Estimating and Visualizing Energy Consumption of LLMs within IDEs: A VS Code Extension for GitHub Copilot

Gyum Cho, Cyprian Bîcă, Denis Krylov, Matteo Fregonara

Abstract

The rapid integration of large language models (LLMs) into software development, exemplified by tools like GitHub Copilot, has enhanced productivity but raised concerns about their substantial energy consumption. This study introduces a Visual Studio Code (VS Code) extension designed to estimate and visualize the energy use and carbon dioxide equivalent (CO_2eq) emissions of AI-generated code suggestions in real time. Focusing on GitHub Copilot, the extension employs a token-based approach, leveraging publicly available data to approximate energy costs (e.g., 2.16 J/token for GPT-4o) and emissions (e.g., 2.14×10 gCO_2eq/J). By embedding these metrics into the integrated development environment (IDE) workflow via status bar updates and detailed logging, the tool reveals the environmental footprint of AI assistance, with energy estimates ranging from 43.20 J for simple loops to 745.20 J for complex neural network implementations. While demonstrating the feasibility of such monitoring, limitations include reliance on aggregated data and lack of model-specific energy profiles beyond GPT-4o. This work underscores the need for transparency in AI energy use and lays the groundwork for sustainable coding practices by balancing LLM benefits with their ecological impact.

1 Introduction

The rise of artificial intelligence (AI), particularly LLMs, has introduced unprecedented capabilities to computing, but at a significant energy cost. A prominent example is ChatGPT, developed by OpenAI, which exemplifies the energy-intensive nature of LLMs. Estimates of ChatGPT's energy consumption per query vary widely, ranging from 0.3 watt-hours (Wh) [1] to 2.9 Wh [2]. The higher estimate, from a 2023 study by de Vries, suggests that a single ChatGPT query consumes nearly ten times the energy of a typical Google search, which requires approximately 0.3 Wh [3]. This disparity underscores the growing energy footprint of AI as its applications expand in scale and complexity. This energy intensity, often concealed within power-hungry data centers, remains largely invisible to users, making it difficult to grasp the environmental and economic toll of AI-driven tools. While research has started to quantify AI's energy footprint, few studies explore its implications for developer tools or offer practical solutions. This gap inspires our study, which aims to shed light on LLM energy use in IDEs by quantifying and visualizing it in real time.

In software development, LLMs have become increasingly prevalent, driven by tools like GitHub Copilot, an AI-powered code completion and generation tool developed by GitHub and OpenAI. Integrated into various IDEs, GitHub Copilot enables users to select from multiple LLMs—specifically ChatGPT-4o, Claude 3.5 Sonnet, and Gemini 2.0 Flash—to generate code tailored to their needs. A 2023 GitHub survey found that 92% of U.S.-based developers used AI-powered coding tools at least occasionally, leveraging them for tasks such as code generation, autocompletion, and debugging [4]. Similarly, the Stack Overflow 2023 Developer Survey reported that 70% of professional developers had experimented with AI tools, including LLMs, with 43% incorporating them into their weekly or daily workflows [5]. These findings highlight the rapid adoption of LLMs, transforming how developers approach coding tasks and boosting productivity.

This paper proposes a novel solution: an IDE extension that embeds real-time energy usage metrics for LLMs. By making energy consumption visible to developers, this tool aims to foster awareness and encourage sustainable coding practices. The study focuses on designing and implementing this extension to measure the energy consumed by LLMs per user request during code generation. Key takeaways from this work include: (1) the feasibility of using a token-based approach to approximate server-side energy use, (2) the successful integration of real-time energy metrics into a widely used IDE, and (3) the demonstration that even simple AI queries contribute meaningfully to energy consumption. These findings highlight the potential for practical tools to bridge the gap between AI innovation and environmental responsibility, while also revealing the need for greater transparency from LLM providers to refine such estimations.

The paper is organized as follows. Section 2 provides background on AI energy consumption and its adoption in software development. Section 3 outlines the methodology employed in this study. Sections 4 and 5 present the results and a discussion of findings, culminating in conclusions in Section 6.

The code for the extension can be found in the following GitHub repository.

2 Background

This section provides foundational context on the key technologies and concepts relevant to understanding the energy consumption of AI-assisted software development, setting the stage for the methodology described later.

