# CYDROFLUX AI

## Sustainability-Aware Request Routing Engine for Global AI Data Centres with Green Intelligence Scoring and LLM-Assisted Workload Interpretation

**Nandhini S[1], Malini A[2], Monisha M[3], Nethra P4, Netra V[5]**

1. Assistant Professor, Dept. of Computer Science and Engineering,
Sri Shakthi Institute of Engineering and Technology, India. Email: nandhiniscse@siet.ac.in
2. Student (714025104131), Department of Computer Science and Engineering,
Sri Shakthi Institute of Engineering and Technology, India. Email: maliniarumugam01@gmail.com
3. Student (714025104148), Department of Computer Science and Engineering,
Sri Shakthi Institute of Engineering and Technology, India. Email: monishamuthuvel11@gmail.com
4. Student (714025104163), Department of Computer Science and Engineering,
Sri Shakthi Institute of Engineering and Technology, India. Email: nethrap603@gmail.com
5. Student (714025104166), Department of Computer Science and Engineering,
Sri Shakthi Institute of Engineering and Technology, India. Email: netrav2672007@gmail.com

**Abstract—CydroFlux AI is a web platform that decides where an AI workload should run, not only whether it can run. It routes inference, batch-training and real-time requests across six AWS-hosted data-centre regions by combining a transparent Green Intelligence Score (GIS) with strict operational guardrails. The GIS is a weighted sum of renewable-energy share (40%), grid carbon intensity (30%), water-usage effectiveness (20%) and server-load headroom (10%). Facilities must first satisfy latency (≤150 ms), load (≤90%), data-residency and telemetry-validity rules; eligible facilities are then ordered by a deterministic tiebreaker hierarchy, and every decision is stored with an audit trail and a human-readable reason. A Gemma-family language model interprets natural-language workload descriptions into structured requirements but cannot invent telemetry or choose the facility; a rule-based parser replaces it when unavailable. The backend uses FastAPI and MongoDB Atlas with an in-memory fallback, and the React interface supports English, Tamil and Hindi. In verification, 118 of 119 automated tests passed; the single failure needs live internet access. An audit log of 1,777 development decisions shows a median routing computation of 0.78 ms, and the GIS ranking was stable across five weight profiles. Carbon, renewable, water and load values are currently modelled from regional baselines, not live feeds, so outputs must not be read as measured emissions or validated savings.**

**Keywords—carbon-aware routing, green AI, data-centre scheduling, sustainability scoring, explainable decisions, large language model**

## I. INTRODUCTION

The computing demand of AI is growing quickly, and so is its footprint. Training and serving large models consumes electricity whose carbon cost depends on the local grid, and it consumes water through evaporative cooling, whose cost depends on climate and cooling design [1]–[3]. Once a model is deployed at scale, inference (every conversational turn, embedding lookup or image request) becomes a continuing draw on regional transmission grids. The carbon intensity of those grids varies enormously: a facility on a hydro-, wind- or nuclear-rich grid may sit near or below 150 $gCO_2$/kWh, while one on a coal- or gas-heavy grid can exceed 600 $gCO_2$/kWh. Two facilities with identical hardware can therefore differ several-fold in the emissions of a single inference call. Yet in everyday practice, requests are sent to the nearest or the cheapest region, and sustainability does not enter the decision.

Four difficulties recur. First, proximity bias: DNS and anycast routing map a request to the topologically nearest site, which suits cached media but is poorly matched to compute-heavy generation, where server energy dwarfs packet-transit energy. Second, sustainability data are scattered: grid carbon intensity, renewable share, water efficiency, server utilisation and network latency come from different sources and use different units, so they cannot be compared directly. Third, a purely green policy is unsafe: sending every request to the cleanest region would overload it and add unacceptable delay for real-time users, so operational constraints must be enforced before sustainability is considered. Fourth, routing decisions are rarely explainable. An operator who is told only that a request went to Oregon cannot check,

challenge or tune the decision, and commercial load-balancing logic is closed and uninspectable.

CydroFlux AI addresses these difficulties in one application. Its objectives are: (1) to collect and normalise telemetry (weather, carbon intensity, renewable share, water usage, server load, network latency) for a fleet of data centres; (2) to compute a single, transparent sustainability score for each facility from stated weights; (3) to dispatch each workload to the greenest facility that satisfies latency, capacity and residency constraints; (4) to interpret free-text workload descriptions with a language model while keeping the model away from operational data and from the final decision; and (5) to record every decision with its reasoning so that it can be audited, replayed and visualised.

A central design choice concerns the division of labour between a language model and deterministic code. The model is useful for reading intent ("this is a real-time customer chat, keep latency low"), but it is not trusted with facts about the world or with the final choice. All facility selection is made by a deterministic engine whose rules can be read, tested and reproduced. This keeps the AI component helpful but contained.

This paper is also deliberate about what the numbers mean. The platform currently models, rather than measures, most of its environmental telemetry (Section III-B). It is therefore best described as a working, well-tested method and prototype for carbon-aware routing, ready to be connected to authoritative data feeds, not as a source of verified emission savings.

The main contributions are: an end-to-end sustainability-aware routing pipeline (telemetry collection, normalisation, scoring, eligibility filtering, tiebreaking and audit logging) exposed through a documented REST API; a transparent four-factor Green Intelligence Score with stated weights, bounds and grade bands; a guardrailed use of a language model that interprets workload intent into a validated schema, cannot output telemetry or pick a facility, and degrades to a rule-based parser; a deterministic multi-level tiebreaker hierarchy with user-policy variants and data-residency control; a secured multi-user application with a multilingual dashboard; and a test and load-testing harness of 119 automated tests. The paper follows the IMRaD structure: Section II reviews related work, Section III presents the method and implementation, Section IV reports results, Section V discusses their meaning and validity, and Section VI concludes.

## II. LITERATURE REVIEW

Work on sustainable computing falls into a few families. Table I summarises their typical coverage, as characterised by the authors; it is a conceptual comparison and not a measured benchmark of specific products.

TABLE I. TYPICAL CAPABILITY COVERAGE BY APPROACH

| Approach | Carbon | Water | Latency limits | Explain-able | NL intake |
|---|---|---|---|---|---|
| Nearest-region / DNS routing | None | None | Partial | Limited | None |
| Cost-optimising schedulers | None | None | Full | Partial | None |
| Carbon dashboards (reporting only) | Reporting | Limited | None | Limited | None |
| Carbon-aware batch shifting | Full | None | Limited | Partial | None |
| DCIM facility tools (single site) | Limited | Limited | None | Limited | None |
| Offline carbon-accounting tools | Reporting | None | None | Limited | None |
| CydroFlux AI | Modelled input | Modelled input | Full | Full | Full |

"Full" refers to the presence of the capability in the design, not to validated accuracy. Carbon and water are "modelled input" because those values are currently derived from regional baselines (Section III-B).

