National College of Ireland

Optimization of Static Code Analysis for Carbon Footprint Reduction in DevOps Pipeline

MSc Research Project
Cloud Computing

Srishti .
Student ID: 23189053

School of Computing
National College of Ireland

Supervisor: Sudarshan Deshmukh

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Srishti .
Student ID:	23189053
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Sudarshan Deshmukh
Submission Due Date:	12/12/2024
Project Title:	Optimization of Static Code Analysis for Carbon Footprint Reduction in DevOps Pipeline
Word Count:	7273
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	*Srishti*
Date:	29th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimization of Static Code Analysis for Carbon Footprint Reduction in DevOps Pipeline

Srishti .
23189053

Abstract

Over the past few years, Cloud computing has attracted significant attention. Many organizations have adopted this way of computing due to its on-demand services, cost savings and scalability which led to rapid expansion of Data Centers consuming huge amount of energy. This increased energy consumption has led to increased carbon emissions, which is significant threat to the environment. The research addresses the need of incorporating green cloud computing practices by identifying and minimizing the carbon footprint associated with DevOps activities widely used in cloud computing. The focus is on optimizing the Continuous Integration/Continuous Delivery (CI/CD) pipeline, specifically static code analysis stage as it is the primary contributor of generating carbon footprint. The static code analysis stage helps in checking any static errors, vulnerabilities or security issues in the workspace. We have proposed the optimization of Pylint, which is a static code analysis tool for reducing its impact on environment, without sacrificing performance. The results show that our optimization helped lowering the carbon footprint associated with static code analysis, contributing to Green DevOps. The proposal aims to benefit organizations by reducing their environmental impact and support corporate social responsibility. However, further research can be done for optimizing other stages of the pipeline.

1 Introduction

In the last few years, a lot of organizations have adopted cloud computing technologies. Cloud computing offers various benefits, the primary benefit being that services can be used anytime from anywhere and organizations will only be charged for the services used. After adopting cloud computing, organization do not have to invest on the physical setup that helps achieving cost savings. As cloud computing offers various benefits, the demand is increasing, causing rapid expansion of Data Centers. These data centers are known to consume high amounts of energy especially for cooling them. A recent Greenpeace research claims that carbon footprint generated by (Information Technology) IT activities accounts for around 2% of global CO2 emissions. The increasing carbon footprint gave rise to Green Computing. Green Computing includes use of eco-friendly practices so environmental impact of carbon footprint can be reduced Rawas et al. (2015). These Green computing practices have gained a lot of attention in past few years. Many organizations are getting aware of the impact of IT on environment and working on implementing green computing practice to reduce this impact. Green Computing can be further classified as a Green Cloud, Green IoT, Green IoT, and more. The research focuses on Green

DevOps area. DevOps involve processes that combines software development and operations for quicker deployment of applications. Green DevOps refers to implementing sustainable practices in DevOps processes. These practices are targeted to minimize the energy consumption and optimizing the resource utilization in order to decrease the carbon footprint generated by DevOps practices. DevOps pipelines i.e. CI/CD pipelines are major contributors for generating Carbon Footprint. The pipeline is responsible for releasing software to market that includes building, testing and deploying of application. The energy consumption while building, testing and deployment processes is responsible for generation of Carbon Footprint. As the demand for DevOps practices is increasing, it is important to work on reduction of carbon footprint generated by these activities. Various researches have been performed in green computing field that proposed strategies for reducing this carbon footprint. The strategies include infrastructure optimization, dynamically scaling of resources and Virtual Machine Migration.

Many researches have been conducted for reduction of carbon footprint. The studies mainly worked on optimizing the energy consumed by data centers via creation of energy-efficient algorithms. According to Shu et al. (2014) and Hussein et al. (2014), optimization of server response time and improvement in power efficiency can be achieved by allocation of resources and managing the power. However, no studies have been conducted for optimization of CI/CD pipeline. This is the identified gap from the review of related work. This research focuses on filling this gap and optimizing the CI/CD pipeline.

Table 1: Objectives

Objectives	**Explanation**
Analysis of Related Work	Review of existing studies in green computing, green cloud computing and green DevOps
Detecting Carbon Footprint	Detection of Carbon gas Emissions generated by cloud computing practices
Reducing Environmental impact of Cloud Computing	CO2 Emissions reduction for decreasing the Environmental impact
Optimizing Static Code Analysis	Pylint optimization for reducing the Carbon footprint

Static code analysis is a useful tool for developers for early detection of bugs. It computes the program without the need of running the program. The analysis of code offers multiple benefits including Early code detection, improved code quality, enhanced security, and more. Currently, there are various tools in the market that can be used for doing static code analysis of the application code. The most widely tools used are Pylint, SonarCloud, SonarQube, etc. These tools helps checking any syntax errors, vulnerabilities, security issues, and potential bugs. Static code analysis is often integrated in CI/CD pipeline for ensuring the code quality before deploying it. This helps in early detection of bugs which can be fixed before deploying the application to production. But these benefits comes with a cost of increased computational overhead. When implemented in DevOps pipeline, these tools consume high amounts of energy and hence contribute to carbon footprint production. The proposed methodology works optimization of Pylint

tool and decreasing the carbon footprint of overall CI/CD pipeline. Pylint is a most widely used tool often integrated in CI/CD pipeline for static code analysis and detecting errors in Python code. It analyses all files for any syntax errors, unused import, undefined variables, etc. Optimizing pylint can contribute to overall reduction in carbon footprint generated by CI/CD pipeline.

Table 1 includes the objectives of the paper. Main objectives being reducing the carbon footprint associated with DevOps pipeline and optimizing static code analysis. A novel technique is introduced in the paper for reducing the carbon emissions generated by DevOps pipelines.

