CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Exploring Automated Reduction of the Carbon Footprint of Webpages

Master's thesis in Computer science and engineering

HAOZHOU LYU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Exploring Automated Reduction of the Carbon Footprint of Webpages

HAOZHOU LYU

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Exploring Automated Reduction of the Carbon Footprint of Webpages
HAOZHOU LYU

Supervisor: Gregory Gay, Department of Computer Science and Engineering
Examiner: Jennifer Horkoff, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Haozhou Lyu
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

An increasing number of people around the world have become concerned about climate change in recent years. With the increasing popularity and ubiquity of software, the ICT (Information Communication Technology) industry is an increasingly important contributor to climate change—and has the potential to significantly contribute to controlling climate change as well. In the ICT industry, software contributes a significant portion of the carbon footprint of the industry. Therefore, it would be beneficial to take measures to reduce the carbon footprint associated with software development.
This thesis explores developers' existing opinions, knowledge, and practices with regard to carbon footprint and energy consumption. It also explores the *automated* reduction of carbon footprint as well, with a particular focus on webpages.
Semi-structured interviews and a survey were conducted to identify the requirements that automated reduction tools must meet to ensure adoption. These requirements guided the creation of a genetic programming-based tool to automatically reduce the carbon footprint of webpages.
We compared the performance of this tool to two main baselines—the original webpage and randomly generated changes. The automated carbon footprint reduction tool can reduce the carbon footprint of webpages, yielding significantly lower quantities of transferred data, page load time, and memory usage than both baselines. Our findings offer a foundation for future research on practices, guidelines, and automated tools that address software's carbon footprint.

Keywords: Carbon footprint, Genetic programming, Software engineering, Case study, Thematic analysis, Web development.

Acknowledgements

I wish to express my sincere gratitude to my supervisor, Professor Gregory Gay, for his invaluable guidance, unwavering support, and constant encouragement throughout my thesis writing process. This thesis would not have been possible without his expertise and dedication. And I really learned a lot from his helpful guidance on how to write a formal academic paper. Furthermore, I would like to extend my appreciation to my examiner, Professor Jennifer Horkoff, for her valuable comments on the interim presentation. I am also thankful for the help from my girlfriend, family, and friends who have encouraged me throughout this process.

Haozhou Lyu, Gothenburg, May 2023

Contents

List of Figures

List of Tables

1

Introduction

The term "carbon footprint" [89], which is becoming increasingly popular worldwide, refers to the amount of greenhouse gases emitted by an individual, an event, or a product, expressed as carbon dioxide equivalent. Nowadays, carbon dioxide production is not limited to traditional methods, including industrial production, electricity, land transportation, etc. The carbon footprint of Information Communication Technology (ICT) has increased dramatically since the advent of cloud computing and the increasing use of mobile computing. The carbon footprint of ICT has become an important issue for the environment.

In order to save all types of energy sources and to support sustainable livelihoods, sustainable actions are necessary to reduce the carbon footprint of the world [42]. A company with approximately 10000 computers can reduce atmospheric CO2 content by 1381 tons by turning off their computers [40]. In addition, data centers contribute to about two percent of global greenhouse gas emissions and consume about three percent of the global electricity supply [7]. Software plays an important role in developing a sustainable society. According to Carla Schlatter Ellis et al. [25], reducing energy consumption should be given first-class status among performance goals when designing software. It is necessary that we pay greater attention to sustainable adaptation when designing software. Every day, almost everyone browses the Internet; web pages are some of the most commonly used pieces of software. Reducing the carbon footprint of popular web pages and making the reduction of carbon footprint an integral part of development would be beneficial to the environment.

This thesis explores how to raise acceptance of automated carbon footprint reduction tools and encourage voluntary adoption of them in order to reduce webpage carbon footprint automatically. Such a tool must be accepted and adopted by software developers in the real development environment in order to impact the way they develop software. Therefore, this thesis investigates how techniques such as Automatic Program Repair (APR) can best be adapted to developers' needs and fit into the development process. In order to determine what can be done to increase trust in such technologies, how to encourage voluntary adoption of such technologies, and what limitations are appropriate to ensure the results are acceptable to both developers and users, we conducted interviews (conducted in Sweden and Japan) followed by a broader survey. We analyzed qualitative data collected from inter-

views using the thematic analysis method. Based on the results of the interviews, a broader survey was conducted in order to include more participants (not limited to developers in Sweden and Japan). The quantitative data collected from the questionnaires were then analyzed. Based on the results of both interviews and surveys, we drew combined conclusions.

Furthermore, based on the requirements collected from the interviews and surveys, we utilized Automatic Program Repair (APR) [47] and Genetic Programming (GP) techniques to reduce webpage carbon footprint. APR is an emerging research area that addresses a critical challenge in software engineering and is able to greatly assist with software debugging and maintenance. Several program modification operators (PMOs) are used by APR in order to correct faulty programs. A PMO that corrects a fault identifies a potential repair. The modification space of a PMO can be determined by using an efficient search algorithm. An APR technique can be classified into two categories: semantic-based and search-based. In search-based APR techniques, a large number of possible repairs is generated, i.e., a large search space and the most appropriate repair is selected at random from the result set [46]. To perform APR, several methods can be utilized, including Genetic Programming (GP), a Search-Based Software Engineering (SBSE) approach that iteratively transforms a population of computer programs into a new generation of computer programs by using analogues of naturally occurring genetic operations [33].

1.1 Problem Description

There is an increasing awareness of the importance of climate change among the general public today. Increasing concentrations of greenhouse gases in the atmosphere cause severe global warming and its associated consequences [58]. Human life has been profoundly affected by global warming. According to agronomic studies [84], crop yields may decrease under various climate change scenarios if crops are grown in the same locations. Carbon footprints can be used as a metric for guiding emissions reduction and measuring greenhouse gas emissions from a variety of products and processes, and its standardization at the international level is therefore necessary [58].

The software has a significant role in contributing to the acceleration of climate change. It is estimated that training a single neural network — the core of Machine Learning—can emit as much carbon as the entire lifetime of five cars [31]. Moreover, the 416.2 terawatt hours of electricity consumed by the world's data centers last year exceeded the UK's total consumption of approximately 300 terawatt hours [7]. The researchers argue ICT's true proportion of global greenhouse gas emissions could be around 2.1–3.9 percent–although they emphasize there are still significant uncertainties around these calculations–indicating that ICT emissions may exceed those of the aviation industry by about 2 percent [86]. Thus, the software industry's carbon footprint should be reduced. What can we do regarding software engineering to positively impact climate change?

Nowadays, webpages are one of the most commonly used programs in the world. Based on Radoslav Ch et al. [30], there are approximately 2 billion websites and 400 million active sites. With over 3.7 billion people online and one million people joining every day, we have access to an endless supply of information. A total of 152 billion visits are generated by the top three websites, which are Google, YouTube, and Facebook [54]. Therefore, reducing the carbon footprint of web pages will contribute significantly to reducing the carbon footprint of the software industry. We should try to contribute to climate change by reducing webpages' carbon footprint.

According to Candy Pang's research [59], programmers lack basic knowledge of how to reduce software energy consumption and are unaware of the causes of software energy consumption. And it is difficult to convince software programmers to take the carbon footprint into consideration when developing and designing software. It is essential to develop methods that can automatically reduce the carbon footprint of software. It will save developers' time and effort, and make it easier for them to contribute to reducing global warming.

Furthermore, there are almost no case studies related to software engineers' opinions about the automated tool for reducing carbon footprints or energy consumption. Several interviews and surveys have been conducted previously in order to gain insight into software developers' experiences and opinions regarding reducing software energy consumption. Therefore, there is no reference to the automated tool for reducing carbon footprint.

1.2 Purpose of the Study

The purpose of this study is to explore the use of genetic programming to reduce the carbon footprint of webpages automatically. In addition, it explores how to ensure that software developers accept and are willing to adopt such technologies.

We explored this topic through two sub-projects. For the first sub-project, we explored developer acceptance of such a tool (and other similar technologies). In order to achieve to reduce the carbon footprint practically, such a tool must be voluntarily used by developers. We investigated how to increase voluntary adaption by identifying the answers to the following questions:

- How should tools fit into developer workflow?
- How can trust in this tool be improved?
- How can awareness of climate change be increased among developers?
- What constraints must be met by this tool during this process?

We adopted interviews in Sweden and Japan to answer the above questions, followed by a corresponding survey to get a broader set of responses. It is necessary to talk with developers to understand their opinions on the tools and carbon foot-

print reduction. We chose Japan and Sweden to represent East-Asia countries and Western European countries, respectively—enabling comparison between different cultures and geographic regions of the world. Both have an advanced development system; thus, we can compare these two countries to compare and contrast Eastern and Western developers' attitudes towards the reduction of carbon footprint and the use of automated tools to perform this task.

Based on this exploration, in the second sub-project, we developed a prototype tool that uses genetic programming to reduce the carbon footprint of a webpage. We considered an excessive carbon footprint as a "fault" in a program and apply search-based techniques from the APR field, such as genetic programming, to automatically reduce the carbon footprint. Since our automated carbon footprint reduction tool does not only involve a single objective, genetic programming can be extended to a multi-objective optimization algorithm, the NSGA-II algorithm, which provides multiple non-dominated solutions by searching the solution space. Thus, to use GP to reduce the carbon footprint, firstly, we need to identify which webpage design factors contribute to webpages' carbon footprint. For example, CPU usage is essential to the nonfunctional requirement to measure computer energy consumption. Less CPU power can result in less electricity consumption and a reduced carbon footprint. This method has previously been used to measure the energy consumption of other software systems. For example, the website for Notepad++, a text editor for Windows, specifically states that it tries to decrease CO2 emissions by "using less CPU power. It also throttles down and minimizes power consumption, resulting in a greener environment" [34].

Reducing the carbon footprint requires identifying factors of webpage design that contribute to the carbon footprint. Based on the factors, GP will generate solutions, which are potential patches to a program. Each patch consists of one or more changes to the source code. Each member of the population is scored using one or more fitness functions. A fitness function is a numeric scoring function based on the goals of performing the optimization [41].

There can be a preference for a smaller or larger score depending on the function. In this case, multiple fitness functions can be considered simultaneously. For instance, we may wish to balance the reduction of the carbon footprint with the loading time of the webpage. A fitness function may reduce a webpage's carbon footprint, and another fitness function may maintain the webpage's qualities (e.g., performance or scalability). Based on the fitness functions, each solution is scored and ranked from best to worst. As a result of genetic operations, a new population is formed, including reproduction (combining elements of reasonable solutions in order to produce even better offspring [82]) and mutation (some of the best solutions are mutated slightly, e.g., adding or removing from the patch). As time progresses, more and more of the population should achieve high scores, since new populations are formed based on the best solutions. As part of the project, we determined the exact scope of the improvement. For example, we could focus on visual elements (e.g., HTML, CSS) and underlying functionality (e.g., JavaScript), e.g., the CPU usage of the webpage and some other performance quality indicators. Furthermore, these factors

are used to design fitness functions and program modification actions intended to impact the fitness score. Fitness functions are used to select solutions that reduce the carbon footprint with the highest performance.

1.3 Significance of the Study

This thesis's scientific contributions include exploring the applicability of genetic programming to reduce carbon footprint and investigating software developers' opinions and knowledge on software's carbon footprint. Our results can inform developers and researchers who hope to design or apply energy consumption reduction tools, inspiring future work.

As a practical contribution, we provide a prototype tool that can automatically reduce a webpage's carbon footprint to improve the IT industry's carbon dioxide emission reduction and the convenience of the developer's operation.

1.4 Thesis Outline

Chapter 2 (Background) presents necessary research topics and terms related to the study. It mainly includes the concepts relevant to carbon footprint, energy consumption, and genetic programming.

Chapter 3 (Related Work) presents a brief overview of other research on software carbon footprint, energy consumption, and genetic programming related to the research topic.

Chapter 4 (Methodology) presents the research questions and describes the methodologies adopted for the two sub-projects.

Chapter 5 (Results) presents the results of the design of the automated carbon footprint tool and the interview and survey study.

Chapter 6 (Discussion) presents the answers to the research questions and the additional relevant observations. Moreover, discuss the limitations and threats to validity.

Chapter 7 (Conclusions) summarizes the conducted work and discusses future work.

2

Background

This chapter presents the essential background knowledge required to understand the topics of carbon footprint, energy consumption, automated program repair, and genetic programming.

2.1 Carbon Footprint

2.1.1 The Definition of Carbon Footprint

A carbon footprint represents the total amount of carbon dioxide (CO2) emissions associated with a person's or organization's activities. It includes direct emissions, e.g., results from fossil fuel combustion during the process of production, heating, transportation, etc. In addition, it includes indirect emissions, such as emissions produced by the consumption of electricity for service and goods [89].

2.1.2 Calculation of Carbon Footprint

It is well known that greenhouse gases released by industrial processes pollute the environment around the world, causing global warming and a continuous negative impact on living things and nature, including higher sea levels, loss of sea ice, and a reduction in animal habitat. In order to measure the gas intensiveness of divergent products, processes, organizations, etc., expressed as a carbon footprint [58], one must adhere to the principle of "What gets measured gets managed." [11]. Calculating the carbon footprint is, therefore, an essential aspect of every aspect of our daily lives and production processes.

An assessment of the carbon footprint of a product or process can be conducted using a Life Cycle Assessment (LCA), a method of assessing the environmental impact and resource consumption of a product throughout the life cycle (i.e., from raw material acquisition through production and use phases to waste management) [27], or by performing carbon accounting calculations. LCA measures the greenhouse gases, also known as GHGs, emitted throughout the life cycle of a product (mainly carbon dioxide and methane). In order to calculate greenhouse gases, the following normative framework is used [85]:

1. Selection of GHGs: It is necessary to examine both the need for carbon footprint calculations as well as the type of activity calculated in order to determine which types of greenhouse gases should be included in the carbon footprint calculation process. A large amount of carbon dioxide is produced during the production of this product, which is primarily derived from thermal power plants. Meanwhile, other gases (such as methane and nitrous oxide) are largely ignored. Therefore, it is important to take into account the specific circumstances when selecting which GHGs should be calculated.

2. Setting the boundary: The term boundary refers to the definition of the scope of activity for the purpose of calculating a carbon footprint. Defining the boundary depends on the characteristics of the object for which the carbon footprint is calculated. In the research paper [85] the scopes are (a) all direct emissions, i.e., emissions generated on-site, (b) all embodied emissions in purchased energy, and (c) all indirect emissions.

3. Collection of GHG data: In collecting data on greenhouse gases, two methods are available: either direct measurement of the gases on-site, or the use of emission factors and models that take into account the consumption of fuels, energy, and other inputs that contribute to the emission of greenhouse gases. Furthermore, databases and other data sources provide information on carbon dioxide emissions worldwide in addition to on-site measurements.

According to the framework discussed above, the carbon footprint can be calculated using a variety of tools and methods. When the carbon footprint value of a project, product, or organization has been identified, a strategy can be developed to reduce its carbon footprint.

2.1.3 Carbon Footprint of Software

Aside from traditional carbon footprint emission sources, Information Communication Technology (ICT, a term that refers to technologies that provide access to information via telecommunications [67]) also contributes significantly to the global warming problem by emitting large amounts of greenhouse gases. ICT's carbon footprint is primarily comprised of individual devices and network products. The mission of green IT is to reduce carbon emissions, especially with regard to software that plays a key role in this process.

Since the software has a life cycle, it produces direct and indirect carbon emissions that can be estimated and measured. According to Taina et al. [78], the software lifecycle is commonly divided into five phases:

1. Development: The development and production of software are based on the requirements.

2. Testing: Testing the software in different environments.

3. Delivery: The software is delivered to customers.

4. Usage: The software is being used.

5. Maintenance: The software is repaired and updated.

Taina et al. [78] proposed that the carbon footprint of each phase consists of two aspects—material carbon footprint and energy usage carbon footprint. When the life cycle of a product is known, the carbon footprint of the product can be calculated based on the production, delivery, use, and disposal of the product. The carbon footprint is determined by how the required energy is produced. When calculating a carbon footprint, the amount of energy required to perform a particular task is multiplied by an energy carbon footprint factor, which determines how much carbon dioxide is emitted during the production of a kilowatt hour of energy.

However, it is difficult to calculate the energy carbon dioxide emissions from software. In the view of Andrae et al. [3], there are a number of components that comprise digital devices and services, including network transmission, data centers, consumer products, and hardware products, and all require energy and resources to operate.

1. Network transmission: Through transmission cables, data must be transferred between data centers, local area networks, and user devices.

2. Data centers: Energy consumed to store and serve data in the data centers.

3. Consumer device: Energy consumed by customers uses and interacts with the product.

4. Hardware production: Embodied energy is consumed in creating hardware, i.e., embedded chips, consumer devices, etc.

2.2 Energy Consumption

In terms of energy consumption, it refers to the amount of energy used to perform an action or manufacture something [17]. In the development of software, large amounts of energy can be consumed (for example, through the use of GPUs, CPUs, and RAM). An application's energy consumption is affected by a number of factors, including its processor, its device, its memory architecture, its display settings, and its active sensors. The amount of energy consumed is also affected by a variety of factors, such as the signal strength used for data transmission, and user preferences, such as the brightness of the screen, among others. According to Ciancarini et al. [13], the most meaningful metrics for calculating energy consumption are battery percentage, battery status, RAM (Random Access Memory), VRAM (Video RAM), and CPU utilization.

As for the factors that impact software energy consumption, according to Gustavo Pinto et al. [65], there are several reasons for software energy consumption as follows:

1. Unnecessary resource usage: When a resource is not in use, it should be powered down. An inactive resource that runs at full capacity can consume a significant amount of energy if it is not in use.

2. High GPU usage: GPUs consume more energy than CPUs because they have multiple cores and cooling devices. As clock rates increase, power consumption increases as well.

3. Background wallpapers: Wallpapers with white backgrounds and animations at a high frame rate are inefficient in terms of energy consumption.

4. Background activities: Unnecessary and inefficient background activity significantly impacts a system's performance, power consumption, and memory footprint.

2.3 Automated Program Repair

Automated Program Repair (APR) is a method of automatically repairing software errors and vulnerabilities [29]. Programmer productivity can be improved, security vulnerabilities can be fixed, self-healing software can be developed for autonomous devices, and hints can be generated for programming assignments using APR. Automatic repair techniques are designed to identify patches automatically, which are then applied without the need for human intervention.

Kui Liu et al. [48] mentioned that Automated program repair workflows consist of three standard steps:

1. Localization of faults, which identifies the locations of code that need to be modified.

2. Generation of patches, which implements the changes to the code locations using change operators.

3. Patch validation, where the patched program is checked to ensure it adheres to the test case's expectations.

Using APR, buggy programs can be corrected to meet specified specifications. Noller et al. [55] proposed that depending on the program specifications, automated program repair can be divided into three types. It is important to note that test suites are treated differently from bug reports in the first type of report. For example, APR techniques such as GenProg [47] and Prophet [49] regard test suites as correctness specifications that need to be corrected to make the buggy program pass all the given test suites.

Other tools, such as Extractfix [28] and CPR [73], use constraints as correctness specifications instead of test suites. It is possible to present constrained inputs as ranges of inputs or as the entire input space in order to correctly patch the program.

In APR, information can be extracted about the underlying cause of a vulnerability, which can then be used as guidance for the APR process. It is necessary to meet a constraint at the location where the vulnerability is observed in order to extract information from the program. Therefore, in order to prevent the recurrence of the vulnerability, the purpose of repair is to satisfy the constraint at the site of the vulnerability.

Current state-of-the-art APR techniques can be divided into three categories—search-based techniques, which apply robust metaheuristic search optimization techniques; semantic-based techniques, which rely on lexical and latent semantic analysis. And learning-based techniques [36], which are based on machine learning techniques. One search-based APR technique is GenProg [47], which involves mutating program sentences to evolve the buggy program. There is an approach to semantic program repair known as Angelix [51] that uses symbolic execution to extract constraints from a program. Furthermore, Prophet [49] analyzes open-source software repositories to determine which patches are the most correct.

2.4 Genetic Programming and Genetic Improvement

The Genetic Improvement (GI) technique [32] is a Search-Based Software Engineering (SBSE) technique [33] that seeks to improve software's nonfunctional properties by treating program code as genetic material that needs to evolve to produce optimal results [8].

Genetic Improvement is a popular method for performing APR since it improves and repairs software by providing patches to the source code [88]. It has been widely recognized that GenProg [47] is an automated program repair technique that utilizes genetic programming as an extended form of evolution to create a program variant, encode defects and functionalities with test suites, and then minimize the amount of repair required between the variant and the original program.

Genetic Improvement is a process of improving software quality automatically through the use of Genetic Programming (GP). GP is a type of Evolutionary Algorithm (EA), which is a subset of machine learning. The process of genetic programming involves genetically breeding a population of programs in order to automate the process of solving a problem. In genetic programming, analogues of naturally occurring genetic operations (such as crossovers, mutations, and reproductions) are applied to a population of programs to create a new generation.

The basic Genetic Programming steps are as follows:

1. Generate an initial stochastic population.

2. An evaluation of each program in the current population is conducted against the given dataset to determine how well it performed (Fitness function), the

value of which is recorded as a fitness score.

3. Compare the fitness scores of randomly selected programs in order to determine whether their performance is good or bad.

4. Using one or more of the three genetic operators, create a copy of the leading program:

   (a) Reproduction (The program is copied without modification.)

   (b) Mutation (The program is modified randomly with a certain degree of probability.)

   (c) Crossover (An individual program is created by combining the code of two selected programs into a new program.)

5. Evaluate all programs in each new generation.

6. Perform Steps 2 - 5 (selection, genetic operation, and evaluation) iteratively until termination criteria are met.

GP is comprised of a population of individual programs within each generation. *Programs* are functions that can be expressed as concrete values. As a first generation of programs, randomly constructed programs are constructed based on user-defined programs. Each program is evaluated within the context of the entire initial population, i.e., each program is evaluated within the context of the given data set, resulting in only one outcome for each program. Based on the comparison between the outcome and the known solution, the fitness score is calculated. Randomly selected programs are entered into a tournament in which the programs with the highest fitness scores are copied into the next generation of programs. Each mutation and crossover may result in a reduction or improvement in the fitness score of each program. A new generation of programs is evaluated, randomly selected for tournaments, and then evolved.

Until now, genetic programming has been used in many applications. By using GP, Bruce et al. [8] reduce energy consumption by finding solutions that were difficult to find by humans. By porting functionality from one program to another, Petke et al. [5] implement new features, and Oliveira et al. [19] reduce the costs associated with testing and bug fixes.

3

Related Work

This chapter describes the state-of-the-art and present research on calculating software's carbon footprint and energy consumption, using genetic improvement to reduce energy consumption and developers' opinions on carbon footprint.

3.1 Estimation of Software Carbon Footprint or Energy Consumption

For the purpose of assisting users in selecting the most appropriate software, Amsel et al. [1] have developed a system titled "Green Tracker", which estimates energy consumption and informs the user of the impact of software to help inform their choice among alternatives. Among the main components of computer energy consumption, Green Tracker focuses on CPU usage. Users are prompted to specify the type of software they wish to test before installing any alternative on the same operating system. Following this, the tool will extract the CPU usage of the software and average it over a period of time. Notepad++, a text editor, also claims to reduce CO2 emissions by "using less CPU power, resulting in a greener environment" [34].

The researchers Narendran et al. [79] examined the energy consumption of mobile browsers, including mobile websites and web elements, from the perspective of battery consumption. An energy multimeter and an available battery are used to measure energy consumption. Their energy is measured in joules or millijoules. To gain a deeper understanding of these numbers, they take into account the amount of energy consumed as a percentage of the battery's capacity. "1% of a fully charged battery is approximately 32.84 Joules.". To control variables, the research uses traffic data (uploads and downloads) and the type of website as independent variables. Finally, they presented an experimental framework for measuring web page power consumption.

Adrian et al. [72] utilized WebChar, a model-mining system that analyzes browsers in the context of real-world HTML and CSS content. By automatically identifying factors that negatively affect a website's performance and energy consumption, WebChar is able to reduce its energy consumption and performance. In order to measure the performance of the web page's sections, they measure the load time, the energy in Joules, and the power in Watts. Upon receiving the browser's first

request and callback, the server calculates the loading time of the page. The page load energy is determined by measuring voltage and current at the same time.

Yuhao et al. [93] developed a statistical inference model that estimates webpage load time and energy consumption by analyzing the performance of the hottest 5,000 web pages. The load time measurement is measured in microseconds. The browser cache has been disabled to prevent contamination of partially cached webpage resources. A power-sensing circuit was developed using a sense resistor located in front of the off-chip voltage reference module to measure energy.

The Python package CodeCarbon [71] measures an application's electricity consumption in three dimensions (GPU, CPU, RAM) and multiplies it by the region's carbon intensity to determine how much carbon dioxide is produced. Intel uses the Inter Power Gadget to monitor the power consumption of its processors. Upon obtaining energy data, this tool identifies two crucial arguments: the first is the carbon intensity of the electricity used in computation, expressed as kilowatt hours of carbon dioxide emitted per kilowatt hour of electricity, and the second is the power consumed by the computational infrastructure, expressed in kilowatt-hours.

Cao et al. [10] present RECON, a model that accurately predicts mobile web pages' energy consumption. During the period of a webpage's loading, it is divided into segments, each segment corresponding to a period during which its components do not change. In particular, RECON monitors the resources, CPU utilization, frequency, bytes, and packets transmitted over WiFi.

Dick et al. [23] have proposed a method for measuring and rating software energy consumption. There are four components in the system: a System Under Test (SUT), a computer hosting the application program, a Workload Generator that generates statistically reproducible workloads for the SUT, and a Power Meter that measures the power consumption of the system.

WattsOn was developed by Radhika et al. [52] to assess the energy consumption of an application in a development environment. In order to calculate the app's energy consumption, it scales down the emulating environment (network, CPU, display) in order to mimic a phone environment and applies empirically derived power models.

In contrast to the evaluation methods described above, our study attributes greater importance to measurement methods through page load time, RAM usage, and transferred data quantity. This combination differs from the measurement indexes previously discussed. When browsing a website, a customer's carbon footprint is determined by RAM usage. In addition, the carbon footprint of the transferred data bytes includes the carbon footprint caused by data transmission over the wire, the energy used in the data centre, and web traffic. The units of measurement are converted from bytes to kilowatt-hours (kWh). As a result, we can measure the carbon footprint by timing electricity carbon intensity.

3.2 Methods of Reducing the Energy Consumption of Webpages

Due to variations in HTML structure (HTML) and CSS style information (CSS), web pages load with varying speeds and consume varying amounts of energy. According to Yuhao et al.'s research [93], HTML tags have different load times and energy consumption. The `<input>` tag used for inputting characters takes 5X execution time compared to a simple `<h3>` header tag and consumes 6X more energy than the `<h3>` tag. The number and type of attributes also influence energy consumption and execution time. Compared to 1,000 cellspacing attributes, the background attribute consumes 19X less energy and 18X more energy. In addition, they observed that the size of the DOM tree is directly proportional to its rendering time and energy consumption. Aside from HTML selectors, CSS selectors also consume varying amounts of energy and time. Pseudo selectors consume 10X and 5X less energy and time than descendant selectors.

Adrian et al. [72] have found that `<strong>` elements are less energy consumed than unstyled `<span>` tags; the former uses about 14% less time and energy, and `<span>` tags are particularly inefficient in mobile device browsers. The use of tables in web development should be avoided since tabular data is resource-intensive. Moreover, translucent elements are displayed more slowly on mobile devices than opaque elements. Even though mobile devices do not consume a great deal of energy, their background energy consumption is significant. With the results of this study, developers can be assisted in selecting tags and contributors for HTML and CSS files.

According to Narendran et al. [79], rendering images consumes a substantial portion of the total rendering energy. A direct correlation exists between the number and size of images displayed on web pages and the amount of energy used in rendering them. In addition, different image formats consume varying amounts of energy. In comparison with PNG and GIF formats, JPEG compresses and renders images more efficiently.

