ANDY BLYTH
TECHNICAL DOGSBODY

The Carbon Cost of Intelligence

Architecting AI in Enterprise Web Applications

Andy Blyth PgDip BSc(Hons) FdSc, Technical Architect, MSQ DX

technicaldogsbody.com

15 April 2026

The Carbon Cost of Intelligence: Architecting AI in Enterprise Web Applications

Author: Andy Blyth PgDip BSc(Hons) FdSc, Technical Architect, MSQ DX
Site: technicaldogsbody.com
Published: 15 April 2026
Status: Published
Version: 1.0

Table of Contents

0. Executive Summary
1. Section 0: Why This Is an Architect's Problem
2. Section 1: AI Energy Consumption: Training vs Inference
3. Section 2: Software Carbon Intensity for AI
4. Section 3: Strategies for Inference Efficiency
5. Section 4: Carbon-Aware AI Scheduling
6. Section 5: .NET Implementation Patterns
7. Section 6: Case Study: Contrasting Pair
8. Section 7: Governance and ESG Reporting

Executive Summary

AI inference is now the dominant source of AI-related carbon emissions in enterprise applications. Training costs are high and recurring, new model versions, fine-tuning runs, and reinforcement learning cycles all add to the total. Inference costs are lower per call but accumulate on every request, every day, across the full lifetime of a product, and at a scale that training costs do not reach.

Over 80% of AI electricity consumption comes from the use phase.[1] The most energy-intensive models exceed 29 Wh per long prompt, over 65 times the most efficient systems.[2] The controls available to architects, including model selection, output length, response caching, and workload scheduling, can reduce carbon cost by up to 90% without changing the model or the provider.[3]

This paper provides a practical framework for .NET architects building AI-integrated enterprise applications. Sections cover the evidence base for inference-led emissions, the Software Carbon Intensity (SCI) methodology for measurement, four engineering strategies for reducing inference energy, carbon-aware scheduling for batch workloads, and concrete .NET 10 implementation patterns using MicroMediator, FusionCache, and the Carbon Aware SDK.

The paper also covers governance: how carbon measurement supports ESG reporting obligations and how architects can influence policy decisions before those obligations tighten further.

The paper covers seven sections:

1. Where AI energy actually goes, and why the training narrative misleads engineers into ignoring the inference problem.
2. How to measure carbon cost per API call using the Green Software Foundation's Software Carbon Intensity specification and ISO/IEC 21031:2024.
3. Proven strategies for reducing inference energy, from response length management to model selection and response caching.
4. Carbon-aware scheduling: running batch AI workloads during periods of clean grid energy.
5. Concrete .NET 10 implementation patterns, including middleware, MicroMediator

[1] Arbor.eco. "AI's Environmental Impact: Calculated and Explained." January 16, 2026. https://www.arbor.eco/blog/ai-environmental-impact

[2] Jegham, N. et al. "How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference." arXiv:2505.09598, May 2025 (revised November 2025). https://arxiv.org/abs/2505.09598

[3] UCL News. "Practical changes could reduce AI energy demand by up to 90%." July 8, 2025. https://www.ucl.ac.uk/news/2025/jul/practical-changes-could-reduce-ai-energy-demand-90

integration, and the Carbon Aware SDK.
6. A contrasting pair case study with reproducible benchmark figures.
7. Governance and ESG reporting obligations for .NET teams.

Section 0: Why This Is an Architect's Problem

Every enterprise web application now ships with AI. A chat widget here. A recommendation engine there. A content generation endpoint buried in the CMS.

Each one makes an API call. Each call consumes energy. At scale, those calls compound into a carbon liability your organisation has never measured and your procurement process has never accounted for.

The common understanding of AI's environmental impact focuses on training. The GPT-4 training run cost an estimated $100 million and consumed 50 gigawatt-hours of energy, enough to power San Francisco for three days (or Greater Manchester for around a day and a half).[4] Those numbers are significant. A single training run is a bounded event, even if new versions, fine-tuning cycles, and reinforcement learning runs add to the cumulative total over time. Inference, not training, now represents an increasing majority of AI's energy demands and will continue to do so for the foreseeable future.[5] Training runs happen periodically. Inference happens on every request, every day, for the lifetime of the product.

This is not a data science problem. The data scientists built the model. Architects and developers decide how, when, and how often the model gets called. Those decisions have a direct carbon cost.

Over 80% of AI electricity consumption comes from the use phase, inference rather than

[4]Derived from UK electricity consumption of 280 TWh per year for 67 million people (Department for Energy Security and Net Zero, Energy Consumption in the UK 2025. https://www.gov.uk/government/statistics/energy-consumption-in-the-uk-2025) and Greater Manchester population of approximately 2.9 million (ONS, 2021 Census). This gives roughly 12 TWh per year, or 32 GWh per day for Greater Manchester. 50 GWh therefore covers approximately 1.5 days.

[5]O'Donnell, J. and Crownhart, C. "We did the math on AI's energy footprint. Here's the story you haven't heard." MIT Technology Review, May 20, 2025. https://www.technologyreview.com/2025/05/20/1116327/ai-energy-usage-climate-footprint-big-tech/

training.[6] The gap between the most and least efficient approaches is large: the most energy-intensive models exceed 29 Wh per long prompt, over 65 times the most efficient systems.[7] A single architectural decision, which model to call, whether to cache the response, how long the output should be, can shift your application significantly across this range.