Research Question: The research questions proposed for addressing these challenges are as follows:

RQ: *How does the integration of real-time carbon intensity data using the CO2Signal API and integrating GitHUB Actions workflow with Pylint can detect and reduce carbon footprint generated by CI/CD pipelines in DevOps processes? How can organizations benefit from monitoring and reducing their carbon footprint?*

Sub-RQ: *To what extent the proposed approach can reduce the carbon footprint compared to traditional approaches?*

The proposed research methodology aims to detect and reduce the carbon footprint generated by DevOps activities. CO2Signal API has been used for fetching the Carbon Intensity data of specific region. The pipeline is continuously monitored for resource utilization metrics. By making use of the Carbon Intensity data and the utilization metrics carbon footprint can be detected. The optimization of Pylint can be done by integrating GitHub actions with it so that Pylint analysis in only done on files that have been updated in previous run rather than linting all files. This solution will help reducing the carbon footprint effectively.

The research solution contributed towards Green DevOps by introducing a solution to reduce carbon footprint in CI/CD pipelines. The research also integrates a C02Signal API to calculate the total carbon Intensity. Lambda function is also integrated for calculation of carbon footprint by using the metrics obtained from cloudwatch. Also, the research contributed towards optimizing static code analysis stage using Pylint and usage of less cloud resources by decreasing the total execution time taken by the CI/CD pipeline.

The proposal continues the following structure: Section 2 of this study discusses the Related work by reviewing existing works and identifying the research gap. The purpose of this section is to give a summary of the existing works in green computing fields. Section 3 discusses the proposed methodology, which includes the proposed solution and tools. Section 4 discusses the Design Specifications and Architecture of the proposed solution. Section 5 covers the implementation of the proposed solution. Section 6 demonstrates the experiments conducted during the research followed by discussion of results obtained from experiments. The proposed research is then concluded highlighting its limitation and future work.

2 Related Work

This section outlines the literature review performed in the Static Code Analysis, Green Computing, Green Cloud Computing and Green DevOps areas. The section is organized into four sub-sections that gives a comprehensive overview of existing studies. The initial section provides insights on Static code analysis in DevOps pipeline. The second section highlights the strategies & various initiatives in the Green computing field which is necessary to comprehend the importance of Green Computing. Third section provides an overview on various techniques and technologies implemented in previous studies for reduction of carbon emissions and contribute towards Green Cloud Computing. Last section explores the contributions done in Green DevOps area. A detailed analysis of existing techniques is conducted, focusing on reducing carbon emissions from devops practices.

2.1 Importance of Static Code Analysis

The usefulness of the static code analysis stage in the software development lifecycle is demonstrated in the paper Nikolić et al. (2021) by comparing three different static code analysis techniques namely CppCheck, FindBugs and SonarQube. The techniques are compared based on four set of features out of which SonarQube performs the best with 69.61%. The study illustrates the importance of static code analysis stage but fails to provide a concrete evidence using experiments on real world applications.

Importance of using static code analysis during the software development process has been discussed in the paper Yeboah and Popoola (2023) and Kannan et al. (2022) . The paper discusses the widely used static code analysis tools such as MLSmellHound, SonarQube, PMD, Checkstyle and FindBugs and evaluate the performance of the analysis tools by detecting software defects. Experiments conducted using multiple datasets and metrics on these static code analysis tools found that the SonarQube outperforms other static analysis tools in terms of efficiency and reliability. The study is useful to the field of research in establishing the importance of static code analysis stage in software development lifecycle.

Integration of Static Application Security Testing (SAST) tool, and automating the SAST tool output into the developer issue tracking software has been proposed by Wadhams et al. (2024), Stanciu and Ciocârlie (2023) and Marandi et al. (2023) . This study is aimed towards automating the software development processes along with emphasizing the significance of static code analysis which detects code smells, vulnerability and security issues. The author demonstrated an experiment using SonarQube in a GitLab environment but fails to illustrate the approach using other tools or workspaces.

This section discussed the importance of static code analysis in DevOps pipeline but at the cost of increased computational overhead. This is a clear gap where the proposed research works upon.

2.2 Shift towards Green Computing

With the increasing demand of cloud technologies, a significant rise in data centers has been observed. These data centers consumes high amounts of energy, leading to increased

carbon footprint production. The study highlights how the rapid extension of data centers amplifies energy consumption and increasing carbon emissions Kinkar et al. (2022). The paper emphasizes on importance of adopting energy efficient techniques known as 'Green Computing' for minimizing these environmental impacts.

Potential of green computing activities to address the escalating environmental impact of Information and Communication Technology (ICT) has been discussed in the paper Patel et al. (2024). Green computing focuses on minimizing the carbon footprint of by reducing energy consumption and pollution and promotes eco-friendly strategies. The paper examines the potency of of green computing practices in reducing carbon emissions generated from ICT's. The study emphasizes the importance of applying green computing strategies in businesses and industries for achieving the global carbon reduction goals. It also acknowledges the implementation challenges in adopting green computing practices on a global scale. The paper highlights the benefits offered by adopting green computing which include cost savings, business continuity and enhanced corporate image.

Impact of Information technology (IT) on Environment is increasing day by day. The paper states that this impact can be reduced by adopting novel strategies for improving energy efficiency Raja (2021). The paper discusses a real life example on carbon footprint analysis that demonstrates how individual IT usage contributes to carbon footprint. The author monitored power consumption of various devices such phones, laptops, and microwaves. The study shows that microwaves consume maximum energy among all devices, and devices like Wii consoles consume minimum energy. The scenario offered individual energy consumption patterns and environmental impact associated. Using sleep modes or hibernation, power management and employing energy efficient practices can lower the impact and contribute to Green computing.

Drastic growth in technology has led to a significant rise in the use of computers and other resources such as sensors, data centers, and storage devices Agarwal et al. (2021) and Badhoutiya (2022). This has resulted in higher power consumption and carbon footprints which is contributing to environmental concerns of global warming. For addressing these concerns, the author emphasizes on including green computing which can be done by focusing on optimizing resource utilization and reducing individual energy usage. Ways recommended by author for reducing include opting for energy-efficient laptops over desktops, smaller screen sizes, and using Ink Jet printers for lower energy consumption. The paper concludes that by combing individual efforts with industry innovation, it is possible to achieve green computing.

