JouleDuel: Energy-Aware Programming Platform
Fostering Sustainable Software Engineering Through Competition

Alessandro Valmori, Bill Vi, Wilhelm Marcu, Maciej Bober, Frederik van der Els
Delft University of Technology
Email: {a.valmori, b.vi, w.p.a.marcu, m.j.bober-1, f.m.h.vanderels}@student.tudelft.nl

***Abstract*—As software systems grow in scale and complexity, their energy consumption has become a critical environmental concern. Despite the proliferation of competitive programming platforms such as LeetCode and Codewars, energy efficiency remains absent as an evaluation metric, leaving developers without incentive or tooling to write energy-aware code. We present JouleDuel, an energy-aware competitive programming platform designed to foster sustainable software engineering practices through gamified challenges. JouleDuel measures and compares the energy consumption of user-submitted solutions, enabling head-to-head competition on both correctness and efficiency. We describe the system architecture, energy measurement methodology, and implementation decisions, and discuss the challenges of producing fair and reproducible energy measurements across heterogeneous hardware. Our evaluation demonstrates that competitive framing meaningfully motivates participants to explore and adopt more energy-efficient algorithmic approaches. JouleDuel represents a step toward integrating sustainability as a first-class metric in software engineering education and practice.**

I. Introduction

The global software industry is a major and quickly growing source of greenhouse gas emissions is. This growth trajectory is expected to continue, driven by the continuous expansion of digital services and infrastructure, along with the recent immense computational demands of artificial intelligence workloads. Projections indicate that by 2040, the ICT sector could exceed more than 14% of the global carbon footprint, up from 1.6% in 2007 [4]. While it is the hardware that consumes energy, it is the software running on it which determines the efficiency. This reality has given rise to the field of sustainable software engineering, which seeks to understand and minimise the environmental impact of software systems throughout their lifecycle.

Understanding what makes software more or less sustainable is therefore an important undertaking. Research has shown that decisions made at every level of development can have a measurable influence on a program's energy consumption. For example, the *Flyweight* design pattern tends to reduce energy consumption by reusing objects instead of creating new ones, while patterns like *Decorator* have been shown to increase energy use due to additional method calls and object creation [14]. However, a recent empirical study complicates this picture, finding that design patterns themselves are energy-neutral; as the authors argue "software energy efficiency should rather be considered within the business logic embedded by a design pattern" [15]. This nuance highlights the need for tools that make the energy implications of implementation decisions visible. Moreover, the principles of green software design remain mostly ignored, and automated support for integrating them early in the software lifecycle is still limited [13]. This disconnect provides the motivation for this work: to explore new ways of making energy efficiency a tangible and engaging concern for the software development community.

Bridging this gap requires addressing a fundamental problem: developers currently lack the tools and incentives to assess the energy implications of their code. While existing skill development platforms manage to establish clear feedback loops for performance metrics such as execution speed and memory usage, they have yet to incorporate energy efficiency aspects. A similar platform focusing on energy optimisation could serve a dual purpose: it would educate developers about the concept of green code, and provide a space to benchmark and compare solutions based on their energy usage. Such a platform could help translate the abstract goal of sustainability into a concrete and engaging challenge.

This paper introduces JouleDuel, a competitive coding platform designed to rank solutions based on their energy efficiency. Our work makes two main contributions. The first is that we detail the design and implementation of the platform, including its architecture and energy measurement system. The second is that we provide an initial evaluation of the platform's design, through a scenario-based analysis, to see whether it is effective in raising awareness of energy-efficient coding practices.

The remainder of this paper is structured as follows: Section II gives an overview of energy measurement methodologies and gamification, Section III showcases the system design and architecture, Section IV describes the implementation details of JouleDuel, Section V presents our evaluation, Section VI discusses JouleDuel's role in the bigger picture, Section VII reviews related work, and Section VIII concludes with key takeaways.

II. Background

This section contextualises JouleDuel by giving a brief overview of the technical methodologies and challenges of measuring software energy consumption, as well as the role of gamification in developer education.

A. Measuring Software Energy Consumption

A prerequisite for any platform seeking to rank code by energy efficiency is a reliable method for measuring and

attributing energy consumption to an individual software process. This is not trivially accomplished since energy is a physical property of the hardware and is not directly tracked by the operating system, unlike other metrics such as execution time or memory allocation.

Historically, energy measurements required external power monitors which are intrusive, insufficiently accurate, and incapable of fine-grained, per process attribution [5]. The landscape shifted significantly with the introduction of hardware-level energy estimation interfaces, most notably Intel's Running Average Power Limit (RAPL) and AMD's equivalent Application Power Management. RAPL exposes energy consumption metrics for various hardware domains such as the CPU package, individual cores, and DRAM, through Model Specific Registers (MSRs). Even though RAPL merely produces estimates, empirical validations have demonstrated that these predictions are highly correlated with actual energy consumption [2].

