Product Vision and Architecture Document of the Parameter Optimization Model for the Impact Framework

Dimitros Stamatios Bouras **(TL)**
Telma Gudmundsdottir
Tomas Kopunec
Jiafan Lou
Jiashun Lu
Mohamed Saleh
Yi-yu Wang
Derek Wang

January 12, 2024

Version History

Version	Date	Authors	Comments
1.0	29/11/2023	All group members	Initialise and first add contents
2.1	01/12/2023	Telma Gudmundsdottir	Refine background information
2.2	03/12/2023	Dimitrios Bouras	Add possible projects
2.3	05/12/2023	Mohamed Saleh	Add domain scenarios
2.4	06/12/2023	Wang Derek	Refine Business goals
3.0	12/12/2023	Dimitrios Bouras	Convert word to latex format
3.1	16/12/2023	All group members	Optimise each section of the report
3.2	23/12/2023	All group members	Optimise each part of the report based on the feedback
4.0	28/12/2023	Dimitrios Bouras	Add new Sections and re-organize existing
4.1	03/01/2024	All group members	Final changes and checks
4.2	10/01/2024	All group members	Feedback additional changes

Contents

1 Purpose and scope of the document **4**

2 Background **4**
- 2.1 Green Computing . . . 4
- 2.2 Carbon Impact of Software . . . 5
- 2.3 The Green Software Foundation . . . 5

3 Introduction to the Impact Framework (IF) **5**
- 3.1 Example User story . . . 6
- 3.2 Overview diagram . . . 7

4 IF Context and Requirements **7**
- 4.1 Goals . . . 7
- 4.2 Stakeholders . . . 8
- 4.3 Main components and Terminology . . . 9
- 4.4 IF Diagrams - Views . . . 10
- 4.5 IF Functional view . . . 10
- 4.6 IF Deployment view . . . 11

5 Using the IF **12**
- 5.1 Models . . . 12
  - 5.1.1 Official Models: . . . 12
  - 5.1.2 Unofficial Models: . . . 13
  - 5.1.3 IF Model Interface . . . 14
  - 5.1.4 Capability of Models . . . 14
- 5.2 Graphs . . . 16
- 5.3 Input: `Impl` files . . . 17
- 5.4 Calculating using the Impact Engine . . . 18
- 5.5 Output: `Ompl` files . . . 19
- 5.6 User story - Example of IF use . . . 19
  - 5.6.1 IF Input . . . 19
  - 5.6.2 IF Output . . . 23
- 5.7 Domain Story Figure . . . 26

6 Rejected Projects **26**
- 6.1 WebUI - Simulation Tool . . . 26
- 6.2 AWS model . . . 27
- 6.3 Right Sizing Simulation tool . . . 27

7 Project Vision and Requirements **28**
- 7.1 Final Project Description . . . 28
- 7.2 Project Goals . . . 29
- 7.3 Stakeholders . . . 30
- 7.4 User story for Parameter Optimization Model . . . 30
  - 7.4.1 IF with Parameter Optimisation Input . . . 31

7.4.2 IF with Parameter Optimisation Output . . . 31

8 Project Architecture and development plan **32**

8.1 Model Structure . . . 32

8.2 Heuristics used . . . 33

8.3 Development plan . . . 33

9 Evaluation Plan **34**

9.1 Evaluation Criteria . . . 34

9.2 Testing and Evaluation Plan . . . 35

Appendices **37**

A Secondary Stakeholders of the IF **37**

B Detailed Examples of Models **38**

B.1 Sci-e . . . 38

B.2 Cloud Instance Metadata . . . 39

B.3 Azure-Importer . . . 41

1 Purpose and scope of the document

This document describes a parameter optimisation model for the Impact Engine Framework (IF) using search-based software engineering. The model enhances IF's environmental impact measurement features by providing suggestions on how a given software can be improved. By analysing and fine-tuning software parameters, the model aims to find the best balance between environmental friendliness and operational efficiency to aid users in decision-making. The ultimate goal of this report is to provide a detailed description, enhance understanding, and serve as a reference for further development of the model.

The document is a collaborative effort between all members of the team. Firstly, it explains the importance of the project and provides relevant background information. The Impact Framework and the Green Software Foundation are introduced with relevant goals and stakeholders. An attempt is made to provide the readers with a clear overview of the current version of the Impact Framework by explaining the key terminology, providing examples, user stories, and a number of diagrams. The product evolved significantly during the initial stages, and many original ideas could not be implemented due to IF being in the alpha state. The ideas would rely on implementations that were likely to change in the near future. An overview of these projects and why they were not chosen is detailed in the report. Next, the project that was designed to be implemented is presented, with primary goals, stakeholders, product architecture, user stories, and an example presentation of how the developed model will work and how it will be used. Finally, development and evaluation plans for the next term are outlined.

The team frequently consulted with the client, the Green Software Foundation, to ensure that the project vision stayed relevant and valuable to the company for the foreseeable future. The primary contact points at Green Software Foundation were Sophie Trinder, Joseph Cook and Asim Hussain as well as Gadhu Sundaram from NTT Data UK. While writing the report, the team held three weekly meetings: one with the clients, one with the supervisor and one where only the team was present. Meetings with the clients were conducted via Google Meet, with additional correspondence done through email. During these meetings, the team could ensure that the project aligned with the client's interest and get further information regarding components like the system and requirements. In the team meetings, the team went over tasks that needed to be done and gave feedback for finished tasks. This time was also used to discuss possible problems or answers that team members might have. In the supervisor meetings the team asked for the supervisor's advice on how to handle any issues they encountered and the supervisor also provided feedback on the work that had been done so far.

2 Background

2.1 Green Computing

Information and communication technology is responsible for a significant amount of carbon emissions and energy consumption worldwide. It is expected that those numbers will increase significantly over the following years. Traditional IT systems tend to use energy-intensive hardware, which causes high power consumption and generates electronic waste. Moreover, some traditional IT systems use fossil fuels or other energy that is based on non-renewable sources [8].

Green computing is the action of developing software, systems, and other computing components to minimise their negative environmental impact while maintaining optimal performance. By using green computing, companies can reduce carbon emissions, minimise electronic waste and promote renewable energy sources [8].

An important part of green computing is measuring the carbon output, energy usage and other metrics. It provides a quantitative basis for understanding the impact of software and makes setting targets and tracking progress possible. Furthermore, creating a uniform way for companies to measure environmental impacts allows for more transparency and accountability in environmental sustainability efforts as well as comparison between systems of very different companies and projects. [8].

2.2 Carbon Impact of Software

The environmental impact of software is multifaceted and includes carbon emissions, water usage, and energy consumption. Several studies have explored the carbon impact of software and computing systems, each focusing on different aspects of the digital technology ecosystem. To name a few: a study on how digital technologies impact carbon emissions in Chinese cities [9], a proposed methodology for estimating a software project's environmental footprint throughout its lifecycle [10], and an analysis of computing systems carbon emissions [5].

Measuring the environmental impact of software can be complicated and nuanced. There are some existing methodologies for measuring this impact, such as the Greenhouse Gas (GHG) Protocol [5], Life Cycle Assessment (LCAs) [5, 10], and the ITU L.1410 standard [10]. However, these methodologies can be complicated and often require adaptation and a good understanding of usage. There is also no industrial standard for companies or developers to follow, making comparisons challenging.

2.3 The Green Software Foundation

The Green Software Foundation, a non-profit under the Linux Foundation, is dedicated to reducing software's ecological footprint through various projects. They aim to create a reliable network of people, tooling and standards to promote Green Software, which emits fewer greenhouse gases than standard software. The Foundation prioritises reducing carbon emissions over neutralising them, with a focus on reducing software-related carbon emissions. It consists of a diverse group of organisations and individuals dedicated to developing sustainable and low-carbon software and provides a platform for collaboration and knowledge-sharing among those that advance the field of Green Software.

Since its establishment, the Foundation has worked on a multitude of projects, each governed by different groups. For instance, the Standards Working Group is responsible for crafting baseline specifications for Green Software. At the same time, the Open Source Working Group focuses on improving Green Software monitoring and measurement by developing open-source tools. One notable project, led by the Standards Working Group, is The Software Carbon Intensity (SCI) Specification, which outlines a methodology for calculating a software system's total carbon emissions.

3 Introduction to the Impact Framework (IF)

As mentioned above, measuring the environmental impact of software has become increasingly complicated. Modern software applications are intricate, consisting of numerous smaller software components operating across diverse platforms. These platforms encompass a diverse spectrum, spanning private and public clouds, bare-metal, virtualized, and containerized environments, in addition to various mobile devices, laptops, and desktops. Each of these necessitates a distinct measurement approach, complicating the process of calculating carbon emissions uniformly across all platforms.

Companies and developers are increasingly aware of the environmental implications of their software. They are seeking methods to quantify and minimise these impacts, both for ecological reasons and financial efficiency. There is a need to identify configurations and designs that minimise environmental impact while balancing factors such as performance, cost, and sustainability. A significant challenge is the lack of standard ways to measure and compare energy usage and environmental impact across different companies and software products.

A key project of the Green Software Foundation is the Impact Engine Framework (IF), which aims to model, measure, simulate and monitor the environmental impact of software. This project has evolved throughout the Green Software Foundation's existence, and a notable development was the creation of the Software Carbon Intensity (SCI) Specification. Initially, the challenge was data acquisition, leading to the launch of a project focused on generating necessary data sets. The team later realised various existing data sources were already available. For example, Our World in Data has measurements of the carbon intensity of electricity based on location from the year 2000 [6]. They anticipated that the challenge would shift from sourcing data to selecting the appropriate data set for each use case. This initiative eventually evolved into the SCI Guide, which provides comprehensive documentation on existing data sets and their application.

The project's current phase aims to establish formal standards and tooling, transforming software measurement into a disciplined and widespread practice. The SCI Specification is positioned as the fundamental standard, while the Impact Framework serves as the essential tooling component.

A crucial part of the IF is composability and modularity. To make the framework as composable and configurable as possible, a component called *model* can be plugged into the framework for additional functionality. A standard library of models is maintained by the IF core team, and another repository of unofficial models is maintained by the community. The models are created to be reusable and modular, allowing users more flexibility. This enables users to create models with the functionality that they need and share them with other users. For example, one of the standard models is a model called SCI-E, which calculates the total energy consumption of a component by adding up all its individual energy contributions.

This report not only clearly documents the current functionality of the IF, but additionally expands on it with a new model that goes beyond calculating software's possible environmental impact. With our proposed model, we aim to help the user even further by using search-based software engineering to analyse and optimise parameters in the form of input data. The model's output will contain the result and provide suggestions on how the user can change their input parameters to reduce the environmental impact of the software.

3.1 Example User story

In this section, a small user story is presented that will allow the readers to get a better understanding of how the IF is currently used. The same user story will be used throughout the document, first by adding new details when more intricate components of the IF have been explained thoroughly in section 5.6 and then by altering it as necessary to describe the use case of the new model for parameter optimisation, developed by our team in section 7.4.

User story:
As a multinational computer technology company, we specialise in database software and cloud engineered systems. Cloud services run on our servers, and they are also responsible for storing data. Our goal is to utilise the IF to measure carbon emissions of those servers. We want to be able to easily break down the details of the complex architecture of our system and to have a standard way of measuring our carbon footprint and compare it to similar companies that specialise in our field.

The IF requires a YAML file as input, which describes the architecture of the system as well as the IF models that will be used in the measurement, and in the order in which they must be triggered.

