International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

CarbonWise: An AI-Driven Framework for Carbon Footprint Estimation and Code Quality

Dr.Swati Dronkar
Department of Computer Science and Engineering
Priyadarshini College of Engineering
Nagpur
swati.dhabarde@gmail.com

Sapana Choudhari
Department of Computer Science and Engineering
Priyadarshini College of Engineering
Nagpur
choudharisapana3@gmail.com

Shruti Wankhede
Department of Computer Science and Engineering
Priyadarshini College of Engineering
Nagpur
wankhedeshruti108@gmail.com

Divya Desai
Department of Computer Science and Engineering
Priyadarshini College of Engineering
Nagpur
divyadesai504@gmail.com

Mohit salodkar
liDepartment of Computer Science and Engineering
Priyadarshini College of Engineering
Nagpur
mohitsalodkar24@gmail.com

Prathamesh Shukla
Department of Computer Science and Engineering
Priyadarshini College of Engineering
Nagpur
prathameshshukla876@gmail.com

Abstract—The increasing adoption of cloud computing, distributed systems, continuous integration and continuous deployment (CI/CD), and large-scale software repositories has increased the computational infrastructure required throughout the software development lifecycle. Although software is intangible, its development, storage, testing, execution, deployment, and data-transfer activities depend on physical computing resources and consequently consume energy. CarbonWise is an artificial intelligence (AI)-driven Green Software Engineering platform designed to analyze GitHub repositories and provide repository-specific sustainability insights through a unified developer-oriented environment.

The proposed framework integrates GitHub repository analysis, software metrics, energy estimation, carbon-emission estimation, sustainability scoring, CI/CD analysis, dependency analysis, AI-assisted recommendations, interactive analytics, and automated reporting. CarbonWise retrieves repository metadata, programming-language information, repository size, dependencies, development dependencies, file structure, large files, configuration files, GitHub Actions workflows, workflow execution information, and CI runtime. These parameters are processed through a deterministic analysis engine to generate verified repository findings. Energy consumption is estimated using CI runtime, storage requirements, and network-related activity, while associated carbon dioxide equivalent (CO_2e) emissions are estimated using grid carbon intensity.

A composite sustainability score combines carbon, CI/CD, dependency, and codebase indicators. The resulting verified findings are then supplied to an AI recommendation module based on the Google Gemini API. Unlike generic AI-based recommendations, CarbonWise uses an evidence-first architecture in which the AI layer reasons over structured and pre-verified repository findings. The platform presents the resulting information through dashboards, analytics, and PDF/CSV reports. The proposed system provides a practical framework for integrating sustainability considerations into everyday software engineering and Green DevOps practices. However, the current energy and carbon values are model-based estimates and should not be interpreted as direct hardware-level measurements. Future work will focus on real-time energy monitoring, cloud-provider integration, region-specific carbon intensity, advanced static analysis, carbon-aware CI/CD, and empirical validation.

Keywords— Green Software Engineering, Software Sustainability, Carbon Footprint, GitHub Repository Analysis, Artificial Intelligence, Energy Efficiency, Green DevOps, Sustainable Software Development.

Abbreviations and Acronyms

Abbreviation	Full Form	Where You Use It
AI	Artificial Intelligence	AI-based sustainability
API	Application Programming Interface	Communication with GitHub and Gemini services
CI	Continuous Integration	Automated software
CD	Continuous Deployment	Automated deployment
CI/CD	Continuous Integration and	GitHub Actions and
CO_2	Carbon Dioxide	Environmental emission
CO_2e	Carbon Dioxide Equivalent	Estimated greenhouse-gas
CSV	Comma-Separated	Exporting sustainability
DB	Database	Storing users, repositories,
DevOps	Development and Operations	Software development
HTTP	Hypertext Transfer	Web communication

HTTPS	Hypertext Transfer	Secure web communicatio
JSON	JavaScript Object	Data exchange between
JWT	JSON Web Token	User authentication
OAuth	Open Authorization	GitHub authentication
PDF	Portable Document	Sustainability report
REST	Representational State	Backend API communicatio
UI	User Interface	CarbonWise dashboard and
UX	User Experience	Usability and user

I. INTRODUCTION

The software industry has experienced rapid growth due to the widespread adoption of cloud computing, web applications, mobile applications, artificial intelligence, distributed systems, and software-as-a-service platforms. Modern software development depends on a large amount of computational infrastructure. Source-code repositories are stored on servers, automated workflows execute builds and tests, dependencies are downloaded and processed, artifacts are transferred through networks, and applications are deployed on physical or virtual infrastructure.
Although software itself does not have a physical form, the infrastructure required to create, maintain, execute, and distribute software consumes electricity. Consequently, software engineering decisions can have indirect environmental implications. Repository size, build frequency, CI/CD runtime, dependency complexity, data transfer, storage requirements, and computational workload can influence resource consumption.

Traditional software engineering assessment generally emphasizes functionality, performance, maintainability, reliability, security, scalability, and code quality. Environmental sustainability is often treated as an independent concern rather than as an integrated software engineering attribute. As a result, developers may know whether a software project is functionally successful without having sufficient visibility into its potential energy and carbon implications.
CarbonWise is proposed to address this problem by connecting repository-level software analysis with sustainability assessment. The system uses GitHub as a primary source of repository evidence. It analyzes repository metadata, programming languages, dependencies, file structures, large files, configuration files, and CI/CD activity. The collected information is transformed into measurable findings and subsequently used for energy and carbon estimation.

The proposed framework also incorporates artificial intelligence to make sustainability findings easier to interpret. Rather than allowing an AI model to independently inspect a repository and generate potentially generic recommendations, CarbonWise follows an evidence-first architecture. The deterministic analysis layer first collects repository facts and generates structured findings. These findings are stored in the analysis database and subsequently provided to the AI recommendation layer.
The overall workflow is:

GitHub Repository → Repository Analysis → Energy Estimation → Carbon Estimation → Sustainability Score → AI Recommendations → Analytics and Reports

This workflow connects software artifacts that developers already use with sustainability-oriented analysis.

