CarbonSage

Track. Reduce. Sustain.

By Team elasticSearch

For **Hacknosis: Sustainable MedTech**

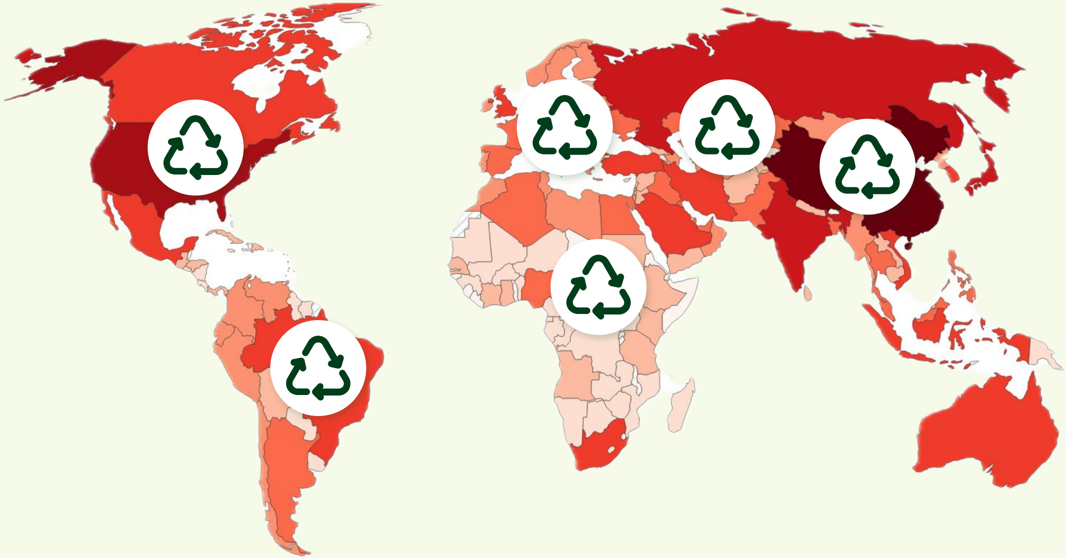
PROBLEM STATEMENT

The carbon footprint of our gadgets, the internet, and the essential systems supporting them accounts for 3.7% of global greenhouse emissions – on par with the airline industry. These emissions are predicted to double by 2025 as more devices hit the market, including smart medical devices.

Key Challenges:

- Increasing carbon emissions from technology usage.
- Need for effective monitoring and reduction strategies.
- Lack of awareness and tools for optimizing energy consumption.

OUR WORLD, NOW!



CO_2 Data of 2022 (tonnes)	
CHINA	11,396,777,000.00
ASIA	10,375,572,000.00
EUROPE	5,105,307,700.00
USA	5,057,303,600.00
AFRICA	1,416,626,600.00
LATAM	1,085,392,400.00

REGION-WISE ANNUAL CO_2 EMISSIONS

Data source: Global Carbon Budget (2023) – Learn more about this data

SOLUTION OVERVIEW

Hackathon Objective: Develop a solution that can calculate, track, and reduce carbon emissions generated by devices or data usage, with auto-reporting using Opentext.

01

MONITOR

Track real-time energy usage and device performance.

02

FORECAST

Predict future energy consumption and identify inefficiencies.

03

REMEDY

Provide actionable insights and suggestions to reduce carbon emissions.

OUR 3 STEP APPROACH

KEY FEATURES

Anomaly Detection preventing Faults

For tracking excessive CO_2 emissions helping prevent potential Faults

Code Alternative

Analyze code block carbon emissions and suggest efficient code alternatives using LLM.

CO_2 Per Microservice

Calculating CO_2 emission and energy use on per microservice basis

Usage Reporting

Generation of daily usage reports on service wise carbon emissions

MONITORING

01 USAGE

Track real-time usage of RAM, GPU, and CPU.