After invoking the Impact engine with the YAML file as input, the produced output will be another YAML file with new fields and values describing the carbon emissions of the servers as well as other environmental metrics such as energy. A more detailed analysis of the Input as well as output YAML files with thorough examples can be found in section 5.6.

3.2 Overview diagram

The Figure 1 depicts the Impact Framework (IF), which acts as the core of the system that interacts with the software developers, environmental scientists, and the open-source community to analyse and improve the sustainability of software. It strictly follows the SCI Specification [4], is supervised and supported by the Green Software Foundation, utilises data from other cloud service providers, and incorporates external measurement models for real-world data.

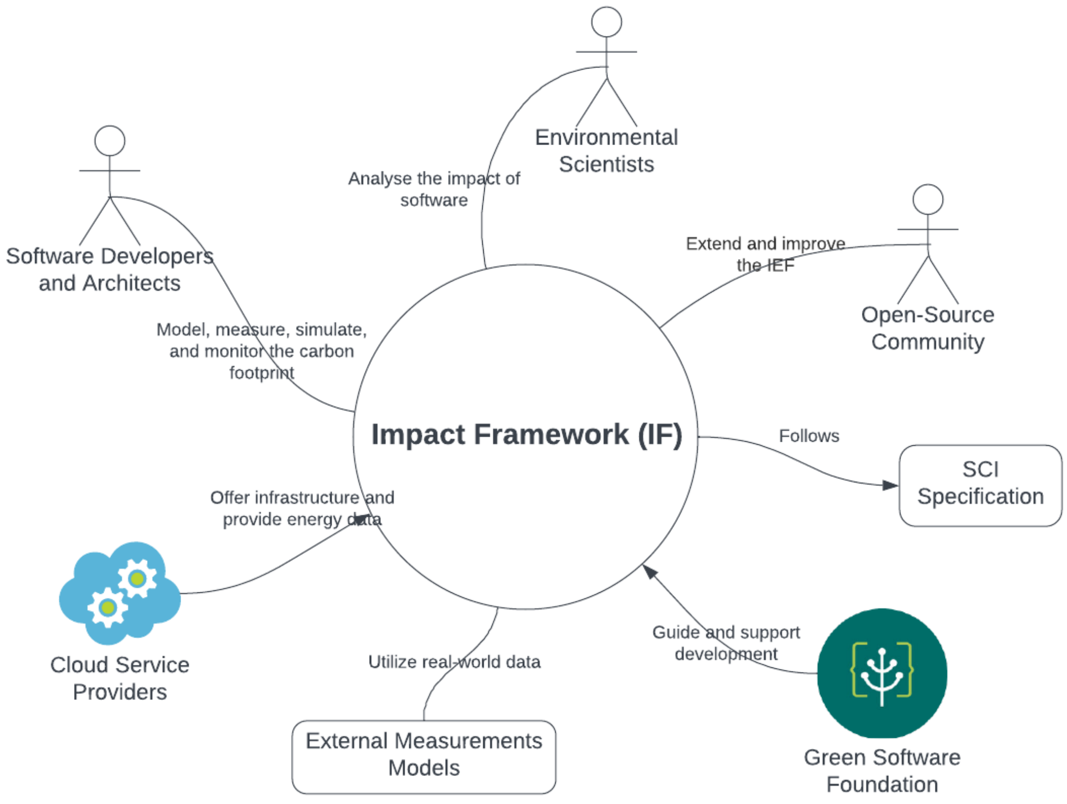
Environmental Scientists
Analyse the impact of software
Software Developers and Architects
Extend and improve the IEF
Open-Source Community
Model, measure, simulate, and monitor the carbon footprint
Impact Framework (IF)
Follows
SCI Specification
Offer infrastructure and provide energy data
Cloud Service Providers
Utilize real-world data
Guide and support development
External Measurements Models
Green Software Foundation

Figure 1: IF Overview

4 IF Context and Requirements

4.1 Goals

In this section, the Goals of the current Impact Engine Framework are presented. Majority of these goals are summarised from the discussion via the online meeting or email communication with the clients, and some are derived from the official documents in their GitHub repository. Overall these goals listed below represents the macro objectives of IEF project from the original developers. We the team will also try adherence these goals as much as possible.

1. **Decrease Software Environmental Impact**: The first and foremost goal of the IF is to help users and companies reduce the environmental impact of their software systems by providing a standard way of measuring said impact.

2. **Simplicity**:

   - **Eliminate technical barrier**: User friendliness is one essential point of attracting new and maintaining existing users. It should be ensured that users with non-technical backgrounds are able to understand everything in the IF.
   - **Hidden technical details**: No technical details should be exposed to users. Manipulation on technical operations (e.g. manifest file editing) should be abstracted and wrapped.

3. **User Retention**: Convince users to continue using the framework to reduce emissions. If the number of users using IF after their first few tries decreases, then the effects of emission reduction would be limited.

4. **Guarantee Data Security**: Since IF may require sensitive data from users, for example, enterprises' server configuration details, which can be considered as commercial secret; Or even locations of servers for national defence system, which could be a threat to national security in extreme situations. Hence privacy and data security is a concern for the users, especially for larger organisations, such as private companies or government agencies. We need to assure the users that IF will not endanger the privacy of their data.

5. **Expand the scope**:

   - **Add more cloud providers**: One of the primary goals is to expand the scope, including the scope of user adoption, and the scope of the sources of data used in IF. Currently, the system only supports Azure as a cloud provider, but in the future it aims to add support for AWS, GCloud, and other cloud providers. This would improve the usefulness of the IF since it will be relevant to a larger user base.

6. **Expand the User Base**:

   - **Advocate more users to use IF**: Attract and advocate more users to start using the IF. This would increase the total carbon footprint that teh IF could help reduce.
   - **Promote IF through collaboration**: Establish partnerships with key industry figures, technology companies, and academic institutions to publicise and promote the IF to a wider audience.

4.2 Stakeholders

In this section the primary stakeholders of the current implementation of the Impact Engine Framework are presented.

1. **Anyone involved in software development**:

   - Use the framework to assess and optimise the environmental impacts of their applications.
   - Are involved in the design and coding of software systems and interested in using more efficient algorithms.
   - Hope for a user-friendly and intuitive framework[7].

2. **Green Software Foundation (GSF)**:

   - Mission to reduce the total global carbon emissions associated with software[3].

- Concerns about development and operation costs, reputation, and strategic objectives.
- Expects software environmental impact measurement to become mainstream practice, with the framework providing crucial insights into that[2].

3. **Open-source community**:
   - Cares about compliance with open-source licenses for transparency and community involvement [7].
   - Hopes for contributions, engagement, and adoption by all developers in the open-source community [7].
4. **Large corporations**:
   - Might use the environmental impact as a selling point.
   - IF can help optimise the energy efficiency of their software, simultaneously saving funds by decreasing energy consumption.
   - Care about the availability of IF on all environments and platforms [7] [1].

See Appendix A, where a table for secondary stakeholders is presented, for more information about IF's stakeholders.

4.3 Main components and Terminology

In this section, the main components of the IF are briefly explained and summarised. More detailed explanations will be provided in the later parts of the report in combination with detailed examples.

- **IF CLI:** The only interface currently available for interacting with the IF to utilise its functionalities.
- **IF Impact Engine:** Performs impact computations based on the input `Impl` file and generates a Manifest file containing the computed impact.
- **Impl Input Manifest File:** A formal report in YAML (or CSV) format that details all assumptions, inputs, and models used for calculating the impact. It describes the system to be measured and serves as an input.
- **Model:** A model converts an input into a specific output impact metric. The are the building blocks of the needed calculations.
- **Model Plugin Interface:** A common class interface that every Model Plugin needs to extend and implement. This interface allows for a pipeline architecture.
- **Model Plugin:** A module external to the IF exposing a class implementing the Model Plugin Interface. This code allows the interaction with a Model using an open-source standard interface.
- **Model Pipeline:** A chain of models that takes the `Impl` input and produces the `Ompl` output manifest files. Calculating the system's impact often requires using multiple models in sequence. Each model takes as input the outputs of the previous model in the chain, or inputs directly from the `Impl` file, with all the models working together to calculate impacts from inputs.
- **Ompl - Output Manifest File:** A formal report in YAML (or CSV) format that includes the impact results, as well as the original `Impl` input manifest file.
- **Standard Library of Models:** A standard library of models built and maintained by the IF core team.

- **Unofficial Library of Models:** An external repository of models that is developed and maintained by the community.
- **Graphs:** An Impact Graph is an outline of the structure of a software system, detailing its interconnected components.

4.4 IF Diagrams - Views

In the Figure 2, we see a brief visualisation of the process of using the IF along with some internal calculations that the Impact Engine performs. Firstly the user must create the `Impl` YAML file providing the necessary information on the system architecture, input parameters and models to use in the pipeline. Then the Impact engine is triggered through the CLI it provides and the Model Computation Pipeline begins. The models and their order in the pipeline (along with the necessary inputs for each model) are specified in the `Impl` file. The models receive inputs either directly from the `Impl` file or from the output of preceding models in the pipeline. Each model performs its designated computations and outputs the results, which are then passed on to the next model in the sequence and in the `Ompl` file where all the output information and measurements will be available to the user.

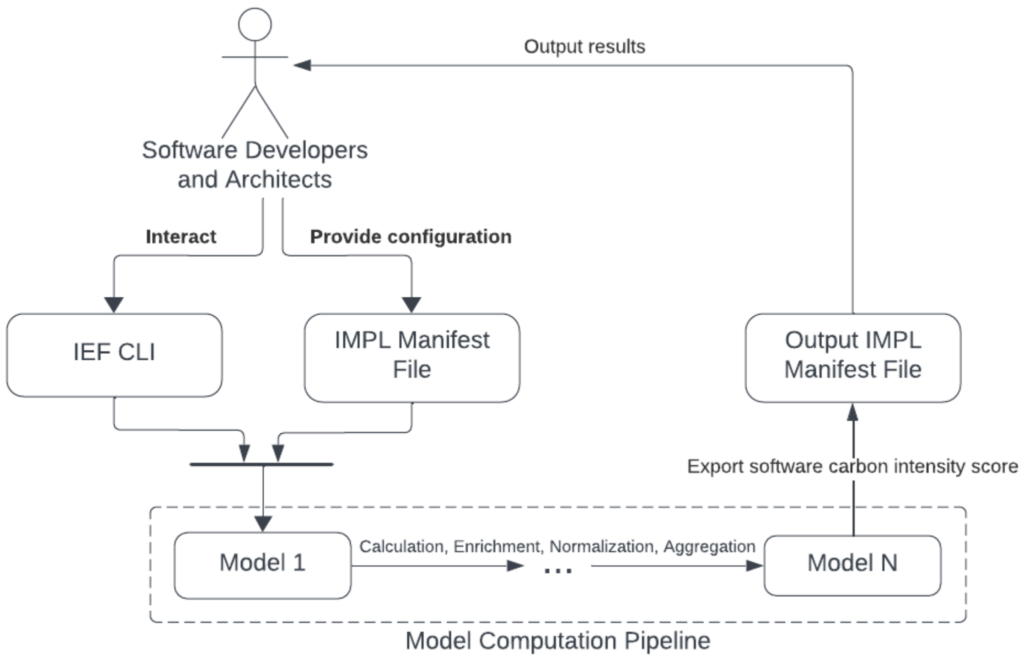
Output results
Software Developers and Architects
Interact
Provide configuration
IEF CLI
IMPL Manifest File
Output IMPL Manifest File
Export software carbon intensity score
Model 1
Calculation, Enrichment, Normalization, Aggregation
...
Model N
Model Computation Pipeline

Figure 2: IF Workflow Diagram

4.5 IF Functional view

In the functional view presented in the Figure 3, we can see that the user triggers the Impact Engine using the Impact Engine CLI by providing an `Impl` file as input. The Impact Engine uses the models specified in the `Impl` file which must have been fetched either from the IF Standard library or the IF Unofficial library. All the models must implement the Model Plugin Interface so that they are able to interact with the Impact Engine. After the calculations in the pipeline have been performed the `Ompl` file with the results is generated.

