UCLouvain

École polytechnique de Louvain

Enhancing NPF with state-of-the-art data visualisation techniques

Author: **Philippe Dan**
Supervisor: **Tom Barbette**
Readers: **Nikita Tyunyayev, Céline Deknop**
Academic year 2022–2023
Master [120] in Computer Science and Engineering

Contents

Acknowledgements

This master's thesis would not have been possible without the invaluable guidance of my supervisor, Tom Barbette. Throughout my work, he provided profound insights while allowing me the freedom to shape the solutions with my ideas. I am also deeply grateful to my two readers. Nikita Tyunyayev offered me constant feedback and showed me how to enhance my paper, especially when I needed it the most. Celine Deknop instilled confidence in my work and provided reassurance regarding my thesis.

Moreover, I would like to express my heartfelt appreciation to my friends, who unwaveringly believed in me and patiently listened to my discussions about my thesis. A special thanks to Loic, my steadfast friend and business partner, for his genuine support. I am grateful to Javier for his unwavering kindness and constant presence, even from a distance. And to Roman, thank you for standing by me all these years, uplifting my spirits during the most challenging times.

Finally, I would like to extend a special thanks to my parents for everything they have given me and for taking care of me throughout the years.

These past five years have been filled with incredible experiences and amazing relationships built along the way. Thank you all.

Abstract

Data visualization is a crucial area of study that enables the transformation of raw information into actionable knowledge. Currently, although researchers face no challenges in data collection thanks to the availability of various automation tools, there is a lack of progress in the development of data visualization tools. Consequently, we have created npf-web-extension as an accessible and interactive solution for visualizing data.

During our research, we explored different architectures while adhering to our vision of offering a seamless and intuitive data visualization tool. Our solution had to be both modern and maintainable, as we aimed to create a tool that could be extended and serve as a foundation for future research and development. Given these circumstances, several design choices were made. On one hand, we built the entire software using React and Typescript to meet quality standards. On the other hand, we wrapped the tool with a Python package, enabling users to quickly get started with it.

As a result, this combination yields software that is not only easy to install but also integrates smoothly into existing workflows, distinguishing it from other available solutions. Currently, our solution supports four different chart types: line chart, bar chart, box plot, and pie chart. It also allows the partitioning of complex data into subsets to create multiple charts from a single dataset.

To validate our software, we conducted a comprehensive test suite to assess our code, and we also gathered feedback on the interface. This master thesis highlights some of the main difficulties when developing a data visualisation tool. Overall, users highly appreciated the interactivity and simplicity of the interface, while their suggestions for missing features were taken into account for our roadmap. Consequently, our solution strikes an excellent balance between ease of use and functionality.

Chapter 1

Introduction

1.1 Context

In today's data-driven world, data centres all around the world hold a vast amount of information. Organisations, individuals and researchers cope with the challenge of extracting meaningful insights from extensive datasets. Figure 1.1 illustrates a substantial surge in the number of published papers incorporating the term 'data visualisation', almost tripling over 5 years. Data visualisation has emerged as a crucial bridge between raw data and actionable knowledge. It is not only a way of building visually appealing graphs but also a matter of experimenting with the data and providing interactive representations. Data analysts need to have a particular set of skills that includes decision-making abilities and pattern identification. As a result, great data analysts also need a top-notch set of data visualisation tools that allow them to explore the data efficiently and easily.

1.2 Motivation

Open source tools have been developed for this purpose like Grafana and Matplotlib. There are also proprietary products, such as Tableau, that require a paid subscription to access the software. Nevertheless, most of these tools usually require either a complex setup (such as a backend server), or certain programming skills. Furthermore, they also typically require an internet connection to work. This becomes problematic for researchers who perform experiments in offline clusters but also for those who have systems that interact with each other and need to collect metrics. Running experiments, collecting the data, and then exploring it is one of the leitmotivs of a researcher. In this context, research has been done to find a tool to fit our needs. In other words, a tool that would save us time by automating our process. Amongst all of the tools we explored, there is one

that got our attention. A research team at the Technical University of Munich developed POS as 'a structured experimental workflow that allows the creation of reproducible experiments without requiring additional efforts of the researcher' [1]. However, despite the promising nature of the tool developed by the research team at the Technical University of Munich, we found its data visualization feature to be rather limited for our specific requirements. As a result, we developed our own visualisation software, based on NPF, that not only fulfils our needs but also surpasses the capabilities offered by POS. We delve into an illustrative comparison between our software (npf-web-extension) and POS in a subsequent chapter of this document.

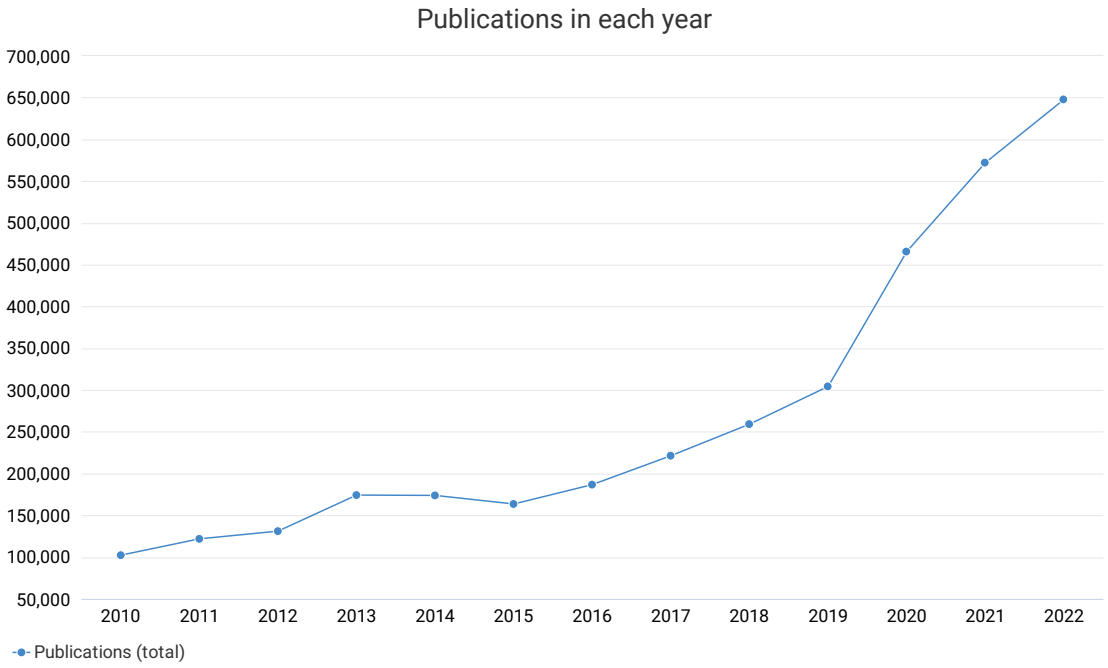


Figure 1.1: Number of papers containing the words 'data visualisation' per year since 2010. There is a clear rising trend over the last five years.

1.3 Challenge

We summarize here the main difficulties of each step involved in the experimental design process. They are described in detail in the next chapter.

1.3.1 Data generation

Producing data should occur through a clear protocol that allows the experiments to be reproduced. The tool needs to have enough abstraction to not interfere with the data being generated and isolate each experiment. All of this while also allowing the researcher to customise the experiment to fit their specific needs in terms of metrics and experimental conditions.

1.3.2 Data collection

Once the data is collected, it has to be stored. 'What format is best to use?', 'Where do we store it?', 'How to ensure privacy is respected when dealing with data collected from real users?', these are some of the questions that make the data collection step difficult. They need to be taken into account when developing a solution like ours.

1.3.3 Data visualisation

The tool should be able to handle a broad set of data types while also allowing researchers to interact with the data. Fine-tuning the set of features available requires a thorough validation through real use cases with an agile development approach. Furthermore, this step typically involves a different technology stack from the previous ones because of the need for a user interface.

1.4 Goals

With this thesis, our objective is to explore the capabilities of NPF and introduce our extension 'npf-web-extension' dedicated to data visualisation. Our goal is to enable researchers to face fewer issues with the setup part of an experiment and allow them to quickly visualise the collected metrics via our offline data visualisation tool. Our extension requires little configuration and allows us to encompass the whole experimental design workflow into a single tool that fits most of the needs of a researcher.

1.5 Outline

In this document, we will start by explaining the experimental design process and then proceed with describing the research that has been done so far regarding network experimentation tools. Chapter 3 will state what to keep in mind when visualising data and compare some commonly used visualisation tools.

Chapter 4 details the process of finding the best tools to fit the requirements of the solution. Then, we proceed in chapter 5 with an explanation of the software architecture. Chapter 6 illustrates the interface through a series of figures.

Finally, we talk about how we validated the solution and compare it with some alternatives in chapter 7 then end with the future work inside chapter 8 and our conclusion in chapter 9.

Chapter 2

Network experimentation and state of the art

This section is dedicated to referencing all the previous work related to network experimentation and providing a comparison between the solutions available. This comparison will also highlight where the extension fits.

2.1 Experimental design

Experimental design is the art of studying the causality between variables and their effects on the performance of software or system. This step is important when designing systems because it allows us to draw practical conclusions on the behaviour of the software or system in some extreme edge cases that might be difficult to reproduce outside the experimental scope. Usually, experimental design involves three steps for the data: generation, collection, and visualisation.

2.1.1 Data generation

The data is usually generated by having a test bench running a myriad of specifically chosen tests. This step typically involves three entities with their role:

- **Controller:** orchestrate the experimentation, collect data and publish the data afterwards.
- **Traffic Generator:** generate the traffic required for the experimentation.
- **Device under test (DuT):** undergo the various tests that are run

2.1.2 Data collection

For each test, data has to be collected and stored for post-experimentation data visualisation. An important aspect to keep an eye on at this step is the format. Using a common data storing format such as *CSV* or *JSON* will open the door for more data visualisation solutions.

2.1.3 Data visualisation

Experimentation serves the purpose of decision-making. It is important to analyze the output of an experiment and draw the right conclusions. Therefore, optimizing the data visualisation process through the right data representation will ease the process of analyzing the data.

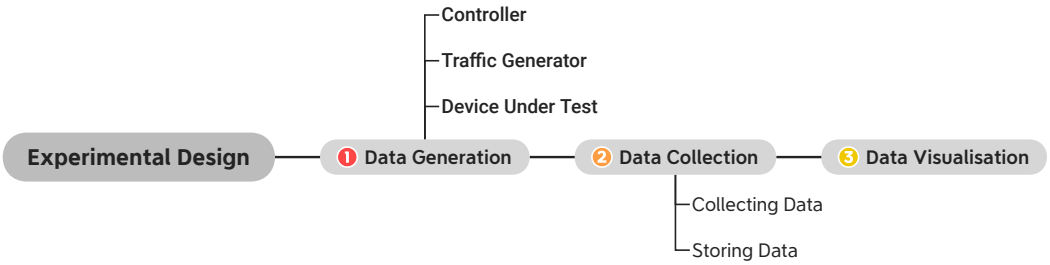


Figure 2.1: Illustration of the three steps of experimental design

2.2 Traffic generation

Traffic generation is an important part of the whole experimentation design. We underline hereunder the key points of traffic generation.

2.2.1 Reproducibility

A crucial part of traffic generation is its reproducibility. Solutions that allow for later reproducibility of the traffic generated are preferred. Indeed, having reproducible tests allows the software designer to iterate over their implementation and rerun the same tests with a different software configuration. Without this central property, network experimentation would be chaotic and improper conclusions would be drawn.

