MEX: System For Managing Machine Learning Experiment Results

Shivay Sharma

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

This report explores the development of MEX, an easy-to-install and use system designed to manage machine learning experiments and the associated data. MEX streamlines the process by incorporating features such as experiment tracking, note-taking, result plotting, LaTeX table generation and saving input in a database. With three accessible themes, MEX aims to be user-friendly for a wide audience. Two user studies, conducted through questionnaires, evaluated MEX's performance in both its initial and final versions. Statistical tests assessed improvements, with results showing that participants universally appreciated the final version's aesthetic and intuitive interface. 100 % agreed that the features offered by MEX are useful and 90 % agreed that MEX has an intuitive design and they did not have to refer the documentation to have an understanding of its working. SUS, Welch's t-test and NPS scores were calculated for statistical analysis to confirm the user studies findings and showed that the improvements made to the system are relevant.

Ethics approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: 409739
Date when approval was obtained: 2024-06-24
The participants' information sheet and a consent form are included in the appendix at appendices B.4 and B.5.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Shivay Sharma*)

Acknowledgements

I would like to express my deepest gratitude to all those who supported and guided, The time has been challenging and the constant support I received is what helped me throughout the completion of MEX.

First and foremost, I am profoundly grateful to my supervisor, Brian Mitchell, for their invaluable guidance, constructive feedback, and continuous encouragement throughout the duration of this project. Their insightful guidance and constructive feedback throughout this research process not only enhanced the quality of my work but also significantly shaped my academic and professional growth. I am grateful for their patience, mentorship, and dedication, which have been a constant source of motivation and inspiration.

I would also like to extend my appreciation to the faculty and staff of the School of Informatics at the University of Edinburgh for providing a stimulating and supportive academic environment. The resources and facilities made available to me were crucial in the successful completion of this work.

Finally, I would also like to extend my heartfelt gratitude to my family, whose love and support have been the cornerstone of my journey. My parents have been my pillars of strength, providing me with the emotional and moral support needed to persevere through the challenges of this research. Their belief in my abilities and their constant encouragement have been invaluable. I am equally thankful to my sister, whose understanding and aid have allowed me to focus entirely on my studies. Their patience and understanding allowed me to focus on my studies and complete this dissertation.

To everyone who contributed to this journey, directly or indirectly, I extend my sincerest thanks.

Tables of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

The ability to conduct, manage, replicate, and accurately report experiments is crucial in machine learning (ML). The management and execution of ML experiments often involve handling large datasets, many permutations, and extensive results tracking. This complexity can quickly lead to disorganisation, inefficiency, reporting wrong sets of results, running unintended experiments, and a confusing workflow — all of which can affect the progress and effectiveness of researchers and practitioners in the field. The reproducibility of experiments is a major concern. Without a systematic way to document the details of each experiment, including hyperparameters, data versions, and code configurations, reproducing experiments can become difficult. This lack of reproducibility can not only impedes progress but could also undermine the credibility and reliability of research findings. There is also no single standardised approach to ML experiment tracking. Each tool has its own approach and learning curve, which may be initially difficult to use.

To address such difficulties, this dissertation presents the development of the MEX system, which is not a machine learning system but a flexible, robust, and intuitive platform designed to streamline the management of machine learning experiments, their data, and their results. This project investigates ways to lower the barrier to entry for beginners and either solve or dilute the difficulties associated with managing multiple experiments by providing a comprehensive solution that emphasises simplicity, configurability, and user-friendliness. By making it easier for beginners, MEX aims to make the process easier for all users including the proficient and experts. Individuals who may not be as adept in

machine learning as an expert might find it difficult to set up an ML tracking system. Thus, MEX is designed to be easy to use, understand, maintain, and install. MEX assists individuals when they run machine learning experiments, especially those who lack experience in using ML tracking systems.

1.2 Problem statement

MEX is designed to address all the following difficulties. They are all addressed differently but the key is adding functionality and logging the data for the user's reference, and giving the user options and discretion.

1. **Barrier to entry:** The system is designed to be easy to use and install. It incorporates different users' needs, thus making it easy for many users.
2. **Understandability:** The system utilises the command line for the majority of the tasks, which simplifies the workflow and the user understands the processes taking place in the system.
3. **Usability:** Many use cases were considered while adding the features to MEX, including but not limited to: generating a LaTeX table of experiment results; the ability to add notes to results; plotting results; running multiple experiments; and a built-in terminal for the user to run additional scripts.
4. **Logging:** All the results are logged and have the necessary information to recreate the experiment, for example the parameter values and the dataset used. Logging also saves the data as an xls file.
5. **Availability:** MEX is available on Windows, macOS, and Linux.

1.3 Research questions

The research milestones can be divided into research questions:

RQ1. What problems or use cases should MEX assist with?

RQ2. What technologies should be used to develop MEX? Programming language, frameworks, development environment, design guidelines.

RQ3. How to make the UI easy to use?

RQ4. What features should be added to the system for usability to cover most of the operations and how will they work?

RQ5. How can MEX be evaluated for efficiency and performance?

RQ6. How does MEX compare with other ML tracking systems?

1.4 MEX workflow

Figure 1.1 shows MEX's workflow, detailed in subsequent chapters. Chapter 2 identifies the market gap; Chapter 3 investigates the design elements; Chapter 4 covers features that have been included or were attempted (also see the Requirements engineering). The user study is covered in Chapter 5. Conclusions reflects on the project overall. Limited experience with this type of project — my background is Electrical and Electronics Engineering — created a steep learning curve requiring substantial project time, but influenced design decisions to help keep MEX beginner-friendly. This meant learning new skills and tools with different stages of development of MEX which could have been common knowledge for Computer Science background but may have not been covered yet due to pursuing a different curriculum. MEX has the features to add notes to already generated outputs, save the parameters used for experiments, generate latex tables, plot previously done experiments, a terminal for extra usage, different colour themes,and a database saving the input history to be reused, which are explained in Chapter 4.

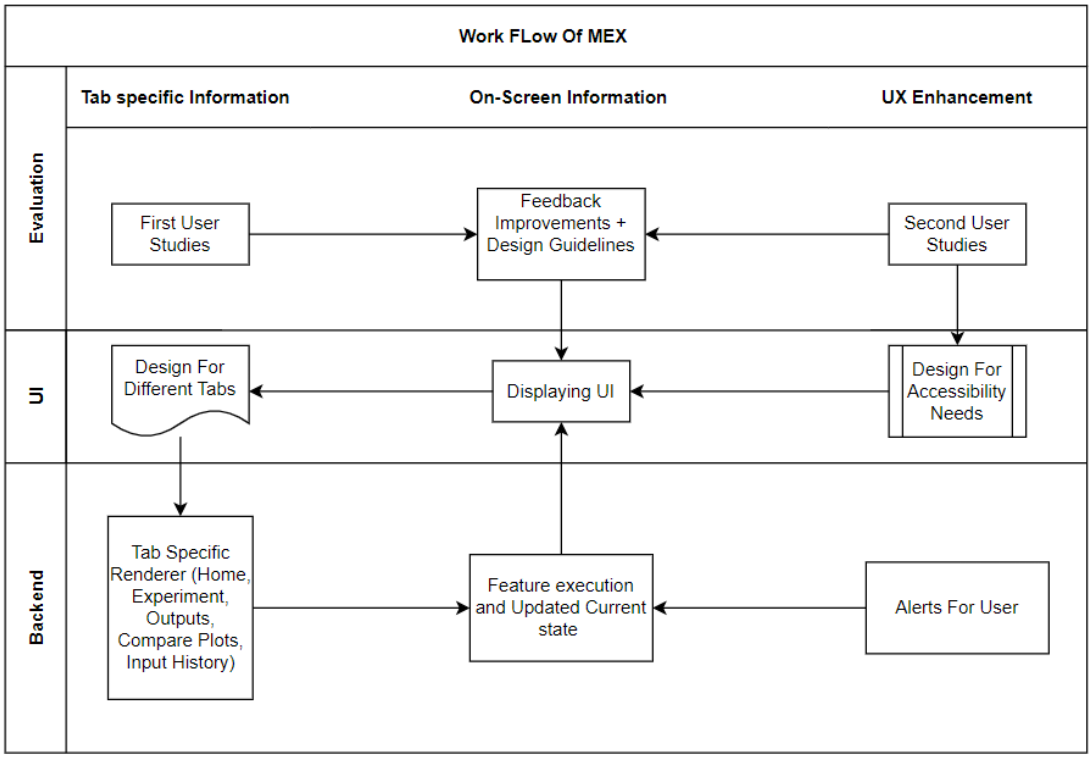


Figure 1.1 Workflow chart for MEX.

Chapter 2

Background

2.1 Existing ML experiment tracking resources

There are many ML experiment-tracking solutions, but no universal standard for essential features or procedures. Eight tools are examined in pairs (Table 2.1) to identify common user difficulties. Despite their power and utility, these tools share similar problems, highlighting a gap in current ML experiment trackers. To highlight the gap, the systems are paired in a way that each comparison between pairs reveals the same shortcomings. The pairs are discussed according to their popularity among the user base. Berger et al. (2015) criticise features often implemented without enhancing user experience — MEX incorporates user-requested features. MEX manages system and generated files for ML experiments, so it has differing motivations and overlaps in operations from the alternatives. Those, while advantageous for managing and optimising ML workflows, pose difficulties in setup, integration, learning curves, and costs, especially for beginners. There is a need for an easy-to-install, user-friendly tool that beginners can understand. MEX addresses these limitations by offering a user-friendly, cost-effective, framework-agnostic solution for tracking, documenting, and visualising ML experiments. Its self-contained nature ensures data privacy and smooth performance, appealing to

Pair 1	**Pair 2**	**Pair 3**	**Pair 4**
MLflow	TensorBoard	Comet.ml	Sacred + Omniboard
Weights & Biases	Neptune.ai	Data Version Control	Guild AI

Table 2.1 ML experiment tracking systems.

both beginners and experienced practitioners. MEX's simple setup, comprehensive visualisation, and efficient handling of large-scale projects make it a viable alternative. Kluge and Jenkner (2024) and Lewinson (2023) are from neptune.ai and an expert user and mention the criteria with which they evaluate competitors, providing valuable insights for the comparison.

2.1.1 MLflow and Weights and Biases (W&B)

MLflow and Weights and Biases (W&B) are both powerful tools. MLflow supports experimentation, reproducibility, and deployment, allowing users to track experiments, package code, and share models. It provides a standardised format for packaging machine learning models for deployment across various platforms. However, MLflow can be complex to set up, especially when integrating with other services, and may require custom scripting for seamless integration. Scalability can also be a concern, with performance bottlenecks and increased latency in projects that log hundreds of experiments daily. Additionally, its visualisation features may not meet advanced needs.

W&B offers extensive tracking and visualisation, but its vast feature set can overwhelm beginners. Setting up a new project requires configuring settings, integrating the logging API, and learning to use the visualisation tools. Integrating W&B into an established ML pipeline might need significant code changes and configurations, including logging calls, handling authentication, and resolving compatibility disparities. New users may struggle with logging different data types and using features like hyperparameter sweeps, leading to a steep learning curve. Additionally, advanced features require a paid subscription, and storing data on external servers can raise data privacy concerns and compliance problems with regulations like GDPR. However, a basic understanding can be gained in about two weeks with proper documentation (*Weights & Biases Documentation*, 2017), making the learning investment worthwhile for powerful visualisation techniques.