Available Code Editors

The software development landscape features a diverse range of code editors and IDEs. Prominent examples include Visual Studio, VS Code, the JetBrains suite (e.g., PyCharm, IntelliJ IDEA), Sublime Text, Android Studio, and Xcode. Selecting an appropriate IDE for research involving extension development requires considering factors such as cross-platform compatibility, underlying architecture, richness of the plugin ecosystem, community adoption size, and active maintenance status. Architecturally, these environments vary significantly: Visual Studio is historically tightly integrated with the .NET framework and Windows ecosystem [10]; VS Code utilizes the Electron framework, enabling a cross-platform, modular, and relatively lightweight structure [8]; the JetBrains IDEs are primarily Java-based, built upon the IntelliJ platform [11]; Sublime Text employs a custom, performance-optimized proprietary engine [12]; while Xcode and Android Studio are deeply integrated with the Apple and Google development platforms, respectively [13, 14]. These architectural differences influence performance, extensibility, and suitability for specific development tasks and research goals.

Visual Studio Code

VS Code was chosen to create the energy monitoring extension for this study. This decision was based on several key advantages of VS Code. First, compared to Visual Studio or IntelliJ-based IDEs, VS Code is more lightweight and runs reliably on Windows, macOS, and Linux. Second, because it's built on the Electron framework, it makes developing extensions using common web technologies (JavaScript or TypeScript) easier. VS Code also offers a large marketplace with many extensions, showing it has a large and active support community. Its system for extensions, using the Language Server Protocol (LSP) and a well-documented VS Code API [9], offered an easier way to track the necessary editor events compared to the possibly more complex plugin systems of other IDEs. Microsoft and a strong open-source community actively maintain VS Code, so it is continuously updated. Finally, many developers use it, especially for web development and scripting, which makes our results relevant to a broad audience.

AI Assistant

Visual Studio Code itself does not include native AI-driven code generation features; such functionality necessitates the installation of third-party extensions. In selecting an AI assistant for integration and monitoring, two prominent options were evaluated: GitHub Copilot and the open-source alternative, CodeGPT. While both provide core AI-assisted coding capabilities—including code completion, generation from prompts, and debugging support—they exhibit differences in performance characteristics, the quality of integration with VS Code, and underlying resource requirements. Following a comparative assessment, GitHub Copilot was chosen for this study. The rationale for this selection was its demonstrably superior contextual accuracy in generating relevant code suggestions and its seamless, robust integration within the VS Code environment. This aligns directly with the research goal of measuring energy consumption within a realistic and widely adopted developer workflow. Furthermore, while GitHub Copilot offers access to various LLMs (like Claude and Gemini alongside GPT models), our energy estimation focuses specifically on ChatGPT-4o because publicly available data allowed us to approximate both its energy consumption profile and its likely tokenizer Tiktoken. Similar public data for approximating these factors was not readily available for the Claude or Gemini models offered within Copilot at the time of this research.

3 Methodology

This section describes the methodology used to monitor and estimate the energy consumption of AI-generated code suggestion within a modern IDE. We begin by outlining the overall system architecture, including the mechanisms for identifying AI-Generated text and capturing code changes. The detail of the approach used for token counting and the estimation of energy and CO_2 equivalent emissions based on those tokens. Finally, there will be an explanation how this data is presented to the user in real time through the IDE interface, and how it is persistently logged for offline analysis.

3.1 Energy and CO_2 Equivalent Emissions Estimation Framework

Since GitHub generate performs computations server-side, direct energy measurement is impractical. Instead, we estimate energy consumption and CO_2 equivalent emissions using a token-based approach, leveraging the number of tokens generated as a proxy for computational effort.