2.2.2 Natural packet generation

Natural packet generation is usually achieved by replaying previously created network flows that have been recorded beforehand. Typically, this is done via the use of PCAP files.

2.2.3 Synthetic packet generation

Generating packets synthetically can be done either software side or hardware side. Commonly used software tools for packet generation include:

- **MoonGen:** DPDK-based packet generator tool able to generate more than 20 mpps[1]/core with high precision measurement capabilities. MoonGen is completely free and scriptable through the Lua language. [2]
- **iPerf:** Cross-platform tool that supports various protocols (TCP, UDP, SCTP with IPv4 and IPv6) for active measurements of the maximum achievable bandwidth. [3]
- **FastClick:** Extended version of the Click Modular Router featuring improved Netmap support and a new DPDK support. [4]
- **TRex:** Based on DPDK, it is open source and supports 10-30 mpps/core. It is backed by Cisco. [5]

Regarding the hardware side, the most common tool is **Open Source Network Tester (OSNT)**. It is a low-cost, open source and extensible hardware traffic generator based on the NetFPGA platform [6].

2.3 State-of-the-art

2.3.1 Plain Orchestrating Service (POS)

The most recent development in terms of network testing framework has been made at the Technical University of Munich. Plain Orchestrating Service [1], or POS, is a fully configurable workflow orchestration testbed and toolchain designed to ease the reproduction of network experiments for researchers. Although POS can be configured with another packet generator, it relies mainly on MoonGen[2].

POS ensures all experiments are heterogeneous, isolated, recoverable, automatable, and publishable.

[1]million packets per second

2.3.2 Network Experimentation Programming Interface (NEPI)

NEPI[7] has been released several years before POS and has been an inspiration as such. The distinction between NEPI and POS lies in the ability of POS to publish the data after the experimentation.

2.3.3 OMF-F

Initially, OMF is an orchestration framework designed to ease the process of managing testbeds. It has been extended by T. Rakotoarivelo, G. Jourjon and M. Ott [8] to support more complex network experimentation scenarios and improve OMF's scalability.

2.3.4 Papermill

Although this is not specifically made for network experimentation, it still provides new opportunities for researchers to use Jupyter as a workflow for experimentation by allowing them to parameterize and execute Jupyter Notebooks. Further details are provided in the docs [9]. The downside of using it as a network experimentation tool is its lack of isolation tools.

2.3.5 NPF

NPF [10] is the foundation of our software (npf-web-extension) since we extend its features to bring those missing data visualisation features. It is a fully featured experimentation software as it can generate data through external libraries (such as iPerf), collect them and visualise them by exporting graphs. It works with user-defined scripts which allow the user to define a set of actions to follow for the experiments along with a set of variables to test and required softwares to build or download. NPF can output the collected data into a pandas data frame or a CSV format. There are three main components inside NPF:

- **npf-run** to run multiple tests across one repository
- **npf-compare** to run one script across multiple repositories
- **npf-watch** to watch for changes inside a repository, monitor performances and notify administrators in case of performance regression

NPF uses Matplotlib internally to generate graphs. Although there are command-line options to alter slightly the graph generation feature, it only allows for limited interactivity with the data.

2.3.6 Comparison

The state of the art is compared with the following requirements.

R1. Configuration A network experimentation tool should allow researchers to fully configure its behaviour and the experiments they run.

R2. Isolation Network experiments should ideally run within an isolated environment. Isolation allows for the mitigation of unwanted effects and maximizes the reliability of the results.

R3. Reproducibility Hopefully, the experiments run with a specific technology should be reproducible. In other words, the configuration part should allow an external researcher to run the same experiments with the same settings.

R4. Data visualisation Once the data has been collected, this point evaluates how data is made available to the qualified network researchers.

	Configuration	Isolation	Reproducibility	Data visualisation
OMF-F	n.a.	n.a.	✓	×
NEPI	∼	n.a.	✓	×
POS	✓	✓	✓	∼
Papermill	✓	×	✓	∼
NPF	✓	✓	✓	∼
NPF extended	✓	✓	✓	✓

Chapter 3

Visualisation background

This section relays all the best practices in terms of data visualisation as well as listing and comparing some visualisation tools. We based this section on the excellent book written by Claus O. Wilke 'Fundamentals of data visualization' [11] and the paper by Stephen R. Midway 'Principles of Effective Data Visualization' [12].

3.1 Data visualisation techniques

3.1.1 Best practices

When it comes to data visualisation, some key characteristics are central. Keeping them in mind right from the start ensures we are on the right way to a great data visualisation set.

- **Make the objective clear** Before starting to represent some data, the objective of the visualisation should be defined. The purpose must be known before designing anything.
- **Emphasize the right data** Having a cluttered graph with tons of parameters is a barrier to visualisation. Thus, the number of different parameters represented should be kept minimal.
- **Pick the right technique** There are many types of charts to pick. They all have a specific purpose and it is important to associate the right data with the right chart.

3.1.2 Types of data

Fundamentally, there are two categories of data: conventional data and high-volume data characterized by its volume, variety, and velocity. Figure 3.1 is an illustration from Google Cloud's blog explaining how to choose the right chart type based on the data [13].

Conventional data

- **Line charts** Good for basic data showing the relationship between two or three patterns at most.
- **Pie charts** Used to represent components percentage of a whole.
- **Bar charts** Best used to compare items of different groups when the amount of data is not huge.
- **Area charts** Best choice when there's a need to show the trend over time.
- **Scattered plots** Shows how close different groups of data are to each other or rather their correlation.
- **Bubble charts** Variation of scattered plots. Fits well for large sets of data.
- **Tree maps** Shows data in a hierarchical form.
- **Heat maps** Compares data visually using colours to distinguish the 'hot' vs 'cold' areas.

High-volume data

High-volume data requires attention to the three V's:

- **Volume** size of the data
- **Velocity** how often the data changes
- **Variety** audio, film, etc.

The main types of high-volume data graphs are :

- **Word clouds** visual representation of text data. It highlights the most frequently used keywords.
- **Symbol maps**
- **Connectivity charts** shows the connection and its strength between actions and their triggers.

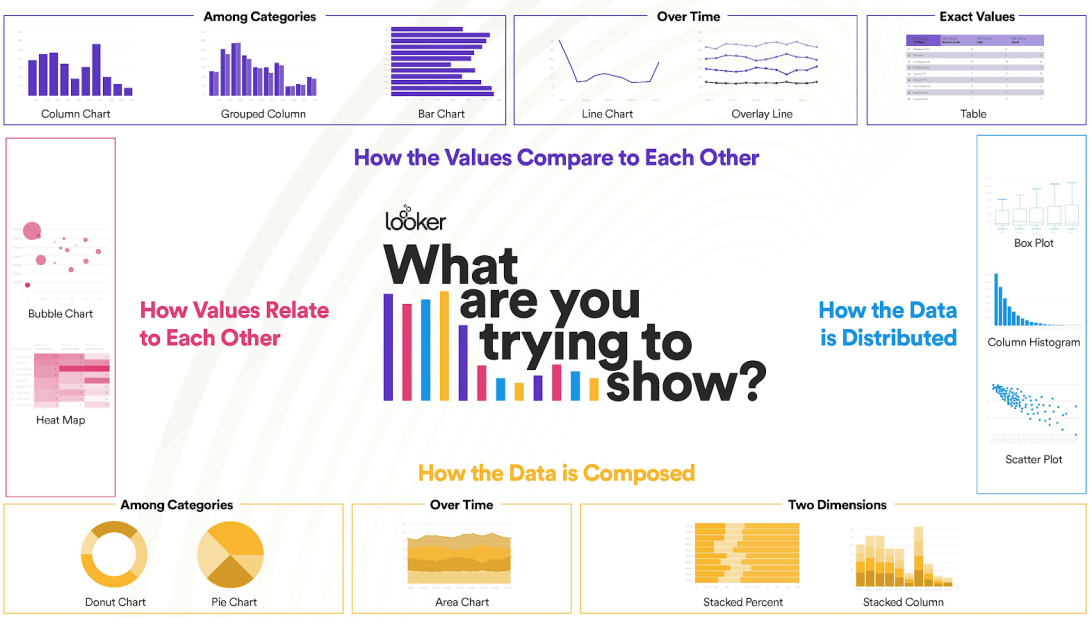


Figure 3.1: Image from Google Cloud's blog on how to pick the right chart type

3.1.3 Other important aspects

Coloring

Colours are excellent for having a higher memorability score. One shall use colours that can be converted to an effective grey scale. Usually, seven or more colours are best.

Paneling

When having frequent small changes amongst the data for different variables, it is interesting to use multiple small figures instead of a big one while still keeping the same overall design and visual structure.

Captions

Detailed captions should be used that fully explain everything in the figure. Ideally, the figure and the caption should be independently self-sufficient.

3.2 Optimizing graph communication

Matching one's available data with a graph type is not an easy task, and neither is optimizing the final graph to effectively communicate the data. We summarize here three key ideas to elevate the quality of a graph, taken from the pioneering work done by William S. Cleveland in his book 'The Elements of Graphing Data' [14].

3.2.1 Data-to-ink ratio

The data-to-ink ratio refers to the proportion of ink or graphical elements used in a graph that directly represents the underlying data. The goal of maximizing the data-to-ink ratio is to reduce non-essential or redundant graphical elements, such as excessive gridlines, unnecessary decorations, or embellishments, that do not contribute directly to conveying the data. By minimizing unnecessary ink, we simplify the visual representation and reduce potential distractions, enabling viewers to more readily discern and interpret the data being presented.

3.2.2 Graphical integrity

By graphical integrity, William S. Cleveland refers to representing the data in such a way as to reflect accurately the underlying data by avoiding distortions or misleading visual elements. Scales are typically elements that should be properly chosen to represent the data faithfully.

3.2.3 Effective graphical elements

Effective graphical elements are visual components or properties in a graph that are well-suited for accurately representing and conveying data. These elements should be carefully chosen and designed to enhance the understanding and interpretation of the information presented in the graph. Examples of effective graphical elements include position, length, angle, colour, shape, and texture. The effectiveness of graphical elements in a graph is often determined by their perceptual characteristics and their ability to accurately encode and communicate the underlying data. For instance, position and length are typically more accurately perceived and interpreted than angles or colour shades. By gaining an understanding of the perceptual strengths and limitations associated with various graphical elements, designers can make informed decisions and select the most appropriate ones to effectively convey information for a specific dataset and visualization objective. This knowledge enables designers to optimize the choice and utilization of graphical elements, enhancing the overall effectiveness of the graph in accurately representing and communicating the intended data.

3.3 Available tools

This section details some of the available data visualisation tools. This list is not exhaustive, however, we gathered the most commonly used tools both within the industry and by researchers.

3.3.1 Grafana

Grafana [15] is an open-source software with more than 10 million users worldwide which focuses mostly on the creation of dashboards based on a set of data. It has a lot of plugins that can be configured to integrate external data sources such as google sheets, PostgreSQL, MongoDB, etc. It supports a large variety of visualisation types. It is free for use with a local setup, although additional premium plans are provided through the Grafana Cloud platform.

3.3.2 Tableau

Tableau [16] is a premium software oriented towards data visualisation for businesses. It has a multitude of extensions that allow it to connect the software to external data sources. It is part of the Salesforce platform and offers different plans based on one's needs. To use Tableau, the user is required to download their software.

3.3.3 Microsoft Power BI

Power BI [17] is a business intelligence tool developed by Microsoft. It allows users to connect to various data sources and create interactive visualizations, reports, and dashboards. Power BI is a premium software which is part of the Microsoft Azure infrastructure and powered by their cloud platform.