### *A. Data-Centre Energy and the Footprint of AI*

Global data-centre energy use has been estimated to have grown far more slowly than workloads thanks to efficiency gains, but AI is changing that trajectory [1]. Studies of large-model training report substantial emissions that depend strongly on where and when the computation runs [2], and water consumption for cooling is a separate, regionally varying cost of AI that is often overlooked [3]. These findings motivate treating location as a first-class scheduling variable.

### *B. Carbon-Aware Computing*

Industrial work has shown that shifting flexible workloads in time and space can reduce emissions without harming service levels [4]. Such systems usually target delay-tolerant batch jobs and rely on proprietary grid forecasts; interactive services such as chatbots and code assistants cannot wait hours in a queue. CydroFlux AI is complementary: it is a small, open, inspectable router that handles latency-sensitive and flexible requests alike by applying guardrails first and sustainability second.

### *C. Facility Management and Carbon Accounting*

Data-centre infrastructure management (DCIM) suites monitor room-level airflow, power and cooling, but are confined to one building and have no view of inter-regional carbon intensity or network latency targets. Offline carbon-accounting tools estimate emissions after the fact from execution time and average grid factors; they are valuable for reporting but do not change where a request runs. Neither class prevents emissions at request time.

### *D. Efficiency Metrics*

Water usage effectiveness (WUE) was defined by The Green Grid as a data-centre sustainability metric expressed in litres per kilowatt-hour [5]. Power usage effectiveness (PUE) is its better-known sibling. CydroFlux AI adopts WUE (L/kWh) and grid carbon intensity ($gCO_2$/kWh) as inputs, and treats renewable share and server load as further dimensions.

### *E. Energy-Proportional Operation*

Servers draw disproportionate power at low utilisation, so consolidation and headroom both matter [6]. The GIS therefore includes a small credit for spare capacity, and the engine refuses facilities above 90% load.

### *F. Language Models for Intent Extraction*

Open-weight model families such as Gemma [7] can parse unstructured text into structured fields when prompted with a schema. The risk is hallucination of facts. CydroFlux AI constrains the model to intent and preferences, validates its output against a schema, and keeps all operational numbers under the control of the sensor layer.

### *G. Authentication and Web Security*

Passwords are stored with a memory-hard function; Scrypt was designed to make large-scale password guessing expensive [8], and current guidance recommends adaptive, salted hashing [9]. JSON Web Tokens give a compact signed session claim [10]. The service follows a resource-oriented REST style [11].

### *H. Research Gap*

Existing approaches each cover a part of the problem. Few combine guardrail-first eligibility, a multi-factor sustainability score, deterministic tie resolution, a guarded language-model front end and a full audit trail in one open application. The contribution of this work is integrative and operational. It does not claim a new optimisation algorithm.

## III. METHODOLOGY

CydroFlux AI processes every workload through a six-stage loop: (1) collect telemetry for each facility; (2) normalise each metric to a common 0–100 scale; (3) score each facility; (4) filter by eligibility rules; (5) choose the winner with deterministic tiebreakers; and (6) log and return the decision with its reasoning. Natural-language interpretation is an optional stage before step 4 that can tighten the constraints. Fig. 1 shows the structure that realises this loop and Fig. 2 shows the loop itself.

### *A. System Architecture*

The system has three tiers. The frontend (React) handles authentication, request submission, results, maps, analytics and chat. The backend (FastAPI) owns telemetry, scoring, routing, authentication, AI interpretation and persistence, exposing REST endpoints under /api. The data tier is MongoDB Atlas, with an in-memory repository used automatically when no database URI is configured, plus a JSON-lines file that receives every routing decision. Weather context comes from the Open-Meteo forecast API [12]. A background scheduler refreshes telemetry every five minutes (configurable), and an initial collection runs at start-up when storage is empty. The frontend never opens the database directly.

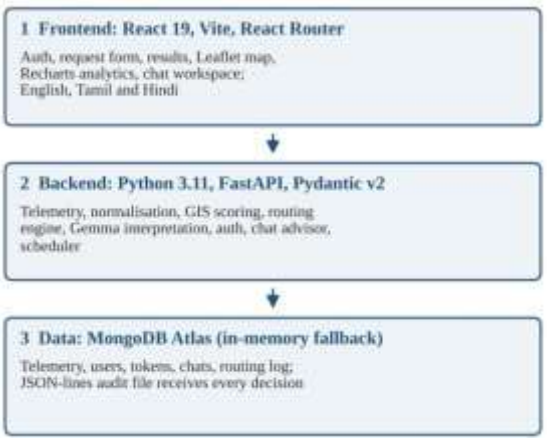



Fig. 1. CydroFlux AI layered architecture.

### *1) Design Principles*

Three principles run through the architecture. *Separation of concerns:* acquisition, decision and platform services are distinct modules. *Replaceable data sources:* each source sits behind a service, so a modelled carbon baseline can later be swapped for a live feed. *Inspectability:* every stage returns its intermediate values, so a decision can be traced from raw telemetry to the selected facility, and the stored record keeps that trace for later replay.

### *2) Module Roles*

Table II falls into three roles. The *acquisition* modules (weather, carbon, latency and aggregator services) gather one unified record per facility. The *decision* modules (normalization, scoring, ranking, routing engine and Gemma service) turn that record and the request into a selected facility. The *platform* modules (auth, chat and repository) provide accounts, the advisor and persistence. If a source fails, the affected

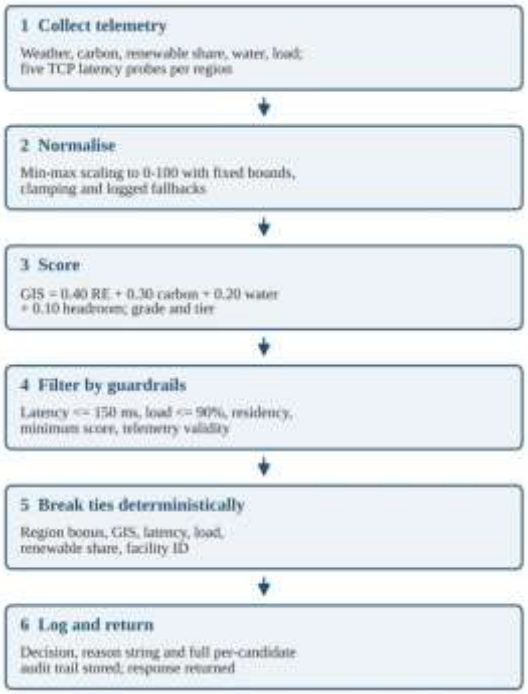



Fig. 2. Routing pipeline from telemetry collection to the stored decision.

facility is marked UNAVAILABLE and excluded, so one outage shrinks the candidate set instead of stopping the request.