MEX is designed to be user-friendly and self-contained, allowing users to track experiments immediately without complex setup or server configurations. It's ideal for beginners and small teams, as it doesn't require paid subscriptions and can store data locally, ensuring data privacy. However, MEX lacks features for sharing trained models and has limited visualization capabilities compared to

tools like W&B. Time constraints and the team's size and experience prevented the inclusion of these features.

2.1.2 TensorBoard and Neptune.ai

TensorBoard and Neptune.ai focus on visualisation and collaboration, respectively, but share drawbacks with MLflow and W&B. TensorBoard's tight integration with TensorFlow limits its convenience for users of other frameworks like PyTorch, which require plugins like 'tensorboardX' and additional setup. PyTorch users might spend significant time configuring logging hooks for TensorBoard compatibility, adding complexity and a steep learning curve. Logging large datasets or thousands of training runs can cause performance bottlenecks such as slow loading times and high latency in visualisations.

Similarly, Neptune.ai requires understanding and implementing API keys, setting up project configurations, and managing dependencies for tracking experiments across multiple environments or cloud services, which can be challenging for new users. Neptune.ai also stores experiment data on its servers, raising potential data privacy and security concerns, and requires a paid subscription for most advanced features, limiting accessibility.

MEX is framework-agnostic, allowing seamless integration with any machine learning framework without extensive modifications. Unlike TensorBoard, it avoids framework dependency and can fit into existing workflows. MEX is free and lightweight, utilising system resources without a paid tier. It can handle frequent logging of high-dimensional datasets if the machine is powerful enough. MEX does not aim to compete with TensorBoard's performance with TensorFlow, recognising TensorBoard's established capabilities in that area.

2.1.3 Comet.ml and Data Version Control (DVC)

Like MLflow, integrating Comet.ml into existing workflows can be complex and time-consuming, particularly for beginners. This involves modifying scripts and understanding extensive documentation, which can be daunting. Setting up advanced features like real-time collaboration and custom dashboards requires additional configuration, increasing the initial setup burden. Although Comet.ml offers a free tier, its advanced features and higher usage limits require a paid subscription much like W&B. Storing experiment data on its servers poses significant

data privacy risks, making it less suitable for projects requiring strict data privacy, such as healthcare startups handling patient data.

DVC primarily relies on command-line operations and lacks a robust graphical user interface (GUI), which is a serious limitation for users uncomfortable with command-line interactions. This reliance makes the learning curve steeper, compounded by complex setup and integration steps similar to other systems.

MEX's design prioritises ease of use, with a straightforward setup process and intuitive interface. Including three accessibility themes, making it particularly suitable for beginners and those with specific accessibility needs. Users can start tracking their experiments with minimal setup, focusing on research rather than tool management.

2.1.4 Sacred + omniboard and Guild AI

Sacred, paired with Omniboard, and Guild AI provide detailed logging and reproducibility. Sacred ensures experiment reproducibility through detailed logs, but integrating it with Omniboard and other tools can be cumbersome. It offers fewer features compared with previous tracking tools. it lacks advanced features such as real-time collaboration, automated hyperparameter optimisation, and extensive visualisation capabilities. Guild AI, despite its comprehensive feature set, lacks a polished user interface and relies heavily on command-line operations, which can be less intuitive. Its smaller user base limits community support, posing additional concern for beginners at the top of the system integration difficulties already faced similar to the previously mentioned system.

2.2 Technologies chosen

Table 2.2 highlights the candidates, and their benefits, considered for the MEX system. The drawbacks and shortcomings of each combination are different and any of them would have been a suitable approach for MEX in some other circumstances. One of the main deciding factor were the prebuilt libraries each of the combinations offers as those would decide the complexity of the functionality being developed. It could be trivial in one setup and taxing in another.

Frontend	Backend	Benefits
C# with WPF or .NET	Java	rich UI capabilities and robust and scalable backend
Java with JavaFx	Golang	mature development environment and offers high performance and efficient concurrency
Python with PyQt or Tkinter	Python	same language ecosystem, simplifying development and maintenance
Electron framework	Javascript	modern and responsive UI and efficiently handles asynchronous operations

Table 2.2 List of candidates for the technologies to be used.

2.2.1 C# with WPF or .NET and Java

Mixing C# and Java can increase project complexity. Communication between the C# frontend and Java backend may require additional configuration and tooling, such as setting up REST APIs. Both .NET and JVM are resource-intensive, potentially leading to higher memory and CPU usage. Modern CPUs are powerful and often idle, so the extra resource usage by .NET and JVM would not have a significant impact for many desktop users. However, it could be a concern in environments where resources are constrained. Additionally, managing deployments for applications built with different ecosystems can be demanding, especially when ensuring consistent environments across development, testing, and production. Developing a system with two languages necessitates skill with both languages. But both ecosystems are well-established and have large communities, meaning that there is a wealth of documentation, tutorials, and community support available. This extensive library support facilitates development and the only drawback is that the developers must be proficient with extensive use of both systems, and debugging between these could be time-consuming.

2.2.2 Java with JavaFx and Golang

JavaFX, while powerful, can be more verbose and slower to develop compared with other UI frameworks including but not limited to JavaScript. For example,

creating a simple application requires multiple lines of boilerplate code, In contrast, a similar application in a more modern front-end framework like HTML with Electron or even a lightweight JavaScript library could be much simpler and more concise. JavaFX also lacks the rich, modern UI components and easy-to-use libraries available in frameworks like React or Vue.js, leading to potentially longer development times. Using different languages for the front end and backend can complicate the project and increase the learning curve. JavaFX does not have as large a community or as many third-party libraries compared with some modern front-end frameworks. Managing deployments for applications built with different ecosystems (Java and Go) can also be complex. Firstly, the build and deployment processes for Java and Go are different. Java applications typically require a JVM and may depend on a specific version of the Java Development Kit (JDK), along with external libraries and application servers like Tomcat or Jetty. Deploying Java applications involves packaging them into WAR or JAR files and ensuring the target environment is properly configured. In contrast, Go applications compile into a single binary, simplifying deployment as they don't require a runtime like the JVM. However, this difference means separate deployment pipelines for the front end (Java) and backend (Go), potentially needing different CI/CD tools and processes. Maintaining consistent environment configurations across development, testing, and production for both languages can be challenging, requiring separate environment variables, dependencies, and build tools. Ensuring secure and efficient communication between the front end and backend also adds complexity, such as setting up API gateways or message brokers.

2.2.3 Python with PyQt or Tkinter and Python

Python is generally slower than compiled languages like C# or Java. For CPU-bound applications requiring heavy computation, such as data analysis, scientific computing, machine learning, and real-time data processing, Python's slower execution speed can become a bottleneck. This should not be a drawback for MEX since machine learning experiments take time and the speed of MEX's processing would not make a substantial difference. While PyQt is quite powerful, Tkinter is relatively basic and does not provide sophisticated UI components or the performance of more modern frameworks. Python GUI applications might not integrate as seamlessly with the operating system's native features as those built

with native user interfaces rendering frameworks like WPF for Windows, part of the .NET framework. Python applications rely on multiple dependencies that need to be packaged together. Tools like PyInstaller and cx_Freeze help bundle these dependencies into a standalone executable. However, these executables can be quite large because they include the Python interpreter and all necessary libraries. This is not a problem with modern computers but ensuring that the packaged application runs smoothly across different operating systems can require significant testing and tweaking. Python's Global Interpreter Lock (GIL) can be a limitation for multi-threaded applications, making it harder to achieve true parallelism. GIL is a mutex that protects access to Python objects, preventing multiple native threads from executing Python bytecodes simultaneously in a single process. This means that, despite Python's support for multi-threading, true parallel execution of threads is limited. This can be a significant drawback for CPU-bound applications that need to perform concurrent processing. For example, the GIL can become a bottleneck if used for processing large datasets or performing complex computations. While Python's multiprocessing module can be used to bypass the GIL by using separate processes, this approach comes with its own overheads and complexities. Other languages like Java or C# do not have this limitation, allowing for more efficient use of multi-core processors for parallel processing tasks.

2.2.4 Electron framework and Javascript

Electron applications are known to be resource-heavy, often consuming more memory and CPU than native applications because they run on a full Chromium browser instance. Electron applications can be quite large in size because they package the entire Chromium browser and Node.js runtime. Managing a full-stack JavaScript application can become complex due to the need to handle both front-end and back-end code, including asynchronous operations and state management, but due to the use of web technologies such as HTML,CSS and React, The interface can be modern and responsive. Another major advantage of using Electron is its cross-platform capability. Electron applications can run on Windows, macOS, and Linux with minimal changes to the codebase, if necessary at all. There is a huge community and documentation for web development components which are utilised by electron and would contribute to the development of a modern UI.

Electron utilises Javascript, which has powerful libraries to assist in implementing features. Many of the latest software are built using Electron framework including Visual Studio Code, Atom text editor and Discord.

2.2.5 Chosen framework benefits

The Electron framework was considered the most suitable choice to develop MEX. The essential criteria for MEX were decided to be the design, easier availability of MEX, functionality of the system and support available. MEX is meant to be an easy-to-use and intuitive software with a shallow learning curve. Modern UI and as much flexibility with the design as possible is needed to make it possible, Web technologies allow the interface to be more modern and customised. Python fulfils the requirements to develop MEX but it proves to be difficult to package for different OS and requires significant testing and tweaking for acceptable performance. Electron framework allows applications to be developed using a single codebase and has better tools to package the application. It is relatively simpler to package apps for different ecosystems when compared with other options. Options like C# and java and javaFx and Go require proficiency with the systems and much of the development time could be divided for debugging and setting up CI/CD pipelines to make them work with each other. This would result in less time for integrating new features and MEX would end up having less functionality than intended. Javascript also has many prebuilt libraries similar to python which would help with much of the functionality implementation and help MEX be scalable. There is a huge community of web developers to support the technologies, this is not the same for JavaFx and Go which makes JavaFx less viable. The downside is that due to chromium being packaged along with it large applications are produced, however, this should not affect the system performance as Machine learning experiments themselves take a significant amount of time and the difference caused by electron would be very low.

Choosing electron framework proved to be adequate as the intended features could be applied with the help of preexisting libraries and resources, however, some features proved to be troublesome to implement due to a lack of documentation and support and took more time than expected.

Chapter 3

User Interface

The graphical user interface (GUI) the user experience (UX) are critical to the project because a main goal is to make the work process simple and intuitive. There are many considerations which are taken into account to add and design each elements. Some of the UI elements have been added as a result of the user studies to incorporate user suggestions. A simple-looking design is not simple to design and requires considering many factors, This is described by Tidwell (2019) along with methods to achieve the same.