3.3 Automated Reduction of Energy Consumption

The majority of existing research into reducing energy consumption is empirically based on understanding how different types of changes affect the overall energy consumption of applications, as described by Manotas et al. [50]—for example, choosing web servers, selecting web browsers, and using various algorithms. In spite of developers' ability to gather knowledge from empirical studies to improve energy savings, the layering of abstractions found in typical applications makes it impossible and inefficient for them to apply empirical knowledge acquired previously artificially. Therefore, developers are unable to predict changes and their impact on energy consumption based on these changes. To reduce software energy consumption au-

tomatically, it is necessary to use better techniques (such as GI). Since developers have difficulty applying this knowledge on their own, automated tools can more easily make changes that will have a significant impact on the development of the application.

According to Petke et al. [61], Genetic Improvement (GI), is a Search-Based Software Engineering (SBSE) technique that treats program code as genetic material that can be evolved to produce optimal solutions. It has been shown in previous research [43] that genetic algorithms can optimize software's execution time and that similar genetic techniques can reduce energy consumption. According to Petke et al. [62], his research demonstrated how MiniSAT, a popular Boolean satisfiability solver (SAT solver), can be optimized for execution time, resulting in a solver that is 17% faster than any human-written equivalent in the domain of Combinatorial Interaction Testing (CIT).

According to Bobby et al. [8], GI techniques have been used to improve MiniSAT's energy consumption based on SAT solver results. Instead of taking into account the number of lines executed in the fitness function, energy estimates have been incorporated by the Intel Power Gadget. In addition, they modified the selection and mutation procedures. Fitness candidate solutions can be identified by measuring the total energy consumed across all tests. Solutions must have a fitness score greater than 0.95, pass a sufficient number of selected test cases, and rank in the top 50 percent to be considered for the next generation. In the crossover process, one parent is selected based on fitness, while the other is selected at random. Following crossover, the selected genotypes are subjected to mutations. There is a 50% chance of a mutation being applied to the remaining selected individuals. The genotype is randomly modified by adding a DELETE, REPLACE, or COPY modification during a mutation.

According to David et al. [87], a low-power pseudorandom number generator was designed using genetic programming and multi-objective optimization to achieve a balance between non-functional requirements (power consumption) and functional requirements. Fitness is determined by an individual's ability to satisfy two objectives: reducing power consumption and optimizing randomness. As a result of comparing the fitness results of divergent experiments, it demonstrates a reduction in power consumption.

Using genetic programming, Mrazek et al. [53] reported that low-cost microcontroller programs could be improved in terms of non-functional properties. A suitable balance was found among the found programs in terms of accuracy, execution time, and power consumption. This technique takes advantage of the fact that genetic programming will be able to find a suitable trade-off between the number of errors and the size of the code, even if it is intentionally constrained.

Manotas et al. [50] present a general framework, SEEDS (Software Engineer's Energy-optimization Decision Support), which automatically optimizes Java applications. Code-level changes are made to optimize application energy usage by selecting the

most energy-efficient library implementations for Java's Collection API. Energy-efficient applications can be developed without applying modifications and the impact on energy consumption can be monitored. Genetic algorithms are used in this system to select versions of the application that consume less energy than the original version, and the optimal version that saves the most energy is ultimately selected.

A new method for designing low-resource systems using genetic programming was investigated by White et al. [88] in order to control the energy consumption of embedded systems using genetic programming.

Using the Genetic Algorithm from search-based software engineering and profile-guided optimization, Dorn et al. [24] proposed a software-oriented framework named PowerGauge. The use of this technique has resulted in an energy consumption reduction of 14 percent in 10 of the 12 benchmarks tested.

In addition to Genetic Improvement, some other techniques are adopted to reduce energy consumption automatically. Steinks et al. [75] proposed a compiler-integrated algorithm for analyzing an application and selecting portions to be placed in an embedded scratchpad to reduce embedded systems' energy consumption. Therefore, designs with the same memory size demonstrate remarkable advantages in terms of energy consumption.

De La Luz et al. [18] describe an automatic data migration strategy that places arrays with temporal affinity into the same banks according to their temporal affinity. With the implementation of low-power modes, a greater number of banks can be placed into energy-saving modes that are more aggressive. The migration of data can result in significant energy savings.

Since the above-mentioned methods focus on underlying unit changes rather than code changes, they are difficult for software developers to adopt and integrate into their development process. A relatively small amount of research has been conducted on automatically reducing the carbon footprint of software. Through the use of Genetic Programming, we are able to achieve a Multi-Object Optimization method for reducing the carbon footprint of a webpage by conducting genetic operations on the source codes (HTML, CSS, and JavaScript) of the webpages.

3.4 Developers' Opinions on Energy Consumption and Carbon Footprint

An anonymous online survey containing 13 questions was conducted by Pang et al. [59] on developer awareness of energy consumption. A total of 122 respondents completed the survey. According to the study, most programmers are unaware of the energy consumption of software and rarely address the issue of energy efficiency. Among the participants, only 14 percent considered minimizing energy consumption to be an important factor. Furthermore, programmers lack knowledge about how

to reduce the energy consumption of software. Although they are familiar with how to measure the overall energy consumption of hardware, they are unable to measure the fine-grained energy consumption of software. The results indicate that only a small percentage of participants are aware of the causes of high energy consumption.

Zakaria et al. [57] have surveyed to investigate the root causes preventing developers from considering software energy consumption more widely. A total of ten participants with experience in software development were interviewed. The main topics of discussion are awareness, knowledge, constraints, and responsibility for reducing software energy consumption. Based on the results of this in-depth qualitative study, it appears that most developers are unaware of the energy consumption of their software. Some developers perceive green software design as a threat. This study aims to better understand the development requirements of green software by software developers.

A study conducted by Pinto et al. [65] in 2014 examined how well programmers understand software energy consumption by surveying approximately 800 participants on StackOverflow and collecting 550 responses to 300 questions. According to their study, software developers are aware of the importance of software energy consumption, but they have only a vague understanding of it.

Furthermore, Lago et al. [42] discuss how the sustainability of software and its environmental impact has become increasingly important in the development of software systems. An evaluation of a software system's greenness requires defining theories about relevant properties, identifying design and coding decisions that consume less energy, and classifying contextual information.

The qualitative and quantitative studies discussed above focused primarily on software developers' awareness and prior knowledge of software energy consumption, but less on their opinions concerning software carbon footprints. By conducting an interview and a survey, we aim to gain additional information about developers' requirements for green software by investigating their awareness and knowledge of software carbon footprints as well as their attitudes toward techniques that automatically reduce software carbon footprints. Furthermore, we conducted interviews with software developers from Sweden and Japan. It enabled us to compare developers' awareness and opinions in these two locations and draw conclusions about their differences. Following the results of the interviews, a survey was designed and conducted to broaden the scope of the study to include individuals from around the world.

4

Methods

This chapter presents the methodologies applied in the study to answer the following research questions:

- **RQ1:** How can the acceptance and voluntary adoption of carbon footprint reduction techniques by developers be improved?
  - **RQ1.1:** What do developers know already about the carbon footprint or energy consumption of software?
  - **RQ1.2:** How do developers currently assess and control the carbon footprint or energy consumption of their own software?
  - **RQ1.3:** What requirements and constraints must be satisfied for developers to trust the result of using automated carbon footprint reduction tools?
  - **RQ1.4:** How should an automated carbon footprint reduction tool fit into the development workflow?
  - **RQ1.5:** How can voluntary adoption of automated carbon footprint reduction techniques be encouraged?
- **RQ2:** What factors of webpage design have an impact on the carbon footprint?
- **RQ3:** What impact does genetic programming have on the carbon footprint of webpages?
  - **RQ3.1:** What is the improvement in carbon footprint after applying genetic programming?
  - **RQ3.2:** Which program modifications have the largest influence on the carbon footprint?

Figure 4.1 provides an overview of the steps involved in answering the research questions. The project is divided into two sub-projects, which focus on defining the requirements and acceptance criteria for automated carbon footprint reduction tools

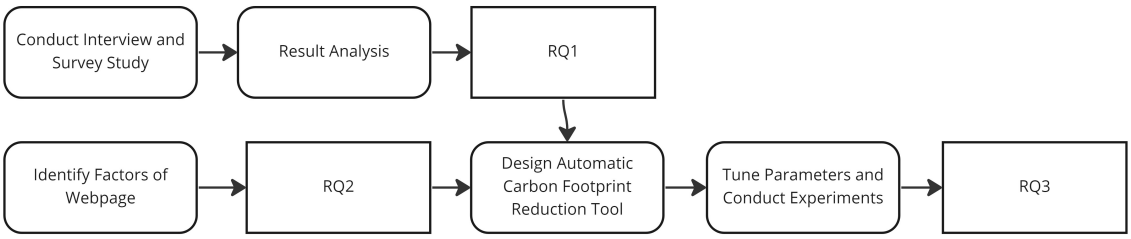

Figure 4.1: Overview of the methodology.

as well as the technical implementation of such tools.

RQ1 focuses on how to ensure that such tools are accepted and voluntarily used by developers. With RQ1.1, we aim to understand participants' prior knowledge of carbon footprint and energy consumption. The purpose of RQ1.2 is to assess participants' past considerations of energy consumption or carbon footprint, including their actions and measurements. RQ1.3 examines the requirements or constraints an automated carbon footprint reduction tool must meet to be used and trusted. In RQ1.4, we investigate how the tool fits into the developer's workflow. RQ1.5 explores how voluntary adoption can be encouraged. In order to address RQ1, we conducted interviews and surveys to determine developers' opinions, experiences, and requirements regarding carbon footprint and automated carbon footprint reduction tools.

To address RQ2-3, the Design Science methodology (DSRM) is applied to develop a prototype automated carbon footprint reduction tool for webpages. It is critical to ensure that the factors of webpage design have an impact on the carbon footprint (RQ2), which forms the basis of the subsequent steps. We then investigate the possibility of reducing carbon footprints by applying genetic programming (RQ3) after identifying the elements. RQ3.1 evaluates the impact of genetic programming on reducing carbon footprint. The purpose of RQ3.2 is to investigate which program modifications have the greatest impact on carbon footprints based on the findings of RQ3.1.

The Automated Carbon Footprint Reduction Tool can be accessed at:
`https://github.com/haozhoulyu416/ARCFW-Tool`

The Random Generation Tool can be accessed at:
`https://github.com/haozhoulyu416/ARCFW-Random-Solution-Generation`

The experimental data can be accessed at:
`https://github.com/haozhoulyu416/ARCFW-Research-Data`

4.1 Interviews and Survey (RQ1)

To answer RQ1, firstly, we conducted semi-structured interviews with developers in both Sweden and Japan to gain an initial understanding of developers' opinions and attitudes. Then, a survey was conducted to gain information from a broader range of participants. The survey was designed based on the interview results and was distributed internationally to enable confirmation or refutation of the interview results.

4.1.1 Interviews

We conducted qualitative semi-structured interviews study to investigate developers' opinions regarding the use of automated technologies to reduce the carbon footprint of the software they build automatically based on the thematic analysis process in the software engineering field [16]. This section explains the methodology for conducting and analyzing the results of the interviews, including the population, questions, participant demographics, data collection, and data analysis.

Population Definition: This research aims to investigate developers' opinions. Therefore, our focus is on participants with experience developing software, including both professionals in the IT industry and students studying computer science or software engineering at a university.

Sampling: From the population, the sample group consists of both employees of IT companies and students pursuing Master's degrees in software development. The sampling method was a mix of purposive and convenience sampling [26]. The IT industry professionals were gathered from companies in Sweden and Japan via the online platform LinkedIn, as well as through personal contacts. The students come from Chalmers University of Technology in Gothenburg, Sweden, and the University of Tokyo in Tokyo, Japan. Each is in the program of software engineering or computer science with experience developing using a variety of languages and tools. All participants have experience in the development of software. In order to obtain sufficient qualitative data, we interviewed ten practitioners. At this point, we achieved some degree of result saturation.

Participant Demographics: Table 4.1 shows demographic information on interview participants, including their location, type of organization (industrial or academic), current position in the organization, responsibilities in the organization, and total years of development experience. To maintain privacy and confidentiality, we omit participants' names and instead use IDs (P1 to P10).

These interviewees come from various roles (e.g., Chief Technical Officer, Technical Director, Senior Software Developer, and so on), with experience ranging from four to twenty-five years and various responsibilities, including overseeing technical road maps and staff, technical strategy, and developing web applications.

Interview Guide: We have conducted semi-structured interviews based on a set of

ID	Country	Context	Position	Responsibility	Exp.(Years)
P1	Sweden	Industry	CTO of the IT company	Overlook technical road maps	25
P2	Japan	Industry	Data Engineer	Data analysis and development for To B product	5
P3	Sweden	Academic	Masters Student	Have worked as a software developer for testing television	4
P4	Sweden	Industry	Technical Director	Work on technical strategy and development ways around the teams	20
P5	Japan	Industry	Software Engineer	Develop software, Apps	5
P6	Japan	Academic	Master Student	Web development	4
P7	Sweden	Industry	Software Engineer	Service planning and development	7
P8	Japan	Academic	Researcher	AI robotics development	6
P9	Japan	Academic	Ph.D. Student	Machine learning development	4
P10	Sweden	Industry	Software Engineer	Development of network operation and maintenance management tool	4

Table 4.1: Demographic information on interviewees, including location, organization context (industry, academia), position, responsibilities, and software development experience.

pre-prepared questions, listed in Table 4.2. It is important to note that the questions are open-ended, so we can more deeply explore participants' answers with follow-up questions if needed, while ensuring we answer the core research questions.

The interview questions were divided into three sections. First, we gather information regarding the participants' backgrounds and prior knowledge of software carbon footprint and energy consumption. The second part of the study concerns participants' experiences and opinions regarding carbon footprint and energy consumption. In the final part of the interview, the acceptance criteria and voluntary adoption of carbon footprint reduction techniques are investigated.

Data Collection: The whole interview process contained three steps. The first step of the interview involves a narrative portion in which we introduce the background of the interview, the purpose of the research, and the basic interview process in order to make the participants understand the context of the interview.

Second, we conducted semi-structured interviews. The semi-structured interview method is used because it allows participants to express ideas and allows us to go deeper into a response using follow-up questions during the interview. The interview begins with questions about the participant's current role, their experiences working on software development, as well as their knowledge of energy consumption and carbon footprint. After that, we proceed with the interview questions, which relate to their previous development experiences and attitudes regarding software energy consumption or carbon footprint.

As a final step, following the completion of the interview, if the participant was interested in reducing software carbon footprint or energy consumption, we answered their questions and shared information on the project and its progress with them.

From November to December 2022, all interviews were conducted online via Zoom and lasted between 20 and 30 minutes. Participants were interviewed in English to

Interview Question	Research Question
Background of Interviewee (Demographic Data)	
1.What is your current role?	N/A
2.How much experience do you have in software development?	N/A
3.What do you know about the energy consumption of software?	RQ1.1
4.What do you know about the carbon footprint of software or the impact that software has on the environment?	RQ1.1
Technical Details Questions	
5.Do you think reducing energy consumption or carbon footprint is the responsibility of the developers of a piece of software?	RQ1.2
6.Have you personally considered energy consumption or carbon footprint when developing software?	RQ1.2
7.At what development stages do you make this consideration?	RQ1.2
8.How did you assess the energy consumption or carbon footprint?	RQ1.2
9.What actions do you take to reduce energy consumption or carbon footprint?	RQ1.2
10.Do you write test cases to perform evaluation of energy consumption or carbon footprint? If so, what process do you follow to design these test cases and how do you execute and evaluate the results of testing?	RQ1.2
Acceptance and Voluntary Adoption	
11.What are your opinions on the use of automated technologies to reduce the carbon footprint or energy consumption of software that you develop?	RQ1
12.If available, would you use tools that automatically reduce the carbon footprint or energy consumption?	RQ1.3, RQ1.4
13.How do you foresee such a tool fitting into your development workflow?	RQ1.3, RQ1.4
14.When and how often would you use this tool?	RQ1.4
15.Would you trust the results of tools that automatically reduce the carbon footprint or energy consumption?	RQ1.3
16.What constraints would you require to be met in order to use and trust the results of such a tool?	RQ1.3
17.What considerations do you think would encourage you or other developers to adopt such techniques voluntarily?	RQ1.5

Table 4.2: Interview questions, mapped to the research questions.

avoid the possibility of inaccurate translations. In order to analyze the results, we recorded all of the interview videos and audio. We transcribed our records using a denaturalism approach [56] following the completion of each interview. The denaturalism approach transcription resolves mostly around the information content of the interviewee's speech. The focus of this approach is on what is said rather than how it is said. The advantage of using this method is that we are able to focus on the interview content while being lighter and more accurate in the manner in which the grammar is corrected and the interview noise and other idiosyncrasies are removed. Transcriptions were performed using an online transcription tool capable of converting speech to text with a high degree of accuracy but may not be completely accurate. To refine and correct the text, we reviewed the text after receiving them and referred to the video or audio recordings when needed to make clarifications.

Data Analysis: To examine the qualitative data collected from the interview, we adopted the thematic coding analysis method based on the inductive analysis guidance outlined by Thomas et al. [80] using the online platform, Miro.

As part of the thematic coding process, we first familiarized ourselves with the data sets by reading the transcript repeatedly and identifying valuable and interesting

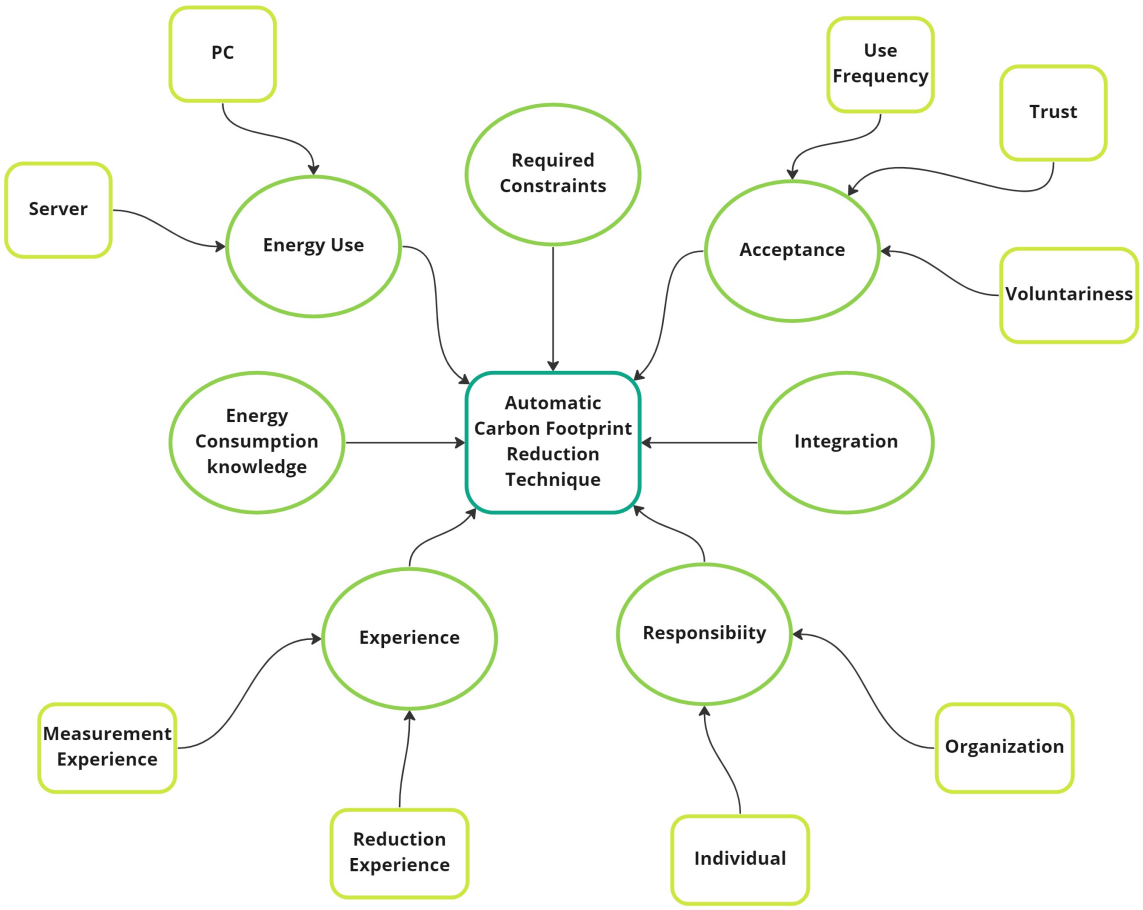

Figure 4.2: Overview of themes and sub-themes.

sentences. Afterward, we created semantic codes based on the noted contexts, in which codes capture the meaning of the data. Following the collection of codes from the noted sentences, these codes were subsequently organized and aggregated into sub-themes clustered into high-level themes. In each iteration of interviews, we modified the codes and sub-themes and paused for discussion when needed. We stopped when no new codes were found in the interview transcripts. Throughout the process of developing the themes, particular attention was paid to ensuring that each code was as accurate as possible to the original interview response.

As coding was conducted alone, the supervisor independently conducted the coding of one interview to ensure reliability. The initial coding was compared to the supervisor's coding. Only minor differences were observed. Therefore, the coding of the full set of interviews was allowed to proceed.

An overview of the thematic codes we derived from the interviews can be found in the following Fig.4.2. Seven high-level themes were identified, which were divided into sub-themes, e.g., the high-level theme Energy Use is divided into two sub-themes: PC and Server. Table 4.3 explains the themes and sub-themes.

Theme	Explanation
Energy Use	The ways that software consume energy, e.g., virtual currency mining or cloud storage.
PC	Uses of energy primarily related to client-side actions (e.g., on a PC or other device).
Server	Uses of energy primarily related to server-side actions.
Energy Consumption Knowledge	Interviewees' prior knowledge of software energy consumption and carbon footprint.
Energy Consumption Experience	Interviewees' prior experience related to energy consumption.
Measurement Experience	Interviewees' experience in measuring the energy consumption or carbon footprint of software.
Reduction Experience	Interviewees' experience in reducing the energy consumption or carbon footprint of software.
Energy Responsibility	Encompasses two sub-themes related to whether software developers should take responsibility for reducing carbon footprint.
Individual	The steps developers should take regarding energy consumption reduction.
Organization	The steps organizations should take regarding energy consumption reduction.
Integration	Methods of fitting automated carbon footprint reduction tools into the development workflow.
Use Frequency	The frequency and situations where the interviewees would apply the tool during development.
Acceptance	Interviewees' acceptance and attitudes towards the adoption of such automated energy consumption or carbon footprint reduction tools and making use of them in daily development.
Voluntariness	The acceptance and adoption of interviewees whether is voluntariness or company's requirement.
Required Constraints	The conditions that should be met for interviewees to trust and adopt automated carbon footprint reduction tools.

Table 4.3: Brief explanation of identified themes.

4.1.2 Survey

In order to get a broad range of responses from software developers, we then conducted a survey via Google Forms [22], designed based on the answers from interviewees.

Population Definition: Our population is the same as in the interviews, including professional software developers in the IT industry, researchers, and Master or Ph.D. students in the field of computer science or software engineering.

Sampling: The sampling method was again a mix of purposive and convenience sampling [26]. We sent the electric questionnaire directly to some participants and also distributed the survey on LinkedIn, Facebook, Twitter, and Mastadon. Between November–December 2022, a total of 40 respondents completed the survey.

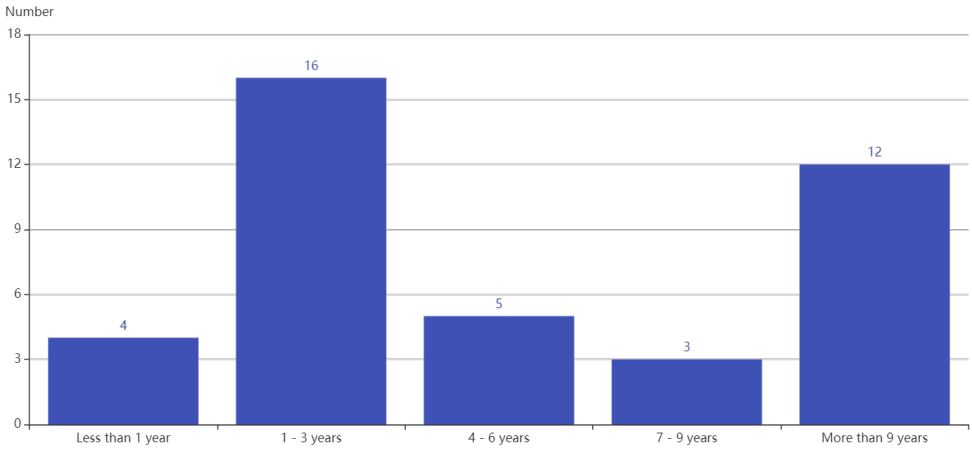

Figure 4.3: Software development experience levels of participants.

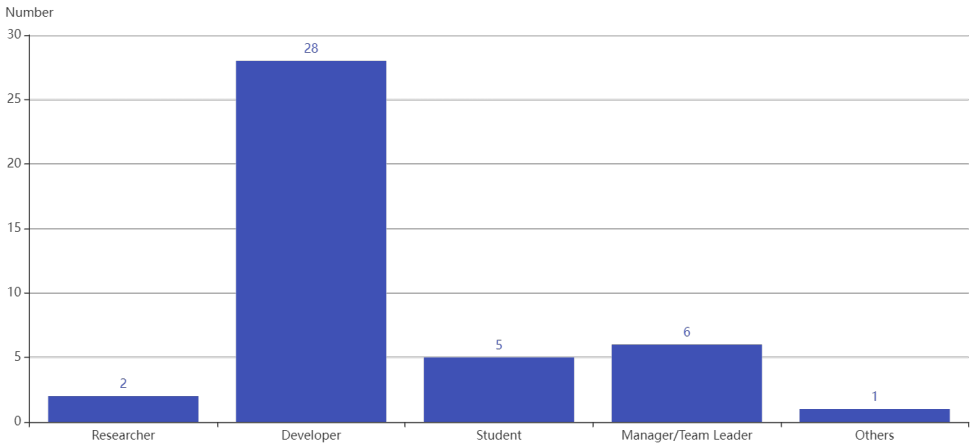

Figure 4.4: Survey participants' positions.

Participant Demographics: Table 4.4 shows demographic information on the 40 respondents, including their locations, current role, types of software they develop, and level of software development experience.

The survey participants come from various countries (e.g., Sweden, Japan, UK, USA, China, etc.) with different roles and experience levels, companies, and work for organizations including TIBCO Spotfire, Amazon, SoftBank, LINE, Ericsson, Yahoo, Red Hat, and others. 13 out of 40 participants are from Japan, and 16 from Sweden. As shown in Figure 4.4, 29 participants currently are developers, two are researchers, three are students, and six are managers or team leaders. The participants' experience in software development is presented in Figure 4.3. 40% of the participants have 1–3 years experience working in software development, and a further 30% have more than nine years of experience.

Survey Guide: To ensure a high response rate, the survey was designed to be as brief as possible, lasting between 10-15 minutes. If a survey is too long, participants

ID	Country	Position	Software Area	Exp.Levels
I1	China	Developer	Embedded system	1-3 years
I2	Japan	Developer	Sever web service	1-3 years
I3	Denmark	Researcher	Web applications	9 years+
I4	Japan	Developer	Internal tools	4-6 years
I5	Sweden	Developer	Embedded system	7-9 years
I6	Ireland	Researcher	Programming environment	9 years+
I7	Japan	Manager	Web applications	7-9 years
I8	Sweden	Student	Embedded system	4-6 years
I9	UK	Developer	Web applications	<1 year
I10	China	Student	Embedded system	1-3 years
I11	UK	Developer	Web applications	1-3 years
I12	Sweden	Student	Varies	1-3 years
I13	USA	Developer	Web applications	<1 year
I14	Sweden	Manager	Analytics	9 years+
I15	Sweden	Developer	Web and desktop applications	7-9 years
I16	Romina	Developer	Web and desktop applications	1-3 years
I17	Sweden	Developer	Data analytics	1-3 years
I18	Sweden	Developer	Varies applications	1-3 years
I19	Sweden	Manager	Web applications	9 years+
I20	Sweden	Developer	Data analysis application	4-6 years
I21	Sweden	Developer	Web applications	9 years+
I22	India	Manager	Web applications	9 years+
I23	Ireland	Developer	Windows and Web applications	9 years+
I24	Sweden	Manager	Enterprise software	9 years+
I25	Sweden	Manager	Business intelligence	9 years+
I26	Sweden	Developer	Visual analysis software	9 years+
I27	Sweden	Developer	Visual analysis software	9 years+
I28	Sweden	Developer	On-prem client, cloud service	9 years+
I29	France	Developer	Web applications	4-6 years
I30	Japan	Developer	Mobile mini applications	<1 year
I31	Japan	Developer	Embedded system	1-3 years
I32	Japan	Developer	System software	1-3 years
I33	Japan	Developer	Web applications	<1 year
I34	Japan	Developer	Web applications	1-3 years
I35	Japan	Developer	Cloud web applications	1-3 years
I36	Japan	Developer	Web applications	1-3 years
I37	Japan	Developer	Artificial Intelligence	1-3 years
I38	Japan	Developer	Web applications	1-3 years
I39	Japan	Currently unemployed	N/A	1-3 years
I40	Sweden	Developer	SAAS	1-3 years

Table 4.4: Demographic information on interviewees, including location, position, software areas, and software development experience.

may not complete it, which may negatively impact the quantity and quality of quantitative data. In the designed survey, we adopted multiple types of questions, including open-ended, multiple choice, ordinal scale, and interval scale questions [77]. Table 4.5 shows the survey questions, mapped to the research questions.