Energy Consumption per Token

Our energy estimation is based on a 2025 Epoch AI report [1], which analyzes ChatGPT-4o, one of three models in GitHub generate. Lacking public data for Claude 3.5 Sonnet and Gemini 2.0 Flash, we focus solely on ChatGPT-4o. The report estimates 0.3 Wh per query for a 500-token response. Converting to joules ($1\,\text{Wh} = 3600\,\text{J}$):

$$\text{Energy per token} = \frac{0.3 \times 3600}{500} \approx 2.16\,\text{J} \tag{1}$$

CO_2 Equivalent Emissions per joule

CO_2 equivalent emissions are derived from Microsoft's 2025 Environmental Sustainability Report [7]. In 2023, Microsoft's data centers consumed 17.8 TWh, with Scope 2 emissions of 1.365 million metric tons of CO_2eq. Converting emissions to grams (1 metric ton $= 10^6$ g) and electricity to kilowatt-hours ($1\,\text{TWh} = 10^9\,\text{kWh}$):

$$\text{Carbon intensity} = \frac{1.365 \times 10^{12}}{17.8 \times 10^9} \approx 76.69\,\frac{\text{gCO}_2\text{eq}}{\text{kWh}} \tag{2}$$

The result from Equation 2, rounded to 77 $\text{gCO}_2\text{eq/kWh}$, we convert to grams per joule ($1\,\text{kWh} = 3.6 \times 10^6$ J):

$$\text{gCO}_2\text{eq/J} = \frac{77}{3.6 \times 10^6} \approx 2.14 \times 10^{-5}\,\frac{\text{gCO}_2\text{eq}}{\text{J}} \tag{3}$$

The result of Equation 3 provides a consistent basis for linking energy to emissions.

3.2 Extension Implementation

We developed a lightweight VS Code extension to apply this framework, integrating energy and CO_2eq estimation into developers' workflows.

Design and Integration

The extension was implemented as a VS Code plugin, using the official extension API to monitor and respond to document-level changes. The core design centers around the detection of AI-generated content, specifically suggestions accepted from generate. This is achieved by maintaining two internal buffers. The first buffer store the document state before the suggestion and another for the state after the suggestion is accepted.

To identify a generate suggestion, the extension listens to the update event in the buffer. Changes that involve multiple lines or span non-trivial character ranges are treated as potential AI insertions. These conditions are evaluated through simple heuristics (change spans multiple lines, or exceeds a minimum token threshold) to reduce noise from typical manual edits. When a potential suggestion is detected, the extension stores the current document as the ”before” buffer and wait for the final form of the change. This before, after buffer mechanism give the extension to isolate the inserted code and differentiate it from user input. The inserted code passed tokenization and energy estimation. The current mechanism is selected to minimize the false positives and ensuring that the user-written edits are not mistakenly included in the analysis ensuring that user-written edits are not mistakenly included in the analysis. Specifically, the extension filters out changes shorter than three tokens and single line edits, which are common when developers type manually. These heuristics are based on structural characteristics of AI suggestions.

Tokenization and Energy Calculation

To compute the energy usage collected from the buffer, the Tiktoken library was used. The Tiktoken library is an OpenAI-released official tokenizer [6]. It was used with the tokenizer model used by ChatGPT-4o to ensure consistency with how AI-generated suggestion calculation. Tokenization is performed on the diffed text. The extension uses encoding to obtain a model-specific encoder, The number of tokens returned is used as the basis for energy estimation. The fixed conversion rate of 2.16 joules per token was taken to estimate the energy used to generate this suggestion. This number is based on research in energy modeling of LLM inference. Furthermore, a CO_2eq emissions estimate is calculated using a constant factor of 0.0000214 grams of CO_2eq per joules. Both estimates are stored and updated across all suggestions accepted during the session.

Real-Time Feedback

The extension provides energy usage feedback directly with the editor through the VS Code status bar. Each time a AI suggestion is accepted, the plugin updates the energy counter shown in the lower right corner of the interface. This status bar displays the total energy consumed during the session. To provide more expanded feedback, a tool tip is also attached to the status bar elements. When hovered over, it displays the amount of energy consumed by the most recent suggestion, as well as the cumulative totals.

Logging for Post-Hoc Analysis

Each accepted generate suggestion is logged to a local file within the user’s workspace. For every recorded event, the log captures the timestamp of the edit, the name of the affected file, the inserted code suggestion, the number of tokens generated, the energy consumed in joules and the cumulative energy. This logging mechanism allows for offline analysis of energy usage and enables aggregation of results across multiple sessions or users.

4 Results

In this section, we present the features of our VS Code extension and provide examples of how it operates in practice. The primary purpose of our extension is to estimate the energy consumption of LLMs within an IDE and display this information to users in real-time. This allows developers to gain insights into the environmental impact of AI-assisted code generation directly within their workflow.