3.1 Design guidelines

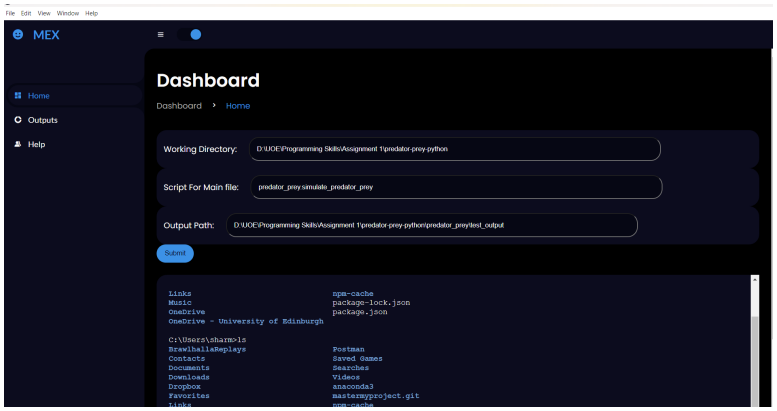
MEX provides six crucial features beyond file management and quality of life additions, but if this dataflow and presentation is not managed properly, it will lead to the under-utilisation of the software and in the worst case, make the learning curve very steep. To avoid this, five guidelines were studied and followed. Among these five guidelines, six specific points were decided to be followed specifically for MEX due to their relevance. There are many different design principles from industries and priority is given to those which benefit MEX most. This section focuses on creating an easy-to-use, effective and aesthetic design.

The guidelines considered are Babich (2019), Issa and Isaias (2015), Memon (2021), Nielsen (1994) and Yablonski (2020). There was considerable overlap within these sources for the guidelines, Nielsen (1994) *10 Usability Heuristics for User Interface Design* is a widely known authority on usability, His 1994 publication formed the basis for this project's interface design. The following points are the accumulation of the most useful points for MEX. They do not

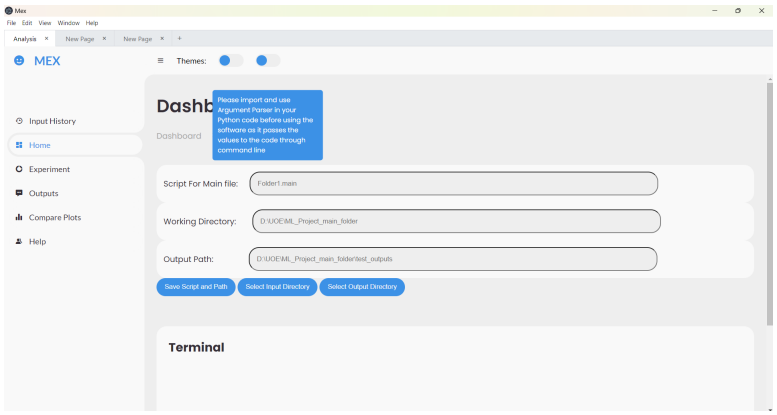
follow any specific guidelines but merge them together or are derived from the principles mentioned in them.

The language and workflow designed should be familiar to users To ensure the system resonates well with its users, it should employ terminology and concepts familiar to data scientists and machine learning practitioners, such as "data," "Experiment," "training," and "validation." Features should be organised logically to mirror the typical workflow in machine learning projects, grouping related actions. This contextual grouping helps users navigate the system intuitively, making it easier for them to understand and use it effectively.

User should have control and freedom over the experiments The user can select the required directories by clicking on a mouse and typing through a



(a) MEX initial version home page (dark theme).



(b) Using New tabs to run multiple instances of MEX at once(light theme).

Figure 3.1 Thumbnail comparison between MEX's initial (3.1a) and final (3.1b) versions.

keyboard. All inputs are saved in the input history and the user can navigate away from any of the tabs. The arguments to be run are also decided by the user and can be changed, put on hold and deleted at any time. The outputs generated are also easily accessible and can be edited and processed if required.

The design should be consistent and intuitive Maintaining a consistent design throughout the application is crucial. This includes a uniform layout, colour schemes, and typography, ensuring that users can easily recognise and use the interface. Standard icons and buttons should be utilised for common buttons including but not limited to Home, Plot and History, promoting a seamless user experience. The interface should also behave consistently across different operating systems (Windows, macOS, and Linux) supported by Electron, providing a familiar experience regardless of the platform.

The UI should prevent errors Input validation is essential to prevent common mistakes, ensuring that data formats and parameter values are correct before proceeding. These are done by performing basic checks and using mouse clicks more often to ensure validated input is used. Guided workflows for complex tasks can help users avoid errors by providing step-by-step instructions and leading them through the process in a structured manner. MEX navigates on its own to different pages to make it easier for the user to understand the flow than figuring out through the terminology or software documentation.

Recognisable patterns should be used The system minimises the need for users to remember information by providing pre-filled example values for experiment settings based on common practices. Dropdown menus and auto-suggestions can assist users in selecting options without needing to recall exact names or details. Tooltips and inline help should be available to explain features and options, reducing cognitive load and making the system easier to use.

The design of the system is made to be clutter-free, focusing on essential features for managing experiments. A clear visual hierarchy is maintained to prioritise important information and actions, making the interface intuitive and easy to navigate. A simple and clean design is crucial to reduce cognitive load and make the system aesthetically pleasing and user-friendly.

Informative Help Section must be present An integrated help section will support users in finding the information they need quickly. Tooltips, example experiments to become proficient with the system. tooltips help out with individual areas where there might be some confusion and the help section expands on such areas to explain the entire process. figure A.4a is the help section for user support for MEX

3.2 Accesibility

MEX is deliberately made to be accessible to a wide range of users. The focus is on visual and cognitive accessibility since MEX does not rely on auditory or mobility interactions. Following the Web Content Accessibility Guidelines (WCAG) standards set by the World Wide Web Consortium (W3C) (W3C, 2018), MEX is aimed to be as inclusive as possible under the project constraints. Mex is meant to be a desktop application but it utilises web technologies and websites are made to work on all operating systems, So, these guidelines are appropriate for our use case. Different standards are followed for each operating system and having a centralised set of rules is easier to follow. An extension of MEX can also be made as a website in future, if needed as similar technologies are used, which makes MEX more accessible and scalable.

To address visual accessibility, we used tools like the WebAIM contrast checker (WebAIM, 2019). MEX is made to be accessible for colour-blind users as well by maintaining a varied contrast to make sure content is readable. who number around 300 million globally, including 1 in 12 men (8 %) and 1 in 200 women, according to Color Blind Awareness (Colour Blind Awareness, 2022). Consequently, additional visual cues and labels to help distinguish error types are incorporated , making the interface more accessible to all users.

For users with low vision, we ensured that all text in MEX has a high contrast ratio against the background. This is crucial for readability and aligns with the WCAG 2.1 AA standard and WCAG 2.1 AAA standard, which requires a contrast ratio of at least 4.5:1 (for the AA standard) and at least 7:1 (for the AAA standard) for normal text, where each A signifies a higher standard. (WebAIM, 2019). By meeting these standards, MEX remains usable and clear for everyone, including those with visual impairments.

Addressing cognitive disabilities, particularly dyslexia, involved careful design choices. Most of the information is separated in a sub-division of the GUI into smaller, manageable boxes to avoid overwhelming users with large blocks of text. Consistent use of intuitive icons aids in forming clear associations between problems and solutions, reducing the cognitive load. We avoided using italics for key shortcuts and refrained from using all capital letters, as these can be difficult or even impossible for dyslexic users. Additionally, to mitigate problems caused by high contrast ratios, the stark combination of pure white (#FFFFFF) and pure black (#000000) were avoided.

Because dyslexia and colour blindness affect individuals differently, MEX offers three themes (in the order: text, background, subdivsion): light (#342E37, #F9F9F9, #EEE), dark (#FBFBFB, #0C0C1E, #060714), and pale yellow (#333333, #FFF2C6, #FEF6E4) to cater to most visual preferences and reduce eye strain, see figures 3.1 and 4.1a. (Fernandez et al., 2021) COSAT 2.0 is a system developed in this study to suggest a colour scheme for software according to their use, and they claim around 80 % accuracy denoting that there can not be one universal theme for all users which is why three themes were used in MEX to make sure most users will find it accessible. Each theme is designed with accessibility in mind, ensuring that text and interface elements remain clear and readable under different lighting conditions. This flexibility allows users to select the visual setting that best suits their needs.

3.3 Interface elements

MEX's designs are split into different design elements across six pages, but the Navbar remains constant across them all: input areas, experiment table, output areas, text editor, tooltips and Resizing. Initially, each component had a basic functionality and then new components were added and improved with extra features to make the tool more helpful for beginners. The initial overview diagram Figure A.1 and Figure 3.1 can also be seen for the improvements made. Two user studies were conducted to receive feedback and to add their suggested features and designs. These studies were conducted to create a feedback loop by listening to participants' opinions on MEX. This section introduces the design process of MEX and the improvements we made.

3.3.1 Navbar

The Navbar is a crucial component of the system, serving as a bridge between all the pages and guiding users through different functionalities. Figure 3.1 (p. 13) shows the difference in the navbars as there are more components in the newer version. It provides a clear flow of the system, with intuitively placed links that users typically navigate in sequence, rarely needing to jump out of order. Initially designed with three main tabs, the final Navbar design includes six tabs, with an additional feature to add notes in the results in outputs accessible through a new window. The six tabs are Input History, Home, Experiments, Outputs, Compare Plots, and Help. The Navbar can be collapsed to provide more space on the main page, and elements turn blue when hovered over, indicating which features can be used to carry out experiments. Positioned at the top and left sides of the screen, the top Navbar contains only the burger icon to collapse the Navbar and a button to change the colour scheme. The left side houses the main features, organised to enhance the user experience by separating essential functionalities from user experience enhancements, preventing clutter.

3.3.2 Input areas

The design for input areas has been kept consistent. A grey rounded bubble-like design is used for inputs, Figure A.3a and Figure 3.1. Users must enter paths and a script to use the software. The user studies revealed that it was not clear to a user running their first experiment what the format should be, thus they made mistakes typing the inputs. To rectify this, placeholder text with example inputs was added to the system and designated buttons were added for inputs like input path and output path which have a higher chance of being entered incorrectly and are also tedious to enter. This helped in error-prevention and enhancing user experience (UX). Other than placeholders, some inputs were initialised with standard values which work even if they are not altered but the user could get a different result such as in plotting graphs of the processed xls files.

Each input element is placed in a separate box with a label to distinguish it as a separate entity. This is helpful for users with visual or cognitive disabilities. The text in these fields changes their colour along with the theme as well.

3.3.3 Experiment table

This table is used to enter required arguments and values. The number of arguments the user may want can vary, and having a fixed table would hinder the design and limit functionality if more arguments were needed later. Thus a dynamic table is adopted with two buttons which add a new row, and delete unselected rows, making the design more compact as the table would only be as long as the user's requirement, which can be seen in Figures 4.1a and A.2d. A sample row is always initialised for easier understanding. There is a checkbox with the fields which signifies whether the value should be included in the experiment. Only the selected values are used to run the experiment. Unselected values are excluded, allowing users to experiment with multiple combinations of the same variables to observe different results. Initially, the checkbox was used to delete the selected rows but this functionality was suggested and improved by the user studies.