Survey Question	Research Question	Type of Question
Background of Respondents (Demographic Data)		
1.What country do you primarily work in?	N/A	Open-ended
2.What type of software does your organization normally produce, or what product domain does it primarily operate in ?	N/A	Open-ended
3.What is your current position	N/A	Multiple choice (select one)
4.How much experience do you have in software development?	N/A	Ratio scale
Knowledge and Practices		
5.How much do you know about the carbon footprint of software?	RQ1.1	Multiple choice (select one)
6.How much do you know about the energy consumption of software?	RQ1.1	Multiple choice (select one)
7.Do you agree that the carbon footprint of software is a contributor to climate change?	RQ1.2	Interval scale
8.Do you agree that the carbon footprint of software should be considered and controlled?	RQ1.2	Interval scale
9.Who do you think should be responsible for considering or controlling the energy consumption or carbon footprint of software?	RQ1.2	Multiple choice (select multiple, custom response)
10.What actions have you considered or applied to reduce the energy consumption of software?	RQ1.2	Multiple choice (select multiple, custom response)
11.Have you considered or applied any actions with the aim of reducing the carbon footprint of software? If so, please explain those actions.	RQ1.2	Open-ended
12.At what development stages did you consider or take actions to reduce energy consumption of software?	RQ1.2	Multiple choice (select multiple, custom response)
13.At what development stages did you consider or take actions to reduce the carbon footprint of software?	RQ1.2	Multiple choice (select multiple, custom response)
14.How did you assess the energy consumption of the software?	RQ1.2	Multiple choice (select multiple, custom response)
15.How did you assess the carbon footprint of the software, if you did so separately from energy consumption?	RQ1.2	Multiple choice (select multiple, custom response)
16.What techniques have you used to evaluate energy consumption of software?	RQ1.2	Multiple choice (select multiple, custom response)
17.If you have evaluated energy consumption, what process do you follow to design test cases, and how did you perform and evaluate the results?	RQ1.2	Open-ended
18.What techniques have you used to evaluate the carbon footprint of software?	RQ1.2	Multiple choice (select multiple, custom response)
19: If you have evaluated carbon footprint, what process do you follow to design test cases and how did you perform and evaluate the results?	RQ1.2	Open-ended
Automated Improvement of Carbon Footprint		
20.Would you be willing to use an automated tool to reduce carbon footprint?	RQ1.3, RQ1.5	Interval scale
21.How skeptical are you of the ability of an automated tool to successfully and adequately reduce carbon footprint?	RQ1.3, RQ1.5	Interval scale
22.How likely are you trust a tool to make automated code changes in service of reducing carbon footprint?	RQ1.3, RQ1.5	Interval scale
23.What conditions must be met for you to trust and use such a tool?	RQ1.3	Multiple choice (select multiple, custom response)
24.What do you feel would encourage you or other developers to adopt such a technique voluntarily?	RQ1.5	Multiple choice (select multiple, custom response)
25.How often would you use this tool?	RQ1.4	Multiple choice (select one, custom response)
26.How would you imagine such a tool fitting into a development workflow?	RQ1.4	Multiple choice (select multiple, custom response)

Table 4.5: Survey questions, mapped to the research questions, with question type.

Data Collection: In order to collect in-depth information, the partially-structured questionnaire design method was adopted to ensure participants had enough freedom to express their opinions. The questionnaire is divided into four parts. The first part describes the background and purpose of the project, as well as the number of questions and the estimated time required to complete the survey. In the first set of questions (Questions 1-4), the questionnaire obtains information about the participant's demographic, the type of software they produce, their current position,

and their experience in software development. We gathered information about participants' knowledge and practices regarding software carbon footprint and energy consumption in the second set of questions (Questions 5-19), including their familiarity with energy consumption and carbon footprint, their opinions on responsibility for controlling these factors, their consideration of—or application of—actions to reduce the energy consumption and carbon footprint, and methods to measure these factors. In the final set of questions (Questions 20-26), we collected their opinions about automated tools for carbon footprint improvement, including their willingness to use such a tool, their level of trust in such techniques, and the conditions required before utilizing such techniques.

Data Analysis: The descriptive statistics analysis methodology [44] was adopted for quantitative data. In addition, qualitative data was incorporated into our previous thematic mapping from the interviews.

4.2 Automated Carbon Footprint Reduction Tool (RQ2-3)

This sub-project involves the application of design science methodology to develop a prototype of an automated tool for reducing the carbon footprint of webpages. As a result of this tool and the design process, RQ2-3 can be addressed. In design science research, researchers are guided and led to identify and solve problems using a systematic approach. By creating artifacts and then evaluating those artifacts, they are able to solve the problems they have identified [83]. Here, we followed the process proposed by Peffers et al. [60]. The design science process is shown in Figure 4.5.

As part of the design science process, we first identified the problem and defined our research objectives. Using genetic programming, we aim to design a tool that automatically reduces webpage carbon footprint. To achieve this goal, we designed a prototype tool in iterative steps. In each iteration described below, additional functionality was added to the tool and incremental evaluations were conducted. The final step (activity 6) involved analyzing the results and observations obtained during the tool evaluation.

4.2.1 Incremental Prototype Tool Design

This prototype tool was designed over three iterations. In Iteration I, we identified methods to measure the carbon footprint of a webpage and developed an initial set of fitness functions. In addition, this iteration focused on identifying actions that affect webpage's carbon footprint.

In Iteration II, genetic programming—based on the NSGA-II algorithm—was utilized to develop the automated carbon footprint reduction tool. Additional actions have been added to the tool to reduce carbon footprints.

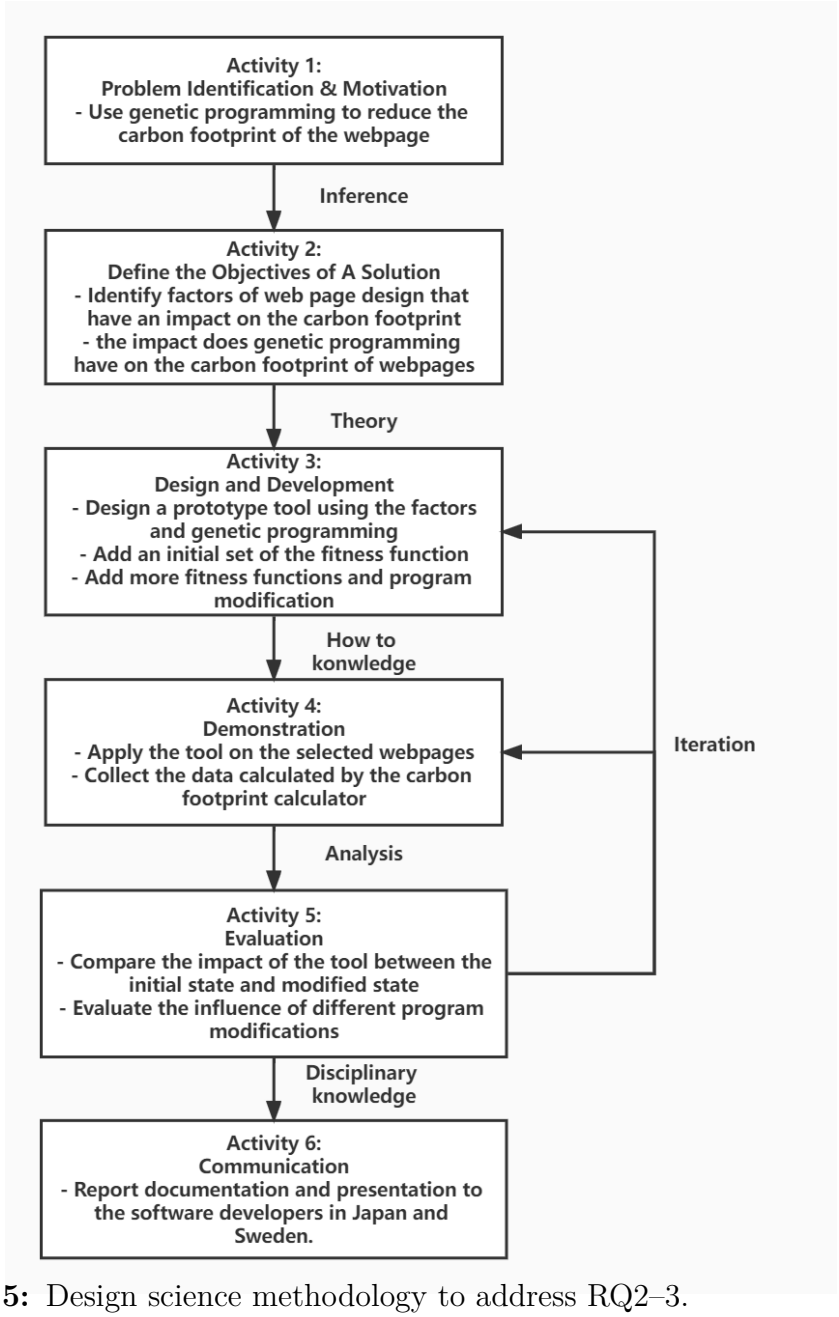

Figure 4.5: Design science methodology to address RQ2–3.

In Iteration III, we conducted parameter tuning experiments to determine which configuration of the automated carbon footprint reduction tool performs best. Following the identification of the tool parameters, we conducted the final experiments comparing the tool against two baselines, the original webpage and random solution generation.

4.2.2 Iteration I (Initial Prototype)

The purpose of this iteration was primarily to develop an initial prototype tool. Firstly, it was essential to determine how the carbon footprint of the web page should be calculated, i.e., potential fitness functions to optimize during solution generation. We examined factors such as CPU usage, RAM usage, and quantity of

data transferred and how they can positively or negatively affect carbon footprint.

Following the identification of the initial fitness functions, we investigated how elements of the webpage source code (e.g., HTML, CSS, and JavaScript) contribute to the reduction of the carbon footprint. For example, removing whitespace from an HTML file and changing the background color could potentially reduce webpage carbon footprint. To determine whether the fitness functions are able to measure distinctively changes that may reduce carbon footprint, we used the initial fitness function to evaluate some potential page modifications.

By modifying HTML, CSS, and JavaScript files, the webpage carbon footprint can be automatically reduced. This includes compression of image files, altering the image format, modifying HTML tags, removing CSS effect properties, etc. The updated HTML, CSS, and JavaScript files are generated by running `Node.js` scripts that change the original source code of these files. For evaluating the effects of these actions, this tool used the quantity of transferred data, memory usage, page loading time, and the number of changes as fitness functions.

4.2.2.1 Identification of Factors and Design of Fitness Functions

Reduction of carbon footprint is actually a multi-objective optimization problem. We hope to minimize the webpage's carbon footprint. In addition, we would like to ensure that we do not negatively affect other performance aspects, such as page load time. The user experience on the website is greatly impacted by this factor. Therefore, a variety of fitness functions were considered from a variety of perspectives. Some of these functions may conflict with each other. Our approach to achieving a multi-objective solution model requires us to identify different objectives and analyze the effects of divergent combinations of fitness functions on the performance of reducing a website's carbon footprint.

To identify factors affecting a webpage's carbon footprint, it is necessary to identify factors affecting its energy consumption. It is important to take into account HTML, CSS, and JavaScript elements. For the purpose of finding out which elements of the webpage have a significant impact on the carbon footprint of the website, we read and applied several methods mentioned in related academic papers. After identifying factors that affect the webpage's carbon footprint, we can apply them in the design of the automated carbon footprint reduction tool.

4.2.2.2 Fitness Functions

Initially, we identified five measurements to use as fitness functions—memory usage, CPU usage, the quantity of transferred data, the webpage loading time, and the number of changes made to a solution.

Memory Usage: In a computer, RAM is used to temporarily store data and information that users access, click on, or browse when they visit websites. According to the results of the research that calculated the energy consumption of websites written by Olivier et al. [64], the correlation coefficient for energy and RAM is 0.77,

which means that the relationship between RAM usage and energy is strong. There is a correlation between the number of requests, the page size, and the consumed RAM on the client. Therefore, we consider a client's RAM usage while viewing a webpage as one fitness function for assessing webpage factors' carbon footprint reduction.

As a means of monitoring memory usage, we use the Python package `Psutil` (Python system and process utilities) [69], which is a cross-platform library used to retrieve information (including CPU and memory utilization) about running processes. As for RAM of the webpage process, we use USS (Unique Set Size) because it is the memory that is unique to a process and which would be freed if the process ended. As compared to RSS (Resident Set Size), USS is more accurate since RSS includes both the memory that is unique to the process as well as memory that is shared with other processes. Thus, USS is able to perform the actual process of memory information.

`Selenium`, a suite of tools for automating web browsers, is used to simulate the user environment of the webpage to ensure the data is accurate and realistic [9]. In `Selenium`, the user can locate the webpage elements, click on the buttons, scroll up and down, and fill in the frame. Therefore, we can obtain as much accurate and actual information on RAM usage as possible, which will contribute to a more convincing result.

CPU Usage: The purpose of a CPU is to execute processes with varying levels of complexity and speed. CPU utilization refers to the percentage of time that the CPU is dedicated to performing its functions. As CPU usage increases, CPU energy consumption increases as well. In addition, other researchers also use CPU usage to measure energy consumption, such as Yuhao et al. [93] and Yicao et al. [10]. After several experiments and inquiries, we found that it was difficult to measure and isolate the CPU usage of the webpage accurately. As a result, we decided to abandon this factor.

Quantity of Transferred Data: Wholegrain Digital, a consultancy company, have developed a "Website Carbon Calculator" [12]. Their calculation is primarily based on the transfer of data to the end user, arriving at a carbon footprint calculation in kWh/GB as the unit of measurement—also the unit of measurement used in most studies on energy consumption.

The first step of their calculation is to determine how much data has been transferred (GB). The amount of energy consumed during the loading of a website is approximately proportional to the amount of data transferred. To calculate this, we utilize the `PageSpeed Insights API` (PSI) [35] to get the bytes for loading pages. A PSI report provides information on the performance of a page on both mobile and desktop devices, as well as suggestions for improving the page. PSI analyzes a URL and generates a score that estimates the performance of the page according to a number of metrics using Lighthouse. In the various subproperties, it contains metadata about the run and the results, including audits of performance, accessi-

bility, progressive web applications, and more. Using SPI, we can determine how many bytes are consumed during loading.

After getting the total transfer data weight, the Website Carbon Calculator identifies the energy intensity of the web data (kWh/GB). They get the related parameters from the raw data provided by Andrae et al. [4]. Energy consumption is associated with data centers, networks, and end-user devices, such as computers, tablets, and smartphones. To provide a uniform assessment of the energy source, it was assumed that all websites tested used standard grid electricity to transfer information through networks and to power their users' devices. To assess the environmental impact of data centers, they examined whether servers are housed in an environmentally friendly facility, which entails taking a number of measures to reduce their energy consumption. As a result, carbon emissions could be reduced proportionally. Four aspects were considered to contribute to energy consumption:

1. **Network use:** transferring data through network is responsible for about 14% of the energy of the whole system.

2. **Data center use:** energy used to store and operating data accounted for approximately 15%.

3. **Hardware production:** energy used in the creation of embedded chips accounted for 19%.

4. **Consumer device use:** users who visit the website and interact by scrolling or clicking account for 52% of the energy.

In the original calculator, the carbon footprint is calculated based on the carbon intensity of electricity (g/kWh). For carbon intensity, the default value is the global average carbon intensity of electricity (442 g/kWh). The following equation can be used to express the calculation method:

$$\begin{aligned} The\ number\ of\ carbon\ footprint(g) = Webpage\ loading\ data\ bytes(GB) \\ \times Energy\ intensity\ of\ web\ data(kWh/GB) \\ \times Carbon\ intensity\ of\ electricity(g/kWh) \end{aligned} \tag{4.1}$$

The energy consumed per visit, including the first-time visit and the returning visit, can be performed by using the following equation:

$$\begin{aligned} Energy\ per\ visit(kWh) &= Data\ Transfer\ per\ visit \\ &\times Energy\ intensity\ of\ web\ data \times First\ time\ visitors(0.75) \\ &\oplus\ Data\ Transfer\ per\ visit \\ &\times\ Energy\ intensity\ web\ data \times Returning\ visitors(0.25) \times Loaded\ data(0.02) \end{aligned} \tag{4.2}$$

In our current study, we do not consider the location of the consumed energy. Therefore, we only use the quantity of data transferred as a measurement. In future work,

we will utilize the full calculation to consider the location of page elements as an additional factor.

Page Load Time: Page load time (PLT) is one metric for evaluating user experience from the perspective of web performance. PLT is pivotal to webpage performance because page load time greatly affects bounce rate. According to Daniel et al. [2], more than 40% of website users may leave if the loading time exceeds three seconds and more, and 90% of website users leave if the loading time exceeds five seconds. In addition, Pourghassemi et al. [74] demonstrate that PLT can negatively impact conversion rates, leading to fewer customers ordering on an e-commerce platform.

PLT and energy are distinct metrics. Yi et al. [10] observed that these two metrics are not always correlated. Thus, we need to ensure performance is not negatively impacted while reducing the carbon footprint. We apply `Selenium` and the `PSI` in the study to capture perceived page load time statistics. Time is measured in milliseconds (ms).

Number of Changes: In addition, the automated carbon footprint reduction tool should not overtly change the original code, as this may negatively affect the user experience. Therefore, the number of changes was also chosen as a fitness function, with the goal of minimizing the number of changes.

4.2.2.3 Webpage Modifications

As a result of differences in HTML, CSS, and JavaScript files, different web pages consume different amounts of energy and have different carbon footprints. According to our analysis, we identified a set of potential actions that may impact the carbon footprint of a website.

Optimize Images: Images typically account for most of the downloaded bytes on a page, and rendering images while loading webpages is energy-consuming. Thus, optimizing images can reduce transferred data bytes and improve performance, resulting in the device using less memory to store the image files and less CPU usage on executing the corresponding process. Two ways to optimize images include compressing and changing image formats.

Change Image Format: Two main graphics formats are performed on the webpages—vector graphics and raster graphics. The former is suited for images that consist of simple geometric shapes such as logos, text, or icons. The latter shows more than the vector graphics when the scene is complex, for example, a scenery or an ID photo, including PNG, JPG, and BMP formats. Thus, here we pay attention to changing raster formats.

Narendran et al. [79] find that the image format's energy consumption also varies. JPEG compresses the image better, rendering it faster than PNG and GIF formats. In addition, WebP images are smaller than their JPEG and PNG counterparts, can

decrease page sizes, and improves performance. When Facebook switched to WebP, they experienced a 25-35% reduction in file size for JPEGs and an 80% reduction for PNGs. This factor is therefore incorporated into the automated carbon footprint tool as an action.

Compress Images: In addition to changing the image to other formats, we can compress the image itself. Compressing images can also reduce energy consumption [63]. On some websites, the image quality is not essential for the users to look through them, as users may not find the difference between the original version and the compressed version from human eyesight. Thus we can reduce the image file's size by sacrificing the image resolution. And this factor is incorporated into the automated carbon footprint reduction tool.

Optimize HTML:

Remove HTML Whitespace: Blank and indent do not influence the HTML code, but account for bytes. Removing whitespace in the source code can reduce the HTML file size and not introduce new faults. This operation could be conducted before deploying the website to the server; in this way, the readability of HTML would not be impacted negatively. However, due to the potential for reducing readability at other times, we did not utilize this action in our initial prototype tool. We will consider adding it in the future.

Change HTML Tags: Adrian et al. [72] have found that `<strong>` elements use less energy and load faster than unstyled `<span>` tags; the former used about 14% less loading time and energy, and `<span>` tags are particularly inefficient in mobile device browsers. Thus, we change the `<span>` tag in HTML to `<strong>` automatically using the Abstract Syntax Tree (AST) of the HTML file. Thus, changing HTML tags is incorporated into the automated carbon footprint reduction tool.

Optimize CSS:

Remove CSS Whitespace: Blank and indent do not influence the CSS style sheet code but account for bytes. Removing the whitespace in the source code can reduce the CSS file size and not introduce new bugs. Due to the potential for reducing readability, we did not utilize this action in our initial prototype tool. We will consider adding it in the future.

Remove Unused CSS: Unused CSS rules contain CSS declarations not used anywhere on a web page. It is normal for some of the CSS not to be used. For example, some CSS declarations are written for another device, such as a desktop or mobile device. Sometimes, this page does not use an element described in the style sheet (such as a button). The loading of a page is slowed down by CSS. An HTML file is first fetched from a page when it is loaded by the browser. This HTML is converted into DOM Nodes. Afterwards, all stylesheets are fetched by the browser. In addition, the CSS files contain stylesheets converted to a different format, CSSOM [45]. A render tree is constructed by combining the DOM and CSSOM. The first content is

produced by a browser only after this render tree has been constructed.

Because of this mechanism, CSS files heavily affect the rendering time and quantity of data transferred for a webpage [21]. When a CSS file contains unused CSS, it takes longer to build under the render tree. Thus, we remove unused CSS rules automatically.

Remove CSS opacity properties: The opacity property can add complexity to rendering the webpage performance. According to the research of Adrian et al. [72], translucent elements are rendered more slowly than opaque elements. Consequently, removing unnecessary opacity effects can reduce energy consumption, and this is incorporated into the automated carbon footprint reduction tool.

Optimize JavaScript:

Move JavaScript to the Bottom: Moving the JavaScript `<script>` to the end of page `body` can improve webpage performance by decreasing the time to first render because the scripts block parallel downloads, e.g., image files. In order for the whole page to continue running, the JS code execution must be completed. It is possible to reduce the amount of energy consumption and time required for HTML and CSS code to run by executing the scripts at the end of the HTML [39]. Thus, this action is incorporated into the automated tool for reducing carbon footprints.

Add Script Tag: According to Zakas et al. [91], `async` and `defer` are two new attributes for HTML5. JavaScript's `async` script tag enables scripts to be loaded asynchronously. As a result, if a script is `async`, it will load independently of other scripts on the page and will not impede loading. Loading several external scripts asynchronously can increase page loading speed because the browser can download and execute them simultaneously. Using the `defer` attribute in HTML, the browser will load the script after parsing (loading) the page. Scripts that are dependent on other scripts or scripts that need to be loaded after the initial page load may benefit from this technique. However, we determined through initial experimentation that these changes could affect the behaviour of a webpage in significant and inadvertent ways. Therefore, we do not impose such changes with our automated tool at this time.

4.2.3 Iteration II (Design of the Tool):

In the second iteration, we implemented a genetic programming model based on the fitness functions and actions from Iteration I. The purpose of this tool is to generate modified versions of webpages based on multi-objective optimization, reducing the carbon footprint without negatively impacting other selected attributes. We constructed this tool based on the NSGA-II algorithm (Non-Dominated Sorting Genetic Algorithm-II) [20].

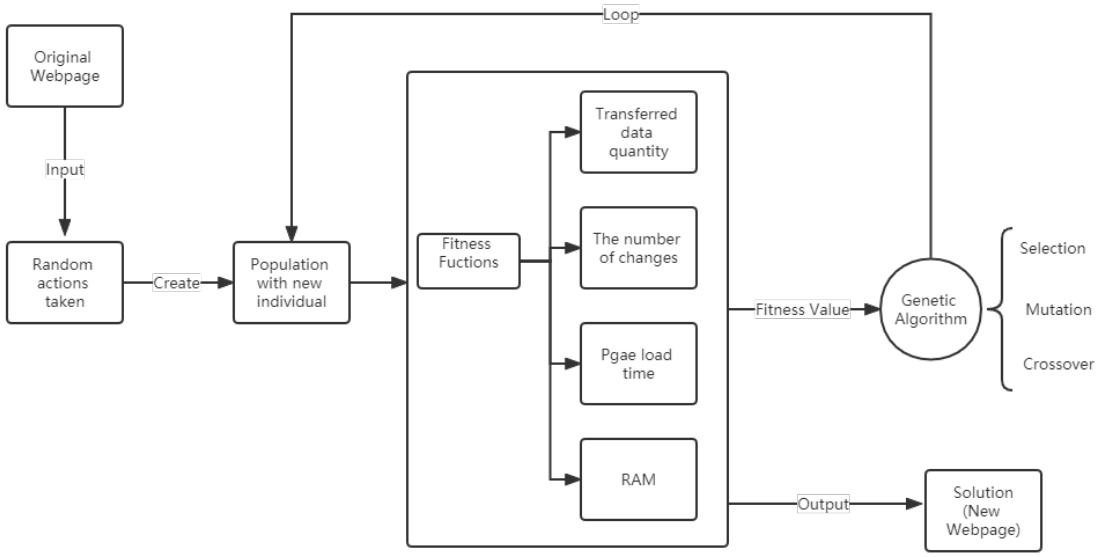

Figure 4.6: Architecture of the automated carbon footprint reduction tool.

4.2.3.1 Tool Overview

We use the automated carbon footprint reduction tool to investigate the effects of optimizing multiple fitness functions on the carbon footprint of webpages. In the case of multi-objective optimization, multiple performance objectives can be targeted simultaneously, such as the reduction of data transfer quantity, page loading time, and other factors. The tool supports four fitness functions from the set investigated in Iteration I—data transfer quantity, memory usage, page loading time, and the number of changes made to the page.

Figure 4.6 illustrates the overall architecture of the tool. The tool generates a population of updated webpages consisting of different combinations of changes made. Each individual in the population represents a version of a webpage. This population is evolved over a series of generations. By default, every generation contains a population of 20 individuals.

Based on the architecture, the random action module receives the original webpage's input and generates the modified webpage with random actions taken. Afterwards, the modified webpage is passed through the fitness functions module to produce the fitness value. According to the fitness values evaluated, the NSGA-II algorithm with genetic algorithm selects the individuals with the best performance and generates the offspring using variation operations. After the search budget has been consumed, the framework outputs the webpage with the best performance.

4.2.3.2 NSGA-II Implementation

We adopted the NSGA-II (Non-dominated Sorting Genetic Algorithm II) algorithm, which is a multi-objective evolutionary algorithm (MOEA) [20]. Our choice of the NSGA-II algorithm is because the automated carbon footprint reduction tool involves multiple objective functions, including data transferred quantity, page loading time, memory usage, and the number of changes to solutions. Through NSGA-II, optimal solutions can be identified by maximizing or minimizing each objective.

Algorithm 1: NSGA-II Algorithm

Input: Webpage document W, population size $popSize$, generation number $maxGen$, selected fitness functions $F \leftarrow$ {f1, f2, f3,..., fn}

Output: Pareto Front PF, Improved webpage document W_n

Initialization: generation $gen \leftarrow 0$, population P_0;
while $gen <= G_n$ **do**
  $fitnessValueSet \leftarrow \phi$;
  $fitnessValueSet \leftarrow$ Evaluate $P_0(P_g)$ with F;
  $NDS \leftarrow$ nonDominated Sorted Solution($fitnessValueSet$);
  $CDV \leftarrow$ crowdedDistanceSorted(NDS);
  $newParent \leftarrow$ tournamentSelection(NDS, CDV);
  $newChild \leftarrow$ variationOperation($newParent$);
  $newPop \leftarrow P_0$;
  while $size(newPop) < 2 * popSize$ **do**
    $newPop$.append($newChild$);
  end
  $newFitnessValueSet \leftarrow \phi$;
  $newFitnessValueSet \leftarrow$ Evaluate $newPop$ with F;
  $newNDS \leftarrow$ non-Dominated Sorted Solution($newFitnessValueSet$);
  $newCDV \leftarrow$ Crowded Distance Sorted($newNDS$);
  $P_g \leftarrow \phi$;
  for *each front level* f *in* $newNDS$ **do**
    if $P_g + f >= popSize$ **then**
      break;
    else
      $P_g \leftarrow P_g \cup f$;
    end
  end
  $P_g \leftarrow P_g \cup f[1 : (popSize - size(pg)]$;
  $gen = gen + 1$;
end
return (PF, W_n);

Furthermore, since this is a complex nonlinear problem, the search space is not small and there is more than one objective, linear analytical methods cannot be used to solve the optimization problem. Thirdly, the NSGA-II algorithm is capable of optimizing non-precise objectives using approximate methods.