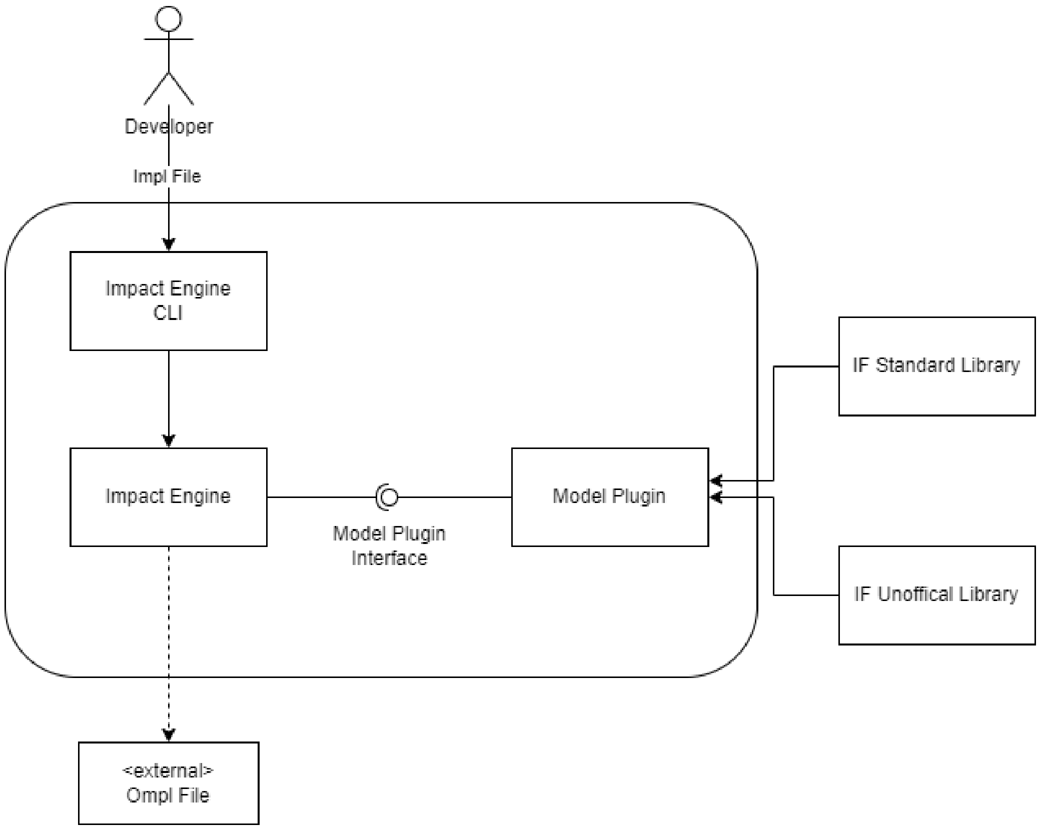

Figure 3: IF Functional view

4.6 IF Deployment view

The key steps that are described in the Deployment View in the Figure 4, are the following:

1. The user downloads on their local machine the Impact Engine from `https://github.com/Green-Software-Foundation/if.git` using `npm` or `yarn`.
2. They create a `Impl` YAML file where they are specifying all the models that they want to use in their pipeline and the system architecture with all necessary input parameters.
3. They download all the models they are specifying in the `Impl` file to their local machine using `npm` or `yarn`, since the models are not part of the Impact Engine install. If they want to download a standard model they can find them in `https://github.com/Green-Software-Foundation/if-models.git` and the unofficial models in `https://github.com/Green-Software-Foundation/if-unofficial-models.git`

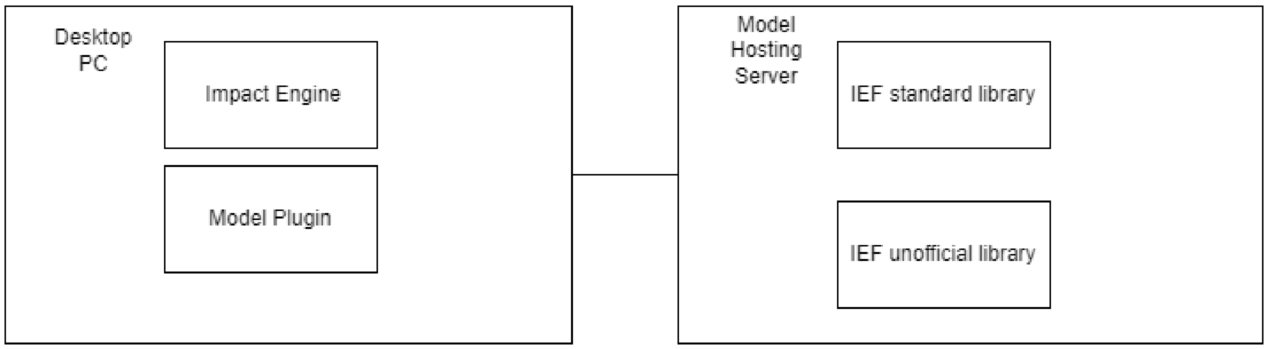

Figure 4: IF Deployment view

5 Using the IF

In this section, we explain in more detail, what is the process of using the IF, what are the Inputs needed, and the outputs produced, explaining thoroughly the concepts of *Models*, *Model Plugin Interface*, *Graphs*, *Pipeline*, *Impl* and *Ompl* files.

- The documentation for the IF can be found at `https://github.com/Green-Software-Foundation/if-docs/tree/master/docs`
- The IF tool can be found in: `https://github.com/Green-Software-Foundation/if`

5.1 Models

There are several models used by the Impact Engine. These can be official models, which are part of the standard library of models, or unofficial models, which are part of the unofficial library of models. We will provide a summary of all the existing models at this point in time and explain a few of them in more detail while also providing example `Impl` and `Ompl` files in the Appendix B.

5.1.1 Official Models:

- **Cloud instance metadata:** Looks up detailed metadata about a given cloud instance type, including the physical processor being used. The user provides as input the cloud platform provider and the name of the specific instance being used and as an output the physical processor used in the given instance, the number of vCPUs allocated to this instance and the total number of vCPUs available to this instance are returned.
- **E-MEM:** Calculates the energy expended due to memory usage, by multiplying the energy used in GB by a coefficient. By using as input the percentage of the total available memory being used in the input period and the total amount of memory available, it calculates energy used by memory. Optionally the user can also provide a coefficient for energy in kWh per GB. If not provided, it defaults to 0.38.

- **SCI-E:** Calculates the sum of all energy components. The energy components are :
  1. The CPU
  2. The memory
  3. The GPU
  4. The network traffic
- **SCI-M:** Calculates the embodied carbon for a component. We have to keep in mind that software systems cause emissions through the hardware that they operate on, both through the energy that the physical hardware consumes and the emissions associated with manufacturing the hardware. Embodied carbon refers to the carbon emitted during the manufacture and eventual disposal of a component. It is added to the operational carbon (carbon emitted when a component is used) to give an overall SCI score. The duration and timestamp of a usage is transformed by this model to the carbon emitted in manufacturing and disposing of the component that this usage is responsible for.
- **SCI-O:** Calculates the operational carbon from the total energy and grid carbon intensity. Operational carbon refers to the carbon generated by a component while it is in use. It is the product of the energy used by the component in kWh and the grid intensity in gCO2e/kWh
- **SCI:** Calculates the software carbon intensity (sci). This is the final value the framework ultimately aims to return for some component or application. It represents the amount of carbon emitted per functional unit. The functional unit in which to express the carbon impact is configured by the model and using the operational carbon and embodied carbon calculated by previous models to calculate the total carbon and the carbon expressed in terms of the given functional unit.
- **SHELL:** A model that enables external models in any language to be run in a child process. It is a wrapper enabling models implemented in any other programming language to be executed as a part of IF pipeline.
- **TDP-FINDER:** Looks up the thermal design power for a given processor in a local database. Using the name of the processor as reference, certain datasets are searched to return its thermal design power , which is the theoretical maximum amount of heat generated by a CPU that its cooling system is designed to dissipate.

5.1.2 Unofficial Models:

- **Azure importer:** Imports usage metrics from an Azure virtual machine, given user credentials and virtual machine details. By providing some basic details about an Azure virtual machine, the model automatically populates your `Impl` file with usage metrics that can then be passed along a model pipeline to calculate energy and carbon impacts.
- **Cloud Carbon Footprint:** Calculates usage metrics using the Cloud Carbon Footprint APIs. The provided cloud utilization of a specific vendor and instance type is transformed into estimated energy usage and carbon emissions.
- **WattTime:** WattTime is an external service for looking up grid emissions based on location. Based on the WattTime API, the model uses as inputs,the location of the software system, the timestamp of the recorded event and the duration of the recorded event in seconds to calculate the local electricity grid's marginal emissions rate.

- **TEADS-CPU:** Calculates the energy in kWh used by the CPU, based on the percentage of CPU utilisation during the observation, using a curve commonly known as Teads Curve.
- **TEADS-AWS:** Calculates the energy in kWh used by the CPU using a model specific to AWS instances. It works similarly to the TEADS-CPU model but is specific to the AWS platform.
- **Boavizta:** Calculates energy and embodied carbon using the Boavizta APIs. Boavizta is an environmental impact calculator that exposes an API that is used by this model to retrieve energy and embodied carbon estimates.

In the appendix B, detailed examples of 3 different models, their use cases, example `Impl` and `Ompl` files are presented. The 3 different models that are analysed there are:

- Cloud instance metadata [B.2]
- SCI-E [B.1]
- Azure importer [B.3]

5.1.3 IF Model Interface

The Impact Engine's model interface is elegantly designed to be both simple and versatile. This design choice ensures that the models within the engine are not only easy to manage but also highly extensible. The interaction between the Impact Engine and the models occurs through a streamlined model interface, following a two-phase process:

- **Configure:** In the initialisation stage, models set up their configurations based on parameters provided in the `Impl` file. This stage is crucial as it prepares the models with the necessary settings and data for execution. The engine initialises all models at this stage to ensure they are ready for the pipeline's processing. (Refer to Figure 5 for an illustration of this stage.)
- **Execute:** During the pipeline's execution stage, models receive inputs either directly from the `Impl` file or from the output of preceding models in the pipeline. Each model performs its designated computations ("black box" processing) and outputs the results, which are then passed on to the next model in the sequence. (See Figure 6 for a depiction of this phase.)

These phases are encapsulated as two abstract functions within the model interface. When a model is implemented, it overrides these functions, defining its specific behaviour during the configuration and execution stages.

5.1.4 Capability of Models

In the Impact Framework, a "model" serves as a basic computational unit within the pipeline. Each model acts as a modular block, performing specific calculations and processes. The true strength of these models lies in their API's flexibility and extensibility. By adhering to the interface standards set in TypeScript, models are not confined to any specific internal implementation. This flexibility allows for a wide range of functionalities, from standard pipeline operations to more complex tasks like plugin development.