Despite this, leveraging RAPL for our purposes still introduces several methodological challenges. The first is the issue of isolation and attribution. RAPL measures the energy consumption of hardware components, not individual software processes. When a program runs, the measured energy includes the target application, the operating system, and any background noise. Tools like PowerJoular [10] address this by combining RAPL data with Linux `perf_events` to estimate the proportional energy share of a specific process based on its CPU utilisation.

Another challenge is ensuring reproducibility and fairness. Software energy measurements are flaky and highly sensitive to confounding factors such as thermal throttling and tail energy consumption from previous measurements [6], [7]. However, fairness is not only about controlling the measurement environment but also accounting for variability introduced by the code itself. Low-level details, such as the choice of data structures, significantly impacts energy usage due to memory access patterns and cache misses. This gives rise to tools such as GreenC5 which automatically recommend the most energy efficient data structure for a given workload [3]. Similarly, the choice of programming language also plays an important role; compiled languages like C and Rust are usually more energy efficient than interpreted languages like python, largely due to the overhead of the runtime environment [9], [12]. Consequently, a platform like JouleDuel must carefully control the execution environment to provide fair and reproducible energy rankings for user submissions.

B. Gamification in Developer Education

In order for JouleDuel to be successful, it must not only measure energy, but also motivate its users to care about it. One way of doing this is through gamification, the concept of applying game design elements throughout a system to increase user engagement. For JouleDuel's purposes, gamification serves two main purposes: educating software engineers on sustainability principles and motivating energy-efficient programming behaviour.

In the context of programming education, meta-analyses have verified that gamification positively impacts student's motivation and engagement [11]. This realisation is important because leaderboards will be a central hook in JouleDuel to drive competition, meaning that they should be carefully designed to ensure information clarity and fairness. Other gamification elements which could be explored are points, levelling, and badges. These would serve as extrinsic motivation mechanisms (external recognition and rewards) which complement the intrinsic motivation (the desire for mastery) provided by JouleDuel's leaderboards.

Furthermore, gamification also strongly influences the behavioural patterns of professional software developers. A natural experiment on GitHub demonstrated that the presence of gamified elements, such as the daily activity streak counter, significantly altered developer habits. When the streak counter was removed, long running streaks of activity were abandoned and weekend contributions decreased [8]. The authors also urge caution that gamification can steer developer behaviour in unexpected directions, and an overemphasis on competition can lead to burnout or overwork. Therefore, these effects highlight that JouleDuel should not just foster competition for the sake of it, but instead, take a more friendly approach where developers can learn from one another.

III. JouleDuel: System Design

This section describes the design and architecture of JouleDuel. We begin by identifying the relevant stakeholders and based on their needs, subsequently propose a set of design goals to help guide JouleDuel's evolution. We then detail the system architecture and component interactions. Finally, we discuss how JouleDuel's design helps promote sustainable software engineering practices among users.

A. Stakeholder Analysis

JouleDuel is primarily aimed at software developers and Computer Science students who already engage with competitive coding platforms like LeetCode. These users are familiar with the concept of submitting solutions to algorithmic problems and receiving performance based feedback. This positions them perfectly to adopt energy efficiency as an additional optimization dimension, which our scoring metric centers around. For this group, the platform must offer a frictionless user experience, immediate and actionable feedback, and a sense of fair competition that motivates iterative improvement.

A secondary audience consists of educators in the field of software engineering who wish to incorporate energy-awareness into their teachings, thus concretizing the concept of sustainability through measurement in practical challenges rather than stopping at theoretical guidelines. For educators, the platform must serve as a reliable pedagogical tool that demonstrates how different implementation choices impact energy consumption.

JouleDuel is not designed for systems programming or high-performance computing experts, as existing tools such as Climbing Mont Blanc already serve that niche [1]. Instead,

JouleDuel targets the broader developer community, where awareness of software energy consumption is currently low and the potential for impact is high.

B. Design Goals and Principles

The needs identified in the stakeholder analysis directly motivate the following core design goals and principles of JouleDuel:

1) Measurement Fairness and Reproducibility: JouleDuel needs to be able to measure user's energy consumption independently of background system noise and other concurrent executions to ensure that energy rankings reflect the code's actual efficiency. Without this principle, users will quickly lose trust in the platform which undermines both of its competitive and educational aspects.

2) Security and Isolation: JouleDuel needs to execute arbitrary, untrusted user code which poses major security risks for the host infrastructure. The system must be able to isolate this code execution to prevent malicious activity while still delivering energy readings. Without this protection, the platform would be vulnerable to bad actors who may exploit infinite loop bugs to compromise the availability of the system.