By employing a non-dominated sorting approach, NSGA-II maintains diversity in the population by ranking candidate solutions according to their dominant relationships with one another. In addition, the algorithm incorporates a crowding distance operator to encourage solutions to be spread across the Pareto frontier—the set of solutions where a change to any objective would worsen the attainment of another objective.

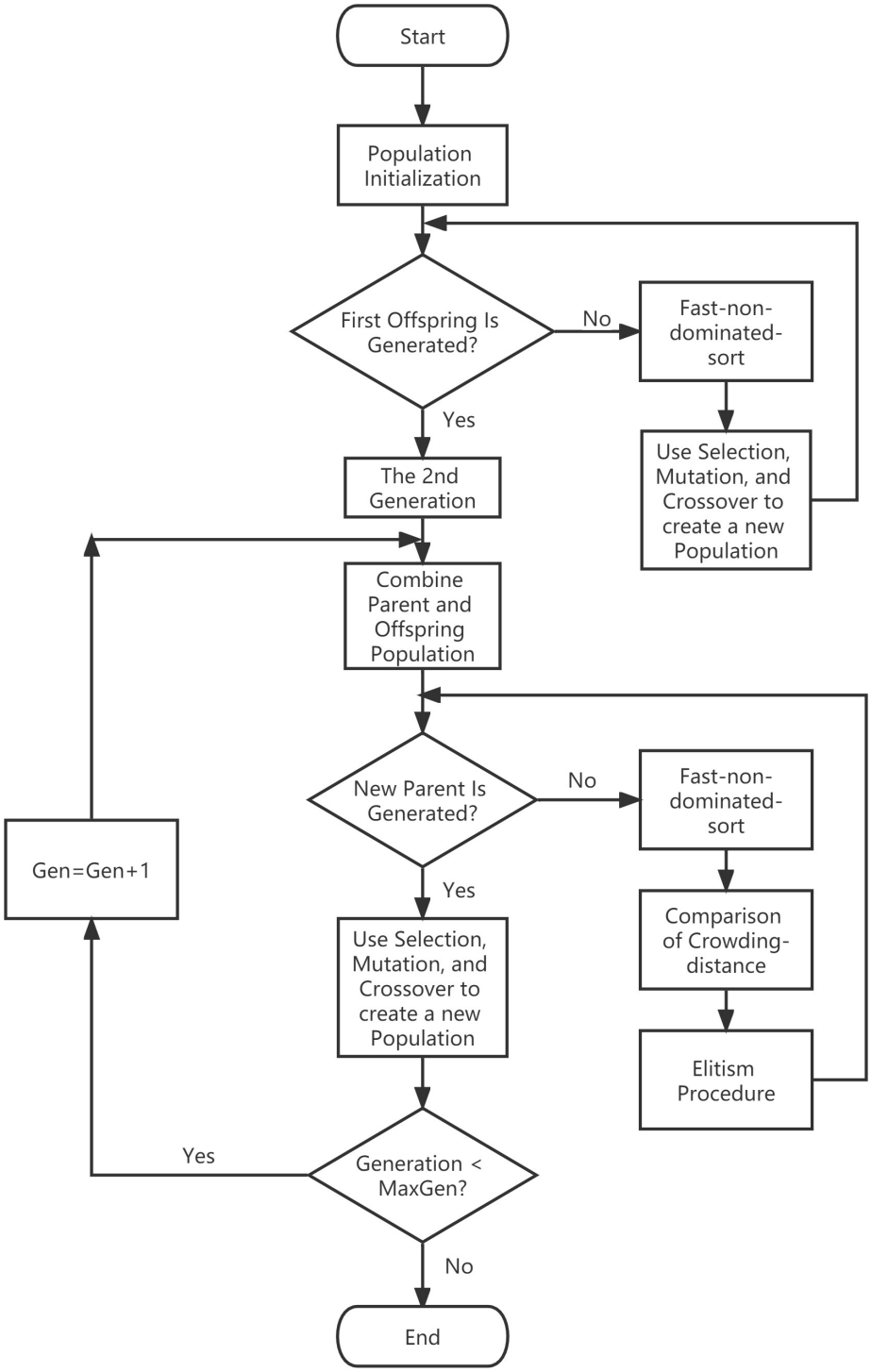

Figure 4.7: NSGA-II flow chart.

The overall process of the NSGA-II algorithm is illustrated in Figure 4.7 and Algorithm 1. The NSGA-II algorithm begins with random solutions and divides the population into multiple fronts based on the degree of dominance between the individuals. In a non-dominated sorting approach, each individual is evaluated according to how many others in the population dominate it and how many others it domi-

nates. If no individual dominates another individual, that individual is considered to be in the first non-domination level, or Pareto frontier, whereas if only one individual dominates another individual, that individual is considered to be in the second non-domination level, and so on.

After individuals have been sorted into different non-dominated levels, a crowding distance metric is used to select individuals from the same non-dominated level for the next generation. The crowding distance is the distance between neighbouring individuals who are at the same level. Selection of individuals with high crowding distances is preferred because they provide a diverse set of solutions covering a larger area of the front, which can contribute to a more diverse future generation and help to avoid getting stuck in a local optimum. The density estimation method estimates the crowding distance between each individual in the population. Therefore, at the first front of non-dominance levels, the individual with the highest crowding distance is chosen.

A key strength of the NSGA-II algorithm is that it preserves elitist solutions while maintaining a diverse pool of solutions. NSGA-II preserves the best solutions from previous generations and prevents the algorithm from becoming stuck in a local optima because it combines parent and offspring populations and performs non-dominated sorting on the combined population. As a result of this strategy, the loss of elitist solutions in the parent generation is reduced.

To summarize, the NSGA-II algorithm has two pivotal attributes for every individual I in a population P:

1. Non-domination rank, the *I-rank*, is a measure of the level of non-domination of the individual within the population. Individuals with lower ranks are preferred over those with higher ranks, as they represent better-performing solutions.

2. Local crowding distance, the *I-distance*, is a measure of how far apart an individual is from its neighbours. Individuals with higher crowding distances are preferred over those with lower crowding distances, as they represent more diverse solutions that cover a larger region of the front.

To develop the automated carbon footprint reduction tool, NSGA-II was selected due to its superior computational complexity, ability to handle elitist solutions, and ability to provide diversified Pareto-optimal solutions. In the following paragraphs, we discuss the details of the NSGA-II implementation, including the Initial Population, the Generations of the Algorithm, the Fitness Evaluation, Candidate Selection, and Variation Operators.

Initial Population: An individual is defined as a modified version of the webpage. The initial population *P0* is created randomly by generating a sequence of solutions. Each individual's fitness value is computed. Upon evaluating an individual's fitness, variation operators (i.e., crossover, mutations, and reproductions) are used to generate a population of offspring with the same size as the original population.

After the offspring population is created, it is combined with the parent population and handed down to the following generation.

Generations of the Algorithm: The tool iterates over a series of generations. An upper limit is specified for the number of generations. Based on the new offspring population, a fitness value is calculated for each individual in the next generation.

NSGA-II uses the selection operator to select the most qualified candidates from the population. In each generation, the parent and the offspring populations are combined, which is called the $\mu + \lambda$ genetic algorithm. This is after the offspring population's fitness is evaluated. Through non-dominated sorting, the selection operator ranks all individuals into different Pareto front levels. The selection operator calculates each individual's crowding distance to preserve diversity and avoid conflict between individuals who have the same Pareto front level. The crowding distance estimates the density of candidate solutions surrounding a particular solution or individual in a population. Individuals for the next generation are selected from the Pareto front based on their front level. To create the next population, test cases with higher non-domination ranks are selected (i.e., lower front-level values), and if two test cases have the same non-domination rank, the selection operator favors the test case with a greater crowding distance (i.e., less dense).

Each generation, the parent population, and the offspring population are combined with maintaining the diversity of Pareto-optimal solutions. Each generation follows the same procedure until the maximum number of generations has been reached. The model outputs the population of the final generation upon reaching the maximum number of generations, in which the best solutions are ranked from lowest carbon footprint score to highest.

Fitness Evaluation: Four fitness functions are implemented in the tool—the quantity of transferred data, the page load time, the number of changes, and the memory usage. NSGA-II is generally considered efficient for problems with up to three objectives [38]. The algorithm may become less efficient as the number of objectives increases, as objectives are more likely to conflict and identifying a Pareto frontier becomes more computationally inefficient. Thus, we only use up to three fitness functions at any one time. In our experiments, we employ the following two combinations of fitness evaluation:

- (Data Transferred Quantity) + (Number of Changes) + (Page Load Time)
- (Data Transferred Quantity) + (Number of Changes) + (Memory Usage)

Candidate Selection: The best combination of actions is identified by combining the evaluated offspring with the parent population and using a non-dominated sorting strategy. Based on their Pareto front rankings, the combined population is sorted into non-dominance levels. Individuals who belong to the first non-dominance level ($F1$) are selected since they represent the best non-dominated solutions in the population. When the size of the selected population ($P1$) is not sufficient for the

specified size of the population, the solutions in the second non-domination level (*F2*) are selected and added to *P1*. In the current non-dominated level (*F1*), the solutions are sorted by their crowding distance in descending order. If *P1* still has k solutions to be filled, the first k solutions from *F1* are added to *P1*. Each time an individual is selected and added to *P1*, its corresponding crowding distance is computed.

In each generation, the parent and offspring populations are combined for selection. The tournament selection method was used here to select the parent population for creating the offspring. In a K-way tournament selection, K individuals are selected and a tournament is conducted among them. Among the selected candidates, only the fittest candidate is selected and passed on to the next generation, and the K may always select two candidates from the compared subset of the population. A combination of the parent and offspring populations reduces the loss of elitist solutions from the previous generation and maintains the diversity of Pareto-optimal solutions. Until the maximum number of generations is reached, the same procedure of selecting the best solutions from the combined parent and offspring populations is repeated. Through this iterative process, the algorithm can reach a set of Pareto-optimal solutions that offer the best trade-offs between the different objectives.

Variation Operators: In each generation of the genetic algorithm, mutation and crossover are utilized to improve the current solutions. An array containing 0 and 1 represents the elements and actions that have an impact on the carbon footprint. A value of 0 means to keep the original version of the element from the code retained, while a value of 1 means to automatically modify the element using the appropriate action. For example, whether to compress an image or not. As a result, the combination of actions applied can be represented as an array. A small change is introduced to the current solution's array as a result of the operation mutation. It selects one action randomly and overturns the value in an array, i.e., replaces 0 with 1 or replaces 1 with 0. Each mutation operation modifies only one element of the array at a time. In this manner, a new array representing a new combination of actions is created.

A crossover process produces a new array of actions by exchanging two parent array elements twice in a single operation. In order to identify the exchange location, we generate a random number, then exchange the parent *P1* and *P2*'s actions array's values, and repeat the above procedure. By merging parents, a new array of actions is created.

4.2.4 Iteration III (Refinement and Final Evaluation):

The automated carbon footprint reduction tool was completed in the last iteration by performing parameter tuning. We then evaluated the tool empirically to compare the impact of the tool on reducing carbon footprint, as well as examining the influence of the different combinations of fitness functions and program modifications.

Levels	1	2	3
Variables	Value		
Population Size (Npop)	10	15	20
Number of Generations (Gn)	10	15	20

Table 4.6: NSGA-II parameters considered during parameter tuning.

4.2.4.1 Parameter Tuning

For tuning two parameters of NSGA-II—the population size (*Npop*) and the number of generations (*Gn*)—we adopted the Taguchi method [37]. In the Taguchi method, fractional factorial experiments (FFEs) are used to reduce the number of experiments required to identify significant influences on the response. Taguchi's method suggests two ways to analyze the results. In experiments that repeat once, analysis of variance is used, and in experiments with multiple runs, signal-to-noise ratio (S/N) is used. In this research, S/N was used to analyze the results since the algorithm requires multiple runs to obtain better solutions [76].

Table 4.6 shows the two parameters, each with three levels (P = 2, L = 3), used to run the experiments based on the Taguchi orthogonal arrays design. This Taguchi orthogonal array is used when all L levels of each (P-1) other parameter are tested at least once for each level of a particular parameter.

To evaluate a multi-objective evolutionary algorithm (MOEA), there are two main properties we need to evaluate:

1. Closeness to the true Pareto front.

2. Diversity of the solutions on the Pareto front.

Thus, several metrics are proposed in the literature to evaluate MOEAs. We have adopted three measures—the best solution (*Best Sol*), the hypervolume (*HV*), and the generational distance (*GD*), are used in this paper.

1. Best Sol: To determine the Best Solutions, the set of solutions is first normalized using the linear dimensionless method in Eq. 4.3 in multi-attribute decision-making (MADM) [66].

$$z_i' = \frac{z_i - z_i^{\min}}{z_i^{\max} - z_i^{\min}} \tag{4.3}$$

For each of the three fitness values, we sum the quantity of data transferred, the number of changes, and one other fitness value (page load time or RAM), each of which is weighted 0.4, 0.3, and 0.3. The solution with the highest sum is selected.

2. HV: Hypervolume is a popular performance metric used to evaluate the effectiveness of NSGA-II. The objective of this measurement is to determine how well a set of solutions covers the domain of the objective [6]. The following steps are followed to calculate the hypervolume:

1. Define a reference point. The reference point is the point in the objective space that represents the ideal solution, which is usually the solution that dominates all others. In this case, we chose (0, 0, 0) as the reference point because the objective of this algorithm is to minimize the fitness functions of all solutions—representing an ideal solution.

2. Sort the solutions. In the population, we sort the solutions according to their non-domination rank and crowding distance. In the NSGA-II algorithm, this step is completed during execution.

3. Calculate the hypervolume. The following formula can be used to calculate the hypervolume contribution of a solution:

$$\mathrm{HV}(S) = (r1 - x1) * (r2 - x2)^* \ldots ^*(rn - xn) \tag{4.4}$$

   In this equation, S represents the solution, r represents the reference point, x represents the objective vector of the solution, and n represents the number of objectives. In addition, the objectives have different units and scales. Therefore, the objectives must be normalized before the hypervolume can be calculated.

The hypervolume can be viewed as a measure of both convergence and diversity. In general, hypervolumes of greater magnitude indicate a better approximation of Pareto's frontier [6].

3. GD: An evaluation of the quality of a set of solutions within a non-dominated set is based on the generational distance. Using this method, the distance between the solutions in the obtained set and the true Pareto front is calculated. The following steps are used to calculate the GD metric:

1. Utilize NSGA-II to identify non-dominated solutions.

2. Determine the true Pareto front in the objective space. We define a reference front by extracting a single Pareto front from the collected solutions since the true front cannot be obtained.

3. The Euclidean distance between each non-dominated solution and its nearest neighbour on the true Pareto front should be calculated.

4. Using the distances obtained above, calculate the average distance.

5. Calculate the square root of the result to obtain the GD metric.

When the GD value is lower, the solutions are closer to the true Pareto front, which is a desirable property for multi-objective optimization algorithms.

To maximize the algorithm's efficiency, the Best Sol, and HV, with higher values, and GD, with lower values, demonstrate better performance [92].

On the basis of Taguchi's (L9) design, the coded NSGA-II model is run three times

for each combination of fitness functions for each experiment subject [14]:

- Combination-1: Transferred Data + Number of Changes + Page Load Time
- Combination-2: Transferred Data + Number of Changes + Memory Usage

As mentioned above, the three chosen metrics are first normalized and then merged into a single value, which will serve as the response in the Taguchi method by summing the three normalized metrics. The Taguchi method aims to find the maximum S/N defined in Eq. 4.5.

$$\frac{\mathrm{S}}{\mathrm{N}} = -10 \log \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{Sum_i^2} \right) \tag{4.5}$$

Experiment Subjects: Five different webpages were identified as experimental subjects in the parameter tuning experiments from GitHub, with different degrees of complexity. These projects were also used in the final experiments, and are the first five subjects listed in Tables 4.7, 4.8, and 4.9. Table 4.7 includes the project name and its Github link. Table 4.8 shows the baseline fitness values for data transferred, page load time, and memory usage. In addition, Table 4.9 presents metadata on each project. The results of parameter tuning will be discussed in Section 5.2.1.

4.2.4.2 Final Evaluation

There are two main aims for conducting the final evaluation experiments. The first is to see if the automated carbon footprint reduction tool is able to reduce the carbon footprint of the webpages. To this end, we compare to the *baseline of the initial fitness values for each webpage with no changes made*.

The second aim is to see if the automated carbon footprint reduction tool is more effective than picking change actions at random. The corresponding *baseline is imposing changes with a random generation tool*. That is, we generate a random population and compare to the final population produced by NSGA-II.

Experiment Subjects: We evaluate using ten webpage projects with different degrees of complexity. Table 4.7 shows the project names and GitHub links, Table 4.8 shows the baseline fitness values, and Table 4.9 presents metadata on each subject.

Experiment Configurations: We perform twenty trials for each experimental subject, using both random generation and multi-objective optimization. We employ the following combination of fitness functions:

- (Data Transferred Quantity) + (Number of Changes) + (Page Load Time)
- (Data Transferred Quantity) + (Number of Changes) + (Memory Usage)

In the experiments, the number of generations and population size identified by

Project Name	Project Link
Poke-Dex	https://github.com/AM1CODES/Poke-Dex
Ecommerce-Website-main	https://github.com/MOUSTAFAAMIN25/Ecommerce-Website-main
htmlBurger-website	https://github.com/Marius-MPA/htmlBurger-website
complex-storm-html	https://github.com/kasumaputu06/complex-storm-html
module1-capstone-project	https://github.com/MahdiAghaali/module1-capstone-project
awesome-portfolio-websites	https://github.com/smaranjitghose/awesome-portfolio-websites
OpenWISP-Website	https://github.com/openwisp/OpenWISP-Website
project-website-template	https://github.com/yenchiah/project-website-template
ProjectSakura	https://github.com/ProjectSakura/ProjectSakura.github.io
Books-bootstrap-website	https://github.com/akashyap2013/Books-bootstrap-website

Table 4.7: Project name and GitHub link for the final experimental subjects.

Project Name	Data Transferred (KB)	Page Load Time (ms)	Memory Usage (KB)
Poke-Dex	5158908	3372	275528
Ecommerce-Website-main	3021120	1870	189700
htmlBurger-website	1907546	1937	163008
complex-storm-html	645713	2476	406188
module1-capstone-project	947203	1540	228976
awesome-portfolio-websites	843424	5802	238592
OpenWISP-Website	1238487	6077	172708
project-website-template	5141268	1743	273548
ProjectSakura	1422532	7052	223280
Books-bootstrap-website	1734210	2119	173412

Table 4.8: Baseline fitness values of the webpages used for the final experiment.

Project Name	Num. Images	CSS Code Lines	HTML Code Lines	JS Code Lines
Poke-Dex	41	324	6713	111
Ecommerce-Website-main	33	275	415	319
htmlBurger-website	26	750	730	91
complex-storm-html	53	125	519	318
module1-capstone-project	10	206	192	83
awesome-portfolio-websites	2	1696	217	20
OpenWISP-Website	3	396	243	402
project-website-template	64	1841	1086	7
ProjectSakura	19	706	473	36
Books-bootstrap-website	6	422	355	51

Table 4.9: Metadata on the webpages used for the final experiment.

the parameter tuning experiments were employed—a population of size 20 and a 20-generation search budget.

Data Collection and Analysis: We record the final value for each selected fitness function. Based on the collected data, we compared the performance of multi-objective optimization to the two baselines defined above first using the median fitness values for each test subject.

In addition, we compared the performance of the tool and the random baseline using statistical analysis for each fitness function. Data collected for each subject are drawn from an unknown distribution, and normality cannot be assumed. The Mann-Whitney Wilcoxon rank-sum test [90] is used, with $\alpha = 0.05$, to determine whether

the multi-objective tool and the random baseline offer fitness values drawn from different distributions. The following null hypothesis and alternative hypothesis are proposed for the analysis of the data collected.

Null Hypothesis:

- H0: Observations of results from the automated carbon footprint reduction tool are drawn from the same distribution as the random generation tool.

Alternative Hypothesis:

- H: Observations of results from the automated carbon footprint reduction tool are drawn from the different distributions as the random generation tool.

The null hypothesis is rejected if the p-value is less than 0.05, indicating a significant difference between the distributions.

In the event that the distributions are different, Vargha-Delaney effect size tests [81] are used in order to measure the magnitude of the difference between the distributions. We interpret effect sizes greater than 0.5 as indicating that the first technique performed better than the second technique. Effect sizes lower than 0.5 show that the second technique outperformed the first. An effect size can then be classified as large, medium, or small based on its magnitude. We follow the general interpretation [81], where an effect size of $0.56 \leq A < 0.64$ is classified as small, while an effect size of $0.64 \leq A < 0.71$ is considered medium, and an effect size of $A \geq 0.71$ is deemed large.

5

Results

This chapter describes the results of the two sub-projects. The interview and survey study results from answer RQ1. The results of designing the prototype tool, based on the Design Science methodology, are divided into three iterations. Each iteration offers corresponding results that collectively answer RQ2-3.

5.1 Interview and Survey Study (RQ1)

5.1.1 Existing Knowledge (RQ1.1)

This section reports participants' prior knowledge of software carbon footprint and energy consumption from the thematic analysis of the interview. As shown in Figures 5.1–5.2, most interviewees, no matter how many years of experience in development, are either unfamiliar or have only heard a little about software energy consumption and carbon footprint and they cannot clearly tell the difference between the carbon footprint and energy consumption. Some of the interviewees have some knowledge about the energy consumption and carbon footprint of the IT industry. However, others know nothing related to this topic. They know a little about some actions that may impact computer energy consumption but do not understand specific causal relationships.

> "That's not something that I think about in our sort of daily job. The only thing I can think about is, sort of, all the things that we store on service, in the cloud, of course, those computers, need electricity. There's been a lot of talk about, you know, mining cryptocurrency. It's not a sustainable way of handling money in that sense." - **P4**

> "To be honest, I'm not so familiar with that area, but I just know GPUs use a bunch of electricity, so I can imagine that has like a big impact on the environment." - **P8**

> "Carbon footprint is not always directly, I mean to the energy, its energy consumption. I mean, it depends on where the energy comes from. If the energy is, I mean, carbon neutral, then in that case, the common footprint is still small,

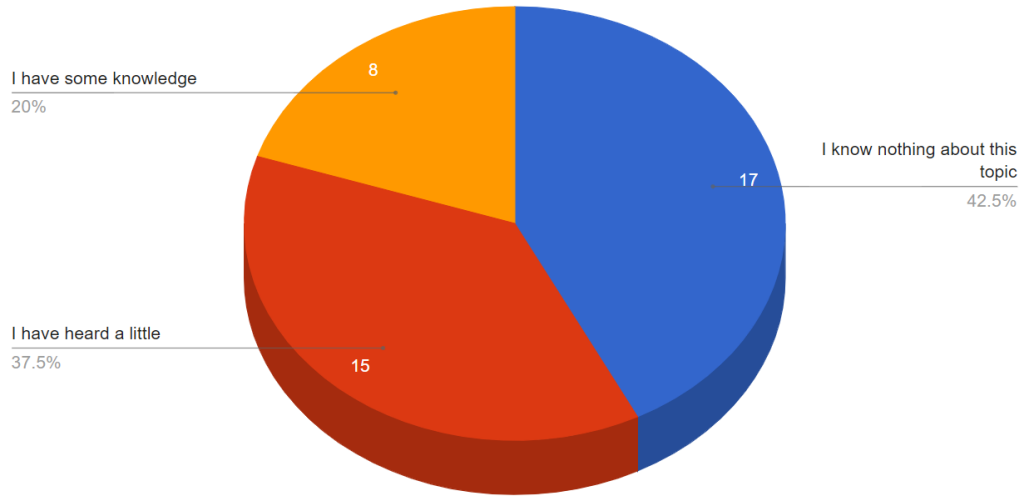

Figure 5.1: Survey participants' knowledge of software carbon footprint.

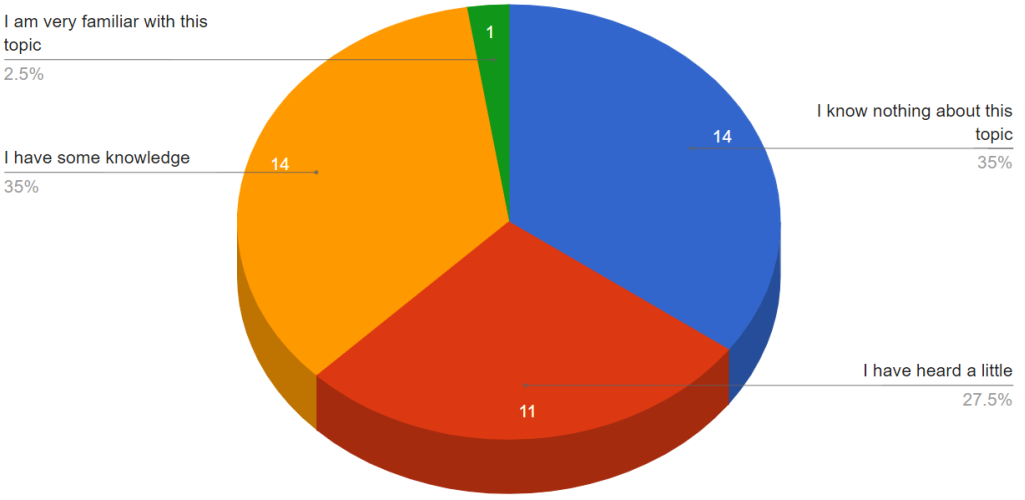

Figure 5.2: Survey participants' knowledge of software energy consumption.

> even if you consume lots of energy. Of course. I mean, I would say that at least most of our customers are not in Sweden, but most of our development is in Sweden. And I would say the majority of the energy in Sweden is not carbon. I mean based, I mean there are some, but most of it is, I mean, water, wind, nuclear, power and so on." - **P1**

In response to the questionnaire question, "How much do you know about the carbon footprint of software?" (illustrated in Figure 5.1), the level of participants' knowledge is disparate. A total of 42.5% of participants have no knowledge of the carbon footprint of software, 37.5% have heard of it or have some knowledge, and none are very familiar with it. The results indicate that most respondents have little or no prior knowledge of the carbon footprint. Compared to the answers to the questions regarding software carbon footprint, participants are more familiar

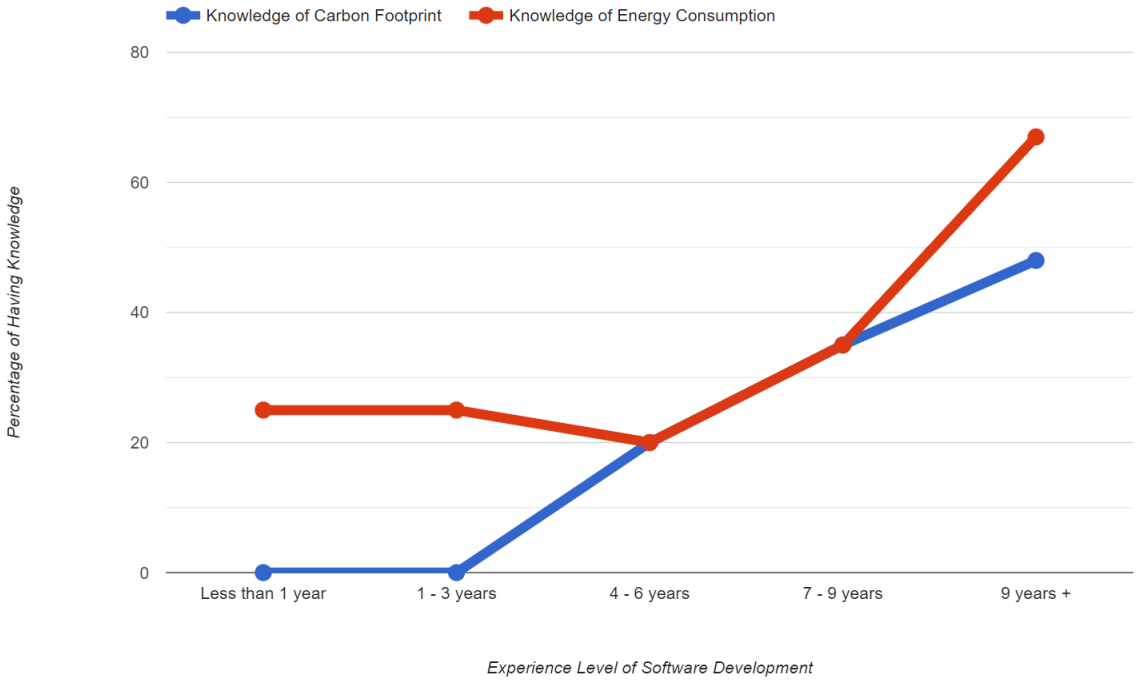

Figure 5.3: Relationship between developers' development experience and knowledge of carbon footprint and energy consumption.

with the term energy consumption. Figure 5.2 illustrates that 35% of participants know nothing about energy consumption—less than the 42.5% who know nothing about carbon footprint. Furthermore, one individual has extensive knowledge of the energy consumption of the software.

