Jamanota
Energy Tracking for Agentic AI

Samuel van den Houten
Computer Science
Delft University of Technology
svandenhouten@tudelft.nl

Jan Kuhta
Computer Science
Delft University of Technology
jkuhta@tudelft.nl

Andrea Vezzuto
Computer Science
Delft University of Technology
avezzuto@tudelft.nl

Aadesh Ramai
Data Science and AI Technology
Delft University of Technology
aramai@tudelft.nl

Rodrigo Montero González
Data Science and AI Technology
Delft University of Technology
rmonterogonzal@tudelft.nl

***Abstract*—The rapid growth of AI agent systems in recent years has introduced significant environmental concerns. However, developers currently lack the tools to assess the energy and carbon footprint of their applications. Existing observability platforms expose token usage as a proxy for cost, but do not translate this into meaningful environmental metrics. Furthermore, no existing solution addresses environmental resource tracking within multi-agent LangChain workflows.**

In this paper, we present Jamanota, a middleware tool for the LangChain framework that enables developers to gain insights into the resource consumption of their agentic pipelines. By improving the transparency of otherwise hidden environmental costs in AI systems, Jamanota helps bridge a gap in sustainable AI development. The middleware intercepts model calls, extracts associated execution metadata, and uses them to derive and store estimated environmental metrics such as energy usage and carbon emissions. These metrics are exposed through a Python API, providing functions that return total or average consumption, as well as aggregations by model or agent, with optional filtering by the last N prompts or time intervals.

We evaluate the tool through a scenario-based analysis across three different developer personas, assessing several key aspects such as its transparency, interpretability, and actionability. The results show that the middleware increases visibility into environmental resource consumption of model calls with meaningful insights. While the solution is a step closer to more sustainable and environmentally interpretable agentic AI systems, our work highlights limitations in evaluation and estimation precision, and outlines future directions including energy-aware optimisation and more fine-grained energy modelling.

I. Introduction

Large language models (LLMs) have rapidly become a core component of modern artificial intelligence systems. In recent years, models such as GPT, Claude, Gemini, and open-weight alternatives like LLaMA, Mistral, and Qwen have enabled a wide range of applications, from conversational assistants to code generation and decision-support systems. More recently, these capabilities have been extended through *AI agent* systems, in which models interact with tools and other agents to perform complex tasks autonomously. Frameworks such as LangChain have accelerated this shift by simplifying the development of multi-agent workflows and enabling the integration of language models with external tools.

Despite these advances, the rapid adoption of LLM-based systems has raised concerns regarding their environmental impact. Running large models requires substantial computational resources, which leads to energy consumption and associated carbon emissions. While much attention has been given to the environmental cost of training large models, the growing use of LLMs in production systems means inference and repeated model calls contribute significantly to overall energy usage.

However, efforts to mitigate this impact are hindered by a lack of visibility into detailed, fine-grained consumption metrics of these complex systems. When using LLM frameworks, developers are mainly exposed to token usage, which represents the number of tokens processed by the model. While tokens are useful for estimating API costs, they provide little insight into the underlying computational or environmental impact of model usage. As a result, energy consumption remains largely hidden, and developers may be unaware of the sustainability trade-offs associated with model size, prompt length, or multi-agent workflows. This creates a gap in transparency for developers who wish to understand and manage the sustainability impact of their applications.

In this work, we aim to address this gap by introducing a middleware tool that estimates the energy consumption and associated carbon emissions of LLM-based agent systems. Developed for the LangChain framework, the tool automatically intercepts model calls within agent workflows. It estimates the computational cost of interactions based on factors such as token usage, model parameters, and assumptions about hardware efficiency. These estimates are translated into energy consumption and associated emissions, which are exposed to the developer. As such, the goal of this work is to help developers better understand the environmental implications of their design decisions and to encourage more

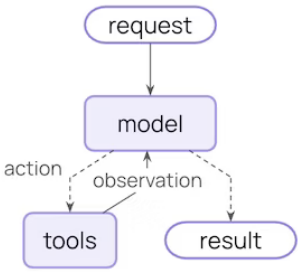

Fig. 1. The LangChain agent loop [1]

informed trade-offs when building LLM-powered systems.

The main contributions of this work can be summarised as follows:

- We propose a middleware architecture that intercepts LLM agent interactions and captures execution metadata for environmental analysis.
- We introduce an estimation model that translates token usage into energy consumption and carbon emissions, based on hardware and carbon-intensity assumptions.
- We design and publish a Python package[1], along with in-depth documentation and tutorials[2], that integrate seamlessly with developer workflows and observability systems.
- We evaluate the middleware through scenario-based analysis across different developer personas to assess its effectiveness in improving transparency.