As illustrated in Figure 6, certain models, such as the shell model, are capable of spawning external subprocesses during execution. This feature enables models to execute external commands, which can be specified in the

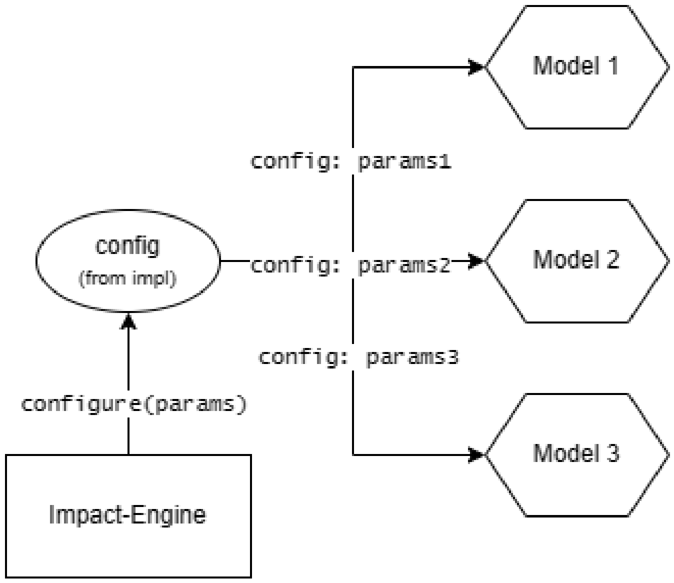

Figure 5: Configure stage of model pipeline

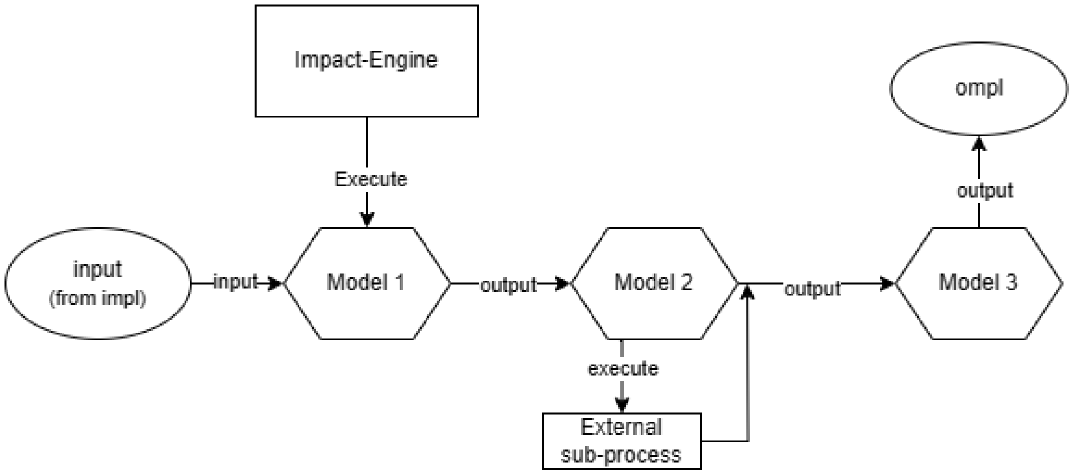

Figure 6: Execute stage of model pipeline

configuration parameters. The Impact Engine imposes no restrictions on the output forms of models, allowing them to either contribute to the pipeline's collective output or handle results in a custom manner, tailored to specific requirements.

5.2 Graphs

A graph outlines the structure of a software system, detailing its interconnected components. It is at the core of the Impact Framework and it consists of a root node explaining calculation processes and multiple nodes housing information about the software's components.

These components, like servers or networking, are part of the graph's structure. The graph enables a comprehensive understanding of the software's impact by calculating individual component impacts and aggregating them to determine the overall impact.

Utilising steps like calculation, enrichment, normalisation, and aggregation, the graph allows for a detailed analysis of the software's environmental impact. It aligns with the "4Ms" of Green Software: Mapping, Measuring, simulating, and Monitoring.

A notable feature is the graph's adaptability, allowing it to evolve from a simple representation to a more complex one over time. The idea is that for every new software system a company will create, then it will also create the corresponding graph of it. The modularity of the graphs will allow this system specific description of the graph to be later used in a larger analysis of the whole company system.

An small example graph can be seen in listing 1. It is composed of a grouping node and 2 children nodes and the measured values for each child are presented. In this example only the teads-curve model is used in the model pipeline.

```yaml
children:
   child: # an advanced grouping node
     pipeline:
       - teads-curve
     config:
       teads-curve:
         thermal-design-power: 65
     children:
       child-1:
         inputs:
           - timestamp: 2023-07-06T00:00
             duration: 10
             cpu-util: 50
             e-net: 0.000811 #kwh
             requests: 380
       child-2:
         inputs:
           - timestamp: 2023-07-06T00:00
             duration: 10
             cpu-util: 33
             e-net: 0.000811 #kwh
             requests: 380
```

Listing 1: Example Graph with 2 children nodes

5.3 Input: `Impl` files

An Impact Manifest is a file format based on YAML to represent a graph, also known as Impact YAML or `Impl` when referring to the input manifest files. Similar to a graph, an `Impl` is a calculation manifest containing everything you want to measure and how you want to measure it.

Manifest files, being YAML, are more human-readable and can be used as a formal method of writing use cases, such as Software Carbon Intensity (SCI) use cases. These files can be then computed on the command line using the Impact Engine tool to measure the environmental impact of the systems described in them. Below are some example use cases:

- **Formal Report:**
  - A Manifest file serves as a computable calculation manifest.
  - It acts as a formal report detailing not just the end impact but also includes all the assumptions, inputs, and models used in calculating the impact.
  - This formal structure allows parsing by software, comparison to other reports, adjustment, running, and verification.
  - Currently, in the Green Software Foundation (GSF), several case studies have been written to calculate an SCI score for an application, and these can all be re-written in Manifest file format.

- **Executable Impact Calculation Manifest:**
  - The command line tool `Impact Engine` can compute a Manifest file and generate impact metrics. Those metrics can either be printed on the console output or be stored in another YAML file which is called the `Ompl` file.
- **Bootstrapping Code:**
  - Manifest files can represent simple calculation manifests.
  - To handle larger, more complex systems, we may need to write graphs as code using our SDK.
  - To help bootstrap the process, humans can write the high-level structure using Manifest files and run through a tool to generate starter graph code in any language our SDK supports.

The `Impl` files have a very structured organisation following a consistent format. They could be split into 3 important parts :

- **Description Section**: Here, the name, description and any tags or variables of the system are initialised. This is the least important part and is usually just used to give some background information of the system that will be measured using this `Impl` file.
- **Pipeline Section**: Here, the model pipeline is being initialised, meaning that it describes which models will be used and in what order. Any inputs that the models might require must also be initialised.
- **Graph Section**: The architecture of the measured system is described here. Certain measurements for every architectural component or the models that will be used to import the metrics for that component must also be described.

For detailed examples of `Impl` files, the reader can refer to section 5.6 where a input YAML file is provided for the user story first introduced in section 3.1. Additionally example Impl files are provided for 3 different models in the Appendix B

5.4 Calculating using the Impact Engine

After an `Impl` file that describes the system to be measured, has been created, the measurements of the system and the model pipeline in the `Impact Engine` can be used to calculate the carbon Impact of the system. To invoke the `Impact Engine` we use the CLI with the following command:

```
impact-engine --Impl <path-to-your-Impl-file>
```

Listing 2: Impact engine command with no output file

In this case the output data will be displayed in the console screen. The impact-framework can also be configured to save the output data to another YAML file. To do this, add the –`Ompl` flag and the path to save the file to, like the following command:

```
impact-engine --Impl <path-to-your-Impl-file> --Ompl <your-savepath>
```

Listing 3: Impact engine command with output file

5.5 Output: `Ompl` files

The file that was used to save the output data to, is the `Ompl` file. This file has the exact same information as the `Impl` file, with another section appended at the end of it. This section is called the outputs section and contains all the calculations about the system that were made based on which models we had decided to use.

The reason that the `Ompl` file contains all the information that the `Impl` file does as well is because we want the `Ompl` file to be a complete and sufficient description of the whole carbon impact measurement. It contains all the information needed about the system: its description, the components it is made of, the measurements that were given as inputs, the model pipeline and at the end the output data that were calculated by the `Impact engine`. In this way it tells the complete story of a system, its architecture, its inputs and the outputs that were calculated along with the way those calculations were made (the models that were used).

In the next section examples of `Impl` and `Ompl` files will be provided to help the reader acquire a deeper understanding of the topic. If the reader deems that he holds a firm grasp on the topic and does not require further explanation then the next section: 5.6 can be skipped.

5.6 User story - Example of IF use

For the scenario presented in section 3.1 we will be adding more details to the original user story by presenting the necessary `Impl` file along with the logic behind its creation, the execution process, and the produced `Ompl` file along with the conclusions that can be derived from it.

5.6.1 IF Input

We would first create a YAML file with inputs as follows to analyse the Software Carbon Intensity (SCI) of our servers. The file is split into several listings for better visual presentation.

```yaml
name: nesting-demo
description: null
tags:
  kind: web
  complexity: moderate
  category: on-premise
initialize:
  models:
    - name: teads-curve
      kind: builtin
    - name: sci-e
      kind: builtin
      verbose: false
      path: ''
    - name: sci-m
      kind: builtin
      verbose: false
      path: ''
    - name: sci-o
      kind: builtin
      verbose: false
      path: ''
    - name: sci
      kind: builtin
      verbose: false
      path: ''
graph:
  children:
    child:
      pipeline:
        - teads-curve
        - sci-e
        - sci-m
        - sci-o
        - sci
      config:
        teads-curve:
          thermal-design-power: 65
        sci-m:
          total-embodied-emissions: 251000
          time-reserved: 3600
          expected-lifespan: 126144000
          resources-reserved: 1
          total-resources: 1
```

Listing 4: input IF YAML file (nesting-demo)

```
    children:
      child-1:
        inputs:
          - timestamp: 2023-07-06T00:00
            duration: 10
            cpu-util: 50
            e-net: 0.000811
            requests: 380
        outputs:
          - timestamp: 2023-07-06T00:00
            duration: 10
            cpu-util: 50
            e-net: 0.000811
            requests: 380
            thermal-design-power: 65
            total-embodied-emissions: 251000
            time-reserved: 3600
            expected-lifespan: 126144000
            resources-reserved: 1
            total-resources: 1
            grid-carbon-intensity: 457
            functional-unit-duration: 1
            functional-duration-time: ''
            functional-unit: requests
            energy-cpu: 0.00013541666666666666
            energy: 0.00013541666666666666
            embodied-carbon: 7.16324200913242
            operational-carbon: 0.061885416666666665
            sci: 0.001901349322578707
```

Listing 5: child-1 of Input IF YAML file (nesting-demo)

```
    child-2:
      inputs:
        - timestamp: 2023-07-06T00:00
          duration: 10
          cpu-util: 33
          e-net: 0.000811
          requests: 380
      outputs:
        - timestamp: 2023-07-06T00:00
          duration: 10
          cpu-util: 33
          e-net: 0.000811
          requests: 380
          thermal-design-power: 65
          total-embodied-emissions: 251000
          time-reserved: 3600
          expected-lifespan: 126144000
          resources-reserved: 1
          total-resources: 1
          grid-carbon-intensity: 457
          functional-unit-duration: 1
          functional-duration-time: ''
          functional-unit: requests
          energy-cpu: 0.00011243956355860435
          energy: 0.00011243956355860435
          embodied-carbon: 7.16324200913242
          operational-carbon: 0.05138488054628219
          sci: 0.0018985860235996583
```