As shown in Figure 5.3, the participants' level of software development experience is potentially correlated with their knowledge of energy consumption and carbon footprint. When the level of experience in software development is higher, the percentage of "having some knowledge" in either carbon footprint or energy consumption is greater. Therefore, as software development experience increases, knowledge of the carbon footprint and energy consumption of the software often increases as well.

In addition, we can compare the knowledge of carbon footprint and energy consumption between those working in Japan and Sweden from the survey responses. There are 13 participants from Japan and 16 participants from Sweden. As advanced countries, both place a high priority on environmental protection. Figures 5.4–5.5 indicate that Japan's percentage of respondents who know nothing about carbon footprint is almost the same as Sweden's. However, more participants from Sweden have some or great familiarity with the topic. As shown in Figures 5.6–5.7, in terms of participants' knowledge of energy consumption, participants from Sweden are also more likely to have some knowledge than Japanese participants, and one respondent from Sweden is very familiar with software energy consumption. Overall, Swedish survey participants seem to have a greater understanding of energy consumption

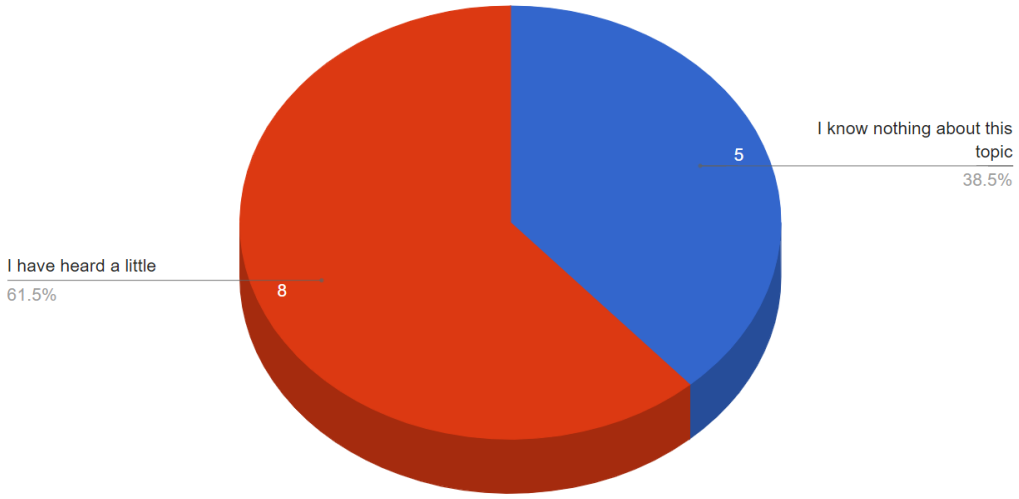

Figure 5.4: Japanese survey participants' knowledge of software carbon footprint.

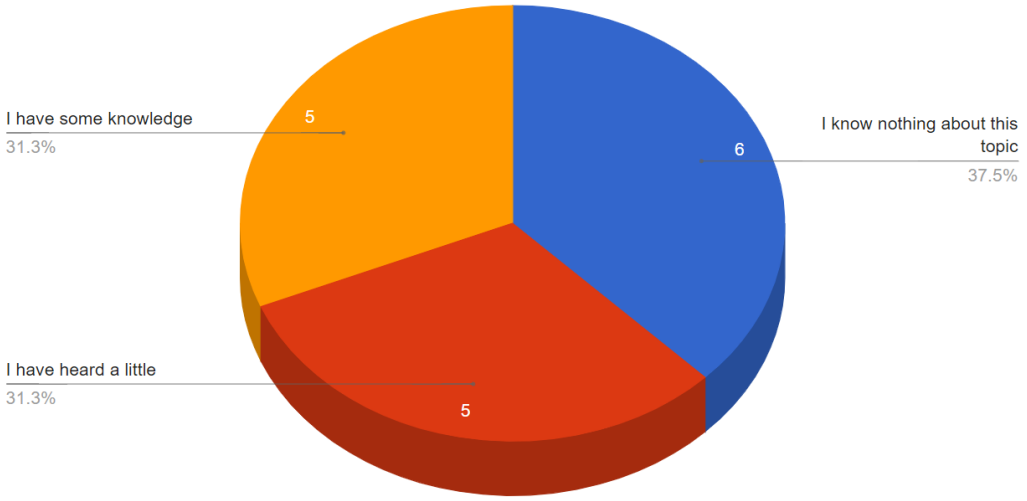

Figure 5.5: Swedish survey participants' knowledge of software carbon footprint.

and carbon footprint than Japanese participants. As for the qualitative interview answers from both countries, there is no distinctive difference in the knowledge of energy consumption and carbon footprint.

Existing Knowledge (RQ1.1): Many developers lack significant knowledge of the carbon footprint or energy consumption of software. Of the two concepts, developers are more familiar with energy consumption. However, developers do appear to gain more knowledge of these concepts as they gain experience. More Swedish survey participants had familiarity with both concepts than Japanese participants.

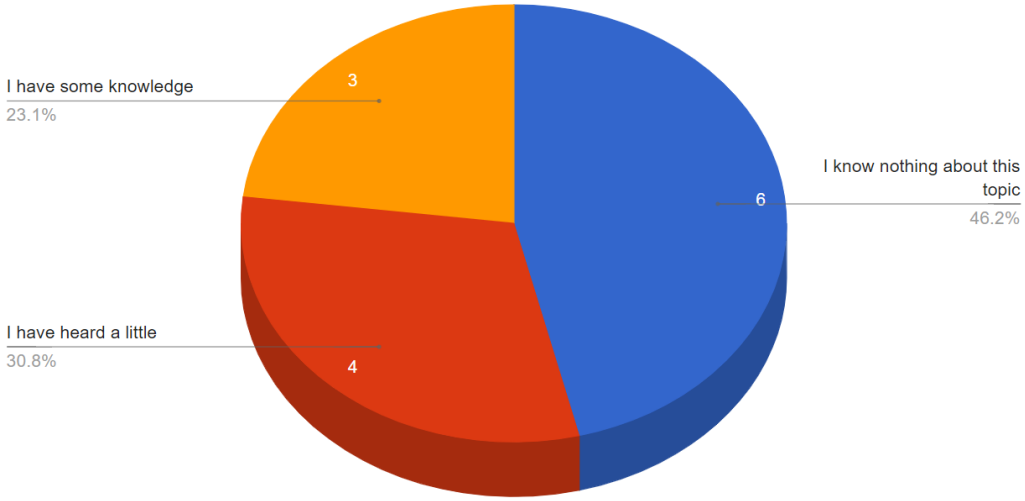

Figure 5.6: Japanese participants' knowledge of software energy consumption.

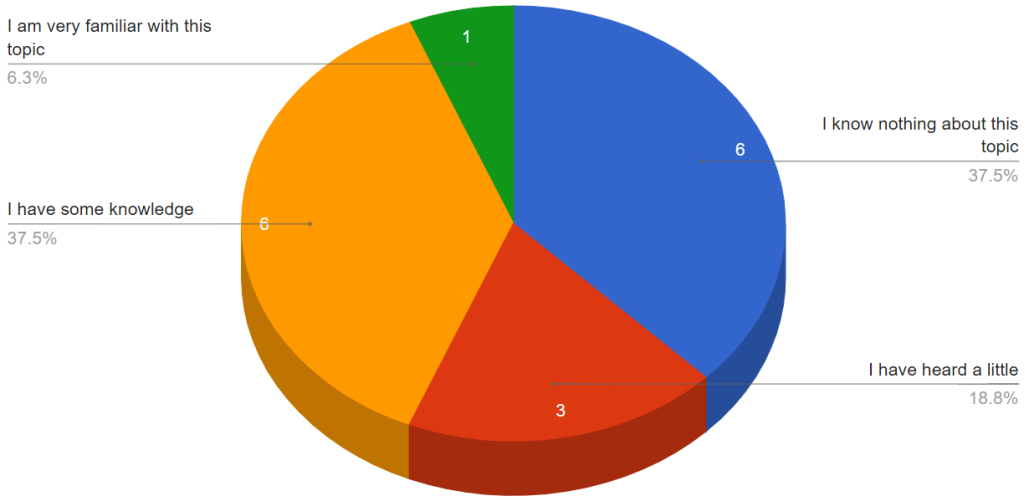

Figure 5.7: Swedish participants' knowledge of software energy consumption.

5.1.2 Opinions and Development Practices Related to Carbon Footprint and Energy Consumption (RQ1.2)

This section examines participants' experience in assessing and controlling energy consumption and carbon footprint during the development process. We consider developers' opinions on the contribution of software to climate change and responsibility for controlling these factors, development stages, actions taken, and the measurements used in performing this assessment.

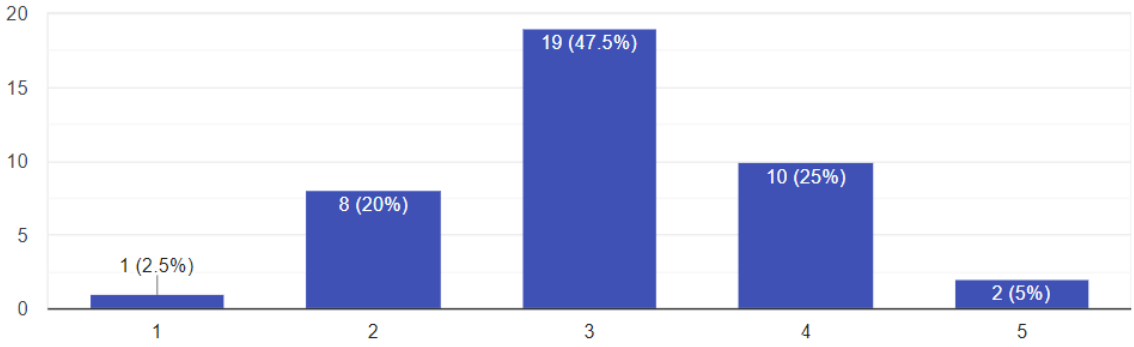

Figure 5.8: Degree of agreement on whether software carbon footprint contributes to climate change (1-5 corresponding to strongly disagree–strongly agree).

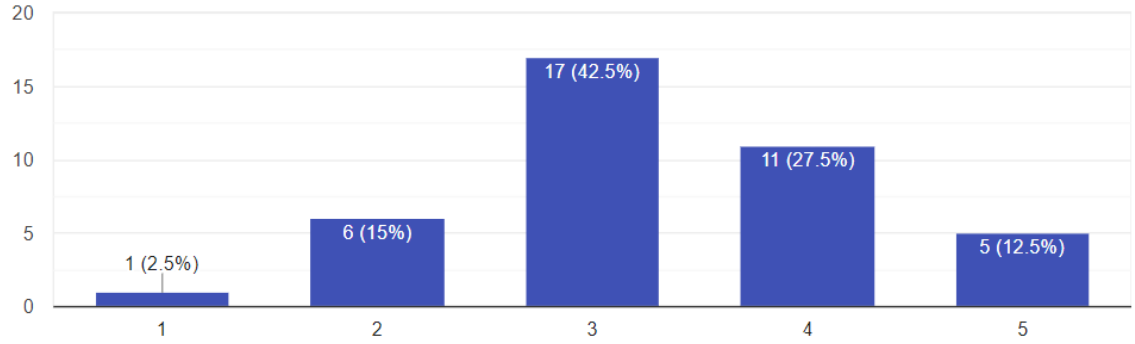

Figure 5.9: Degree of agreement on whether software carbon footprint should be considered and controlled (1-5 corresponding to strongly disagree–strongly agree).

5.1.2.1 Opinions on Carbon Footprint and Energy Consumption

Figure 5.8 shows a largely balanced view on whether participants agree that software carbon footprint contributes to climate change, with a plurality (47.50%) showing a neutral view. A somewhat larger percentage of participants agree that software carbon footprint contributes to climate change (30.00%) than disagree (22.50%).

Regardless of their view on whether software carbon footprint contributes to climate change, Figure 5.9 shows that more participants (40.00%) agree that the carbon footprint of software should be considered and controlled than disagree (17.50%). Again, however, the plurality feels neutral on this topic (42.50%). On both questions, the distributions of opinions are largely the same for participants from both Sweden and Japan.

Figure 5.10 shows participants' views on who holds responsibility for considering or controlling the energy consumption or carbon footprint of software. Participants could select more than one option. The majority of respondents felt that both development organizations (65.00%) and regulatory agencies (60.00%) should bear responsibility. However, many (37.50%) still felt that individual developers bear responsibility. Only 10.00% of participants believe that no one should be required

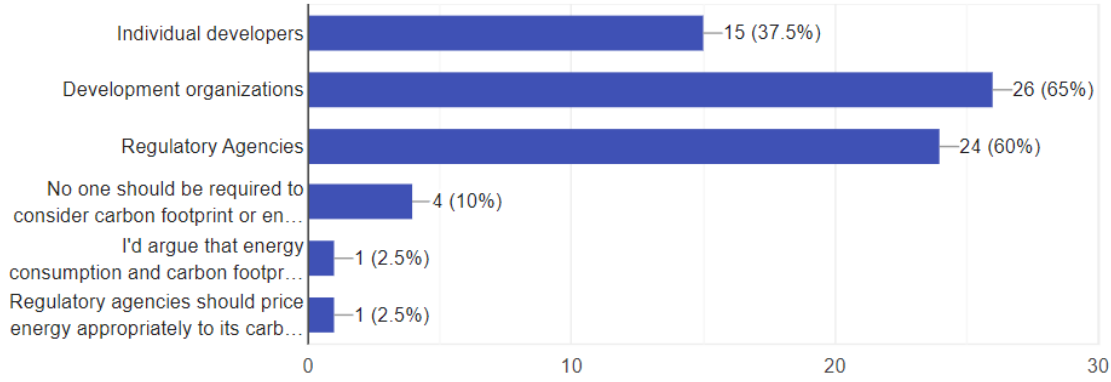

Figure 5.10: Responsibility party for considering or controlling the energy consumption or carbon footprint of software.

to consider their carbon footprint.

Two participants left complementary comments. One clarified their view on the role of energy consumption:

> "Energy consumption is important for operation (e.g. limited energy is available), maintenance (more energy, more heat, things break down sooner), and regulations (e.g. Devices that need to have a particular energy rating, such as screens, light bulbs, etc)." - **I15**

Another participant noted that:

> "Regulatory agencies should price energy appropriately to its carbon footprint." - **I21**

The interviewees also discussed both individual responsibility and organizational responsibility. It is the engineer's responsibility to comply with the company's development rules, business goals, and criteria. Therefore, they do not have the option of reducing their energy consumption.

> "You know what's important for the business is the company is they have a feature that they want to develop, and they have criteria, and that's important for them no matter so and they want to do it in an effective way, which means they need to." - **P7**

> "... I think I need to understand the domain a little bit more in order to say if it's the responsibility of the developer or if it's more of an organization responsibility." - **P4**

Table 5.1 contrasts opinions on responsible parties between Japanese and Swedish participants. The proportion of Japanese respondents who believe that individual

	Individual Developers	Development Organizations	Regulatory Agencies	No One
Japan	38.00%	85.00%	70.00%	7.00%
Sweden	20.00%	40.00%	67.00%	20.00%

Table 5.1: Distribution of responses from Swedish and Japanese participants on who bares responsibility for considering and controlling energy consumption or carbon footprint.

developers are responsible for considering or controlling energy consumption or carbon footprint is almost twice that of Swedish respondents—38.00% versus 20.00%. Developers in Japan may feel greater individual responsibility for considering the carbon footprint or energy consumption of their software than their Swedish counterparts. Additionally, over twice as many Japanese respondents (85.00% versus 40.00%) feel that development organizations bear responsibility. Japanese respondents place the most responsibility on companies—over both individuals and regulatory agencies—to take responsibility for controlling their carbon footprint.

In contrast, Swedish respondents place the largest burden on regulatory agencies to take responsibility—over both companies and individual developers. In addition, a greater proportion of Swedish respondents (20.00%) feel that no one bears responsibility than Japanese respondents (7.00%).

Developer Opinions (RQ1.2): A plurality of respondents are neutral on whether software carbon footprint contributes to climate change and whether carbon footprint should be considered and controlled. However, a greater proportion of participants agree with both than disagree. The majority of participants feel that development organizations and regulatory agencies both bear responsibility for considering and controlling the energy consumption or carbon footprint of software.

5.1.2.2 When Carbon Footprint and Energy Are Considered

Carbon footprint and energy consumption can be taken into account during multiple development stages, including requirements engineering, design, implementation, testing, deployment, and maintenance.

Interview participants considered energy consumption and carbon footprint during design, testing, and maintenance. During the design phase, developers have to estimate the number of end users and identify the resources (e.g., the number and capability of servers) they will need:

> "Before we even start making an application, we design the application. During the design phase itself, we're going to make some estimations. We're going to decide what technologies are we going to use, how many servers we are going to have, what is the lord that we are expecting." - **P2**

Additionally, problematic energy consumption can be observed and mitigated after the system is deployed:

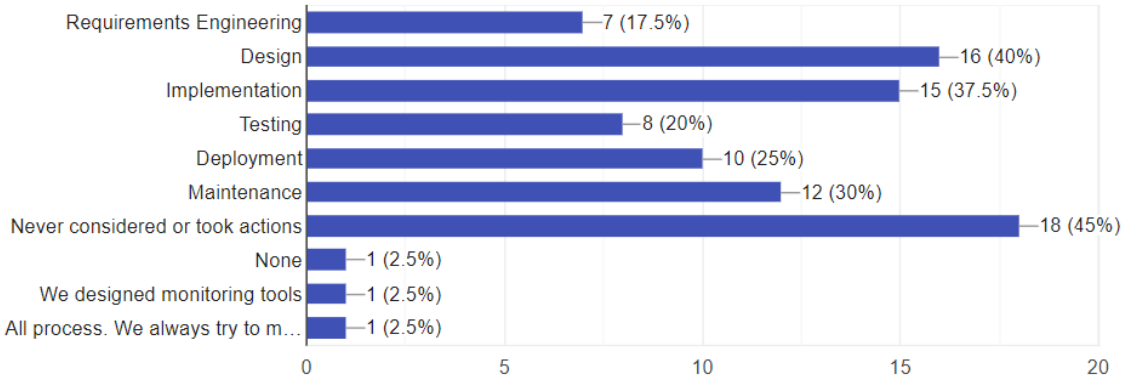

Figure 5.11: Development stages where survey participants considered energy consumption.

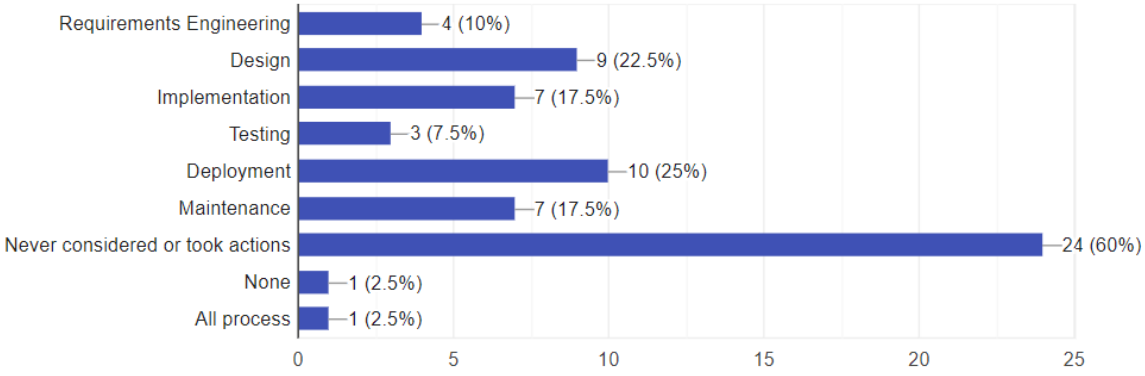

Figure 5.12: Development stages where survey participants considered carbon footprint.

> "... When we actually run something in production and see that ... we are actually overloading the systems, we need to do something about that." - **P3**

Figures 5.11 and 5.12 illustrate the different stages where survey participants considered energy consumption and carbon footprint. Participants could make multiple selections. In both cases, the largest percentage of participants—45.00% for energy consumption and 60.00% for carbon footprint—never considered or took action. However, for both factors, a subset of participants considered or took action at all stages.

For energy consumption, the most consideration is given during the design (40.00%) and implementation (37.50%) phases, followed by maintenance (30.00%), deployment (25.00%), testing (20.00%), and requirements (17.50%) phases. During the design phase, the requirements are transformed into an implementable form. Software engineers must consider architecture design, resource consumption, *etc.* In this phase, decisions are made on how the architecture will function, which often must incorporate decisions about energy consumption. These decisions are realized during the implementation phase [68].

Many considered carbon footprint as well during the design (22.50%) and implementation (17.50%) phases. However, deployment (25.00%) is the most important phase for carbon footprint consideration. Maintenance (17.50%) is also important. Gaining an accurate estimation of carbon footprint may be difficult before the system is operating in its final production environment, where clear usage statistics can be gathered as well as knowledge of where users and data centers are in the world. Energy consumption decisions can be made without such information, and energy consumption can be considered earlier in development in conjunction with—and as a way to control—carbon footprint. Once the system is deployed, statistics on carbon footprint can be gathered and changes can be made, if needed.

Development Stages (RQ1.2): When energy consumption is considered or controlled, actions take place most often during the design and implementation phases. Carbon footprint is considered most often during deployment and design, but implementation and maintenance are both important as well.

5.1.2.3 Actions Taken to Consider or Control Carbon Footprint and Energy Consumption

Both interviewees and survey participants were asked what concrete actions they had taken to address energy consumption or carbon footprint. Most interviewees have not taken action with regard to either. However, multiple interviewees took various actions that reduce resource usage. These include compressing elements or offering shortcuts to users—reducing resource usage that is unnecessary for some users:

> "In order to make the web pages faster, or save up time or save resources, we took some methods to reduce our resource, e.g., we compressed images, or compressed music, or some easier way to click to the bottom or go to the next pages." - **P6**

Others focused on code efficiency. One interviewee stressed that improving code performance will lead indirectly to improvements in carbon footprint:

> "As a software engineer, if I'm a good software engineer ... I indirectly reduce the cost. I indirectly improve the code and indirectly reduce the carbon footprint. There's no direct link, but indirectly, I'm all doing that." - **P5**

Survey participants were asked which actions they had taken to reduce energy consumption, illustrated in Figure 5.13. They were presented with a set of known practices—of which they could select multiple—and could add additional answers. The most common practice was to improve performance (50.00%), followed by reducing CPU consumption (37.50%), reducing memory consumption (27.50%), and reducing data upload frequency (27.50%). Again, much of the focus was on improving energy consumption by either increasing the response time or throughput of the software or by decreasing resource consumption.

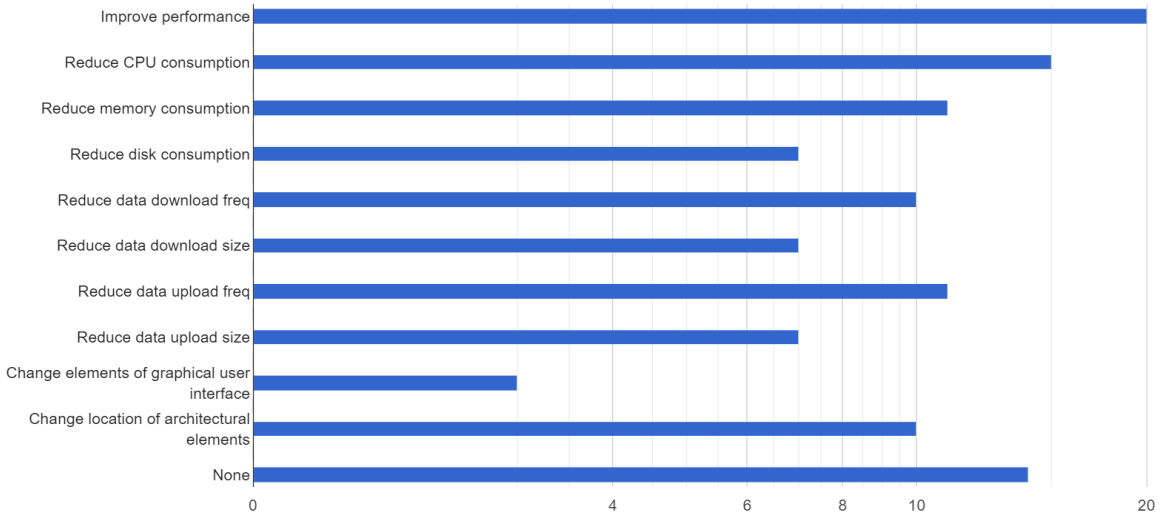

Figure 5.13: Actions have been taken to reduce the energy consumption of software.

Survey participants were presented with an additional optional open-ended question asking what actions they had taken to reduce the carbon footprint of their software. Many of the responses specifically discussed improved energy consumption as a way to also address carbon footprint. Again, the focus was primarily on reducing resource consumption:

> "I have mostly cared about energy consumption in battery-powered devices where the goal is to have the device in a lower power state ("deep sleep") as much as possible." - **I15**

> "Elastic scaling so services are scaled up when there is a need only." - **I20**

> "Reduce memory consumption: tried to make the software efficient starting from design. Tried to use objective language and reuse data using instances." - **I31**

Another survey participant also noted that the development process itself contributes to carbon footprint and that developers can take actions during development to limit their footprint:

> "I have taken steps to reduce the energy consumption needed for developing the software (by shutting down unused machines and reducing background processes and other things on development machines to reduce power consumption)." - **I21**

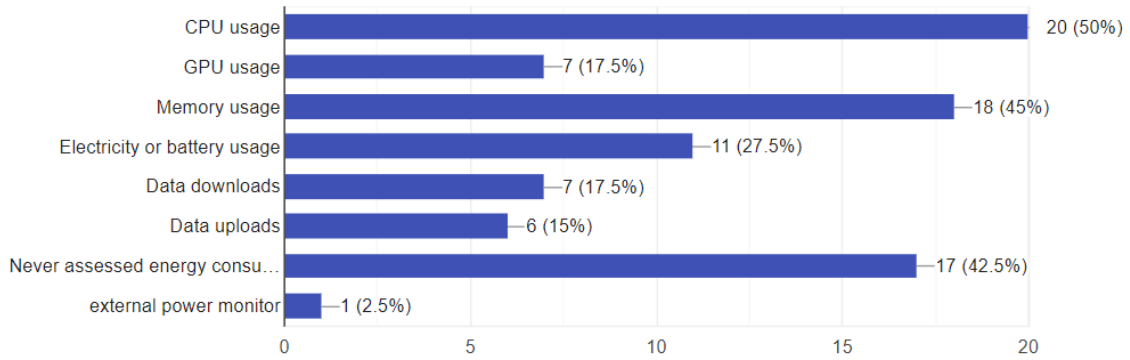

Figure 5.14: Measurements of energy consumption employed by survey respondents.