II. Background and Related Work

LangChain is a popular open-source framework for building LLM-powered applications. It provides a standard API for interacting with models from many providers, both commercial and self-hosted. It also provides *tools*, which models can use to interact with the world. AI agents, therefore, combine these functionalities to perform actions in a loop until they have completed a given task. LangChain implements agents using the agent graph, shown in Figure 1. This functions like a finite state machine. The flow starts with a request to the model, which can call tools through actions and get back an observation. Additionally, tools can themselves invoke other agents, creating nested agent loops. After the model has called all the chains of tools it needs, it outputs the result. LangChain *middleware* can hook onto these execution states in the graph to both modify and log their execution.

There exist multiple web-based observability solutions for LangChain, most notably *LangSmith* [2] and *Langfuse* [3]. The former is a proprietary, subscription-based service developed by LangChain, while the latter is an open-source alternative. These platforms offer similar features, including in-depth traces of the LangChain state machine and dashboards showing aggregate data. However, neither of these solutions report data on energy usage or carbon emissions. While they do provide metrics that serve as proxies for energy usage (token consumption, model usage), actually calculating these estimates requires exporting the data and analysing it externally. By default, these platforms also aggregate data only over time. While useful for monitoring system activity, this form of aggregation is less suitable for evaluating the efficiency of individual tasks or interactions. In LLM-based systems, a single user request may involve multiple model calls and tool invocations, meaning that time-based aggregation can obscure the cost of a specific task. When evaluating task-level efficiency, aggregating the last N prompts would be more valuable as it allows developers to compare the environmental impact of different queries, workflows, or agent configurations more directly.

Additionally, various projects directly address LLM energy consumption. For example, *LLM Energy Consumption Monitoring* [4] is a Python package that tracks energy usage by directly measuring the electricity used by the hardware running local LLMs. While this is the most accurate way to track energy usage, it is not suitable for most developers, who use proprietary, externally hosted models. Another project of interest is Greenchat [5], which estimates CO2 emissions produced by using LLM-based apps like *ChatGPT*. However, it is intended for end users and only works when an integration has been made for the specific app. It is therefore unsuitable for developers who want to estimate the energy consumption of their own prototypes. Finally, all solutions addressing LLM energy usage focus solely on measuring a single LLM call at a time, making them unable to trace multi-agent systems.

III. Methodology

A. System Overview

Jamanota uses the LangChain framework to provide transparent environmental impact metrics for LLM-based applications. The main flow of the system starts with an agent call intercepted by our middleware, which uses execution metadata to estimate the agent's environmental impact. This data is then stored and can be fetched by developers to analyse and support decision-making processes. This flow is depicted in Figure 2, where the three main parts of our system are clearly distinguished: the *middleware* design, the *energy estimation model*, and the *prototype* design.

- **Middleware Design:** We develop a middleware that intercepts all model interactions. It captures relevant metadata, such as token usage and timestamps, to estimate and aggregate energy usage data. Furthermore, identifiers are propagated across agent calls to group data from sub-prompts belonging to the same initiating prompt. Finally, our middleware exposes various API functions for fetching estimated environmental data and precomputed aggregates.
- **Energy Estimation Model:** Using the data captured by the middleware, we estimate the energy consumption and

[1]PyPI package available at: https://pypi.org/project/jamanota/.
[2]Documentation available at: https://jamanota.readthedocs.io.

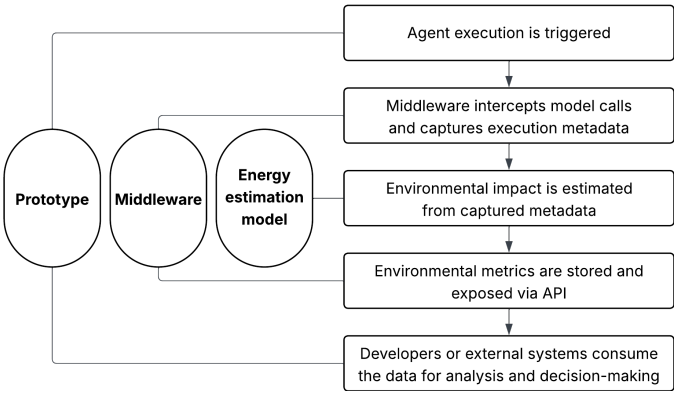

Fig. 2. Middleware pipeline for capturing, estimating, and exposing environmental metrics

associated carbon emissions of the interactions with the agent(s).

- **Prototype**: To test and demonstrate how our solution might be used in a real-world setting, we design a multi-agent system with a main agent and two specialised agents. Additionally, we create a simple `streamlit` web application to interact with these, supplemented with visualisations of the energy usage data with aggregation options. This prototype functions primarily as a tutorial through which the developer can be guided when going through our documentation[3]. The prototype design is further detailed in section IV.

B. Middleware Design

The custom middleware extends LangChain's base middleware class, allowing us to hook into the lifecycle of an agent call. Specifically, Jamanota is designed primarily to support the following three features.