02 STORAGE

Uses AWS timestream to store and perform analytics on time series data

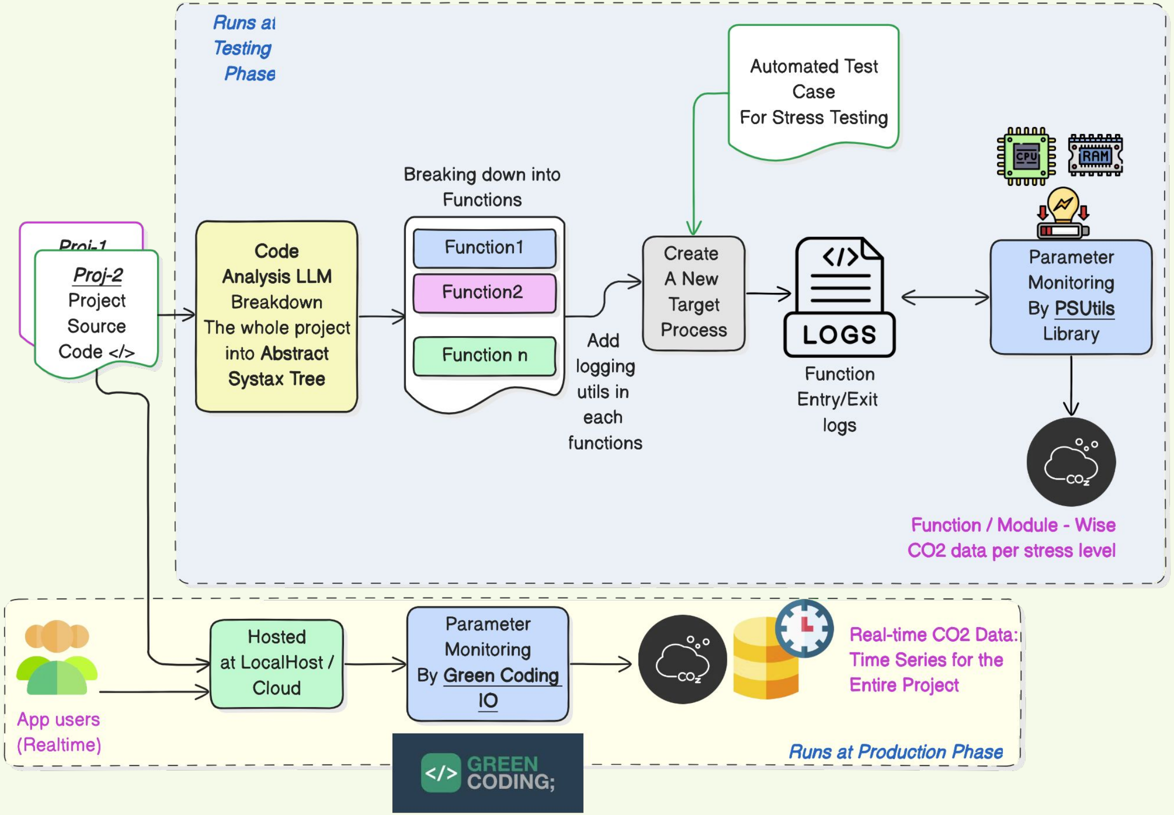
03 MICROSERVICE WISE

Generate function/service wise CO2 generation reports

04 GLOBAL REPORT

Generate time series CO2 generation report for the entire project using Opentext

Data Collection and Storage (1/2)



Data Collection and Storage (2/2)

We store the data mainly in two phases:

1. **Function/Module-Wise CO_2 Data per Stress Level (Stored only at testing phase)**
   a. We break the whole project into multiple microservices using code analysis LLMs such as CodeLlama and host them on different endpoints for stress testing.
   b. We automatically generate test cases with respect to different stress levels.
   c. We monitor different parameters like CPU usage, RAM usage, and power consumption using monitoring and tracking tools such as Green Coding IO.
2. **Real-time CO_2 Data: Time Series for the Entire Project (Stored at production phase / realtime)**
   a. We host the target project on cloud platforms such as AWS, GCP, or Localhost.
   b. We monitor the system parameters with respect to the actual real-time user traffic.
   c. We store the data in a time series format in AWS Timestream.