This section targeted on the importance of adopting green computing practices and provided ways for achieving the same. The review showed how many cloud providers like AWS, Google and IBM are working on lowering the environmental impact of IT by including green computing strategies. Next section is focused on reviewing works done in reducing carbon emission specifically in Green Cloud Computing.

2.3 Advancements in Green Cloud Computing

A novel fuzzy logic based resource management algorithm has been discussed in the paper Hussein et al. (2020). The algorithm works on enhancing VM allocation and migration

policies by employing a Fuzzy Rule Based System (FBRS) for VM placement during resource allocation. Experimental results claim that the power consumption was reduces by 30-40% compared to conventional methods but with a limitation of increased execution times. The paper also highlights the need for optimizing the execution time to address performance concerns.

Different approaches for achieving Green cloud computing have been discussed in the paper Sailesh et al. (2023) and Suratia et al. (2023). It includes strategies like upgrading old data centers to modern for lesser heat generation, heat reuse and power management techniques. The study highlights the burden on computer server due to increased use of virtualized environments. It also discusses the strategies of Virtual Machine load management and efficient job allocation for minimizing the energy consumption. The author emphasizes on adopting Green cloud computing activities by focusing on energy efficiency and optimizing resource utilization.

A significant drawback in cloud computing is the under utilization of resources. When cloud provides the services to end users, some services may remain underutilized. As per the paper Jayalath et al. (2019), since some services are not used to their full potential, energy usage is done in an inefficient way which results in increased carbon emissions. The paper includes the methods for mitigating the under-utilization of resources to achieve green cloud computing. The study also highlights the that major cloud providers like Google, Microsoft, AWS and IBM have are actively working towards adopting green computing by working on developing energy efficient and green practices.

The research provides a novel algorithm of dealing with energy consumed by Idle Virtual Machines. Author claims that idle VM consumes around 50-70% of total server energy as stated Hossain et al. (2020). To address this issue, a novel Active & Idle Virtual Machine Migration (AIVMM) algorithm has been implemented that reduces the energy consumed by idle Virtual Machines. It also addresses the Virtual Machine Placement (VMP) problem along with reduction in energy consumption. The new approach efficiently migrates the idle virtual machines from an actively working server and places them in an in-active server with the objective of reducing power interruption for the active machines. Findings show that the AIVMM has achieved the objectives of improving resource utilization, and reducing energy consumption and achieving green cloud computing.

Various Load balancing techniques have been analysed in the paper for enhancing the performance metrics. A correlation between green cloud computing and load balancing is explored in the paper Girsa et al. (2023). Optimizing load balancing algorithms can help achieving resource efficiency and minimizing the impact on environment. Overall, the author explains the relation between load balancing and green cloud computing and emphasizes on adopting green cloud strategies.

The major contributor of carbon emissions is the inefficient use of power during transmission of data, data storage and data handling. The paper Mehta et al. (2023) introduces a new approach for reducing the power usage by implementing various techniques. These techniques include software optimization, Network optimization and hardware optimization. Software optimization approach aims to improve software efficiency by shutting down under-utilized servers. Hardware optimization technique adjusts the server's com-

putational capacity by optimizing the frequency and voltage. Energy usage can be reduced by minimizing the 'Network Traffic' across servers. Adopting these Green Cloud Computing techniques for reducing energy consumption can decrease the environmental impact of cloud technologies.

The section emphasized on techniques for reducing the energy consumption specific to Green Cloud Computing. The techniques involve Load balancing, task scheduling and Virtual Machine Migration. In next section, studies focusing on Green DevOps have been covered.

2.4 Advancements in Green DevOps

Rifa et al. (2021) introduced sustainable testing methods in CI/CD pipelines which contribute in the field of Green DevOps. This research introduced a novel approach where a machine learning model which classifies the tools and the test cases in the testing stage, which results in a perfection of the test results. The model focused on optimization of the testing procedure and tools which lowers the energy consumption, hence reducing the carbon generated and contributing to the field of Green DevOps.

There are various techniques for testing in CI/CD pipelines which result in the generation of a significant amount of carbon footprint Nayak et al. (2024) among which one is Green Continous Test (Green CT). It was observed that by decreasing the number of application test cycles reduced the carbon generated without impacting the performance efficiency. The study highlights three main strategies for Tests which reduce the carbon footprint significantly - 'Design and optmization', 'Automation and its scripting and execution time' and 'Management for continuous corpus management and for maintaining it relevancy and status' but the study only emphasises on the testing phase in the pipeline and other stages remain uncovered.

This section covered the studies related to Green DevOps field. From the literature review, it is evident that most of the researches do not focus on the code analysis which is the identified gap we have worked upon in the proposed research.

2.5 Summary of Literature Review

Each article in the overall literature review of the related studies mainly focused on the importance and adoption of green computing which lowered down the carbon footprint generated. But it could also be observed that the studies lack in enough practical approaches for Green DevOps. They focus more on reducing generated carbon with testing techniques but in this research an emphaisis on code analysis is also done which helps in filling a clear gap in the contributions in Green DevOps field.

3 Methodology

This section outlines the detailed methodology and research tools and methods used in addressing the issue of increased carbon footprint associated with CI/CD pipeline, especially during the code analysis stage. To reduce the impact on environment, various

tools and techniques have been integrated to measure, monitor, and reduce the carbon emissions.

3.1 Code analysis stage in CI/CD Pipeline

Before deployment of the software to the production environment, analysis of code is done for ensuring the code quality. The analysis checks for any security issues, syntax errors, unused imports and undefined variables. This is a basic procedure for checking code quality and is included in almost every CI/CD pipeline. This can be done by using various tools available in the market such as Pylint, SonarQube, SonarCloud, and more. In our research we have used Pylint as the Static Code Analysis tool. Pylint is a most widely used tool that checks for any syntax errors, undefined variables and unused imports in the python code. Whenever the Pylint stage in pipeline gets initiated, it checks all the files for any errors in the code and logs the errors in the log file. As the pylint checks all the files, it consumes high amounts of energy especially in large applications with around 1000 code files. This results in increased carbon footprint. The research aims to reduce this carbon footprint by optimizing pylint.

