ Medium model
- Hard $\rightarrow$ Large model

a) Energy estimation: To quantify the sustainability impact, the system makes use of a simplified energy model based on token usage, with each model tier being assigned a relative energy cost per token.

The total energy consumption is estimated as:

$$E = T_{total} \cdot C_{model} \tag{2}$$

Where C_{model} represents the per-token energy cost of the selected model.

Energy savings are then computed by comparing the energy usage of the selected model against that of the large-model baseline:

$$E_{saved} = E_{baseline} - E_{selected} \tag{3}$$

This allows the system to report on both the absolute energy savings and the percentage reduction.

2) Command-Line Interface: The CLI component provides the user-facing interface of *EcoCode*. It accepts a prompt as input and displays a structured dashboard that shows the estimated task complexity, recommended model tier, estimated energy consumption, and potential savings. The interface emphasizes transparency, enabling developers to understand the trade-offs between model choices.

3) Code Transformation Module: In addition to sustainable model recommendation, the system includes an execution module that can apply model-driven code transformations directly onto source files. This module follows a read–modify–write loop, where a file is read, processed by a language model, and written back to source file.

The implementation makes use of an external model interface and a structured prompt system to ensure that only valid source code is returned, including additional safeguards such as cleaning model outputs to remove formatting artifacts and handling API errors during execution and any malformed outputs.

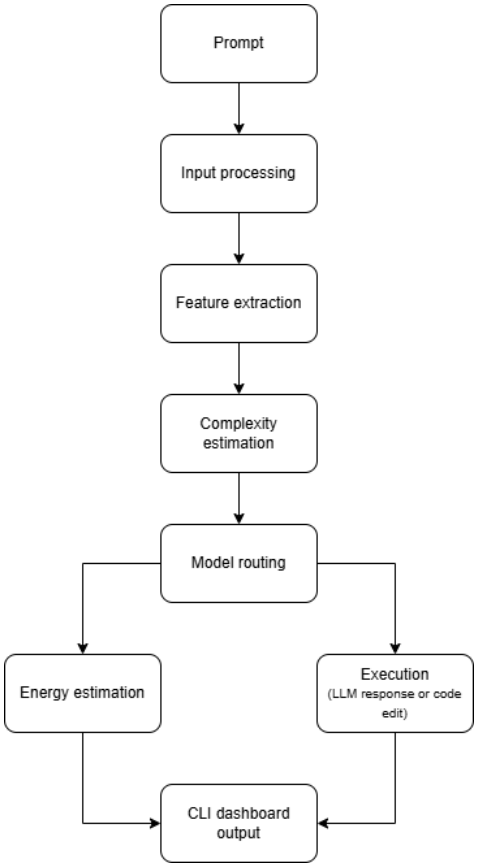

Fig. 2. System pipeline

This module enables practical integration into development workflows beyond just sustainable model recommendation.

To validate the basic functionality of this module, simple internal checks were used during development to ensure that generated or modified code could be processed correctly by the system pipeline. However, the main evaluation of EcoCode CLI is based on file-level comparison against the reference patch from SWE-bench Lite rather than on unit-test-based assessment.

Overall, the implementation demonstrates a system that combines prompt analysis, heuristic decision-making, and energy estimation to support more sustainable AI usage in software engineering. Figure 2 below shows the overall pipeline EcoCode CLI follows.

VI. Validation

This section explains how EcoCode CLI is evaluated in terms of both output quality and sustainability impact. The goal of the evaluation is to determine whether the routing strategy can reduce estimated energy use without causing a large drop in the quality of the generated fixes.

A. Testing Strategy

The evaluation compares two modes: EcoCode CLI and a baseline that always selects the largest model. For each task in the evaluation set, the system generates a file-level repair based on the task description, relevant context, and the target source file. The generated output is then compared against the reference patch provided in the SWE-bench Lite dataset.

The evaluation is based on file matching metrics rather than unit tests. This means that the generated file is assessed according to how closely its changes match the expected patch. In this way, the evaluation focuses on whether the model reproduced the intended fix and whether it introduced unnecessary changes.

B. Metrics

Three main metrics are used in the evaluation. The first is *Patch F1*, which measures the overlap between the lines added by the model and the lines added in the reference patch. The second is *Hunk BLEU*, which measures similarity between the generated output and the expected output within the modified regions of the file. The third is *Exact Match*, which is recorded when the generated file reproduces the reference additions perfectly.