Listing 6: child-2 of Input IF YAML file (nesting-demo)

Excluding the straightforward parameters such as name, the overall structure of the YAML input file can be broken down into tags, models, and graphs (Listing 3, 4, and 5 are parts of one single YAML file):

- **Tags** contain the basic description of the software running on the servers. Web means that it is a web based application, and has moderate complexity. Additionally it is on-premise which means that the software runs locally instead of on the cloud.
- **Models** These five models are the ones that are most suited for our servers to calculate the environmental impacts: Teads-curve: A model that returns an embodied value given the sci embodied attribution equation Sci-e, Sci-m,Sci-o, and Sci, are all model that are used in calculating SCI. These details of these model are further elaborated in section 5.1.
- **Graph** as briefly described back in Section 5.2, it lists the structure of the system evaluated and measured. The Children variable represents all the nodes, and each individual node is a child. As shown in the code,

the structure actually contains more than one child, it has 2 children in total, representing 2 sub-nodes, each responsible for different functionality of the overall system.
In the graph Section more subsections are initialised:

1. **Pipeline**: here the models that will be part of the pipeline and their order are initialised.
2. **Config**: refers to the configuration of each model inside the pipeline, which contains the parameters of the models that we input. These configurations are directly used to calculate the results of the output.
3. **Children**: refers to the group of node components that make up the graph. As shown in listing 5 and listing 6, there are 2 children, child-1 and child-2. Each child has an input observation that will affect the environmental impacts. The observation is a dictionary that contains model parameters such as timestamp, and duration as shown in listings 5, 6. Input observations would generate an output observation that could be used to analyse its environmental impacts. These observations are the only information needed for the children architecture since the parameters given are enough to measure the energy used and the carbon footprint.

5.6.2 IF Output

After running the input YAML file on the Impact Engine, it would output another YAML file, which contains the calculations the IF made based on the input parameters. We want this operation process to be simple and user-friendly even to users without prior technical background. The example output file will look as follow:

```yaml
      children:
        child-1:
          inputs:
            - timestamp: 2023-07-06T00:00
              duration: 10
              cpu-util: 10
              e-net: 0.000811
              requests: 380
            - timestamp: 2023-07-06T00:10
              duration: 10
              cpu-util: 10
              e-net: 0.000811
              requests: 380
            - timestamp: 2023-07-06T00:20
              duration: 10
              cpu-util: 10
              e-net: 0.000811
              requests: 380
            - timestamp: 2023-07-06T00:30
              duration: 10
              cpu-util: 10
              e-net: 0.000811
              requests: 380
          aggregated-outputs:
            aggregated-carbon: 2.87585858143074 54
            aggregated-energy: 0.00023111111111111111
```

Listing 7: Output IF YAML file for child-1

```
        outputs:
          - timestamp: 2023-07-06T00:00
            duration: 10
            cpu-util: 10
            e-net: 0.000811
            requests: 380
            thermal-design-power: 65
            total-embodied-emissions: 251000
            time-reserved: 3600
            expected-lifespan: 126144000
            resources-reserved: 1
            total-resources: 1
            grid-carbon-intensity: 457
            functional-unit-duration: 1
            functional-duration-time: ''
            functional-unit: requests
            energy-cpu: 0.0000577777777777777776
            energy: 0.0000577777777777777776
            embodied-carbon: 7.16324200913242
            operational-carbon: 0.0264044444444444442
            carbon: 0.7189646453576863
            sci: 0.0018920122246254903
          - timestamp: 2023-07-06T00:10
            duration: 10
            cpu-util: 10
```

Listing 8: Output IF YAML file for child-2

For each child node, there would be two types of outputs. The ‘aggregated output’, and ‘outputs’.

- **Aggregated output** refers to the overall sum of the carbon emission and the energy consumption for all children
- **Outputs** are more in-depth breakdown of the energy usage and the carbon intensity in each time period of each child node. Energy-cpu, energy, embodied-carbon, operational carbon, carbon, sci are all metrics of the environmental impacts of the servers.

With the data provided in the output YAML file, we could extract an accurate and detailed breakdown of the overall carbon emission of our servers and we come to the following conclusions. As a multinational computer technology company, we wanted to evaluate the carbon emissions of the software system of our servers that are on Azure. By running the IF, the output YAML file shows that the total aggregated carbon emission is 2.876 gCO2eq after rounding up, and the total aggregated energy consumption is 0.000231 kWh respectively. There are also available the individual contribution of each child node with a more thorough breakdown behind the emission and energy consumption of each child. For example, e-net which is measured in kWh, CPU util, total embodied emissions, grid carbon intensity, energy CPU, and embodied and operational carbon are all directly related to the calculation of the carbon emissions. With these metrics available, the engineers could analyse them and adjust the input parameters such as CPU util and e-net, accordingly to lower the emissions and ultimately lower the impact on the environment.

5.7 Domain Story Figure

The user story that was just presented earlier in section 5.6 is also visualised in figure 7 as a simple domain story. As we can see in the figure an enterprise company requests one of their developers to perform a environmental impact assessment of their Azure instances. The developer in turn, leverages the functionality of the IF. Firstly they create the Manifest Input file (`Impl` file) where they describe the architecture of their system, measurement from the usage of those system and the models they want to use in their pipeline. Then, using the IF CLI they invoke the Impact Engine and get the environmental Impact results (maybe in a YAML format as well, called the output `Ompl` file).

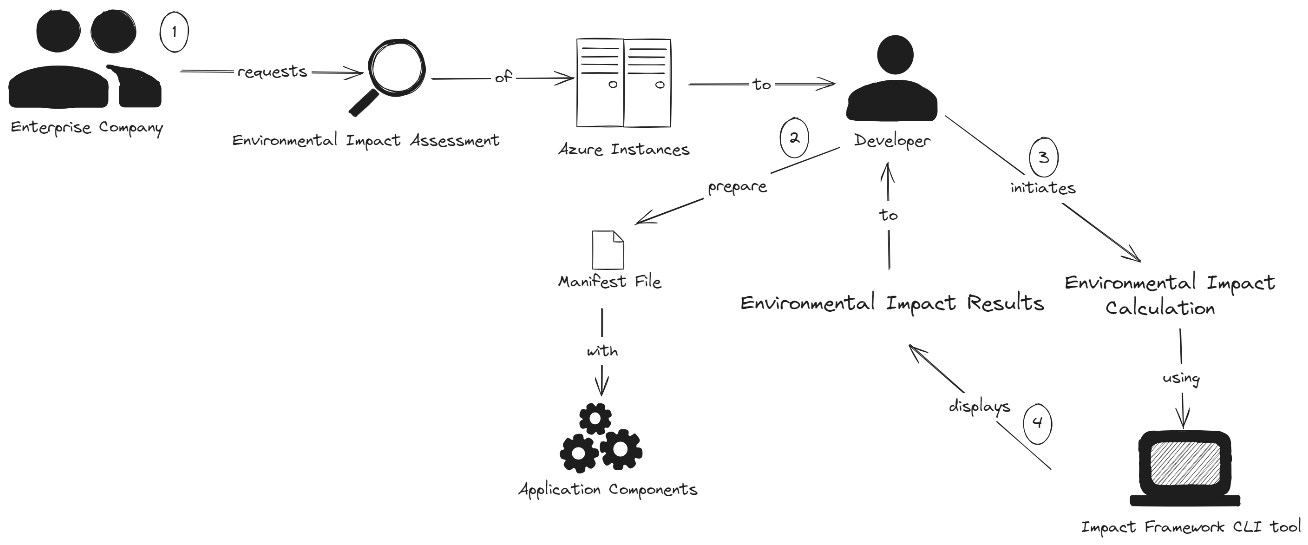

Figure 7: Domain Story showing the main scenario of assessing environmental impact of software running on Azure instances

6 Rejected Projects

A discussion took place between the members of the UCL team and the Green Software Foundation, concerning potential ideas for projects that would provide value to the IF. A number of ideas that seemed hopeful, but were ultimately rejected are summarised in this section.

6.1 WebUI - Simulation Tool

The original idea was to create a web UI for the IF that would also serve as a simulation tool. By providing an accessible and user-friendly interface for the Impact Framework, which currently operates through a command line, the team aimed to help users model, measure, simulate, and monitor the environmental effects of their software without the need for high-level programming knowledge.

The IF tries to tackle a very difficult and complex issue: the environmental impacts of modern software, which often runs in diverse environments and involves numerous components. In turn, the web UI would be able to simplify the use of the IF even more by adding another tool, an abstraction level above it so that users don't have to interact with the IF at all. By simplifying the input process, and enhancing the output visualisation, it would be possible for users to try different inputs and scenarios to find the one with the lowest environmental impact.

Unfortunately, this option is not viable due to the maturity level of the project. The IF has just come into alpha and everything a UI will rely on (types and number of inputs) will change in the following 6-12 months, but still remains a great project idea once the IF projects is closer to its final form.

6.2 AWS model

There is currently an unofficial model for the IF called Azure-importer that allows you to provide some basic details about an Azure virtual machine and automatically populate your `Impl` input file with usage metrics that can then be passed along a model pipeline to calculate energy and carbon impacts. For more information refer to section B.3

A proposed project idea was to create a similar model for the AWS cloud platform to measure the environmental impact of VMs or other cloud services provided by them. Such a model would help expand the IF user base with AWS users which would be extremely significant since AWS is the current market leader for cloud infrastructure with a 33% market share. After discussion with the IF development team though it was decided that this was not a viable project since AWS does not expose similar APIs to Azure to get the required metrics. Thus, it would not be possible to provide a working model within the available time frame.

6.3 Right Sizing Simulation tool

Most users pick servers that are far too powerful for their needs, so they run at low utilisation. Right-sizing is the task of picking a more appropriate server. A model which helps figure out the right sized server to use given the workload utilisation would be thus very helpful.

For the Impact Framework (IF), right-sizing is a crucial feature. It enables users to accurately estimate the computational resources they need, thereby preventing energy wastage due to excess capacity. For instance, consider a basic web service running on a single thread of an 8-thread CPU. The remaining seven threads remain idle and are unneeded but they still contribute to unnecessary power consumption. Ideally, the right-sizing module of the IF would identify such scenarios and recommend more appropriate hardware configurations that meet the minimum requirements of the task.

Given its potential to significantly reduce energy consumption and improve efficiency, right-sizing is an essential functionality for IF. However, the implementation of this feature presents substantial challenges. Accurately determining the computational resources needed by software is complex, as it often depends on various unpredictable factors, including the specific usage patterns of users. Additionally, handling diverse scenarios such as virtualisation and containerisation adds to the complexity. The development of a right-sizing module would also require extensive datasets detailing resource consumption for different software, information on various CPU models, and strategies for hardware resource allocation in cloud environments. Currently, as the IF is still in its alpha release stage, access to such comprehensive datasets is limited, making the implementation of this feature impractical within our current time frame and resource constraints.

While the right-sizing feature remains unfeasible at this stage, its potential benefits make it a worthy candidate for future development as more resources and data become available. Actually we will try to add some right-sizing suggestion capabilities, (as long as the development time is sufficient) to the model we finally decided to implement.

Instead of actual right-sizing we will try to provide suggestions from a number of available instance types and sizes, which is a task we believe can be implemented within the available time frame.