FORECAST

PREDICTION

Based on the CO_2 time series data and service usage, predict growth of CO_2 emissions

COST-BENEFIT ANALYSIS

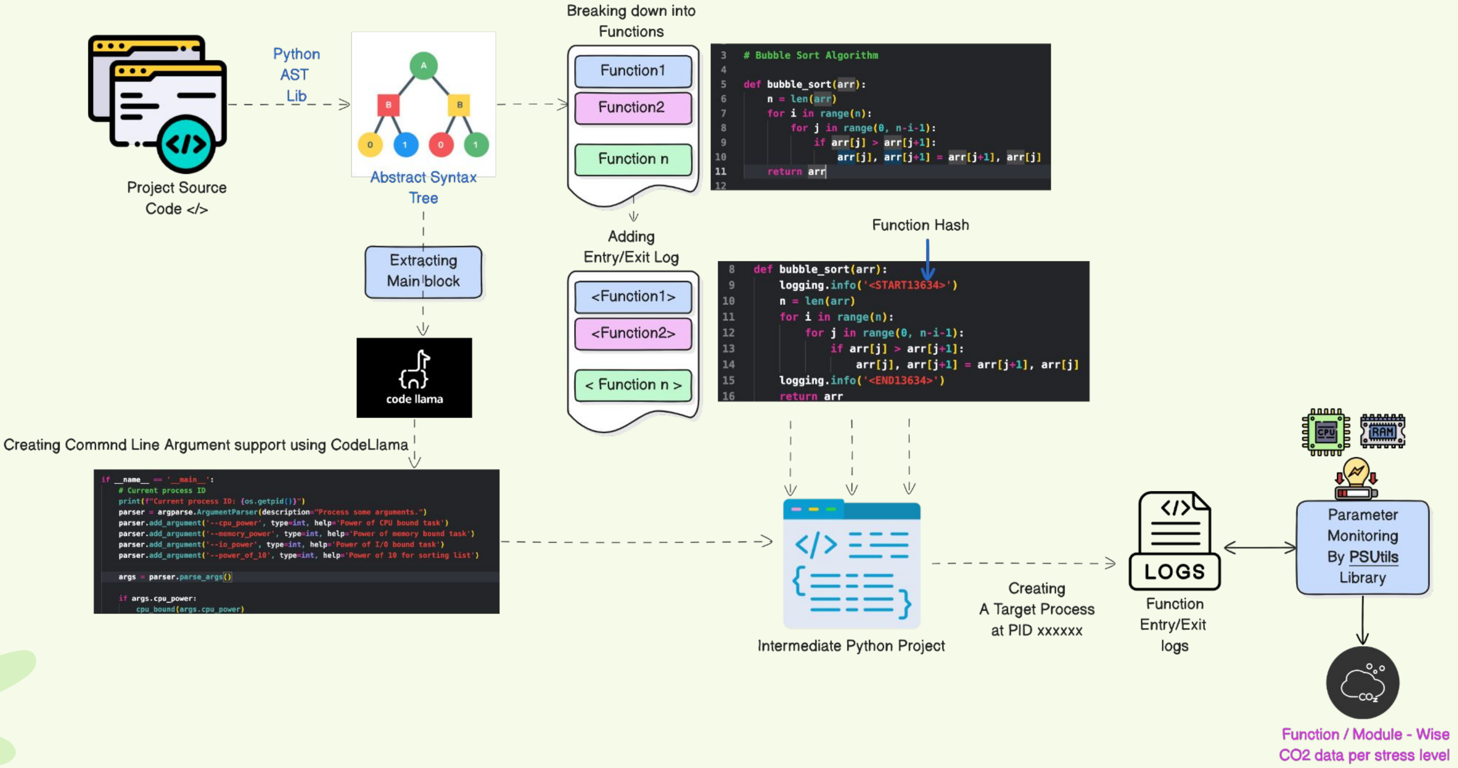
Corroborate CO_2 emission reductions with reduction in lines of code, code efficiency. Estimate effort for target reduction of CO_2 emissions

OPENTEXT ALERT NOTIFICATION

Based on abnormal or high CO_2 emissions, notification is sent to project lead and writer of specific function/microservice

Code profiling for resource usage

Detailed Mechanism



Usage Analytics Generation

Parameter Monitoring Logs Generated By
PSUtils at 1 sec interval

```
cpu_percent,rss,vms,pfaults,pageins,time,iteration
63.9,12361728,34615349248,4265,73,2024-08-24 23:21:14,0
44.8,17432576,34618540032,5560,147,2024-08-24 23:21:15,1
45.5,23371776,34628603904,7319,682,2024-08-24 23:21:16,2
20.5,25726976,34663649280,7954,791,2024-08-24 23:21:17,3
0.2,25726976,34663649280,7954,791,2024-08-24 23:21:18,4
```