An April 2025 report from the International Energy Agency predicts global electricity demand from data centres will more than double by 2030, to around 945 terawatt-hours.[8] Sixty percent of this growth is projected to be met by fossil fuels. The trajectory is not theoretical. The data centre build-out is already underway.

This paper gives .NET architects a practical framework for measuring and reducing the carbon cost of AI at the point where they have the most control: the code.

Section 1: AI Energy Consumption: Training vs Inference

The training narrative is misleading

When AI energy consumption enters the news, the story centres on training. GPT-3 training emitted an estimated 500 metric tons of CO2, equivalent to driving from New York to San Francisco 438 times (or London to Edinburgh around 5,800 times).[9] GPT-4 training consumed 50 gigawatt-hours. These figures are real and they matter, but they create a distorted picture for engineers.

Training is a recurring cost, each new model version, fine-tuning run, or reinforcement learning cycle adds to the total. Once a specific model version is deployed, energy flows

[6] Arbor.eco. "AI's Environmental Impact: Calculated and Explained." January 16, 2026. https://www.arbor.eco/blog/ai-environmental-impact

[7] Jegham, N. et al. "How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference." arXiv:2505.09598, May 2025 (revised November 2025). https://arxiv.org/abs/2505.09598

[8] MIT News. "Responding to the climate impact of generative AI." September 30, 2025. https://news.mit.edu/2025/responding-to-generative-ai-climate-impact-0930

[9] Climate Impact Partners. "The Carbon Footprint of AI." October 28, 2025. https://www.climateimpact.com/news-insights/insights/carbon-footprint-of-ai/

in a completely different pattern. The cumulative cost of all training activity is real, but inference compounds continuously across every deployment. The headline numbers belong to bounded events. The operational cost is the one your application generates without pause.

Inference dominates at scale

Between 80% and 90% of AI computing power in production environments now supports inference rather than training.[10] This figure reflects what happens when models move from research labs into products used by millions of people daily.

Google's own measurements illustrate why inference compounds so quickly. A median Gemini text prompt uses 0.24 watt-hours of energy and emits 0.03 grams of CO2 equivalent.[11] Those numbers look small. Multiply them by 700 million queries per day, a figure within reach of any major AI product, and the annual electricity consumption becomes comparable to 35,000 US homes (roughly 136,000 UK homes).[12][13]

Your enterprise application is not running at this scale. The principle holds at every order of magnitude though. A .NET API calling an LLM on every page load, without caching, without model selection, without output length constraints, accumulates carbon cost invisibly. There is no line in your Azure bill labelled "carbon."

[10] AIMultiple. "AI Energy Consumption Statistics." 2026. https://aimultiple.com/ai-energy-consumption

[11] Elsworth et al. "Measuring the environmental impact of AI inference." Google Cloud Blog, 2025. https://cloud.google.com/blog/products/infrastructure/measuring-the-environmental-impact-of-ai-inference/

[12] Jegham, N. et al. "How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference." arXiv:2505.09598, May 2025 (revised November 2025). https://arxiv.org/abs/2505.09598

[13] Ofgem. "Average gas and electricity use explained." Typical Domestic Consumption Values. https://www.ofgem.gov.uk/information-consumers/energy-advice-households/average-gas-and-electricity-use-explained UK average of 2,700 kWh per household per year. US average of 10,500 kWh per household per year per US Energy Information Administration (EIA), "Use of energy explained: Energy use in homes." https://www.eia.gov/energyexplained/use-of-energy/homes.php

The efficiency gap is significant

Not all inference costs the same. A benchmarking study across 30 state-of-the-art models found the most energy-intensive models exceed 29 Wh per long prompt, over 65 times the most efficient systems.[14]

The variation comes from several factors.

Model size. Larger models consume more energy per token. A 70B parameter model does not simply cost twice as much as a 7B model per equivalent task.

Task type. Reasoning models generating internal chains of thought before responding can use 50 to 100 times the energy of standard inference.[15] A task not requiring reasoning should never use a reasoning model.

Output length. Research from UCL found halving the length of an AI response reduces energy consumption by 54%.[16] Instructions to the model about response length are not a stylistic preference. They are an energy control.

Provider infrastructure. Google reports a 33x reduction in energy per prompt over a 12-month period through hardware and software optimisation.[17] The carbon intensity of the same model call varies significantly by provider and region. A call routed to a US east coast data centre at peak demand carries a different carbon cost to the same call routed to a Nordic data centre running on hydro.

Where this leaves architects

The data establishes three things relevant to how you write code:

---

[14] Jegham, N. et al. "How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference." arXiv:2505.09598, May 2025 (revised November 2025). https://arxiv.org/abs/2505.09598

[15] Arbor.eco. "AI's Environmental Impact: Calculated and Explained." January 16, 2026. https://www.arbor.eco/blog/ai-environmental-impact

[16] UCL News. "Practical changes could reduce AI energy demand by up to 90%." July 8, 2025. https://www.ucl.ac.uk/news/2025/jul/practical-changes-could-reduce-ai-energy-demand-90

[17] Elsworth et al. "Measuring the environmental impact of AI inference." Google Cloud Blog, 2025. https://cloud.google.com/blog/products/infrastructure/measuring-the-environmental-impact-of-ai-inference/

1. The carbon cost of your AI integration is dominated by inference, not the initial model selection or vendor contract.
2. The range between efficient and inefficient inference is large enough to matter at enterprise scale.
3. The controls reducing energy consumption, output length, model choice, caching, scheduling, all live in application code, not in the model itself.