A. Research Contributions

The major contributions of CarbonWise are as follows:

1. A unified pipeline connecting GitHub repository analysis with energy and carbon estimation.
2. A repository-level sustainability scoring mechanism combining carbon, CI/CD, dependency, and codebase indicators.
3. An evidence-based AI recommendation mechanism that operates on deterministic and pre-verified repository findings.
4. Integration of GitHub Actions and CI/CD metrics into Green DevOps analysis.
5. An interactive developer-oriented dashboard for sustainability analytics.
6. Automated PDF and CSV sustainability reporting.
7. A modular web architecture capable of supporting future sustainability-analysis capabilities.

These contributions are consistent with the project specification, which defines CarbonWise as a unified developer-facing platform rather than a collection of independent sustainability tools.

II. RELATED WORK AND RESEARCH GAP

Software sustainability is a multidisciplinary problem involving software engineering, cloud infrastructure, energy efficiency, environmental computing, and resource optimization. Existing approaches generally focus on one or more specific aspects of sustainability.

A. Energy-Efficient Software

Energy-efficient software research investigates how software behavior can influence resource consumption. Optimization may involve reducing unnecessary computation, improving algorithms, reducing execution time, minimizing memory usage, or optimizing workload distribution.
Although such approaches are valuable, they often require developers to perform detailed analysis using specialized tools. A developer-oriented platform that directly connects repository characteristics with sustainability metrics can simplify this process.

B. Carbon-Aware Computing

Carbon-aware computing considers the carbon intensity of electricity used to execute workloads. The carbon intensity of electricity can vary according to geographical region, energy source, infrastructure, and time.
CarbonWise incorporates carbon intensity into its estimation model. However, the current system treats carbon intensity as a model input and does not claim to provide direct real-time measurement of infrastructure emissions.

C. Software Energy Metrics

Software energy analysis can measure or estimate the energy required to execute particular workloads. Such analysis can be performed at application, process, hardware, or infrastructure levels.
CarbonWise takes a repository-level perspective. Instead of focusing only on application runtime, it incorporates CI runtime, repository storage, and network-related activity into a conceptual energy model.

D. Code and Repository Analysis

Repository analysis tools commonly provide information about source files, dependencies, languages, configuration files, and code organization. These metrics can be useful for understanding the structure of software projects.
CarbonWise extends repository analysis by interpreting selected repository characteristics from a sustainability perspective.

E. AI-Assisted Software Recommendations

AI-based development tools can provide code suggestions, optimization recommendations, and debugging assistance. However, generic recommendations may not necessarily reflect the actual characteristics of a specific repository.
CarbonWise addresses this issue by introducing a deterministic analysis stage before AI processing.

F. Research Gap

Existing approaches address energy-efficient software, carbon-aware computing, software energy metrics, code analysis, CI/CD optimization, or AI-assisted development. However, these concerns are not necessarily integrated into one developer-facing framework.

The research gap addressed by CarbonWise is therefore the integration of:

- GitHub repository evidence,
- repository structural analysis,
- dependency analysis,
- CI/CD analysis,
- energy estimation,
- carbon estimation,
- sustainability scoring,
- evidence-based AI recommendations,
- analytics, and
- automated reporting.

The proposed framework combines these activities into a single workflow.

III. MOTIVATION

The motivation for CarbonWise originates from several challenges in practical software sustainability.

A. Integration Opacity

Developers generally work with repositories, source code, dependencies, CI/CD pipelines, and deployment systems. Sustainability information may exist separately from these development artifacts.
CarbonWise attempts to reduce this separation by analyzing the repository itself.

B. Interpretation Difficulty

Raw energy and carbon values may not be meaningful to developers without additional context. A numerical estimate alone may not explain why a repository has a particular sustainability profile.
CarbonWise therefore introduces a sustainability score, structured findings, dashboard visualization, and AI-assisted explanations.

C. Generic Sustainability Advice

General sustainability recommendations may not apply equally to all software projects. For example, a repository with large binary assets has different optimization opportunities from a repository dominated by CI workloads.
CarbonWise uses repository-specific evidence before generating recommendations.

D. CI/CD Resource Consumption

Modern development relies heavily on automated workflows. Repeated builds, failed workflows, testing pipelines, and deployment processes may require repeated computation.
CarbonWise therefore incorporates CI/CD information into sustainability analysis.

E. Need for Developer-Oriented Reporting

Sustainability information should be accessible to developers and project stakeholders. Interactive dashboards and automated reports provide a mechanism for communicating analysis results.

V. PROBLEM DOMAIN

The CarbonWise problem domain combines several technical areas:

- GitHub repository and metadata analysis.
- Software repository analytics.
- Dependency analysis.
- CI/CD analysis.
- Software energy estimation.
- Carbon-emission estimation.
- Sustainability scoring.
- AI-assisted recommendation generation.
- Secure web application development.
- Green Software Engineering.
- Green DevOps.

The intersection of these areas creates a software sustainability assessment problem in which repository-level evidence must be converted into understandable and actionable information.

V. PROBLEM DEFINITION

Let **R** represent a GitHub repository containing source code, dependencies, configuration files, repository storage, and CI/CD workflows.
The objective is to produce a repository-specific sustainability assessment:
S(R)
such that:

1. Relevant repository characteristics are collected.
2. Repository structure is analyzed.
3. Dependency information is analyzed.
4. CI/CD characteristics are analyzed.
5. Energy consumption is estimated.
6. Carbon emissions are estimated.
7. Sustainability indicators are combined into a score.
8. Evidence-based recommendations are generated.
9. Results are presented through a dashboard and report.

The system therefore attempts to transform:
Repository Data → Sustainability Evidence → Sustainability Metrics → Actionable Recommendations.

The current framework treats carbon and energy values as estimates based on defined assumptions rather than direct physical measurements.

VI. PROPOSED SYSTEM

CarbonWise is designed as a web-based software sustainability platform consisting of three major architectural layers.
A. Presentation Layer
The presentation layer provides interaction between users and the CarbonWise platform.
React.js is used to implement the component-based interface.
The presentation layer provides:

- Authentication interface.
- GitHub account connection.
- Repository management.
- Repository analysis initiation.
- Sustainability metric visualization.
- AI recommendation display.
- Dashboard.
- Analytics.
- Reports.
- Settings.