Function Entry/Exit logs

```
2024-08-24 23:21:17 - <START17049>
2024-08-24 23:21:18 - <END17049>
2024-08-24 23:21:18 - <START19555>
2024-08-24 23:21:28 - <END19555>
2024-08-24 23:21:28 - <START41611>
2024-08-24 23:21:38 - <END41611>
```

Comparison
Based on time

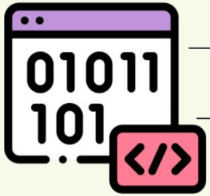
Creating Analytics
-> Total CO2
-> Total Energy Consumed
-> Elapsed time per module etc.

Energy Consumption: $E_{\text{CPU}} = \text{CPU Utilization}(\%) \times \text{CPU Power Consumption}(W) \times \text{Time (h)}$

$E_{\text{Memory}} = \left(\frac{\text{RSS}+\text{VMS}}{2}\right) \times \text{Memory Power Consumption per Byte (W/byte)} \times \text{Time (h)}$

$\text{Total Energy Consumption}(E) = E_{\text{CPU}} + E_{\text{Memory}}$

CO2 Emissions: $\text{CO2 Emissions} = \text{Total Energy Consumption}(E) \times \text{Emission Factor}$



Critical Code Blocks
Where Energy/CO2 >
Threshsold

code llama

Optimized Code Suggetion Using
Code Llama

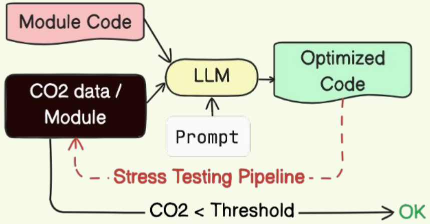
opentext™

Email / Notification
Service

Module Wise parameter
visualization in dashboard

Data Utilisation & Processing (1/2)

Module wise Optimized Code Suggestion



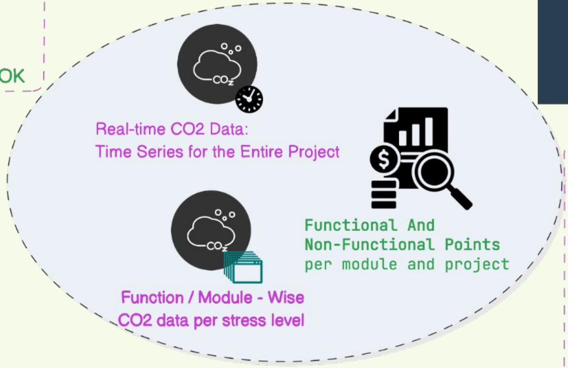
Cost-Benefit Analysis: Comparing Cost-to-Make vs. CO2 Emission Improvement

To assess if CO2 level improvements are significant, check the required changes in functional and non-functional aspects like lines of code, cost, complexity, and time.

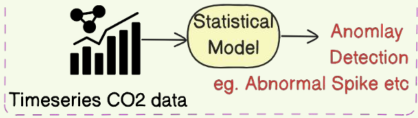
code llama

opentext™

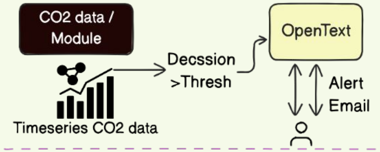
GREEN CODING;



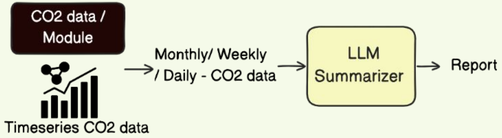
Anomaly Detection on Time Series Data to Prevent Potential Faults



Email Notification in case of Anomaly (OpenText)



Usage Reporting



Data Utilisation & Processing (2/2)

We mainly use three kinds of data :

1. Real-time CO2 Data: Time Series for the Entire Project
2. Function / Module - Wise CO2 data per stress level
3. Functional And Non-Functional Points per module and project

Here are some use cases we offer:

1. Module wise resource estimation
2. Optimized Code Suggestion
3. Anomaly Detection to prevent potential Faults
4. Scheduled Reports
5. Notification Alerts for high CO2 emission and energy use

TOOLS REQUIRED

Services

1. OpenText
2. green-coding

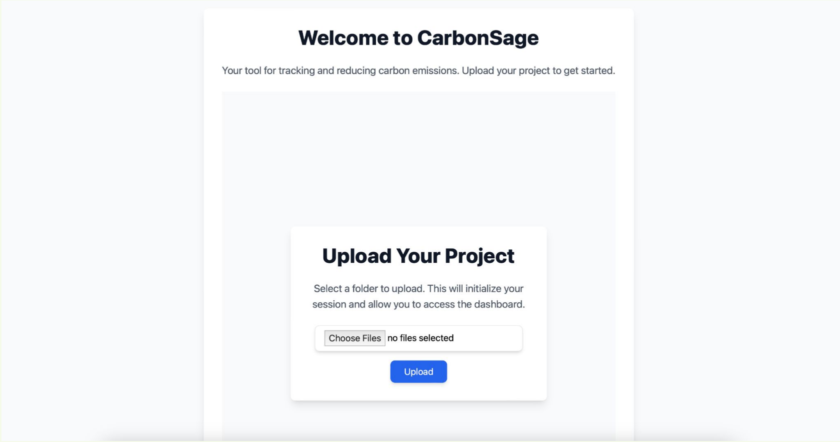
AI Frameworks

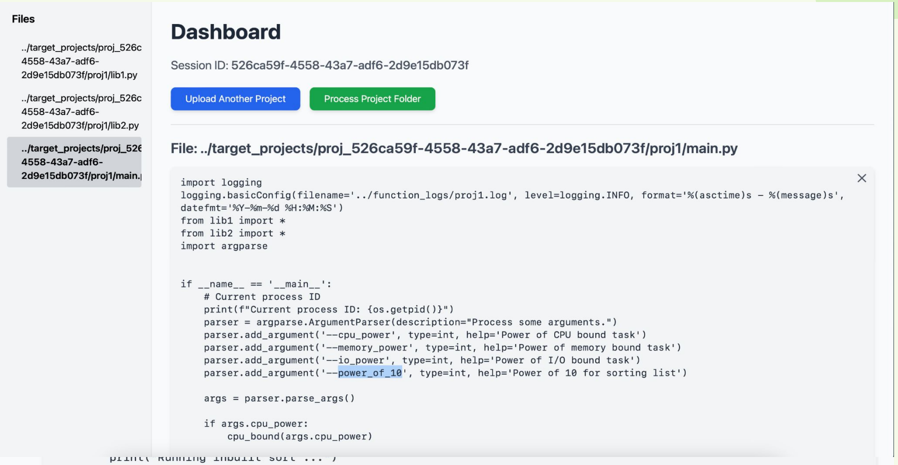
1. Python
2. LlamaIndex
3. codellama

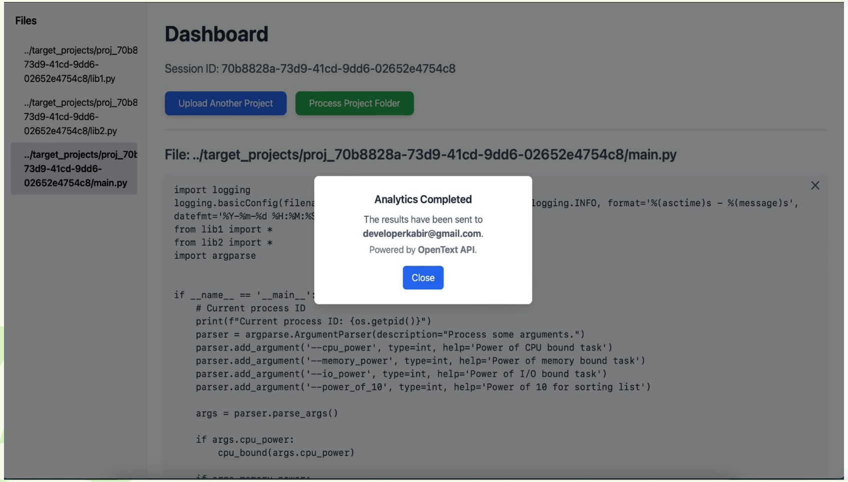
Web Frameworks and databases

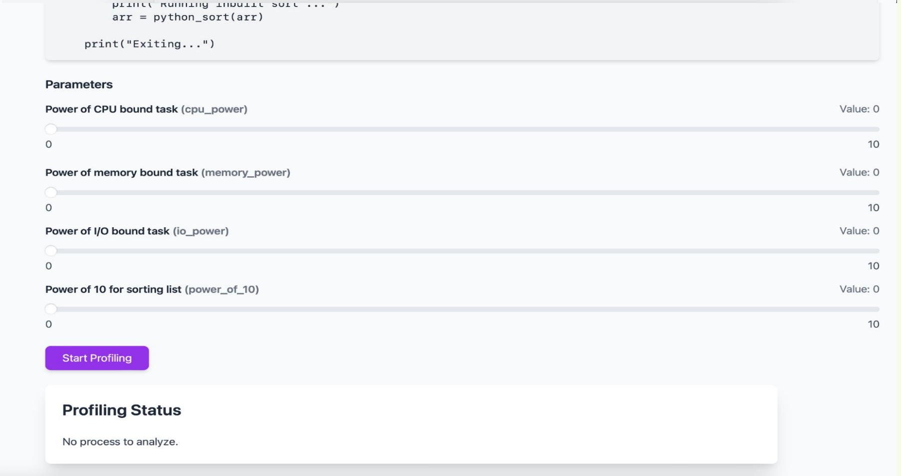
1. NextJS (For frontend and dashboards)
2. Flask (For backend API design)

USER EXPERIENCE









USER EXPERIENCE

Function Profile Map

FUNCTION ID	CPU %	PAGEINS	PFAULTS	PHYSICAL MEMORY (RSS MB)	VIRTUAL MEMORY (VMS MB)	ELAPSED TIME (SECS)	ENERGY CONSUMPTION (KWH)	CO2 EMISSIONS (KG)
4247	0.00%	165	2079	22.66 MB	400848.14 MB	1.00 secs	0.00 kWh	0.000029 kg
8550	0.00%	165	2079	22.66 MB	400848.14 MB	1.00 secs	0.00 kWh	0.000029 kg
9412	0.00%	165	2079	22.66 MB	400848.14 MB	1.00 secs	0.00 kWh	0.000029 kg
9996	0.00%	165	2079	22.66 MB	400848.14 MB	1.00 secs	0.00 kWh	0.000029 kg
13634	0.00%	165	2079	22.66 MB	400848.14 MB	1.00 secs	0.00 kWh	0.000029 kg
13723	0.00%	165	2079	22.66 MB	400848.14 MB	1.00 secs	0.00 kWh	0.000029 kg
14331	0.00%	165	2079	22.66 MB	400848.14 MB	1.00 secs	0.00 kWh	0.000029 kg
17049	6.40%	165	2071	22.53	400830.14	1.00	0.00 kWh	0.000030