- **Calculate and store environmental data:** To actually get hold of environmental data, an `after_model` hook allows us to obtain the input tokens, the output tokens, the model name, and the current timestamp for each model call. The input and output tokens are used to estimate energy usage and carbon emissions. The model name, on the other hand, is used as a multiplier for our estimates, since different models have different numbers of parameters, which directly affect their energy usage and carbon emissions. Lastly, the timestamp enables time-based aggregation of the collected information. Together, this data is structured into an `EnergyDataPoint` class. The object is then appended to a list, which is stored as an attribute of the middleware.
- **Grouping agent calls:** To report the data in a meaningful way, we must keep track of which sub-prompts belong to which main-prompt. This is implemented using `before_agent` and `after_agent` hooks, which propagate a prompt identifier from the main prompt to all its sub-prompts via a stack-based approach. For each prompt, this identifier is also stored inside its corresponding data point.
- **Aggregate and return data:** To access the data stored by the middleware, several utility methods within the tool are implemented. These allow users to retrieve all raw data points, fetch filtered points for those executed in the last N hours, or get the last N data points. Finally, data may be grouped by model or agent name, enabling developers to conduct a more fine-grained environmental analysis of their multi-agent system.

[3]Documentation available at: https://jamanota.readthedocs.io.

C. Energy Estimation Model

To provide meaningful information regarding the environmental impact of LLM-based agent systems, we estimate the energy consumption and associated carbon emissions of model interactions. The goal of this estimation is not to provide exact measurements, but to give developers a reasonable approximation of the environmental cost of their systems, enabling them to compare different models and/or architectures.

Overall, the estimation process is based on three main components: the computational cost of model inference, the hardware efficiency of the device executing the model, and the carbon intensity of electricity production.

a) Computational Cost of Inference: The first step of the estimation process is to calculate the approximate amount of computation required to process a request. Following common methods used in transformer literature, the number of floating-point operations (FLOPs) required to generate a token during inference can be approximated as:

$$FLOPs_{token} \approx 2P \tag{1}$$

where P represents the number of parameters in the model [6]. This approximation reflects the forward pass of a dense transformer model during inference.

For a request containing both input and output tokens, the total computational cost is therefore estimated as:

$$FLOPs_{total} = 2P \times T \tag{2}$$

where T represents the total number of processed tokens.

b) Hardware Efficiency: Having obtained the computational cost of an interaction with a model, we now convert it into an estimate of energy consumption. This calculation requires an assumption about the hardware executing the model. In this work, we assume a datacenter-grade GPU as the baseline execution environment, as most large-scale LLM inference is performed on specialised hardware in cloud or production settings. Specifically, we use the specifications of an NVIDIA H100 GPU [7].

The H100 provides approximately 1,979 TFLOPs of half-precision (FP16) tensor core performance, with a thermal

design power (TDP) of up to 700 W. From these specifications, we derive the approximate computational efficiency:

$$FLOPs_{perJoule} = \frac{1.979 \times 10^{15}}{700} \tag{3}$$

which results in an estimated efficiency of approximately 2.83×10^{12} FLOPs per Joule.

Using this value, the energy consumption for a model interaction can be estimated as:

$$Energy = \frac{FLOPs_{total}}{FLOPs_{perJoule}} \tag{4}$$

Overall, the assumptions made for hardware efficiency reflect a production-oriented deployment scenario in which inference is performed on highly optimised data centre GPUs with theoretical peak efficiency. Therefore, the energy calculations represent an approximation of real-world performance.

c) Carbon Emissions Estimation: Finally, we convert the computed energy consumption into carbon emissions to estimate the environmental impact of the agent system. Following global energy statistics [8], we assume an average carbon intensity of:

$$0.45 \text{ kg } CO_2/\text{kWh} \tag{5}$$

Since one kilowatt-hour corresponds to 3.6×10^6 Joules, this value can be converted to emissions per Joule:

$$CO_2/Joule = \frac{0.45}{3.6 \times 10^6} \tag{6}$$

which results in approximately 1.25×10^{-7} kg of CO_2 per Joule.

Finally, the estimated emissions are computed as:

$$CO_2 = Energy \times CarbonIntensity \tag{7}$$

IV. Evaluation

Due to the limited time frame of this project and the complexity of conducting a full user study, we adopt a scenario-based evaluation methodology. This approach is commonly used in Human-Computer Interaction (HCI) to assess how different types of users may interact with a system in realistic contexts, without requiring large-scale empirical testing [9].

As stated in the problem definition in Section I, the main objective of this work is to increase transparency regarding the environmental impact of LLM-based agent systems. However, evaluating transparency in practice requires more than checking whether metrics are exposed. A developer-facing middleware must also fit into existing workflows, provide data in a consumable form, and support meaningful reasoning about the reported information. For this reason, our evaluation considers not only whether the middleware reveals previously hidden information, but also whether that middleware can be integrated into development practice and used productively.