Upon activation within VS Code, the extension initializes the energy consumption counter displayed in the status bar to 0.0 joules (J), as shown in Figure 1. This indicates that no code generation requests monitored by the extension have yet been completed in the current session.

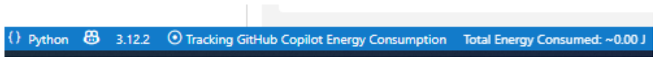
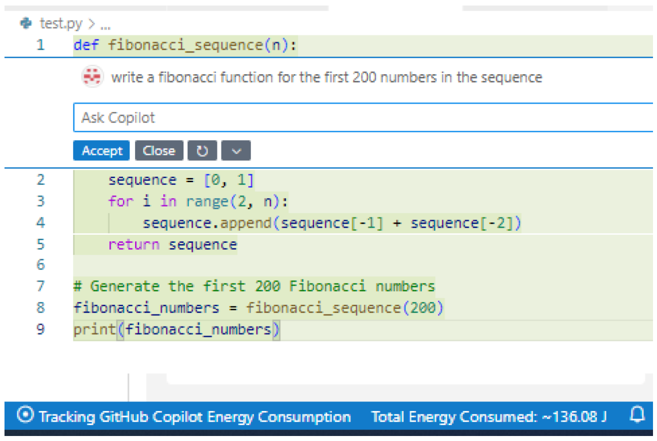

Python 3.12.2 Tracking GitHub Copilot Energy Consumption Total Energy Consumed: ~0.00 J

Figure 1: VS Code status bar showing the initial energy counter reading (0.0 J) upon extension activation.

When a user accepts a code suggestion generated by GitHub Copilot, the extension intercepts the inserted code, calculates the number of tokens (using the methodology described in Section 3), estimates the corresponding energy consumption, and updates the cumulative total displayed in the status bar.

Example 1: Estimating Energy for a Simple Function

To illustrate the core functionality, consider a request for GitHub Copilot to generate a standard Fibonacci sequence function. The code generated for this request is displayed in Figure 2. Based on the token count of this generated code snippet, the extension calculated an estimated energy consumption of 136.08 J. This demonstrates how the extension quantifies the energy impact of a typical code generation task.


test.py > ...

```
def fibonacci_sequence(n):
```

write a fibonacci function for the first 200 numbers in the sequence

```
    sequence = [0, 1]
    for i in range(2, n):
        sequence.append(sequence[-1] + sequence[-2])
    return sequence

# Generate the first 200 Fibonacci numbers
fibonacci_numbers = fibonacci_sequence(200)
print(fibonacci_numbers)
```

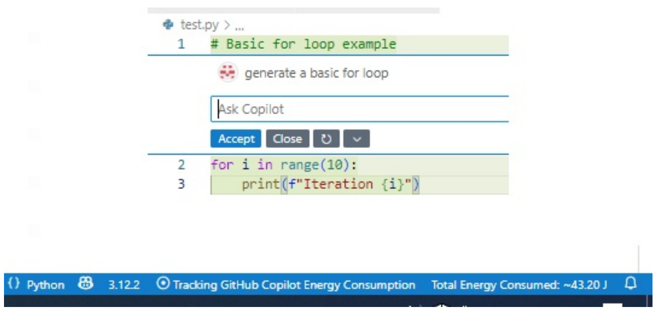
Tracking GitHub Copilot Energy Consumption Total Energy Consumed: ~136.08 J

Figure 2: Generated code for Fibonacci function, with the status bar indicating the corresponding cumulative energy estimate (example shows 136.08 J for this request).

Example 2: Energy Variation with Request Complexity

The estimated energy consumption varies based on the complexity and length (i.e., token count) of the generated code. To demonstrate this, two distinct prompts were issued to GitHub Copilot:

Prompt 1: A Basic For-loop A request for a simple for-loop structure resulted in the code and corresponding energy estimate shown in Figure 3. The relatively short code snippet resulted in a lower estimated energy consumption of 43.20 J.