Code from ../target_projects/proj_70b8828a-73d9-41cd-9dd6-02652e4754c8/lib1.py

Optimize Code

Original Code:

```
# Memory Heavy Task

def memory_bound(p):
    logging.info('')
    print("Starting memory bound task")
    x = []
    for i in range(10**p):
        x.append(i)
        if i % 10**(p-1) == 0:
            print(f"Memory bound task: {i}")
            time.sleep(1)
    print("Exiting memory bound task")
    logging.info('')
    return x

# I/O Heavy Task

def io_bound(p):
    logging.info('')
    # Generate a large file
    print("Starting I/O bound task")
    with open("large_file.txt", "w") as f:
        for i in range(10**p):
```

Optimized Code:

```
reduce the memory usage as we're not storing the entire range of
numbers in memory at once.

Here's the optimized code snippet:
```

```
from itertools import islice

def memory_bound(p):
    logging.info('')
    print("Starting memory bound task")
    x = []
    for i in islice(xrange(10**p), 1000):
        x.append(i)
        if i % 10**(p-1) == 0:
            print(f"Memory bound task: {i}")
            time.sleep(1)
    print("Exiting memory bound task")
    logging.info('')
    return x
```

```
In this optimized version, we're using the `xrange` function to
generate the range of numbers and the `islice` function to iterate
over the range of numbers in chunks of 100
```