3) Actionable Feedback: JouleDuel must provide clear feedback regarding the energy consumption of submitted solutions so that users can meaningfully compare different algorithmic approaches. This is the principle that will allow the platform to provide an engaging, iterative feedback loop to help drive the learning process.

C. System Architecture

JouleDuel employs a modular architecture comprising four primary components, as illustrated in Figure 1.

The frontend and backend are served by a single monolithic web application that provides both the user interface and the API routes that drive the submission pipeline. The user interface includes a problem browser, a code editor, and a leaderboard. The API layer coordinates validation, energy measurement, and database interaction.

Before any energy profiling takes place, submitted code must pass functional validation against a set of test cases. This validation runs in a sandboxed code execution engine that isolates user code in disposable containers, preventing malicious patterns such as infinite loops, filesystem access, or network calls from affecting the host system.

Once a solution passes validation, its energy consumption is measured by a host-level power monitoring tool that reads hardware power counters. The measurement process involves baseline calibration, warmup runs, and multiple measured executions to produce a reliable energy score.

All persistent state, including user accounts, problem definitions, test cases, and submission results, is stored in a relational database.

The main interaction flow is as follows: when a user submits a solution, the API receives the code and problem identifier, wraps the solution in template code that runs the relevant test cases, and forwards it to the execution engine for validation.

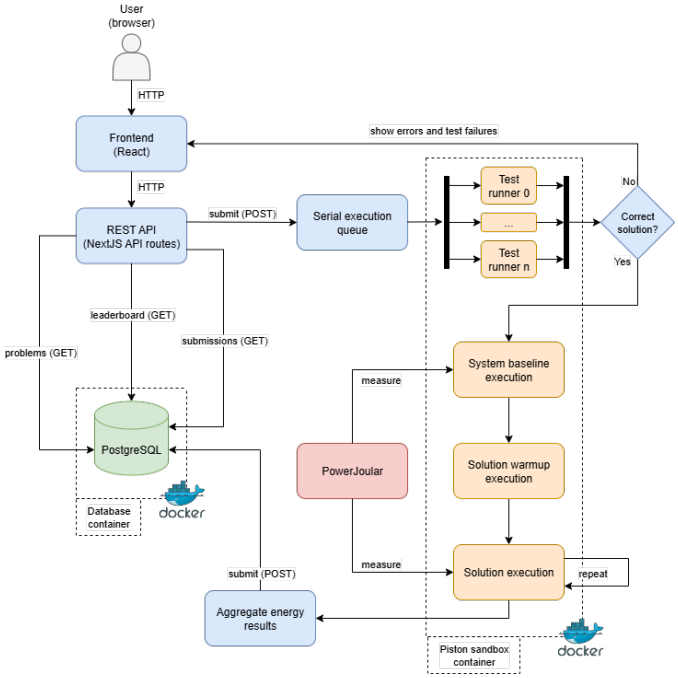

Fig. 1. JouleDuel system architecture

If validation succeeds, the power monitoring tool is invoked to profile energy consumption while the code executes in controlled loops on the host. The results are then persisted and returned to the user interface.

D. Promoting Sustainable Practices

JouleDuel promotes sustainable software development through several reinforcing mechanisms. The most fundamental of these is making energy consumption visible in the first place. Developers rarely, if ever, encounter energy cost information during routine development, since conventional toolchains simply do not surface it. By presenting energy metrics alongside correctness verification, the platform forces an awareness that functionally equivalent implementations of the same algorithm can differ substantially in their energy footprints.

To get the user up to speed with green design patterns, a dedicated page called "code help" is available. This page explains how both time and space complexity are crucial to consider when writing energy-efficient code. Time complexity matters because using suitable algorithms reduces the amount of work the program has to do. The faster the program finishes, the less energy is consumed. Space complexity matters because using as little space as possible reduces memory allocations and accesses. These memory operations are not free, and minimising them reduces energy consumption. In the explanatory page, the user is therefore informed that writing energy-efficient code boils down to finding the optimal balance between the time and space complexity of the program.

The leaderboard builds on this trade-off by introducing a competitive element. As users are ranked by their solutions'

energy efficiency, there is a concrete motivation to revisit and refine submissions. Developers are encouraged to experiment with different algorithmic strategies and coding patterns in pursuit of lower energy consumption, rather than treating a passing solution as sufficient. Users are able to use the leaderboard scores to deduce how close their solution is to the optimal solution. For example, if there is a sudden large difference in energy consumption between two consecutive rankings, it may indicate the boundary between solutions with different complexities. Based on this, users can decide whether to pursue a more optimal solution or move on to a different problem.