3.2 Optimizing Code Analysis

As discussed earlier, the code analysis stage consumes high amounts of energy and is the major contributor to increased carbon footprint in CI/CD pipeline. Since our research uses pylint as the analysis tool we have worked on optimizing pylint. Pylint works by checking all the code files for any syntax or unused imports errors. Optimization of pylint can be done by reducing the number of files pylint checks whenever the pipeline gets initiated. The proposed solution includes the integration of GitHub Action with pylint, that enables Pylint to check only the files which have been modified in the last push rather than checking all the files. This will have huge impact on large application with around 1000 files. Suppose out of these 1000 files only 100-150 files have been updated in last push, the pylint now checks only those modified 100-150 files rather than checking all 1000 files. This solution helps in decreasing the amount of energy consumed and hence, carbon footprint is significantly reduced.

3.3 Research Tools

This section describes the tools used in conducting the research. The tools used along with their purpose is included in Table.

3.3.1 AWS Cloud9

Cloud based IDE that enables developers to write, test and run code in multiple programming languages without the need of local setup. In our reseaarch AWS Cloud9 has been used to develop the Application under test. The application is developed using Django framework.

Table 2: Summary of Tools Used

Tools	Purpose
AWS Cloud9	IDE for developing the Application under test
AWS CodePipeline	To build, analyse code and deploy application under test
Pylint	For code analysis stage in CI/CD pipeline
GitHUB Actions	Helps in pylint optimization
CO2Signal API	API integrated with lambda for detecting real time carbon emissions
AWS Lambda	Fetches carbon footprint generated by CI/CD pipeline
AWS CloudWatch	Monitors CPU usage, Memory usage for CI/CD pipeline
Elastic Beanstalk	Used in deployment stage for deploying the application under test

3.3.2 AWS CodePipeline

CodePipeline is a continuous integration and continuous delivery (CI/CD) tool that helps automating the steps required in release process. A CI/CD pipeline consists of various stages. Our pipeline includes three stages namely source, code analysis and deploy. The first stage being the source stage is connected with a private GitHub repository, which gets initiated whenever a code push is made to the pipeline. The next stage is the code analysis stage integrated with pylint for static code analysis. Last stage is the Deploy stage integrated with Elastic Beanstalk for deploying the application.

3.3.3 Pylint

Pylint is a static code analysis tool used for detecting issues in python code. The tool usually checks for any syntax errors, undefined variables and unused imports. It is integrated with the CI/CD pipeline for ensuring code quality before deployment.

3.3.4 GitHub Actions

GitHub Actions is mainly used for automating workflows. In our research it is integrated with pylint for optimizing the code analysis stage in CI/CD pipeline.

3.3.5 CO2Signal API

A third-party API that provides real time carbon intensity data based on electricity consumed in a given region. Integrated with lambda function it helps detecting the carbon footprint generated by CI/CD pipeline.

3.3.6 AWS Lambda

Lambda is a serverless computing service that enables developers to run the code without provisioning or managing servers [1]. The Lambda function fetches the CPU usage, Memory usage and runtime minutes metrics from the CI/CD pipeline and calculates carbon footprint generated by pipeline by using the carbon intensity data fetched from the API.

[1]https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

3.3.7 AWS CloudWatch

CloudWatch is a tool for monitoring application performance. It automatically collects metrics from various services such as Lambda, Build, EC2, and more. CPU utilization, Memory Utilization, Storage Read/Write consumed by CI/CD pipeline can be monitored using CloudWatch.

3.3.8 Elastic Beanstalk

Beanstalk is a Platform as a service (PaaS) that is responsible for deploying applications quickly in the cloud. This is integrated in the last stage of pipeline for deploying the Application under test.

4 Design Specification

The proposed research methodology is based on systematic integration of various tools and services for optimizing the code analysis stage of CI/CD pipeline. Objective is to minimize the carbon emissions generated from CI/CD pipeline. This section outlines the Design Specification, Architecture and techniques used during the research.

4.1 Architecture Design

Figure 1 represents the architecture design of the proposed methodology. As shown in the figure, Cloud9 is used as IDE for development of the application under test which is connected to GitHub. The source code of the application under test is stored in a GitHub repository. Each time code change is pushed to repository, CI/CD pipeline gets triggered. CI/CD pipeline comprises of three stages mainly Source, Build and Deploy stage.
Whenever a change is made to the application's source code stored, the modified code is pushed to the private GitHub repository. This triggers the pipeline, and initiates the source stage which is the first stage of the pipeline. The source stage fetches the code from the repository. After the completion of source stage the pipeline then moves to the next stage of the pipeline, the Build stage. For build stage we have used AWS CodeBuild. This stage is code analysis stage which is integrated with pylint that ensures the code quality by checking for any syntax errors, unused imports and more in the code. When the build stage gets initiated, pylint comes into action and starts checking all the files for errors and logs the detected errors in the log file. AWS CodeBuild requires a build specification file to be included in the build project. This file contains the commands and settings that AWS CodeBuild uses to build and package the code. A buildspec.yml file for pylint contains the build steps and configuration commands.