Actions Taken (RQ1.2): Developers typically focus on improving performance or reducing resource consumption (reducing CPU, memory usage, page size, or upload frequency) to improve energy consumption. When carbon footprint is considered, it is usually addressed through the same actions and considered specifically in relation to energy consumption instead of separately.

5.1.2.4 Measurement and Assessment of Carbon Footprint and Energy Consumption

The interviewees indicated that they have not directly evaluated energy consumption or carbon footprint. However, some interviewees have indirectly assessed energy consumption through costs and resource usage:

> "We don't evaluate energy consumption, but we do evaluate cost, which is directly related to energy consumption. [To reduce] cost of service, we first and foremost design good systems ... which are more efficient, consume less cost, and improve user experience." - **P8**

> "... It's more about, okay, how can we make sure that we are not using so many resources on the service, more than talking about the energy consumption and assessing it ... It's more like, well, we discovered that we are loading the service too much ... We're not doing any formal assessment of it." - **P1**

We also asked survey respondents what measures, either direct or indirect, they have used to assess energy consumption (Figure 5.14) or carbon footprint (Figure 5.15). Again, they could select one or more options from a set of pre-filled responses or provide their own. Figures 5.14–5.15 indicate that many have never considered either—42.50% for energy consumption and 70.00% for the carbon footprint. Among the respondents who evaluated either factor, CPU usage and memory usage were the most common methods for both energy consumption (50.00% and 45.00%) and carbon footprint (22.50% and 20.00%).

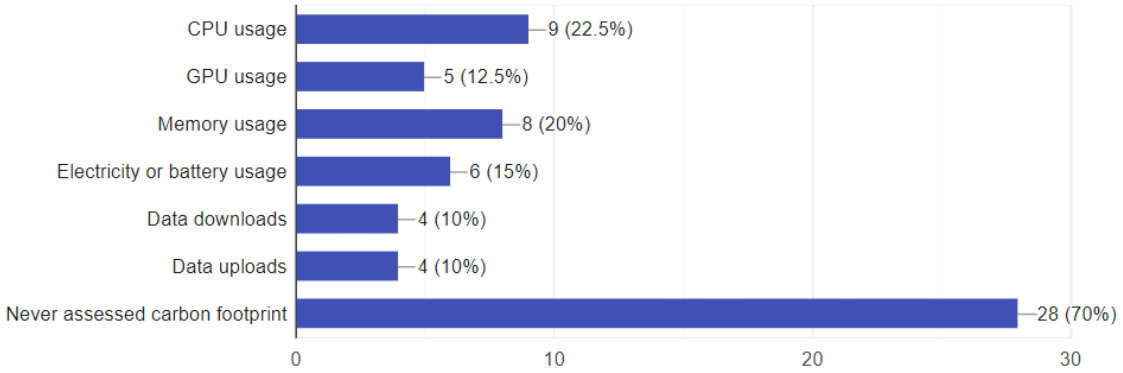

Figure 5.15: Measurements of carbon footprint employed by survey respondents.

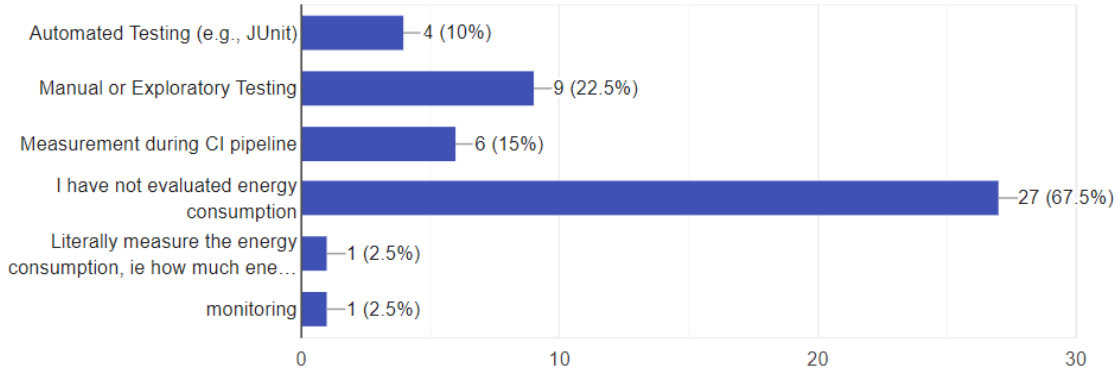

Figure 5.16: Techniques used to evaluate software energy consumption.

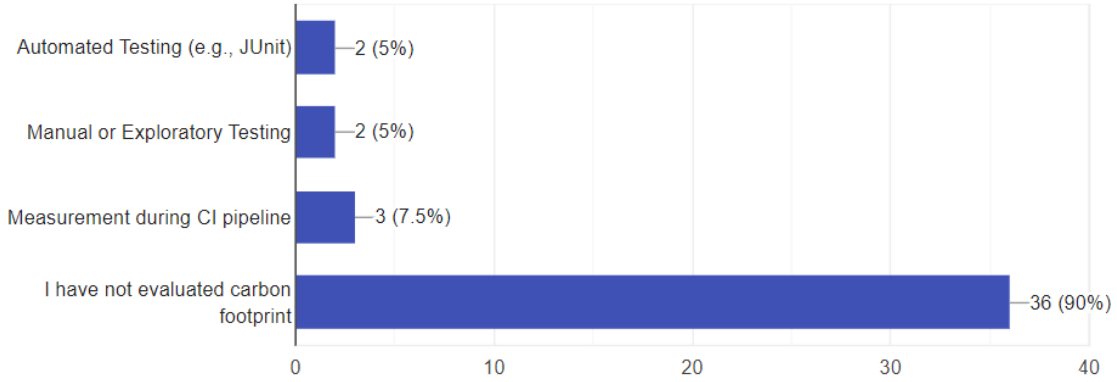

Figure 5.17: Techniques used to evaluate software carbon footprint.

Survey participants were also asked what techniques they have used to evaluate energy consumption or carbon footprint. As shown in Figures 5.16–5.17, most developers have never directly performed an evaluation of either factor—67.50% for energy consumption and 90.00% for the carbon footprint.

Of those who have performed an evaluation of energy consumption, the most common method was through manual or exploratory testing (22.50%), followed by a

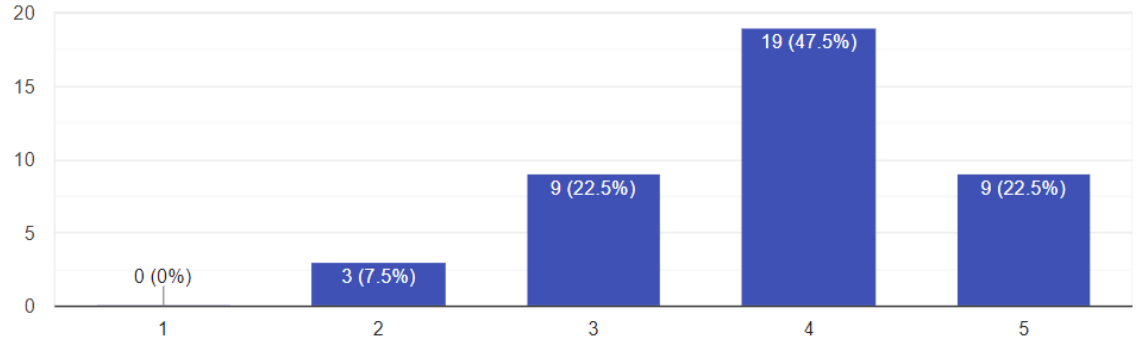

Figure 5.18: Willingness to use such a tool among survey participants (1 = "Strongly Unwilling", 5 = "Strongly Willing").

measurement taken as part of Continuous Integration (15.00%). For carbon footprint, the most common technique was to take a measurement as part of Continuous Integration (7.50%).

Measurement and Evaluation (RQ1.2): Energy consumption and carbon footprint are most commonly measured through CPU or memory usage. Energy consumption is most often evaluated using manual or exploratory testing, while carbon footprint is most commonly evaluated through measurements taken during Continuous Integration.

5.1.3 Requirements for Automated Carbon Footprint Reduction Tools (RQ1.3)

A major purpose of this study is to establish the requirements that must be met for developers to actively use and trust an automated carbon footprint reduction tool. During the interviews, all participants expressed a positive attitude toward such a tool and regarded the concept as exciting and potentially helpful. Additionally, all interviewees expressed interest in trying such a tool:

> "Of course. We want to reduce the amount of power our software requires. ... it's part of the whole green message we need to send to the world ... I don't really like bitcoin mining and that kind of thing, so I like what the theory is doing, changing to a much more energy-reduced algorithm for the mining." - **P1**

Survey participants were also asked about their willingness to try such a tool, with the results illustrated in Figure 5.18. 70.00% of participants were either willing or strongly willing to try such a tool, and only 7.50% of participants were unwilling to try a tool.

However, Figure 5.19 shows that participants were somewhat skeptical of the ability of such a tool to successfully reduce carbon footprint. 35.00% of respondents were

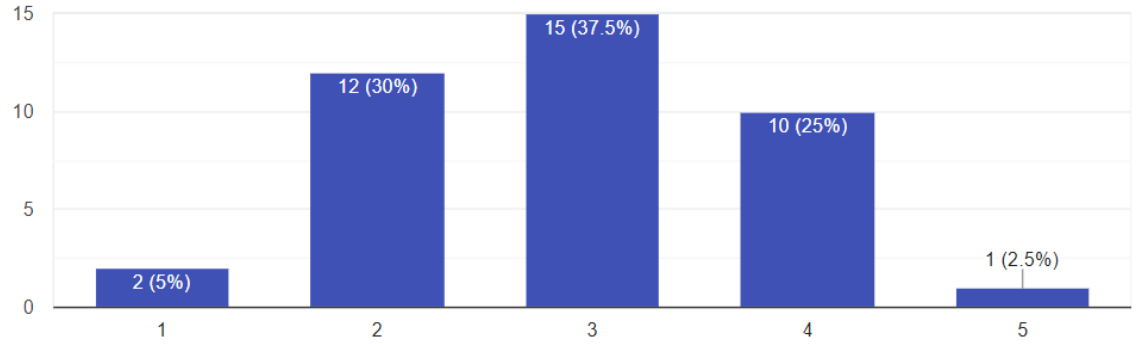

Figure 5.19: Skepticism of survey participants towards a tool's ability to reduce carbon footprint (1 = "Highly Skeptical", 5 = "Highly Unskeptical").

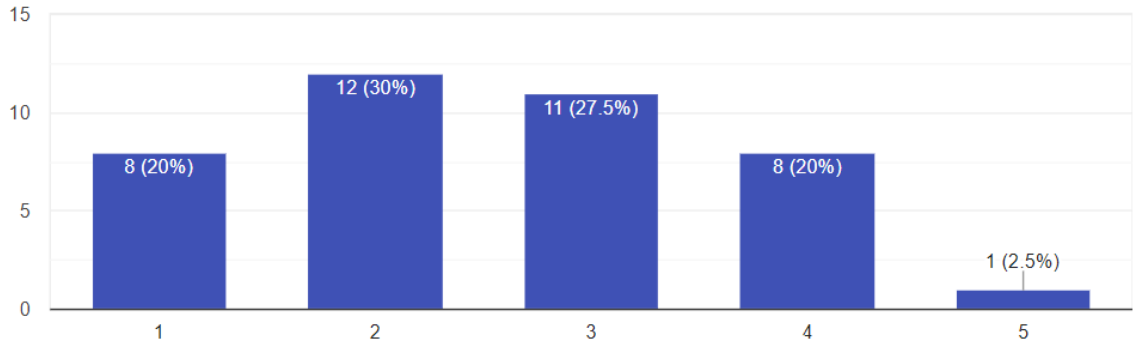

Figure 5.20: How likely survey participants are to trust an automated tool to modify their code to reduce carbon footprint (1 = "Very Unlikely", 5 = "Very Likely").

either skeptical or highly skeptical of the potential capabilities of such a tool, and a further 37.50% had neutral expectations.

In addition, participants were asked how likely they were to trust an automated tool to modify their source code. Again, the level of up-front trust in an automated tool's results was low, with 50.00% either being unlikely to very unlikely to trust the tool, and a further 27.50% being neutral. Only one respondent indicated that they would be strongly likely to trust a tool. Some skepticism seems reasonable. Even if a tool could reduce its carbon footprint without supervision, the changes it made could result in incorrect program results or reduced quality in another non-functional aspect. Care must be taken in the design of such a tool to reduce the likelihood of such an occurrence.

> "When you have these kinds of tools that would analyze and diagnose your code, I think it's usually very tricky to get a hundred percent correct." - **P4**

Developers in Japan and Sweden were almost equally interested in trying a tool, with average scores of 3.69 and 3.73, respectively. However, there were differences in their expectations. On average, Japanese respondents were rated at 3.23 on

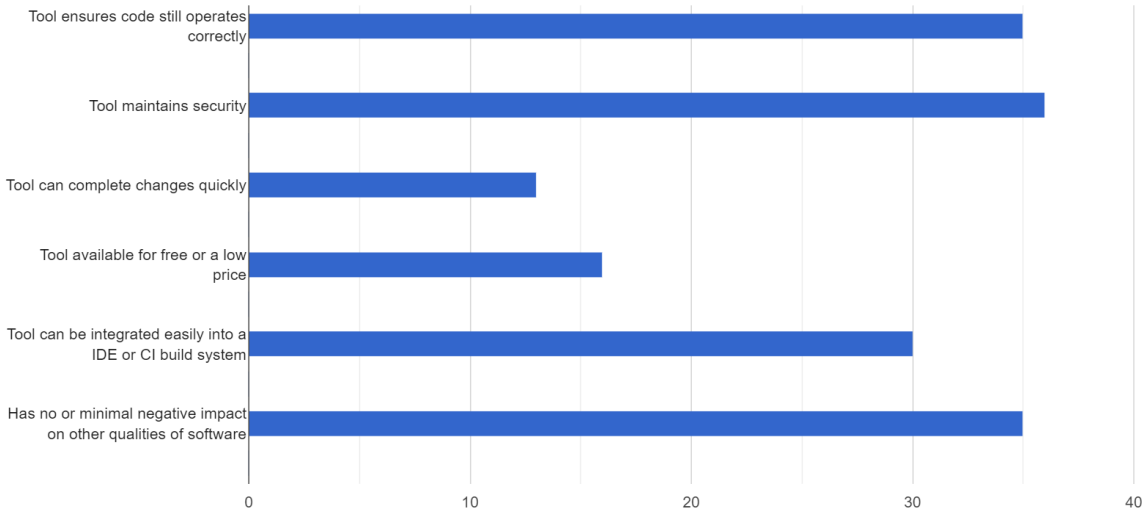

Figure 5.21: Survey respondents' requirements that need to be met to trust an automated carbon footprint reduction tool.

skepticism—approximately neutral—while Swedish respondents were rated at 2.40 on average—approximately skeptical. Similarly, Japanese developers rated roughly neutral on average—2.76—in their likeliness to trust a tool, while Swedish developers rated 2.06, (unlikely). Swedish developers seem willing to try a tool, but they have low expectations and trust in its capabilities.

Requirements and Trust (RQ1.3): Participants are willing to try an automated tool to reduce their carbon footprint. However, many participants had neutral-to-low expectations of its capabilities or trust in the tool to modify their code.

Both interview and survey respondents were asked about the requirements that would have to be met for a tool to be trusted. The results for survey participants are shown in Figure 5.21. Among survey participants, the top three requirements are that security is maintained (90%), that the code still operates correctly (87.50%), and that there is no—or minimal—negative impact on other qualities of the software (87.50%). The importance of such requirements is unsurprising. Security is one of the most significant qualities of software, with major legal and financial implications. Tools should also not introduce faults, and reducing carbon footprint may not be universally prioritized if it comes at the expense of other qualities of the software, such as performance.

Other participants asked that the tool be easily integrated into a Continuous Integration (CI)/Continuous Delivery (CD) pipeline, be available for free or at a low price, and be able to complete changes quickly.

Further, the tool should produce understandable and maintainable code and verifiable, interpretable, and accurate results. It should produce patches that actually reduce carbon footprint, and users should understand how and why changes were

made. For example, the tool could provide a report with an explanation of where and how code has been changed and how the carbon footprint has been impacted. A detailed report with data and rationale for code changes is essential for users to verify and trust the tool's results.

> "To actually start using the tool if I think I can trust it, it depends on how well it will fit into our development processes and how easy it is to integrate it into a normally built pipeline and, and so on. ... if you have to buy it, of course, the price has some implications as well. If it's very expensive, then you have to weigh that against how much are we willing to spend on detecting energy consumption." - **P1**

Requirements and Trust (RQ1.3): To trust a tool, reductions in carbon footprint must not compromise security, correctness, or other important software qualities. In addition, developers would like it to be easy to integrate into a CI pipeline, be reasonably priced, and complete changes quickly.

5.1.4 Use in Development Workflow (RQ1.4)

Interview and survey participants were asked about how they would apply an automated carbon footprint reduction tool and how often they would use the tool.

Interviewees were primarily interested in integrating this tool into existing CI/CD pipelines. CI/CD is a series of steps performed to build and test software when code changes are pushed to a shared repository.

> "I assume it would be something that we would put into the CI/CD pipeline somehow in order to get fast feedback to the developers." - **P4**

As shown in Figure 5.22, 65.00% of survey participants would apply the tool automatically as part of CI during non-peak times (e.g., overnight). Running a tool at non-peak times would allow it to make changes while not interfering with normal development. This way, it would not be applied when the code is being changed rapidly and is more likely to be in a working state. In addition, the tool could take more time if the results are not needed quickly. 40.00% of respondents would also like to be able to apply the tool automatically as part of CI on a normal commit. Approximately a third of respondents (32.50%) believe that the tool could also be applied periodically or by an individual developer before committing code.

When they are asked about the frequency they would apply such tools (Figure 5.23), 20.00% of survey participants indicated that they would use it after every commit, and 17.50% would use it once per month or periodically. A further 15.00% would apply once per day or following a dedicated coding session.

Fit in Workflow (RQ1.4): Developers would apply the proposed tool as part of a CI/CD pipeline either at non-peak times or after each commit.

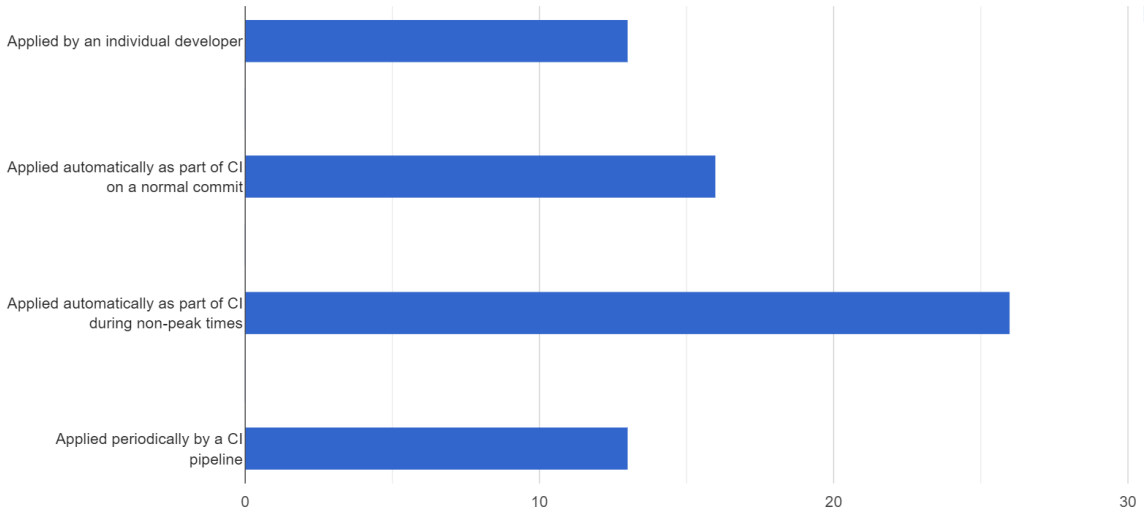

Figure 5.22: Survey participants' views on how such a tool would fit into the development workflow.

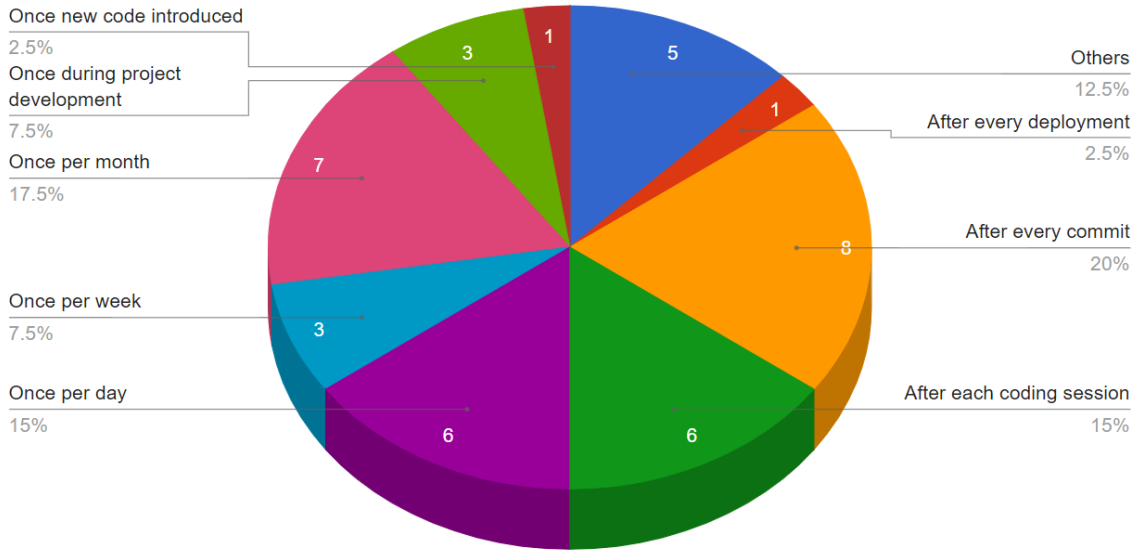

Figure 5.23: Frequency that survey participants would apply the tool.

5.1.5 Voluntary Adoption of Tool (RQ1.5)

Both interview and survey participants were also asked how to encourage voluntary adoption of tools that reduce software carbon footprint. The views of survey participants are shown in Figure 5.24.

As might be expected, the most important factor in encouraging adoption was that the tool show a track record of clear and trustworthy results (92.50%). Testimonials from existing users of positive experiences in using the tool could encourage others to apply it. One interviewee noted:

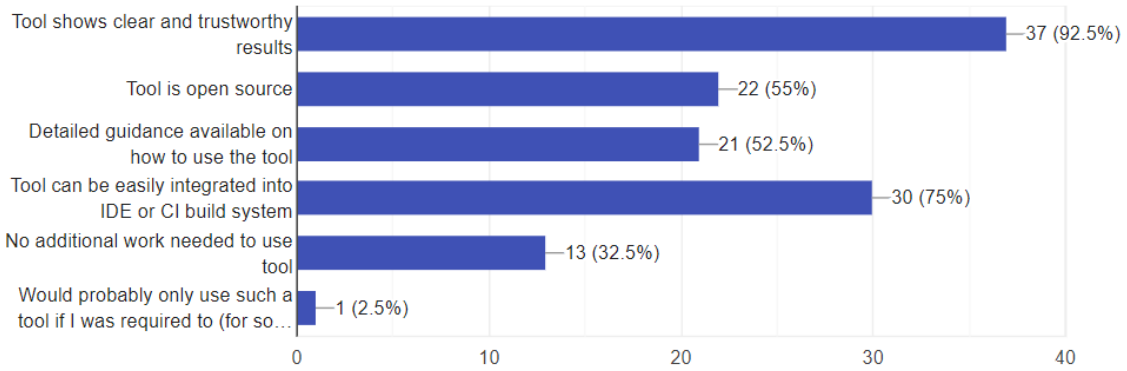

Figure 5.24: Survey participants' views on how to encourage developers to use such a tool.

> "I will wait until the wider community accepts it. First, I will see if this is doing good. I will try to understand as much as possible as to what it is doing before I start using this tool." - **P6**

Apart from that, the ability to easily integrate the tool into an IDE or CI/CD pipeline was essential for 75.00% of the participants. If a tool is easy to use or to integrate into an existing development workflow, then developers are more likely to adopt it. Many respondents (32.50%) also asked that no additional work be needed to use the tool. This tool should be easy to install, or require undue effort to use. Interviewees added:

> "It increases the productivity or does not decrease the productivity of the developers." - **P6**

> "I think the main constraint, in this case, would be that it does not use a lot of resources and development time ... should be not interrupting any development work. It's okay if it takes longer to analyze once the changes are pushed, because it can just run overnight and I don't need to worry about that. But if I use it during development, I would like it to not affect any of my development experience." - **P7**

Other important factors included the tool being open-source and there is detailed guidance on how to apply the tool. If the tool is open-source, those who are interested in it can gain a better understanding of its algorithm and methods. A community could even be created that improves the tool and expands its capabilities and performance. Detailed documentation should accompany this tool, providing clear and understandable instructions on how to install and use the tool.

Voluntary Adoption (RQ1.5): To encourage adoption, the tool should have a track record of transparent and trustworthy results. It should be easy to install and use as part of existing development workflows, with sufficient guidance. It should also be open-source, so that others can understand or expand its capabilities.

Population Size	Num. Generations	Generational Distance	Hypervolume	Best Sol	Sum
10	10	0	0	0.6328	0.6328
10	15	0.1625	0.6435	0.5751	1.3811
10	20	0.2897	0.8366	0.9473	2.0736
15	10	0.3312	0.7870	0	1.1182
15	15	0.5078	0.8271	0.6298	1.9647
15	20	1.0	0.9016	0.5385	2.4401
20	10	0.2641	0.8614	0.8154	1.9409
20	15	0.8435	1.0	1.0	2.8435
20	20	0.8032	0.9004	0.8448	2.5484

Table 5.2: Normalized results of parameter tuning on `Poke-Dex` with the fitness function combination of quantity of data transferred, number of changes, and page load time.

Population Size	Num. Generations	Generational Distance	Hypervolume	Best Sol	Sum
10	10	0.1330	0.4666	0.0	0.5996
10	15	0.0	0.0313	0.6787	0.7100
10	20	0.1130	0.0	0.6439	0.7569
15	10	0.4975	0.5175	0.7632	1.7782
15	15	0.6315	0.5282	0.8151	1.9748
15	20	0.6822	0.5029	0.8138	1.9989
20	10	0.8599	0.6503	0.9584	2.4686
20	15	0.9880	1.0	1.0	2.9980
20	20	1.0	0.9811	0.9665	2.9476

Table 5.3: Normalized results of parameter tuning on `Poke-Dex` with the fitness function combination of quantity of data transferred, number of changes, and memory usage.

5.2 Prototype Tool Design (RQ2–3)

5.2.1 Parameter Tuning

Before performing the final experiments, we performed parameter tuning for the NSGA-II algorithm. As an example, Table 5.2 presents the normalized values of the generational distance, hypervolume, and best solution for the fitness function combination of quantity of data transferred, number of changes, and page load time for the `Poke-Dex` website and Table 5.3 presents the values of the metrics for the fitness function combination of data transfer quantity, number of changes, and memory usage. The last column presents the summation of the three normalized metrics. This is used as the response in the Taguchi method, with the highest value being the best.

The results of parameter tuning for project `Poke-Dex` are shown in Tables 5.4 and 5.5 for the two fitness function combinations. The best combination of parameter settings corresponds to the largest mean S/N value. As shown in Figure 5.25, a population size of 20 and a search budget of 20 generations yields the best performance for fitness function combination of data transfer quantity, number of changes, and page load time for the `Poke-Dex` project. A population size of 20 and a search budget of 15 generations yield the best performance of the fitness function combination of the

Population Size	Num. Generations	Sum1	Sum2	Sum3	S/N
10	10	0.6328	0.7421	0.7654	-6.9649
10	15	1.3811	1.3234	1.2769	5.6293
10	20	2.0736	1.8871	1.6578	12.2952
15	10	1.1182	1.2378	1.1765	3.2158
15	15	1.9647	1.8289	1.8774	12.7083
15	20	2.4401	2.4526	2.3232	17.5350
20	10	1.9409	2.0238	1.9980	13.7279
20	15	2.8435	2.7445	2.7863	20.5234
20	20	2.5484	2.6312	2.6043	19.0649

Table 5.4: Parameter tuning results for `Poke-Dex` with the fitness function combination of quantity of data transferred, number of changes, and page load time.