Vite supports development and production builds, while Tailwind CSS provides responsive styling. React Router handles client-side navigation and Chart.js supports visualization.

B. Application Layer

The application layer contains the core business logic.
Node.js is used as the server-side runtime and Express.js provides the REST API layer.

The application layer performs:

- User authentication.
- GitHub communication.
- Repository data collection.
- Repository structural analysis.
- Dependency analysis.
- CI/CD analysis.
- Energy estimation.
- Carbon estimation.
- Sustainability scoring.
- AI recommendation generation.
- Dashboard data aggregation.
- Report generation.
- Notification processing.

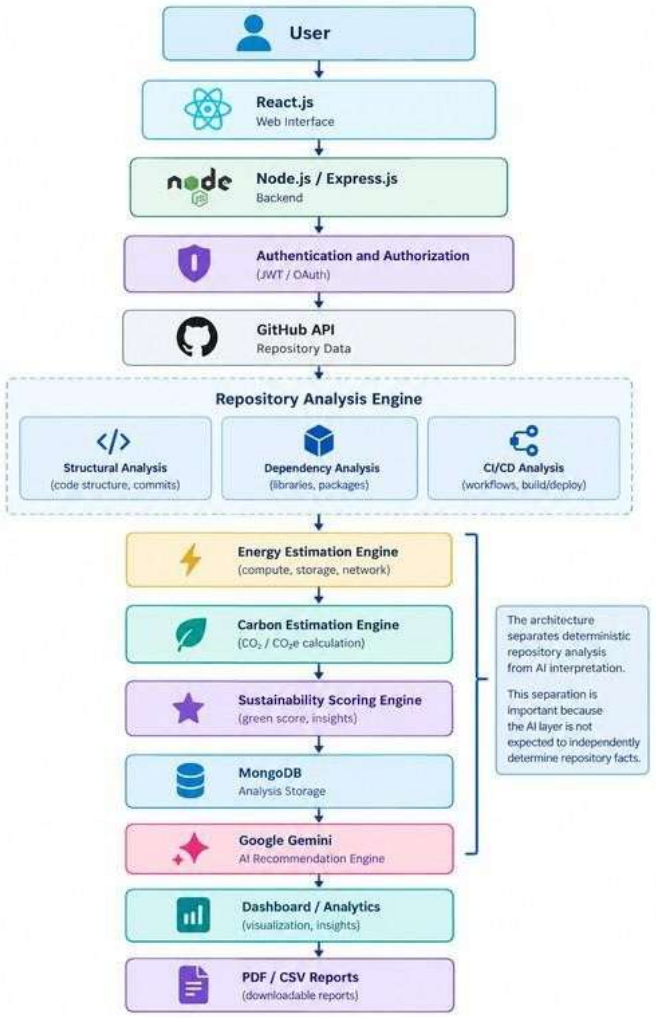
C. Data Layer

MongoDB provides persistent data storage.

Mongoose is used for schema definition and database operations.

The data layer maintains relationships between users, repositories, analyses, metrics, AI suggestions, and reports.

VII. SYSTEM ARCHITECTURE
The overall CarbonWise architecture can be represented as:



The architecture separates deterministic repository analysis from AI interpretation. This separation is important because the AI layer is not expected to independently determine repository facts.

VIII. TECHNOLOGY STACK

A. Frontend Technologies

1) React.js
React.js is used to construct the interactive user interface.
2) Vite
Vite supports rapid frontend development and production builds.
3) Tailwind CSS
Tailwind CSS provides utility-based responsive styling.
4) React Router
React Router manages navigation between application views.
5) Chart.js
Chart.js is used to visualize energy, carbon, and analytical information.
B. Backend Technologies
1) Node.js
Node.js provides the server-side execution environment.
2) Express.js
Express.js provides REST API functionality and backend routing.
3) MongoDB
MongoDB stores application and analysis data.
4) Mongoose
Mongoose provides structured interaction with MongoDB.

IX. GITHUB INTEGRATION

GitHub is a central component of CarbonWise because repository-specific sustainability analysis requires access to actual development artifacts.

The platform retrieves information such as:

- Repository metadata.
- Repository size.
- Programming languages.
- Dependencies.
- Development dependencies.
- File tree.
- Large files.
- Configuration files.
- GitHub Actions workflows.
- Workflow execution information.
- CI runtime.
- Successful workflow runs.
- Failed workflow runs.

This information provides the evidence used by the deterministic analysis engine.

For example, repository size contributes to storage-related estimation, CI runtime contributes to CI energy estimation, and workflow failure information can identify possible repeated computation.

X. GITHUB OAUTH AND AUTHENTICATION

CarbonWise uses GitHub OAuth to authorize access to GitHub resources.
The authentication sequence is:
User
→ Login/Connect GitHub
→ GitHub Authorization
→ Authorization Code
→ CarbonWise Backend
→ GitHub Access Token
→ GitHub API
JWT is used for CarbonWise application sessions.
The application also applies:

- Password hashing.
- Protected routes.
- Authentication middleware.
- User ownership checks.
- Resource-level authorization.

Repository and analysis records are associated with authenticated users.

This prevents users from directly accessing another user's repository analysis through application endpoints.

XII. DEPENDENCY ANALYSIS

Modern software applications rely on external packages for development, execution, testing, building, and deployment. CarbonWise analyzes supported package-management information.

For JavaScript and Node.js projects, package.json can provide:

- Production dependencies.
- Development dependencies.
- Package versions.
- Dependency structures.

Dependency analysis can identify potential optimization opportunities.

However, dependency existence does not automatically indicate that a package is unused. A package may be referenced indirectly or through project-specific configuration.

Therefore, CarbonWise avoids treating dependency presence alone as evidence for removal. Source-code evidence is required before making sufficiently strong dependency-removal recommendations.
This design reduces the possibility of generating unsupported AI recommendations.