7 Project Vision and Requirements

7.1 Final Project Description

The final project, agreed upon by the members of the UCL team as well as the Impact Framework (IF) project team, is the implementation of a Parameter Optimisation Model. Most of the existing models focus primarily on measuring carbon emissions and other environmental metrics, but do not provide any suggestions for improvement. Therefore, our work aims to bridge this gap in the IF ecosystem and go a step further by helping users with optimising the environmental impact of their software. For example, the model can suggest a different time and location to run the server, or a more optimal instance type.

we have decided to allow suggestions, based every time on the output metric that the user wants to optimise. To be more precise, the user will provide what metric or metrics they want to minimise, for example the carbon footprint sci or the total energy. Additionally they will choose from the 3 available parameters (location, time, and instance type), for which they want suggestions on, by providing a list or range with available values. For example they could provide 4 different locations. Then our work will be to efficiently search through the search-space of these input parameters to find the optimal combination that minimises the desired output metric. We have decided to implement this as an IF model. We will later be providing a detailed example of how this model will work in section 7.4, extending the examples presented in sections 3.1 and 5.6.

The documentation and the implementation of this model can be found in: `https://github.com/TomasKopunec/comp0101-ief`

This model would be developed such that it adheres to the IF's standards by extending and implementing a common class interface. And by following this design choice, which was strongly advised by the IF team, the final project aims to align with the IF's core design principles. Thus, allowing the model to be used interchangeably within the IF alongside other models. Moreover, the model design will be generic due to the possibility of many future models being added to the ecosystem. This helps ensure that our work remains compatible with these future models that have not been developed yet.

This model being able to easily integrate into the existing systems is one the most important aspects of this project for the Green Software Foundation. As it can be inferred from their decision to develop the IF using a plugin architecture where anyone can create a model that can be easily used into the existing pipelines, we can understand that modularity, extensibility and integrability are the key principles of the Impact Engine Framework project. That is why this functionality has to be implemented through a model and not through an external tool. Firstly, to scale the impact framework horizontally, it is best to do so in line with the IF's model-based approach. And following this approach would make our project compatible with an arbitrary number of models that adhere to the IF's standards. Also, building an external tool only makes sense if the goal is to add an extra layer of abstraction to the IF project. Otherwise, there is a high likelihood that the external tool could become redundant or irrelevant. For example, the IF team could decide to build an optimisation model which would be more versatile, or the IF project's core functionality could change such that our external tool becomes incompatible. After all we are still in the Alpha phase of the IF so creating a model that is easily integrated and easy to extend, is extremely valuable to the GSF since it will allow the IF to grow around it with new models and parameters.

Additionally, by using established search-based techniques, the model aims to explore how different parameters may affect carbon emissions. This exploration can help identify parameter combinations that are within acceptable limits. The goal of this model is not to change or recalculate anything. But rather to behave as an advisor by analysing the provided data, and returning useful suggestions. Therefore, this model can be leveraged by consultants to offer better solutions to customers on reducing emissions, leading to potential cost savings. We believe that this provided extra functionality will be of great value to the current stakeholders of the IF while also helping expand the user base.

In conclusion, the main parameters the project aims to be optimised by this model are location, time, and instance type, which relates to right-sizing. However, further investigation and testing may uncover additional parameters that could be optimised or reveal the limitations of current ones. Also, each recommendation provided by the model will be accompanied by a detailed analysis of its impact. This can help users understand the overall benefits of each recommendation, and make informed decisions based on this data. Last but not least, some visualisations of combinations of input parameters in relation to environmental impact metrics will be provided, in order to offer a better insight to the trade-offs between the different parameters.

7.2 Project Goals

In this section the goals for the parameter optimisation model will be summarised.

1. **Optimisation** Giving users feedback on how to improve their software should be one of the main purposes of this framework. Users should not only get information about the environmental impact of their software but also have an idea of how to make it better. The optimisation features focus on the following goals:
   - **Optimisation suggestions**: IF should provide users with optimisation suggestions/solutions for their input configuration. Alternative options/parameters in their configurations that result in less carbon footprint will be given.
   - **Visualise trade-offs**: While providing optimised suggestions, IF could also demonstrate trade-offs between different inputs and environmental impact metrics to users with informative figures.
2. **Comparison** Comparison between different options is a frequently demanded functionality, especially for software developers/companies. In reality, environmental issues are not the only thing considered by stakeholders, but also cost and performance. Giving users a comprehensive understanding of the differences and trade-offs between their input options is really important. The IF should enable users to compare their input configurations horizontally, and explicitly highlight significant differences between different configurations in regards to environmental impact metrics.
3. **Automation**
   - **Minimise unnecessary manual operations**: Automate the simulation procedure as much as possible, and minimise unnecessary manual operations. Only crucial operations should require decisions from users.
   - **Promote IF to a wider audience**: The entire IF and optimisation process should be as easy as possible through automation in order to be accessible to as many users as possible.
4. **Versatility**
   - **Multiple input parameters**: There should be inputs for multiple parameters such as location and time for users to decide on. Since there does not exist a universal solution for all environments, this would enable users to obtain better optimisation results through combinations of parameters.
   - **Flexibility and freedom of choices**: Allow users to choose which parameters they would want to prioritise to optimise and based on which metric. For example, certain users might only wish to optimise the location parameter and not the time.

5. **User-friendly**
    - **Easy to use**: The input and output process of the IF parameter optimisation should be easy for users of any technological background to operate.
    - **Easy to understand**: The output suggestions of the IF optimisation simulations should be highly readable and simple to understand. For example, the output could also include metadata to provide a more holistic understanding for the user. Additionally, metadata does not require any programming knowledge to understand. By visualising output results it is also even easier for users to draw conclusions.
6. **Integrability**
    - **Easy to integrate with other models**: The optimisation model should correctly recognise, and coordinate with other existing models (official or non-official) to get the desired output. The optimisation model should also be able to be easily modified to optimise the parameters of configurations of other existing models.
    - **Easy to be adapted by new models**: In the future many more models with new input parameters will be developed. It is important for our model to be able to adapt so that it can optimise these new parameters.

7.3 Stakeholders

In this section, stakeholders that have an interest in the parameter optimisation model that the UCL team will be developing, are summarised along with their stake in the project.

1. **End users**:
    - Are concerns with the accuracy of the outputs that the model provides.
    - They hope that the model will help reduce energy consumption and carbon emissions to save costs and improve operational efficiency.
2. **IF Project Team**:
    - The model will enhance capability of the framework in the aspect of providing more optimisation options.
    - Hope that the model can be integrated well with the current model pipelines and withing the current ecosystem of the impact framework.
3. **UCL Team**:
    - Concerns about the scalability and maintainability of this model.
    - They want to gain experience and knowledge in the field of energy efficiency and sustainable development.

7.4 User story for Parameter Optimization Model

> This section extends the examples provided in sections 3.1 and 5.6 by adding the new parameter optimization model into the pipeline.

In addition to the existing features of the IF, the company wants to also receive some suggestions on how to reduce the environmental footprint of their servers. They will use the parameter optimisation model to receive feedback on how they can change their input parameters to better reduce the impact of their software on the environment.

7.4.1 IF with Parameter Optimisation Input

The **pipeline** must be extended with the parameter optimiser and its inputs. To avoid presenting again the long `Impl` file we will just be showing the changed section. The rest of the file remains the same.

```
     - name: parameter-optimizer
      kind: builtin
      verbose: false
      path: ''
graph:
  children:
    child:
 pipeline:
         - teads-curve
         - sci-e
         - sci-m
         - sci-o
         - sci
         - parameter-optimizer
      config:
        parameter-optimizer:
          - allowed_locations: ['UK', 'Europe-North', 'Europe-East']
          - allowed-timeframes: [('2023-07-06T10:00', 2023-07-06T12:00), ('2023-07-08T00:00', 2023-07-08T2
          - allowed-downsize-percentage: 10
          - improvement_metric: 'operational-carbon'
```

Listing 9: `Impl` file for using parameter-optimiser as pipeline to conduct parameter optimisation

We insert four simple **parameters**, location, time frame, downsize limits, and the improvement metric to generate suggestions that would allow us to reduce the environmental impact of our servers. Another options instead of downsize limit that would be more easy to implement, could be a list of available architectures to choose from.

7.4.2 IF with Parameter Optimisation Output

The output YAML file is as follows. The output section **suggestions** consists of five parameters, the first two are suggested locations and time which correspond to the locations and timeframes from our input file. In addition, it also suggests a new system architecture for the software running on our servers. This new architecture will be chosen from a list of available ones since exact right-sizing capabilities will not be available by this model. In total, there are three output variables with suggestion for the 3 parameters that were improved. The other two variables, are about the amount of improvement we would achieve, if we follow the suggestions, and the resulting value for the metric that was decreased, in this case the carbon footprint.

```
outputs:
  suggested:
    - suggested_location: 'UK'
    - suggested_time: 2023-07-06T11:00
    - suggested_architecture: graph
    - imporvement_percentage: 21
    - operational-carbon: 1.57g gCO2e
```

Listing 10: Output IF file for providing suggestions to the user

In addition, certain visualisations of input parameters in relation to environmental impact metrics will be provided, offering better insights to the trade-offs between different solutions. These could be diagrams showing how a single input parameter or a combination of them affect the desired output or a pareto front diagram, if we are trying to optimise more than one output environmental metrics. For example if we are trying to optimise both the energy and carbon footprint then certain locations might decrease the total energy but increase the carbon footprint, since the energy used there originates from more ”unclean” sources.

Overall, we want this new parameter optimisation feature of IF to provide the company with a user-friendly way to help them to reduce the carbon footprint of their servers, that integrates easily into their current use of the IF.

8 Project Architecture and development plan

8.1 Model Structure

In this section we will describe what are the different steps that will take place during the execution of this model. The actual implementation will be done using python and leveraging the existing model plugin interface and the shell model along with certain GSF APIs that provide feedback on environmental impact based on location and time.
The exact specifications of the model have not yet been decided. The main concept is:

- Get as input from the user a list of available locations that the software is able to be executed at
- Get as input from the user a list of available time frames for the software to be run
- Get as input from the user the metric (or metrics) they wants to improve (for example energy or carbon footprint)
- Get as input from the user The percentage of down-sizing they are willing to accept(Another possible option would be a list of available architectures)
- Use a genetic heuristic to perform an efficient search of the search-space (or perform an exhaustive search if the search space is small enough)
- Find the best solutions (as far as minimising the desired metric) and store them

- The best input parameter modifications are presented in the`Ompl` file, along with the percentage of improvement and the new value of the improved metrics
- Present a number of visualisations of the inputs in relation to the output metric as well as a pareto front diagram if more than one metrics are being minimised, so that the trade-offs are better understood.

8.2 Heuristics used

In our approach to optimising user-defined metrics within specified locations and time frames, we incorporate a flexible strategy that adapts to the complexity of the problem at hand. For scenarios with a relatively small number of combinations, we might employ an exhaustive search approach, which will ensure a comprehensive coverage of all possible options, guaranteeing the identification of the optimal solution withing a manageable search space.

As the complexity of the problem increases, especially when dealing with a greater number of combinations or when handling continuous stream of input values, we employ advanced heuristic algorithms. Those algorithms are particularly effective in navigating vast and complex search spaces, striking a balance between thorough exploration and efficient exploitation. Our primary heuristic tool is a genetic algorithm, which simulates the process of natural selection, where the fittest solutions are selected for reproduction to produce offspring in the next generation. The evolving solution gets constantly improved through a feedback loop that refines parameters and strategies based on the past results.