TABLE II. BACKEND MODULES AND RESPONSIBILITIES

| Module | Responsibility |
|---|---|
| weather_service | Current temperature, humidity and wind from the Open-Meteo forecast API |
| carbon_service | Regional grid carbon intensity (modelled baseline with ±3% variation, flagged as estimated) |
| latency_service | Five TCP handshake probes to each region's EC2 endpoint on port 443 |
| aggregator_service | Parallel collection (thread pool, up to 8 workers) into a unified record per facility |
| normalization_service | Min–max scaling of each metric to 0–100 with clamping and fallbacks |
| scoring_service | GIS calculation, letter grade and sustainability tier |
| ranking_service | Leaderboard ordering and pagination |
| routing_engine | Eligibility rules, tiebreakers, reasons, fallback, batch and what-if simulation |
| gemma_service | Workload interpretation with schema validation and rule-based fallback |
| auth_service | Registration, verification, login, lockout, tokens and roles |
| chat_service | Rule-driven green-routing advisor that calls the routing engine |
| repository | Telemetry, user, token, chat and routing-log persistence |

## *B. Telemetry and Its Provenance*

Six regions are configured, each with coordinates, an EC2 endpoint and baseline values (Table III). The provenance of each metric matters for interpretation.

TABLE III. CONFIGURED FACILITIES AND BASELINES

| ID | Location | Carbon ($gCO_2$/kWh) | RE (%) | Water (L/kWh) | Load (%) |
|---|---|---|---|---|---|
| AWS-MUM | Mumbai, India | 632 | 24 | 1.40 | 68 |
| AWS-SIN | Singapore | 395 | 18 | 1.20 | 72 |
| AWS-TYO | Tokyo, Japan | 465 | 32 | 0.65 | 58 |
| AWS-FRA | Frankfurt, Germany | 320 | 54 | 0.35 | 64 |
| AWS-IRE | Ireland | 280 | 72 | 0.45 | 52 |
| AWS-ORE | Oregon, USA | 145 | 89 | 0.25 | 42 |

- **Weather** is retrieved live from Open-Meteo and stored with each record. It is collected for context and display; it is not an input to the GIS formula.
- **Network latency** is measured by TCP handshake probes from the server (five samples with a short pause between them). When the user supplies a location, the engine instead estimates transit latency from the great-circle (haversine) distance $d$ in km: $8 + 0.02d$ ms below 50 km, and $12 + 0.018d$ ms otherwise.
- **Carbon intensity** is a regional baseline with ±3% random variation, and each record is flagged is_estimated = true.
- **Renewable share, water usage and server load** are baselines with bounded random variation (±4%, ±3% and ±5% respectively), clamped to valid ranges.

This design makes the system runnable anywhere, offline included, and makes tests reproducible. It also means that the outputs are simulated telemetry. Replacing the modelled sources with authoritative feeds is the first item of future work (Section VI-C). Missing data must not be silently replaced by an environmentally favourable value, and a production observation should carry its source, timestamp, unit and estimation flag so that a fresh measurement can be told from a stale or modelled one.

## *C. Normalisation*

Each metric $x$ is scaled to 0–100 with fixed domain bounds (Table IV) and clamped:

$$= lm(100 \cdot \frac{- mi}{m - mi}, 0, 100) \quad (1)$$

Missing or non-numeric values use a fallback and are logged; a degenerate range returns a neutral 50. Fixed bounds, rather than bounds computed from the current fleet, keep a facility's score stable when other facilities change.

TABLE IV. NORMALISATION BOUNDS

| Metric | Lower bound | Upper bound |
|---|---|---|
| Carbon intensity ($gCO_2$/kWh) | 50 | 800 |
| Water usage (L/kWh) | 0.10 | 2.50 |
| Server load (%) | 0 | 100 |
| Renewable share (%) | 0 | 100 |
| Latency (ms) | 5 | 300 |

## *D. Green Intelligence Score*

The GIS rewards high renewable share, low carbon, low water use and spare capacity:

$$I = 0.40\,E + 0.30 + 0.20 + 0.10 \quad (2)$$

$$= 100 - I, = 100 - , = 100 - \quad (3)$$

where RE is the renewable share (%), $CI_n$ and $W_n$ are the normalised carbon and water values, and L is server load (%). The weights sum to 1.0, which the code asserts at import time, and the result is clamped to [0, 100]. Each component is returned separately so that a user can see exactly where points came from. Table V lists the weights and Table VI the grade bands; a facility with a score of 70 or more is flagged as recommended.

TABLE V. GIS WEIGHTS

| Dimension | Weight | Max points | Goal |
|---|---|---|---|
| Renewable energy | 0.40 | 40 | Maximise clean energy |
| Carbon intensity | 0.30 | 30 | Minimise grid emissions |
| Water usage (WUE) | 0.20 | 20 | Minimise evaporative cooling |
| Server load | 0.10 | 10 | Maximise capacity headroom |

TABLE VI. GIS GRADE BANDS

| Score | Grade | Tier |
|---|---|---|
| 85–100 | A+ | Ultra Green |
| 70–84.99 | A | High Efficiency |
| 55–69.99 | B | Moderate |
| 40–54.99 | C | Low Efficiency |
| below 40 | D | High Carbon |

*Worked example (AWS-ORE).* RE = 89 gives 35.60 points. Carbon 145 normalises to (145 − 50)/750 ×

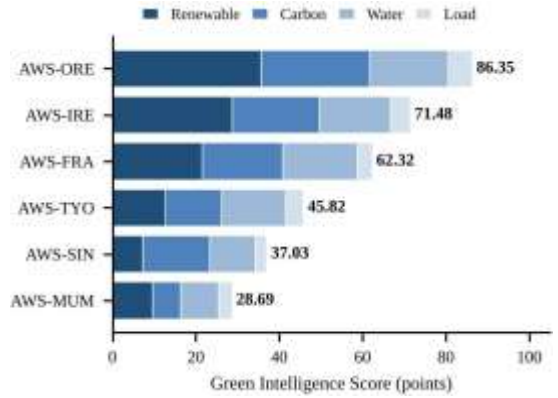



Fig. 3. GIS composition by facility; points contributed by each dimension at baseline values.

100 = 12.67, giving (100 − 12.67) × 0.30 = 26.20. Water 0.25 normalises to 6.25, giving 18.75. Load 42 gives 5.80. The total is 86.35, grade A+. Applying the same code to the six baselines (without random variation) gives the ranking in Table VII and Fig. 3. Because live values vary by a few per cent, scores shown in the application differ slightly from these baselines.