3.3.4 Output areas

MEX displays four types of outputs: plots, terminal outputs, and generated file lookups. Example of these can be seen in figures 4.1a and 4.2a. These outputs are intentionally non-interactive to ensure that the reference data remains unchanged. To facilitate interaction with the outputs, additional buttons such as Generate Plots, Add Notes, Get LaTeX, and Current Outputs are provided. These buttons allow users to work with copies of the outputs without altering the original. Outputs generated by MEX are also given accessibility considerations to maintain readability. Having vertical lines in a table makes reading harder for people with dyslexia, so when a LaTeX table is generated using the 'Get LaTeX' button the resulting table does not have vertical lines. It has a clean design and always fits the page to ensure if there are many columns it will not overflow the page width. Implementation of these function can be seen on the output page figure A.4b .

3.3.5 Text editor

Adding notes requires a substantial viewport size because the generated terminal output can be extensive. Therefore, a separate window with a text editor is opened figures 4.1b and A.2b, allowing users to select and annotate the desired output. The design is kept minimalistic for ease of use. Once done, users can

simply close the window without needing extra prompts to save, as the updates are saved automatically.

3.3.6 Tooltips

figures A.3b and A.3c shows tooltips, The design element incorporated based on user study results. It was found that many components were unclear to first-time users, so the most common tips are addressed in context-specific tooltips. These tooltips are displayed only when hovered over and remain hidden otherwise, ensuring that the interface design remains clean and uncluttered. More about these will be discussed in the user studies section.

3.3.7 Resizing

MEX may not always be used in a full screen or wide screen mode and may be made to have only a small portion. This has become more common with newer OS features such as Windows automatically resizing apps for multi-tasking. All the components are made to adjust to the width available to make sure all the important data is displayed at all times if possible with the set width. Collapsing the navbar is another way more width can be given to the main screen when it is not taking up the entire screen.

3.3.8 Alerts

A major improvement from the initial design was adding alerts to interactable sections of MEX Figure A.3d. initially, the user was not conveyed if there was any error with MEX's internal inputs. Now, an alert is displayed on top of the screen to describe the current state of the system or if any wrong input may have been parsed. The user is able to fix it on their own and is in the know about the processes taking place,

Chapter 4

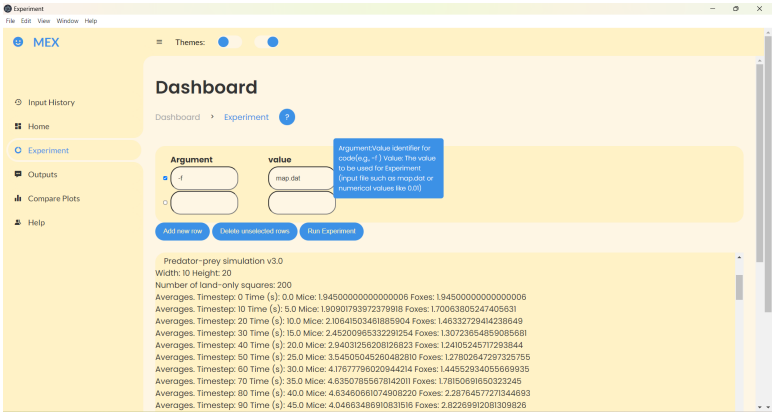
Implementation

Node.js (Dahl and Cantrill, 2009) is used with Electron (*Electron Framework*, 2013) for developing MEX. Node.js has several built-in modules that can be used without further installation and the npm package manager for easily installed JavaScript packages required for MEX. MEX takes the necessary inputs from the user generates a command line script to be run and saves the script's output for future reference and processing. This chapter introduces the backend implementation process of MEX, including all the functionalities introduced in the system. The app has 6 pages: input history, home, experiments, outputs, compare plots and help. Figure 4.1 has several features of MEX and their enlarged version for better understanding can be found in the Appendix at Figure A.2.

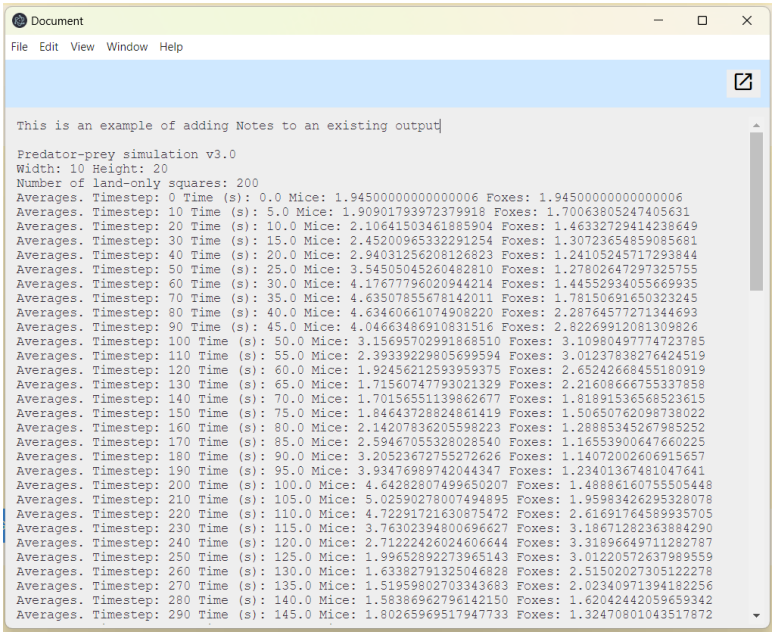
4.1 Features

4.1.1 Data management

The home page instance transmits the input form data to the main process and saves it in the sqlite database Section 4.1.5, where the working directory, output path, and script are stored globally. This ensures that the data does not need to be re-entered for the same session. The process is explained in depth in Appendix A.1. Inputs can be entered manually, and designated buttons have been added to allow users to use their cursor to enter the path using the "dialog" module available with Electron. Input taken using "dialog" module is filled as part of the input and the user may also edit it manually. "dialog" module cannot be used outside main JS file so the request is sent to the main file from the renderer JS to save the



(a) MEX experiment page (yellow theme).



(b) Adding notes to results.

Figure 4.1 Experiment outputs and editing with MEX.

inputs globally. Users can update this data by resubmitting the form on the home page. However, the table on experiments page for variables does not retain its values once the page is navigated away from. Only the checked values in the table are used for processing. Users can add or delete rows in the variables table. Basic errors that users might encounter have been handled, such as manually entering the path and including quotes in the string. This error is common because ctrl + ⇧ + c is a common shortcut in Windows to copy the file path, which includes quotes. This problem is resolved by reading the input as a string and removing any quotes present at the beginning or end.

Predator-prey simulation v3.0							
Timestep:	0	Time	0.0	Mice:	1.9450000000	Foxes:	1.9450000000
Timestep:	10	Time	5.0	Mice:	1.9090179397	Foxes:	1.7006380525
Timestep:	20	Time	10.0	Mice:	2.1064150346	Foxes:	1.4633272941
Timestep:	30	Time	15.0	Mice:	2.4520096533	Foxes:	1.3072365486
Timestep:	40	Time	20.0	Mice:	2.9403125621	Foxes:	1.2410524572
Timestep:	50	Time	25.0	Mice:	3.5450504526	Foxes:	1.2780264730

Table 4.1 Example table of the test dataset generated using MEX.

To simplify user input, the "ModPath" function was developed to require only the `main.py` file path. This function identifies the working directory and the main script independently. For example, an ML project may have a root directory with subdirectories for configuration files and datasets. The function assumes that the second directory from the main file is the working directory and automatically generates the command line script accordingly. The main hurdle was ensuring the project follows a standard structure; deviations, such as non-standard relative paths, could lead to user confusion and workflow complications.

Another concern was that a Python file could be executed either as part of a module (e.g., `ml_project.main`) or individually (`main.py`), with differing syntax for each method, causing errors if the correct script is not run. Although the "ModPath" function worked as intended and was initially included in MEX, a third input field for "script to run" was added in later versions to increase MEX's flexibility and accommodate more use cases, ultimately leading to the discontinuation of "ModPath".

The backend of the experiments page has three main functions: AddRow, DelRow, and ExtractValues. AddRow inserts a new row with empty fields and a checkbox at the end of the row. DelRow deletes any unchecked rows. ExtractValues retrieves values from each row's cells. If a row's checkbox is selected, its data are added to a string, which is then split into an array for the main process.

4.1.2 LaTeX tables

Users can interact with the generated outputs on the outputs page. LaTeX tables are created using the saved XLS file and the text output generated after running

experiments. The data, already sorted before being saved in the XLS file, are used to create the table. Standard LaTeX commands, such as `\begin{table}`, and the `\usepackage{booktabs}` package are utilised to generate the table. The XLS file is processed and tabulated for LaTeX. `\resizebox` has also been used to make sure that the generated content for the table would not overflow from the page size and makes it usable for a large width of data as well. The worksheet generated earlier in XLS is embedded within these LaTeX commands and text to be used in the document for displaying the table. Table 4.1 is an example table generated by MEX for the testing dataset predator-prey. The table can be further customised by the user according to their preference and only a small portion of the entire base table is displayed here as an example.

4.1.3 Adding notes

The feature allowing users to add notes is designed to enable the review and modification of existing results and an implementation can be seen in figures 4.1b and A.2b. This functionality lets users append their insights or observations regarding the specific hyperparameters. The "Add Notes" button launches a dedicated edit window featuring a text editor. Users can then select the desired results file via the "dialog" module. Any modifications to the file are monitored by the JavaScript associated with the page renderer, which subsequently communicates these updates to the main application. The "fs" module is used to persist these changes in real-time. Xls file is generated along with the text file so that the notes mentioned later by the user are not used for tables and sorted data.

4.1.4 Plots

Figures 4.2a and A.2a is an example of comparing these plots. This feature allows users to compare the performance of different or similar models to identify any noticeable differences in results. Chart.js (Downie, 2015), an open-source JavaScript library for visualisation, is used to generate the plots. it stands out for several reasons. Its simplicity and ease of use make it ideal for quickly implementing complex visualisations without a steep learning curve. The library's support for responsive design ensures that charts automatically adapt to different screen sizes and devices, enhancing the user experience. Chart.js also offers extensive customisation and flexibility, which means that the plotting features could be

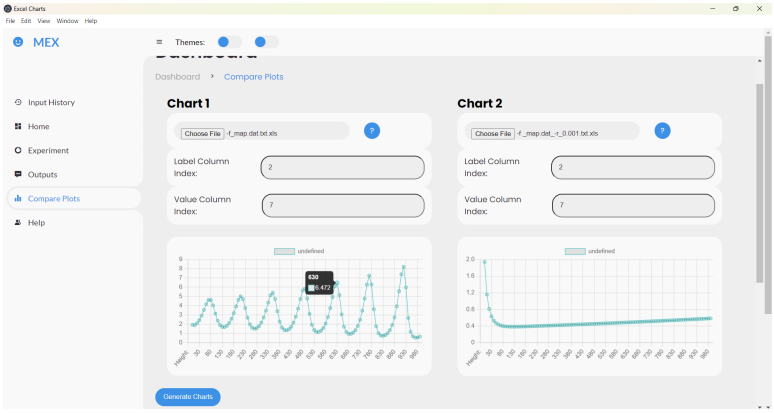
extended in future for different use cases. Additionally, Chart.js is lightweight, crucial for maintaining efficient resource usage in an Electron application.