3.3.4 Looker Studio

Looker Studio, formerly known as Google Data Studio, is a free data visualization software [18] geared towards businesses. It allows users to connect to various data sources and create interactive reports and dashboards. With its extensive library of connectors, Data Studio enables seamless integration with external data. As a part of the Google ecosystem, it provides collaborative features and different sharing options. Accessible through a web browser, Data Studio eliminates the need for software installation and offers a user-friendly experience for data visualization.

3.3.5 Plotly

Plotly is a versatile data visualization library [19] that caters to businesses. It offers a comprehensive range of interactive charting options, making it ideal for creating engaging visualizations. With support for multiple programming languages, including Python and Javascript, Plotly enables seamless integration with various data analysis workflows. It provides extensive customization capabilities and interactive features, allowing users to explore and analyze their data effectively. Plotly is a valuable tool for data visualization and analysis, offering flexibility and a wide array of visualization options to meet diverse business needs.

	Opensource	Serverless	Programming skills needed
Grafana	✓	×	Yes
Tableau	×	×	No
Power BI	×	×	No
Looker Studio	×	×	No
Plotly	✓	✓	Yes

Table 3.1: Visual comparison of the five tools listed above

Chapter 4

Technical foundations

4.1 Purpose and approach

The goal of this thesis is to enhance the capabilities of NPF by providing better ways to publish the experimental data through the development of a web page generator extension. The key characteristics of these web pages are :

- **C1** Easy to share, any modern browser should be able to open them.
- **C2** Self-sufficient, no external configuration or setup should be required to visualise the data. The solution might be used in clusters where access to the internet is not provided. Thus, no extra steps should be involved other than installing and configuring NPF.
- **C3** Visually attractive, all data should be easy to read and pleasant to the eye.

Since the final result is delivering to the end user a web page that can be accessed by any modern browser and without any external configuration, there are two possible ways to approach this.

4.1.1 Data oriented approach

The first solution would be to rely upon the technologies already used by NPF (Python) to generate the HTML code needed to port the plots currently generated through Plotly into the browser. This way, the data is generated before the website and the latter is built around the data. The following table lists the pros and cons of such a solution.

Pros	Cons
No need for external packages, everything stays inside NPF which simplifies the architecture.	Development is slowed down because everything is tied to the Python environment
Easily available offline	The final result might be not as interactive as expected
	There's a need to reinvent the wheel and/or provide a lot of work into integrating web frameworks into the workflow
	Less flexibility since development is directly tied to NPF's development
	Integrating the tool into a workflow different from NPF is impossible

Table 4.1: Pros and cons of generating the website within NPF

4.1.2 Website oriented approach

The other solution is using powerful web frameworks like React or Angular to build the website and then insert the data inside the website.

Pros	Cons
More flexibility since we are relying on modern web technologies	More challenging since we need a way to connect the website and NPF and the overall architecture is more complex.
Total control over interactivity and website development	
Faster prototyping and development of the website	More complex since we need to maintain two projects.
The tool can be integrated into workflows different than NPF	

Table 4.2: Pros and cons of generating the website outside of NPF

The first option looks easier to implement, yet we chose the second one because it provides more flexibility and access to powerful tools. The most interesting

challenge was linking everything together, which we solved by wrapping the solution with a Python interface. This is explained later in the document.

4.2 Building the foundations

To generate the web page we had to pick a frontend framework that meets the following requirements :

- **R1** Modern and maintained. Since the web is an environment where technologies evolve constantly, our solution needs a framework shipped with constant updates and unlikely to become outdated in the following years.
- **R2** Serverless. Most web frameworks come in two categories. On one hand, we have frameworks that do **client side rendering** where the browser renders all the HTML code required for the application to run. On the other hand, we have frameworks with **server-side rendering** where the server generates the HTML code before delivering it to the client. A serverless framework is preferred because it results in an easy-to-share and self-sufficient solution.
- **R3** Large community. By choosing a framework with a large community, there are greater chances of finding mature solutions to the potential problems occurring in the development phase.
- **R4** Learning curve. Ideally, the framework should be easy to learn. Emphasis is put on the frameworks we already had a grasp on.

4.2.1 Choice of the framework

Measuring the size of a community is difficult, therefore we can ensure picking a framework with a large enough community by looking at the most used web frameworks. Below lies 15 of the most used web frameworks in 2022 according to Statista* [20] along with some additional details, of which :

- **Ranking**
- **Technology** describes the language and technology mostly used within the framework. JS and TS stand respectively for Javascript and Typescript.
- **Rendering** describes whether the framework uses SSR (server-side rendering), CSR (client-side rendering) or mixed.
- **Purpose** describes in a couple of words what are the typical use cases based on the official website of the framework. We regrouped them under the following categories: frontend, backend, API, and other.

- **Proficiency** describes how well we know the framework.

*We excluded the following from the list provided by Statista:

- **Nodejs** is a backend Javascript runtime environment and not a web framework designed to build websites.
- **Jquery** is a Javascript framework and not a web framework designed to build websites.

#	Name	Technology	Rendering	Purpose	Proficiency
1	React.js	JS, Nodejs	CSR	frontend	+++
2	Express	JS, Nodejs	SSR	backend, api	++
3	Angular	TS, Nodejs	CSR	frontend	+
4	Vue.js	JS,TS, Nodejs	CSR	frontend	+
5	ASP.NET Core	.NET, C#	SSR	front, back, API	+
6	ASP.NET	.NET, C#	SSR	front, back, API	+
7	Django	Python	SSR	front, back, api	+
8	Flask	Python	SSR	front	+
9	Next.js	JS/TS, Nodejs	SSR	frontend, backend	+++
10	Laravel	PHP	SSR	frontend	+++
11	Angular.js	JS	CSR	frontend	+
12	FastAPI	Python	SSR	api	+
13	Ruby on Rails	Ruby	SSR	frontend	+
14	Svelte	JS,TS	SSR	frontend	+
15	Blazor	.NET, C#	SSR	frontend	+

Based on the table and the criteria listed above, the most suitable frameworks are React, Angular and Vue. All the other frameworks lean too much towards SSR which would slow down the prototyping phase. Therefore, given our proficiency levels, **React** is the solution that fits best.

4.2.2 Choice of the plotting library

The selection criteria for the plotting library are the following:

- **C1** Compatibility with React. The integration with React should be seamless.
- **C2** Features available. The library should be able to build any of the possible types of graphs listed in chapter 3.

- **C3** Actively maintained. The library should have an active community providing updates continuously.
- **C4** Opensource. The library should be available for free and ideally open source.

Here is a list of commonly used plotting JS plotting libraries.

Name	Compatibility	Features	Actively maintained	Opensource
Chartjs	✓	∼	✓	✓
Plotly	✓	✓	✓	✓
D3.js	✓	✓	✓	✓
Highcharts	✓	✓	✓	×
ApexCharts.js	✓	✓	✓	✓
Google Charts	✓	✓	✓	✓

After some quick prototyping and exploring the demos, Chartjs with its wrapper for react (react-chartjs-2) was easier to get started with and the most customizable. It also provides interesting out-of-the-box interactivity features and is also highly performant as shown by this codepen. [21]

4.3 Setting up

So far, we explained how we picked the main framework (react) and the plotting library (react-chartjs-2). Additionally, we also needed :

- **A CSS framework** to determine the style of the web page.
- **A development language** since react can be used either with Javascript or Typescript.

Since we have already worked with React in the past, we are used to getting started with Tailwind CSS [22] because it allows us to do fast prototyping without having to worry too much about styling. Using Typescript[1] is preferred to pure Javascript because it accelerates the development process by setting up a development less prone to errors and bugs.

[1]Typescript is a powerful extension of the Javascript language that brings in a lot of syntactic sugar while developing such as type inference, in-editor syntax error detection and much more. [23]

Chapter 5

Implementation

At this stage, we described the foundations of our architecture. In this chapter, we iterate over the challenges that we faced during the implementation of our architecture and how we solved them. Most of the challenges we faced during the development were due to our requirements. Indeed, we wanted our solution to result in an 'easy-to-install, easy-to-use' software. As a reminder, here are the characteristics of the solution we envisioned:

1. **Easy to set up**: As mentioned earlier, NPF might be used in clusters where access to the internet is not provided. Therefore, ideally, the solution should come already shipped with NPF. Since NPF is built with Python and exported to PyPi, the simplest solution is to build another Python package and add it to the dependencies of NPF.

2. **Easy to use**: When exporting the results, it should not involve any complicated process. Ideally, exporting the results should only require some data processing to inject into the extension. The Python package will act as the interface between the web app and NPF.

3. **Easy to share**: All the results should be shipped into one final HTML file that is compatible with most modern browsers.

To qualify the term 'modern browser', we refer to the data provided by W3Schools [24] (more than 60 million users per month) which shows that the most used browsers are Chrome, Edge, Firefox, Safari and Opera (together representing 99.2% of all traffic, as of February 2023). Since Chrome, Edge and Opera use the Blink Engine [25], taking into account Chrome instead of all three is likely to be enough.

5.1 General workflow

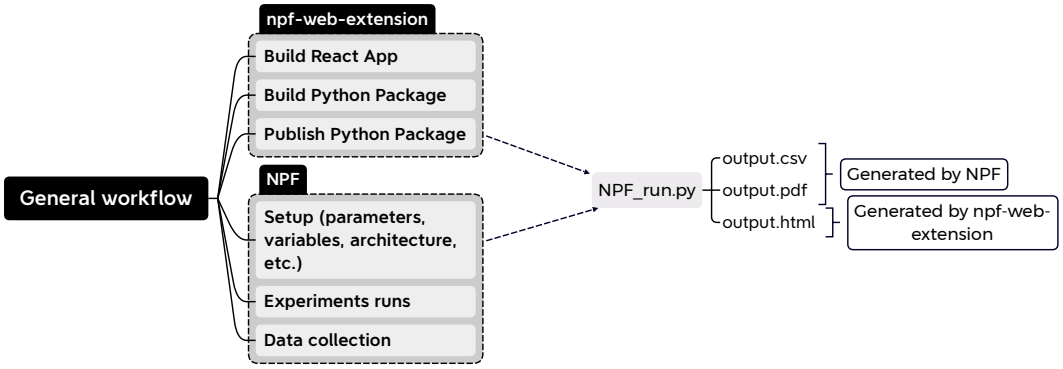


Figure 5.1: The big picture illustrating the general workflow behind the final solution

The general workflow of the solution consists of two parts. On one hand, there is the npf-web-extension workflow that consists in:

- Building the react app and generating the template
- Building the Python package and the interface for NPF
- Publishing the package to PyPi along with the template

On the other end, there is NPF that runs all the experiments with the **npf-run** and exports the results via the **npf-web-extension** package along with the initial outputs of **npf-run**.

5.1.1 React setup

The react app has been initialized with the template generator provided by create-react-app [26] (CRA). Since React works on top of Nodejs with Javascript components (JSX)[1], it cannot be used directly within a browser (except if it is added to a website and rendered with the ReactDOM utility). Therefore, it is necessary to build the React app into a format that can be used in browsers. This is where the generator comes in handy. It eases a lot the creation of React apps by encapsulating all the more advanced settings and providing ready-to-use React apps. For instance, create-react-app takes care of:

[1]JSX components can be seen as an enhanced version of HTML that has been empowered with Javascript capabilities.