For problems that require optimisation of multiple metrics, we employ multi-objective optimisation techniques. These techniques are essential for identifying a set of Pareto-optimal solutions, offering users a spectrum of optimal choices that align with their specific criteria. Additionally, by incorporating user-defined constraints such as locations, time frames, and downsizing percentages, our approach ensures that the search is confined to feasible and relevant areas of the solution space.

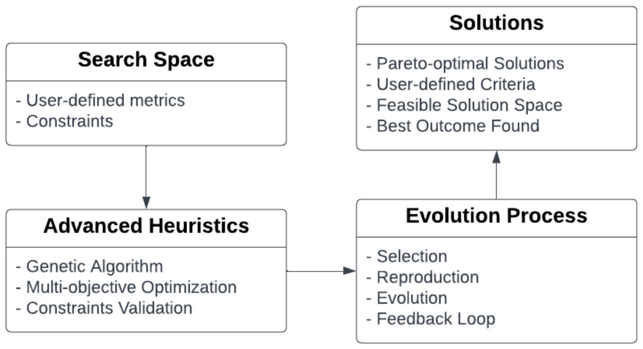

Figure 8: Optimisation Diagram

8.3 Development plan

In the second term we will have 10 weeks to finish this project. We will be prioritising the complete development and testing of the parameter optimisation model for location and time frame and hopefully there will be available time to also finish the right-sizing part of the model.

The team will try to emulate an agile type of development with Kanban, splitting into several smaller groups which will simultaneously be working on different tasks. For example, some of the teams will be working on background search for the next task or on testing, while other teams focus on the development and the remaining on testing.

We will be presenting a first draft of the development plan for the next term in the following figure 9. Tasks will be undertaken by teams of 2-3 people. We will try to make a small alpha implementation of our model that is functional for the small example presented in section 7.4, which gives suggestions for location and time, by week 5.

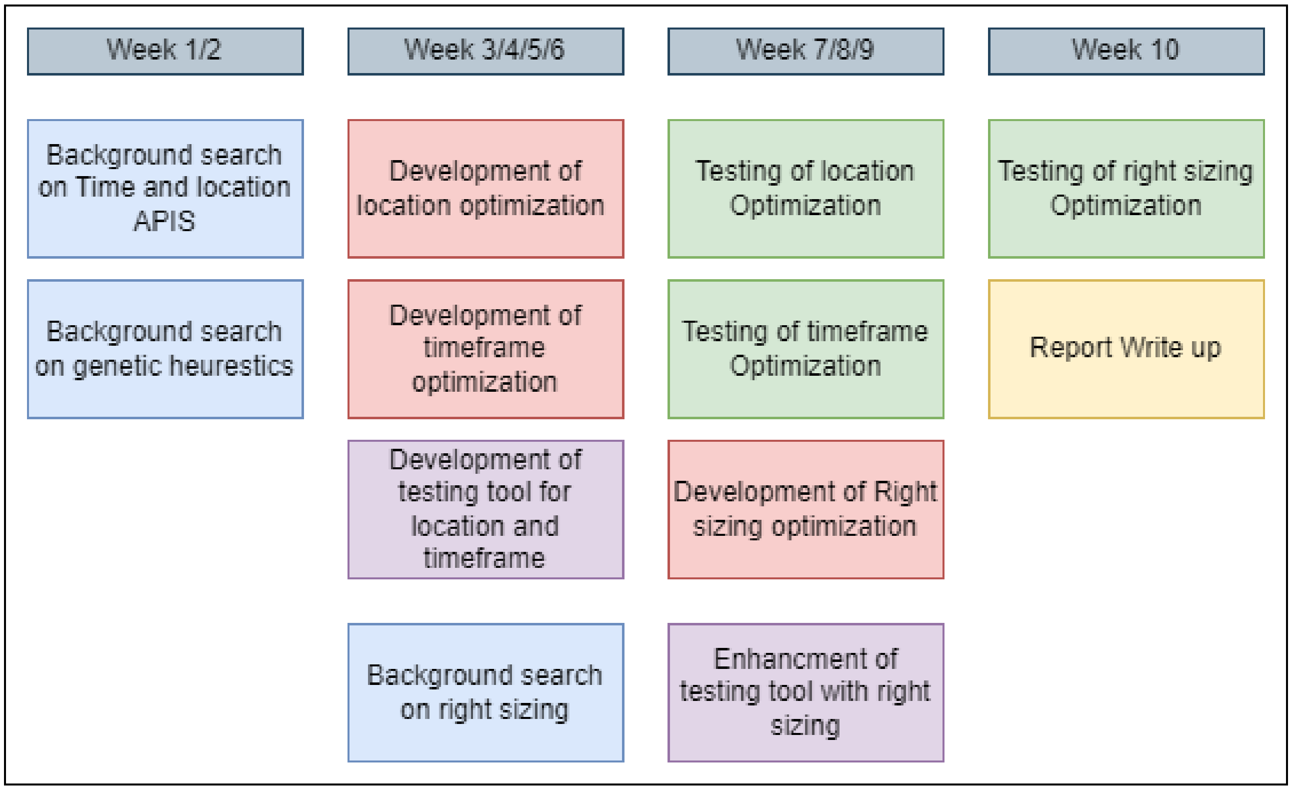
Week 1/2
Week 3/4/5/6
Week 7/8/9
Week 10
Background search on Time and location APIS
Development of location optimization
Testing of location Optimization
Testing of right sizing Optimization
Background search on genetic heurestics
Development of timeframe optimization
Testing of timeframe Optimization
Report Write up
Development of testing tool for location and timeframe
Development of Right sizing optimization
Background search on right sizing
Enhancment of testing tool with right sizing

Figure 9: Development plan for term2

9 Evaluation Plan

9.1 Evaluation Criteria

1. **Correctness**
    - **Definition:** The ability of the model to perform an accurate search of the search-space and provide valid results.
    - **Evaluate method:** Comparing the results of the model with the results of a simple python tool that will perform exhaustive searches.

2. **Optimisation**
   - **Definition:** Reduction ratio of software carbon emissions before and after optimisation.
   - **Evaluate method:** Comparing the carbon emissions of the software system with the original parameters with those after the optimisation our model suggests, to see if it does indeed provide added value.
3. **Performance**
   - **Definition:** How fast the optimisation model will generate the corresponding output and how large an input the model can handle.
   - **Evaluate method:** Calculate the time overhead that is caused by including the parameter optimisation model into a pipeline.
4. **Usability**
   - **Definition:** Evaluating the user-friendliness of the Parameter Optimisation Model.
   - **Evaluate method:** We can evaluate it by using the System Usability Scale (SUS) questionnaire.
5. **Flexibility**
   - **Definition:** How well the model can adapt to different software designs and new models.
   - **Evaluate method:** Find more user cases and contact the GSF team about new models being developed and analyse how easy it would be to alter the parameter optimisation model to optimise the new parameters.

9.2 Testing and Evaluation Plan

In parallel to the parameter Optimisation model a simple python tool will also be created which will facilitate the testing of our application. The concept of this tool will be to invoke the IF for all the possible combinations of the optimisation parameters, thus performing an exhaustive search through the search space. All the results will be stored and the best one will be selected. This optimal input will be then compared to the result from the parameter optimisation model.

It is important to note that for a small number of combinations, both the model and the Python tool will function similarly since an exhaustive search will be performed in the model as well. The key distinction is that the Python tool operates externally while the model is an integral part of the pipeline. This tool will also be different from the model implementation not only because it performs an exhaustive search but also because it does not leverage existing APIs the GSF has implemented to measure energy and carbon footprint based on location and time. We will be providing more information about these implementation details and how the differ from a constant invoking of the Impact Engine in the manual of the model, presented once the implementation is complete in the next term.

Undoubtedly, this exhaustive search and constant invoking of the `Impact engine` will be extremely slow but this is not an issue for this case since the testing tool will only be run on a selected suite of test cases only once. If the workload is too big to perform an exhaustive search, even once, an alternative is to perform a selective search of the entire search space through sampling. Instead of testing every possible combination, we can randomly sample a subset of combinations for evaluation. We will test our new tool to a number of `Impl` files starting from very simple architectures with a small number of nodes (such as the one in section 7.4) and incrementally test on more complex architectures. We can also compare the time this tool needs to find a result to the time the parameter optimisation model will need, to understand how efficient the genetic heuristic we are using is as far as search space exploration is concerned.

Lastly a number of different heuristics might be used and compared with each other, in order to see which performs a more efficient exploration of the search space.

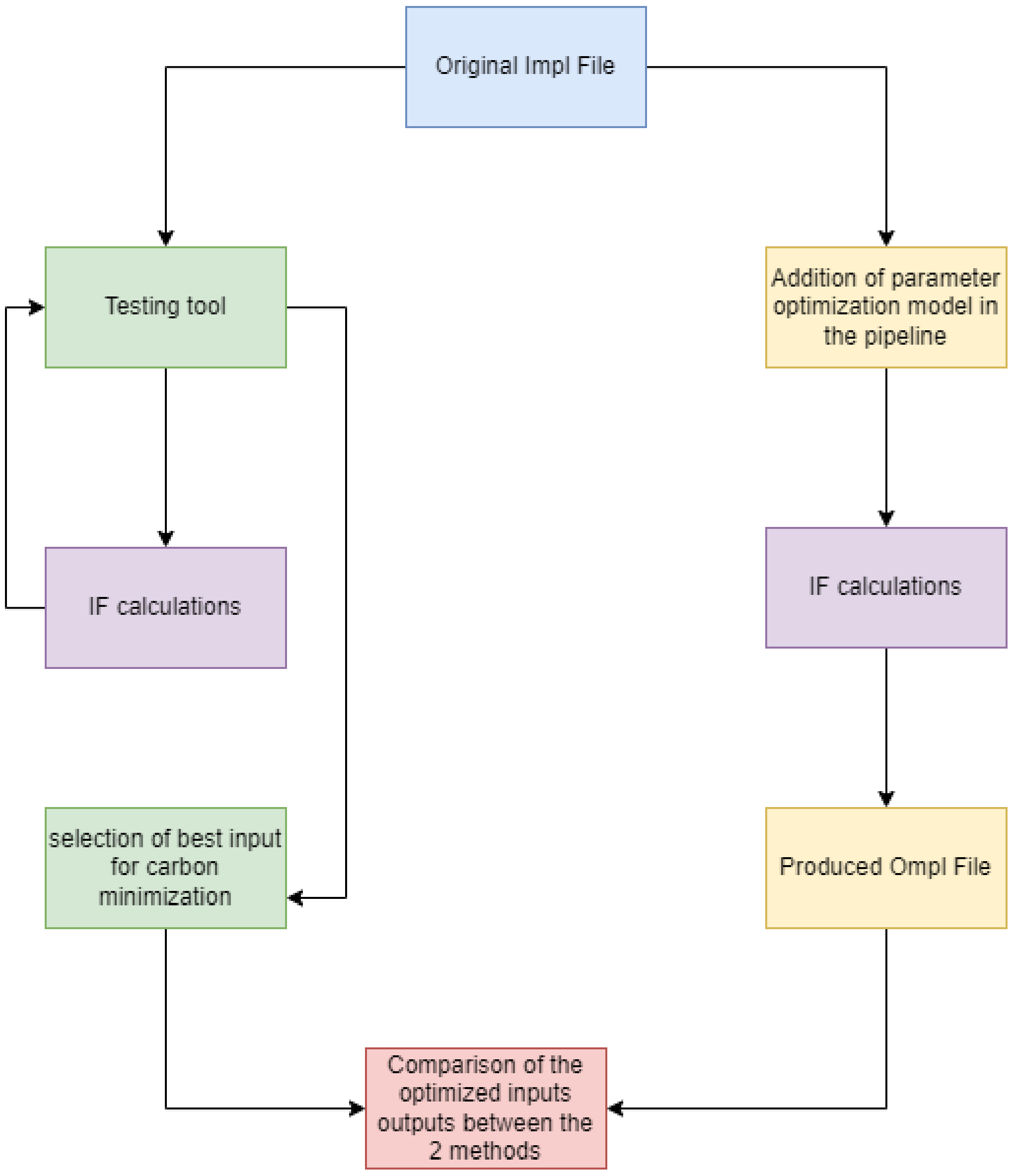
Original Impl File
Testing tool
Addition of parameter optimization model in the pipeline
IF calculations
IF calculations
selection of best input for carbon minimization
Produced Ompl File
Comparison of the optimized inputs outputs between the 2 methods

Figure 10: Testing methodology

Appendices

A Secondary Stakeholders of the IF

Secondary Stakeholder	**Concerns, Wishes, and Expectations**
NGOs	• Hope for a meaningful impact on the environment and raising public awareness.
Environmental Engineers/Researchers	• Their works or studies will be affected by the data based on the framework's results.
National and Local Government Politicians	• Data and insights from the framework will affect policy and regulatory standards related to environmental sustainability.
UCL Student Team	• Develop, implement, test, maintain, and evolve user-friendly solutions for simulating, measuring, and monitoring software's impact on the environment.
Energy Supplier	• Expects insights into the future demand for energy based on the framework's results.
Competing Frameworks or Tools	• Are concerned about the impact on users' retention. Doesn't want IF to be the first choice for users needing an environmental impact assessment framework.