XIII. CI/CD ANALYSIS

Continuous Integration and Continuous Deployment are important parts of modern software development.

A typical CI/CD workflow can perform:
- Source checkout.
- Dependency installation.
- Compilation.
- Testing.
- Linting.
- Security scanning.
- Artifact generation.
- Deployment.

Each execution consumes computational resources.
CarbonWise analyzes GitHub Actions information where available.

The system considers:
- Number of workflows.
- Workflow names.
- Number of analyzed executions.
- Runtime.
- Successful executions.
- Failed executions.
- Success rate.

Repeated workflow failures can cause repeated executions and therefore additional computational activity.
CI/CD analysis consequently contributes to both sustainability findings and the composite sustainability score.

composite sustainability score.

XIV. ENERGY ESTIMATION MODEL

CarbonWise uses a model-based approach to estimate energy consumption.
The model consists of three primary components.

A. CI Energy

CI energy is estimated from workflow runtime and average runner power:
$E_CI = T_CI \times P_runner$ (1) where:
- E_CI = estimated CI energy,
- T_CI = CI runtime,
- P_runner = assumed average runner power.

B. Storage Energy

Storage-related energy is modelled using repository size, storage energy factor, and time:
$E_storage = S_repo \times F_storage \times T$ (2)

where:
- E_storage = estimated storage energy,
- S_repo = repository size,
- F_storage = storage energy factor,
- T = relevant time factor.

C. Network Energy

Network-related energy is estimated using transferred data:

$E\ network = D\ transfer \times F\ network$ (3)
where:
- E network = estimated network energy,
- D transfer = amount of transferred data,
- F network = network energy factor.

D. Total Energy

The total estimated energy is:

E total = E CI + E storage + E network (4)
The three components provide a simplified framework for connecting repository characteristics with estimated resource consumption.

XV. CARBON EMISSION MODEL

After calculating estimated energy, CarbonWise applies a grid carbon-intensity factor.
The carbon estimation equation is:
$CO_2e = E\ total \times CI\ grid$ (5)
where:
- CO_2e = estimated carbon dioxide equivalent.
- E total = estimated total energy consumption.
- CI grid = grid carbon intensity.

The model allows carbon estimation to depend on electricity-related carbon intensity.
However, the result is an estimate rather than a direct measurement.
Actual emissions may differ because of:

- Geographic location.
- Electricity generation mix.
- Data-center infrastructure.
- Hardware efficiency.
- Workload characteristics.
- Time-dependent grid conditions.
- Cloud infrastructure configuration.

Consequently, the CarbonWise carbon value should be interpreted as a model output.

XVI. SUSTAINABILITY SCORING MODEL

A single carbon value does not fully represent software sustainability.
CarbonWise therefore combines several indicators.
The conceptual scoring model is:
$S = W_C\ C + W_CI\ CI + W_D\ D + W_B\ B$ (6)
where:
- S = sustainability score.
- C = carbon-related score.
- CI = CI/CD score.
- D = dependency score.
- B = codebase score.
- W_C, W_CI, W_D, W_B = respective weights. The score is mapped to a sustainability grade.

The purpose of the score is to provide a consistent indicator within CarbonWise. It should not be interpreted as a universal environmental standard.

This distinction is important because sustainability scoring methodologies can differ depending on their objectives and assumptions.

XVII. AI RECOMMENDATION ENGINE

The AI recommendation engine is a major component of CarbonWise. The system uses the Google Gemini API to generate recommendations. However, CarbonWise does not directly provide the entire repository to the AI model with the expectation that the model will independently discover all facts.

Instead, the architecture is:

GitHub Repository
↓
Deterministic Analysis
↓
Verified Findings
↓
MongoDB
↓
Gemini AI
↓
Repository-Specific Recommendations

This architecture has several benefits.

A. Evidence-Based Recommendations

Recommendations are connected to actual analysis findings.

B. Reduced Generic Advice

The model receives repository-specific evidence instead of producing recommendations solely from general sustainability concepts.

C. Traceability

Each recommendation can be associated with evidence.

D. Structured Output

Recommendations can include:

- Suggestion type.
- Title.
- Description.
- Evidence.
- Recommended action.
- Expected impact.
- Impact level.
- Confidence.

For example, if structural analysis identifies unusually large repository assets, the AI layer can interpret that finding and provide an optimization suggestion.

XVIII. DASHBOARD DESIGN

The CarbonWise dashboard provides a centralized view of repository sustainability.

The dashboard can contain:

A. Repository Summary

- Number of connected repositories.
- Recently analyzed repositories.
- Repository metadata.

B. Sustainability Summary

- Sustainability score.
- Sustainability grade.
- Estimated carbon emissions.
- Estimated energy consumption.

C. Energy Distribution

The dashboard can represent estimated energy across:

- CI.
- Storage.
- Network.

D. Analysis Findings

Repository-specific findings are displayed to the developer.

E. AI Recommendations

Recommendations can be organized according to impact or recommendation type.

dates.

E. AI Recommendations

Recommendations can be organized according to impact or recommendation type.

F. Notifications

The dashboard can provide notifications and analysis-related updates.

G. Quick Actions

Developers can initiate repository analysis, review recommendations, or generate reports.

XIX. ANALYTICS

Analytics convert raw analysis results into visual information.

Charts can represent:

- Energy distribution.
- Carbon estimates.
- Sustainability scores.
- Repository-level comparisons.
- CI/CD behavior.
- Analysis history.

Summary cards can provide high-level information while detailed charts provide deeper interpretation.

The objective is to make sustainability information understandable without requiring developers to manually inspect raw API responses or analysis records.

XX. REPORT GENERATION

CarbonWise supports automated PDF and CSV report generation.

A report can contain:

1. Repository information.
2. Repository analysis summary.
3. Sustainability score.
4. Sustainability grade.
5. Estimated energy.
6. Estimated carbon emissions.
7. Energy breakdown.
8. Structural findings.
9. Dependency findings.
10. CI/CD findings.
11. AI recommendations.
12. Expected impact and confidence information.
13.

PDF reports are suitable for human-readable documentation, while CSV reports can support further data processing and analysis.