In addition to these quality metrics, the evaluation also records token usage and estimated energy consumption. This makes it possible to compare EcoCode CLI against the baseline not only in terms of output similarity, but also in terms of sustainability.

C. Results

The evaluation shows that EcoCode CLI was able to reduce estimated energy consumption while maintaining competitive output quality compared with the always-large baseline. Across the evaluated SWE-bench Lite tasks, As shown in Table II, EcoCoute CLI reduced estimated energy consumption by 45.2% while improving Patch F1 and Exact Match performance. This suggests that the routing strategy was effective in selecting lighter model tiers when appropriate, which directly supports the main sustainability goal of the project.

At the same time, the quality related results indicate that this reduction in energy use did not come at the cost of worse file level repairs. EcoCode achieved a 28.8% improvement in Patch F1, which suggests that the generated edits more closely matched the expected reference patches and contained fewer unnecessary changes. In addition, the system achieved a 50% improvement in Exact Match, indicating that more tasks were solved with a perfect reproduction of the expected fix.

The only metric that showed a small decrease was Hunk BLEU, which fell by 4.3%. However, this drop was relatively minor compared with the gains in Patch F1 and Exact Match. This suggests that although the generated outputs may have been slightly less similar in local text form, the overall fixes were still more precise and more aligned with the intended patch.

Overall, these results suggest that EcoCode CLI is able to support more sustainable model usage while preserving, and in some cases improving, repair quality. This aligns with the aim of the project, which is to guide developers toward lighter model choices without causing a major reduction in practical usefulness.

TABLE II
SUMMARY OF ECOCOUTE CLI COMPARED WITH THE ALWAYS-LARGE BASELINE

Metric	**Change**
Energy	45.2% saved
Patch F1	+28.8%
Exact Match	+50.0%
Avg BLEU	-4.3%

These findings should be interpreted in light of the selected evaluation subset and the use of file matching metrics, but they provide encouraging evidence that sustainability aware model routing can reduce energy use without significantly harming repair quality.

VII. DISCUSSION

A. Strengths and Limitations

EcoCode CLI shows that sustainable model selection can be integrated into developer workflows with minimal disruption. The system operates as a lightweight command-line tool that requires minimal change in how developers work and formulate prompts, making it practical for real-world adoption.

Another strength of the approach lies in the interpretability of the tool. By relying on a heuristic-based routing strategy instead of learned approaches, the system provides clear and understandable reasoning behind its decisions. With this tool, developers can see how task complexity affects the model selection, therefore increasing the transparency of the system.

In addition to being able to see how task complexity affects the selected model, the tool also introduces energy-awareness directly into the development process. As a result, instead of having sustainability be an afterthought or treating it as an infrastructure concern, the tool incorporates sustainability into everyday developer decisions and encourages more responsible AI usage in software development.

Like with any other system, EcoCode CLI also has limitations that are important to acknowledge.

Firstly, task complexity is estimated primarily based on prompt length and simple heuristics. Although practical, this approach does not capture the semantic complexity of a task, and may thus lead to misclassifications. For example, a complex task may be formulated in a short and concise prompt or a simple task may be formulated in a long prompt with a lot of unnecessary tokens.

Secondly, the energy model is based on relative estimates and does not reflect precise real-world measurements. Consequently, the reported energy savings do not capture other factors such as hardware differences, runtime conditions, and model implementation details. The estimations should therefore only be interpreted as approximate trends rather than exact values, limiting their reliability for precise quantitative analysis.

Finally, the evaluation is limited to a selected subset of SWE-bench Lite tasks due to API cost and practical runtime constraints. The subset and the models are found in table 1. The current setup also only evaluates single-file fixes and excludes very large files, which narrows the range of tasks that can be assessed. In addition, because the evaluation is based on file level matching against reference patches, it does not fully capture all aspects of behavioural correctness. Although SWE-bench Lite provides realistic software engineering issues, broader validation across more repositories, multi file fixes, different programming languages, and a wider range of task types would be needed to fully assess the generalizability of the approach.

B. Future Work

There are several ways EcoCode CLI could be improved and extended on.

One direction could be to explore more advanced methods for estimating task complexity, such as semantic analysis, lightweight machine learning models, or intent detection. This could improve the accuracy of routing decisions beyond simple heuristics based on input length.