Section 3 addresses each of those controls directly. First, you need a way to measure what you are producing.

Section 2: Software Carbon Intensity for AI

Why measurement comes before optimisation

You cannot reduce what you do not measure. Before applying any of the strategies in Section 3 (Strategies for Inference Efficiency), you need a consistent method for quantifying the carbon cost of an AI call. Without a baseline, there is no way to confirm a change made a difference, or to compare two architectural approaches.

The Green Software Foundation's Software Carbon Intensity (SCI) specification, formalised as ISO/IEC 21031:2024, provides this method.

The SCI formula

SCI expresses carbon cost as a rate rather than a total:

```
SCI = (E × I + M) per R
```

Where:

- `E` = energy consumed by the software (kWh)
- `I` = carbon intensity of the electricity grid (gCO2e/kWh)
- `M` = embodied carbon of the hardware
- `R` = functional unit, the unit you are measuring per

The functional unit is the critical design decision. For a web application, `R` is typically a single user request. For an AI integration, the unit should be more specific: per API call, per token generated, or per inference task completed.

ISO/IEC 21031:2024 formalises this methodology as an international standard, giving regulatory weight for organisations operating under ESG reporting requirements.[18]

Applying SCI to AI calls

The Green Software Foundation extended the SCI specification to AI systems in December 2025, covering classical ML, generative AI, and agentic AI pipelines.[19]

For an LLM API call, the components map as follows.

Energy (E). You will not measure GPU power draw directly. Use published per-token or per-prompt energy figures from your provider, or from independent benchmarking studies such as Jegham et al. (2025).[20] Google publishes detailed methodology for Gemini.[21] For other providers, use the arXiv benchmarking framework as an approximation.

Grid intensity (I). This varies by region and time of day. The Carbon Aware SDK provides real-time grid intensity data for most major regions. In the UK, the National Grid ESO publishes carbon intensity forecasts via a public API.[22] Use the region where your inference runs, not where your application server sits.

Embodied carbon (M). This covers the hardware running the model. Under GHG Protocol Scope 3 Category 1, a portion of the data centre's embodied hardware emissions belongs to your workload based on usage.[23] Most teams exclude this initially and add

[18] ISO/IEC 21031:2024. "Information technology: Software Carbon Intensity (SCI) specification." March 22, 2024. https://www.iso.org/standard/86612.html

[19] Green Software Foundation. "SCI for AI Specification." December 17, 2025. https://github.com/Green-Software-Foundation/sci-ai/blob/main/SPEC.md

[20] Jegham, N. et al. "How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference." arXiv:2505.09598, May 2025 (revised November 2025). https://arxiv.org/abs/2505.09598

[21] Elsworth et al. "Measuring the environmental impact of AI inference." Google Cloud Blog, 2025. https://cloud.google.com/blog/products/infrastructure/measuring-the-environmental-impact-of-ai-inference/

[22] National Grid ESO. Carbon Intensity API. https://api.carbonintensity.org.uk/

[23] Arbor.eco. "AI's Environmental Impact: Calculated and Explained." January 16, 2026. https://www.arbor.eco/blog/ai-environmental-impact

when reporting requirements demand.

Functional unit (R). Define this at the level mapping to user value. For a content generation feature, use per-document-generated. For a search enhancement, use per-search-query. Consistency matters more than precision: choose a unit and stick to the choice.

A worked example

Consider a .NET API endpoint generating a product description using an LLM:

```
E  = 0.24 Wh per call (median Gemini text prompt, Google, May 2025)
I  = 233 gCO2e/kWh (UK grid average, National Grid ESO, 2025)
M  = excluded (initial baseline)
R  = per product description generated

SCI = (0.00024 kWh × 233 gCO2e/kWh) per call
    = 0.056 gCO2e per call
```

At 10,000 product descriptions per day, this is 560 grams of CO2e daily, or approximately 204 kg per year from a single endpoint.

This baseline gives you something concrete to work against. If response caching eliminates 60% of calls, the annual figure drops to 82 kg. If model selection replaces a 70B model with a 7B model for this specific task, the energy figure changes accordingly. Every architectural decision becomes auditable.

Tracking SCI in .NET

The Carbon Aware SDK 1.4, upgraded to .NET 8 in 2024, includes Prometheus exporters for carbon metrics.[24] Instrumenting your application and feeding carbon cost into existing observability pipelines alongside latency and error rate is straightforward.

Section 5 covers the integration code. Carbon cost should live in your dashboards beside the metrics you already act on.

---

[24] Green Software Foundation. "Carbon Aware SDK 1.4, behind the scenes." May 16, 2024. https://carbon-aware-sdk.greensoftware.foundation/blog/dotnet-8-upgrade

Section 3: Strategies for Inference Efficiency

Measurement gives you a baseline. These strategies give you the levers to shift the number.

UCL research found practical, non-specialist changes can reduce AI energy demand by up to 90%.[25] The four strategies below are all within the control of the application layer. None require changes to the model, the provider, or the infrastructure.

Strategy 1: Babbling suppression: control output length

The single highest-impact change you can make is constraining how much the model outputs.

LLMs have a documented tendency to pad responses with preamble, restatement, qualification, and conclusion that adds no value to the output. Researchers call this babbling suppression, the practice of actively constraining this behaviour rather than accepting it as a default. It is not a stylistic concern. It is an energy control.