XXI. DATA MODEL

MongoDB is used to maintain relationships between repository evidence and generated sustainability information.

The major entities are described below.

A. User

The User entity contains stakeholder identity, credentials, authentication information, and GitHub OAuth linkage.

B. Repository

The Repository entity stores GitHub repository information such as:

- Repository name.
- Repository metadata.
- Repository size.
- Languages.
- Dependencies.
- File structure.
- Owning user.

C. Analysis

The Analysis entity represents an execution of the repository analysis pipeline.

It connects a repository with:

- Structural findings.
- Dependency findings.
- CI/CD findings.
- Energy estimation.
- Carbon estimation.
- Sustainability score.

1. Login Page

Userscan sign in with GitHub OAuth or with an email/password combination. The screen introduces the

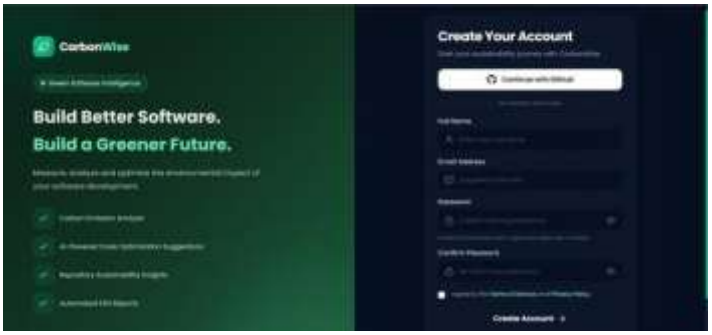
platform's value proposition — carbon emission analysis, AI-powered code optimization suggestions, repository sustainability insights, and automated ESG reports.



2. Sign-Up Page

Newusers cancreate anaccount via GitHub or by filling in their name, email, and a password meeting the

minimum complexity requirements, after agreeing to the Terms of Service and Privacy Policy.



3. Terms of Service

TheTermsofService clarifythat CarbonWise is a final-year academic project demonstrating sustainable

software engineering concepts, and define acceptable use of the platform and its GitHub account connection.



4. Dashboard

TheGreen Software Dashboard greets the user and summarizes key metrics at a glance: number of connected repositories, average CO2 emission per analysis, average energy consumption, and an overall sustainability score, alongside CO2-emission-by-repository and energy-breakdown charts.



5. Repository Analysis

TheRepositoriespage listsallconnected GitHub repositories with per-repository sustainability percentage,
carbon score, primary language, and last-analyzed date, along with options to analyze a repository or view its report.



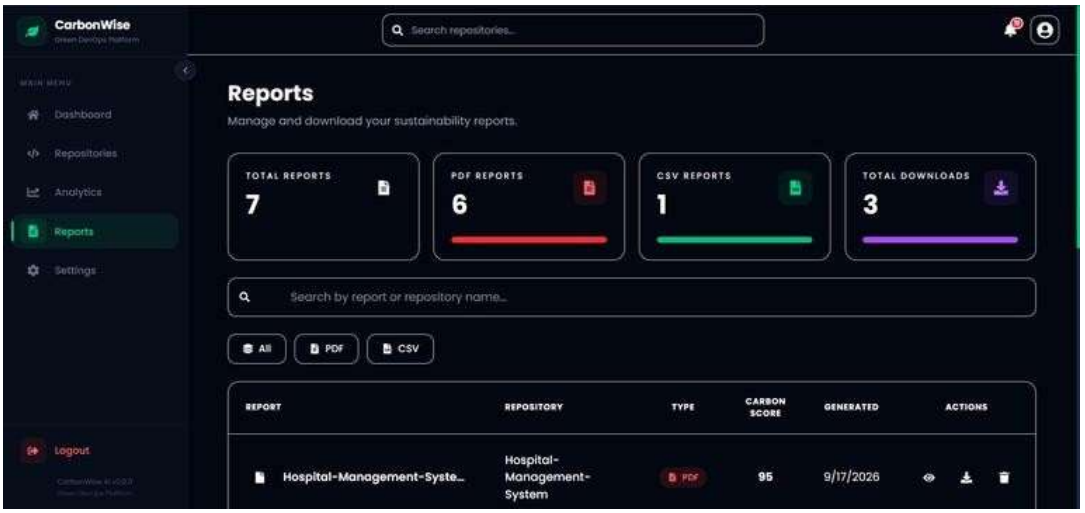
6. Analytics

TheAnalytics page aggregates carbon, energy, and sustainability imp across all analyzed repositories,
showing total CO2 emission, total energy consumption, and a breakdown of emissions by repository and by source (CI/CD compute repository storage, dependencies/network).



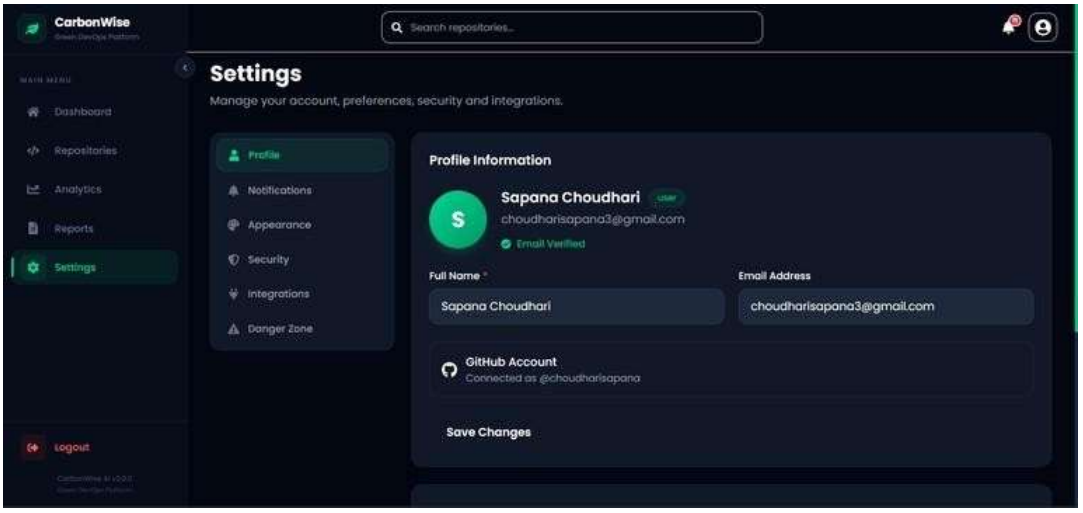
7. Reports

TheReportspage manages generated sustainability reports, allowing users to filter by PDF or CSV type, and to view, download, or delete individual reports along with their carbon score.



