A Manifesto for Sustainable Computing

Principles for Building Software in the Age of Climate Awareness

David Jean Charlot, PhD Open Interface Engineering, Inc. (openIE) University of California, Santa Barbara (UCSB) david@openie.dev | dcharlot@ucsb.edu

February 2026 | Version 1.0

Abstract

We are software developers, engineers, architects, and creators. We build the digital infrastructure that powers modern civilization. Our code runs on billions of devices, in millions of data centers, consuming a significant and growing share of humanity's energy resources. Data centers alone now consume **240-340 TWh** annually, with projections reaching **1,000 TWh by 2026** [1]. Global ICT could account for **up to 8% of electricity demand by 2030** [2]. This manifesto proposes ten principles for building energy-aware software—not as restrictions, but as a framework for building better software that respects the finite resources of our planet while delivering value to its users.

Keywords: sustainable computing, energy efficiency, software engineering principles, green computing, programming languages, environmental impact

1. The Principles

1.1 Energy is a First-Class Concern

We will treat energy consumption as a primary metric, alongside correctness, performance, and maintainability.

For too long, energy has been invisible in software development. We measure test coverage, code complexity, and response latency, but rarely ask: how much energy does this code consume?

Research shows that the same algorithm can consume **75x more energy** depending on implementation language [3]. These differences compound across billions of devices and millions of servers.

We commit to: - Measuring energy consumption alongside other metrics - Setting energy budgets for our applications - Treating energy regressions as seriously as performance regressions

Energy awareness begins with energy visibility.

1.2 Efficiency Serves Users

We recognize that energy-efficient software benefits everyone: users, organizations, and the planet.

Efficient software: - Extends battery life on mobile devices - Reduces cloud computing costs - Decreases data center cooling requirements - Lowers the carbon footprint of digital services

When we optimize for energy, we optimize for our users. A phone that lasts all day, an application that doesn't spin fans, a service that costs less to run: these are features users value, whether or not they think about them explicitly.

1.3 Simplicity is Sustainable

We will resist unnecessary complexity, recognizing that every abstraction has an energy cost.

Software complexity compounds. Each layer of abstraction, each framework dependency, each indirection adds overhead. Niklaus Wirth observed that “software is getting slower more rapidly than hardware is getting faster” [4].

We commit to: - Questioning whether each dependency is truly necessary - Preferring simple solutions over clever ones - Removing code that no longer serves a purpose - Measuring before adding abstraction

The most energy-efficient code is often the code that doesn’t exist.

1.4 Hardware Deserves Respect

We will learn how our code interacts with hardware, and design with hardware efficiency in mind.

Modern hardware is remarkably efficient when used correctly. A CPU in idle state consumes a fraction of its peak power. A GPU can perform certain computations at **30–80x better energy efficiency** than a CPU [5]. Memory access patterns dramatically affect cache efficiency: DRAM access consumes **~100x more energy** than cache access [6].

We commit to: - Understanding the hardware our code runs on - Using specialized hardware when appropriate - Writing cache-friendly code - Allowing hardware to enter low-power states

Ignorance of hardware is not an excuse for waste.

1.5 Defaults Matter

We will choose energy-efficient defaults, while allowing users to override when necessary.

Most users accept defaults. A framework’s default configuration, a language’s default behavior, a library’s default settings: these choices affect millions of deployments.

We commit to: - Setting energy-efficient defaults in our tools and libraries - Making the efficient path the easy path - Documenting the energy implications of configuration choices

Good defaults multiply good practices across the ecosystem.

1.6 Measurement Enables Improvement

We will build and share tools that make energy visible and measurable.

You cannot improve what you cannot measure. The software industry lacks standardized, accessible tools for energy measurement. Profilers measure time. Monitoring measures throughput. But energy remains a mystery to most developers.

Modern hardware provides measurement capabilities (Intel RAPL, ARM Energy Probe) that most developers have never used [7].

We commit to: - Building energy measurement into our development tools - Sharing energy benchmarks and profiles - Creating standards for energy reporting - Making energy data actionable

Visibility creates accountability.

1.7 Education Precedes Transformation

We will teach and learn sustainable computing practices, recognizing that change requires understanding.