As per the proposed methodology, pylint is configured with GitHub actions to only analyze modified files, that optimizes the process and avoids the unnecessary checks which in return decrease the energy consumed by pylint. The pylint should be able detect the modified files and perform the analysis only on these modified files. The resource utilization from the code build stage can be monitored using AWS CloudWatch. After the successful completion of code analysis stage, the last stage that is the deploy stage

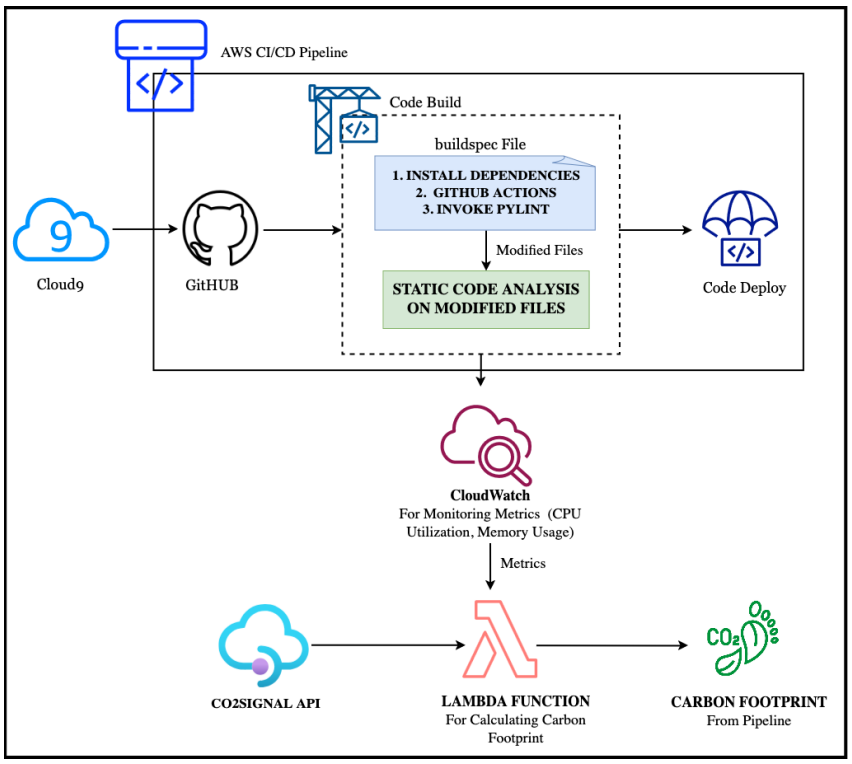

Figure 1: Architecture of Proposed Solution

gets initiated. For deploying the application we have implemented Elastic beanstalk that automatically deploys the application to the cloud.

4.2 Lambda Function and Carbon Footprint Detection

The main aim the research is the detection of carbon footprint associated with CI/CD pipeline, specifically from the CodeBuild. We have implemented a lambda function for detection of carbon footprint. The function makes use of CO2Signal API that provides the Carbon Intensity (gCO2/kWh) for a specific region. This intensity is used to calculate the Carbon Footprint generated by the build stage of the pipeline. Resource Utilization of build stage is monitored using CloudWatch and metrics like CPU Utilization, Memory usage and more, can be collected from the CloudWatch. These metrics are then sent as test event into the Lambda Function and by using these metrics we can calculate the Energy Consumption (in kwH) as mentioned in the equation 1.

$$EnergyConsumption(kwh) = (\frac{CPU_{usage}}{100} \times 0.1) + (Memory_{usage} \times 0.05) \quad (1)$$

After the calculation the Energy consumption by code analysis stage, the carbon footprint can be calculated using this energy consumption and the carbon intensity as stated in equation 2.

$$CarbonFootprint(gCO2) = EnergyConsumption(kwh) \times CarbonIntensity(gCO2/kwh) \quad (2)$$

```
pre_build:
  commands:
    - echo "Reinitializing Git repository in CodeBuild environment"
    - git init  # Initialize a new Git repository
    - git remote add origin Github_Repo
    - git fetch origin main  # Fetch the main branch
    - git clean -fdx  # Clean working directory to prevent conflicts
    - git checkout -b main origin/main  # Check out the main branch
    - echo "Fetching updated Python files from the last Git push"
    - git diff --name-only HEAD~1 HEAD > changed_files.txt  # Find updated files
    - echo "Files detected for linting:"
    - cat changed_files.txt

build:
  commands:
    - echo "Running Pylint on updated Python files"
    - |
      exit_code=0
      pylint --disable=R0903,R0901,E1101 --output-format=colorized $(cat changed_files.txt) ||
      exit_code=$?
      echo "Pylint completed with exit code: $exit_code"
      exit $exit_code
```

Figure 2: buildspec.yml File

5 Implementation

The proposed novel technique aims to detect and reduce the carbon footprint by optimizing the code analysis stage of the pipeline. To implement this, pylint was integrated with GiHub actions that enables pylint to check only the files which have been updated in last git push. This decreased the CPU and Memory Utilization consumed by pylint. For detection of carbon footprint these metrics were passed as test event to the Lambda function. Let's delve deeper into the implementation of GitHub Actions, pylint and Lambda function.

5.1 Integration of GitHub Actions with Pylint in CI/CD Pipeline

A build specification file is a configuration file used by AWS CodeBuild that includes the steps which should be performed during the build process. The file is written in YAML format and defines various phases such as, installation, pre-build, build and post-build. Whenever a build is triggered, AWS CodeBuild uses the instructions in the buildspec.yml file and executes each phase in sequence. The install phase installs all the dependencies, the build phase runs the build commands, and the post-build phase can be used for other tasks. A code snippet of buildspec file is depicted in Figure 2.

The above buildspec.yml file defines two phases; pre-build and build. These phases collectively initializes a Git repository, fetch the modified files from the latest Git push, and then run Pylint only on the updated Python files.