TABLE VII. GIS OF EACH FACILITY AT BASELINE VALUES (POINTS)

| Facility | RE | Carbon | Water | Load | GIS | Grade |
|---|---|---|---|---|---|---|
| AWS-ORE | 35.60 | 26.20 | 18.75 | 5.80 | 86.35 | A+ |
| AWS-IRE | 28.80 | 20.80 | 17.08 | 4.80 | 71.48 | A |
| AWS-FRA | 21.60 | 19.20 | 17.92 | 3.60 | 62.32 | B |
| AWS-TYO | 12.80 | 13.40 | 15.42 | 4.20 | 45.82 | C |
| AWS-SIN | 7.20 | 16.20 | 10.83 | 2.80 | 37.03 | D |
| AWS-MUM | 9.60 | 6.72 | 9.17 | 3.20 | 28.69 | D |

### *E. Eligibility Guardrails*

Before any facility can receive a workload, it must pass the following rules, evaluated for every candidate and recorded with a reason when it fails:

1. **Latency:** latency ≤ 150 ms (default; the request can tighten or relax it).
2. **Capacity:** server load ≤ 90%, guaranteeing at least 10% headroom.
3. **Data residency:** if the request names a policy (IN, EU, APAC, US and similar), only facilities in that region qualify (for example EU permits Frankfurt and Ireland).
4. **Minimum score:** an optional floor on the GIS.
5. **Telemetry validity:** a facility whose data status is UNAVAILABLE is excluded.

Because guardrails are applied first, sustainability can never override an operational or legal constraint.

### *F. Deterministic Selection and Tiebreakers*

Among eligible facilities, the winner is the maximum of an ordered key $\kappa(f)$, compared lexicographically (Eqs. (4) and (5), Section III-H): a requested region first; then the higher GIS; then lower latency; then lower server load; then higher renewable share; and finally the facility identifier, so that no two runs can disagree. Policy preferences re-order the key (Table VIII): a real-time request or the prioritize_latency flag puts latency first; prefer_lower_carbon puts carbon first; prefer_renewable puts renewable share first. Ties are annotated in the explanation, for example "resolved tie via lower latency".

TABLE VIII. POLICY OPTIONS AND THE ORDERING THEY SELECT

| Request condition | Ordering applied to eligible facilities |
|---|---|
| Default (no flags) | GIS, then latency, load, renewable share, facility ID |
| Real-time request or prioritize_latency | Latency first, then GIS and the remaining keys |
| prefer_lower_carbon | Carbon intensity first, then GIS and the remaining keys |
| prefer_renewable | Renewable share first, then GIS and the remaining keys |
| Preferred region named | Preferred-region bonus first, if that facility is eligible |
| Residency policy (IN, EU, APAC, US…) | Not an ordering: restricts the candidate set (EU permits Frankfurt, Ireland) |

If no facility is eligible, the engine returns a diagnostic audit explaining why each candidate failed (HTTP 404/422), or, when the request allows fallback, selects the best available facility and marks the decision is_fallback. A fallback is never reported as though every original constraint had been satisfied.

*Worked example (client in Chennai).* Estimated transit latencies from Chennai are Mumbai 30.6 ms, Singapore 64.2 ms, Tokyo 127.4 ms, Frankfurt 148.6 ms, Ireland 167.1 ms and Oregon 247.9 ms. Ireland and Oregon fail the 150 ms rule. Among the four eligible facilities the highest baseline GIS belongs to Frankfurt (62.32), which therefore wins, even though Mumbai is five times closer. For a client in Portland, Oregon (8.0 ms) and Ireland (146.7 ms) are eligible and Oregon wins with 86.35. The two examples show the intended trade-off: sustainability decides among facilities that are already fast enough.

### *G. Language-Model Workload Interpretation*

Users may describe a job in plain language, for example "nightly batch embedding of 200 GB, cost and carbon matter more than speed". The gemma_service module sends the description to the model configured in settings (default gemma-4-26b-a4b-it, a Gemma-family model [7]) through the Google GenAI SDK, with a direct HTTPS call as a second path and a 10-second timeout. A system instruction restricts the model to one role: returning a single JSON object with the fields listed in Table IX (Fig. 4).




Fig. 4. Guardrailed workload interpretation, from free text to a validated structure; the engine alone chooses.

TABLE IX. STRUCTURED OUTPUT OF THE INTERPRETATION STAGE

| Field returned by the model | Use by the engine |
|---|---|
| Workload category | Context for the audit record and the chat reply |
| Requested priority | May set real-time handling (latency first) |
| Latency preference | Selects tight, normal or relaxed latency handling |
| Suggested maximum latency | Can tighten the 150 ms ceiling |
| Suggested maximum server load | Can tighten the 90% load ceiling |
| Resource requirements | Recorded; not used to override guardrails |
| Routing preferences | Maps to latency, carbon or renewable policy flags |
| Short explanation | Shown to the user as a suggestion |
| Ambiguity flag and clarification request | Prompts the user to refine the description |

Every field is validated against the schema; unknown values are replaced by safe defaults, and none of the fields can name a facility.

Three safeguards keep the model contained. (1) *Negative constraints:* the instruction forbids inventing CPU load, latency, carbon, renewable share or health data and forbids choosing a facility. (2) *Validation:* the reply is treated as untrusted input; it is stripped of any markdown fences, parsed, and checked against the schema, and unknown values are replaced by safe defaults. (3) *Fallback:* when no API key is configured, or the call fails or times out, a domain-grounded rule-based parser produces the same structure from keywords, so the platform works without the model. The interpretation can tighten latency and load limits and add preference flags, but the facility is always chosen by the deterministic engine of Section III-F.

## H. Formal Statement and Complexity

For a request $r$ let $C$ be the set of configured facilities and $E(r) \subseteq C$ the subset that passes every guardrail of Section III-E. The router returns

$$^{*} = r\,m_{\in E(r)}\,() \quad (4)$$

$$() = (_{r}(), I(), -P(), - (), E(), id()) \quad (5)$$

where the maximum is taken in lexicographic order, $b_{r}$ is the preferred-region bonus, $\ell$ is latency and id is the facility identifier. Because the identifier is unique, the maximum is always unique, so the mapping from (request, telemetry snapshot) to facility is a function: identical inputs give identical outputs. With $N = |C|$ facilities, normalisation, scoring, filtering and selection are each a single pass, so a decision costs $O(N)$ arithmetic operations, and $O(N \log N)$ when a full ranking is returned. For the six configured facilities this is negligible next to the cost of network and database calls, which is consistent with the sub-millisecond median reported in Section IV-C.