Most computer science curricula ignore energy efficiency. Most programming books omit it. Most code reviews don't consider it. We cannot expect change from developers who have never been taught to think about energy.

We commit to: - Incorporating energy efficiency into education - Mentoring others in sustainable practices - Writing about and speaking about energy-aware development - Leading by example in our own codebases

Knowledge spreads through communities.

1.8 Progress Over Perfection

We will make incremental improvements rather than waiting for perfect solutions.

Sustainability is not binary. There is no threshold below which energy consumption is "sustainable" and above which it is not. Every improvement helps. Every watt saved matters.

We commit to: - Making small improvements continuously - Celebrating progress rather than demanding perfection - Sharing partial solutions that others can build upon - Treating sustainability as a journey, not a destination

The goal is direction, not arrival.

1.9 Collaboration Accelerates Change

We will share knowledge, tools, and practices with the broader community.

No organization can solve computing's energy challenge alone. The tools we build, the techniques we discover, the lessons we learn: these have value beyond our immediate projects.

We commit to: - Open-sourcing energy-related tools when possible - Publishing our energy benchmarks and findings - Participating in industry efforts toward sustainable computing - Welcoming contributions from all who share these values

Together we can move faster than alone.

1.10 Technology Serves Humanity

We will remember that sustainable computing is ultimately about preserving our planet for future generations.

Software is not an end in itself. It exists to serve human purposes: to connect, to create, to solve problems, to improve lives. The energy our software consumes comes from somewhere. The heat it generates goes somewhere. The climate impact is real.

The International Energy Agency projects that without significant efficiency improvements, data center energy consumption could triple by 2030 [1]. The choices we make today about how software is built will shape that trajectory.

We build software because we believe technology can make the world better. Let us ensure that our creations do not undermine the world we are trying to improve.

---

2. Implications and Recommendations

These principles describe not where we are, but where we aim to be. We recommend the following concrete steps for practitioners:

1. **Measure your software's energy consumption** and share what you learn
2. **Make one improvement** to energy efficiency in your current project
3. **Teach one colleague** about sustainable computing practices
4. **Advocate for energy awareness** in your organization's technical decisions
5. **Contribute to community efforts** at https://joule-lang.org/community

---

References

[1] International Energy Agency. "Data Centres and Data Transmission Networks." *IEA*, 2024. https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks

[2] Andrae, A.S.G., Edler, T. “On Global Electricity Usage of Communication Technology: Trends to 2030.” *Challenges*, vol. 6, no. 1, pp. 117-157, 2015. https://doi.org/10.3390/challe6010117

[3] Pereira, R., Couto, M., Ribeiro, F., et al. “Energy Efficiency across Programming Languages.” *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE)*, 2017. https://joule.openie.dev/research/energy-efficiency-languages

[4] Wirth, N. “A Plea for Lean Software.” *Computer*, vol. 28, no. 2, pp. 64-68, 1995. https://doi.org/10.1109/2.348001

[5] Jouppi, N.P., et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit.” *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017. https://doi.org/10.1145/3079856.3080246

[6] Drepper, U. “What Every Programmer Should Know About Memory.” *Red Hat, Inc.*, 2007. https://www.akkadia.org/drepper/cpumemory.pdf

[7] Khan, K.N., Hirki, M., Niemi, T., et al. “RAPL in Action: Experiences in Using RAPL for Power Measurements.” *ACM TOMPECS*, vol. 3, no. 2, 2018. https://doi.org/10.1145/3177754

[8] Charlot, D.J. “Metabolic Cascade Inference: Hardware-Aware Adaptive Routing for Energy-Efficient AI.” OpenIE Technical Report, January 2026.

[9] Charlot, D.J. “The AI Inference Crisis: Why Current Approaches Are Unsustainable.” OpenIE Technical Report, January 2026.

[10] Charlot, D.J. “Cortex: Neural-Symbolic Programming for Energy-Efficient Code Execution.” OpenIE Technical Report, January 2026.

Contact: david@openie.dev | dcharlot@ucsb.edu **Latest Updates**: https://openie.dev/projects/joule