- **Transpiling** the source code from Typescript to Javascript. Typescript is an extension of Javascript and is not directly usable within the browser. The app has been set up to use the default Typescript compiler (tsc).
- **Bundling** all the modules (also known as Javascript packages) used within the react app into a single Javascript file and chunk it into multiple pieces if needed (for optimization). This is done via the use of Webpack.
- **Hot reloads** via the development server that allows developers to iterate quickly through the development cycle without having to rebuild the app after every modification to the code.
- Many other things such as dependencies, linting, maintenance, etc.

Essentially, it comes with these essential commands (npm scripts):

- **start** to start the development server for hot reloads
- **build** to build the app and bundle everything
- **test** to run JUnit tests
- **eject** to eject the create-react-app wrapper and expose all the internal settings such as webpack, tsc, etc. It is preferable to avoid ejecting the wrapper as it is a one-way step and getting future updates is incredibly more difficult afterwards.

Fixing the default build

The default build with create-react-app did not meet our requirements. We encountered two issues related to the build process:

- The bundled Javascript files are injected in the <head> part of the website. Normally, it is considered best practice performance-wise to place <script> tags inside the body tags. However, react injects them as external sources and defers them. This means that they will be downloaded in parallel to loading the page. This is the opposite of what we want, which is only a single HTML file.
- The bundled Javascript files are chunked into several pieces for optimization purposes. Therefore, there is only one main script that is injected into the template and the other chunks are loaded asynchronously by the main script.

To fix both of these issues and end up with only one HTML file, we had to change the build settings of create-react-app. However, create-react-app does not expose its internal settings out-of-the-box without the eject app feature. The eject app approach complicates the maintenance process since the create-react-app package is removed from the dependencies. At the same time, since we had to change only a small subset of the settings, ejection seemed excessive as it exposes all the settings.

Hopefully, this approach has been avoided with the use of **CRACO** [27]. CRACO stands for 'Create React App Configuration Override' and allows for quick and easy internal configuration override of create-react-app. With this package, we managed to update the webpack configuration and fix both of the issues mentioned above. Nevertheless, the build output still has the wrong format. Indeed, the template has references to the external resources and they are not injected into the template.

There were a couple of solutions to inline[2] JS and CSS sources but Gulp[3] turned out to be the easiest to implement. There is a gulp plugin called 'gulp-inline-source' [29] that helped us inline all the sources within an HTML file.

Entry point

After ensuring that the build process generates only a single HTML file with all the sources inlined, we had another concern. An entry point inside the HTML file was still needed to inject the data from NPF (or the wrapper). The difficulty arises from the need to interact with react app components after the export process to inject the data. Indeed, once the app is bundled, there is no straight way to interact with the components states and we had to rely on a workaround: the inherent Javascript event listener methods.
There is one event handled when the configuration is being injected:

- `APP_READY`: this event is fired by the app once it is ready to get updated.

This alternative was the easiest to implement because it did not rely on any external packages while still allowing asynchronous app hydration.

5.1.2 Python package setup

To link the react app with NPF, an interface was necessary to ease the process of injecting the configuration inside the app. A Python package was built with the

[2]Inlining refers to the process of getting rid of external sources and injecting the source code into the main file.

[3]Gulp is a toolkit that helps automate painful or time-consuming tasks in the development workflow [28].

Python default package distribution builder. During the development, there were two viable options:

- Either build and export only the Python interface along with a pre-built app template.
- Or build and export everything including the Python interface, the pre-built app template and the app sources.

The first solution is the lightest regarding the size of the distribution but it is also the less flexible one. Indeed, the development of this architecture is mainly focused on integrating it with NPF but we also want to allow its use outside of NPF. Therefore the second alternative seemed a better choice as it allows anyone to modify locally the template and customize it for their needs.

5.1.3 Package release

The package is published to the Python Package Index platform (PyPi) [30][4]. An alternative would have been to publish it to Conda[5] but for such a small project, there are no real benefits in choosing one over the other. Therefore, this platform has been chosen for simplicity as NPF is also published with PyPi.

5.1.4 Organization of the repository

To ensure the quality of the repository, several initiatives have been set up.

Main branch protection

The main branch of the repository is protected against direct commits, therefore any changes to the repository must follow the traditional pull request process. This protection can be bypassed by administrators of the repository if needed or by CI/CD automation.

Status checks and approvals

All pull requests:

- must be up to date with the main branch
- must pass the CI/CD workflow dedicated to pulling requests
- must be approved by code owners

[4] `https://pypi.org/project/npf-web-extension`
[5] Conda is the package index platform that comes with Anaconda [31].

Automation

There are two main CI/CD workflows designed to assess the quality of the repository.

Build checks This workflow is triggered on all pull requests. It is intended to check that:

- the app builds properly
- package tests run properly

These checks are minimal but required to be sure that the release will be successful.

Build and release This workflow does much more work than the previous one. It is triggered when a pull request has been merged into the main branch. Its behaviour is illustrated in figure 5.2.

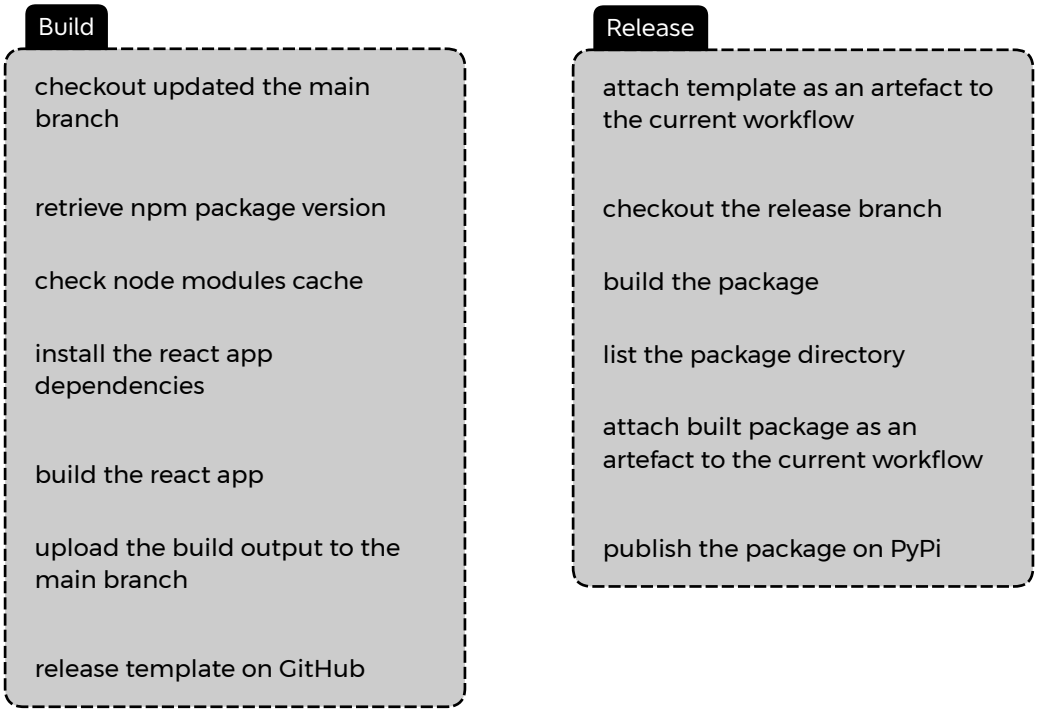


Figure 5.2: Illustration of the build and release automation

5.1.5 NPF usage with npf-web-extension

Using npf-web-extension inside NPF is effortless. It is only necessary to specify the `--web` option to `npf_run` along with an export path. For instance, the following

command runs the `iperf` tests and exports everything to `tmp.html`.

```
python3 npf_run.py --test tests/tcp/01-iperf.npf --web tmp.html
```

5.2 npf-web-extension interface

The package can be used outside of NPF. There are two methods available inside npf-web-extension. A `demo` method which toggles the demonstration datasets of the application and an `export` method with two parameters:

- **configurationData**: configurationData or array of configurationData to inject into the template
- **outdir**: output path of the exported template

Chapter 6

Interface and features

6.1 Main interface

Figure 6.1 shows the main interface when opening an HTML file that has been generated with npf-web-extension. There are three main sections to explore:

1. A navigation menu that allows the end user to explore the available datasets.
2. A container where the graphs are being generated.
3. A toolbar with more actions available:
    - A **settings button** that opens a modal with a variety of parameters to tune to fit the end user's needs in terms of graph rendering.
    - A **full-screen button** which expands the rendering component to get rid of the navigation menu and toolbar. This mode can be exited by pressing the 'Escape' key.

6.2 Settings interface

6.2.1 General settings

All settings in this section allow the end user to change how the graph is rendered. When altering these settings, they are saved automatically locally. However, they are not saved into the HTML file (this is an actual limitation of browser features for security concerns). The following points are also detailed inside our updated online documentation available at `https://dpcodebiz.github.io/npf-web-extension/#/`.

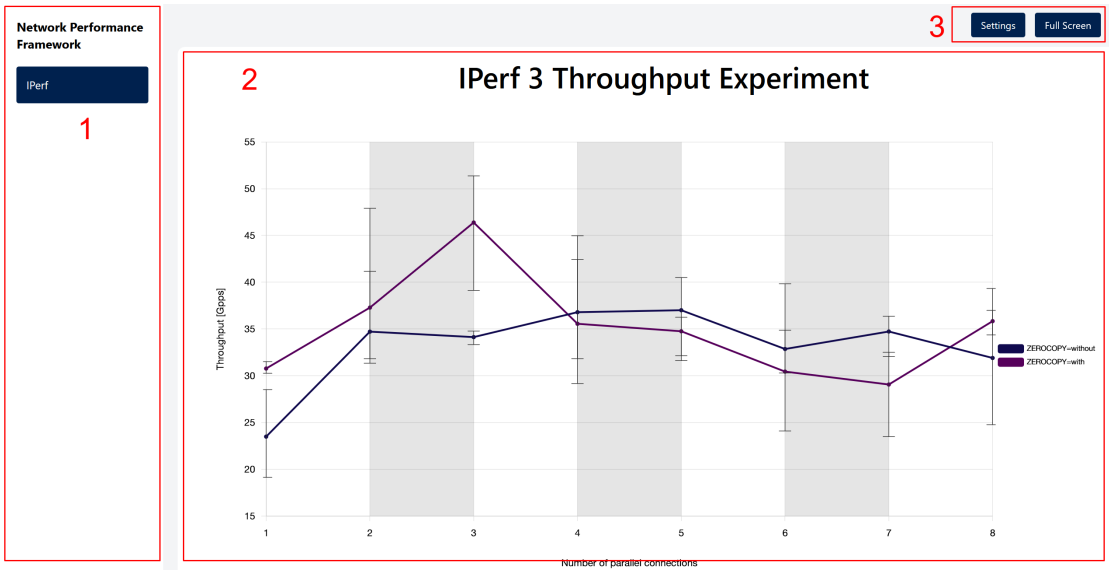


Figure 6.1: Illustration of the main interface.

Title

By default, the title is set to the one provided by the relevant NPF test to which the dataset is tied. When using npf-web-extension as part of NPF, this corresponds to the **title** attribute of the **ConfigurationData** object. All of this can be changed via this input.

Chart type

The application can infer a recommended chart type based on the parameters and measurements present in the dataset. The algorithm is fairly simple and described in figure 6.3. Unfortunately, it was not possible to develop this feature to recommend more graph types because of the following:

1. Bar charts and line charts are too similar and choosing one over another is a matter of preference.

2. Pie charts are used to show proportions between different subsets. However, it is difficult to define an algorithm that relevantly recommends a pie chart based only on a dataset.

Therefore, we did not dive too much into this feature as this is a classification problem that could be achieved more easily with a machine learning approach rather than a single algorithm. This is discussed further in chapter 8.