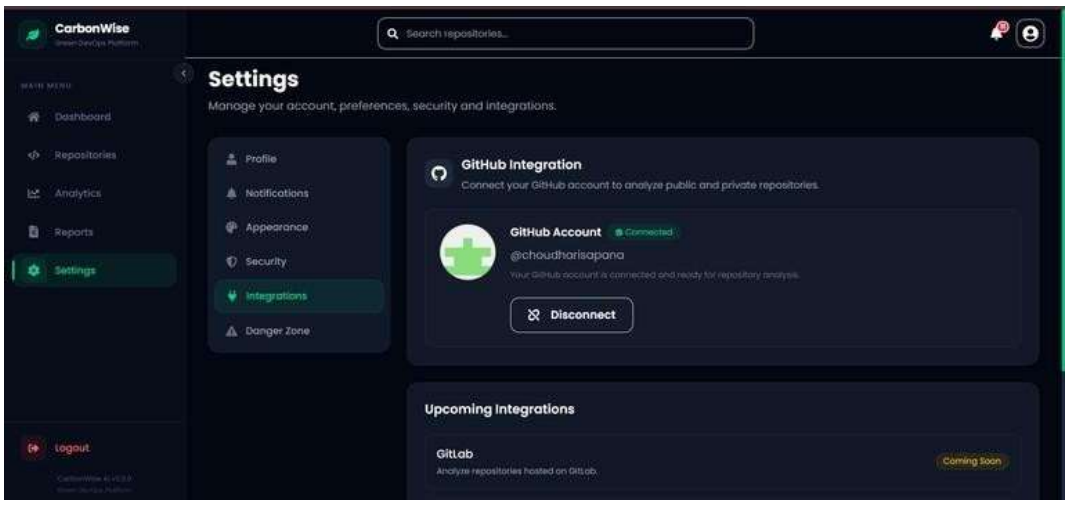
8. Settings — Profile

The Profiletab under Settings displays the user's account details (name, email, verification status) and shows the connected GitHub account used for repository analysis.



9. Settings — Integrations

The Integrationstabshows the status ofthe connected GitHub account and lists upcoming integrations, such as GitLab support, planned for future releases.



D. Sustainability Metrics
This entity stores:
- Estimated energy.
- Estimated carbon emissions.
- Sustainability score.
- Sustainability grade.

E. AI Suggestion
An AI Suggestion stores:
- Type.
- Title.
- Description.
- Supporting evidence.
- Recommended action.
- Expected impact.
- Impact level.
- Confidence.

F. Report
The Report entity stores generated PDF/CSV report information associated with a particular analysis.
The data model is designed to preserve the connection between repository facts, analysis results, sustainability metrics, and AI recommendations.

XXII. COMPLETE SYSTEM WORKFLOW
The complete CarbonWise process consists of the following stages.
Stage 1: Authentication
The user logs into CarbonWise.
Stage 2: GitHub Connection
The user authorizes GitHub access through OAuth.
Stage 3: Repository Selection
The user selects a repository for sustainability analysis.
Stage 4: Data Collection
CarbonWise retrieves repository information through GitHub APIs.
Stage 5: Structural Analysis
The system analyzes files, sizes, extensions, configuration files, and repository structure.
Stage 6: Dependency Analysis
Dependencies and development dependencies are analyzed.
Stage 7: CI/CD Analysis
GitHub Actions workflow information is analyzed where available.
Stage 8: Energy Estimation
CI, storage, and network energy are estimated.
Stage 9: Carbon Estimation
Estimated energy is converted into estimated CO_2e using carbon intensity.
Stage 10: Sustainability Scoring
Carbon, CI/CD, dependency, and codebase indicators are combined.
Stage 11: Data Storage
Analysis results are stored in MongoDB.
Stage 12: AI Recommendation
Verified findings are supplied to Gemini.
Stage 13: Dashboard
Results are displayed through interactive dashboards and analytics.
Stage 14: Reporting
The developer can generate PDF and CSV reports.

XXIII. SECURITY AND PRIVACY CONSIDERATIONS
CarbonWise interacts with repository and user information; therefore, application security is important.
A. Password Protection
Passwords are hashed using bcrypt rather than being stored in plain text.
B. Session Authentication
JWT provides application-level session authentication.
C. GitHub Authorization
GitHub OAuth is used instead of directly requesting users to provide GitHub passwords.
D. Protected Routes
Backend routes requiring authentication are protected by authorization middleware.
E. User Ownership
Repository and analysis records are associated with their owning users.
F. Evidence Traceability
AI recommendations are connected to stored analysis findings, improving the traceability of generated recommendations.

XXIV. IMPLEMENTATION CONSIDERATIONS
The implementation follows a modular architecture.
The frontend communicates with the backend through APIs. The backend communicates with GitHub to retrieve repository information and stores analysis data in MongoDB.
A typical analysis request follows:
Frontend → Backend API → GitHub API → Analysis Engine → MongoDB → AI Engine → Backend → Frontend
The modular approach allows individual components to be extended without redesigning the complete platform.
For example, additional programming-language analysis can be incorporated into the repository analysis engine while maintaining the same dashboard and reporting interfaces.
Similarly, a future hardware energy-monitoring component could provide measured values to the energy estimation layer.

XXV. COMPARISON WITH EXISTING APPROACHES

CarbonWise is designed around the integration of several concerns that are often handled independently.