Another improvement would be to integrate real energy measurement tools or more accurate energy estimation tools that would allow the system to provide more reliable and validated sustainability metrics.

Finally, A potential extension to this work could be to attempt to integrate EcoCode CLI into existing development environments through, for example, IDE plugins and extensions. This would drastically improve on the usability and adoption of the tool.

VIII. Conclusion

This paper introduced EcoCode CLI, a lightweight tool for sustainable model selection in AI-assisted software engineering. The system addresses the problem of unnecessary reliance on large and energy-intensive models for routine development tasks when a smaller, more energy-efficient model would have been more appropriate. The results demonstrate that sustainability can be made more of an active choice and can be incorporated into AI-assisted workflows without compromising usability.

The tool is lightweight and adopts a heuristic-based routing mechanism, enabling developers to make use of appropriate model tiers based on estimated task complexity. Additionally, the tool provides energy-aware feedback, highlighting potential energy savings when a smaller model is used instead of the larger baseline model.

While the current approach relies on simplified assumptions, it provides a practical starting point for exploring how AI tools can be used more sustainably and contributes to the growing area of sustainable software engineering. This is especially relevant as AI tools become increasingly integrated into software engineering and the need for efficient and responsible model usage becomes essential.

Acknowledgment

The authors would like to thank the course staff and reviewers for their guidance and feedback throughout the project.

IX. GitHub repository

Presented here is the link to the public GitHub repository: https://github.com/Aiman-prog/ecocode

References

[1] A. Fan, B. Johnson, L. Ma, A. Serebrenik, and M. Harman, "Large Language Models for Software Engineering: Survey and Open Problems," *arXiv preprint arXiv:2310.03533*, 2023.

[2] C. König, "Sustainable Software Engineering: Concepts, Challenges and Future Research Directions," *ACM SIGSOFT Software Engineering Notes*, 2025.

[3] B. C. Mourão and others, "Green and Sustainable Software Engineering," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018.

[4] GitHub, "GitHub Copilot: Your AI pair programmer," [Online]. Available: https://github.com/features/copilot

[5] GitHub Docs, "AI model comparison," [Online]. Available: https://docs.github.com/en/copilot/reference/ai-models/model-comparison

[6] M. A. Islam et al., "Evaluating the Energy-Efficiency of the Code Generated by LLMs," arXiv preprint arXiv:2505.20324, 2025.

[7] S. Ilager, L. F. Briem, and I. Brandic, "GREEN-CODE: Optimizing Energy Efficiency in Large Language Models for Code Generation," arXiv preprint arXiv:2501.11006, 2025.

[8] Q. Liu, W. Wang, and J. Willard, "Effects of Prompt Length on Domain-specific Tasks for Large Language Models," *arXiv preprint arXiv:2502.14255*, 2025.

[9] M. Levy, A. Jacoby, and Y. Goldberg, "Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models," *arXiv preprint arXiv:2402.14848*, 2024.

[10] Prompt Engineering, "The Power of Concise Prompts in Large Language Models," online article, 2024.

[11] S. Hulse, "More Words, Less Accuracy: The Surprising Impact of Prompt Length on LLM Performance," *Grit Daily News*, 2024.

[12] PromptPanda, "Maximizing AI Outputs: Strategies for AI Prompt Length Optimization," online article, 2024.

[13] SonarSource, "SonarQube Server analysis overview," [Online]. Available: https://docs.sonarsource.com/sonarqube-server/analyzing-source-code/analysis-overview

[14] J. Fernandez, M. Morrow, N. Schrana, V. Krishnan, and E. Bashir, "Energy Considerations of Large Language Model Workloads," *arXiv preprint arXiv:2504.17674*, 2025.

[15] ESLint, "Find and fix problems in your JavaScript code," [Online]. Available: https://eslint.org/

[16] Pylint, "Frequently Asked Questions – What is Pylint?," [Online]. Available: https://docs.pylint.org/faq.html

[17] Jimenez, C., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O. & Narasimhan, K. SWE-bench: Can Language Models Resolve Real-world Github Issues?. *The Twelfth International Conference On Learning Representations*. (2024), https://openreview.net/forum?id=VTF8yNQM66

[18] Levy, M., Jacoby, A. & Goldberg, Y. Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models. (2024), https://arxiv.org/abs/2402.14848