## I. System Implementation

### 1) Technology Stack

The backend is Python 3.11 with FastAPI [13] and Uvicorn, Pydantic v2 schemas for every request and response, pydantic-settings and python-dotenv for configuration, requests and httpx for outbound calls, pymongo for MongoDB Atlas [14] (with mongomock for tests), and APScheduler for the refresh cycle. Interactive OpenAPI documentation is generated automatically at /docs, and middleware adds security headers (X-Content-Type-Options, X-Frame-Options, X-XSS-Protection, Referrer-Policy). The interface uses React 19 and Vite, React Router for protected routes, Axios for API calls, Leaflet for the global map, Recharts for analytics and Lucide icons, with Bootstrap utilities. It has about twenty page components covering registration and verification, request submission, results and history, a route explorer, data-centre pages, a global map, analytics, service health and a chat workspace. Context providers hold authentication, request state, settings and language, and the interface is available in English, Tamil and Hindi.

### 2) Running and Configuration

The backend starts with python main.py (default port 8000) and the frontend with npm run dev. With no MongoDB URI the system runs on its in-memory repository, so it can be demonstrated on a single laptop; with a URI it persists to MongoDB Atlas, and with a model key the Gemma interpretation stage is enabled. Each step adds capability without changing the routing logic. Table X lists the parameters that shape a decision; a request may tighten or relax the latency ceiling, while the interpretation stage can only tighten the load ceiling.

TABLE X. KEY CONFIGURATION DEFAULTS

| Parameter | Default |
|---|---|
| Latency ceiling (request may tighten or relax) | 150 ms |
| Maximum server load for eligibility | 90% |
| Recommended-facility threshold (GIS) | ≥ 70 |
| Telemetry refresh interval (configurable) | 5 minutes |
| TCP probes per region (port 443) | 5 samples |
| Parallel collection workers | up to 8 |
| Model call timeout | 10 seconds |
| Language-model default | gemma-4-26b-a4b-it |
| Email verification token validity | 24 h, single use |
| JWT session expiry (HS256) | 24 hours |

| | |
|---|---|
| Account lockout | 5 failures → 15 min |
| Telemetry variation (carbon / RE / water / load) | ±3% / ±4% / ±3% / ±5% |

### *3) REST Interface*

In total, the six routers expose 28 REST handlers, in addition to the interface and documentation routes; Table XI lists the principal endpoints. For the Chennai client of Section III-F, a POST to /api/route returns the selected facility (AWS-FRA), its GIS (62.32) and grade (B), the latency (148.6 ms) and the method that produced it (haversine), the counts of candidates evaluated (6) and eligible (4), the exclusion reason for each rejected facility (for example, Ireland: latency 167.1 ms > 150), the fallback flag and a reason string. The same fields are written to the audit log, so the displayed explanation and the stored record cannot drift apart.

TABLE XI. PRINCIPAL REST ENDPOINTS

| Method | Endpoint | Purpose |
|---|---|---|
| POST | /api/route | Dispatch a workload to the optimal facility |
| GET | /api/route/history | Recent decisions from the audit log |
| GET | /api/route/stats | Totals, mean score and latency, facility distribution |
| POST | /api/route/simulate | What-if routing on a hypothetical candidate set |
| GET | /api/scores | All facilities ranked by GIS with component breakdown |
| GET | /api/datacenters | Current telemetry for all facilities |
| GET, POST | /api/ai/status, /api/ai/interpret | Model identity; interpret a workload description |
| POST | /api/auth/… | Register, verify email, log in, current user |
| … | /api/chat/conversations… | Create, list, rename, delete conversations; post messages |
| GET | /health | Database, scheduler and record-count status |

### *4) Authentication and Access Control*

Passwords are hashed with Scrypt through Werkzeug. Registration creates a single-use verification token valid for 24 hours and sends it by SMTP when mail is configured. Login is refused for unverified accounts, and five consecutive failures lock the account for 15 minutes. Successful login issues an HS256-signed JWT (24-hour expiry) with the user's role. Two roles exist, regular_user and cydroflux_privileged, and a dependency blocks privileged-only routes for everyone else. The reviewed registration routine does not read the optional role field, so the privileged role is not self-assignable through that path.

### *5) Routing Audit Log*

Every decision is written to MongoDB and appended to a JSON-lines file. A record holds the decision identifier, selected facility, score, latency, load, carbon, renewable share, grade, the reason string, whether fallback was used, counts of candidates evaluated and eligible, the request context, the processing time and a full per-candidate audit trail with ineligibility reasons. This record is what makes a decision replayable and explainable.

### *6) Chat Advisor and Load Testing*

A rule-driven advisor recognises routing intent in a message, infers an origin and workload type from keywords, calls the same routing engine and replies with the chosen facility, estimated latency, carbon, renewable share, GIS and the reason; conversations are stored per user. run_load_test.py is a dependency-light multi-threaded tester that reports requests per second, success rate and p50/p95/p99 latency, and locustfile.py provides Locust scenarios. Results of runs with these tools are not included in this paper (Section V-C).

### *7) Security Controls*

Table XII maps each threat to the control in the release. Three items need tightening before production use and are release risks, not completed protections: wildcard CORS is enabled alongside credentials, default secrets require review, and IP-based API rate limiting was not observed.

TABLE XII. THREATS AND CONTROLS

| Threat | Control in the release | Status |
|---|---|---|
| Password disclosure | Scrypt salted hashing | Implemented |
| Credential guessing | 5-attempt, 15-minute lockout | Implemented |
| Fake accounts | Single-use email verification, 24 h expiry | Implemented |
| Session forgery | HMAC-SHA256 signed token, 24 h expiry | Implemented |
| Privilege misuse | Role dependency on privileged routes | Implemented |
| Clickjacking / MIME sniffing | Security response headers | Implemented |
| Over-permissive CORS | Wildcard origin enabled together with credentials | To be tightened |
| Secret exposure | Defaults exist in code; a populated .env is present in the project folder | To be reviewed |
| Brute force at network level | No IP-based API rate limiting observed | To be reviewed |

## *J. Evaluation Design*

Three kinds of evidence are used. First, an automated suite of 119 tests exercises every layer from telemetry to the end-to-end pipeline, using fixed inputs for normalisation, score arithmetic, grade bands, tiebreaker ordering and residency filtering, mocked HTTP, database and model responses for services, and schema, status-code and role checks for the API. Second, the routing audit log written during development is analysed descriptively. Third, the routing rules are applied analytically to the baseline values of Table III for nine client locations, five weight profiles and several latency ceilings. The test suite, audit log and source code form part of project release v1.0.0 [15], and the analytical results in Sections IV-D to IV-F can be reproduced from Table III and the formulas above.