File operations such as reading and processing Excel files using the XLSX library are handled by the main process. This system design choice avoids potential performance bottlenecks in the renderer process, which is primarily responsible for updating the UI. The renderer process manages data transmission and tracks necessary updates. Each plot is assigned to a window, which is removed and recreated every time the "generate plot" button is used, ensuring that plots are updated correctly. There are many different types of experiments and the data generated differs from each other and cannot be generalised. Therefore, this feature allows users to specify the columns they wish to visualise. Although an algorithm could be developed to automatically retrieve essential data, it would require significant time and resources, potentially detracting from other aspects of the project. The values for these columns are pre-initialised and typically do not need to be changed if the data is sorted.

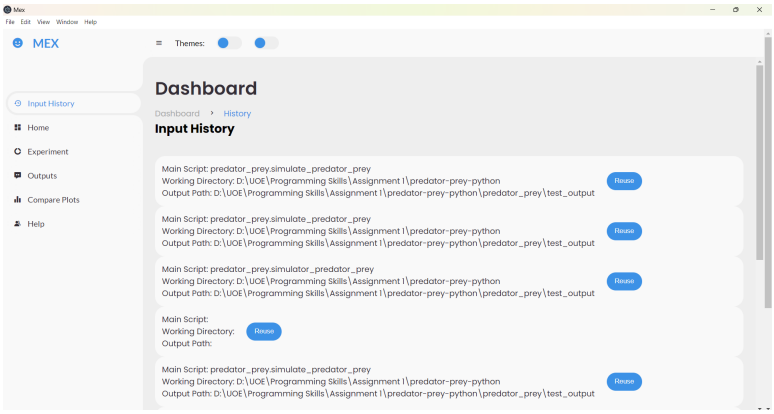
4.1.5 History

SQLite is used to save all the previously made inputs by the user and made available on an input history tab to be seen and used. Unlike more complex databases such as MySQL or PostgreSQL, SQLite operates without requiring a separate server, making integrating and managing within an Electron environment easy. Its self-contained architecture ensures efficient performance with minimal resource overhead. Furthermore, SQLite's portability, with databases stored in a single file, simplifies deployment and versioning, as the database can be easily bundled with the application. compared with NoSQL databases or file-based storage methods, SQLite provides structured data management and efficient querying capabilities, making it a robust and practical solution for small to medium-sized desktop applications that require reliable and efficient data handling.

As the history of inputs grows with the system being used more frequently, it may become cluttered and confusing if displayed alongside the current form being submitted. To maintain clarity, the history is shown in a separate tab, allowing users to review and reuse past inputs easily as seen in figures 4.2b and A.2c. This design choice adds complexity to the backend but enhances user experience by keeping the interface organised. Achieving this requires communication between



(a) Comparing plots for different results.



(b) Input History for previous experiments.

Figure 4.2 Plotting and input saving windows of MEX.

three files: the input renderer, which saves inputs; the history renderer, which reads and sends inputs back to the initial form; and the main file, which coordinates these interactions. Direct communication between the renderer files is not possible, so they rely on the main file to relay information and manage roles for further computation. A reuse button on the history tab enables users to redirect to the input tab, where the selected data is automatically filled into the input form.

4.1.6 Themes

Additional theme presets are at the top of the screen to be changed according to user preference. Having three themes gave rise to preference errors among the themes figures 3.1 and 4.1a. The colours would get mismatched and a separate function is made to change the term colour area for the terminal. These were resolved by making class names and specifying their colours in the CSS which would be applied when the specific class name is added to the body using JS.

4.1.7 Terminal

"node-pty" (*node-pty*, n.d.) is used to fork processes with pseudoterminal file descriptors. It returns a terminal object which allows reads and writes. The pseudoterminal for MEX is maintained constantly by this module as "child_process" alone cannot maintain a pseudoterminal. Xterm.js make a fully-featured and working frontend for the terminal (*xterm.js*, n.d.). As seen in (*node-pty*, n.d.) and (*xterm.js*, n.d.), This combination is used by many current and popular applications to make pseudoterminal such as visual studio code, hyper and theia. An example of this terminal can be seen in the figures Figure 3.1a .

This has been one of the most problematic features to add due to the number of difficulties faced. "node-pty" is not supported directly in Windows and thus requires extra dependencies and toolkits. The documentation is outdated as the commands mentioned for the dependencies to be installed and the toolkits have been deprecated, so the process fails. A separate module "node-pty-win" (*node-pty-win*, n.d.) was attempted to be used for Windows since it was said to support Windows using the winpty library but new errors such as ENONENT and node version mismatch occurred. Even though similar errors and queries were found online, the community support for node-pty is very limited and most of the questions remain unanswered. The fix to use node-pty was to install node.js again through the setup file and enable the download for chocolatey and other tools during the installation. The fix would only work while using the setup file and using nvm to update node will have no effect.

A white textbox with the class name `Xterm_helper_text_area` is displayed with the terminal. Limited online support was found for hiding or removing it using xterm only. Removing the classes directly from the xterm.js div caused the terminal to stop registering inputs. The final workaround was adding the class to the CSS file for mex and setting its display to none. This removed the text area, but left a small empty space.

The terminal can display only up to 120 columns of data and doesn't save previous data. While this doesn't affect system functionality, users working with data longer than 120 columns might need to refer to generated files more often. To address this, a separate output div was created to save and display all results for each experiment run in the same session.

4.1.8 New tabs

This feature was planned for MEX and efforts were made to add it but ultimately had to be withheld due to complications. “electron-tabs” was originally used which uses “Webview” under the hood, but ultimately “Browserview” (*Electron Process Model*, 2013) had to be substituted. There were many errors faced with “Webview” while implementation the tabs feature, such as the content not being loaded or the tab having an empty instance instead of an instance of MEX. “Browserview” fixed these problems and the logic for managing tabs could also be implemented with it, but the security considerations for the same were interfering with MEX’s processes. MEX has a terminal working on each page and to protect against attacks such as XSS “Browserview” does not let MEX use new spawn processes. The initial tab can still use the processes but when a new tab tries to spawn a terminal the rest of the JS is not loaded. To overcome this error, the JavaScript could be bundled with each renderer tab made but it would take up more of the processing as it would be like using two MEX at the same time and it would require more processing power as new tabs are made and used , (Williams, 2019) gives an in-depth analysis of the effects on the performance which can be observed with MEX in this scenario. This is not ideal since MEX is intended to be lightweight and this could impact performance severely. An ideal solution would be to make a separate service for terminals and each rendered page could call this service for their terminal. That way the number of spawn processes can be limited and performance would not be impacted, however, it would take significant changes in the present architecture of the system and there is not enough time to make and test these changes. An example of current newtab implementation can be seen in Figure 3.1b.

4.2 Packaging

MEX is an easy-to-install, cross-platform application for Windows, Linux, and macOS. It uses Electron Forge for packaging, which simplifies the process by handling many requirements that would otherwise need to be specified. The main requirement is to package it on the intended operating system, as Electron Forge checks the OS, required permissions, and dependencies, and includes them in the package. I primarily used Windows for developing MEX, with limited experience on macOS and Linux. This limited my ability to intuitively solve platform-specific

errors, requiring me to test and implement solutions while learning about these systems. Differences in software behaviour across platforms were studied and addressed to ensure a consistent user experience, with some adjustments made after testing and packaging.

The codebase remains the same across all platforms, but configurations were added to ensure it can be packaged for different systems. Packaging MEX for Mac required extensive trial and error to resolve bugs and errors. For Linux, Dice machines were used, which led to several errors due to the lack of sudo access needed for setting up the development environment. Workarounds involved adding necessary files and dependencies directly to the project folder to bypass sudo commands, although this was time-consuming. An error with Electron on Linux related to the SUID sandbox helper binary (chrome-sandbox), which typically occurs due to permission errors related to the sandboxing mechanism, but changing the permission required sudo command, was resolved by disabling the chrome sandbox in the config and main file, which had no negative impact.

Using a virtual machine (VM) as an alternative proved inefficient due to its poor performance, which slowed down basic tasks and required a development environment setup for packaging. Running a machine learning experiment to verify the build was not feasible with the level of performance the VM had. After addressing all Linux packaging errors, it took the Dice machine about 20 minutes to package a .deb file, which would have been more intensive on a VM.

Operating systems differ in user interaction and control, so platform-specific functionality was added to MEX for intuitive use. Windows uses Command Prompt or PowerShell, while Linux and macOS use bash, requiring “python3” for script execution. Shortcut creation/removal during installation/uninstallation on Windows must be handled separately. Additionally, applications close when all windows are closed, except on macOS, where the app and menu bar remain active until explicitly with ⌘+Q.

4.3 End-time running and testing

Applications packaged with Electron are typically self-contained, incorporating all necessary runtime components and dependencies. This self-contained nature guarantees that end users do not need to install any additional software to run MEX. On macOS the packaged application would not run, complaining “MEX is

damaged." This is because applications not signed by a trusted Apple developer are flagged as dangerous. The only method to have an application signed is to register as an Apple developer for a fee of $99 per month. This was found through the user studies and this error does not arise on the machine the software is packaged upon as it is considered to be a trusted software because of being made on the same machine. To overcome this error, the user would have to these commands: `sudo spctl --master-disable`, `xattr -cr <path/to/mex.app>` and `sudo spctl -master-enable`

Basic tests were conducted to verify the compliance of MEX's results with sample machine learning experiments, ensuring compatibility with various user experiments. The generated files were tested using Notepad, Microsoft Excel, and Overleaf. All files were accessible, and the desired LaTeX tables were successfully generated. These preliminary checks confirm the system's functionality, although a production system would undergo much more rigorous testing.

Electron applications are susceptible to security vulnerabilities such as Cross-Site Scripting (XSS) and code injection due to their use of web technologies. To mitigate these risks, external link imports were prohibited and Content Security Policy (CSP) to prevent XSS attacks were implemented. Electron's security features, like context isolation and disabling Node.js integration in renderer processes, are used, to reduce the attack surface, and using the sandbox attribute for renderer processes to further enhance security.

Chapter 5

Evaluation

A user study is valuable for assessing whether the system functions as intended, achieves the desired use cases, and to improve the design. This can potentially correct designers' assumptions that users share their understanding of how to use the design. Two user studies were conducted to evaluate MEX's design and functionality, with participants comprising university students and faculty from Informatics ($n = 10$). All participants were there for both user studies so 20 responses were collected in total.

The first user study aimed to gather feedback to identify the parts needing improvement. The second study assessed whether the applied improvements resolved the drawbacks identified in the first study. Both studies were conducted using questionnaires to collect responses from the target users.They had knowledge of the initial version of MEX so the second user study focused on collecting data on the improvements made by asking similar questions rating different aspects of the system followed by two new questions. The summary for both the user studies can is referred in Appendices B.2 and B.3.

The System Usability Scale (SUS) is a widely used tool for evaluating the usability of a system or product (Brooke, 1995; Sauro, 2016). SUS provides a "quick and dirty" method of assessing usability through a simple, 10-item questionnaire covering necessary aspects of the user experience. Each item is scored on a five-point Likert scale, ranging from "Strongly disagree" to "Strongly agree." The scores are then combined and converted to a single usability score, ranging from 0 to 100. MEX must be evaluated by more than one means so SUS is taken as a standardised measure for designing the form to collect the data but other tests and scales would be used.