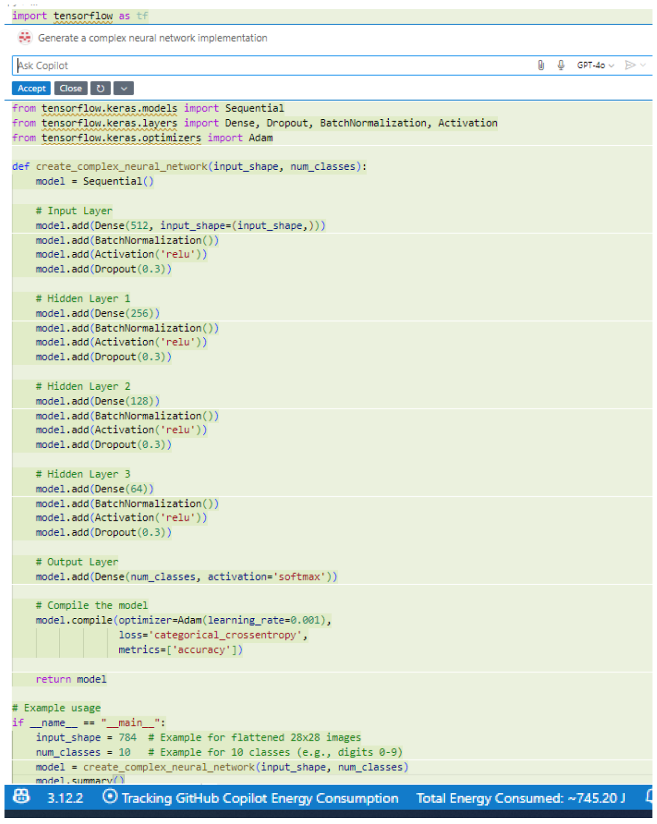
```
test.py > ...
# Basic for loop example
generate a basic for loop
Ask Copilot
Accept  Close
for i in range(10):
    print(f"Iteration {i}")
```

```
Python  3.12.2  Tracking GitHub Copilot Energy Consumption  Total Energy Consumed: ~43.20 J
```

Figure 3: Generated code for a basic for-loop request, with the status bar indicating the corresponding cumulative energy estimate (example shows 43.20 J for this request).

Prompt 2: A Complex Neural Network Implementation A request for a more complex neural network implementation yielded a significantly longer code block, as shown in Figure 4. Correspondingly, the estimated energy consumption for generating this code was substantially higher at 745.20 J, reflecting the greater number of tokens generated.

These examples illustrate the extension's ability to provide real-time feedback that correlates the estimated energy cost with the amount and complexity of code generated by the AI assistant.



```
import tensorflow as tf
Generate a complex neural network implementation
Ask Copilot                                   GPT-4o
Accept  Close
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, Activation
from tensorflow.keras.optimizers import Adam

def create_complex_neural_network(input_shape, num_classes):
    model = Sequential()

    # Input Layer
    model.add(Dense(512, input_shape=(input_shape,)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.3))

    # Hidden Layer 1
    model.add(Dense(256))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.3))

    # Hidden Layer 2
    model.add(Dense(128))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.3))

    # Hidden Layer 3
    model.add(Dense(64))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.3))

    # Output Layer
    model.add(Dense(num_classes, activation='softmax'))

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Example usage
if __name__ == "__main__":
    input_shape = 784  # Example for flattened 28x28 images
    num_classes = 10   # Example for 10 classes (e.g., digits 0-9)
    model = create_complex_neural_network(input_shape, num_classes)
    model.summary()
```

```
3.12.2  Tracking GitHub Copilot Energy Consumption  Total Energy Consumed: ~745.20 J
```

Figure 4: Generated code for a neural network implementation request, with the status bar indicating the higher cumulative energy estimate (example shows 745.20 J for this request).

5 Discussion

This study successfully demonstrated the feasibility of developing an IDE extension to provide real-time estimations of energy consumption and CO2 equivalent emissions associated with AI-driven code generation tools, specifically focusing on GitHub Copilot within the Visual Studio Code environment. By leveraging publicly available data on LLM energy use per token and data center carbon intensity, the extension translates the abstract computational cost of AI suggestions into tangible metrics displayed directly within the developer's workflow. The results confirm the initial premise: even seemingly minor AI assistance requests incur a measurable energy footprint. Furthermore, the examples presented in the Results section (Section 4) illustrate a clear correlation, showing that longer generated code snippets, corresponding to higher token counts, result in higher estimated energy consumption. This contributes incrementally but cumulatively to the overall environmental impact of software development.