Halving the length of an AI response reduces energy consumption by 54%.[26] Every token in the overhead costs energy.

Apply length constraints at three levels.

System prompt. Instruct the model to be concise. "Respond in under 100 words. Do not repeat the question. Do not add a conclusion." This is the cheapest intervention: one line in a system prompt.

Max tokens parameter. Set `max_tokens` explicitly on every API call. Do not rely on the model's default behaviour. For structured output tasks, calculate the theoretical maximum and set a ceiling close to the result.

Output schema. For tasks returning structured data, use JSON mode or function calling to constrain the output format. A model forced to return a JSON object cannot pad its

[25] UCL News. "Practical changes could reduce AI energy demand by up to 90%." July 8, 2025. https://www.ucl.ac.uk/news/2025/jul/practical-changes-could-reduce-ai-energy-demand-90

[26] UCL News. "Practical changes could reduce AI energy demand by up to 90%." July 8, 2025. https://www.ucl.ac.uk/news/2025/jul/practical-changes-could-reduce-ai-energy-demand-90

response with explanatory prose.

These are not stylistic preferences. They are energy controls compounding across every call your application makes.

Strategy 2: Right-size the model for the task

Reasoning models generating chains of thought before responding use 50 to 100 times the energy of standard inference models.[27] Deploying a reasoning model for a task not requiring reasoning is the equivalent of running a diesel generator to charge a phone.

Classify your AI tasks by what they actually require:

TASK TYPE	APPROPRIATE MODEL TIER	EXAMPLE
Classification or extraction	Small (7B or equivalent)	Sentiment detection, entity extraction
Summarisation	Medium	Document condensation, meeting notes
Generation with constraints	Medium	Product descriptions, templated emails
Complex reasoning	Large or reasoning model	Multi-step analysis, code review

In most enterprise applications, the majority of AI calls fall into the first two rows. Routing them to smaller models reduces energy cost significantly without measurable quality loss for those specific tasks.

Implement a task router in your AI middleware layer. The router selects the appropriate model endpoint based on task type, not a single application-wide default. Section 5 shows this pattern in .NET.

[27] Arbor.eco. "AI's Environmental Impact: Calculated and Explained." January 16, 2026. https://www.arbor.eco/blog/ai-environmental-impact

Strategy 3: Cache AI responses

LLM APIs are stateless and expensive. Many enterprise applications call them for inputs repeating across users. Product descriptions, FAQ answers, content summaries, search result augmentations: these often involve the same or semantically similar prompts.

Caching an AI response costs the carbon of one call. Serving the cached response to one hundred users costs nothing additional.

Implement caching at two levels.

Exact match caching. Hash the prompt and cache the response. Suitable for deterministic inputs: report generation, content seeding, structured data extraction from fixed templates. FusionCache on Redis makes this straightforward in .NET.

Semantic caching. Embed the prompt and cache based on cosine similarity to previous prompts. Suitable for user-generated queries where the same question arrives in many phrasings. Requires a vector store: Azure AI Search supports this.

Set TTLs reflecting the volatility of the underlying data. A cached product description for an item changing monthly can have a TTL of days. A cached answer to a policy question should be invalidated when the policy changes, not on a fixed schedule.

The carbon saving from caching is not incremental. On high-traffic endpoints, caching can eliminate the majority of LLM API calls entirely.

Strategy 4: Suppress unnecessary calls

Not every user interaction needs an LLM. Applications routing all input through an AI layer, regardless of whether AI adds value, generate carbon cost with no corresponding benefit.

Apply a classification gate before the LLM call:

- Does this input require inference, or can a deterministic function handle the query?
- Is the user requesting a capability the application already has the answer to?
- Does the query fall below a confidence threshold justifying the cost of an LLM call?

This is where Green UX design intersects with carbon efficiency. UI patterns encouraging shorter, more specific queries reduce the token count of the resulting call. Auto-

complete, structured input forms, and faceted search all reduce the surface area of AI calls without degrading the user experience.[28]

Section 5 shows a .NET middleware pattern implementing a classification gate as a pipeline behaviour.

Section 4: Carbon-Aware AI Scheduling

The grid is not constant

Grid carbon intensity, the grams of CO2 emitted per kilowatt-hour of electricity consumed, varies continuously. In the UK, the figure swings between approximately 50 gCO2e/kWh during periods of high wind generation and over 300 gCO2e/kWh during peak demand on calm, overcast days.[29]

Real-time AI workloads cannot be moved. When a user asks a question, the inference must happen immediately. A significant proportion of enterprise AI workloads are not real-time though:

- Nightly content re-generation
- Batch document summarisation
- Scheduled report production
- Embedding generation for search indexes
- Training data preparation

These workloads have flexibility in when they execute. This flexibility has a carbon value.

Time-shifting workloads

Carbon-aware scheduling moves deferrable AI workloads to periods when the grid runs on cleaner energy. The approach is called time-shifting: instead of running a batch job at 2am because the servers are quieter, you run the job when carbon intensity is lowest.

---

[28] Agnikii Digital. "Sustainable Web Design Best Practices: 2026 Guide." March 2, 2026. https://agnikii.co.uk/insights/sustainable-web-design-best-practices/

[29] National Grid ESO. Carbon Intensity API. https://api.carbonintensity.org.uk/

This may be midday on a windy day, or 3am on a calm one.