Feature	Conventional Separate Tools	CarbonWise
Repository Analysis	Available in repository/code	**Integrated**
Dependency Analysis	Available separately	**Integrated**
CI/CD Analysis	Available separately	**Integrated**
Carbon Estimation	Specialized sustainability tools	**Integrated**
Sustainability Scoring	Depends on tool	**Integrated**
AI Recommendations	General or code-focused	**Evidence-based**
Dashboard	Tool-dependent	**Integrated**
PDF/CSV Sustainability	Tool-dependent	**Integrated**
Repository-Specific Findings	Varies	**Core design principle**

The distinguishing characteristic of CarbonWise is not simply the presence of individual features but their integration into a single repository-level sustainability workflow.

XXVI. INNOVATIVE ASPECTS

The proposed system contains several design innovations.

A. **Unified Repository Sustainability Pipeline**
Multiple sustainability indicators are analyzed within one framework.

B. **Evidence-First AI**
AI recommendations are generated only after deterministic analysis produces structured findings.

C. **Repository-Specific Recommendations**
Recommendations are connected to the analyzed repository.

D. **Green DevOps Integration**
CI/CD workflow behavior contributes to sustainability analysis.

E. **Sustainability Reporting**
Analysis results can be communicated through automated reports.

F. **Traceable Data Model**
The database maintains relationships between repositories, analysis results, sustainability metrics, and recommendations.

XXVII. RESULTS AND DISCUSSION

CarbonWise is presented as a proposed and implemented software platform architecture. The supplied project material does not contain a controlled empirical evaluation involving measured energy consumption, benchmark datasets, RMSE, MAE, transaction-level measurements, or statistically validated comparisons.

Therefore, numerical performance claims are intentionally not presented.

The design-level outcomes of the system are:

1. Repository-specific analysis.
2. Energy estimation.
3. Carbon estimation.
4. Sustainability scoring.
5. Dependency analysis.
6. CI/CD analysis.
7. Evidence-based AI recommendations.
8. Interactive dashboard.
9. Analytics.
10. Automated PDF/CSV reporting.

The evidence-first AI design is particularly important. A conventional generative AI system may generate recommendations based on general knowledge without having sufficient information about a specific repository. CarbonWise first performs deterministic analysis and then supplies structured findings to the AI module.

This approach does not eliminate AI uncertainty, but it establishes a stronger connection between recommendations and repository evidence.

XXVIII. ADVANTAGES

The major advantages of CarbonWise are summarized below.

1. **Unified Analysis**
Multiple sustainability dimensions are analyzed within one platform.

2. **Repository-Specific Evidence**
The system uses actual GitHub repository characteristics.

3. **AI-Assisted Interpretation**
Complex findings can be converted into developer-oriented recommendations.

4. **Green DevOps Support**
CI/CD behavior is included in sustainability analysis.

5. **Automated Reporting**
Reports can be generated without manually compiling analysis information.

6. **Developer-Friendly Visualization**
Dashboards and charts simplify interpretation.

7. **Modular Architecture**
The architecture allows future components to be added.

8. **Traceability**
The data model links repository information with findings and recommendations.

XXIX. LIMITATIONS

Despite its capabilities, CarbonWise has several limitations.

A. **Model-Based Estimation**
Energy and carbon values are estimates rather than direct hardware-level measurements.

B. **GitHub Dependency**
The analysis depends on the availability and completeness of information provided by GitHub APIs.

C. **Ecosystem Differences**
Different programming languages and package-management ecosystems expose different levels of dependency information.

D. **AI Dependency**
Recommendation quality depends on the quality and completeness of deterministic findings.

E. **Carbon-Intensity Variability**
Carbon intensity varies by location, infrastructure, workload, and time.

F. **Lack of Empirical Validation**
The current project material does not contain a controlled empirical evaluation of estimation accuracy.

These limitations should be clearly acknowledged in any research-paper submission.

XXX. FUTURE WORK

Future development can significantly extend CarbonWise.

A. **Real-Time Energy Monitoring**
Hardware or infrastructure monitoring can provide measured energy values.

B. **Cloud Provider Integration**
Cloud platforms can provide infrastructure-level resource and sustainability information.

C. **Region-Specific Carbon Intensity**
Real-time regional carbon-intensity information can improve carbon estimation.

D. **Additional Programming Languages**
The repository analyzer can be expanded to support additional ecosystems.

E. **Advanced Static Analysis**
Advanced source-code analysis can identify computationally expensive or inefficient code patterns.

F. **Dependency Usage Verification**
Source-code analysis can determine whether dependencies are actually used.

G. **JavaScript Bundle Analysis**
Frontend bundle size can be incorporated into sustainability analysis.

H. **Container Sustainability**
Docker images and container workloads can be analyzed.

I. **Kubernetes Sustainability**
Kubernetes resource utilization and workload configuration can be incorporated.

J. **Carbon-Aware CI/CD**
CI workloads can potentially be scheduled according to carbon-intensity conditions.

K. **Sustainability Gates**
Sustainability checks can be incorporated into CI/CD pipelines.

L. **Pull Request Analysis**
CarbonWise can analyze proposed repository changes before merging.

M. **Historical Trends**
Developers can track sustainability scores over time.

N. **Additional Git Platforms**
Future versions can support additional Git hosting platforms.

XXXI. CONCLUSION

CarbonWise presents a unified AI-driven framework for integrating software sustainability assessment into the software development lifecycle. The proposed platform combines GitHub repository analysis, software metrics, energy estimation, carbon-emission estimation, sustainability scoring, CI/CD analysis, dependency analysis, AI-assisted recommendations, dashboards, analytics, and automated reporting.

The framework begins with repository-specific evidence obtained through GitHub APIs. Structural, dependency, and CI/CD characteristics are analyzed deterministically. These findings are subsequently used to estimate energy consumption and associated carbon emissions. A composite sustainability score combines carbon, CI/CD, dependency, and codebase indicators.

A central feature of the architecture is the separation between deterministic analysis and generative AI. CarbonWise first verifies repository characteristics and stores structured findings before passing them to the AI recommendation layer. This design is intended to reduce generic recommendations and provide guidance that is grounded in repository-specific evidence.

