How do we “green the digital”?

DEVS AND PLATFORM ENGINEERS

1. **Write energy-efficient code.** Choose efficient algorithms and data structures. Avoid bloated logic and excessive loops. Be VERY cautious with AI-generated code. The Software Carbon Intensity (SCI) standard from the GSF gives an actionable approach to measure the carbon impacts of software systems.

2. **Minify and optimise code.** Use build tools to remove redundant code from HTML, CSS, and JavaScript. Use static code review tools like carbon-related SonarQube plugins.

3. Consider using **greener languages** like Rust and Go.

4. **Explore GreenOps** (which also goes by other names, e.g. SusDevOps, Sustainable DevOps), to integrate sustainability data into day-to-day development and engineering decisions.

5. **Green your CI/CD pipelines.** Reduce unnecessary builds, eliminate redundant steps, and explore grid-aware computing.

6. **Consider demand shaping.** Your service can adjust according to how much clean energy is available in the grid. See e.g. Branch magazine, where images only display by default when the grid is green.

7. **Apply Green Infrastructure as Code practices.** Modularise, right-size, and use auto-shutdown defaults to avoid over-provisioning.

8. **Monitor energy use in real time.** Tools like Intel RAPL, Scaphandre, and OpenTelemetry + Kepler help visualise power usage across services and machines.

ML ENGINEERS AND DATA SCIENTISTS

1. **Measure models' carbon footprints.** Integrate tools like CodeCarbon or CarbonTracker into your training and inference pipelines to estimate CO_2 emissions in real time based on hardware usage and electricity grid carbon intensity. For cloud or HPC workloads, you can use tools like GreenAlgorithms4HPC to estimate emissions using compute job logs, runtime, and facility data.

2. **Use lightweight models.** Use the AI Energy Score leaderboard to compare models not just by accuracy but by energy consumption and emissions.

3. **Aim for proportionality.** Avoid unnecessary training or retraining—use existing models and fine-tuning where possible. BUT be mindful of trade-offs. A large pre-trained model might meet your needs, but a smaller model could offer similar performance with significantly lower inference costs.

4. **Optimise training cycles.** Reduce frequency, use efficient model architectures, and minimise unnecessary hyperparameter searches. Stop training when performance plateaus, or even earlier if it is sufficient for the task at hand. Early stopping, efficient architectures (MobileNet, DistilBERT), and minimal hyperparameter tuning save compute.

5. **Streamline data movement.** Avoid redundant steps in ETL pipelines and over-fetching of datasets.

6. Explore techniques like **federated model routing** to direct queries to models of appropriate size, **distillation** to create lighter versions of large models, **speculative decoding** to combine large and small models, **quantization** to reduce model precision and

energy use, and **pruning** to remove redundant parameters and shrink model size.

FOR LEADS / EVERYBODY

1. **Invest in skills and community.** Encourage team learning (e.g. Linux Foundation's LFC131), and support contributions to open source climate tech (e.g. ClimateTriage).
2. **Start building in sustainability early.** Design lightweight, modular systems that minimise compute, storage, and data movement.
3. **Schedule data clean-ups,** and adopt good practices around retention. Implement regular audits of cloud log data to reduce unnecessary storage and emissions.
4. **Explore grid-aware computing.** Use cloud regions powered by renewables. Schedule big compute jobs. Aim to use energy generated by renewables that would otherwise have been curtailed. Start with the Carbon-Aware SDK (GSF).
5. **Explore GreenOps.** Make carbon emissions an explicit design constraint alongside performance and cost. Treat carbon a bit like latency. Make environmental costs visible: bring carbon emissions and other sustainability metrics into build reports and deployment dashboards.
6. **Understand a bit about direct, indirect, and systemic impacts.** Also known as first-order, second-order, third-order impacts. It gets complicated fast, but don't feel you have to know *everything* - just enough not to be fooled by methodologically weak greenwashing.
7. **Understand a bit about how energy is generated and accounted for.** Get immersed in the controversies of greenwashing and carbon accounting. Resources: GHG Protocol, Carbon Market Watch, 24/7 Carbon Free Energy Compact.

8. **Keep an eye on emerging standards.** Resources: ISO, ITU, IEEE, GSF.
9. **Sustainability is about more than carbon.** Resources: Planetary Boundaries model.

OLD VERSION

1. **Get involved in digital sustainability communities:** Explore open source sustainability projects via ClimateTriage.
2. **Invest in training:** Invest in developer training such as LFC131: Green Software for Practitioners (Linux Foundation).
3. **Measure the energy footprint of code and services:** Use tools like Intel RAPL (Running Average Power Limit) and OpenTelemetry (framework for instrumenting, generating, collecting, and exporting telemetry data such as traces, metrics, and logs).
4. **Think about sustainability early:** Build lightweight, modular architectures that reduce compute needs, network calls, and idle runtime.
5. **Measure using CodeCarbon:** A lightweight Python package that estimates CO_2 emissions from compute. Integrate it into ML pipelines and support its development.
6. **Monitor real-time grid intensity:** Use the Electricity Maps API to optimise workloads based on carbon intensity by region and time.
7. **Minify front-end code:** Reduce file sizes in HTML, CSS, and JS by removing whitespace and comments. Use text editor plug-ins or automated build steps.
8. **Avoid unnecessary model training / retraining:** Reuse and fine-tune existing models instead of training from scratch.
9. **Write efficient code:** Prefer algorithms and data structures optimised for time and space. Avoid unnecessary loops, large

batch operations, and repeated I/O. Be very cautious of AI-generated code.

10. **Choose sustainable languages:** Consider energy-efficient languages like Rust where possible.
11. **Explore green infrastructure as code:** Avoid over-provisioning, auto-shutdown idle infrastructure, and favour reusable modules in Terraform or similar tools.
12. **Build sustainability into your pipelines:** Green CI/CD tools can reduce build frequency, eliminate wasteful steps, and schedule jobs when carbon intensity is low.
13. **Explore carbon-aware computing grid-aware computing:** Choose data centres powered by renewables. Immerse yourself in greenwashing and carbon accounting controversies. Aim to use compute powered by renewable energy that otherwise would have been curtailed.