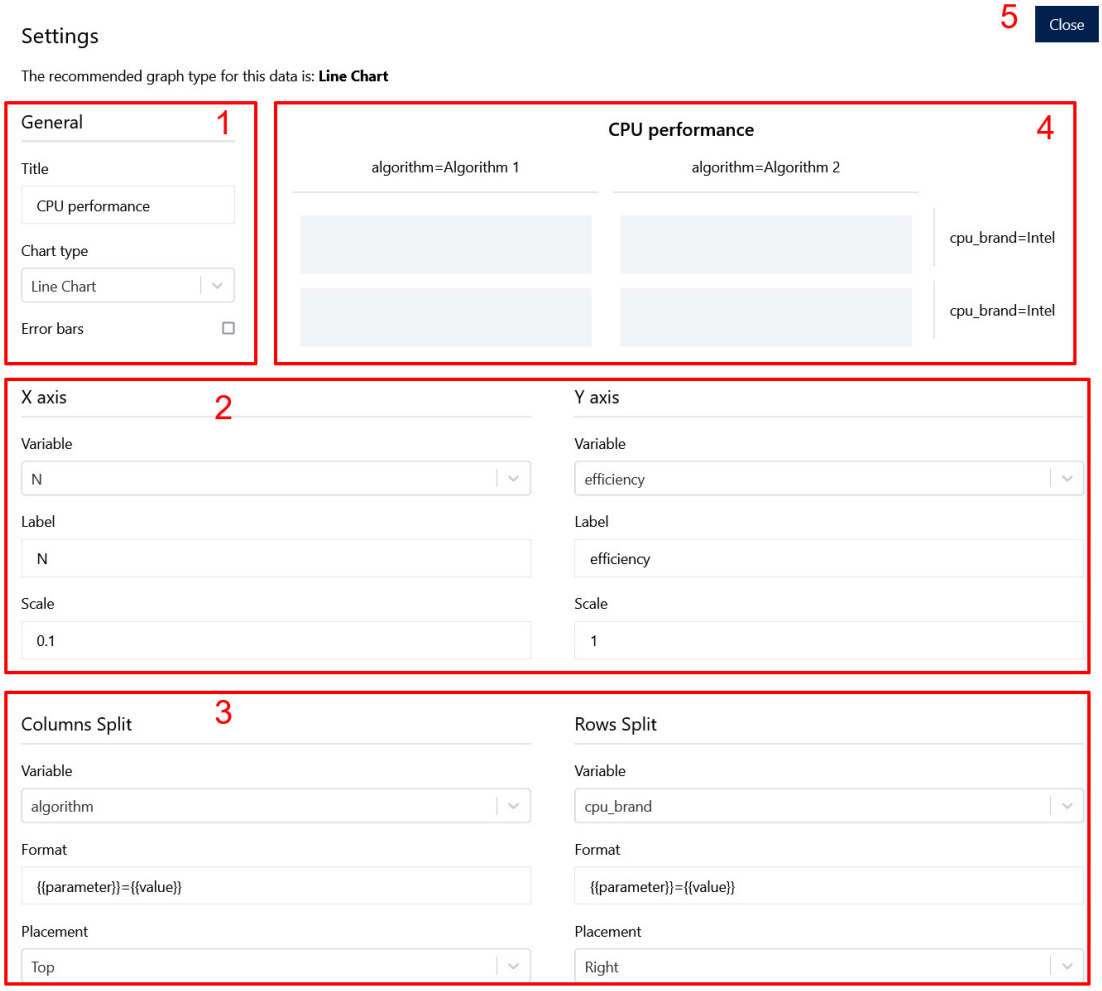


Figure 6.2: Illustration of the settings interface revealing three sections. Each one is described below.

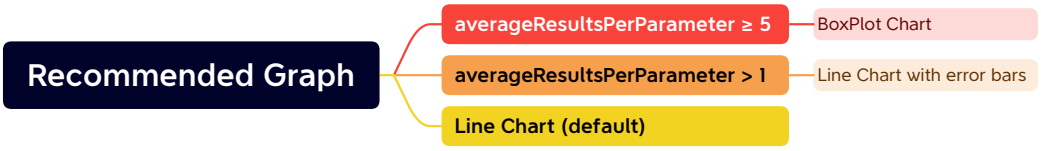


Figure 6.3: Illustration of the algorithm of the recommended graph feature

Out of the box, there are 4 chart types supported:

- **Line chart**
- **Bar chart**
- **BoxPlot chart**
- **Pie chart**

Although there is a recommended chart type, which is also shown at the top of the Settings window, all types are available for selection and it is up to the end user to choose the type that fits best their use.

Error bars

This feature allows displaying error bars on line and bar charts. It is selected by default when the recommended graph feature detects that there is more than one result for each entry of the x-axis.

6.2.2 X and Y axis

These two sections allow the user to change the labels of both axes for all graphs, the variables used for each axis, and the scaling factor of each axis. They are only displayed for line, bar and boxplot charts as it is irrelevant for pie charts. By default, as mentioned above, all the data of a provided configuration is divided into two categories: **parameters** and **measurements**. The horizontal axis allows only the selection of **parameters** whereas the vertical axis allows only to select **measurement** variables.

Inside the application, the x-axis variable is referred to as the **main parameter** and set to the first parameter of the configuration, in the absence of any settings. The y-axis variable is also set to the first measurement variable by default. Labels are initially set to the respective names of these variables.

6.2.3 Chart splitting

As described in the state of the art, having too many parameters directly impacts a graph's readability. Therefore, when more than two parameters are provided in the dataset, the application automatically splits the graph into several sub-graphs.

Variable

The preset settings select the third and fourth parameters provided as the split variables for respectively the columns and rows. Still, this may not suit everyone's needs and this input allows the user to tweak them. To avoid too many graphs

being rendered and an unreasonable amount of columns or rows, the list of variables is filtered. Indeed, a parameter can only be selected either as a column split or as a row split. Furthermore, for each parameter, a list of changing values is kept and if there are more than 6 different values possible (which implies more than 6 columns or rows), the parameter is not available for selection. Both columns and rows split parameters can be set to **None** to disable the feature.

Format

The label for each split column or row can be customized to fit the user's needs. It is initially set up as a pair of `{{parameter}}={{value}}` for each column or row, sorted by their order of appearance in the dataset. These meta-variables are also available for the end user. This allows them to inject them into the input and specify a customized format. Since these metavariables are searched for and replaced with regular expressions, the following aliases are also allowed:

- **parameter**: param, p, 0
- **variable**: var, v, 1

Typically, chart panelling should fit most users' needs. However, it can be disabled by setting both columns and rows split parameters to **None**. In this case, the whole dataset will be rendered into a single chart.

Position

By default, the split columns and rows are shown respectively on the top and right sides of all graphs. This can be changed to respectively top or bottom and right or left. Alternatively, these headers can be removed by specifying an empty string in the format.

6.2.4 Preview

Settings are automatically applied and saved each time an input is edited. However, the settings window covers a majority of the viewport, hiding the graphs. This is why this preview is necessary and useful for the user to see how the overall charts are split and rendered, in real time.

6.2.5 Buttons

The settings window can be closed either by clicking on the darkened background or by clicking this close button.

6.3 Demonstration

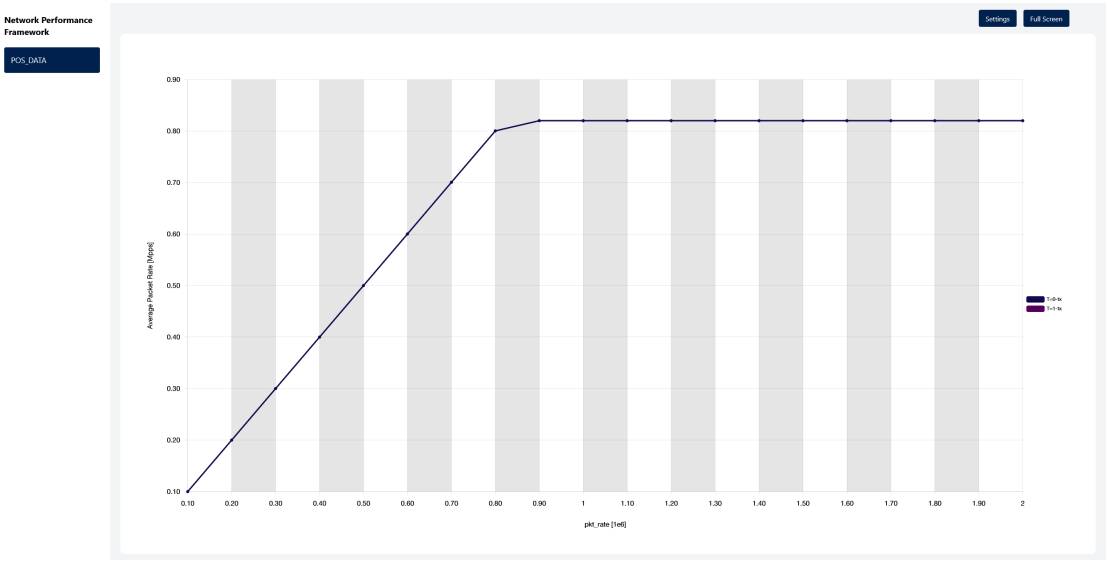
Example datasets are included in each exported file. They can be explored at the following address: `http://npf.info.ucl.ac.be/demo.html`.

6.4 Responsive design

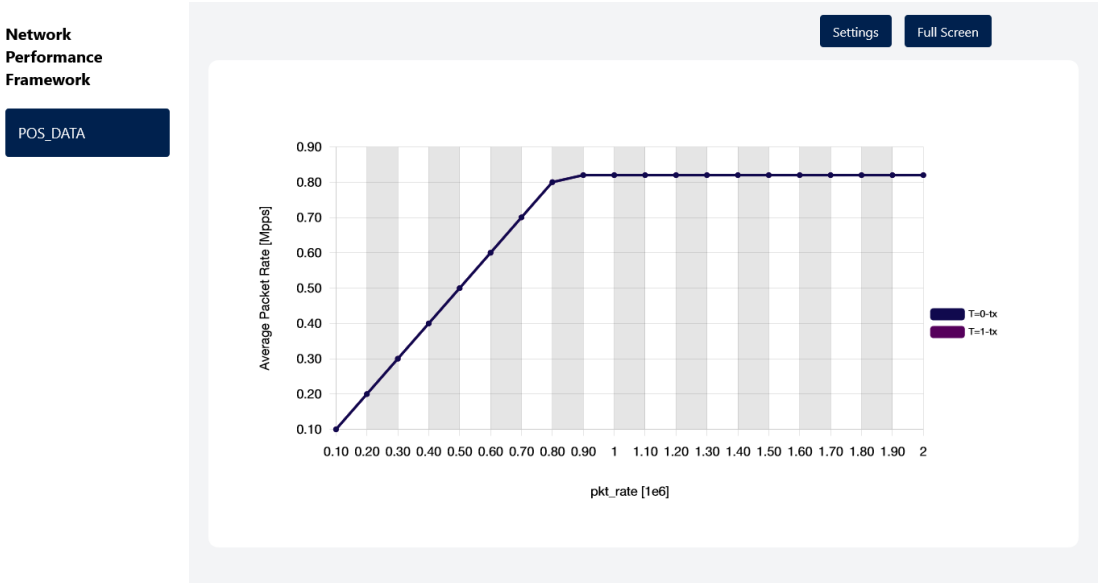
Responsive design is the art of designing a website or software in such a way that it can dynamically adapt its rendering to fit different screen sizes. Screen sizes are typically divided into three categories: mobile, tablet and desktop. The standard usage of the application is not intended for mobile compatibility for two reasons:

- Experiments are run normally on a computer and therefore we expect a majority of users to be on desktop platforms.
- The interface requires enough space to interact with the settings and visualise the preview. This can be achieved on some tablets since screens are wider but mobile screens would require too much work for potentially unsatisfying results.