The bluehands Carbon-Aware-Computing library provides a .NET implementation of this pattern.[30] The library queries real-time carbon intensity data via the Carbon Aware SDK and returns the optimal execution window within a specified time range.

As an illustrative benchmark, Intel's Gaudi 3 hardware demonstrated a 40% improvement in inference power-efficiency for Llama models compared with the previous generation (vs NVIDIA H100) at launch in 2024.[31] Hardware efficiency and carbon-aware scheduling compound: the same workload, on more efficient hardware, during a clean energy window, carries a fraction of the carbon cost of the unoptimised baseline. The specific hardware will change; the principle that hardware choice multiplies scheduling gains will not.

What carbon-aware scheduling looks like in practice

A practical implementation has three components.

Carbon intensity data. The Carbon Aware SDK aggregates data from multiple grid operators. In .NET, the SDK is a NuGet dependency. Query for a location and a time window, and the SDK returns the lowest-carbon execution slot.

A deferral mechanism. AI batch jobs need to be expressible as deferred work items. Azure Durable Functions, Azure Service Bus, or a background job library such as Hangfire all provide this. The key constraint is the job must tolerate a variable start time within a defined window.

A window constraint. Specify the latest acceptable completion time. The scheduler finds the optimal slot within the window. A report needing to be ready by 9am can be scheduled to run at any point in the preceding 12 hours. The scheduler chooses the cleanest window.

Section 5 shows a concrete .NET implementation combining the Carbon Aware SDK

[30] bluehands. "Carbon-Aware-Computing: Execute tasks on a point in time with minimal grid carbon intensity." GitHub, 2025. https://github.com/bluehands/Carbon-Aware-Computing

[31] HPCwire. "Intel Introduces Gaudi 3, Elevating HPC and AI with Increased Speed and Scalability." April 9, 2024. https://www.hpcwire.com/off-the-wire/intel-introduces-gaudi-3-elevating-hpc-and-ai-with-increased-speed-and-scalability/

with Azure Durable Functions.

Measuring the impact

Apply SCI measurement from Section 2 (Software Carbon Intensity for AI) to your scheduled workloads. Track carbon cost per batch run. Over time, compare the carbon cost of scheduled runs against what the same workload would have cost at a fixed time. The Carbon Aware SDK 1.4 Prometheus exporters make this visible in Grafana alongside your existing infrastructure metrics.[32]

Section 5: .NET Implementation Patterns

This section provides concrete code for each strategy described above. All examples target .NET 10 with the Microsoft.Extensions.AI abstractions introduced in .NET 9.

The patterns connect directly to the TechnicalDogsbody.MicroMediator library, a high-performance mediator for .NET benchmarking 2 to 45 times faster than MediatR.[33] Pipeline behaviours in MicroMediator make the carbon-aware middleware patterns below composable and testable without coupling them to any specific AI provider.

For guidance on structuring a .NET 10 application to support this pattern, see the Clean Architecture for .NET 10 video reference.[34]

5.1 Carbon-aware middleware pipeline

The entry point for all AI calls in your application should be a pipeline. A single pipeline gives you one place to apply measurement, caching, model routing, and output length enforcement without scattering logic across handlers.

Using MicroMediator pipeline behaviours:

[32] Green Software Foundation. "Carbon Aware SDK 1.4, behind the scenes." May 16, 2024. https://carbon-aware-sdk.greensoftware.foundation/blog/dotnet-8-upgrade

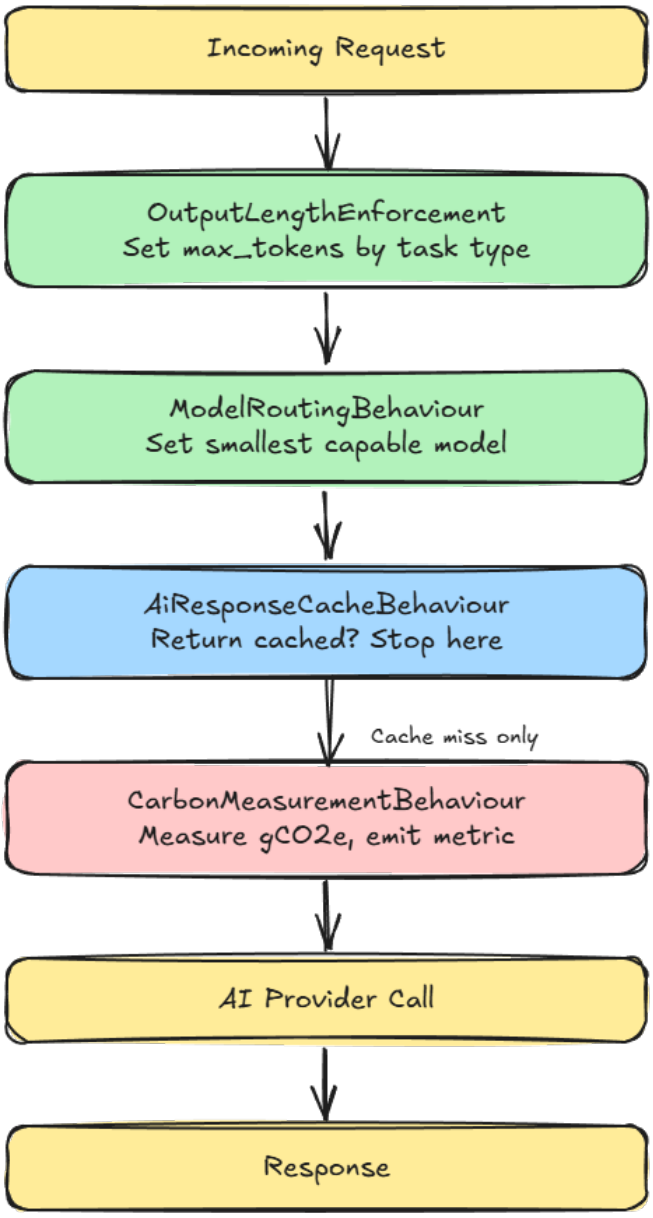
[33] TechnicalDogsbody. MicroMediator. NuGet, 2025. https://www.nuget.org/packages/TechnicalDogsbody.MicroMediator