However, while the competitive element does incite the user to write better energy-efficient code, it does not explain to them how to attain these rankings. That is, for a particular program, the user may get stuck on their suboptimal implementation, without a clear way to improve. For this reason we also provide the user with the ability to request AI feedback for their code. Below the submission's energy consumption statistics, there is a button which the user can press to request an LLM to provide algorithmic feedback on their code. The LLM has access to the problem context, the optimal reference time and space complexities, and the user's code. The AI first determines the time and space complexity of the submitted program, after which it compares it against the reference complexities. In the feedback, the detected complexities are always reported. Moreover, if the submitted code is deemed optimal, this is reported to the user. If the submitted code is suboptimal, a short hint to improve the program is additionally given. For instance, an example output for a suboptimal submission could be:
[1]

```
Time Complexity: O(n^2)
Space Complexity: O(1)
Feedback: The time complexity can be improved
    from O(n^2) to O(n) by using a hash map (
    dictionary) to store and look up the
    complements of the numbers as you iterate
    through the array.
```

IV. Implementation

A. Technology Stack

The architecture described in Section III identifies four component roles: a web application, a sandboxed execution engine, a power monitoring tool, and a relational database. In this section we detail the specific technologies chosen for each and the rationale behind them.

The web application is built on Next.js, a React-based framework whose integrated API routes eliminate the need for a separate backend server. TypeScript is used throughout the codebase to provide compile-time type checking across both frontend and backend code. The code editor uses Monaco, the same component that powers Visual Studio Code, providing syntax highlighting and auto-completion that should feel familiar to most developers.

Piston was chosen over alternatives such as Judge0 for its simpler self hosted deployment and active maintenance. It is important to note that while containerisation introduces some runtime overhead, Piston is used exclusively for functional validation; energy profiling deliberately bypasses the sandbox entirely, as described in Section IV-B.

Energy measurement relies on PowerJoular, which reads directly from Intel RAPL hardware counters exposed through the Linux `powercap` interface. Unlike application-level profilers that estimate energy from CPU cycle counts, RAPL provides direct hardware measurements of package, core, and DRAM power consumption. PowerJoular was selected for its its simplicity and its ability to monitor power either system wide or per process.

For the relational database we chose PostgreSQL. The reasons being its reliability and its support for JSON columns, which are used to store test case definitions and problem examples. Database access is managed through Drizzle ORM, which provides full type inference from the schema definition as well as unmatched speed amongst other alternatives in the ORM category.

The entire system is deployed on a bare metal server running Alma Linux 10. RAPL interface behaviour is highly variable across different Linux distributions and kernel versions. Alma Linux provides a stable, long term support platform with a conservative kernel release policy that ensures the behaviour of the hardware power counters remains consistent over time. It is crucial in this system's context that comparability stays consistent to support the objectiveness and durability of the submission leaderboard, as unexpected changes would alter the scoring policy and harm user trust over time.

B. Energy Measurement

Producing fair and reproducible energy measurements for arbitrary user code was a major challenge throughout JouleDuel's development. Because energy consumption is highly sensitive to background system noise, thermal state, and execution duration, we employ a rigorous, multi-step measurement pipeline to isolate the user's code and normalise the results.

At the core of this pipeline is PowerJoular, which reads hardware-level power data from Intel RAPL via the Linux `powercap` interface. To attribute energy to a specific submission, PowerJoular monitors the process ID (PID) of the execution engine. It calculates the proportion of total CPU cycles consumed by that PID using Linux statistics from `/proc/stat` and `/proc/pid/stat`, and allocates the corresponding fraction of the total RAPL package energy to the process [10].

To ensure fairness, the measurement pipeline is strictly serialised. We use a promise-based mutex queue to guarantee that only one submission is profiled at any given time, thus preventing concurrent executions from competing for CPU resources or polluting the cache. Furthermore, the execution

[1]Note that we have taken significant measures to prevent jailbreaking. The LLM is provided with a concise and lengthy system instruction to prevent abuse. While we cannot prove it is immune, with our own testing we were unsuccessful to jailbreak it.

is deliberately bypassed from the Piston sandbox during this phase and run directly on the host. This is necessary because containerization obscures the PID from PowerJoular's process-level attribution logic. Crucially, by the time a submission reaches this phase, it has already passed functional validation in the sandboxed environment, confirming it is neither an infinite loop nor otherwise malicious, as described in Section IV-A. To further reduce variance, the execution can be pinned to a specific CPU core using the Linux `taskset` utility, minimizing context-switching overhead and producing more reliable results.