5.1 Questionare design

The same questionnaire was used for both but with 2 additional questions at the end of the second to evaluate the changes. One question covered participation in the first user studies and the other the satisfaction level with the changes. The questionnaire was designed both with optional long answers and MCQ questions to facilitate quick completion and aiding analysis. The questionnaire was designed considering that user study 1 is for a base system of MEX with only the core functionalities, but the second user study will have a more complete MEX with functionality and improvements suggested by the first studies. The questionnaire was designed with the following considerations:

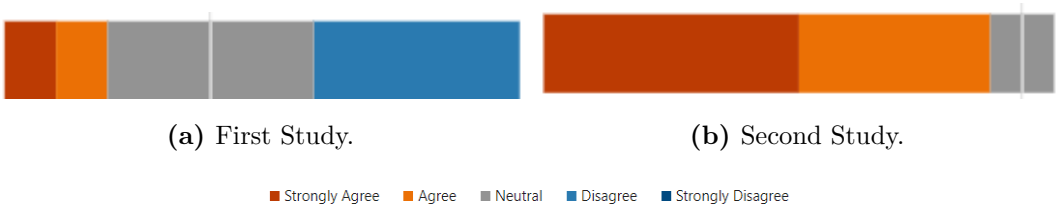
1. Keeping the question brief and relevant
2. There should not be more questions than necessary
3. Making them easy to answer by keeping text questions to a minimum
4. Desiging important questions to have a agree or disagree pattern to measure statistical performance

5.2 Responses collected

The same 10 users were present for both studies. 20 responses were collected in total. The median completion time of the questionnaire is less than 5 minutes, indicating that the duration is acceptable and that users can focus on the answers. All the participants are Informatics faculty or students including undergraduates and postgraduates. 50 % of the participants had not previously used any machine learning experiment tracking tools. Those who had used the tools covered in Section 2.1. It was observed in the user studies that the design was mostly easy to follow for users as `Use of words is easy to understand` had a 75 % positive rating and `Use of icons is easy to understand` had a rating of 80 %.

5.3 Analysis

The questionnaire is in 4 parts: interface design, features, user experience, and continued usage. The study focused on negatively reported parts of MEX to improve the system overall. Most of the information was collected using Likert scales to facilitate processing.



(a) First Study. **(b)** Second Study.

Figure 5.1 Ratings for "I did not need to see the help option to understand the functionality".

(a) First Study. **(b)** Second Study.

Figure 5.2 Ratings for "Features offered by MEX are useful to me".

Category	Min	Max	Avg	Med
Interface	68.8	88.5	83.2	80.3
Features	67.9	93.4	81.5	82.3
UX	75.0	90.0	85.7	85.0
Overall	79.4	88.5	80.0	81.1

(a) First user studies.

Category	Min	Max	Avg	Med
Interface	80.2	100	93.2	90.5
Features	83.6	100	96.2	95.5
UX	85.4	100	94.0	93.8
Overall	89.4	96.5	93.5	92.0

(b) Second user study.

Table 5.1 Likert table statistics from both user studies.

Category	t-statistic	p-value
Interface	−7.54	<0.000 1
Features	−8.62	<0.000 1
UX	−6.87	<0.000 1
Overall	−9.23	<0.000 1

Table 5.2 Welch's t-test Results.

5.3.1 Likert analysis

Figure 5.1 shows `"I did not need to see the help option to understand the functionality"` had 40 % of responses as neutral and 40 % as disagree. This meant users had to refer the documentation to use MEX. Improvements are discussed in Section 3.3.2 and Section 3.3.6. These changes resulted in the second user studies having a 90 % responses as agree or strongly agree for `I did not need to see the help option to understand the functionality`. An increase of 30 % in `Design is intuitive` is also seen for positive responses.

Section 4.1 discusses the functionality that MEX offers and `"Features offered by MEX are useful to me"` had 40 % responses as neutral and 30 % responses as disagree. This meant that users required further functionality and wanted to achieve more use cases with MEX. To accommodate their suggestion, new features were added which are discussed in Section 4.1.3, Section 4.1.5, Section 4.1.4 and some other features which could not be added due to the constraints are mentioned in Section 6.2. These changes resulted in the second user studies having a 100 % responses as agree or strongly agree for `"Features offered by MEX are useful to me"`.

`"Features offered by MEX are useful to me"`. received more neutral responses than negative ones, It may be because users tend to do additional tasks on the results to find an optimal solution. MEX was saving the experiments just as intended but the users had to rely on additional apps for extra tasks. This insight is proved as the `"MEX helped with my experiments and fulfilled my needs"` gained a 30 % more positive responses, after Adding new features.

The results of the studies can be understood better if observed as scores or percentages. The Likert scores are considered as Strongly disagree - 1 point, Disagree - 2 points, Neutral - 3 points, Agree - 4 points, and Strongly agree - 5 points. Interface has a total score of 25, features has a score of 25 and UX has a score of 20. The overall score for the Likert scale is considered in percentages.

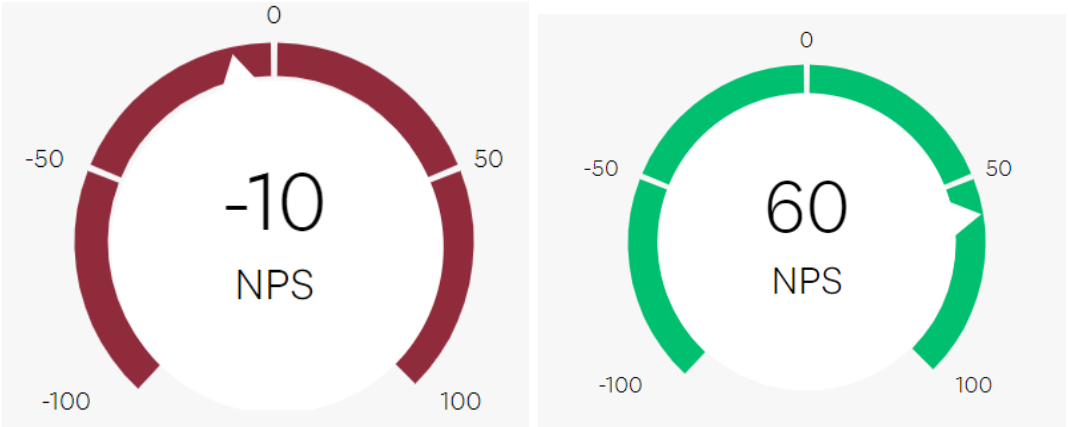
Table 5.1 shows substantial improvement in the feedback with the second user studies. Each topic has received the maximum marks possible and each average is more than 90 % whereas it was only more than 80 % for the first user study.

5.3.2 Welch's t-test

To confirm the findings of the user studies, a Welch's t-test (Pananos, 2020) is also done for all the categories mentioned in Table 5.1. Welch's t-test is a statistical test used to determine if there is a significant difference between the means of two independent groups when the variances of the groups are not equal. It is an adaptation of the Student's t-test and is particularly useful when the two groups have different variances and sample sizes. The Welch's t-test takes the standard deviation, sample and size mean for each group to calculate the t-score, which can be converted to the p-value(probability of the means being the same and having no difference). Table 5.2 shows that the p-value is much less which means that the changes made are relevant to the userbase.

5.3.3 Net Promoter Score

Net Promoter Score, NPS, (Reichheld, 2003) is a metric used to gauge customer loyalty and satisfaction for a product. There are 3 categories for users: Promoters (9–10) who are enthusiastic and satisfied, Passives (7–8) who are satisfied, and Detractors (0–6) who are not. NPS does not necessarily have the rigour of some mathematical measures, but it is still a useful indicator when used properly. NPS has been criticised (Keiningham et al., 2007) for being misused and overstated in business. It has no standard method either for governing collecting the data or for using its results. However its use here is considered acceptable as an indicator. A positive NPS score indicates that MEX has more promoters than detractors.



(a) NPS score for first user study. **(b)** NPS score for second user study.

Figure 5.3 NPS scores of MEX.

Figure 5.3 shows the NPS score of MEX in both studies. The first study got a score of -10 which means that there are more detractors for the system, however, the second study got an NPS score of 60 which means that most of the users are promoters now. The participants are the same for both the studies which means that the initial detractors were satisfied with the performance of MEX and changed to promoters. Initially, MEX had 4 detractors but after the improvements, no detractors were there for MEX. This result suggests that the user satisfaction of MEX has increased noticeably as even if the final version alone is analysed, we still see that the NPS result is highly positive. All users are satisfied with MEX and nearly half of them are enthusiastic about it.

Chapter 6

Conclusions

6.1 Project summary

In this report, we discussed the development of MEX an experiment manager for machine learning. MEX was further improved and equipped with new features by evaluating it using user studies. The work can be summarised by answering the questions from Section 1.3.

RQ1. MEX has 8 types of use cases: keeping track of already done experiments, running multiple variations of experiments, making data more usable by saving and sorting data in 2 file formats, generating latex tables for results, adding notes to present data, comparing results by plotting them, Saving a history of previously run inputs for experiments to be reused and a terminal for troubleshooting and environment setup

RQ2. MEX is made using electron framework which utilises web technologies (HTML and CSS) for the frontend and node.js (Javascript) for the backend implementation. Visual Studio code is used as the IDE while developing MEX because VScode itself is made using electron framework and it gave us insight into what the user experience and performance of an electron application would be like. MEX is designed to be a self contained application and this arrangement has made it possible. 6 sets of guidelines and standards were studied, with *10 Usability Heuristics for User Interface Design* (Nielsen, 1994) and *Web Content Accessibility Guidelines (WCAG) 2.1* (W3C, 2018) followed the most.

RQ3. The UI has three themes to increase accessibility and user experience. each element is separated in a sub-division of the screen to make it easier to

discern and to accustom users with dyslexia, placeholder text is written to explain the inputs and tooltips are placed over the elements where users need the most help (Identified through user studies). The design is kept linear and consistent and a navbar is present on the side to use other pages of MEX.

RQ4. MEX has the features to add notes to already generated outputs, generate latex tables, plot previously done experiments, a terminal for extra usage, different colour themes and a database saving the input history to be reused and the core functionality of saving and cleaning data generated through experiments

RQ5. The system is evaluated through two user studies with the same participants who rated the performance and usability of MEX.

RQ6. Out of the 8 ML tracking systems that were discussed it is observed that they have a steep learning curve and require integration with their frameworks to function but MEX is a completely self-contained and easy-to-use system, the learning curve is very shallow and no integration or changes are required it can be used directly with already present experiment without any prior knowledge about such tools, ensuring that the user can focus on their experiments.