[34] Digital TechJoint. "Clean Architecture for .NET 10." YouTube, 2025. https://www.youtube.com/watch?v=K9DVAPRWJsA

```
// Register the pipeline
services.AddMicroMediator(cfg =>
{
    cfg.AddPipelineBehaviour<CarbonMeasurementBehaviour>();
    cfg.AddPipelineBehaviour<AiResponseCacheBehaviour>();
    cfg.AddPipelineBehaviour<ModelRoutingBehaviour>();
    cfg.AddPipelineBehaviour<OutputLengthEnforcementBehaviour>();
});
```

Each behaviour is independently testable and can be toggled per environment.

The request flow through the pipeline:



5.2 Measuring carbon per AI call

The `CarbonMeasurementBehaviour` records the estimated carbon cost of each inference call and emits a Prometheus metric:

```
public class CarbonMeasurementBehaviour<TRequest, TResponse>
    : IPipelineBehaviour<TRequest, TResponse>
    where TRequest : IAiRequest
{
    private readonly ICarbonIntensityProvider _carbonIntensity;
    private readonly ITokenCounter _tokenCounter;
    private readonly Counter _carbonCounter;

    public CarbonMeasurementBehaviour(
        ICarbonIntensityProvider carbonIntensity,
        ITokenCounter tokenCounter,
        IMeterFactory meterFactory)
    {
        _carbonIntensity = carbonIntensity;
        _tokenCounter = tokenCounter;

        var meter = meterFactory.Create("AI.Carbon");
        _carbonCounter = meter.CreateCounter<double>(
            "ai_carbon_gco2e",
            unit: "gCO2e",
            description: "Estimated carbon cost of AI inference calls");
    }

    public async Task<TResponse> HandleAsync(
        TRequest request,
        CancellationToken cancellationToken,
        Func<Task<TResponse>> next)
    {
        var response = await next();

        var tokens = _tokenCounter.Count(response);
        var gridIntensity = await _carbonIntensity.GetCurrentIntensityAsync();
```

```
        // 0.0003 kWh per 1000 tokens is a conservative approximation.
        // Replace with provider-specific figures for accuracy.
        var energyKwh = (tokens / 1000.0) * 0.0003;
        var carbonGco2e = energyKwh * gridIntensity.GramsPerKwh;

        _carbonCounter.Add(carbonGco2e, new TagList
        {
            { "model", request.Model },
            { "task_type", request.TaskType },
            { "endpoint", request.EndpointName }
        });

        return response;
    }
}
```

5.3 Response caching with FusionCache

The `AiResponseCacheBehaviour` applies exact-match caching using FusionCache with a Redis backplane:

```
public class AiResponseCacheBehaviour<TRequest, TResponse>
    : IPipelineBehaviour<TRequest, TResponse>
    where TRequest : IAiRequest, ICacheable
{
    private readonly IFusionCache _cache;

    public AiResponseCacheBehaviour(IFusionCache cache)
    {
        _cache = cache;
    }

    public async Task<TResponse> HandleAsync(
        TRequest request,
        CancellationToken cancellationToken,
        Func<Task<TResponse>> next)
    {
        var cacheKey = $"ai:{request.GetCacheKey()}";
```

```
        return await _cache.GetOrSetAsync(
            cacheKey,
            async ct => await next(),
            options => options
                .SetDuration(request.CacheDuration)
                .SetFailSafe(true),
            cancellationToken);
    }
}
```

The `ICacheable` interface requires the request to declare its own cache key and TTL. This keeps the caching logic in the domain model, not scattered across handlers.

5.4 Model routing by task type

The `ModelRoutingBehaviour` selects the appropriate model endpoint based on the declared task type:

```
public class ModelRoutingBehaviour<TRequest, TResponse>
    : IPipelineBehaviour<TRequest, TResponse>
    where TRequest : IAiRequest
{
    private static readonly Dictionary<AiTaskType, string> _modelMap = new()
    {
        [AiTaskType.Classification]   = "gpt-4o-mini",
        [AiTaskType.Extraction]       = "gpt-4o-mini",
        [AiTaskType.Summarisation]    = "gpt-4o",
        [AiTaskType.Generation]       = "gpt-4o",
        [AiTaskType.ComplexReasoning] = "o3"
    };

    public async Task<TResponse> HandleAsync(
        TRequest request,
        CancellationToken cancellationToken,
        Func<Task<TResponse>> next)
    {
        if (string.IsNullOrEmpty(request.Model))
        {
            request.Model = _modelMap.GetValueOrDefault(
```

```
                request.TaskType,
                "gpt-4o-mini"); // Default to smallest capable model
        }

        return await next();
    }
}
```

Classification tasks never reach a reasoning model with this pattern in place. Move the model map to configuration to allow environment-level overrides.