Population Size	Num. Generations	Sum1	Sum2	Sum3	S/N
10	10	0.5996	0.6871	0.6523	-8.8238
10	15	0.7100	1.2343	1.3144	-0.7046
10	20	0.7269	1.2218	1.3032	-0.4920
15	10	1.7782	1.8823	1.7765	11.8701
15	15	1.9748	1.9897	1.8774	13.3093
15	20	1.9989	1.9724	1.8698	13.3017
20	10	2.4686	2.3412	2.2130	16.9518
20	15	2.9980	2.9763	2.9877	21.8866
20	20	2.9476	2.9487	2.8961	21.5043

Table 5.5: Parameter tuning results for `Poke-Dex` with the fitness function combination of quantity of data transferred, number of changes, and memory usage.

	`Poke-Dex`		`Ecommerce-Website-main`		`htmlBurger-website`		`complex-storm-html`		`module1-capstone-project`	
	Pop	Gens	Pop	Gens	Pop	Gens	Pop	Gens	Pop	Gens
Data-Changes-Time	20	20	20	20	20	20	20	20	20	20
Data-Changes-Memory	20	15	20	20	20	15	20	20	20	20

Table 5.6: Identified parameter settings for each project.

data transfer quantity, the number of changes, and the memory usage.

The results of the parameter tuning experiment are summarized in Table 5.6 for all five considered projects. In both `Poke-Dex` and `Ecommerce-Website-main`, the best parameters for the second fitness function combination are a population size of 20 and a search budget of 15 generations. For the other projects, the best parameters are a population size of 20 and a 20-generation search budget. Thus, in the following experiments, we use a population size of 20 and a 20-generation search budget.

5.2.2 Final Experiment Results

The goal of our final experiments is to compare the effectiveness of the automated carbon footprint reduction tool with two baselines:

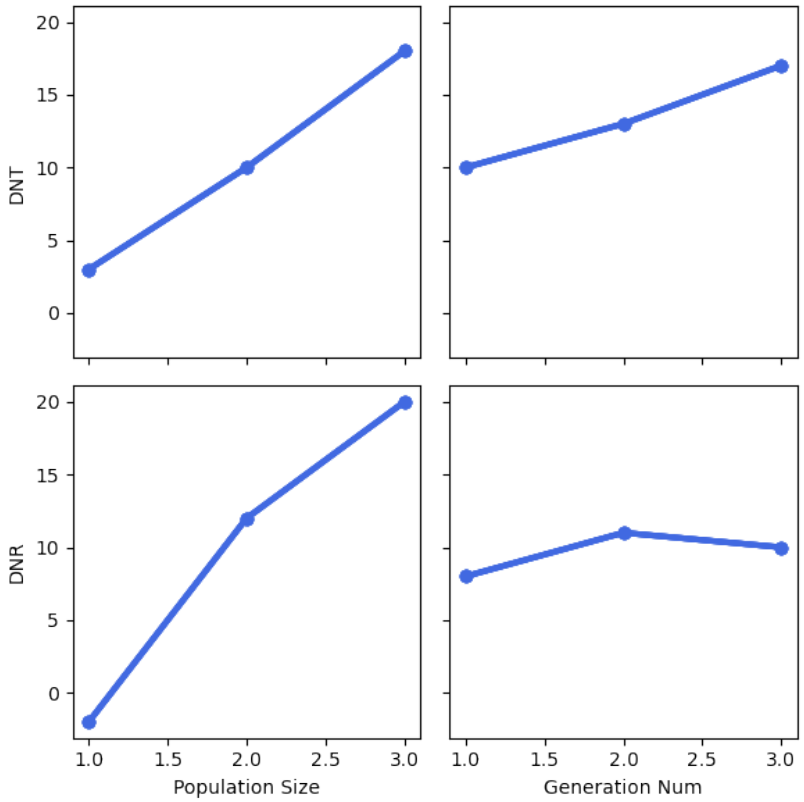

Figure 5.25: The mean S/N plotted for each chosen level of the parameters. The top row DNT represents the fitness function combination of the quantity of data transferred, number of changes, and page load time, and the bottom DNR represents the quantity of data transferred, number of changes, and memory usage.

1. The original webpage with no changes made.

2. A randomized set of actions that could potentially reduce the carbon footprint.

The first baseline shows that the carbon footprint reduction tool can truly reduce the carbon footprint. The second baseline aims to demonstrate further that the automated carbon footprint reduction tool is more effective than simply picking and applying random actions.

We conduct our experiments on a set of 10 websites of varying size and complexity with two different combinations of fitness functions:

1. **DNT:** (Quantity of Transferred Data) + (The Number of Changes) + (Page Load Time)

2. **DNR:** (Quantity of Transferred Data) + (The Number of Changes) + (Memory Usage)

Note that we perform one trial of random solution generation for every trial using the carbon footprint reduction tool. We refer to random trials paired with DNT-trials as “Random-DNT” and random trials paired with DNR-trials as “Random-DNR”.

Project Name	Random-DNT	Random-DNR	DNT	DNR
Poke-Dex	18.00	15.00	18.50	17.50
Ecommerce-Website-main	19.50	17.00	22.00	20.00
htmlBurger-website	13.50	12.50	16.50	16.00
complex-storm-html	9.50	9.50	12.50	13.00
module1-capstone-project	6.00	5.00	9.00	9.00
awesome-portfolio-websites	2.00	3.00	4.00	4.00
OpenWISP-Website	4.00	3.00	5.00	5.00
project-website-template	19.00	30.50	32.50	33.00
ProjectSakura	11.00	11.00	13.00	13.00
Books-bootstrap-website	4.00	8.50	12.00	13.00

Table 5.7: Median number of changes made by randomized solution generation and the automated carbon footprint reduction tool.

Project Name	Original Baseline	Random-DNT	Random-DNR	DNT	DNR
Poke-Dex	5158908	4942889	5069008	**4088047**	4181722
Ecommerce-Website-main	3021120	2136636	2167752	1477222	**1403839**
htmlBurger-website	1907546	1848460	1076644	928236	**881270**
complex-storm-html	645713	519497	444831	415664	**383898**
module1-capstone-project	947203	824963	824964	**803259**	818164
awesome-portfolio-websites	843424	767178	767155	**524879**	563573
OpenWISP-Website	1238487	766532	721795	736714	**716632**
project-website-template	5141268	4574004	4179609	4126205	**4110037**
ProjectSakura	1422532	1218049	1083724	**762353**	841990
Books-bootstrap-website	1734210	1090758	1026710	**651616**	1014127

Table 5.8: Median values for quantity of data transferred for the original website, randomized solution generation, and automated carbon footprint reduction tool. Lower values are better, and the lowest is bolded.

We compare median final values for the considered fitness functions. We also perform statistical analysis, first using the Mann-Whitney Wilcoxon rank-sum test to determine whether there is a significant statistical difference between the fitness values of the tool and the random baseline for a fitness function. In cases where statistically significant differences are observed, the Vargha-Delaney effect size test is applied to determine the magnitude of the observed differences.

Number of Changes: Table 5.16 shows the median number of changes made to solutions for each project by the randomized solution baseline and the automated carbon footprint reduction tool with the two fitness function configurations. The automated reduction tool tries to balance improvements in other performance metrics against the number of changes made to the page. We can see that, however, more changes are made by this tool than by random generation on average.

Median Fitness Values: Table 5.8 shows the median data transfer quantity for each baseline and the two fitness function configurations of the tool. The automated carbon footprint reduction tool's performance—generally for both fitness function configurations—is better than the original baseline and random solution generation for all subject projects. Random generation only attains comparable results for

	Load Time			Memory Usage		
Project Name	**Original**	**Random-DNT**	**DNT**	**Original**	**Random-DNR**	**DNR**
Poke-Dex	**3372**	3637	3526	275528	273394	**263079**
Ecommerce-Website-main	**1870**	1999	1883	189700	170972	**169447**
htmlBurger-website	1937	1870	**1836**	**163008**	165868	165013
complex-storm-html	2476	2474	**1873**	406188	444831	**402963**
module1-capstone-project	1540	1409	**1396**	228976	148444	**139848**
awesome-portfolio-websites	5802	6306	**5132**	238592	195604	**182057**
OpenWISP-Website	6077	985	**937**	172708	160578	**142629**
project-website-template	1743	1409	**1403**	273548	232752	**206689**
ProjectSakura	7052	6826	**6366**	223280	180134	**107218**
Books-bootstrap-website	2119	1859	**1855**	**173412**	182876	176933

Table 5.9: Median values for page load time and memory usage for the original website, randomized solution generated website, and automated carbon footprint reduction tool. Lower values are better, and the lowest is bolded.

`OpenWISP-Website`.

Performance (RQ3.1): In the median case, the automated carbon footprint reduction tool reduces the quantity of data transferred by the original webpage by 39.14% (DNT) and 40.68% (DNR) and by randomized changes by 25.42% (DNT) and 15.60% (DNR).

As discussed in Section 4.2.2.1, the transferred data and the carbon footprint value are positively correlated [12]. Therefore, this is an indication that the automated carbon footprint can reduce a webpage's carbon footprint significantly.

We also considered additional factors as part of the set of fitness functions optimized. In particular, we can examine the memory usage and page load time. As was also discussed in Section 4.2.2.1, memory usage has also been found to have a high correlation with energy consumption [64]. The page load time was considered not because of a relationship with carbon footprint—past research found that it was not always correlated [10]—but because long load times can create a negative user experience [2]. We, therefore, minimize page load time to balance carbon footprint reductions with potentially negative changes to user experience[1].

Table 5.9 shows the median page load time and memory usage for each baseline and the two fitness function configurations of the tool. We can see that the carbon footprint reduction tool's performance is better in both metrics for the majority of the subject projects, with the exception of the page load time of `Poke-Dex` and `Ecommerce-Website-main`, as well as memory usage of `htmlBurger-website` and `Books-bootstrap-website`.

Performance (RQ3.1): In the median case, the carbon footprint reduction tool reduces memory consumption of the website by 10.64% and of randomized changes by 3.96%. It reduces the load time of the original site by 14.05% and of randomized changes by 6.36%.

[1]We also minimize the number of changes made for the same reason.

Project Name	DNT	DNR
Poke-Dex	8.28e-08	2.08e-05
Ecommerce-Website-main	1.51e-06	2.04e-05
htmlBurger-website	3.11e-04	1.37e-03
complex-storm-html	2.33e-05	1.11e-04
module1-capstone-project	3.45e-07	2.25e-06
awesome-portfolio-websites	1.02e-05	9.04e-07
OpenWISP-Website	0.05	1.52e-03
project-website-template	2.27e-06	8.27e-08
ProjectSakura	5.49e-06	2.81e-04
Books-bootstrap-website	5.23e-07	3.27e-08

Table 5.10: P-Values for the Mann-Whitney-Wilcoxan test for data transfer quantity for carbon footprint reduction tool versus random solution generation.

Project Name	DNT	DNR
Poke-Dex	**0.99**	**0.88**
Ecommerce-Website-main	**0.93**	**0.88**
htmlBurger-website	**0.82**	**0.84**
complex-storm-html	**0.88**	**0.84**
module1-capstone-project	**0.96**	**0.93**
awesome-portfolio-websites	**0.89**	**0.94**
OpenWISP-Website	-	**0.78**
project-website-template	**0.93**	**0.99**
ProjectSakura	**0.91**	**0.82**
Books-bootstrap-website	**0.95**	**1.00**

Table 5.11: Effect sizes from Vargha-Delaney A Measure for data transfer quantity in cases where a statistically significant difference exists between the carbon footprint reduction tool and random solution generation. Large effect sizes are in bold.

We can see from the deviating cases that improvements to these three fitness functions do not always correlate. Changes intended to reduce the data transferred can reduce memory consumption and load time as well, but they can also have a slight negative impact in some cases.

Statistical Analysis: Table 5.10 shows that there is a statistical difference in the data transfer quantity between the carbon footprint reduction tool and random solution generation for almost all subject projects, with the exception of `OpenWISP-Website` with the DNT fitness function combination.

Table 5.11 shows that, in nine of the ten projects, the DNT fitness function combination outperforms random solution generation with a large effect size. In all of the ten projects, the DNR combination also outperforms random generation with a large effect size. The exact effect size varies from one project to another, but the automated footprint reduction tool is clearly more effective than random solution

Project Name	Page Load Time	Memory Usage
Poke-Dex	4.55e-07	0.023e-01
Ecommerce-Website-main	5.45e-04	0.052e-01
htmlBurger-website	1.29e-03	0.12
complex-storm-html	3.57e-04	0.27e-03
module1-capstone-project	5.02e-06	2.94e-05
awesome-portfolio-websites	6.25e-05	3.31e-05
OpenWISP-Website	0.01	2.34e-05
project-website-template	3.31e-03	7.22e-05
ProjectSakura	2.34e-04	1.98e-03
Books-bootstrap-website	2.24e-03	5.51e-06

Table 5.12: P-Values for the Mann-Whitney-Wilcoxan test for page load time and memory usage for carbon footprint reduction tool versus random solution generation.

Project Name	Page Load Time	Memory Usage
Poke-Dex	**0.96**	**0.76**
Ecommerce-Website-main	**0.80**	**0.76**
htmlBurger-website	**0.77**	-
complex-storm-html	**0.81**	**0.75**
module1-capstone-project	**0.91**	**0.87**
awesome-portfolio-websites	**0.80**	**0.87**
OpenWISP-Website	0.70	**0.88**
project-website-template	**0.75**	**0.85**
ProjectSakura	**0.83**	**0.77**
Books-bootstrap-website	**0.77**	**0.91**

Table 5.13: Effect sizes from Vargha-Delaney A Measure for page load time and memory usage in cases where a statistically significant difference exists between the carbon footprint reduction tool and random solution generation. Large effect sizes are in bold.

generation.

With regard to load time and memory usage, Table 5.12 shows that statistical differences exist for almost all subject projects, with the exception of the load time of `OpenWISP-Website` and the memory usage of `htmlBurger-Website`.

Table 5.13 further shows that the carbon footprint reduction tool outperforms random generation in each case where a statistical difference was observed. In terms of page load time, the tool outperforms random solution generation with a large effect size for nine projects and one with a medium effect size. In terms of memory usage, the tool outperforms random generation with a large effect size for nine of the ten projects. In the remaining case, there was insufficient evidence of a difference between the two configurations.

Project Name	P-Value	Effect Size
Poke-Dex	6.66e-03	**0.07**
Ecommerce-Website-main	0.104	-
htmlBurger-website	1.29e-05	**0.89**
complex-storm-html	2.58e-06	**0.92**
module1-capstone-project	1.47e-07	**0.03**
awesome-portfolio-websites	3.97e-08	**0.00**
OpenWISP-Website	3.97e-08	**1.00**
project-website-template	6.67e-03	**0.73**
ProjectSakura	6.17e-03	**0.27**
Books-bootstrap-website	3.93e-08	**0.00**

Table 5.14: Mann-Whitney-Wilcoxan and effect size results for data transfer quantity between the two fitness function configurations. Large effect sizes are in bold.

Performance (RQ3.1): The automated reduction tool outperforms random generation in data transfer quantity, load time, and memory usage for almost all projects, and with a large effect size in almost all cases.

We can also compare the two configurations of the automated reduction tool in terms of the quantity of data transferred—a fitness function shared in both configurations.

Table 5.14 shows that there are statistically significant differences in the results of the two fitness function configurations in nine of the ten projects. However, there is not a clear pattern in *which* tool is better. In four of ten projects, the DNR fitness function combination outperforms the DNT fitness function combination with a large effect size. However, in the other five projects where a distribution difference was noted, the DNT fitness function combination outperforms the DNR with regard to the quantity of data transferred.

It is not clear exactly why one fitness configuration may outperform another with regard to the quantity of data transferred. It may be that pursuit of one of the other fitness functions—in this case, memory usage or page load time—may offer feedback on how to also reduce the quantity of transferred data. Future research should explore both additional fitness functions as well as different combinations of fitness functions to identify the tool configurations most widely effective.

Actions Taken: We also can examine which actions are taken the most often by the automated carbon footprint reduction tool. The actions that can be taken include moving a script to the bottom of the page, removing CSS opacity properties, changing HTML Tags, removing unused CSS, compressing images, and converting image formats. Each action affects one element of a particular type at a time. For example, if there are multiple images on a page, then a single action could change one of those images.

Table 5.15 shows the percentage of final solutions where a particular type of action has been taken when targeting the DNT fitness function combination. Table 5.16

Project Name	Move Script(%)	Remove Opacity(%)	Change HTML(%)	Remove Unused CSS(%)	Compress Image(%)	Convert Image(%)
Poke-Dex	25.00	75.00	65.00	80.00	100.00	100.00
Ecommerce-Website-main	30.00	55.00	55.00	60.00	100.00	100.00
htmlBurger-website	35.00	35.00	55.00	65.00	95.00	100.00
complex-storm-html	45.00	40.00	50.00	50.00	100.00	100.00
module1-capstone-project	35.00	50.00	75.00	80.00	100.00	100.00
awesome-portfolio-websites	25.00	25.00	40.00	90.00	100.00	100.00
OpenWISP-Website	45.00	25.00	65.00	50.00	100.00	100.00
project-website-template	15.00	15.00	60.00	75.00	100.00	100.00
ProjectSakura	10.00	40.00	70.00	70.00	100.00	100.00
Books-bootstrap-website	25.00	55.00	55.00	90.00	100.00	100.00
Overall	28.50	52.00	59.00	71.50	99.50	100.00

Table 5.15: The percentage of final solutions that apply an action of a particular type for the **DNT** fitness function combination.

Project Name	Move Script(%)	Remove Opacity(%)	Change HTML(%)	Remove Unused CSS(%)	Compress Image(%)	Convert Image(%)
Poke-Dex	15.00	40.00	45.00	80.00	100.00	100.00
Ecommerce-Website-main	35.00	50.00	50.00	55.00	100.00	100.00
htmlBurger-website	30.00	50.00	65.00	55.00	100.00	100.00
complex-storm-html	15.00	60.00	65.00	70.00	100.00	100.00
module1-capstone-project	55.00	40.00	50.00	50.00	95.00	100.00
awesome-portfolio-websites	40.00	30.00	40.00	65.00	100.00	100.00
OpenWISP-Website	30.00	15.00	70.00	65.00	100.00	100.00
project-website-template	25.00	30.00	50.00	80.00	100.00	100.00
ProjectSakura	15.00	15.00	55.00	65.00	100.00	100.00
Books-bootstrap-website	10.00	50.00	65.00	80.00	100.00	100.00
Overall	26.50	43.00	57.00	62.50	99.50	100.00

Table 5.16: The percentage of final solutions that apply an action of a particular type for the **DNR** fitness function combination.

shows the same for the DNR fitness function combination.

As might be anticipated, changes to the images—both compression and format changes—are applied particularly often and clearly have an impact on the quantity of transferred data, memory usage, and page loading time. However, the other actions are applied as well. In particular, unused CSS is frequently removed, perhaps because many websites make use of existing templates and their creators do not optimize the templates.

Actions Taken (RQ3.2): Compressing, converting images and removing unused CSS are the most common actions applied by the automated carbon footprint reduction tool.

6

Discussion

In this chapter, we discuss the research questions based on the results from Chapter 5, additional observations that were not included in the results, and threats to validity.

6.1 Summary and Observations

6.1.1 Interview and Survey Study (RQ1)

To answer RQ1: *"How can the acceptance and voluntary adoption of carbon footprint reduction techniques by developers be improved?"*, interviews and a survey were conducted to assess software developers' experiences, prior knowledge, behaviours, and opinions related to the carbon footprint and energy consumption of software, along with their expectations and willingness to adopt automated carbon footprint reduction tools.

Current Knowledge and Practices: It was observed that many developers lack significant knowledge of the carbon footprint or energy consumption of software. However, developers do appear to gain more knowledge of these concepts as they gain experience. More Swedish survey participants had familiarity with both concepts than Japanese participants. A plurality of respondents was neutral on whether software carbon footprint contributes to climate change and whether carbon footprint should be considered and controlled. However, a greater proportion of participants agree with both than disagree.

The majority of participants feel that development organizations and regulatory agencies both bear responsibility for considering and controlling the energy consumption or carbon footprint of software. We suggest that these organizations could play a larger role in increasing the sustainability of the IT industry:

- These organizations can provide learning opportunities for developers to raise awareness, e.g., holding seminars with professionals working in climate and sustainability-related areas, or offering training programs that teach developers how to develop in a more ecologically sustainable manner.

- Industrial organizations can consider reducing energy consumption and carbon

footprint as part of planning project goals. They could impose organization-wide development policies to control their impact on the environment. Higher importance could be given to energy consumption and carbon footprint as quality goals of projects.

- Government organizations responsible for environmental protection could formulate stronger policies regarding the energy consumption and carbon footprint of the IT industry.

When energy consumption is considered or controlled, we observed that actions take place most often during the design and implementation phases. Carbon footprint is considered most often during deployment and design, but implementation and maintenance are both important as well.

Developers typically focus on improving performance or reducing resource consumption (reducing CPU, memory usage, page size, or upload frequency) to improve energy consumption. When carbon footprint is considered, it is usually addressed through the same actions and considered specifically in relation to energy consumption instead of separately. Energy consumption and carbon footprint are most commonly measured through CPU or memory usage. Energy consumption is most often evaluated using manual or exploratory testing, while carbon footprint is most commonly evaluated through measurements taken during Continuous Integration.

In future work, we will expand the scope and sample size of respondents to gather more information on current practices and make more detailed recommendations on best practices that developers should follow to ensure that software is energy and carbon-conscious from the start.

Requirements for Automated Carbon Footprint Reduction Tools: To trust a tool, reductions in carbon footprint must not compromise security, correctness, or other important software qualities. Security was given the highest priority. Changes to the code to reduce its carbon footprint should not leave information vulnerable or open vectors of attack. The tool should also ensure that the original code yields the same results without introducing new faults. Respondents also stressed the importance of the modified code still being readable and understandable by human developers.

In addition, developers would like it to be easy to integrate into their existing workflow. In particular, many respondents indicated that they would run this tool as part of a CI/CD pipeline either at non-peak times or after each commit. To support integration into a CI/CD pipeline, the tool should offer an API or CLI, and should be installable through a command-line package manager, e.g., `pip`. It should be easy to install and use, with sufficient guidance for both tasks.

To encourage adoption, the tool should have a track record of transparent and trustworthy results. The changes made by the tool should be interpretable—it should be possible to understand why and how the code was changed, and the

impact on carbon footprint should be possible to verify. Our suggestion is that the tool should provide a clear and trustworthy result that shows the best solutions as well as the values and changes in RAM usage, carbon footprint, and other factors.

The tool should also be open-source, so that others can understand or expand its capabilities. This would allow developers to investigate and better comprehend the capabilities of the tool—as well as allow them to adjust the tool for specialized use cases.

Contrasting Swedish and Japanese Respondents: We compared results between respondents working in Sweden and Japan—western and eastern developed countries. Software developers in Sweden are more knowledgeable about software carbon footprint and energy consumption than those in Japan. There may be more opportunities for developers working in Swedish IT companies to acquire knowledge related to this field. However, we did not investigate their sources of knowledge.

Software developers in Sweden and Japan had similar feelings on the contribution of software to climate change and on whether software carbon footprint should be considered and controlled. However, they placed different emphases on where responsibility should lie. Japanese respondents place the strongest emphasis on the development organization, which should be responsible for its own products. Swedish respondents place a stronger emphasis on regulatory agencies that formulate legal policies.

Respondents from both countries believe that organizations should take greater responsibility than individuals for changing the status quo of software's carbon footprint and energy consumption. Swedish respondents also potentially have greater uncertainty on whether any party bears responsibility. Across all three parties—developers, companies, or agencies—lower proportions of Swedish respondents believed that any party was responsible than Japanese respondents. A higher proportion of Swedish respondents also specifically stated that no one was responsible for considering or controlling their carbon footprint.

Participants from Japan and Sweden both had an interest in trying an automated carbon footprint reduction tool. However, respondents from Sweden were more skeptical of a tool's potential for success and were less likely to trust the changes a tool made.

6.1.2 Prototype Tool Design (RQ2–3)

To answer RQ2: *"What factors of webpage design have an impact on the carbon footprint?"*, the factors in HTML, CSS, and JavaScript identified include the location in the code where JavaScript is invoked, CSS properties—e.g., opacity—HTML tags—e.g., `<span>`—unused code, image format, and image size. The automated carbon footprint reduction tool incorporates these factors as actions to reduce the carbon footprint.

We addressed RQ3—*"What impact does genetic programming have on the carbon footprint of webpages?"*—by conducting parameter tuning and experiments with our tool on a variety of webpages. In the final experiments, we demonstrated that the automated carbon footprint reduction tool can reduce the carbon footprint of the original webpage, and that genetic programming can yield better results than making random changes.

Our tool outperforms these baselines with significance—often with large effect size—for most webpages in terms of reducing the quantity of data transferred and memory consumption. Both factors have been found in past research to correlate with energy consumption (and, thus, carbon footprint). The tool also ensures that minimal changes are made, and can often reduce—or maintain—the load time of the page, not worsening user experience. Converting image formats, compressing images, and removing unnecessary CSS are the most frequent actions taken to reduce the carbon footprint.

Comparison to Tool Requirements from Interviews and Survey: Interviews and surveys conducted in the first part of the research provided support for the design and development of the second part of the research. The final version of the automated carbon footprint reduction tool meets many of the requirements mentioned by the respondents in the interviews and survey. In particular, we highlight the following aspects:

1. **The tool ensures code operates correctly**. By utilizing the tool, a new version of the webpage can be generated with modified HTML, CSS, and JavaScript files, while keeping the original files intact. The modified code maintains the original logic, ensuring that the generated file functions correctly and is consistent with the original version. During the final experiments, it was observed that the modified version of the web project did not contain any errors, indicating that the tool effectively optimizes the code without introducing any new issues. This ensures that the webpage continues to operate correctly and maintains its intended functionality after applying the modifications generated by the tool.

2. **The tool maintains the security of the system**. As noted above, the changes made do not change the underlying functionality of the page. In addition, the source code of the tool is available on GitHub, providing transparency and allowing for review of the code for security purposes.

3. **The tool is available for free**. There are no limitations to the use of this tool since it is open source on GitHub with an MIT license. The MIT license is a permissive license that allows for the free use, modification, and distribution of the software without imposing many restrictions on the users.

4. **The tool is provided with detailed instructions on how to use it**. In the markdown file `README.md`, detailed instructions are provided on how to use the tool, including the installation process, parameter settings, script usage,

and explanations of input and output. This can be helpful for ensuring the correct and successful usage of the tool.

5. **The tool can be easily integrated into the workflow of a web project**. The tool can be applied as part of a CI/CD pipeline or other command-line process. To get started, users can create a virtual environment within their CI/CD pipeline to isolate the dependencies and ensure a controlled environment for running the tool. Dependencies can be installed using the `pip` command or by batch installing them from a `requirements.txt` file using `pip install -r requirements.txt`.

   The automated carbon footprint reduction tool is also available as a `pip` package, making it easy to install and use. Users can install the tool by running the following command: `pip install autocarbonreductool==1.0`, following the instructions provided in the guidance. This makes it convenient and efficient to incorporate the tool into the existing workflow, helping to optimize the environmental sustainability of the web project without adding significant overhead to the development and deployment process. Overall, the process of installing and integrating the tool into the CI/CD pipeline is straightforward and can be done quickly.

6. **The results provided by the tool are clear and trustworthy**. The tool offers the Pareto frontier, as well as fitness values for both the original and generated versions. This information can be valuable for evaluating the performance and impact of modifications made using the tool.

7. **There is no additional work required to use this tool**. Using this tool requires minimal additional effort to set up and use. To utilize the tool, the user only needs to provide three inputs—the location of the local web project, the URL of the webpage to be analyzed, and the parameters for initialization, which include the initial population size and the number of generations. The initialization parameters can also be set as default parameters.