Pre-build phase: This phase includes the steps necessary for preparing the environment and identifying which files needs to linted by pylint. The phase starts by initializing and setting up of Git repository. It fetches the main branch of the repository to ensure the latest version of the code is available. After the Git setup, the code then checks for files which have been modified in the last push by using the "git diff" command that compares the current commit (HEAD) with the previous commit (HEAD 1) and lists the updated files. These files are then saved in a text file named "changed_files.txt" file. This ensures that only the modified files are considered for linting.

```
# Fetch carbon intensity data from CO2Signal API
headers = {'auth-token': CO2SIGNAL_API_TOKEN}
url = f"https://api.co2signal.com/v1/latest?countryCode={region_iso}"
response = requests.get(url, headers=headers)
response.raise_for_status()
# Parse API response
data = response.json()
carbon_intensity = data.get('data', {}).get('carbonIntensity', 'N/A')
fossil_fuel_percentage = data.get('data', {}).get('fossilFuelPercentage', 'N/A')
# Log the fetched carbon intensity data
logger.info(f"Region: {region_iso}")
logger.info(f"Carbon Intensity: {carbon_intensity} gCO2/kWh")
logger.info(f"Fossil Fuel Percentage: {fossil_fuel_percentage}%")
cpu_energy_kwh = (cpu_usage / 100) * 0.1 * (runtime_minutes / 60)    # Convert resource utilization to energy consumption (kWh)
memory_energy_kwh = memory_usage_gb * 0.05 * (runtime_minutes / 60)  # CPU power consumption = 0.1 kWh per 10% CPU usage per hour
total_energy_kwh = cpu_energy_kwh + memory_energy_kwh                # Memory power consumption = 0.05 kWh per GB per hour
carbon_footprint = total_energy_kwh * float(carbon_intensity)        # Calculate the carbon footprintin grams of CO2
# Log the energy consumption and carbon footprint
logger.info(f"Total Energy Consumption: {total_energy_kwh} kWh")
logger.info(f"Calculated Carbon Footprint: {carbon_footprint} gCO2")
```

Figure 3: Lambda Function

Build phase: In this phase, pylint is invoked and runs the analysis only on the updated files identified in the text file. This ensures Pylint only lints the updated files, reducing unnecessary linting of unchanged files. Pylint now runs on the modified files and checks for coding errors.

5.2 Monitoring CI/CD pipeline Metrics

After the integration of Github Actions with Pylint, the next step is to monitor the energy consumption of the pipeline, specifically in the build stage where pylint is integrated. Whenever a pipeline is triggered and build stage gets initiated, pylint starts analyzing the files for any potential and erros. As pylint is incorporated with Github Actions, it lints only files which have been updated in last git push. For this stage, the metrics such as CPU utilization and memory utilization are continuously monitored using CloudWatch. These values are necessary for the calculation of energy consumption. These values are then passed to Lambda function for detection of Carbon Footprint.

5.3 Populating Lambda Function with Pipeline Metrics for Carbon Footprint Detection

For detection of Carbon Footprint associated with the pipeline, a Lambda function is created. The function uses CO2Signal API to fetch the carbon intensity as per the specified region. The carbon intensity data obtained is then used for the calculation of carbon footprint. For detecting the Carbon footprint produced from the pipeline, calculations are performed on the Carbon Intensity information retrieved from CO2Signal API and the metrics obtained from the pipeline. A test event is created where the metrics (CPU utilization, runtime minutes, memory usage) obtained from the pipeline are passed

to the Lambda function. A code snippet of Lambda Function is shown in Figure 3, that starts with fetching the carbon intensity data from CO2Signal API. Then, the pipeline values which were passed through the test event are used for the calculation of Total energy consumption which is the sum of CPU utilization (kwh) and Memory utilization (kwh). The value obtained for Total energy consumption is then multiplied with carbon intensity to get the Carbon Footprint (gCO2) produced by the pipeline.

6 Evaluation

This section covers the experiments conducted during the research. The experiments evaluates how Integration of pylint in CI/CD pipeline affect resource utilization and what is the effect of integrating Github actions on resource utilization in the CI/CD pipeline compared to using Pylint alone.

6.1 Experiment 1: Pipeline without Pylint vs. Pipeline with Pylint

In this experiment, we monitored the Resource Consumption of pylint when incorporated in CI/CD pipeline. This experiment compares the resource usage between two pipelines where both pipelines are connected to same Github repository and gets triggered whenever the code change is made and pushed into Github repository.

1. Pipeline without Pylint: This pipeline runs normally without the code analysis during build process.
2. Pipeline with Pylint: This pipeline performs code analysis using Pylint during the build process.

During the execution of both the pipelines, metrics such as CPU utilization and memory usage were monitored continuously using Cloudwatch. The values obtained are shown in Table 3.

Table 3: Values obtained from conducting Experiment 1

Metric	**Pipeline without Pylint**	**Pipeline with Pylint**
Average CPU Utilization (%)	10.7	15.1
Average Memory Usage (MB)	69	78

These values are then passed as a test event to the lambda function for the calculation of Carbon Footprint. The lambda function uses the Carbon intensity data provided by CO2Signal API for a specific region, and the metrics values received from pipeline. Value Carbon Intensity obtained from CO2Signal API for region Ireland (IE) while conducting the experiment was 627 gCO2/kwh. The values for total energy consumption and Carbon footprint (grams per CO2) obtained from all the pipelines is mentioned in Table 5.

6.2 Experiment 2: Pipeline with Pylint vs. Pipeline with Pylint and GitHub Actions Integrated

As per the second experiment, two pipelines were created both with pylint included in the build process. In one of the pipelines, we integrated Github Actions with pylint which allows pipeline to analyze only the files updated in last push.

1. Pipeline with Pylint: This pipeline has pylint integrated into the build process of pipeline.
2. Pipeline with Pylint and Github Actions: This pipeline has pylint integrated along with Github actions.

During the execution of both the pipelines, metrics such as CPU utilization and memory usage were monitored continuously using Cloudwatch. The values obtained are shown in Table 4.

Table 4: Values obtained from Experiment 2

Metric	**Pipeline with Pylint**	**Pipeline with Pylint & Github Actions**
Average CPU Utilization (%)	15.1	12.9
Average Memory Usage (MB)	78	75

The values obtained are then passed as a test event to the lambda function for the calculation of Carbon Footprint. The lambda function uses the Carbon intensity data provided by CO2Signal API for a specific region, and the metrics values received from pipeline. Value Carbon Intensity obtained from CO2Signal API for region Ireland (IE) while conducting the experiment was 627 gCO2/kwh. The values for total energy consumption and Carbon footprint (grams per CO2) obtained from all the pipelines is mentioned in Table 5.