To this end, we define three personas representing different kinds of developers, and construct one scenario for each. Together, the personas and scenarios allow us to study both *who* is using the middleware and *how* it is used in practice. Each scenario is analysed with respect to several key aspects of developer interaction, including integration into existing workflows, how data is consumed, how the resulting metrics are interpreted, whether they support actionable decisions, and potential risks arising from their use.

In this evaluation, the middleware is treated as a transparency tool that provides approximate and comparative insights into the environmental cost of model interactions. The reported values should therefore be interpreted as indicative rather than exact, and as reflecting model-level behaviour rather than full system energy consumption.

A. Personas

a) Persona A: Prototyping developer with beginner to intermediate expertise. This developer is building a prototype of a multi-agent system using frameworks such as LangChain. Their primary focus is on functionality and correctness rather than optimisation. They have limited experience with performance profiling and are not familiar with estimating the computational or environmental cost of LLM usage.

b) Persona B: Backend developer with intermediate to advanced expertise. This developer works on production systems that rely on LLM-based pipelines. They are concerned with efficiency, scalability, and cost. While they may already track token usage or latency, they lack tools that translate this information into meaningful estimates of energy consumption or environmental impact.

c) Persona C: Platform engineer with advanced expertise. This developer builds infrastructure or tools for other developers, such as platforms that support LLM-based applications. They are interested in observability and monitoring, and aim to expose metrics to downstream users. They have strong technical expertise and are capable of integrating middleware into larger systems

B. Scenarios

a) Scenario A: The prototyping developer is building a multi-agent system using LangChain. Since their primary goal is to have a functional prototype, they follow the tutorials published in the documentation of the middleware, demonstrating how they can integrate Jamanota into a multi-agent pipeline and how to visualise the collected data in a simple frontend application. Using these guides, they attach the middleware to a lightweight main agent and two specialised subagents, and create a `streamlit` interface with a sidebar for energy monitoring. As such, the developer can observe how many tokens each agent processes, the estimated energy consumption, and the contributions of different components to the overall cost.

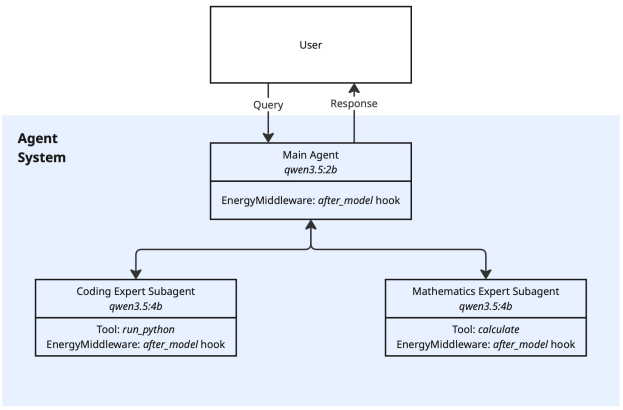

Fig. 3. Experimental Agent Architecture

This provides visibility into the system's internal behaviour, which was previously opaque.

The components described in the tutorials and implemented by the developer are detailed in the following sections.

Agent Architecture - As shown in Figure 3, the sample multi-agent setup is composed of a lightweight main agent and two specialised subagents. The former receives the user's queries and decides whether they can be answered directly or should be delegated to a specialised agent. As such, the main agent uses a smaller language model (`qwen3.5` with 2 billion parameters) to minimise computational cost when simple reasoning suffices.

Should more complex reasoning be necessary for mathematics or coding-related questions, the main agent can delegate the question to two larger subagents (`qwen3.5` with 4 billion parameters):

- **Coding Expert Subagent:** a simulated specialised agent for coding that assists users with programming tasks such as writing code, debugging, and explaining program behaviour. This agent has access to a tool that can execute Python code and return the output or error message, allowing it to analyse the runtime behaviour of programs.
- **Mathematics Expert Subagent:** a simulated specialised agent for numerical reasoning and mathematical problems. This agent has access to a calculator tool that evaluates mathematical expressions, enabling it to perform exact computations when necessary.

To interact with the agents, the tutorial uses `streamlit`, an open-source Python framework for data scientists and AI/ML engineers to build interactive data apps. Using minimal code and a single command, we can obtain a simple frontend that allows us to query the agents.

During execution, the custom middleware intercepts every model invocation (after it has returned the query result), thereby recording token usage to estimate the energy consumption of a particular interaction.

Data Reporting - the `streamlit` application can be extended to include an expandable sidebar that displays useful information about the agent's environmental impact, sourced from the middleware. This information consists of the following.