5.5 Carbon-aware scheduling with Azure Durable Functions

For deferrable batch workloads, combine the Carbon Aware SDK with a Durable Functions orchestration:

```
[FunctionName("CarbonAwareBatchOrchestrator")]
public async Task RunOrchestration(
    [OrchestrationTrigger] IDurableOrchestrationContext context,
    ILogger log)
{
    var input = context.GetInput<BatchAiJobInput>();

    var optimalWindow = await context.CallActivityAsync<DateTimeOffset>(
        "FindOptimalCarbonWindow",
        new CarbonWindowRequest
        {
            Location = input.GridRegion,
            WindowStart = context.CurrentUtcDateTime,
            WindowEnd = input.Deadline,
            EstimatedDurationMinutes = input.EstimatedDurationMinutes
        });

    if (optimalWindow > context.CurrentUtcDateTime)
    {
        await context.CreateTimer(optimalWindow, CancellationToken.None);
    }

    await context.CallActivityAsync("ExecuteAiBatchJob", input);
```

```
}

[FunctionName("FindOptimalCarbonWindow")]
public async Task<DateTimeOffset> FindOptimalCarbonWindow(
    [ActivityTrigger] CarbonWindowRequest request,
    ILogger log)
{
    var handler = new CarbonAwareVariableTimeIterationHandler();

    var parameters = new CarbonAwareVariableTimeIterationParameters
    {
        Location = request.Location,
        EarliestExecutionTime = request.WindowStart,
        LatestExecutionTime = request.WindowEnd,
        EstimatedExecutionDuration =
            TimeSpan.FromMinutes(request.EstimatedDurationMinutes)
    };

    var result = await handler.GetBestExecutionTimeAsync(parameters);
    return result.ExecutionTime;
}
```

The `CarbonAwareVariableTimeIterationHandler` comes from the bluehands Carbon-Aware-Computing NuGet package.[35]

5.6 Output length enforcement

Enforce max token limits on all outbound requests as a pipeline behaviour:

```
public class OutputLengthEnforcementBehaviour<TRequest, TResponse>
    : IPipelineBehaviour<TRequest, TResponse>
    where TRequest : IAiRequest
{
    private static readonly Dictionary<AiTaskType, int> _maxTokens = new()
    {
        [AiTaskType.Classification]   = 50,
```

[35] bluehands. "Carbon-Aware-Computing: Execute tasks on a point in time with minimal grid carbon intensity." GitHub, 2025. https://github.com/bluehands/Carbon-Aware-Computing

```
        [AiTaskType.Extraction]       = 200,
        [AiTaskType.Summarisation]    = 300,
        [AiTaskType.Generation]       = 500,
        [AiTaskType.ComplexReasoning] = 1500
    };

    public async Task<TResponse> HandleAsync(
        TRequest request,
        CancellationToken cancellationToken,
        Func<Task<TResponse>> next)
    {
        if (!request.MaxTokens.HasValue)
        {
            request.MaxTokens = _maxTokens.GetValueOrDefault(
                request.TaskType,
                300); // Conservative default
        }

        return await next();
    }
}
```

Combined with the system prompt instruction to be concise, this creates a hard ceiling on output tokens the model cannot exceed.

5.7 Putting it together

The full pipeline gives you a measurable, carbon-efficient AI integration layer. Every AI call passes through all four behaviours in sequence. Output length is bounded before the model is selected. The model is selected before the cache is checked. Carbon is measured only on cache misses, where a provider call actually occurs. Cached responses generate zero additional carbon cost.

Pair this with carbon-aware scheduling for batch workloads (Section 5.5) and you have measurable carbon reduction across both real-time and deferred AI usage.

Section 6: Case Study - Contrasting Pair

The strategies in Sections 3 and 5 are grounded in published research. This section shows what they produce in practice, using a reproducible benchmark built against the same MicroMediator pipeline described throughout the paper.

The scenario

A CMS platform serves 1,000 page-render requests. Each request requires an AI-generated content summary. The workload reflects a realistic distribution: 60% of requests are for pages already seen, 30% are new pages requiring a short summary, and 10% are new pages requiring a longer structured analysis.

Two architectural approaches run against the same workload.

Option A uses a single large model for every request with no caching and no output length constraint. Every request reaches the provider. Every response returns 500 tokens.

Option B uses the MicroMediator pipeline from Section 5: output length enforcement, model routing (small model for summarisation, large for complex reasoning), FusionCache exact-match caching with a one-hour TTL, and carbon measurement via `CarbonMeasurementBehaviour`.

Results

METRIC	OPTION A	OPTION B	SAVING
Total requests	1,000	1,000	
Provider calls made	1,000	400	60%
Cache hit rate	0%	60%	
Tokens consumed	500,000	70,000	86%
Estimated energy	150.0 Wh	21.0 Wh	86%
Estimated carbon (gCO2e)	34.95 g	4.89 g	86%
Simulated duration	~50,000 ms	~20,000 ms	60%

Carbon figures use the SCI formula from Section 2: 0.0003 kWh per 1,000 tokens at 233 gCO2e/kWh (UK grid average, National Grid ESO, 2025).