Table 5: Carbon Footprint Values obtained from Lambda Function

Metric	**Pipeline without Pylint**	**Pipeline with Pylint**	**Pipeline with Pylint & Github Actions**
Total Energy Consumption (kwh)	0.00023583	0.00095	0.0008325
Carbon Footprint (gCO2)	0.1478675	0.59565	0.521977

6.3 Discussion

With reference to the experiments conducted, it is evident that integrating static code analysis tools in the pipeline can lead to increased resource utilization. According to values of metrics obtained from Experiment 1, it can be seen that the integration of Pylint increased the average CPU utilization by 4.4% and increased average memory usage by

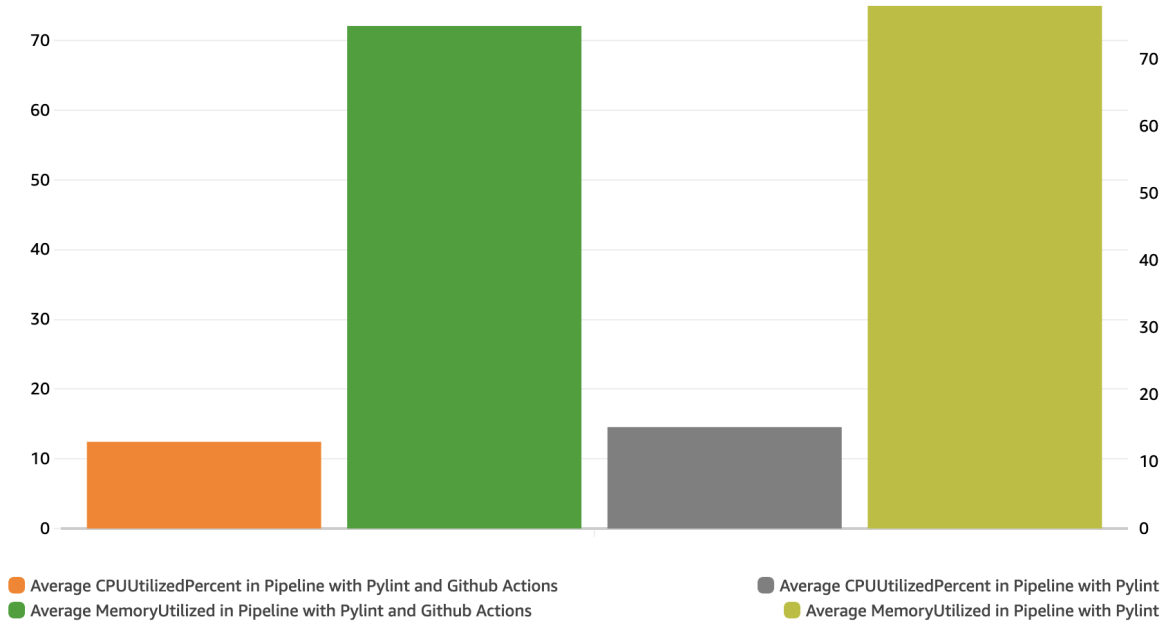

Figure 4: Metrics values obtained from Experiment 2

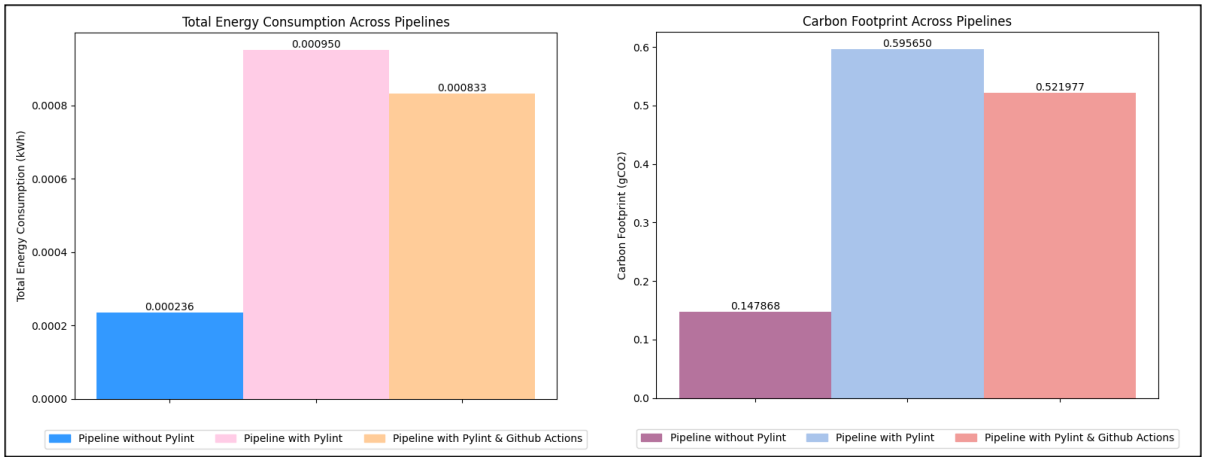

Figure 5: Total Energy Consumption and Carbon Footprint across pipelines

9 MB. This shows that Pylint contributes to increased CPU and memory consumption, leading to increased carbon emissions.

As per the results obtained from Experiment 2, it can be deduced that integration of Github actions with pylint reduced the average CPU consumption by 2.2% and memory usage by 3 MB. The results for Experiment 2 are shown in the Figure 4. This shows that limiting the analysis to modified files is a more efficient way to use resources.

The values of metrics obtained from CI/CD pipeline were then passed as test event in Lambda for calculation of carbon footprint. The results obtained for total energy consumption and carbon footprint are shown in Table 5. Visual representation of results can be seen in Figure 5. The percentage savings calculated for values obtained for carbon footprint is 12.36%. This is the result for a simple application, if the proposed method is applied to large codebases it will have huge impact.

The integration of GitHub Actions with Pylint reduced both CPU and memory usage,

thereby reducing carbon footprint. By only running Pylint on updated files, we effectively reduced the number of code files for analysis by Pylint, leading to a more efficient CI/CD pipeline. The results will have more improvement in Large complex applications where running static analysis only on the updated files rather than entire repository can lead to decreased computational overhead.