- **Total energy:** The total energy, CO2e, number of input tokens and the number of output tokens are shown.
- **Filters:** A slider that filters the data by either the last N prompts or prompts executed in the last N hours.
- **Aggregates:** Data can be aggregated per agent or per model, with the options to do so by summing or taking the average over the data points.

b) Scenario B: The backend developer maintains a production pipeline containing multiple LLM calls and agent interactions. Although the system is functional, it is costly to run and difficult to optimise because the developer has limited visibility into where computational resources are being consumed. The developer integrates the middleware into the pipeline and accesses the collected information programmatically through API functions such as `breakdown_by_model`. The returned data is then used in scripts or internal analysis workflows to compare models, inspect costly agent interactions, and identify pipeline components that contribute disproportionately to the estimated environmental cost.

This scenario represents a workflow in which the middleware is not primarily consumed through a dedicated user interface, but as structured data that can support diagnosis and optimisation. The backend developer is not only interested in seeing the reported metrics, but also in using them to prioritise engineering effort and compare alternative designs.

c) Scenario C: The platform engineer is building a system that allows other developers to create, run, and monitor LLM-based agents. As part of this platform, they aim to provide observability features that include sustainability-related metrics. The engineer integrates the middleware into the backend and uses its API to collect structured data about energy consumption and emissions for each request. This data is then surfaced to downstream developers through platform-level interfaces such as dashboards, logs, or other reporting layers.

Unlike the previous scenarios, the engineer is not the final consumer of the information. Instead, they use the middleware as a building block inside a larger observability pipeline. This scenario, therefore, focuses on how well the middleware supports platform integration and whether its outputs can be responsibly communicated to other developers at scale.

C. Analysis

Following the definition of the personas and scenarios, we now analyse how each type of developer interacts with the middleware across the full lifecycle of use, from integration to interpretation, and whether it supports the intended goal of

increasing transparency in LLM-based agent systems.

a) Scenario A – Prototyping Developer: For the prototyping developer, the middleware fits naturally into the development workflow. Using the tutorials, they are able to quickly develop a system that is relatively small, controlled, and already oriented toward experimentation. Attaching the middleware to the agent requires little effort and does not substantially disrupt the implementation of the prototype. In this sense, the middleware is well-suited to early-stage development, where developers often want to instrument their systems quickly and observe internal behaviour with minimal engineering overhead.

The way in which the data is consumed also aligns well with this scenario. Because the developer implements both the agent system and the interface, the middleware API can be used immediately with a lightweight frontend such as the `streamlit` prototype shown in the tutorials. Features such as prompt-level grouping, filtering, and aggregation support exploratory analysis by allowing the developer to inspect individual runs and compare how different queries are routed across the agent architecture. This is particularly useful in a multi-agent setting, where the internal distribution of work is otherwise difficult to observe.

However, while the middleware improves visibility, the interpretation of the reported metrics is more difficult. Raw values such as Joules and CO2e could be unfamiliar to some low-level developers, and the current system does not provide contextual benchmarks that would help them determine whether a given value is high or low. As a result, the tool increases awareness, but may not immediately lead to well-informed decisions. There is also a risk that the developer interprets the estimates as exact measurements.

This limitation affects actionability. The middleware helps the prototyping developer understand how the system behaves and where agent interactions are concentrated, but it offers limited support for deciding what to optimise or whether a given design is environmentally reasonable. In this scenario, the main benefit is therefore the ability for exploring transparency rather than being able to confidently make decisions.

Key takeaways:

- Using the documentation, a prototype system with the middleware can be implemented easily and supports quick experimentation.
- Transparency is improved for interactions that would otherwise remain opaque.
- Raw environmental metrics are difficult to contextualise without benchmarks or interpretive support.
- The tool primarily supports awareness and exploration, rather than strong optimisation decisions.
- There is a risk of overconfidence if estimated values are interpreted as exact measurements.

b) Scenario B – Backend Developer: For the backend developer, the middleware remains easy to integrate, but its value lies less in visual exploration and more in structured analysis. Because the middleware exposes its results through a Python API, the collected data can be consumed programmatically and incorporated into scripts, logs, or internal monitoring workflows. This makes the tool compatible with the kinds of engineering practices common in backend development, where developers often reason through aggregated metrics rather than through a dedicated interface.

By grouping results by model, agent, or recent prompts, the developer can compare components of the pipeline and identify which parts of the system are likely to be the main contributors to estimated energy usage. This makes the middleware useful as a comparative diagnostic tool. It helps reveal, for example, whether a larger model is consistently responsible for most of the cost, or whether repeated sub-agent calls create unnecessary overhead.

Compared with the prototyping developer, this persona is better positioned to interpret the estimates appropriately. A backend developer is more likely to understand that the numbers should be treated as approximate indicators rather than as direct physical measurements. Even so, the estimates remain partial. They capture the model-inference component of system behaviour, but do not account for other contributors to real-world energy use, such as networking, memory usage, orchestration overhead, or cooling. This limits the credibility of the values as absolute measures of environmental impact. Furthermore, the middleware still does not provide benchmarks or reference values. As a result, the developer may be able to compare alternatives, but still lack a clear basis for judging the practical significance of a given result.