Close

def generate_random_list(power_of_ten):

Analytics Dashboard

Data from **Aug 25, 2024 22:56:33** to **Aug 25, 2024 22:56:53**

Average Values For Function Parameters

METRIC	VALUE
AVG CPU PERCENT	0.875000
AVG PAGEINS	165.000000
AVG PFAULTS	2077.409091
AVG RSS	23730734.545455
AVG VMS	420315986478.545471
CO2 EMISSIONS KG	0.000102
ELAPSED TIME SECS	3.500000
ENERGY CONSUMPTION KWH	0.000205

pageins

From 25/8/2024, 10:56:33 PM to 25/8/2024, 10:56:53 PM

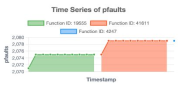
Min: 165.00

Max: 165.00

Average: 165.00

pfaults

From 25/8/2024, 10:56:33 PM to 25/8/2024, 10:56:53 PM



Min: 2071.00

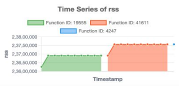
Max: 2079.00

Average: 2076.71

Rss

rss

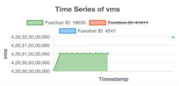
From 25/8/2024, 10:56:33 PM to 25/8/2024, 10:56:53 PM



Vms

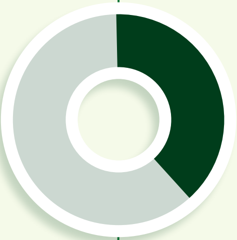
vms

From 25/8/2024, 10:56:33 PM to 25/8/2024, 10:56:53 PM



IMPACT

30%

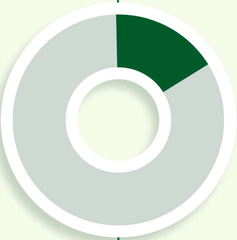


Cost Savings

Reducing energy consumption can save businesses millions annually (Energy Star).

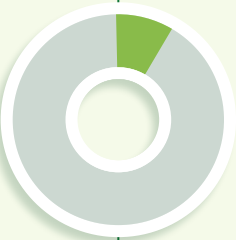
Environmental Benefits

Green technologies can reduce carbon emissions (McKinsey & Company).



15%

8%

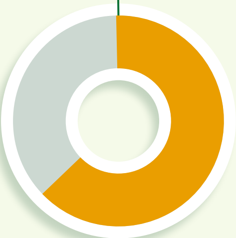


Data Driven Innovation

Leveraging data analytics can achieve higher operational efficiency (Deloitte).

Competitive Advantage

Global consumers are willing to pay more for sustainable brands (Nielsen).

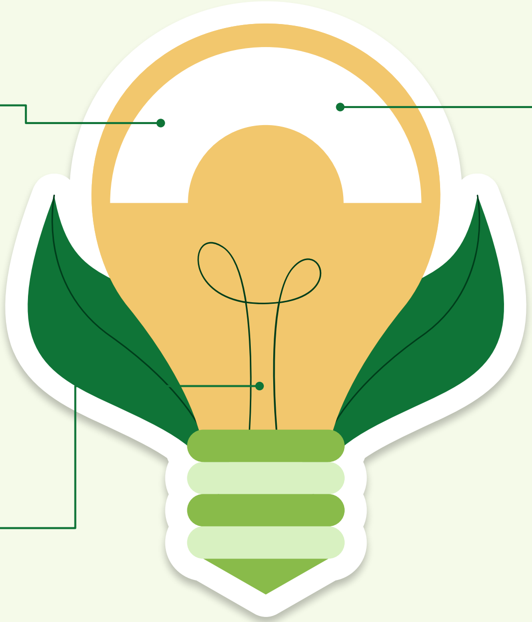


66%

FUTURE WORKS

Integration with Cloud Providers

Develop plugins and APIs for seamless integration with major cloud platforms (AWS, Azure, GCP).



Regulatory Compliance

Help users comply with global and local emission regulations. Provide regular updates on regulatory changes and their impact on emissions tracking.

Data Security and Privacy

Ensure compliance with data protection regulations (e.g., GDPR, CCPA).

Thank You