# IV. RESULTS

## *A. Automated Test Suite*

The project contains 119 automated tests in 18 test modules (Table XIII). In our verification run, 118 passed and 1 failed. The failing test,

test_complete_pipeline_flow, asserts that every facility reports valid data after a real collection cycle. The verification environment had no internet access, so the weather API and the TCP probes failed and the facilities were correctly marked UNAVAILABLE. The failure therefore reflects the environment, not a logic fault, but the test should be made network-independent by mocking the weather provider and TCP probes, with a separately labelled integration test for real connectivity (Section VI-C).

TABLE XIII. TEST COVERAGE BY AREA

| Area | Modules | Tests |
|---|---|---|
| Telemetry (weather, carbon, latency, aggregation, database, API) | 6 | 25 |
| Normalisation | 1 | 8 |
| Scoring, ranking and score API | 3 | 28 |
| Routing engine, dynamic routing, routing API, audit logging | 4 | 38 |
| Gemma interpretation and fallback | 1 | 12 |
| Authentication, roles and chat | 2 | 3 |
| End-to-end pipeline | 1 | 5 |
| Total | 18 | 119 |

### *B. Analytical Behaviour of the Score*

Because the GIS is linear, its sensitivity can be derived exactly. One percentage point of renewable share is worth 0.40 points. One $gCO_2$/kWh of carbon is worth $0.30 \times 100/750 = 0.04$ points. One additional 0.1 L/kWh of water is worth $0.20 \times 100/2.4 \approx 0.83$ points, and one percentage point of load is worth 0.10 points. Renewable share therefore dominates, and the spread between the best and worst baseline facilities (86.35 versus 28.69) is driven mostly by the combination of renewable share and carbon. The ±3–5% telemetry variation moves a facility's score by only a few points, which is why Oregon remains first under all variation seen in the log, while middle-ranked facilities could in principle swap places.

The latency guardrail has a large effect on outcomes. With the distance model, a client in Chennai cannot use Oregon or Ireland under the default 150 ms limit and is routed to Frankfurt (148.6 ms estimated) on its baseline GIS. This sits just under the limit, so a small change in the estimate would alter the result. The result shows how strongly geography and the latency threshold shape routing.

### *C. Observations from the Routing Audit Log*

During development, the engine wrote 1,777 decision records between 21 and 27 October 2026. They come from manual use, automated tests and ad hoc load, not from production traffic. Of these, 1,163 records evaluate all six real facilities (excluding synthetic test facilities and reduced candidate sets), and Table XIV and Fig. 5 summarise them. One of these 1,163 decisions used fallback. The skew toward Oregon is expected from the scoring: Oregon has the highest GIS and wins whenever it passes the guardrails. Tokyo and Singapore appear when the client is nearer Asia and the latency rule excludes the western




Fig. 5. Facility selected in logged decisions (six-candidate records, n = 1,163).

facilities. Because the log mixes test scenarios and does not record real workload volume or compute demand, the distribution shows that the rules behave as designed and must not be taken as a fleet-wide efficiency result.

TABLE XIV. FACILITY SELECTED IN LOGGED DECISIONS (SIX-CANDIDATE RECORDS, N = 1,163)

| Facility | Decisions | Share |
|---|---|---|
| AWS-ORE (Oregon) | 910 | 78.2% |
| AWS-TYO (Tokyo) | 162 | 13.9% |
| AWS-IRE (Ireland) | 44 | 3.8% |
| AWS-SIN (Singapore) | 30 | 2.6% |
| AWS-MUM (Mumbai) | 16 | 1.4% |
| AWS-FRA (Frankfurt) | 1 | 0.1% |

The median routing computation time in these records was 0.78 ms, the 95th percentile 3.4 ms and the 99th percentile roughly 12.6 seconds. The long tail was not investigated. A plausible, unconfirmed explanation is that some timed calls included outbound requests such as model interpretation or telemetry refresh. A controlled load test with the supplied tools is needed to characterise throughput; the 0.78 ms median is an observation from the development log, not an end-to-end performance guarantee.

### *D. Policy-Variant Analysis*

To show how policy choices change outcomes, the routing rules of Section III were applied analytically to the six baseline facilities (no random variation) for client locations in nine cities, using the distance-based latency model and the default 150 ms ceiling. Table XV lists the facility selected under the default policy and under each policy variant. These runs are deterministic evaluations of the method on baseline values; they are not measurements and do not use the audit log.

TABLE XV. FACILITY SELECTED PER CLIENT LOCATION AND POLICY (BASELINE VALUES)

| Client | Elig. | Default | Latency | Carbon | Renew. |
|---|---|---|---|---|---|
| Chennai | 4 | FRA (148.6 ms) | MUM (30.6 ms) | FRA | FRA |
| Portland | 2 | ORE (8.0 ms) | ORE | ORE | ORE |
| Frankfurt | 3 | IRE (31.6 ms) | FRA (8.0 ms) | IRE | IRE |
| London | 3 | IRE (20.3 ms) | IRE | IRE | IRE |
| Mumbai | 5 | IRE (148.9 ms) | MUM (8.0 ms) | IRE | IRE |
| Singapore | 3 | TYO (107.7 ms) | SIN (8.0 ms) | SIN | TYO |
| Tokyo | 3 | TYO (8.0 ms) | TYO | SIN (107.7 ms) | TYO |

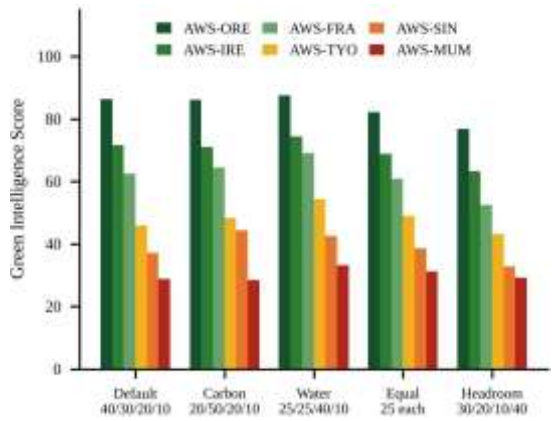



Fig. 6. GIS of each facility under five weight profiles (renewable / carbon / water / headroom, baseline values).

| | | | | | |
|---|---|---|---|---|---|
| Sydney | 1 | SIN (125.5 ms) | SIN | SIN | SIN |
| São Paulo | 0 | none eligible | none | none | none |

Columns: eligible facilities within 150 ms and ≤ 90% load; facility chosen by the default (GIS-first), latency-first, lowest-carbon and most-renewable policies. For São Paulo the nearest facility is beyond the ceiling, so the engine returns a diagnostic result, or a marked fallback if the request allows it.