What the per-request breakdown shows

The aggregate saving of 86% comes from three behaviours working together. The per-request sample makes each one visible.

#	PAGEID	TASK TYPE	MODEL	HIT	TOKENS	CARBON (GCO2E)
1	PAGE-067	Summary	-	Y	0	0.0000
2	PAGE-015	Summary	-	Y	0	0.0000
3	PAGE-013	Summary	-	Y	0	0.0000
4	PAGE-053	Summary	-	Y	0	0.0000
601	PAGE-101	Summary	Small	N	100	0.0070
602	PAGE-102	Summary	Small	N	100	0.0070
603	PAGE-103	Summary	Small	N	100	0.0070
901	PAGE-401	Complex	Large	N	400	0.0280
902	PAGE-402	Complex	Large	N	400	0.0280
903	PAGE-403	Complex	Large	N	400	0.0280

Rows 1–4 are cache hits. The pipeline short-circuits before reaching the model router. Zero tokens, zero carbon.

Rows 601–603 are cache misses on summarisation tasks. The model router selects the small model tier. 100 tokens, 0.0070 gCO2e each.

Rows 901–903 are cache misses on complex reasoning tasks. The model router selects the large model tier. 400 tokens, 0.0280 gCO2e each, four times the cost of a summarisation miss, which is exactly why task routing matters.

In Option A, all 1,000 requests produce 500 tokens at large model cost regardless of task type. The cache never fires. No routing decision is made.

The benchmark is reproducible

The workload uses a seeded random generator (`seed = 42`) so every run produces identical output. The benchmark runs without API keys, network calls, or cloud dependencies. All carbon arithmetic is in a single `CarbonConstants` class.

The source code is published at `https://github.com/technicaldogsbody/ai-carbon-benchmark`. Clone and run `dotnet run` to reproduce the figures above.[36]

Section 7: Governance and ESG Reporting

Why architects own more of this than they think

Carbon measurement for AI is not a sustainability team problem. The data needed to populate an ESG report, grams of CO2 equivalent per inference call, carbon cost per product feature, total emissions from AI workloads in a financial year, all comes from instrumentation in the application layer.

If your application does not measure carbon, your organisation cannot report on emissions. If carbon reporting is absent, any ESG commitment covering Scope 3 emissions from software will be based on estimates rather than evidence.

Architects who build measurement in from the start put their organisations in a stronger position when reporting requirements tighten. Architects who do not will be asked to retrofit measurement later under pressure.

The regulatory direction

ISO/IEC 21031:2024 exists and is already cited in procurement and due diligence processes.[37] The EU AI Act entered into force in August 2024 with a phased rollout of obligations. Requirements for general-purpose AI providers applied from August 2025; requirements for high-risk AI systems apply from August 2026. While the Act does not mandate carbon reporting directly, the legislation establishes a precedent for techni-

[36] Blyth, A. ai-carbon-benchmark. GitHub, 2026. `https://github.com/technicaldogsbody/ai-carbon-benchmark`

[37] ISO/IEC 21031:2024. "Information technology: Software Carbon Intensity (SCI) specification." March 22, 2024. `https://www.iso.org/standard/86612.html`

cal accountability over AI system behaviour, and carbon cost is increasingly part of the conversation.

The Green Software Foundation's SCI specification, now extended to cover AI systems, provides the methodology for calculating emissions in a way auditors and regulators can follow.[38] Adopting this methodology now is low-cost. Adopting the methodology under regulatory pressure later is not.

Practical governance steps for .NET teams

Add carbon as a first-class metric. The Prometheus exporter pattern in Section 5.2 emits `ai_carbon_gco2e` tagged by model, task type, and endpoint. Feed this into your existing Grafana dashboards. Carbon cost then lives alongside latency, error rate, and cost per call.

Set carbon budgets per feature. Express the expected carbon cost of a feature as a number. "This content generation endpoint should not exceed 5 kg CO2e per month." Track against the budget. Treat overruns the same way you treat performance regressions.

Include AI energy in architectural decision records. When choosing between two model tiers or two providers, document the carbon cost difference alongside the cost and latency difference. This creates an audit trail.

Review scheduled workloads quarterly. Carbon-aware scheduling requires a defined execution window. As business requirements change, windows may have tightened or the opportunity for time-shifting may have grown. A quarterly review keeps the scheduling configuration aligned with actual flexibility.

Engage procurement. When renewing cloud or AI provider contracts, ask for emissions data per API call. Google publishes this for Gemini. Other providers are under increasing pressure to follow. The question itself signals to the provider the data matters to you.

---

[38] Green Software Foundation. "SCI for AI Specification." December 17, 2025. https://github.com/Green-Software-Foundation/sci-ai/blob/main/SPEC.md

Conclusion

AI inference now dominates AI-related carbon emissions in enterprise applications. The controls reducing this cost, model selection, output length, caching, and scheduling, are all application-layer decisions.

The SCI framework provides a consistent method for measuring carbon cost per functional unit, making the figure auditable and comparable across architectural changes. The .NET ecosystem has mature tooling for implementing all of these patterns today: FusionCache, the Carbon Aware SDK, Azure Durable Functions, and MicroMediator.

The direction of travel is clear. The IEA projects data centre electricity demand doubling by 2030. ESG reporting requirements are tightening. Carbon cost will become a line item engineering decisions need to account for.

The architects building this accounting into their applications now will not need to retrofit the work later.