6.2 Threats to Validity

Validity refers to the accuracy of a method in measuring what it is intended to measure. Several threats to validity are presented in this section, including threats to construct, external, and internal validity [70].

Construct Validity: Construct validity refers to the extent to which a test measures the concept that it is intended to measure. There are primarily two threats to constructing validity in our study.

In interviews and surveys, subject bias may be present. Since participants and interviewees have been provided with an introduction to the interview or questionnaire, they are familiar with the topic under study. By holding expectations about the study, their responses and choices may be influenced by their own bias. Thus, we

may be unable to accurately measure what truly interests us.

Furthermore, since this interview study is mainly conducted by a single author, this author's expectations may negatively affect the results. To counter this threat, the author's supervisor assisted by independently coding two interviews. We compared the difference between codes and had a discussion with them to eliminate any disagreements. As we cannot discuss and identify every transcript's code for every interview, there may be portions of transcripts that should be coded but are not.

External Validity: Generally, external validity refers to the extent to which the study results can be generalized to broader situations and populations.

Our study focuses only on measuring and reducing the carbon footprint of the webpage as well as modifying the source code of HTML, CSS and JavaScript. We are not focusing on other software and programming languages, such as mobile applications and PC software. A great deal of research can be conducted on reducing software energy consumption and carbon footprint across various application scenarios.

In addition, sampling bias may be present during the interview process. Participants were found on LinkedIn. The participants who accepted the invitation and agreed to share their ideas are likely to be interested in the area of software carbon footprint. It is possible, however, that many people without responses have more experience in such areas, which might indicate that the sample is not representative of the overall population.

Internal Validity: When it comes to internal validity refers to the fact that the trustworthy causal relationship being tested is not affected by other factors. To mitigate the risk of bias, we tried to ask questions and collect interview data without influencing the participants.

To ensure the internal validity of the interview and survey process and avoid a "self-fulfilling prophecy" during the design of the interview questions, we conducted a pre-testing interview. A master's student with a background in web development and a sustainable environment was interviewed without preconceived notions about the expected answers to questions. The questionnaire was pre-tested by a software developer working in a Japanese IT company with five years of development experience. Through this procedure, we adjusted the length of the interview and questionnaire. In addition, we improved the quality of the interview and questionnaire questions by removing redundant questions and refining the wording to make them more nonaligned. It can also help eliminate the subjective bias associated with interview questions by modifying the terminology and structure.

The main limitation of the internal validity of interviews and surveys is bias. Considering that only one author primarily designed, conducted, and analyzed the interviews and surveys, it is essential to avoid a "self-fulfilling prophecy" when collecting and analyzing data. Therefore, the supervisor assisted and took an active role in developing the interview and survey guides. In addition, the supervisor gave feedback

on the analysis and independently coded two interviews to assess the reliability of the coding process.

Reliability: In research, reliability refers to the consistency of measurement. In order to ensure the reliability of the thematic analysis as much as possible, in addition to following the guidelines provided by Braun et al. [15], whose paper identifies the process and essential steps clearly, we used the method that one author designs the code and theme mainly and the others sample some interview transcripts to perform an independent coding. We discussed inconsistent codes and made further modifications.

7

Conclusion

We investigated the carbon footprint of software in this study, focusing on how to reduce webpage carbon footprints automatically and what software developers think of the technique. The project was divided into two parts. One sub-project involved designing and conducting interviews and survey studies with software developers primarily based in Sweden and Japan to understand developers' knowledge and opinions on the carbon footprint of software and to gather the requirements for voluntary adoption an use of software tools that automatically reduce the carbon footprint of software. The other sub-project was, based on these requirements, developing a prototype tool for automatically reducing a webpage's carbon footprint.

For the first sub-project, we interviewed ten software developers from Sweden and Japan, respectively, and received forty responses to a survey from developers all over the world. The qualitative data gathered from the interviews were analyzed with thematic analysis, while the quantitative data collected from the questionnaires were analyzed with descriptive statistics.

We observed that many developers lack knowledge. However, some had a basic understanding of factors affecting energy consumption and gain knowledge over time. Many felt neutral on whether software contributes to climate change and whether carbon footprint should be controlled. However, a greater proportion agrees than disagrees with both. The majority of participants feel that both development organizations and regulatory agencies bear responsibility for controlling their carbon footprint. We also explored when energy and carbon footprint are considered, how they are measured, and what practices are used to perform evaluations of both.

Many participants are willing to try an automated tool, but there was significant skepticism and a lack of trust. It is clear, however, that such tools must not compromise security, correctness, or other important qualities. They also must integrate into a CI pipeline, be well-documented, be reasonably priced or—preferably—open source, and offer transparent and trustworthy results.

In the second sub-project, we developed an automated tool for reducing carbon footprints. The entire process was divided into three iterations. During the first iteration, we identified the fitness function that would be used to calculate the carbon footprint of the webpage, including the quantity of data transferred, the

amount of memory used, the amount of time it took for the page to load, and the number of changes made to the page. In addition, a set of actions that may affect the carbon footprint of a webpage were identified. In the second iteration, we developed a tool based on multi-objective optimization utilizing the NSGA-II genetic algorithm. In the third iteration, we conducted parameter tuning and final experiments using the tool, comparing to two main baselines—the original webpage and randomly generated changes.

The final experiments show that the automated carbon footprint reduction tool can reduce the carbon footprint of webpages, yielding significantly lower quantities of transferred data, page load time, and memory usage than both baselines. The most common changes made include converting image format, compressing images, and removing unused CSS.

Our study provides a foundation for future research addressing software's carbon footprint. Attention should be paid in future work to increasing developer awareness of these issues, offering recommendations and best practices, exploring policy implications, and developing automated tool support.

Our prototype tool also offers a basis for future exploration of automated tool support. In the future, we recommend expanding the scope of the tool and experiments to encompass other types of software, a wider variety of fitness functions and actions, and further exploration of search budgets and parameter settings. In particular, our tool focuses purely on the reduction of energy consumption. This is only one aspect of the carbon footprint—another is the location where that energy is consumed. In future work, we will also incorporate this aspect into the tool—e.g., considering locating elements of a program at different data centers.

Bibliography

[1] Nadine Amsel and Bill Tomlinson. Green tracker: a tool for estimating the energy consumption of software. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 3337–3342. 2010.

[2] Daniel An. Find out how you stack up to new industry benchmarks for mobile page speed. *Think With Google*, 2018.

[3] Anders SG Andrae. New perspectives on internet electricity use in 2030. *Engineering and Applied Science Letters*, 3(2):19–31, 2020.

[4] Anders SG Andrae and Tomas Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157, 2015.

[5] Earl T Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke. Automated software transplantation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 257–269, 2015.

[6] Mohadese Basirati, Romain Billot, and Patrick Meyer. Two parameter-tuned multi-objective evolutionary-based algorithms for zoning management in marine spatial planning. 2022.

[7] Tom Bawden. Global warming: Data centres to consume three times as much energy in next decade experts warn. Available at: `https://www.independent.co.uk/climate-change/news/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html`, 2016. Accessed: 12.11.2022.

[8] Bobby R Bruce, Justyna Petke, and Mark Harman. Reducing energy consumption using genetic improvement. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1327–1334, 2015.

[9] Andreas Bruns, Andreas Kornstadt, and Dennis Wichmann. Web application tests with selenium. *IEEE software*, 26(5):88–91, 2009.

[10] Yi Cao, Javad Nejati, Pavan Maguluri, Aruna Balasubramanian, and Anshul Gandhi. Analyzing the power consumption of the mobile page load. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement*

and Modeling of Computer Science, pages 369–370, 2016.

[11] Bino Catasús, Sofi Ersson, Jan-Erik Gröjer, and Fan Yang Wallentin. What gets measured gets... on indicating, mobilizing and acting. *Accounting, Auditing & Accountability Journal*, 20(4):505–521, 2007.

[12] Tim Frick Chris Adams, Rym Baouendi. Website carbon calculator. 2022.

[13] Paolo Ciancarini, Shokhista Ergasheva, Zamira Kholmatova, Artem Kruglov, Giancarlo Succi, Xavier Vasquez, and Evgeniy Zuev. Analysis of energy consumption of software development process entities. *Electronics*, 9(10):1678, 2020.

[14] John M Cimbala. Taguchi orthogonal arrays. *Pennsylvania State University*, pages 1–3, 2014.

[15] Victoria Clarke, Virginia Braun, and Nikki Hayfield. Thematic analysis. *Qualitative psychology: A practical guide to research methods*, 222(2015):248, 2015.

[16] Daniela S Cruzes and Tore Dyba. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*, pages 275–284. IEEE, 2011.

[17] Sarah Darby et al. The effectiveness of feedback on energy consumption. *A Review for DEFRA of the Literature on Metering, Billing and direct Displays*, 486(2006):26, 2006.

[18] Victor De La Luz, Mahmut Kandemir, and Ibrahim Kolcu. Automatic data migration for reducing energy consumption in multi-bank memory systems. In *Proceedings 2002 Design Automation Conference (IEEE Cat. No. 02CH37324)*, pages 213–218. IEEE, 2002.

[19] André Assis Lôbo de Oliveira, Celso Gonçalves Camilo-Junior, and Auri MR Vincenzi. A coevolutionary algorithm to automatic test case selection and mutant in mutation testing. In *2013 IEEE Congress on Evolutionary Computation*, pages 829–836. IEEE, 2013.

[20] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[21] Chrome Developers. Remove unused css. Available at: `https://developer.chrome.com/docs/lighthouse/performance/unused-css-rules/`, 2020. Accessed: 09.11.2022.

[22] Google Developers. Google forms: Free online surveys for personal use. Available at: `https://www.google.com/forms/about/`, 2022. Accessed: 06.10.2022.

[23] Markus Dick, Eva Kern, Jakob Drangmeister, Stefan Naumann, and Timo Johann. Measurement and rating of software-induced energy consumption of desktop pcs and servers. In *EnviroInfo*, pages 290–299, 2011.

[24] Jonathan Dorn, Jeremy Lacomis, Westley Weimer, and Stephanie Forrest. Automatically exploring tradeoffs between software output fidelity and energy costs. *IEEE Transactions on Software Engineering*, 45(3):219–236, 2017.

[25] Carla Schlatter Ellis. The case for higher-level power management. In *Proceedings of the seventh workshop on hot topics in operating systems*, pages 162–167. IEEE, 1999.

[26] Ilker Etikan, Sulaiman Abubakar Musa, Rukayya Sunusi Alkassim, et al. Comparison of convenience sampling and purposive sampling. *American journal of theoretical and applied statistics*, 5(1):1–4, 2016.

[27] Göran Finnveden, Michael Z Hauschild, Tomas Ekvall, Jeroen Guinée, Reinout Heijungs, Stefanie Hellweg, Annette Koehler, David Pennington, and Sangwon Suh. Recent developments in life cycle assessment. *Journal of environmental management*, 91(1):1–21, 2009.

[28] Xiang Gao, Bo Wang, Gregory J Duck, Ruyi Ji, Yingfei Xiong, and Abhik Roychoudhury. Beyond tests: Program vulnerability repair via crash constraint extraction. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(2):1–27, 2021.

[29] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. Automated program repair. *Communications of the ACM*, 62(12):56–65, 2019.

[30] Kathy Haan. How many websites are there? Available at: `https://www.forbes.com/advisor/business/software/website-statistics/`, 2023. Accessed: 04.10.2022.

[31] Karen Hao. Training a single ai model can emit as much carbon as five cars in their lifetimes. *MIT technology Review*, 2019.

[32] Mark Harman, William B Langdon, Yue Jia, David R White, Andrea Arcuri, and John A Clark. The gismoe challenge: Constructing the pareto program surface using genetic programming to find better programs (keynote paper). In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–14. IEEE, 2012.

[33] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. 2009.

[34] Don Ho. What is notepad++. Available at: `https://notepad-plus-plus.org/`, 2022. Accessed: 08.11.2022.

[35] PageSpeed Insights. Pagespeed insights. *Análisis de sitio web Mi Alegría*, 2020.

[36] Masoud Kargar, Ayaz Isazadeh, and Habib Izadkhah. Semantic-based software clustering using hill climbing. In *2017 International Symposium on Computer Science and Software Engineering Conference (CSSE)*, pages 55–60. IEEE, 2017.

[37] Shyam Kumar Karna, Rajeshwar Sahai, et al. An overview on taguchi method. *International journal of engineering and mathematical sciences*, 1(1):1–7, 2012.

[38] Vineet Khare, Xin Yao, and Kalyanmoy Deb. Performance scaling of multi-objective evolutionary algorithms. In *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, Faro, Portugal, April 8–11, 2003. Proceedings 2*, pages 376–390. Springer, 2003.

[39] Kirupa. Running your code at the right time. Available at: `https://www.kirupa.com/html5/running_your_code_at_the_right_time.html`, 2020. Accessed: 01.12.2022.

[40] Jonathan G Koomey et al. Estimating total power consumption by servers in the us and the world, 2007.

[41] John R Koza and Riccardo Poli. Genetic programming. In *Search methodologies*, pages 127–164. Springer, 2005.

[42] Patricia Lago. Challenges and opportunities for sustainable software. In *2015 IEEE/ACM 5th International Workshop on Product Line Approaches in Software Engineering*, pages 1–2. IEEE, 2015.

[43] William B Langdon and Mark Harman. Optimizing existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*, 19(1):118–135, 2014.

[44] Anne Lazaraton. Current trends in research methodology and statistics in applied linguistics. *TESOL quarterly*, 34(1):175–181, 2000.

[45] Louis Lazaris. An introduction and guide to the css object model (cssom). Available at: `https://css-tricks.com/an-introduction-and-guide-to-the-css-object-model-cssom/`, 2018. Accessed: 08.11.2022.

[46] Xuan-Bach D Le. Towards efficient and effective automatic program repair. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 876–879. IEEE, 2016.

[47] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. Genprog: A generic method for automatic software repair. *Ieee transactions on software engineering*, 38(1):54–72, 2011.

[48] Kui Liu, Li Li, Anil Koyuncu, Dongsun Kim, Zhe Liu, Jacques Klein, and

Tegawendé F Bissyandé. A critical review on the evaluation of automated program repair systems. *Journal of Systems and Software*, 171:110817, 2021.

[49] Fan Long and Martin Rinard. Automatic patch generation by learning correct code. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 298–312, 2016.

[50] Irene Manotas, Lori Pollock, and James Clause. Seeds: A software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering*, pages 503–514, 2014.

[51] Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. Angelix: Scalable multiline program patch synthesis via symbolic analysis. In *Proceedings of the 38th international conference on software engineering*, pages 691–701, 2016.

[52] Radhika Mittal, Aman Kansal, and Ranveer Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 317–328, 2012.

[53] Vojtech Mrazek, Zdenek Vasicek, and Lukas Sekanina. Evolutionary approximation of software for embedded systems: Median function. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 795–801, 2015.

[54] Dorothy Neufeld. The 50 most visited websites in the world. Available at: `https://www.visualcapitalist.com/the-50-most-visited-websites-in-the-world.html`, 2021. Accessed: 06.11.2022.

[55] Yannic Noller, Ridwan Shariffdeen, Xiang Gao, and Abhik Roychoudhury. Trust enhancement issues in program repair. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering*, 2022.

[56] Daniel G Oliver, Julianne M Serovich, and Tina L Mason. Constraints and opportunities with interview transcription: Towards reflection in qualitative research. *Social forces*, 84(2):1273–1289, 2005.

[57] Zakaria Ournani, Romain Rouvoy, Pierre Rust, and Joel Penhoat. On reducing the energy consumption of software: From hurdles to requirements. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, 2020.

[58] Divya Pandey, Madhoolika Agrawal, and Jai Shanker Pandey. Carbon footprint: current methods of estimation. *Environmental monitoring and assessment*, 178(1):135–160, 2011.

[59] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E Hassan. What do programmers know about software energy consumption? *IEEE Software*, 33(3):83–

89, 2015.

[60] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[61] Justyna Petke, Saemundur O Haraldsson, Mark Harman, William B Langdon, David R White, and John R Woodward. Genetic improvement of software: a comprehensive survey. *IEEE Transactions on Evolutionary Computation*, 22(3):415–432, 2017.

[62] Justyna Petke, Mark Harman, William B Langdon, and Westley Weimer. Using genetic improvement and code transplants to specialise a c++ program to a problem class. In *European Conference on Genetic Programming*, pages 137–149. Springer, 2014.

[63] Olivier Philippot. Which image format choose to reduce energy consumption and environmental impact? Available at: `https://greenspector.com/en/which-image-format-to-choose-to-reduce-its-energy-consumption-and-its-environmental-impact/`, 2022. Accessed: 15.01.2023.

[64] Olivier Philippot, Alain Anglade, and Thierry Leboucq. Characterization of the energy consumption of websites: Impact of website implementation on resource consumption. In *ICT for Sustainability 2014 (ICT4S-14)*, pages 171–178. Atlantis Press, 2014.

[65] Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 22–31, 2014.

[66] R Venkata Rao. Introduction to multiple attribute decision-making (madm) methods. *Decision Making in the Manufacturing Environment: Using Graph Theory and Fuzzy Multiple Attribute Decision Making Methods*, pages 27–41, 2007.

[67] K Ratheeswari. Information communication technology in education. *Journal of Applied and Advanced research*, 3(1):45–47, 2018.

[68] F Schmidt Richard. Software engineering architecture-driven software development. *Massachusetts: Elsevier*, 2013.

[69] Giampaolo Rodola. Psutil package: a cross-platform library for retrieving information on running processes and system utilization. *Google Scholar*, 2016.

[70] P Runeson and M Höst. Department of computer science, lund university, institutionen för datavetenskap lunds universitet 2009," guidelines for conducting and reporting case study research in software engineering. *Empirical Software*

Engineering, 14(2):131–164.

[71] Amine Saboni. Codecarbon. Available at: `https://github.com/mlco2/codecarbon/`, 2020. Accessed: 02.02.2023.

[72] Adrian Sampson, Călin Caşcaval, Luis Ceze, Pablo Montesinos, and Dario Suarez Gracia. Automatic discovery of performance and energy pitfalls in html and css. In *2012 IEEE International Symposium on Workload Characterization (IISWC)*, pages 82–83. IEEE, 2012.

[73] Ridwan Shariffdeen, Yannic Noller, Lars Grunske, and Abhik Roychoudhury. Concolic program repair. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 390–405, 2021.

[74] Wiktor Stadnik and Ziemowit Nowak. The impact of web pages' load time on the conversion rate of an e-commerce platform. In *International Conference on Information Systems Architecture and Technology*, pages 336–345. Springer, 2017.

[75] Stefan Steinke, Lars Wehmeyer, Bo-Sik Lee, and Peter Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, pages 409–415. IEEE, 2002.

[76] Genichi Taguchi, Subir Chowdhury, and Yuin Wu. Taguchi's quality engineering handbook, john willey & sons. *Inc., Hoboken, New Jersey, USA*, 134, 2005.

[77] Hamed Taherdoost. How to design and create an effective survey/questionnaire; a step by step guide. *International Journal of Academic Research in Management (IJARM)*, 5(4):37–41, 2016.

[78] Juha Taina. How green is your software? In *International Conference of Software Business*, pages 151–162. Springer, 2010.

[79] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. Who killed my battery? analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50, 2012.

[80] David R Thomas. A general inductive approach for qualitative data analysis. 2003.

[81] András Vargha and Harold D Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[82] Gate Vidyalay. Genetic algorithm in ai, operators, and work-

ing. Available at :https://www.gatevidyalay.com/tag/flowchart-for-genetic-algorithm, 2020. Accessed: 06.01.2023.

[83] Jan vom Brocke, Alan Hevner, and Alexander Maedche. Introduction to design science research. In *Design science research. Cases*, pages 1–13. Springer, 2020.

[84] Jinxia Wang, Robert Mendelsohn, Ariel Dinar, Jikun Huang, Scott Rozelle, and Lijuan Zhang. The impact of climate change on china's agriculture. *Agricultural Economics*, 40(3):323–337, 2009.

[85] WRI Wbcsd. The greenhouse gas protocol. *A corporate accounting and reporting standard, Rev. ed. Washington, DC, Conches-Geneva*, 2004.

[86] Samuel Webb. Computers worse for environment than plane travel. Available at: https://www.independent.co.uk/climate-change/ict-computers-climate-change-carbon-footprint-b1917767.html, 2021. Accessed: 18.10.2022.

[87] David R White, John Clark, Jeremy Jacob, and Simon M Poulding. Searching for resource-efficient programs: Low-power pseudorandom number generators. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1775–1782, 2008.

[88] David Robert White. *Genetic programming for low-resource systems*. PhD thesis, University of York, 2009.

[89] Thomas Wiedmann and Jan Minx. A definition of 'carbon footprint'. *Ecological economics research trends*, 1(2008):1–11, 2008.

[90] Frank Wilcoxon. *Individual comparisons by ranking methods*. Springer, 1992.

[91] Nicholas C Zakas. Javascript in html. *Professional Javascript® for Web Developers*, pages 13–23, 2015.

[92] Yong Zheng et al. Multi-objective recommendations: A tutorial. *arXiv preprint arXiv:2108.06367*, 2021.

[93] Yuhao Zhu and Vijay Janapa Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–24. IEEE, 2013.

A

Interview Guides

A.1 Introduction

This semi-structured interview aims to investigate the opinions of developers regarding the use of automated technologies to reduce the carbon footprint of the software they build automatically. We will ask a series of questions related to this topic. The following is the project's basic information, including the background, purpose, and explanation of related terms

A.1.1 Background

Sustainable actions are essential for the world to adopt to conserve energy resources and support sustainable livelihoods. Software is a major contributor to creating a sustainable society. However, it is also a major contributor to climate change through, for example, the energy consumed by data centers. For example, web pages are some of the most-used pieces of software—almost everyone browses the internet daily. If the carbon footprint of popular web pages could be reduced, and if reduction of carbon footprint is made a normal part of development, then the impact on the environment of the IT industry could be reduced.

It may be possible to automatically reduce the carbon footprint of software, such as web pages. For example, Genetic Programming (GP) techniques generate potential patches (changes to the source code) to a program and assess their ability to improve the attainment of one or more measurable qualities. GP could be used to automatically reduce the carbon footprint of a web page. Over a series of generations of evolution, patches are generated and modified. Each member of the population is scored using one or more scoring functions that assess the minimization or maximization of different goals. After this process, the best patches are presented to the user to be applied to the program.

The goal of these interviews is to examine (a) the knowledge of developers of the potential carbon footprint and energy consumption of software, (b) the actions they take the measure or reduce carbon footprint and energy consumption, (c) their opinions on the use of automated tools to perform the reduction of carbon footprint or energy consumption, and (d), the constraints that would need to be met to

voluntarily adopt such tools.

A.1.2 Protocol

The interviews are conducted in 3 steps. The first step is a narrative part when we describe the background and purpose of the study and the interview's purpose and process to provide the participants with a sense of their roles and make them more familiar with the context of the interview. It also includes the confidentiality agreement with the participant. The second step is a semi-structured interview, beginning with questions about the participants' profiles, collecting their demographic information, experience working on web development, and knowledge of energy consumption. Then we continue with the interview questions that focus on the research questions discussed below. Finally, we conclude the interview with a post-question step. We answer the participants' questions and share information and references if the participants are more interested in software carbon footprint reduction. The interviews last 40-50 minutes and are conducted online via Zoom and in-person hybrid.

A.1.3 Term explanation

Carbon footprint means the number of greenhouse gases (including carbon dioxide and methane) generated by actions.

Automatic Program Repair (APR) refers to a family of automated technologies that produce patches that correct program faults. This is an emerging research area.

Genetic Programming (GP) is a specific technique that generates and evolves a population of program patches in a model similar to Darwinian evolution. Patches are scored and ranked according to one or more factors of interest, then a new population is formed based on the best solutions of the current population. This is one technique to perform APR or to otherwise automatically transform a program to better meet goals of interest.

A **fitness function** is a numeric scoring function based on the goals of performing an optimization. Fitness functions are minimized or maximized when performing an optimization process, such as GP.

A.2 Research Questions

We introduce several research questions to guide and keep track of our interview survey study. The overarching research question is as follows: **RQ0: What are the opinions of developers regarding the use of automated technologies to automatically reduce the carbon footprint of the software they build?**

To obtain more specific information, we refine this overall question into more detailed questions as follows:

1. RQ1: What do developers know already about the carbon footprint or energy consumption of software?

2. RQ2: Do developers consider the carbon footprint or energy consumption of their own software?

    - RQ2.1: At what development stages do they make this consideration?

    - RQ2.2: What actions have they taken to reduce their software's energy consumption?

    - RQ2.3: What measurements do they use to assess carbon footprint or energy consumption?

    - RQ2.4: Do they write test cases to evaluate carbon footprint or energy consumption?

3. RQ3: Would developers use and trust tools that automatically reduce the carbon footprint or energy consumption?

    - RQ3.1: How should a carbon footprint reduction tool fit into a developer's workflow?

    - RQ3.2: What constraints must be met for such a tool to be used and trusted?

    - RQ3.3: How can voluntary adoption of such techniques be encouraged?

With RQ1, we aim to know the participants' prior knowledge about carbon footprint and energy consumption. RQ2 overarchingly focuses on participants' work experience on consideration of energy consumption. RQ2.1 - RQ2.4 focus on knowing the technical details of participants' experience. RQ2.1 aims to know when they take reducing energy consumption into account. RQ2.2 - RQ2.4 look at the actions and measurements they take to reduce the carbon footprint. In RQ3, we aim to understand the impact and acceptance of the carbon footprint reduction tool for developers. RQ3.1 investigate the way of fitting the tool into the developer's workflow. RQ3.2 and RQ3,3 look at how can make developers use such a tool trusty and voluntarily.

To answer these questions, we will conduct an interview study followed by a questionnaire survey study. With the interview study, we aim to get an initial understanding of limited people's thoughts and attitudes. With the questionnaire survey, we aim to extend and improve the survey based on the result of interviews.

We use semi-structured interviews to give participants enough freedom to add follow-up questions to express in-depth ideas and opinions.

A.3 Participants

The population of this interview research mainly focuses on software developers who develop the website from Sweden and Japan. The sample group from the population comprises IT company employees and bachelor's or master's students with web development experience.

In order to obtain enough qualitative data, the number of interview participants is at least ten. Besides, ten is a decent number of participants close to the similar works' studied population.

For privacy and confidentiality, the process omits the participants' names and sensitive and private information but uses ID names ranging from P1 to P10.

A.4 Questions

Using semi-structured interviews is essential in our case. It allows identify the main questions defining the purpose of my investigation. The main questions will be predefined and prepared before the interview so the process can go fast and keep track of our research questions and topic. The questions and answers will be open, so we will not get a simple "yes" or "no" answer but go deep into every participant's answer with the follow-up questions and keep maintaining the theme of the main questions. We will divide the interview questions into three parts; every part is a sub-theme. The first part aims to know more about participants' experience and attitudes towards the software's energy consumption. The second part is more about technical details about reducing carbon footprint using genetic programming. The last part is to investigate developers' acceptance and voluntary adoption of carbon footprint reduction techniques. The following are the main questions we designed:

1. RQ1
    - What is your current role?
    - How much experience do you have in software development?
    - What do you know about the carbon footprint of software or the impact that software has on the environment?
    - What do you know about the energy consumption of software?
2. RQ2:
    - Do you think reducing energy consumption or carbon footprint is the responsibility of the developers of a piece of software?
    - Do you personally have to consider energy consumption or carbon footprint when developing software?

- At what development stages do you make this consideration?
- How did you assess the energy consumption or carbon footprint?
- What actions do you take to reduce energy consumption or carbon footprint?
- Do you write test cases to perform the evaluation of energy consumption or carbon footprint? If so, what process do you follow to design these test cases and how do you execute and evaluate the results of testing?

3. RQ3:

- What are your opinions on the use of automated technologies to reduce the carbon footprint or energy consumption of software that you develop?
- If available, would you use tools that automatically reduce the carbon footprint or energy consumption?
- How do you foresee such a tool fitting into your development workflow?
- When and how often would you use this tool?
- Would you trust the results of tools that automatically reduce the carbon footprint or energy consumption?
- What constraints would you require to be met in order to use and trust the results of such a tool?
- What considerations do you think would encourage you or other developers to adopt such techniques voluntarily?