Despite this limitation, the middleware still supports meaningful actionability in practice. It allows the developer to compare alternatives, prioritise optimisation work, and identify costly parts of the pipeline. What it does not provide is proof that a redesign will lead to a specific real-world energy reduction. In other words, the tool can be strong for prioritising investigation and comparative optimisation, but not for validating total system savings with precision.

Key takeaways:

- The middleware integrates well into backend workflows and can be consumed programmatically.
- Aggregation functions align with common optimisation and diagnosis tasks.
- The reported values remain partial estimates and do not reflect total system-level energy use.

- The tool provides comparative values but still lacks benchmarks for judging whether reported values are environmentally acceptable or significant.
- The middleware supports prioritisation of optimisation effort, but not precise validation of real-world savings.
- There is a risk of over-optimising based on incomplete metrics if the scope of the estimates is not understood clearly.

c) Scenario C – Platform Engineer: For the platform engineer, the middleware is valuable because it is exposed as an API-oriented backend. This makes it suitable as a building block within larger observability or platform infrastructures. The engineer can integrate it into the backend and use it to collect structured data that is then passed onward to logs, dashboards, or other interfaces for downstream users. In this sense, the design of the middleware supports composability and makes it possible to extend sustainability-related observability beyond a single local development setting.

The main strength of this scenario is the scalability of impact. Instead of informing only the developer who integrates the tool, the middleware can be used to expose sustainability information to many developers working through the same platform. This gives the tool a broader reach and makes its transparency benefits potentially more significant. At the same time, this wider reach also creates greater responsibility. Once the data is shown to downstream users, the platform engineer must ensure that the estimates are communicated with sufficient clarity, including their approximate nature and their limited scope.

The aforementioned strength can also become the main limitation. In this setting, incomplete or misleading interpretations do not affect only one developer, but may propagate across many users. The experienced engineer may understand the approximate nature of the reported values, but downstream developers using the platform may not. Moreover, there are additional requirements that are outside the current implementation, such as isolation of data across different users or applications. The middleware therefore can work as a core component, but not yet as a complete production-ready observability solution on its own.

In terms of actionability, the platform engineer can enable sustainability awareness and comparison at scale, potentially influencing the behaviour of many downstream developers. However, because the tool exposes approximate rather than complete measurements, this actionability must be accompanied by careful communication.

Key takeaways:

- The middleware's API-based design makes it suitable as a backend building block inside larger platforms.
- It enables sustainability transparency to be surfaced to multiple downstream users.
- Platform-level usage increases the importance of communicating uncertainty and scope clearly. Downstream users may still lack the context needed to interpret raw environmental values.
- The consequences of approximation and misinterpretation are amplified when metrics are exposed at scale.
- Additional infrastructure, such as isolation, is needed for robust production deployment.

d) Cross-Scenario Analysis: Across all scenarios, the middleware improves transparency by exposing aspects of model usage that would otherwise remain hidden. This supports the study's claim that developers lack visibility into the environmental impact of their systems, and even approximate metrics can already make important parts of the agent workflow more interpretable.

A second recurring finding is that the middleware is most reliable as a comparative tool rather than as a precise measurement system. Across all scenarios, the estimates are useful for identifying relative differences between prompts, models, and agent structures, but not for claiming full system-level environmental impact. This makes the tool effective for diagnosis and prioritisation, while also defining a clear boundary on what conclusions should and should not be drawn from the results.

Finally, the evaluation highlights that the impact of the middleware increases with scale, but so do the associated risks. In local prototype use, misinterpretation affects a single developer. In platform-level deployment, the same misunderstanding may reach many downstream users. As a result, the strengths of the middleware, such as ease of integration, must be considered together with the need for interpretive support, careful communication, and clear acknowledgement of the limits of estimation.

V. Discussion

The middleware can successfully increase transparency by exposing metrics such as token usage and estimated energy consumption. Nonetheless, this transparency is not equally useful for all developers. Beginner developers gain awareness but may struggle to interpret or act on the data. Intermediate and more experienced users can directly use the data for optimisation. This shows that transparency is a positive step towards increasing sustainability awareness among developers, but it alone is insufficient to achieve a complete impact.

While the middleware does not optimise systems directly, it enables developers to reason about trade-offs and support informed decision-making. Through a simple package installation and a few additional lines of code, developers can have more information when building their applications. However, this requires developers to be able to act on the information received. In practice, this is challenging due to the

lack of standardised benchmarks for sustainability in LLM-based systems. Although various sustainability frameworks and certification schemes exist (such as the Green Software Foundation and the GHG Protocol), they often operate at different levels (e.g., infrastructure, organisations, or full lifecycle analysis) and define efficiency or environmental impact in different ways. As a result, the same energy or emission estimate may be interpreted differently depending on the framework used. This lack of alignment limits developers' ability to determine what constitutes "good" or "acceptable" performance, reducing the practical actionability of the provided metrics.