6.2 Future work

Eight improvements were considered but could not be added due to time constraints. The database initialised currently only saves the inputs by the user to be used again. An extension could be made to save all the outputs generated with the timestamps such as the plots and LaTeX table, and all of these could be retrieved. A search function would also be useful to find specific results already generated and to look up notes added to results. This functionality could have been added for each file individually but it would be more useful when applied along with the database to find all such results. One improvement was to change the design of the outputs page and make the generated outputs interactive with functionality mentioned above. However, since there was not enough time, the design was not updated. Two quality of life improvements where the user would be emailed a copy of the result and a notification, if they desire so as ML experiments may take significant time and a user could devote their attention elsewhere until it

is done and a copy of the current working model and code could be zipped and shared within teams if needed.

One major improvement that would make MEX more useful is more methods for visualising results and data extraction algorithms. currently, only graphs can be generated to compare results but methods including but not limited to histograms or piecharts could make it more usable.

The second major improvement identified through user studies would be to increase the types of experiments which MEX handles. Currently MEX is useful for logging text data which is later processed. However, the features mentioned cannot be used for audio or image outputs. Such data can be saved but not logged and MEX cannot process such data. Additional features for these are an essential extension of MEX. A web version of MEX could also be made as web technologies were used to develop MEX allowing users to bypass the need to install MEX.

6.3 Reflections

I have gained experience with the electron framework and conducting user studies both of which I had no experience with them prior to this. I also gained knowledge of system architecture and enhanced my HCI (Human Computer Interaction) skills which I gained with the HCI course I studied in the first semester. Through this project, I have also learned about standards and guidelines which must be followed to make the software more inclusive and reach out to more people, researching and studying guidelines is something I also learned in my second semester with the course Standards Compliant: Software Development. A deeper knowledge of web and desktop design aided in the development. Fortunately, due to these factors, I was able to design MEX in a manner that needed new elements but did not require rework to the initial design other than the first body diagram. I got the experience of developing an application for 3 different operating software and what factors and difficulties should I be wary of in such an environment. I also gained experience with virtual machines and dockers during the packaging phase of MEX and have a better understanding of their use cases and drawbacks.

The “New tabs” feature would have been an excellent improvement of the system and even though the functionality was achieved, it was not a scalable solution. Due to the current architecture, a not ideal workaround was used to make it work and as new instances of the tabs are opened it would increase the

load for each system as each tab would utilise a separate main process. It would not have been good practice to include this feature with the workaround in the final version of MEX as it was known that the performance could be impacted and it could start affecting other tabs when a higher number of tabs are used. If I had known about this shortcoming, I would have designed the system to have a separate call service for the terminal as the errors encountered were due to multiple "spawn" processes being used. This would have allowed MEX to not suffer the excessive overload when new tabs are used as main process would not need to be bundled with each instance.

Learning all these new skills and technologies, and considering the best development alternatives for MEX has been a major struggle. I learned a lot from this project, However, There was much I did not understand as I began and had to keep on learning to make sure that I delivered before the deadline. I had not developed a system before for three OS and was unfamiliar with extra details that change within systems and how VMs can be used for development. Many errors required insightful solutions and to reach them I had to go through extensive documentation and forums to have an in-depth knowledge of the processes before solving them. Although many difficulties were faced, the technologies chosen have proved to be useful and most of the intended features could be applied. Javascript has many strong built-in libraries that helped me in processing and using elements present on the screen. I used VScode as my IDE while developing MEX because it is developed using the electron framework as well, it gave me many ideas as to what the design could look like and what functionalities are possible to be achieved for an app developed with electron. I have learnt a lot from the development of MEX in both front-end and back-end implementation and these intricacies will help me with my future projects which I could have only learned by working on a project of this scale only. If I had known a few factors such as the complications of using spawn processes with electron-tabs or difficulties of using node-pty with Windows, then more time could have been allocated and a different architecture would have been designed for the system. The only shortcoming of selecting electron framework has been the lack of documentation and community for specific dependencies such as the ones I implemented and had to look for solutions on my own with limited support. However, There was immense support available for design as initially planned and the other features including but not limited to plotting with chart.js were implemented because of electron and node.js.

References

Babich, N., 2019. *The 12 do's and don'ts of web design* [Online]. Ideas. Available from: https://xd.adobe.com/ideas/principles/web-design/12-dos-donts-web-design-2/ (cit. on p.12).

Berger, T., Lettner, D., Rubin, J., Grünbacher, P., Silva, A., Becker, M., Chechik, M. and Czarnecki, K., 2015. What is a feature? a qualitative study of features in industrial software product lines. *Proceedings of the 19th international conference on software product line* [Online], Splc '15. Nashville, Tennessee: Association for Computing Machinery, pp.16–25. Available from: https://doi.org/10.1145/2791060.2791108 (cit. on p.4).

Brooke, J., 1995. SUS: a quick and dirty usability scale. *Usability eval. ind.*, 189, November (cit. on p.30).

Colour Blind Awareness, 2022. *Colour blindness* [Online]. Colour Blind Awareness. Available from: https://www.colourblindawareness.org/colour-blindness/ (cit. on p.15).

Dahl, R. and Cantrill, B., 2009. *Node.js* [Online]. OpenJSFoundation. Available from: https://nodejs.org/en (cit. on p.20).

Downie, N., 2015. *Node.js* [Online]. Opensource. Available from: https://github.com/chartjs/Chart.js (cit. on p.23).

Electron framework [Online], 2013. OpenJSFoundation. Available from: https://www.electronjs.org/ (cit. on p.20).

Electron process model [Online], 2013. OpenJSFoundation. Available from: https://www.electronjs.org/docs/latest/tutorial/process-model (cit. on p.27).

Fernandez, S.V., Majid, M.A., Akma Abu Bakar, N. and Fakhreldin, M., 2021. Enhanced colour scheme assessment tool (cosat 2.0) for improving webpage colour selection. *2021 international conference on software engineering & computer systems and 4th international conference on computational science and information management (icsecs-icocsim)* [Online], pp.459–464. Available from: https://doi.org/10.1109/ICSECS52883.2021.00090 (cit. on p.16).

Issa, T. and Isaias, P., 2015. Usability and human computer interaction (hci). In: *Sustainable design*. Springer, pp.19–36 (cit. on p.12).

Keiningham, T.L., Cooil, B., Andreassen, T.W. and Aksoy, L., 2007. A longitudinal examination of net promoter and firm revenue growth. *Journal of marketing* [Online], 71(3), July, pp.39–51. Available from: https://doi.org/10.1509/jmkg.71.3.39 (cit. on p.34).

Kluge, K. and Jenkner, P., 2024. *13 best tools for ml experiment tracking and management in 2024* [Online], August. Available from: https://neptune.ai/blog/best-ml-experiment-tracking-tools [Accessed 19 August 2024] (cit. on p.5).

Lewinson, E., 2023. *A comprehensive comparison of ml experiment tracking tools* [Online], April. Available from: https://towardsdatascience.com/a-comprehensive-comparison-of-ml-experiment-tracking-tools-9f0192543feb [Accessed 19 May 2024] (cit. on p.5).

Memon, M., 2021. *The 21 main ux laws every designer must follow + examples* [Online]. Maze, June. Available from: https://maze.co/collections/ux-ui-design/ux-laws/ (cit. on p.12).

Nielsen, J., 1994. *10 usability heuristics for user interface design* [Online]. Nielsen Norman Group, April. Available from: https://www.nngroup.com/articles/ten-usability-heuristics/ [Accessed 11 April 2023] (cit. on pp.12, 36).

Node-pty [Online], n.d. Microsoft. Available from: https://github.com/microsoft/node-pty (cit. on p.26).

Node-pty-win [Online], n.d. NPM. Available from: https://www.npmjs.com/package/node-pty-win (cit. on p.26).

Pananos, D., 2020. *Unpaired student's t-test and welch's t-test* [Online]. StackExchange, June. Available from: https://stats.stackexchange.com/a/471791 [Accessed 26 August 2022] (cit. on p.34).

Popen() [Online], n.d. 6.9.1. Linux/UNIX system programming training. Available from: https://man7.org/linux/man-pages/man3/popen.3.html (cit. on p.45).

Reichheld, F.F., 2003. The one number you need to grow. *Harvard business review*, 81(12), December. PMID: 14712543, pp.46–54 (cit. on p.34).

Sauro, J., 2016. *Measuring usability with the system usability scale (SUS)* [Online], May. Available from: http://userfocus.co.uk/articles/measuring-usability-with-the-SUS.html#:~:text=What%5C%20is%5C%20a%5C%20good%5C%20SUS,through%5C%20a%5C%20process%5C%20called%5C%20normalizing. [Accessed 11 April 2023] (cit. on p.30).

Tidwell, J., 2019. Designing interfaces. *How to engineer software* [Online]. Available from: https://api.semanticscholar.org/CorpusID:114170273 (cit. on p.12).

W3C, 2018. *Web content accessibility guidelines (wcag) 2.1* [Online]. W3.org, June. Available from: https://www.w3.org/TR/WCAG21/ (cit. on pp.15, 36).

WebAIM, 2019. *Webaim: contrast checker* [Online]. Webaim.org. Available from: https://webaim.org/resources/contrastchecker/ (cit. on p.15).

Weights & biases documentation [Online], 2017. wandb.ai. Available from: https://docs.wandb.ai/ (cit. on p.5).

Williams, R., 2019. *Analysing multi-window electron application performance using chromium tracing* [Online]. Scott Logic. Available from: https://blog.scottlogic.com/2019/05/21/analysing-electron-performance-chromium-tracing.html (cit. on p.27).

Xterm.js [Online], n.d. xtermjs. Available from: https://github.com/xtermjs/xterm.js (cit. on p.26).

Yablonski, J., 2020. *Laws of ux* [Online]. O'Reilly, December. Available from: https://lawsofux.com/en/ [Accessed 31 July 2022] (cit. on p.12).

Appendix A

Requirements engineering

MEX **MUST**:

1. be easy and intuitive to use
2. be easy to install
3. be easy to configure
4. have useful configurability
5. be easy to extend functionality
6. provide some genuine benefit
7. correct at least some typical rookie errors
8. run reliably with a consistent GUI on at least two commonly used main-stream browsers

MEX **MUST NOT**:

1. mislead users
2. do bad stuff

MEX **SHOULD**:

1. support users with simple accessibility requirements
2. meet or exceed accessibility requirements or guidelines
3. adhere to good HCI design principles
4. follow good software engineering and programming practices
5. be easy to maintain
6. allow personal preferences for user control

MEX **SHOULD NOT**:

1. unduly confuse typical Edinburgh Informatics students
2. exclude users with simple accessibility needs
3. be limited to one browser or operating system

Check the `enumitem` package for extra features for lists. Here is how to make a description list that is automatically enumerated:

1. **Always remember to include everything**
   You'd be surprised how often people forget.
2. **Always check what you are submitting**
   You'd still be surprised how often people forget.

A.1 Workflow

To understand the back-end process occurring in the system, we would first look into the flow of running an experiment. On the home page, Three inputs(script to be run, working directory and output path) are taken from the user and saved. These inputs are saved in the database and used to generate the first half command line script and the page is redirected to the experiments page on submission. The experiments page has a table for the attribute to be updated for the experiment and its new value and is used for the second half of the command line. There are three buttons on the page which are to add new rows delete unselected rows and run the experiments with the values mentioned in the table.

Upon executing the script, the output is generated, displayed, and saved. Both a text file and an XLS file are created for the user's reference. The text file is useful for taking notes and reviewing the results, while the XLS file facilitates data maintenance, insight extraction, sorting, and further data processing. "xlsx" module is used for making and saving worksheets for the format.