The system also provides developer-oriented dashboards and automated reports, allowing sustainability information to become part of normal software development activities.

However, the current framework provides model-based estimates rather than direct physical measurements. The carbon and energy values therefore require appropriate interpretation, and future empirical validation is necessary to determine estimation accuracy and practical environmental impact.

Future development involving real-time energy monitoring, cloud-provider data, regional carbon-intensity information, advanced static analysis, dependency verification, container analysis, Kubernetes integration, carbon-aware CI/CD, and historical sustainability tracking can further strengthen the framework.

Overall, CarbonWise provides a structured approach for connecting software engineering artifacts with sustainability analysis and demonstrates how repository analytics and evidence-based artificial intelligence can support Green Software Engineering and Green DevOps practices.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to their project guide, faculty members, and the Department of Computer Science and Engineering for their guidance, encouragement, and support throughout the development of the CarbonWise project.

The authors also acknowledge the technical resources and development platforms that supported the design and implementation of the proposed system.

REFERENCES

[1] O. Danushi, S. Forti, and J. Soldani, “Carbon-efficient software design and development: A systematic literature review, ” ACM Computing Surveys, 2025.

[2] S. M. K. Murthy, U. Airsang, K. Rajamani, and B. M. Schuller, “From code to carbon: Being energy efficient while coding, ” in Proc. Int. Conf. Sustainable & Resilient Futures (ICSRF), 2025.

[3] Verdecchia et al. discuss how software engineering practices can be considered from an environmental sustainability perspective, particularly in relation to the energy and resource demands of software systems.

[4] Castor presents an introductory approach for understanding and assessing the energy consumption associated with software execution.

[5] S. Podder, H. Date, and S. Murthy Green prompt engineering for sustainable generative AI, Environmental Science and Ecotechnology, vol. 30, 2026.

[6] J. Rashid, T. Mahmood, and M. W. Nisar A study on software metrics and its impact on software quality, Technical Journal, University of Engineering and Technology (UET) Taxila, vol. 24, no. 1, 2019.

[7] R. Baghdadi, M. Merouani, M. H. Leghettas, K. Benatchba, S. Amarasinghe, and T. Arbaoui, A deep learningbased cost model for automatic code optimization, in Proc. 4th MLSys Conference, 2021.

[7] R. Baghdadi, M. Merouani, M. H. Leghettas, K. Benatchba, S. Amarasinghe, and T. Arbaoui, A deep learningbased cost model for automatic code optimization, in Proc. 4th MLSys Conference, 2021.

[8] S. Ergasheva, I. Khomyakov, A. Kruglov, and G. Scucci,Metrics of energy consumption in software systems: A systematic literature review, IOP Conference Series: Earth and Environmental Science, 2020.

[9] Y. G. Kim, U. Gupta, A. McCrabb, Y. Son, V. Bertacco, D. Brooks, and C.-J. Wu, GreenScale: Carbon-aware systems for edge computing, arXiv preprint arXiv:2304.00404, 2023.

[10] A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, S. Talukdar, E. Mullen, K. Smith, M. Cottman, and W. Cirne, Carbon-aware computing for datacenters.

[11] S. S. Shenoy and R. Eeratta, Green software development model: An approach towards sustainable software development in Proc. 2011 Annual IEEE India Conference (INDICON), Hyderabad, India. [12]Sehgal, R., Mehrotra, D., Nagpal, R., & Sharma, R. (2022). Green software: Refactoring approach. Journal of King Saud University – Computer and Information Sciences, 34(10), 4635–4643.

[13] Gupta, U., Kim, Y. G., Lee, S., Tse, J., Lee, Wei, G. Y., Brooks, D., & Wu, C. J. (2020). Chasing carbon: The elusive environmental footprint of computing.

[14] Kern, E., Hilty, L. M., Guldner, A., Maksimov, Y. V., Filler, A., Groger, J., & Naumann, S. (2018). Sustainable software products – Towards assessment criteria for resource and energy efficiency. Future Generation Computer Systems, 86, 199–210.[15]Pathania, P., Bamby, N., Mehra, R., Sikand, S., Sharma, V. S., Kaulgud, V., Podder, S., & Burden, A. P. (2025). Calculating software’s energy use and carbon emissions: A survey of the state of art, challenges, and the way ahead. [16]Amsel, N., Ibrahim, Z., Malik, A., & Tomlinson, B. (2011). Toward sustainable software engineering (NIER track). In Proceedings of the 33rd International Conference on Software Engineering (ICSE ’11) (pp. 976–979). ACM.

[17] Capra, E., Francalanci, C., & Slaughter, S. A. (2012). Is software “green”? Application development environments and energy efficiency in open source applications. Information and Software Technology, 54(1), 60–71.

[18] Ournani, Z., Rust, P., Rouvoy, R., & Penhoat, J. (2020). On reducing the energy consumption of software: From hurdles to requirements. In Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM ’20) (pp. 1–12). ACM.

[19] Hindle, A. (2016). Green software engineering: The curse of methodology. Proceedings of the 2016 IEEE/ACM International Conference on Software Engineering Companion (ICSE-C), 507– 510. IEEE.

[20] Guldner, A., Bender, R., Calero, C., Fernando, G. S., Funke, M., Gröger, J., Hilty, L. M., Hörnschemeyer, J., Hoffmann, G.-D., Junger, D., Kennes, T., Kreten, S., Lago, P., Mai, F., Malavolta, I., Murach, J., Obergöker, K., Schmidt, B., Tarara, A., Naumann, S. (2024). Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components – Green Software Measurement Model (GSMM). Future Generation Computer Systems, 155, 402–418.

[20]Guldner, A., Bender, R., Calero, C., Fernando, G. S., Funke, M., Gröger, J., Hilty, L. M., Hörnschemeyer, J., Hoffmann, G.-D., Junger, D., Kennes, T., Kreten, S., Lago, P., Mai, F., Malavolta, I., Murach, J., Obergöker, K., Schmidt, B., Tarara, A., Naumann, S. (2024). Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components – Green Software Measurement Model (GSMM). Future Generation Computer Systems, 155, 402–418.