Therefore some efforts have been put to allow the application to be used on tablets. Tailwindcss has 5 breakpoints by default which eases the process of adapting the website to different resolutions. Crosschecking the data with statistics on most common desktop screen sizes [32] and tablet screen sizes [33] yields that more than 50% of tablet resolutions have a width greater than 768px. As a result, only screens wider than 768px are taken into account. Figure 6.4 shows the user interface for three different resolutions. As we can see, the overall design is identical. The only thing that changes is the font size and spacing around the components.



(a) Desktop 1920x1024



(b) Desktop/Tablet 1280x720

(c) Tablet 1024x768

Figure 6.4: Comparison of the interface between three different resolutions

Chapter 7

Validation

In this chapter, we will go through our different steps in making sure that our solution is validated. We will introduce our automatic testing suite and then gather some feedback from real users. After this, we will provide several comparisons with some tools that have been described in the state-of-the-art section.

7.1 Automatic testing

There are two parts to test regarding the code source: the react code designed to render the web app and the internal methods used to process the data. The former can be tested through many different frontend testing libraries such as Jest [34], Cypress [35] or Mocha [36]. However, the application architecture is relatively small and it is designed to be used directly by researchers. Furthermore, our development process was iterative and implied many internal testing iterations over the interface. As a result, implementing an exhaustive test suite over the interface would be overdoing. Instead, we focused on creating a complete test suite over all our methods used within the app. We used Jest because it's the default unit testing library shipped with React. Figure 7.1 summarizes the coverage with all 9 suites comprising 64 tests in total [1]. We achieved an overall score of 100% code coverage except for the **components/settings/utils.test.ts** suite.

[1] As of the date of writing this document, the latest release is version 0.6.0. Versions released after this one might have different results.

Test suite (9 suites, 64 tests)				
	% Statements	% Branch	% Funcs	% Lines
components/settings/utils.test.ts	100	97.87	100	100
utils/chart.test.ts	100	100	100	100
utils/misc.test.ts	100	100	100	100
utils/configuration/data_ analyzer.test.ts	100	100	100	100
utils/configuration/parser.test.ts	100	100	100	100
utils/configuration/utils.test.ts	100	100	100	100
utils/configuration/parser/ boxplot.test.ts	100	100	100	100
utils/configuration/parser/line. test.ts	100	100	100	100
utils/configuration/parser/ doughnut.test.ts	100	100	100	100
	100	99.76	100	100

Figure 7.1: Code coverage report of the 9 test suites

7.1.1 Browser compatibility

The software has been tested on Chrome (versions 112, 111 and 110), Firefox (versions 112, 111 and 110) and Safari (versions 16 and 15) with the demonstration datasets and no issue has been raised. These versions have been chosen to cover at least 90% of the user base of each browser, based on the statistics provided by W3Schools [24].

7.2 User testing

7.2.1 Christophe Crochet (UCLouvain, Belgium)

Christophe Crochet[2], a Research Assistant currently pursuing a PhD. at UCLouvain, made valuable contributions to our project, in the form of rich feedback. As part of his research study, he utilized our npf-web-extension to facilitate a CSV generation flow. Additionally, he leveraged our tool alongside another tool for data visualization purposes. Christophe Crochet's engagement exemplifies the diverse applications of our software in real-world research scenarios [37].

Pros	Cons
Once the tool is understood, its ease of use is satisfying.	The documentation is not complete and it does not provide information about the format of the data nor the necessary conditions to produce a graph, which led to blank graphs with no explanation.
The graphs generated are great: clear, precise and interactive.	It was sometimes difficult to select the proper measurement columns. It would have been great if it [could infer it automatically and explain better how to configure the x and y axes.
The user interface is simple but efficient.	We were not able to change the settings of our graphs inside the Python package. It would be interesting to provide a way to set them at the export step.
It is extremely simple to produce graphs, even for large datasets.	Axes should by default be named after the main variable being represented.
Integrating it inside our workflow was a piece of cake.	

Table 7.1: Feedback from C. Crochet regarding his use of our software

[2]christophe.crochet@uclouvain.be, UCLouvain, Pl. de l'Université 1 Louvain-la-Neuve 1348, Belgium

7.2.2 Loic Grumiaux (Polytechnique Montreal, Canada)

Loic Grumiaux[3] is a machine learning intern at Airudi Inc. pursuing a double degree at Polytechnique Montreal and UCLouvain. He used our software in the context of his research in the MlOps field. His feedback is the following:

> "I do data analysis regularly and I need to work with other data scientists working on the same datasets. We lack a good data visualisation tool to quickly compare our work and see the differences between our results. Therefore, my problem is that I do not have a clear overview of what has been done so far because we do not know how to seamlessly organize and store our graphs. This software seems promising and I managed to quickly prototype an integration inside our MlOps workflow. I will investigate further integration."

7.2.3 Félix Gaudin (UCLouvain, Belgium)

Félix Gaudin[4] is a master's student at UCLouvain. His research for his master's thesis is centred around measuring the performance of the internet and evaluating different factors that lead to more latency. Felix's engagement with our software was focused on providing feedback rather than utilizing it for his specific use case. He assessed the graphs generated by our tool, comparing them to his graphs.

Pros	Cons
The generated graphs are coherent with the graphs I made with Matplotlib.	The app renders after each keystroke, this could cause performance issues.
The interface is interactive and intuitive.	The fullscreen feature seems a little bit useless because the navigation interface is small.
The software was fast and worked even with a large number of parameters.	The software is missing a heatmap type for the charts.
It is a great tool to explore datasets when a programming environment is not available.	The ticks of the axes should be smoothed when too many values are available.

Table 7.2: Feedback from F. Gaudin

[3] loic.grumiaux@polymtl.ca, Department of Computer Engineering, Polytechnique Montréal, Canada

[4] felix.gaudin@student.uclouvain.be, UCLouvain, Pl. de l'Université 1 Louvain-la-Neuve 1348, Belgium

7.2.4 Samy Bettaieb (UCLouvain, Belgium)

Samy Bettaieb[5] is a master's student at UCLouvain. He is working on a malware analysis toolchain, developing exploration algorithms that lead to graphs representing the dependencies between the syscalls inside the malware. Npf-web-extension has been used as an alternative way of generating those graphs and validating our software in his use case.

Pros	Cons
The interface is easy to use.	String values are not handled properly when generating the graphs.
Being able to switch between different graph types to visualise the data is a time-saver.	We need a feature to lock a variable and generate the graphs based on the value of that variable.

Table 7.3: Feedback from S. Bettaieb

7.2.5 Key takeaways

Overall, the feedback received on the tool was a mix of positive aspects and areas for improvement. On the positive side, once users grasped the tool's functionality, they found it remarkably easy to use and satisfying. The generated graphs were particularly commendable, characterized by their coherence and interactive features. The simplicity and efficiency of the user interface were also appreciated, allowing users to effortlessly produce graphs, even when dealing with large datasets. Additionally, integrating the tool into existing workflows proved to be a seamless process.

However, some drawbacks were identified. The documentation was found to be incomplete. This resulted in instances where users encountered blank graphs without any explanation. Furthermore, there were difficulties in selecting the appropriate measurement columns. Another limitation was the inability to adjust graph settings within the Python package, highlighting the need for a means to modify settings during the export step. Some unexpected behaviours were also discovered during this validation process and are set up on our roadmap. Overall, these constructive criticisms provide valuable insights for enhancing the tool's functionality and user experience.

[5]samy.bettaieb@student.uclouvain.be, UCLouvain, Pl. de l'Université 1 Louvain-la-Neuve 1348, Belgium

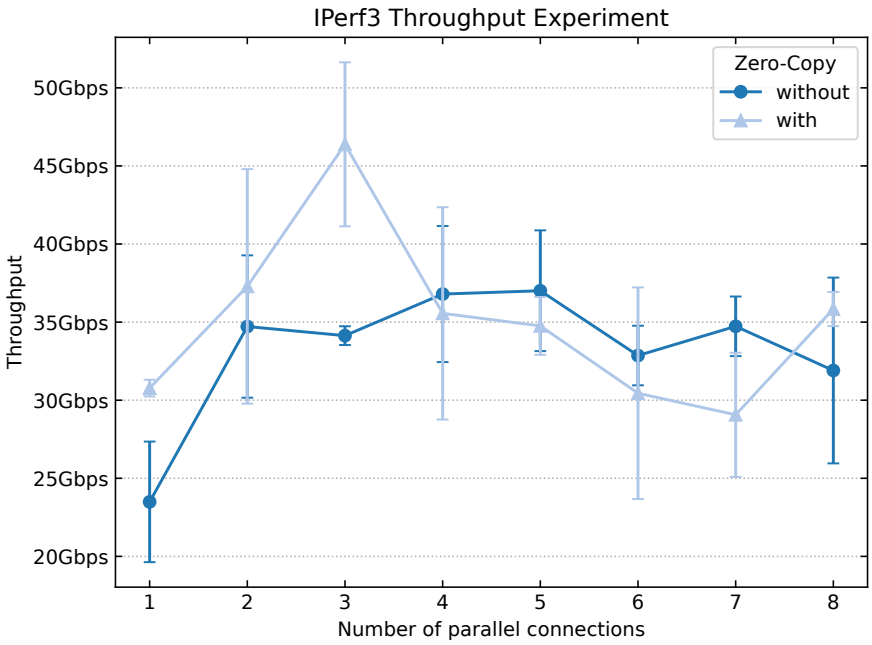
7.3 Comparison with other tools

7.3.1 NPF vs NPF-web-extension

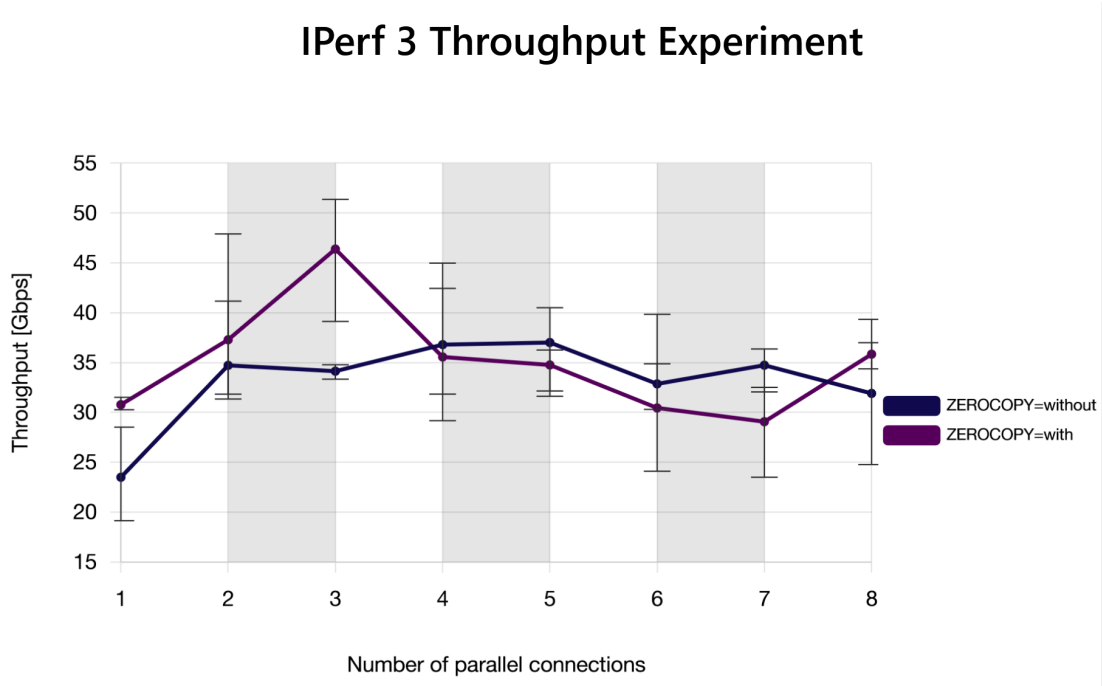
Figure 7.2 below compares a dataset generated via the IPerf 3 example tests available with NPF and exported as graphs visualisations. The first one has been made with the available data visualisation within NPF whilst the other one is the result of our extension.