The measurement lifecycle for a single submission proceeds as follows:

1) **Baseline Calibration:** The system first measures the idle baseline power for two seconds. This baseline wattage is later subtracted from the active measurements to isolate the energy delta introduced by the user's code.
2) **Warm-up:** The user's function is executed for three warm-up runs. This primes the CPU cache and ensures that any initial memory allocation overhead or lazy-loading mechanisms do not skew the final results.
3) **Measurement Loop:** The function is then executed for ten independent measurement runs. Modern CPUs execute simple algorithmic functions in fractions of a millisecond which is too fast for RAPL's polling frequency to capture accurately. Because of this, the function is wrapped in a continuous `while` loop that runs for a guaranteed minimum duration of three seconds per run. The wrapper script counts the exact number of function executions completed within this time.
4) **Aggregation and Normalization:** For each of the ten runs, the baseline energy is subtracted from the total measured energy to yield the delta joules. This delta is then divided by the number of function executions to calculate the energy cost per single execution of the function. Finally, to filter out transient system spikes, JouleDuel takes the median of these ten runs.

This methodology ensures that the final ranking metric is independent of the host's background noise and robust against run-to-run variance, providing a reliable basis for the platform's competitive leaderboard.

C. User Interface

The user interface of JouleDuel is designed to minimise cognitive load and provide a frictionless transition for developers familiar with existing coding platforms. By adhering to established human-computer interaction (HCI) principles, the platform ensures that users can focus entirely on algorithmic problem-solving and energy optimization rather than navigating a new interface.

The primary workspace, shown in Figure 2, is divided into a split-pane layout. The left pane houses the code editor, the top right pane presents the problem description, constraints, and example test cases, and the bottom right pane displays code output and energy results. To leverage users' existing mental models, the code editor is powered by Monaco which

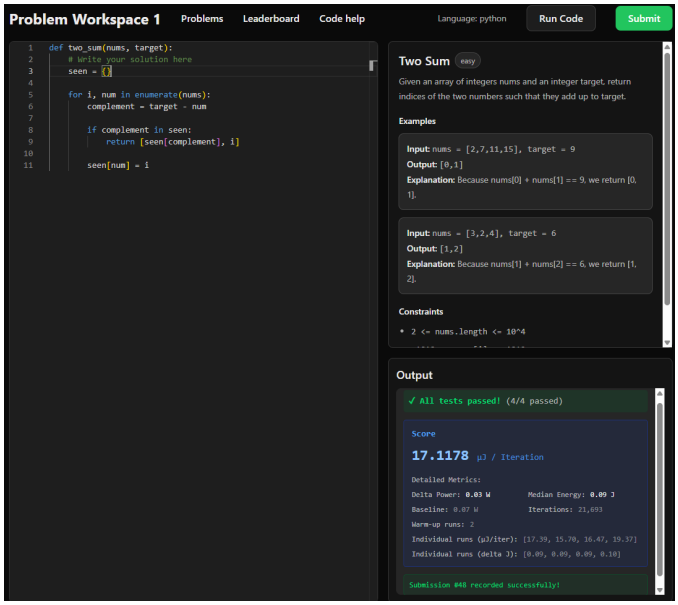

Fig. 2. User interface for solving a problem

is the same underlying component used in Visual Studio Code. This provides familiar syntax highlighting, auto-completion, and indentation behaviours, significantly reducing the friction of writing code in a browser environment.

The leaderboard, shown in Figure 3, serves as the primary gamification element. Unlike traditional platforms that only display a user's single best submission, JouleDuel's leaderboard displays all successful submissions ranked by their energy consumption (μJ/exec). This design choice transforms the leaderboard from a simple ranking mechanism into an exploratory learning tool. Users can browse highly ranked submissions to study the specific coding patterns and algorithmic choices that led to superior energy efficiency, fostering a collaborative learning environment alongside the competitive framing.

V. Evaluation

To evaluate the utility and limitations of JouleDuel's design, we employ a scenario-based analysis using four distinct user personas. This method allows us to critically examine how different users interact with the platform, what benefits they derive, and where the current implementation falls short of their needs.

A. Alice: The Novice Bachelor's Student

Background: Alice is a first-year Computer Science student who is just learning the basics of Python. She occasionally introduces syntax errors and basic logic bugs. She is primarily focused on getting her code to pass the automated test cases.