Three behaviours stand out. First, the policy matters only where several facilities are eligible: a Portland client has two candidates and both Oregon-first and latency-first agree. Second, a latency-first policy gives up substantial sustainability for proximity: a Mumbai client moves from Ireland (GIS 71.48, 148.9 ms) to Mumbai (GIS 28.69, 8.0 ms). Third, the lowest-carbon policy can differ from the default: for a Tokyo client it prefers Singapore (395 $gCO_2$/kWh) over Tokyo (465) even though Tokyo has the higher GIS. A São Paulo client shows the guardrail-first principle most clearly: no facility is both reachable and acceptable, so no sustainability score is allowed to decide the outcome.

### *E. Sensitivity to the GIS Weights*

The 40/30/20/10 weights were chosen by the developers, so it is important to know whether the ranking depends on them. Five alternative weight profiles were evaluated on the baseline facilities (Fig. 6). The ordering of the six facilities was identical under every profile: Oregon, Ireland, Frankfurt, Tokyo, Singapore, Mumbai. Only the spread between facilities changed, from about 58 points (default, 86.35 to 28.69) to about 48 points (headroom-led, 76.7 to 29.1).

A second test lowered the renewable weight from 0.40 to zero while rescaling the other three weights proportionally. Oregon and Ireland remained first and second throughout; Frankfurt closed the gap to Ireland but did not overtake it (71.1 versus 67.9 at zero renewable weight). Applying the full ±3%/±4%/±3%/±5% telemetry variation of Section III-B in the least and most favourable directions moves Oregon between 84.5 and 88.2, Ireland between 69.6 and 73.3 and Frankfurt between 60.7 and 64.0, so the top three never swap places. At baseline, therefore, the ranking is robust to both weights and noise, and it is the guardrails, chiefly the latency ceiling, that determine which facility actually wins for a given client. This conclusion is conditional on the six baseline values; with live telemetry the ranking should be re-examined.

### *F. Sensitivity to the Latency Ceiling*

Table XVI varies the latency ceiling for the Chennai client, holding everything else at the default. Each increase in the ceiling admits one more facility and, because the more distant facilities are cleaner, moves the winner westward until Oregon becomes reachable.

TABLE XVI. WINNER FOR A CHENNAI CLIENT AS THE LATENCY CEILING VARIES

| Ceiling (ms) | Eligible facilities | Winner | GIS |
|---|---|---|---|
| 50 | MUM | AWS-MUM | 28.69 |
| 100 | MUM, SIN | AWS-SIN | 37.03 |
| 130 | MUM, SIN, TYO | AWS-TYO | 45.82 |
| 150 (default) | MUM, SIN, TYO, FRA | AWS-FRA | 62.32 |
| 170 | … + IRE | AWS-IRE | 71.48 |
| 250 or more | all six | AWS-ORE | 86.35 |

The table makes the cost of the default explicit. A 150 ms ceiling excludes Ireland by only 17 ms, and Frankfurt qualifies by only 1.4 ms, so the result for this client is sensitive to the latency estimate. Operators who accept a higher ceiling for batch work, or who measure real round-trip times instead of estimating them, would obtain a cleaner choice at the price of slower responses. This is exactly the trade-off the request-level ceiling is meant to expose.

## V. DISCUSSION

### *A. Interpretation and Implications*

The results support seven properties of the design. (1) *Guardrails before greenness:* sustainability is a tie-breaker among facilities that already meet latency, capacity and residency rules, so the router cannot trade user experience or compliance for a better score (Sections IV-D and IV-F). (2) *Explainable and reproducible decisions:* every number comes from a stated formula, every rejection carries a reason, and ties are resolved by a fixed ordering. (3) *A contained use of AI:* the language model reads intent only; its absence degrades convenience, not correctness. (4) *Runs anywhere:* the in-memory repository, simulated telemetry and rule-based fallback allow a full demonstration on one machine. (5) *Extensible by design:* facilities, weights, bounds and data sources are configuration entries or services that can be swapped. (6) *Operational visibility:* dashboards, a map, history and an audit log show where work went and why. (7) *Tested:* 118 of 119 tests pass, and the failure is environmental.

The GIS is a weighted composite, not a direct measurement of energy consumed or emissions avoided. Returning each component alongside the total prevents a single number from appearing more authoritative than its inputs. The strongest conclusion

supported by the current release is that CydroFlux AI demonstrates an inspectable decision pipeline combining a transparent sustainability score with operational guardrails. A credible environmental claim would additionally need a defined baseline, measured energy or a defensible workload energy model, location- and time-specific grid intensity, and equivalent work under the same service constraints, with water impact reported separately.

### *B. Threats to Validity*

*Construct validity:* the GIS summarises four modelled indicators; it does not measure the emissions of a workload. *Internal validity:* telemetry variation is random, so repeated runs differ slightly, and the log mixes test and manual traffic. *External validity:* six regions, one cloud provider and one development environment limit generalisation. *Weighting:* the 40/30/20/10 weights were chosen by the developers and not fitted to outcome data; other weights may be equally defensible.

### *C. What Has Not Been Measured*

The paper does not include measured carbon, energy or water savings; a comparison against nearest-region or cost-based routing; accuracy of the model's interpretation against labelled workload descriptions; sustained-load or concurrent-user results; or a user study. These are the priorities for the next phase. Fig. 7 summarises the boundary between what was verified and what remains pending.

### *D. Limitations, Risks and Ethical Considerations*

Table XVII lists the main limitations with their current mitigation and the planned further action.

TABLE XVII. LIMITATIONS AND MITIGATIONS

| Limitation | Current mitigation | Further action |
|---|---|---|
| Environmental telemetry is modelled, not live | Records flagged as estimated; limits stated openly | Integrate real grid and facility data sources |
| Latency estimated from distance when a location is given | Measured TCP probes used otherwise | Blend measured round-trip data per client region |
| Fixed developer-chosen weights | Weights and bounds exposed in code and API | Sensitivity analysis; configurable policy profiles |
| Model output may be wrong or ambiguous | Schema validation, safe defaults, ambiguity flag, no authority over facility choice | Evaluate against labelled prompts |
| Rule-based chat advisor uses simple keyword matching | Delegates all decisions to the routing engine | Intent classification and tests for edge phrases |
| Concentration on the greenest region | Latency, load and residency guardrails | Capacity-aware load spreading and quota limits |
| Personal data (accounts, chats) | Hashed passwords, role checks | Encryption at rest, retention policy, privacy review |




Fig. 7. Verification boundaries: verified behaviour and pending independent evidence.