In the related work review, studies on CI/CD pipelines focused on the trade-offs between resource consumption and code quality. Using static analysis tools like Pylint in pipeline improves the reliability and maintainability of software but leads to increased computational overhead, especially in large applications. The proposed novel solution effectively decrease this overhead by running Pylint only on modified files, eliminating the need to compromise between code quality and resource consumption.

7 Conclusion and Future Work

The literature review conducted confirms that Cloud Computing technologies contributes to global Carbon Footprint. Due to the increasing demand for cloud computing technologies, resulting in increased carbon footprint. Many organizations are adopting various green computing strategies for reducing the environmental impact. A novel research methodology is proposed in the paper for minimizing the carbon footprint associated with DevOps activities. The results obtained from the experiments conducted show that optimizing code analysis stage in CI/CD pipeline effectively reduces the carbon emissions. This will help organization to decrease the carbon footprint and contribute towards corporate social responsibility (CSR).

However, the solution is implemented on a simple application. This can be extended to complex applications such as CPU Intensive applications, Machine learning applications and comparison can be done among the results obtained.

This approach can further be extended to optimize other static code analysis tool like Pixy, ESLint, or SonarCloud for increasing the versatility and applicability of solution.

References

Agarwal, V., Sharma, K. and Rajpoot, A. K. (2021). A review: Evolution of technology towards green it, *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 940–946.

Badhoutiya, A. (2022). Green cloud computing- next step towards eco-friendly work stations, *2022 6th International Conference on Electronics, Communication and Aerospace Technology*, pp. 809–813.

Girsa, D., Chauhan, D. and Sharma, D. (2023). Optimised load balancing for green cloud computing, *2023 13th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pp. 171–175.

Hossain, M. K., Rahman, M., Hossain, A., Rahman, S. Y. and Islam, M. M. (2020). Active idle virtual machine migration algorithm- a new ant colony optimization approach to consolidate virtual machines and ensure green cloud computing, *2020 Emerging Technology in Computing, Communication and Electronics (ETCCE)*, pp. 1–6.

Hussein, S. R., Alkabani, Y. and Mohamed, H. K. (2014). Green cloud computing: Datacenters power management policies and algorithms, *2014 9th International Conference on Computer Engineering & Systems (ICCES)*, IEEE, pp. 421–426.

Hussein, S. R., Alkabani, Y. and Mohamed, H. K. (2020). Green cloud computing: Datacenters power management policies and algorithms, *2020 9th International Conference on Computer Engineering Systems (ICCES)*, pp. 421–426.

Jayalath, J. M. T. I., Chathumali, E. J. A. P. C., Kothalawala, K. R. and Kuruwitaarachchi, N. (2019). Green cloud computing: A review on adoption of green-computing attributes and vendor specific implementations, *2019 International Research Conference on Smart Computing and Systems Engineering (SCSE)*, pp. 158–164.

Kannan, J., Barnett, S., Cruz, L., Simmons, A. and Agarwal, A. (2022). Mlsmellhound: A context-aware code analysis tool, *2022 IEEE/ACM 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pp. 66–70.

Kinkar, K., Bhosale, P., Kasar, A. and Gutte, V. (2022). Carbon footprint analysis: Need for green cloud computing, *2022 International Conference on Electronics and Renewable Systems (ICEARS)*, pp. 1–6.

Marandi, M., Bertia, A. and Silas, S. (2023). Implementing and automating security scanning to a devsecops ci/cd pipeline, *2023 World Conference on Communication Computing (WCONF)*, pp. 1–6.

Mehta, D., Tripathi, A. K. and Moazzam, J. (2023). An extensive review of optimized energy saving approaches for green cloud computing (gcc), *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, pp. 1270–1274.

Nayak, K., Route, S., Sundararajan, M., Jain, A. and R, S. (2024). Sustainable continuous testing in devops pipeline, *2024 1st International Conference on Communications and Computer Science (InCCCS)*, pp. 1–6.

Nikolić, D., Stefanović, D., Dakić, D., Sladojević, S. and Ristić, S. (2021). Analysis of the tools for static code analysis, *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–6.

Patel, U. A., Patel, S. and Nanavati, J. (2024). Towards a greener future: Harnessing green computing to reduce ict carbon emissions, *2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT)*, Vol. 1, pp. 960–969.

Raja, S. P. (2021). Green computing and carbon footprint management in the it sectors, *IEEE Transactions on Computational Social Systems* **8**(5): 1172–1177.

Rawas, S., Itani, W., Zaart, A. and Zekri, A. (2015). Towards greener services in cloud computing: Research and future directives, *2015 International Conference on Applied Research in Computer Science and Engineering (ICAR)*, pp. 1–8.

Rifa, P., Rakesh, V. and Kumar, S. S. (2021). Building sustainable software testing using machine learning for green engineering, *TERI Information Digest on Energy and Environment* **20**(4): 431–441.

Sailesh, C., Praneeth, V. S., Koushik, S. S., Sai, V. G. N., Vurukonda, N. and Burugari, V. K. (2023). A review on adoption of green cloud computing, *2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 1–6.

Shu, W., Wang, W. and Wang, Y. (2014). A novel energy-efficient resource allocation algorithm based on immune clonal optimization for green cloud computing, *EURASIP Journal on Wireless Communications and Networking* **2014**: 1–9.

Stanciu, A.-M. and Ciocârlie, H. (2023). Analyzing code security: Approaches and tools for effective review and analysis, *2023 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pp. 1–6.

Suratia, N., Thakkar, P., Sheth, K., Ramoliya, D. and Patel, A. K. (2023). An extensive analysis of green cloud computing: Overview, associated challenges and research directions, *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pp. 1147–1152.

Wadhams, Z., Reinhold, A. M. and Izurieta, C. (2024). Automating static code analysis through ci/cd pipeline integration, *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C)*, pp. 119–125.

Yeboah, J. and Popoola, S. (2023). Efficacy of static analysis tools for software defect detection on open-source projects, *2023 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 1588–1593.