Secondary Stakeholder	**Concerns, Wishes, and Expectations**
General Public	• The IF will help raise awareness on green technology solutions, driving innovation and development of other environmental technologies.

Table 1: Secondary Stakeholders' Concerns and Expectations

B Detailed Examples of Models

B.1 Sci-e

It is a model that simply sums up the contributions to a component's energy use. Those contributions can be by the `CPU`, `GPU`, `Memory` or the `Network`. The model returns the total energy, which is then used as the input to the `sci-o` model that calculates operational emissions for the component.

Inputs:

- `energy-cpu:` energy used by the CPU, in kWh
- `energy-memory:` energy used by the memory, in kWh
- `energy-gpu:` energy used by the GPU, in kWh
- `energy-network:` energy used to handle network traffic, in kWh
- `timestamp:` a timestamp for the input
- `duration:` the amount of time, in seconds, that the input covers.

Outputs:

- `energy:` the sum of all energy components, in kWh

Energy is calculated as the sum of the energy due to CPU usage, energy due to network traffic, energy due to memory, and energy due to GPU usage. Using the following formula:

$$\texttt{energy = energy-cpu} + \texttt{energy-network} + \texttt{energy-memory} + \texttt{energy-gpu}$$

Additional details:

In any model pipeline that includes the `sci-o` model , it must be preceded by the `sci-e` model. This is because `sci-o` does not recognise the individual contributions, `energy-cpu`, `energy-network`, etc., but expects to find the total `energy`.

```yaml
name: sci-e-demo
description:
tags:
initialize:
  models:
    - name: sci-e
      kind: plugin
      verbose: false
      model: SciEModel
      path: "@grnsft/if-models"
graph:
  children:
    child:
      pipeline:
        - sci-e
      config:
        sci-e:
      inputs:
        - timestamp: 2023-08-06T00:00
          duration: 3600
          energy-cpu: 0.001
```

Listing 11: `Impl` file for test case using sci-e model

To create the `Ompl` file the following section would be appended at the end of the `Impl` file:

```yaml
outputs:
- energy: 0.00013541666666666666
```

Listing 12: Output section of the `Ompl` file for test case using sci-e model

B.2 Cloud Instance Metadata

It is a model that allows you to determine an instance's physical processor and thermal design power based on its instance name.

Inputs:

- `cloud-vendor:` the cloud platform provider, e.g., AWS
- `cloud-instance-type:` the name of the specific instance being used, e.g., m5n.large

Outputs:

- `cloud-instance-type:` the input instance-type
- `cloud-vendor:` the input vendor
- `physical-processor:` physical processor used in the given instance
- `vcpus-allocated:`the number of vCPUs allocated to this instance
- `vcpus-total:`the total number of vCPUs available to this instance

Additional details:

IF implements this plugin using data from Cloud Carbon Footprint. This allows determination of CPU for the type of instance in a cloud and can be invoked as part of a model pipeline defined in an `Impl`. `Cloud Instance Metadata` currently implements only for 'AWS'

```
name: cloud-instance-metadata-demo
description: example \texttt{Impl} invoking Cloud Instance Metadata model
initialize:
  models:
    - name: cloud-instance-metadata
      model: CloudInstanceMetadataModel
      path: '@grnsft/if-models'
graph:
  children:
    child:
      pipeline:
        - cloud-instance-metadata
      config:
      inputs:
        - timestamp: 2023-07-06T00:00 # [KEYWORD] [NO-SUBFIELDS] time when measurement occurred
          vendor: aws
          instance_type: m5n.large
          duration: 100
          cpu-util: 10
```

Listing 13: `Impl` file for test case using Cloud Instance Metadata model

```yaml
name: cloud-instance-metadata-demo
description: example \texttt{Impl} invoking Cloud Instance Metadata model
initialize:
  models:
    - name: cloud-instance-metadata
      model: CloudInstanceMetadataModel
      path: '@grnsft/if-models'
graph:
  children:
    front-end:
      pipeline:
        - cloud-instance-metadata
      inputs:
        - timestamp: 2023-07-06T00:00
          cloud-vendor: aws
          cloud-instance-type: m5n.large
          duration: 100
          cpu: 10
      outputs:
        - timestamp: 2023-07-06T00:00
          cloud-vendor: aws
          cloud-instance-type: m5n.large
          physical-processor: Intel Xeon Platinum 8259CL
          duration: 100
          cpu: 10
```

Listing 14: `Ompl` file for test case using Cloud Instance Metadata model

B.3 Azure-Importer

The Azure importer model allows you to provide some basic details about an Azure virtual machine and automatically populates your `Impl` file with usage metrics that can then be passed along a model pipeline to calculate energy and carbon impacts.

Prerequisites:

- Create an Azure VM instance
- Provide an identity to access VM metadata and metrics
- Create an App registration/service principal for the Azure Importer
- Provide `IAM` access
- Add credentials to `.env`

Inputs:

- `timestamp:` An ISO8601 timestamp indicating the start time for your observation period. The time span is calculated by adding the duration to this initial start time.
- `duration:` Number of seconds your observation period should last.
- `azure-observation-window:` The time interval between measurements (temporal resolution) as a string with a value and a unit, e.g., 5 mins. The value and unit must be space-separated.
- `azure-observation-aggregation:` This indicates how you want the metrics to be aggregated between each interval. The recommended default is average.
- `azure-subscription-id:` Your Azure subscription ID
- `azure-resource-group:` Your Azure resource group name
- `azure-vm-name:` Your virtual machine name

Outputs:

- `duration:` the per-input duration in seconds, calculated from azure-observation-window
- `cpu-util:` percentage CPU utilisation
- `cloud-instance-type:` VM instance name
- `location:` VM region
- `mem-availableGB:` Amount of memory not in use by your application, in GB.
- `mem-usedGB:` Amount of memory being used by your application, in GB. Calculated as the difference between `total-memoryGB` and `memory-availableGB`.
- `total-memoryGB:` The total memory allocated to your virtual machine, in GB.
- `mem-util:` memory utilised, expressed as a percentage (`memory-usedGB/total-memoryGB * 100`)

```yaml
name: azure-demo
description: example \texttt{Impl} invoking Azure model
initialize:
  models:
    - name: azure-importer
      model: AzureImporterModel
      path: '@grnsft/if-unofficial-models'
graph:
  children:
    child:
      pipeline:
        - azure-importer
      config:
        azure-importer:
      inputs:
        - timestamp: '2023-11-02T10:35:31.820Z'
          duration: 3600
          azure-observation-window: 5 min
          azure-observation-aggregation: 'average'
          azure-subscription-id: 9cf5e19b-8b18-4c37-9541-55fc47ad70c3
          azure-resource-group: my_group
          azure-vm-name: my_vm
```

Listing 15: `Impl` file for test case using Azure-Importer model

```
outputs:
  - timestamp: '2023-11-02T10:35:00.000Z'
    duration: 300
    azure-observation-window: 5 min
    azure-observation-aggregation: 'average'
    azure-subscription-id: 9cf5e19b-8b18-4c37-9541-55fc47ad70c3
    azure-resource-group: my_group
    azure-vm-name: my_vm
    cpu-util: '0.314'
    mem-availableGB: 0.488636416
    mem-usedGB: 0.5113635839999999
    total-memoryGB: '1'
    mem_util: 51.13635839999999
    location: uksouth
    cloud-instance-type: Standard_B1s
  - timestamp: '2023-11-02T10:40:00.000Z'
    duration: 300
    azure-observation-window: 5 min
    azure-observation-aggregation: 'average'
    azure-subscription-id: 9cf5e19b-8b18-4c37-9541-55fc47ad70c3
    azure-resource-group: my_group
    azure-vm-name: my_vm
    cpu-util: '0.314'
    mem-availableGB: 0.48978984960000005
    mem-usedGB: 0.5102101504
    total-memoryGB: '1'
    mem_util: 51.021015039999995
    location: uksouth
    cloud-instance-type: Standard_B1s
```

Listing 16: Output section of the `Ompl` file for test case using Azure-Importer model

List of Tables

1 Secondary Stakeholders' Concerns and Expectations 38

List of Figures

1 IF Overview 7
2 IF Workflow Diagram 10
3 IF Functional view 11
4 IF Deployment view 12
5 Configure stage of model pipeline 15

6 Execute stage of model pipeline 15
7 Domain Story showing the main scenario of assessing environmental impact of software running on Azure instances 26
8 Optimisation Diagram 33
9 Development plan for term2 34
10 Testing methodology 36

References

[1] Greeen Software Foundation. Overview — impact framework, 2023.

[2] Green Software Foundation. Background — impact framework, 2023.

[3] Green Software Foundation. Green software foundation — gsf, 2023.

[4] Green Sofware Foundation. Sci specification, 2023.

[5] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Chasing carbon: The elusive environmental footprint of computing. *IEEE Micro*, 2022.

[6] Our World in Data. Data page: Carbon intensity of electricity per kilowatt-hour. Part of the following publication: Hannah Ritchie, Pablo Rosado and Max Roser (2023) - "Energy". Data adapted from Ember, Energy Institute.

[7] E. lettier. *2023 COMP101 Project briefs.*

[8] Showmick Guha Paul, Arpa Saha, Mohammad Shamsul Arefin, Touhid Bhuiyan, Al Amin Biswas, Ahmed Wasif Reza, Naif M. Alotaibi, Salem A. Alyami, and Mohammad Ali Moni. A comprehensive review of green computing: Past, present, and future research. *IEEE Access*, 2023.

[9] Yang Shen, Zhihong Yang, and Xiuwu Zhang. Impact of digital technology on carbon emissions: Evidence from chinese cities. *Frontiers in Ecology and Evolution*, 2023.

[10] Thibault Simon, Pierre Rust, Romain Rouvoy, and Joël Penhoat. Uncovering the environmental impact of software life cycle. *International Conference on Information and Communications Technology for Sustainability*, 2023.