Scenario: Alice attempts an easy problem, and after several failed attempts, finally submits a brute-force nested loop solution that passes validation. She sees the energy ranking of her solution but does not immediately understand what it means or how to improve it. She then goes to the "Code

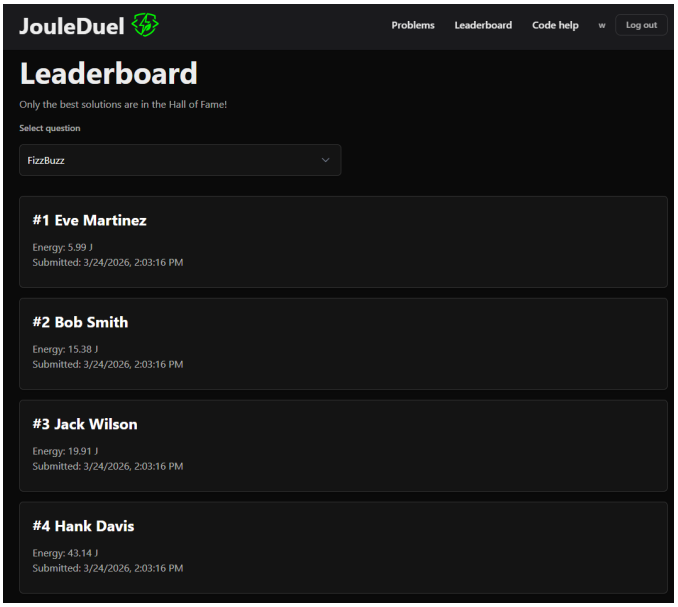

Fig. 3. Public leaderboard for the 'FizzBuzz' problem

Help" page, which gives her some general pointers for how to improve.

Benefits: Alice is exposed to the concept of software energy consumption early in her education, planting a seed of awareness that code has a physical cost. She may recall this knowledge at a later point in her studies when she learns about designing optimal code.

Limitations: The platform implicitly assumes the user has basic programming skills. Because Alice is still focusing on learning the basics, she may not reap the full benefits of understanding how to optimise for energy. The lack of explicit educational content offered by JouleDuel also means that Alice might experience additional friction points during her experience.

B. Bob: The Competitive Master's Student

Background: Bob is a Master's student who is approaching the end of his studies. Consequently, he is in the process of preparing for technical interviews and has extensive experience with online programming learning platforms, often ranking high in the leaderboards. Bob is aware of the concept of sustainable software engineering from his courses but has never had a concrete reason to apply it.

Scenario: Bob discovers JouleDuel and is immediately drawn in by its competitive framing. He works through several problems and treats each energy score as a target to beat. Bob often revisits his code to attempt different approaches, driven by the desire to climb the leaderboard. Bob eventually finds himself actively reading about Python's memory model and runtime behaviour, gaining new understanding to further improve his score.

Benefits: JouleDuel successfully captivates Bob's competitive spirit and manages to teach him about energy-aware programming without him realising. The leaderboard provides a persistent, social motivation that keeps Bob engaged beyond a single session, thus allowing Bob to continue to learn and develop new skills for an extended period of time.

Limitations: Bob's engagement is ultimately constrained by the size of the problem set. Due to his knowledge and competitiveness, he quickly finishes all of the available problems on offer and needs to wait for the developers to add more problems to the platform. Bob is also somewhat disappointed that the current problems did not provide any novelty, since he had already seen similar problems on other coding platforms. Bob would have also liked to be able to submit his own algorithmic challenges or engage with problems submitted by other users.

C. Cleo: The Software Engineering Educator

Background: Cleo teaches a university course on sustainable software engineering. Cleo struggles to move students beyond abstract guidelines and she wants a practical tool to make lectures and assignments more engaging and exciting.

Scenario: Cleo assigns a problem on JouleDuel as homework. During the lecture, Cleo presents the platform's public leaderboard to showcase the energy differences between solutions that achieve different time and space complexities.

Benefits: The platform provides Cleo with learning material she can present to her students. The public visibility of all submissions allows Cleo to use highly ranked solutions as pedagogical examples in class. Furthermore, the platform also provides a simple way for students to quickly engage with programming and energy optimisation without needing to go through the difficulties of setting up a programming environment or energy profiler.

Limitations: Cleo would like to demonstrate how language choice impacts energy consumption by having students solve the same problem in Python, Java, and C. However, JouleDuel's current implementation is restricted to Python, limiting its utility when dealing with projects that involve multiple programming languages. As an educator, Cleo is also disappointed that the platform does not prioritise offering educational material, making it hard for her to fulfil concrete learning objectives. Additionally, Cleo is restricted to using JouleDuel in advanced classes, where students are already familiar with programming, rather than introductory classes where students might struggle with the open-ended nature of the platform.

D. Dan: The Enterprise Developer

Background: Dan is a backend engineer working on cloud-native microservices. His company has recently put him in charge of reducing cloud compute costs and carbon emissions. He joins JouleDuel to gain new skills and build intuition for writing greener code.

Scenario: Dan works through various problems on offer, experimenting with different advanced Python features to see which one yields the lowest energy consumption.

Benefits: Dan finds that JouleDuel is useful for algorithmic benchmarking. He uses it to compare alternatives to different

algorithmic approaches currently used by his company. He uses the leaderboard to gauge the effectiveness of his optimisations and selects the best solution to try to minimise costs for his company.