Acknowledging the broader context requires discussing potential trade-offs related to overall energy usage and code quality. Our focus is specifically on the energy consumed by the LLM during the code generation process. A second, distinct aspect to consider for overall software sustainability is the energy efficiency of the generated code itself during execution. Complex trade-offs exist here: code requiring more tokens (and thus more generation energy) might execute significantly faster, consuming less energy over its lifetime compared to shorter but less optimized alternatives. Conversely, while smaller, potentially less energy-intensive LLMs might generate code faster, the quality and correctness of that code could be lower, potentially requiring multiple re-generations or extensive manual debugging, which could negate initial energy savings or even increase overall effort and consumption. Evaluating the runtime efficiency of the output code is a separate challenge and outside the direct scope of this extension's measurement capabilities. Notably, few studies directly compare the energy cost of LLM code generation versus the execution energy cost of the resulting code; research on this specific trade-off is still emerging.

5.1 Limitations

Dependency on External and Aggregated Data

The core limitation based on the reliance on estimations derived from publicly available, aggregated data rather than direct, real-time measurements from the service infrastructure. Accuracy is fundamentally constrained by the general lack of transparency from LLM providers regarding their operational energy footprints. Our calculations depend on third-party reports for average energy-per-token (derived from query-level data in [1]) and overall data center carbon intensity [7]. Also, this data doesn't account for the specific hardware (e.g., GPU type, efficiency) utilized for a given inference, network latency impacts, cooling system efficiency (PUE), or other dynamic factors influencing actual energy draw per request. Consequently, the presented values should be interpreted as approximations rather than exact measurements.

Model Specificity and Tokenizer Uncertainty
The energy estimation currently uses parameters based solely on data available for the ChatGPT-4o model. GitHub Copilot integrates various LLMs, and energy consumption profiles for other models (e.g., Claude, Gemini variants) were not publicly available. This restricts the accuracy when users select models other than the one characterized. Furthermore, the precise tokenization algorithms used by different LLM providers can vary. While we assume the use of tiktoken [6] for OpenAI models, a different tokenizer employed by another provider would yield a different token count for the same text, directly impacting the energy calculation's accuracy for suggestions generated by non-OpenAI models. Our methodology cannot currently adapt to these undisclosed, model-specific tokenizers.

Lack of Official API and Logging Reliability
The lack of an official Copilot API from Microsoft presented development challenges, so the Copilot suggestions have to be pulled directly through VS Code's API calls making the creation of an extension necessary. This indirect method proved workable but can be less reliable than direct API integration, occasionally failing to capture the complete text of very large or rapidly streamed suggestions. This may lead to underestimations of token counts and, therefore, energy consumption, particularly for complex, multi-line code generations. Integration with an official API, if made available, would significantly improve data capture reliability.

5.2 Future Improvements

Incorporating Input Processing Energy Costs
A primary limitation of the current estimation is its focus solely on output tokens generated by the LLM. However, processing the input prompt and context provided by the user often constitutes a significant portion of the computational load, especially with large context windows common in modern development workflows. A key future direction is to extend the energy estimation model to include the energy cost associated with processing these input tokens. This requires access to currently unavailable server-side data for specific LLM inference processes. If such data emerges, we could adapt the extension to multiply input token counts—obtained via tiktoken - by a corresponding energy-per-token value, providing a more comprehensive energy profile for each suggestion.

Expanding Model Support and Specificity
The current implementation relies on energy data specific to ChatGPT-4o. To improve accuracy and relevance across the diverse LLMs offered by GitHub Copilot (and potentially other AI coding assistants), future versions should incorporate model-specific energy profiles and tokenizers as data becomes available. This would allow the extension to dynamically adjust its calculations based on the LLM selected by the user or detected automatically.

Enhanced Visualization and Reporting
Beyond the current status bar display, more sophisticated visualization and reporting features could greatly enhance user understanding and engagement. Potential improvements may include:

- *Granular Aggregation:* Displaying energy consumption aggregated per-file, per-directory, or per-project, helping developers identify energy "hotspots" within their codebase interactions.
- *Historical Tracking:* Implementing logging and visualization of energy usage trends over time (e.g., daily or weekly summaries), allowing developers to monitor their cumulative impact.
- *Dedicated UI Panel:* Creating a dedicated view or panel within VS Code to present these richer visualizations and reports, rather than relying solely on the status bar and tooltips.