Although the overall data visualisation remains the same, the main difference lies in the format of the graph being generated. NPF relies upon the Matplotlib library to generate graphs in PDF format which allows them to scale with the document they are integrated to. We are looking forward to releasing a version of our tool which allows also for quick exportation features, including PDF and SVG. This feature is further described in chapter 9.

Additionally, the NPF version looks slightly more polished. For instance, the legend is placed inside the graph with NPF while our version places it on the right side of the graph. The title is also smaller on the NPF version compared to ours, and units are placed along each tick of the graph. Instead, our version places the units along with the axis label.



(a) Figure generated with npf



(b) Figure generated with npf-web-extension

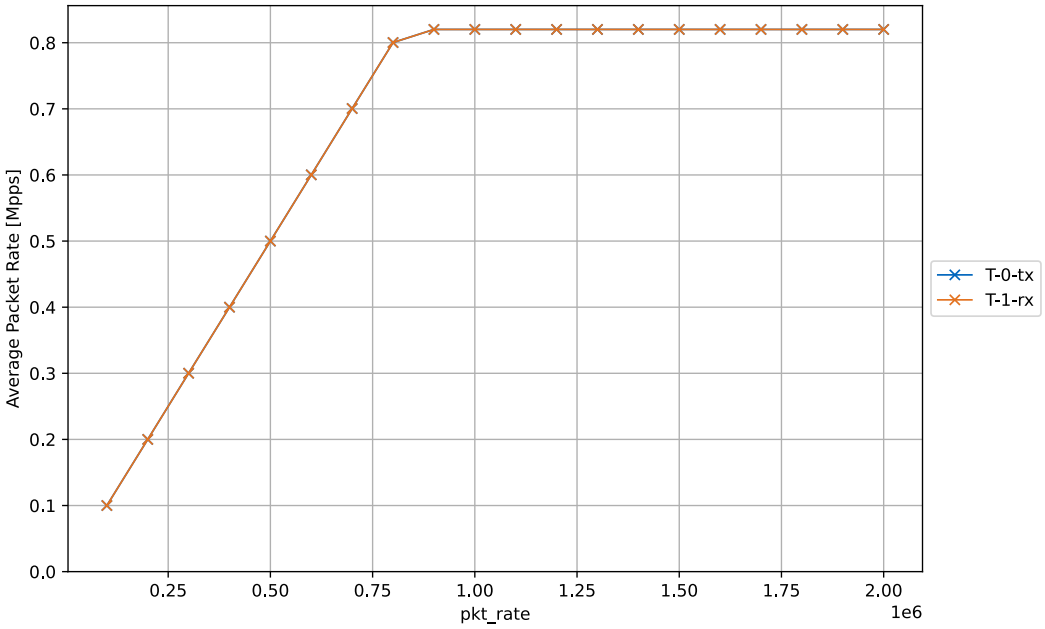
Figure 7.2: Comparison of two figures generated with the same dataset but one with NPF and the other one with npf-web-extension

7.3.2 POS vs NPF

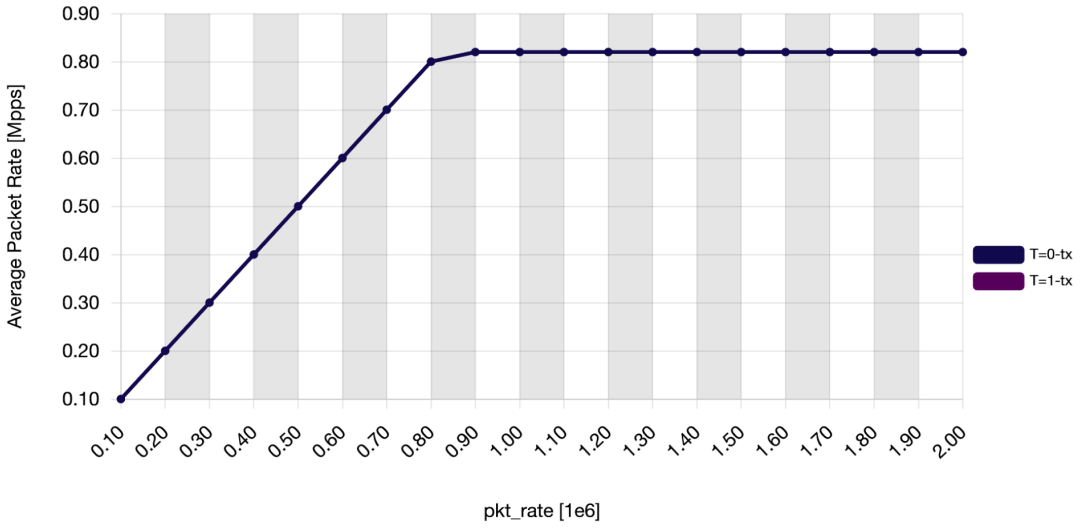
We can see in figure 7.3 that the graph generated with our tool is almost a carbon copy of the one made with POS. The advantage of using our tool lies in the variety of visualisations available for this dataset (for instance, see figure 7.3c which is an alternative visualisation of the same dataset). Compared to POS, we do not require any further reconfiguration of the initial setup. The settings available inside the app allow one to quickly change the type of the graph.

7.4 Visual comparison

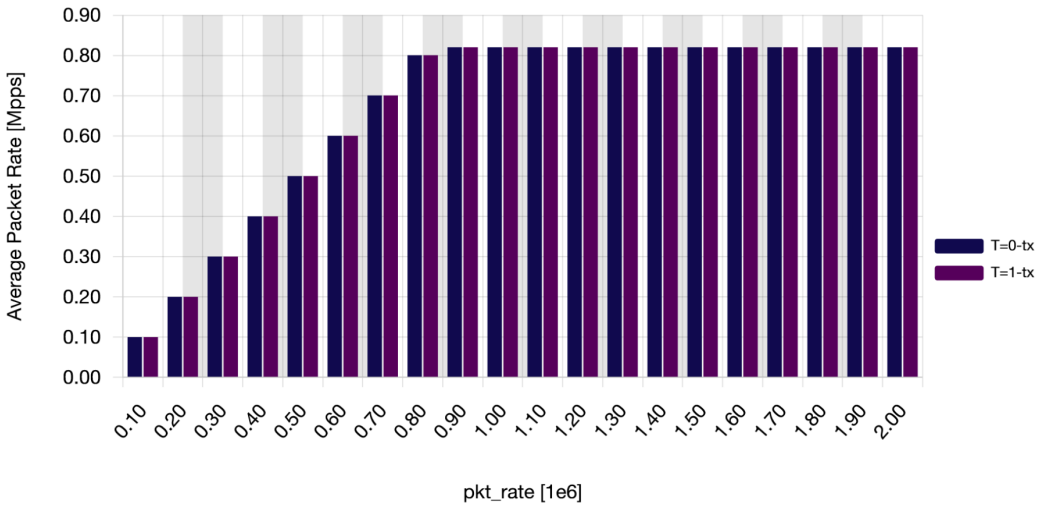
Our solution is working and we were able to reproduce the graphs generated with some of the other tools like 'original' NPF and POS. Since NPF and POS both use Matplotlib as their main graph generation library, they have a similar look and appear slightly more polished. Nevertheless, we were able to generate our versions of these graphs with no effort while still maintaining the overall graph quality. This is a significant accomplishment as we could iterate through different versions of these graphs to fit the style of the graphs to reproduce without involving any programming skills. On top of that, our versions include a striped background to make the graphs easier to read.



(a) Figure generated with pos



(b) Figure generated with npf-web-extension



(c) Alternative visualisation of the figure generated previously with npf-web-extension

Figure 7.3: Comparison of three figures generated with the same dataset but one with pos and the other ones with npf-web-extension

Chapter 8

Future work

We will now direct our attention to what was achieved and assess whether the initial requirements have been met. After this, we will introduce a few ideas on how to enhance the existing features. This section is based on release **0.6.0**, the following statements might not be relevant for more recent releases.

8.1 Requirements

To refresh our memory, in chapter 4, we stated that the software should be easy to share, self-sufficient and visually attractive.

8.1.1 Easy to share

The app encapsulates all data within a single HTML file, facilitating effortless sharing due to its consolidated nature. The feedback in Chapter 7 pointed out that there is no way to export some data with predefined settings. Therefore, this feature is part of our next milestone.

8.1.2 Self-sufficient

Our efforts have been put into providing a fully automated build-and-release workflow. As a result, npf-web-extension does not require any external configuration or tools and can be used right after being installed. Furthermore, the package is shipped with the source code of the website which allows the user to rebuild the app themselves and customize it.

8.1.3 Visually attractive

We have used a modern CSS framework to start on good foundations for the overall app design. However, this point is still a matter of personal preference. Based on the feedback gathered in the previous chapter, we can consider that this requirement is met.

8.2 Future work

This section explores a series of features that would add a new dimension to our software. At the moment of writing, we are already working on implementing some of them. However, we will outline them hereunder regardless of their current implementation status.

8.2.1 Artificial intelligence

The algorithm behind the recommended chart type for a given dataset is limited. With the rising trend towards artificial intelligence over the last decade, it would be interesting to integrate some form of classification based on the number of parameters, measurements and data values.

8.2.2 Further customisation

Chart types

Up to this point, only four types of charts are available inside the app. More types have been described earlier, amongst which the most interesting ones are the tree and heat maps because they would allow encompassing a whole new set of usages such as data forensics or funnels. Furthermore, all of these charts are two-dimensional and three-dimensional diagrams could provide more visualisation capabilities. However, to handle 3D charts, additional research should be done to find an alternative to ChartJS which is currently not able to handle 3D.

Second parameter

Although there is a feature to select the main parameter for each axis, a second parameter is always chosen to distinguish the datasets inside a single chart. This parameter can not be changed in the current version of the software and it defaults to the first available parameter other than the main parameter. This behaviour is not optimal and a feature should be considered to change this.

Styles

A considerable amount of work has been put into providing enough customisation features regarding data manipulation but there are no features available to change the current styles. Interesting and useful features may include:

- Different font styles, including font size and font family.
- More colour palettes, potentially with solutions for colourblind people.
- Background and overall colour settings for the chart title, labels, etc.
- More line stroke types, for instance, dotted or dashed.

Graph customisation through settings override

In the future development of our tool and its accompanying Python wrapper, a significant focus will be placed on refining the functionality of the 'export' method. As part of the ongoing polishing process for our application, it is crucial to implement a mechanism that empowers users to predefine and customize the settings of the generated graphs. By allowing users to specify graph settings beforehand, such as axis labels, colours, and plot types, we can provide greater flexibility and control, enabling them to tailor the visual representation of their data to their specific requirements. This enhanced graph customization feature will empower users with the ability to override default settings and achieve more personalized and meaningful visualizations, thereby enhancing the overall user experience of our tool.