Limitations: Dan quickly realises JouleDuel cannot provides insights into all aspects of his day-to-day work. His enterprise may involve bottlenecks with database queries, network latency, and framework overhead. These aspects cannot be modelled in JouleDuel's isolated, single-function sandbox.

VI. Discussion

The development and evaluation of JouleDuel reveal several broader implications for the field of sustainable software engineering, particularly regarding how developers interact with and internalise the concept of energy efficiency.

A. JouleDuel as a Complement to Existing Tooling

The field of sustainable software engineering has produced a variety of tools aimed at reducing energy consumption, ranging from static code analyzers to production-level profilers and automated compliance frameworks. However, these tools are typically deployed late in the software development lifecycle, often as compliance checks or debugging instruments for existing systems. They assume the developer already understands what to look for and how to fix it.

JouleDuel occupies a distinct and complementary niche within this ecosystem: it is not a profiler for production code, but rather a foundational training ground. By isolating algorithmic logic from the complexities of real-world I/O, network latency, and framework overhead, the platform allows developers to build an intuitive understanding of how low-level implementation choices—such as data structure selection and memory allocation—impact energy consumption. This intuition can then be carried forward into their professional work, making them more effective users of the complex, production-grade profiling tools that already exist in the industry.

B. The Role of Competition in Normalising Sustainability

Beyond technical tooling, JouleDuel explores the cultural aspects of software engineering education. By integrating energy measurement into a competitive coding environment, JouleDuel leverages the established behavioural drivers of online programming learning platforms. In these traditional platforms, execution time and memory usage are normalised as the primary indicators of code quality, implicitly teaching developers that these are the only metrics that matter.

JouleDuel extends this paradigm by elevating energy efficiency to the same level of visibility and importance. If energy rankings become as culturally ingrained as execution speed, it could fundamentally shift the default mindset of an entire generation of developers. Rather than treating sustainability as an afterthought or a specialised domain, developers trained in such environments are more likely to internalise energy efficiency as a core component of algorithmic problem solving.

C. Limitations and Future Work

While JouleDuel successfully establishes a proof-of-concept for energy-aware competitive coding, its current implementation has two main limitations that can be addressed in future work. The first is that JouleDuel is currently restricted to Python. While Python's popularity makes it an excellent starting point, its managed runtime obscures many low-level hardware interactions. Future iterations must support compiled languages like C, C++, and Rust, as well as other managed languages like Java. This would serve to broaden the platform's appeal so that it may spread to more developers in different communities.

The second limitation lies in the scalability of the measurement pipeline. To guarantee fairness, JouleDuel relies on a strictly serialized execution queue. While effective for a small user base, this architecture inherently limits the platform's scalability. During peak usage, the queue wait times could grow exponentially, disrupting the rapid feedback loop that competitive programmers expect. Future work must investigate distributed measurement architectures, potentially utilising fleets of identical bare-metal worker nodes to parallelise energy profiling without compromising reproducibility.

VII. Related Work

Online coding platforms have become a popular tool used to enhance modern software education and skill refinement. Well known examples of existing platforms include LeetCode [17] and Codewars [16] which provide gamified environments where users solve algorithmic puzzles to improve their critical thinking skills and prepare for technical hob interviews.

These platforms evaluate submitted code against predefined test cases, and if correct, further ranks solutions based on specific performance metrics. For example, LeetCode ranks a user's submission by showing the percentile of execution time and memory usage compared to all other accepted submissions. This competitive benchmarking encourages developers to iteratively refine their code to achieve more optimal solutions.

However, none of the mainstream competitive coding platforms measure or rank solutions based on energy efficiency. Consequently, these platforms are training developers to optimise for speed and memory, but not for sustainability. While execution time and energy consumption are often strongly correlated, they are not equivalent and they give no indication on the sustainability and environmental impact of the implementation. For example, aggressive parallelisation can reduce the execution time while increasing total energy consumption due to thread management overhead.

The closest prior work in this domain is Climbing Mont Blanc (CMB), an open online judge used for training in energy efficient programming [1]. It evaluates C and C++ programs submitted by users based on time, energy used, and energy-delay product (EDP). While CMB seems to align heavily with our goals and use cases, its main limitation is its highly specialised scope: it was designed specifically for training in parallel programming on heterogeneous multicore processors

rather than for general-purpose algorithmic problem solving. JouleDuel builds upon the foundation of CMB but targets the broader audience of mainstream competitive coding platforms.

VIII. Conclusion

As the environmental impact of the software industry continues to grow, equipping developers with the tools and intuition to write energy-efficient code is no longer optional. This paper introduced JouleDuel, a novel competitive programming platform that ranks algorithmic solutions based on their hardware-measured energy consumption. By integrating Intel RAPL measurements with a strictly serialised, container-isolated execution pipeline, JouleDuel successfully translates the abstract principles of sustainable software engineering into a concrete, gamified challenge.