To make such tools more broadly effective, additional support mechanisms are needed. For instance, additional tooling could provide reference benchmarks (e.g., what constitutes high or low energy usage) or higher-level summaries. At a broader level, the community could contribute by standardising sustainability metrics and integrating them into commonly used observability tools. Such improvements would lower the barrier for less experienced developers.

While the proposed middleware provides greater transparency into the energy consumption of LLM-based systems, there are several limitations in system design and practical use that are worth discussing.

First, the middleware provides estimates at the level of individual model calls, but does not capture the full system-level energy consumption. In particular, aspects such as memory usage and infrastructure overhead are not considered. As a result, the reported values should be interpreted as partial indicators rather than complete measurements of system impact.

Second, the middleware exposes low-level metrics without providing guidance on how to interpret them. There are no reference points to help developers determine whether a given value is high or low, or what actions to take. This may limit the tool's practical usefulness, particularly for less experienced developers.

Third, the current implementation focuses on transparency rather than actionability. While the middleware enables developers to observe and analyse system behaviour, it does not provide automated suggestions. As a result, any improvements to the system depend entirely on the developer's ability to interpret and act on the information provided.

Finally, the integration is currently demonstrated within a specific framework (LangChain) and a limited set of models. Although the middleware is designed to be general, additional work would be required to support a wider range of frameworks, models, and deployment environments in practice.

VI. Impact and Broader Implications

This study aims to address the lack of transparency regarding the environmental costs of AI systems. Overall, the middleware makes sustainability more visible and measurable to some extent.

This increased visibility has the potential to influence the developer behaviour. Since users have increased awareness of hidden costs, the middleware enables comparisons between design choices, which could encourage more sustainable practices. These include actions such as more efficient prompts, simpler pipelines or smaller models. These benefits are particularly useful in multi-agent systems, where complexity can increase rapidly, and environmental costs become harder to track.

The decision to build a middleware that exposes an API ensures that the tool can serve as a building block for different types of systems, and be easily integrated into existing architectures. In this report, we showed the system's integration with LangChain and a basic UI prototype. Still, with further development, it could be integrated within more complex systems to provide even more meaningful insights.

It must be noted that there is a risk of misinterpreting the tool's measurements. As stated numerous times in this report, the values we are returning are estimates. If users assume these estimates are exact, they may draw incorrect conclusions. This can be especially problematic in platform-level deployment, where measurements can reach many parties, implying that the transparency our tool provides must be paired with clear communication of its uncertainty.

Finally, the tool is not a solution to 'green AI', a subfield of AI that focuses on deploying models that prioritise, among other things, environmental sustainability. This tool is made as a lightweight, practical step focused on awareness at inference time.

VII. Threats to Validity

a) Construct validity: In this work, energy consumption is estimated using theoretical compute approximations based on the relation FLOPs $= 2P \cdot T$, which assumes dense transformer architectures and forward-pass inference. This simplification may not hold for all model types or implementations.

Furthermore, the conversion from compute to energy relies on peak GPU FLOPs and assumes ideal hardware efficiency, which does not always reflect real-world utilisation. Similarly, the use of a global average carbon intensity introduces additional approximation, as actual emissions depend on geographic location and time-dependent energy mixes, which are hard to obtain precisely.

Finally, the system measures energy only at the level of model inference. It does not account for other sources of energy consumption, such as data transfer, memory usage, or infrastructure overhead. As a result, the reported values represent partial estimates rather than complete measurements of system-wide impact.

b) Internal validity: In this work, the evaluation is based on scenario-based analysis rather than empirical user studies. As such, the results depend on assumptions about developer behaviour and how users would interpret and act upon the provided information.

The personas and scenarios are designed by the authors and may not accurately reflect real-world usage. Additionally, no comparison is made against a baseline (e.g. systems without the middleware or alternative approaches), making it difficult to isolate the effect of the proposed solution.

Finally, while the middleware is intended to influence developers' decision-making, no direct evidence is provided that developers would change their behaviour in response to the metrics it exposes.

c) External validity: The current implementation is developed and evaluated within the LangChain framework and supports a limited set of models, which may restrict applicability to other frameworks or model architectures.

Moreover, the evaluation is conducted on relatively simple example agents, which may not reflect the complexity of real-world systems. The effectiveness of the middleware in large-scale or production environments remains uncertain.

In addition, the personas used in the evaluation may not capture the full diversity of developers who interact with LLM-based systems, particularly across different domains, levels of expertise, and deployment settings. As a result, the generalisability of the findings should be interpreted with caution.

VIII. Future Work

Although we believe this study takes a step in the right direction towards transparent environmental metrics for LLM-based agent systems, there are still several avenues for future research.