Several ethical points deserve attention. A routing system that always favours the greenest region can concentrate load and local impact, including water use, in one place; a fairness or capacity-spreading policy, for example capping any facility's share of fleet demand without overriding residency or service-level requirements, is worth studying. Sustainability claims must be backed by verified data, and a modelled score should never be quoted as a carbon saving. User prompts sent to an external model may contain sensitive information, so operators should state what is sent and obtain appropriate consent. Audit logs need role-restricted access and a defined retention period. Finally, a language model's explanation of a request is a suggestion that the user can override, not a decision.

## VI. CONCLUSION

### *A. Takeaways*

CydroFlux AI implements an end-to-end, sustainability-aware routing engine for AI workloads. It collects and normalises telemetry for six data-centre regions, scores each facility with a transparent Green Intelligence Score, enforces latency, capacity and residency guardrails, chooses a facility by a deterministic hierarchy and records the full reasoning. A language model helps interpret natural-language requests but is confined to intent. The system is secured with verified accounts, lockout, signed sessions and roles, and is covered by a 119-test suite in which all environment-independent tests pass. At baseline the facility ranking is robust to the weights and to telemetry noise, and the latency guardrail is what determines the winner for a given client. The contribution is the orchestration and its inspectability, not a new optimisation algorithm.

### *B. Limitations*

The principal caveat is data provenance: because carbon, renewable, water and load values are currently modelled, the platform demonstrates the method and not verified environmental savings. Further limits are the unexplained 12.6 s 99th-percentile timing, the absence of load-test results, and the three open security items of Section III-I.

### *C. Future Work*

Table XVIII lists the proposed development roadmap by phase. Future work should preserve the central principle of the project: the model may interpret, but the guardrails and the audit trail decide. With authoritative data feeds, a controlled evaluation and

the security items noted above, CydroFlux AI could become a practical basis for carbon-aware request routing.

TABLE XVIII. PROPOSED DEVELOPMENT ROADMAP

| Phase | Work item | Dependency |
|---|---|---|
| Near term | Replace modelled carbon and renewable values with live grid-intensity feeds | Data-provider access |
| Near term | Make the end-to-end test independent of the network (mock external calls) | Test refactor |
| Near term | Restrict CORS origins, remove default secrets, rotate any exposed keys, add API rate limiting | Security review |
| Near term | Record load-test results (p50/p95/p99, throughput) with the supplied tools | Test environment |
| Mid term | Baseline comparison against nearest-region and cost-only routing, with measured emissions | Evaluation design |
| Mid term | Evaluate model interpretation accuracy on labelled workload descriptions | Annotated dataset |
| Mid term | Configurable policy profiles and weight sensitivity analysis | Study data |
| Long term | Time-shifting for flexible batch jobs using carbon forecasts | Forecast data |
| Long term | Capacity-aware load spreading and fairness across regions | Simulation study |
| Long term | Additional providers and regions; production hosting with encryption at rest | Operations budget |

## ACKNOWLEDGMENT

The authors thank the Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, Coimbatore, for the facilities and academic support provided during the development of CydroFlux AI.

## REFERENCES

[1] E. Masanet, A. Shehabi, N. Lei, S. Smith and J. Koomey, “Recalibrating global data center energy-use estimates,” *Science*, vol. 367, no. 6481, 2020.

[2] D. Patterson *et al.*, “Carbon emissions and large neural network training,” arXiv:2104.10350, 2021.

[3] P. Li, J. Yang, M. A. Islam and S. Ren, “Making AI less ‘thirsty’: Uncovering and addressing the secret water footprint of AI models,” arXiv:2304.03271, 2023.

[4] A. Radovanović *et al.*, “Carbon-aware computing for datacenters,” *IEEE Trans. Power Syst.*, 2023.

[5] The Green Grid, “Water usage effectiveness (WUE): A Green Grid data center sustainability metric,” White Paper, 2011.

[6] L. A. Barroso and U. Hölzle, “The case for energy-proportional computing,” *IEEE Computer*, vol. 40, no. 12, 2007.

[7] Gemma Team, Google DeepMind, “Gemma: Open models based on Gemini research and technology,” arXiv:2403.08295, 2024.

[8] C. Percival, “Stronger key derivation via sequential memory-hard functions,” BSDCan, 2009.

[9] OWASP Foundation, *Password Storage Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/

[10] M. Jones, J. Bradley and N. Sakimura, “JSON Web Token (JWT),” RFC 7519, IETF, 2015.

[11] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, Univ. California, Irvine, 2000.

[12] Open-Meteo, *Weather Forecast API documentation*. [Online]. Available: https://open-meteo.com/

[13] *FastAPI documentation*. [Online]. Available: https://fastapi.tiangolo.com/

[14] *MongoDB Atlas documentation*. [Online]. Available: https://www.mongodb.com/docs/atlas/

[15] CydroFlux AI project records: backend README, Week 1 Routing and Scoring Specifications, source code and automated test suite. Local release v1.0.0, Oct. 2026.

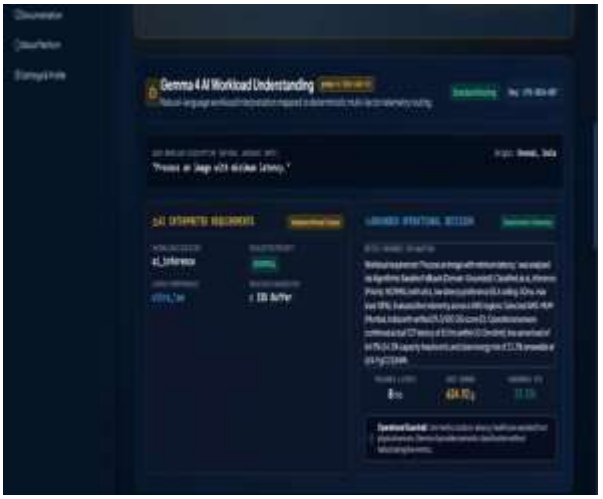

Intelligent Green Routing for a Sustainable Cloud.
AI-Powered Real-Time Carbon, Latency & Renewable Placement


Monitored AWS Data Center Regions
AWS Asia Pacific (Mumbai)
AWS Asia Pacific (Singapore)
AWS Asia Pacific (Tokyo)
29.8
36.9
45.8


Intelligent Routing Verification & Telemetry Audit
AWS Asia Pacific (Mumbai)
29.0
Gemma 4 AI Workload Understanding


Route Workload
Describe Your Workload in Plain English
1. Client Origin & Identity
2. Workload & Guardrail SLA


mapTitle
AWS US West (Oregon)
92.8


Gemma 4 AI Workload Understanding