Our scenario-based evaluation demonstrated that framing energy efficiency as a competitive metric effectively motivates developers to explore the physical costs of their implementation choices. While the current implementation faces limitations regarding language support and scalability, it establishes a foundation which can be expanded upon. Ultimately, JouleDuel proves that when energy consumption is made visible, measurable, and competitive, developers will actively optimize for it, marking a necessary step toward a more sustainable software ecosystem.

Acknowledgment

We would like to thank Enrique Barba Roque for his supervisory role throughout the project.

References

[1] L. Natvig, T. Follan, S. Støa, S. Magnussen, and A. García-Guirado, "Climbing mont blanc - a training site for energy efficient programming on heterogeneous multicore processors," *ArXiv*, vol. abs/1511.02240, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:15309294.

[2] S. Desrochers, C. Paradis, and V. M. Weaver, "A validation of dram rapl power measurements," ser. MEMSYS '16, Alexandria, VA, USA: Association for Computing Machinery, 2016, pp. 455–470, ISBN: 9781450343053. DOI: 10.1145/2989081.2989088.

[3] J. Michanan, R. Dewri, and M. J. Rutherford, "Greenc5: An adaptive, energy-aware collection for green software development," *Sustainable Computing: Informatics and Systems*, vol. 13, pp. 42–60, 2017, ISSN: 2210-5379. DOI: https://doi.org/10.1016/j.suscom.2016.11.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210537916300403.

[4] L. Belkhir and A. Elmeligi, "Assessing ict global emissions footprint: Trends to 2040 & recommendations," *Journal of Cleaner Production*, vol. 177, pp. 448–463, 2018, ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2017.12.239.

[5] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, Mar. 2018, ISSN: 2376-3639. DOI: 10.1145/3177754.

[6] L. Ardito, R. Coppola, M. Morisio, and M. Torchiano, "Methodological guidelines for measuring energy consumption of software applications," *Scientific Programming*, vol. 2019, no. 1, p. 5 284 645, 2019. DOI: https://doi.org/10.1155/2019/5284645.

[7] L. Cruz, *Green software engineering done right: A scientific guide to set up energy efficiency experiments*, http://luiscruz.github.io/2021/10/10/scientific-guide.html, Blog post., 2021. DOI: 10.6084/m9.figshare.22067846.v1.

[8] L. Moldon, M. Strohmaier, and J. Wachs, "How gamification affects software developers: Cautionary evidence from a natural experiment on github," May 2021, pp. 549–561. DOI: 10.1109/ICSE43902.2021.00058.

[9] R. Pereira et al., "Ranking programming languages by energy efficiency," *Science of Computer Programming*, vol. 205, p. 102 609, 2021, ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2021.102609.

[10] A. Noureddine, "Powerjoular and joularjx: Multi-platform software power monitoring tools," in *2022 18th International Conference on Intelligent Environments (IE)*, 2022, pp. 1–4. DOI: 10.1109/IE54923.2022.9826760.

[11] Z. Zhan, L. He, Y. Tong, X. Liang, S. Guo, and X. Lan, "The effectiveness of gamification in programming education: Evidence from a meta-analysis," *Computers and Education: Artificial Intelligence*, vol. 3, p. 100 096, Sep. 2022. DOI: 10.1016/j.caeai.2022.100096.

[12] A. Gordillo et al., "Programming languages ranking based on energy measurements," *Software Quality Journal*, vol. 32, no. 4, pp. 1539–1580, Jul. 2024, ISSN: 0963-9314. DOI: 10.1007/s11219-024-09690-4.

[13] N. Ahuja et al., "Automatically assessing software architecture compliance with green software patterns," in *2025 IEEE/ACM 9th International Workshop on Green and Sustainable Software (GREENS)*, 2025, pp. 68–75. DOI: 10.1109/GREENS66463.2025.00015.

[14] K. Lano, L. Alwakeel, and Z. Rahman, "Design patterns for software sustainability," English, in *Pattern Languages of Programs*. Feb. 2025. DOI: 10.64346/PLoP2024p10.

[15] A. Noureddine and O. Le Goaer, "Investigating the impact of software design patterns on energy consumption," in *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*, 2025, pp. 153–163. DOI: 10.1109/ICSA65012.2025.00024.

[16] Codewars. "Codewars: Achieve mastery through coding challenge," Accessed: Mar. 31, 2026. [Online]. Available: https://www.codewars.com/.

[17] LeetCode. “Leetcode - the world’s leading online programming learning platform,” Accessed: Mar. 31, 2026. [Online]. Available: https://leetcode.com/.