Format options and settings

The format feature only allows two meta parameters: `parameter` which holds the parameter name and `value`, which holds the value of the parameter after each iteration. Additionally, this feature could be extended by exposing more values such as the **number of values** and the **number of parameters**. Furthermore, it could also be extended with some ways to manipulate the rendering of the injected value. For instance, it would be interesting to allow the user to capitalize or lowerize the meta parameter. On another level, some extra settings could be brought to the end-user:

- A toggle between a logarithmic scale and a linear scale for both axis to improve the rendering of exponentially distributed values.
- A toggle between ascending and descending order for columns and rows when splitting or even a way to allow for custom reordering.

- Inputs to specify min and max values for each parameter. Useful when working with outliers in the datasets.
- Buttons to easily swap variables between axes. This would allow to render for instance horizontal bar charts instead of vertical ones.

8.2.3 Exports

Currently, there is no way to export the graphs other than using a browser add-on. We propose two different ways to enhance this feature.

Export to image

It would be interesting to generate an image in some common formats such as PNG, JPEG or SVG. However, the current implementation might need some changes because the main component renders both HTML and canvas elements mixed up together. It would most likely be necessary to rethink how the rendering component works and generate everything inside a single canvas component.

Integration with LaTeX environments

At the moment, the tool can be seen as a data visualisation tool but we envision something greater. We would like to see a feature to export the graph to a TikZ figure and be able to use it directly in a LaTeX document. This would make the tool more valuable to the researcher and improve the quality of the graphs displayed inside the documents and at the same time ease the process of going from raw data to a high-quality chart rendering. The main advantage of developing this integration lies in the ability to alter the internal data of the graph, due to the TikZ format.

8.2.4 Further optimisation

Although there are no performance issues to mention regarding the app since we are far from reaching the limits of chartjs, we could expect to achieve better performance when working with very large datasets by switching to WebGL chart rendering libraries. Indeed, Horak et al. (2018) [38] found that WebGL performs better than plain Canvas for web-based graph drawing.

8.2.5 Further testing

The app is relatively small with an interface easily testable through human interaction. However, as features keep getting implemented, it would eventually

become necessary to implement a proper end-to-end testing suite to assess that all react components behave correctly. There are tools available for this purpose. For instance, our GitHub automation could be extended with the use of Cypress or Playwright.

8.2.6 Online SaaS

The current software can be easily deployed as an online software. NextJS [39] would probably be the easiest wrapper to implement since it has been developed to use React server-side. Some interesting features to develop along with an online version of the software:

- A storage system to save each configuration online instead of locally and thus allow one to access the datasets from anywhere.
- An upload feature where users could upload their local versions of the software and host them online for better shareability.

8.2.7 Github io support

Given that our solution produces a singular HTML file upon the completion of an experiment, accessibility to this file might be hindered for end users in situations where the experiment is conducted via an SSH connection. Notably, the GitHub platform inherently facilitates file uploads through its GitHub io platform. In such scenarios, researchers would greatly benefit from utilizing NPF alongside a command-line argument, enabling them to seamlessly upload the experiment results directly to this platform.

8.3 Threats to validity

The test suites consist of structural and functional testing. In other words, we based our tests on the expected behaviour of our internal app and explored only the set of branches returned by our testing framework. However, no work has been conducted regarding mutation testing. Moreover, the software encompasses only five use cases. While the overall feedback is positive, this limited feedback set may not adequately represent the diverse range of potential use cases for our software. Although various bugs have been discovered and fixed during our validation process, more bugs may remain undiscovered since our automated test suite does not encompass end-to-end testing or mutation testing.

Chapter 9

Conclusion

In the first place, we developed NPF to fit our experimental design workflow. Our tool can automatically run experiments, collect the data and store them in a convenient format such as pandas or CSV. Additionally, we needed a tool to visualise the data collected after our experiments. NPF is already shipped with some data visualisation features, notably graph generation with Matplotlib. However, we wanted a tool that would allow us to interact with the data and quickly sketch some graphs to analyze and understand the data. We could not find such a tool. Indeed, most tools require a license (Tableau, Microsoft BI, ...) or a backend architecture (Grafana) and integrating them with our NPF workflow ended up being inconvenient. Therefore we developed our own extension to NPF for this sole purpose.

On one hand, we wanted our tool to be part of the NPF experience and thus result in an all-in-one tool. On the other hand, we also envisioned our tool to be used out-of-the-box and integrated with other workflows, which is why we prefer the 'extension to npf' approach. We developed a tool that makes life easier for running experiments. We also wanted to reach a broader audience who might already have the data but no visualisation tool like ours.

We imagined a tool that would benefit from the latest advancements in web technology and this has been achieved with the use of the react framework. Furthermore, we also imagined a tool that would require only a minimal setup. This has also been achieved, with our efforts put into DevOps with a Python wrapper directly published to PyPi after each release, making it easy to integrate with virtually any workflow. We imagined a tool that would be easy to use and share. This was achieved by encapsulating our whole app into a single HTML that can be run inside any modern browser.

At present, our primary focus is on refining our solution and enabling graph export in formats compatible with external environments, such as PNG or SVG. Further enhancements could entail delivering our software as an online service to improve the overall user experience or introducing additional features to enhance the extension's capabilities. Nonetheless, we have gathered valuable feedback that validates our solution, affirming its effectiveness in addressing the initial data visualization problem and meeting our initial requirements.
Ultimately, although our solution may not possess the same level of refinement and an extensive range of features as other software options, it presents an excellent **trade-off between ease of use and functionality**.

Bibliography

[1] Sebastian Gallenmüller, Dominik Scholz, Henning Stubbe, and Georg Carle. The pos framework: A methodology and toolchain for reproducible network experiments. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '21, page 259–266, New York, NY, USA, 2021. Association for Computing Machinery.

[2] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, page 275–287, New York, NY, USA, 2015. Association for Computing Machinery.

[3] Jon Dugan, Seth Elliott, Bruce A Mah, Jeff Poskanzer, and Kaustubh Prabhu. Iperf3: The tcp/udp bandwidth measurement tool. `https://iperf.fr`, 2005. Accessed: May 1, 2023.

[4] Tom Barbette. Github - fastclick. `https://github.com/tbarbette/fastclick`, 2015. Accessed: May 1, 2023.

[5] TRex Team. Trex. `https://trex-tgn.cisco.com/`, 2021. Accessed: May 1, 2023.

[6] Gianni Antichi, Muhammad Shahbaz, Yilong Geng, Noa Zilberman, Adam Covington, Marc Bruyere, Nick McKeown, Nick Feamster, Bob Felderman, Michaela Blott, et al. Osnt: Open source network tester. *IEEE Network*, 28(5):6–12, 2014.

[7] Alina Quereilhac, Mathieu Lacage, Claudio Freire, Thierry Turletti, and Walid Dabbous. Nepi: An integration framework for network experimentation. In *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, pages 1–5, 2011.

[8] Thierry Rakotoarivelo, Guillaume Jourjon, and Max Ott. Designing and orchestrating reproducible experiments on federated networking testbeds. *Computer*

Networks, 63:173–187, 2014. Special issue on Future Internet Testbeds - Part II.

[9] nteract team. Papermill. https://papermill.readthedocs.io/en/latest/index.html, 2019. Accessed: May 20, 2023.

[10] Tom Barbette. Network performance framework (NPF). https://github.com/tbarbette/npf, 2020. Accessed: May 7, 2023.

[11] Claus O. Wilke. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly Media, 2019.

[12] Stephen R. Midway. Principles of effective data visualization. *Patterns*, 1(9):100141, 2020.

[13] Google Cloud. How to choose the best chart or graph for your data. https://cloud.google.com/blog/products/data-analytics/different-types-graphs-charts-uses, July 2019. Accessed: May 20, 2023.

[14] William S. Cleveland. *The elements of graphing data*. Wadsworth Publ. Co., 1985.

[15] Grafana Labs. Grafana. https://grafana.com/, 2021. Accessed: May 2, 2023.

[16] Tableau Software. Tableau. https://www.tableau.com/, 2021. Accessed: May 2, 2023.

[17] Microsoft. Power BI. https://powerbi.microsoft.com/, 2021. Accessed: May 20, 2023.

[18] Google. Looker Studio. https://datastudio.google.com/, 2016. Accessed: May 20, 2023.

[19] Plotly Technologies Inc. Plotly. https://plotly.com/. Accessed: May 20, 2023.

[20] Lionel Sujay Vailshery. Most used web frameworks among developers worldwide, as of 2022. https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/, 2022. Accessed: March 26, 2023.

[21] Tanner Linsley. Chartjs performance test (codepen). https://codesandbox.io/s/github/tannerlinsley/react-charts/tree/beta/examples/simple, 2020. Accessed: December 10, 2022.

[22] Tailwind Labs. Tailwindcss official website. https://tailwindcss.com/, 2018. Accessed: March 26, 2023.

[23] Microsoft. Typescript official website. https://www.typescriptlang.org/, 2012. Accessed: March 26, 2023.

[24] W3Schools. Browsers statistics. https://www.w3schools.com/browsers/, 2023. Accessed: May 1, 2023.

[25] Wikipedia, The Free Encyclopedia. Comparison of browser engines. https://en.wikipedia.org/wiki/Comparison_of_browser_engines, 2023. Accessed: May 1, 2023.

[26] Meta. Create react app. https://create-react-app.dev/, 2013. Accessed: May 1, 2023.

[27] Formidable Labs. Github - create react app configuration override (CRACO). https://github.com/gsoft-inc/craco, 2018. Accessed: May 1, 2023.

[28] Eric Schoffstall. Gulp. https://gulpjs.com/, 2013. Accessed: May 1, 2023.

[29] Filip Malinowski. Github - gulp-inline-source. https://github.com/fmal/gulp-inline-source, 2014. Accessed: May 1, 2023.

[30] Python Software Foundation. Pypi - the python package index. https://pypi.org/, 2003. Accessed: May 2, 2023.

[31] Anaconda, Inc. Anaconda. https://www.anaconda.com/, 2012. Accessed: May 2, 2023.

[32] StatCounter Global Stats. Browser desktop display statistics. https://gs.statcounter.com/screen-resolution-stats/desktop/worldwide, 2023. Accessed: May 2, 2023.

[33] StatCounter Global Stats. Browser tablet display statistics. https://gs.statcounter.com/screen-resolution-stats/tablet/worldwide, 2023. Accessed: May 2, 2023.

[34] Meta. React testing with jest. https://jestjs.io/docs/tutorial-react, 2021. Accessed: May 7, 2023.

[35] Cypress. Cypress documentation. https://docs.cypress.io/, 2017. Accessed: May 7, 2023.

[36] Mocha. Mocha documentation. https://mochajs.org/, 2012. Accessed: May 7, 2023.

[37] Bertrand Van Ouytsel, Christophe Crochet, and Axel Legay. Tool paper - sema: Symbolic execution toolchain for malware analysis. In *17th International Conference on Risks and Security of Internet and Systems*, 2022.

[38] Jakob Horak, Christoph Buchheim, Tobias Gleßner, and Michael Jünger. Graph drawing performance on the web: Webgl vs. svg. In *International Symposium on Graph Drawing and Network Visualization*, pages 186–198. Springer, 2018.

[39] Vercel. Next.js. `https://nextjs.org/`, 2016. Accessed: May 7, 2023.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**