A. Empirical User Studies

Due to time constraints, our evaluation uses self-constructed personas and scenarios to develop a realistic narrative of how our proposed solution would be used. However, this is a very limited way to evaluate a solution, since it is speculative and does not necessarily reflect how our solution will be used in a real-world setting. That is why an important next step would be to conduct a comprehensive empirical user study involving actual software/AI engineers. We could then design a controlled experiment to compare the design decisions of developers with access to our calculated environmental metrics with those without. This study would help validate whether increased transparency in energy use actually leads to more sustainable designs in real-world scenarios.

B. Agent-Aware Sustainability

Currently, our middleware's sole purpose is to provide developers with an estimate of their agents' environmental cost via a Python API. In the future, we could feed this data back into the agent so it can take environmental costs into account when deciding which model to call for a given prompt. This would allow the system to autonomously optimise its environmental impact by choosing smaller, more efficient models for simple tasks and using larger models only when strictly necessary, such as tasks requiring complex reasoning.

C. Refined Energy and Carbon Estimates

Our current estimation relies on some generalised assumptions, such as a global average carbon intensity of 0.45 kg CO_2/kWh and theoretical compute approximations based on the peak performance of a data centre-grade GPU. In the future, we could use more granular estimates to increase the precision of the returned information. We could, for instance, use more fine-grained local estimates, since carbon emissions per kilowatt-hour likely vary widely between countries and even within regions. Additionally, we can expand our estimates for different hardware profiles, ranging from consumer-level GPUs to data centre-level GPUs.

D. State-Specific Estimates

Currently, our middleware aggregates environmental data based on individual model calls. This could pose the problem that, if certain processes in a prompt cause a disproportionate environmental footprint, developers may not be able to pinpoint exactly where the excess energy expenditure comes from. In the future, we could provide state-specific estimates, enabling developers to analyse and optimise specific components of their agent architecture.

E. Framework Expansion

Finally, our solution is specifically implemented as a middleware for the LangChain framework. To increase the impact our tool can have in the AI community, future work could extend our solution to support other popular LLM frameworks, such as LlamaIndex [10], CrewAI [11] or Haystack [12].

IX. Conclusion

This work presented the design and evaluation of Jamanota, a LangChain middleware solution for tracking the environmental impact of inference in LLM-based agent systems. We provide a structured approach to enhance transparency within multi-agent pipelines, enabling a more fine-grained understanding of how different components of an agentic system contribute to overall environmental resource usage, and offering developers meaningful insights to support more sustainable design decisions.

The solution is an easy-to-use middleware distributed as a Python package, accompanied by user documentation. It encapsulates the full pipeline, from intercepting model calls and extracting execution metadata to applying our energy estimation model to estimate energy usage and carbon emissions and enabling flexible retrieval and aggregation of these metrics through an API. Additionally, we provided a prototype demonstrating how the middleware can be applied in a realistic multi-agent setting.

Our scenario-based evaluation demonstrated that our solution improves transparency and supports more informed, sustainable decision-making in the development of multi-agent systems with LangChain, particularly for more experienced developers. At the same time, our findings highlight important limitations: energy estimates cover only individual model calls rather than full system overhead, metrics lack reference points, the tool does not provide automated optimisation, and the current implementation is limited to LangChain.

In future work, empirical user studies with actual developers could provide grounded validation of whether increased transparency leads to measurably more sustainable design decisions. Beyond evaluation, the middleware itself could be extended with refined, region-aware carbon estimates, dynamically fine-tuned for different hardware profiles, and generalised to support other development frameworks beyond LangChain. Most ambitiously, feeding environmental metrics back into the agent's decision-making could allow agents to autonomously select smaller, more efficient models where possible, taking a concrete step towards more sustainable agentic systems.

References

[1] "Langchain docs." [Online]. Available: https://docs.langchain.com/oss/python/langchain/middleware/overview

[2] LangChain, "Langsmith." [Online]. Available: https://www.langchain.com/langsmith/observability

[3] Langfuse, "Langfuse: Open source llm observability." [Online]. Available: https://langfuse.com

[4] E. J. Husom, "Llm energy consumption monitoring," 2024. [Online]. Available: https://github.com/ejhusom/MELODI

[5] S. Biennier, J. Heijne, P. Lindhorst, and H. Sprangers, "Greenchat," Apr 2025.

[6] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020. [Online]. Available: https://arxiv.org/abs/2001.08361

[7] 2022. [Online]. Available: https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306

[8] [Online]. Available: https://thundersaidenergy.com/downloads/global-electricity-prices-vs-co2-intensities/

[9] M. B. Rosson and J. M. Carroll, "Scenario-based design," in *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*, J. Jacko and A. Sears, Eds. Mahwah, NJ: Lawrence Erlbaum Associates, 2002, pp. 1032–1050.

[10] J. Liu, "LlamaIndex," 11 2022. [Online]. Available: https://github.com/jerryjliu/llama_index

[11] J. Moura, "CrewAI," 1 2024. [Online]. Available: https://github.com/crewaiinc/crewai

[12] M. Rusic, "HayStack," 1 2020. [Online]. Available: https://github.com/deepset-ai/haystack