Official API Integration
Should an official API for GitHub Copilot become available, refactoring the extension to leverage this API would be a priority. This would likely resolve current inaccuracies in capturing generated code snippets and provide a more robust and reliable foundation for measurement.

User Studies and Behavioral Analysis
To understand the real-world impact of this tool, conducting user studies would be invaluable. Such studies could assess how developers perceive and interact with the real-time energy feedback, whether it influences their usage patterns of AI assistants (e.g., prompt strategies, acceptance frequency), and gather qualitative feedback for further refinement of the tool's design and features.

6 Conclusion

This study developed a VS Code extension to estimate and visualize the energy consumption and CO_2eq emissions of AI-driven code generation, focusing on GitHub Copilot. Key takeaways include the feasibility of using a token-based approach to approximate server-side energy use, the successful integration of real-time energy metrics into a widely used IDE, and the demonstration that even simple AI queries contribute meaningfully to energy consumption. The extension empowers developers with visibility into the environmental impact of their tools, revealing, for instance, that complex requests like neural network implementations consume significantly more energy than basic loops. However, the research also underscores the challenges of precise estimation due to limited transparency from LLM providers and the absence of an official Copilot API, which restricts accuracy to rough estimates.

These findings highlight the urgent need for greater industry openness regarding AI energy profiles to enable sustainable software development practices. Future efforts should prioritize securing detailed energy data, incorporating input token processing, and refining integration through direct API access if available. By addressing these gaps, this tool can evolve into a more robust instrument for promoting energy-conscious coding. Ultimately, this work serves as a stepping stone toward balancing the productivity benefits of LLMs with their environmental costs, urging the computing community to prioritize sustainability alongside innovation.

References

[1] Josh You, "How much energy does ChatGPT use?," Epoch AI, 2025, https://epoch.ai/gradient-updates/how-much-energy-does-chatgpt-use, Accessed: March 26, 2025.

[2] Alex de Vries, "The growing energy footprint of artificial intelligence," *Joule*, vol. 7, no. 10, pp. 2191–2194, October 2023, https://www.sciencedirect.com/science/article/pii/S2542435123003653.

[3] P. Kumar, "Energy consumption of information retrieval: A comparative study of traditional and AI-driven search engines," *Journal of Sustainable Computing*, vol. 12, no. 3, pp. 45–52, 2021.

[4] GitHub, "2023 Octoverse Report: The State of Open Source and AI," GitHub, 2023, https://octoverse.github.com.

[5] Stack Overflow, "2023 Developer Survey Results," Stack Overflow, 2023, https://survey.stackoverflow.co/2023.

[6] OpenAI, "Tiktoken," GitHub repository, 2025, https://github.com/openai/tiktoken.

[7] Microsoft, "Sustainability Report," Microsoft, 2025, https://www.microsoft.com/en-us/corporate-responsibility/sustainability/report.

[8] Microsoft, "Visual Studio Code - Code Editing. Redefined," Microsoft, Nov. 2021. Available: https://code.visualstudio.com/.

[9] Microsoft, "Visual Studio Code API," Microsoft. Available: https://code.visualstudio.com/api.

[10] Microsoft, ".NET Development with Visual Studio," Microsoft. Available: https://visualstudio.microsoft.com/vs/features/net-development/.

[11] JetBrains, "IntelliJ IDEA - The Leading Java and Kotlin IDE," JetBrains. Available: https://www.jetbrains.com/idea/

[12] Sublime HQ, "Sublime Text," Sublime HQ. Available: https://www.sublimetext.com/.

[13] Apple, "Xcode," Apple Developer. Available: https://developer.apple.com/xcode/.

[14] Google, "Android Studio," Google Developers. Available: https://developer.android.com/studio.

[15] GitHub, "GitHub Copilot," GitHub. Available: https://github.com/features/copilot.

[16] CodeGPT, "CodeGPT - AI-Powered Coding Assistant," CodeGPT. Available: https://codegpt.co/.