The files are processed using the first two specified pages, with the results accessible via the output page. This page provides three options: displaying the currently generated outputs, adding notes to the current results, and generating a LaTeX table from the results. Additionally, a terminal remains available on the screen, allowing the user to perform further operations such as downloading new dependencies.

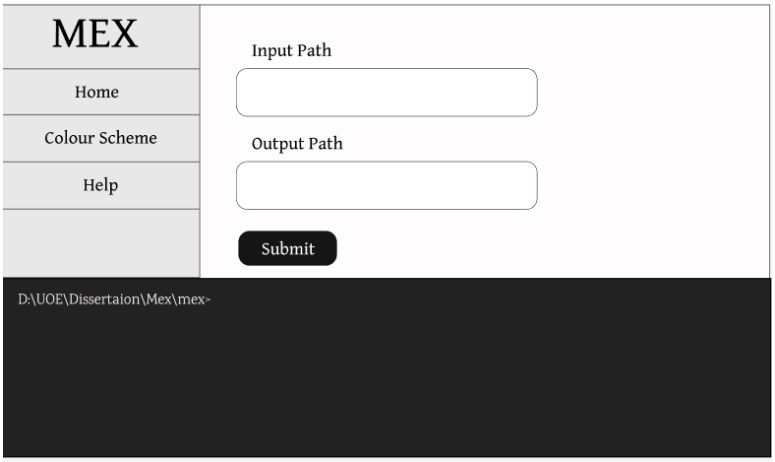
On the experiments page, Unselected rows are deleted if the delete button is used and a new row which is selected by default is made if add rows is used. The experiments page has 3 functions for its backend processes, which are AddRow, DelRow and ExtractValues. AddRow function looks up the variable table and checks the length of the entire table. A new row is inserted at the end with the empty fields of attributes and value and a selected box. DelRow function traverses through each row of the table and deletes the unselected rows. ExtractValues is the key function for the processing, the entire table is traversed through and

all three cell values are retrieved. If the checked box is selected or checked then the data is included in a string for final processing or skipped otherwise. A data array is created using the split function on the string made and passed to the main process for running the scripts,

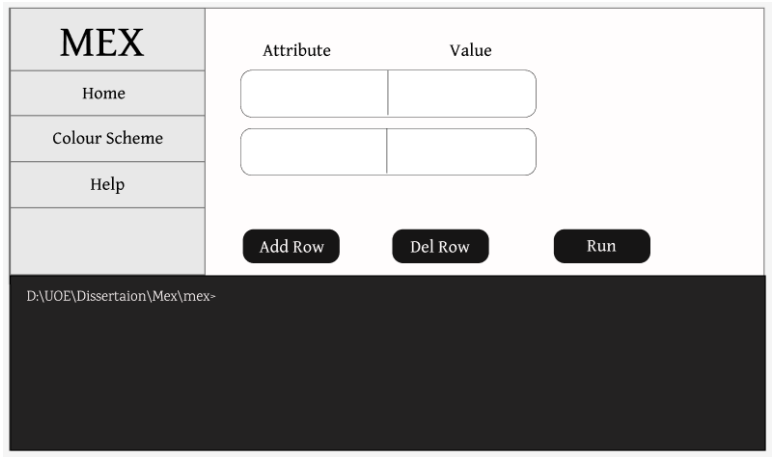
A.1.1 Terminal details

A terminal is present on every page for the user to run additional tasks which might be necessary for the user. The modules “child_process” and “node-pty” are used for the terminal. “child_process” provides the ability to spawn subprocesses in a similar manner, but not identical to popen() (*Popen()*, n.d.). The popen() function opens a process by creating a pipe, forking, and invoking the shell. Since a pipe is unidirectional, the type argument may specify only reading or writing, not both; the resulting stream is correspondingly read-only or write-only. The spawn function primarily provides this capability.

A.2 Important Figures

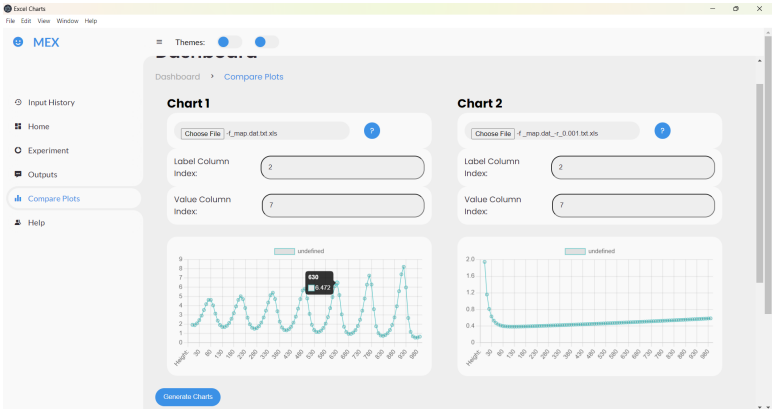


(a) MEX's home page.

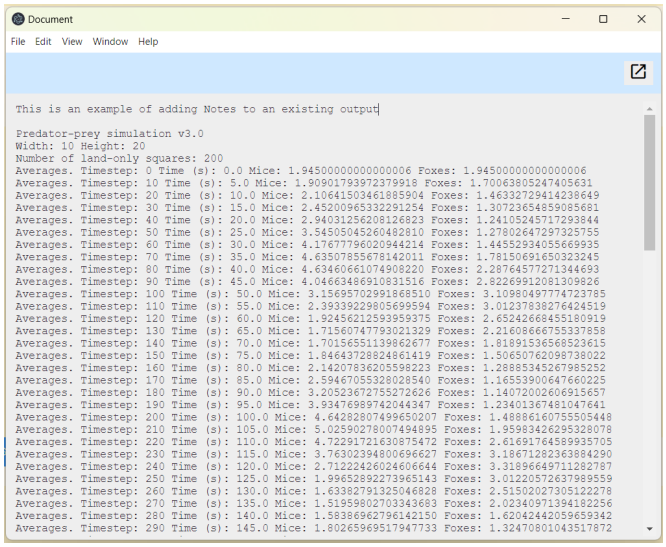


(b) MEX's experiment page.

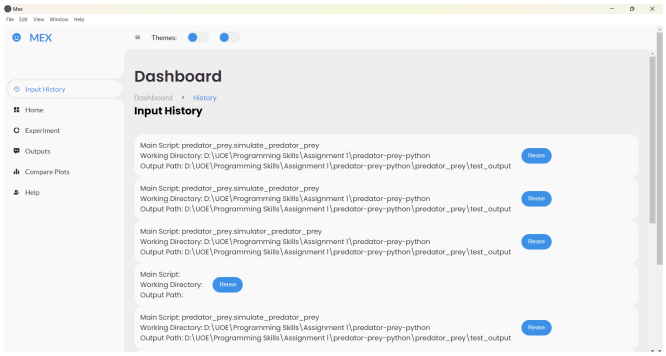
Figure A.1 MEX's first overview diagram design.



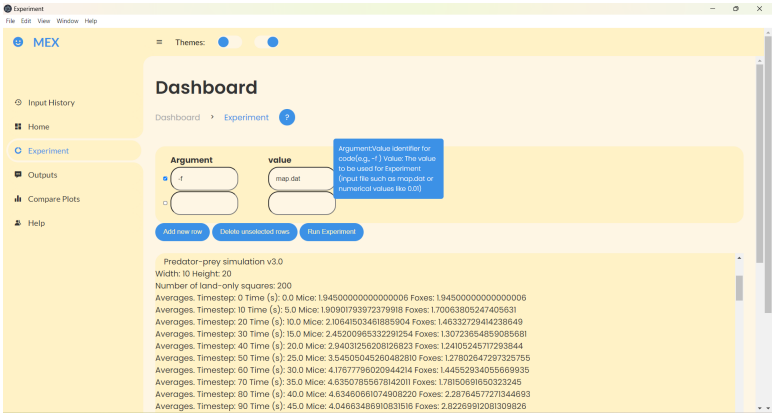
(a) Comparing plots for different results.



(b) Adding notes to results.



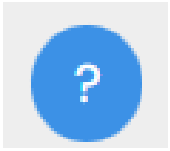
(c) Input History for previous experiments.



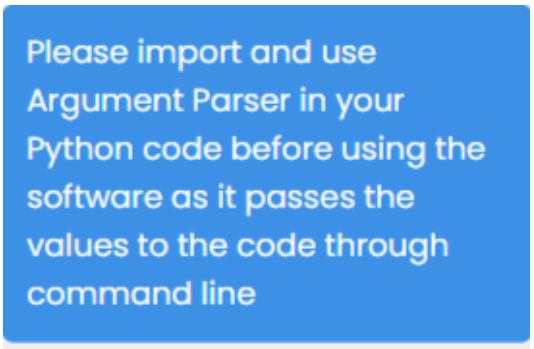
(d) MEX experiment page (yellow theme).

Figure A.2 Enlarged screenshots of MEX Features of MEX.

(a) Input field design.



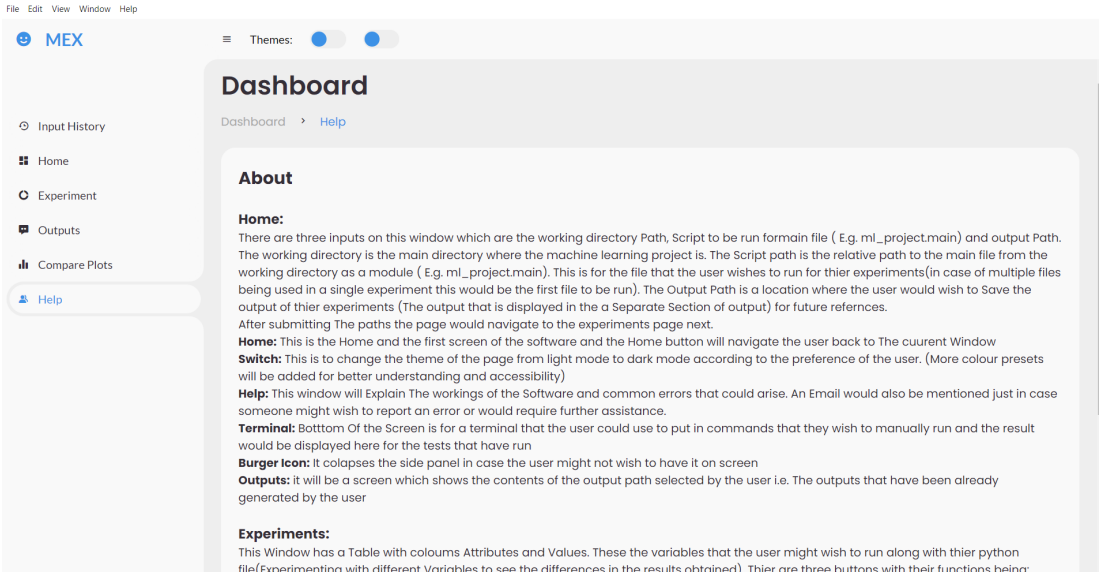
(b) Tooltips while not hovering over.



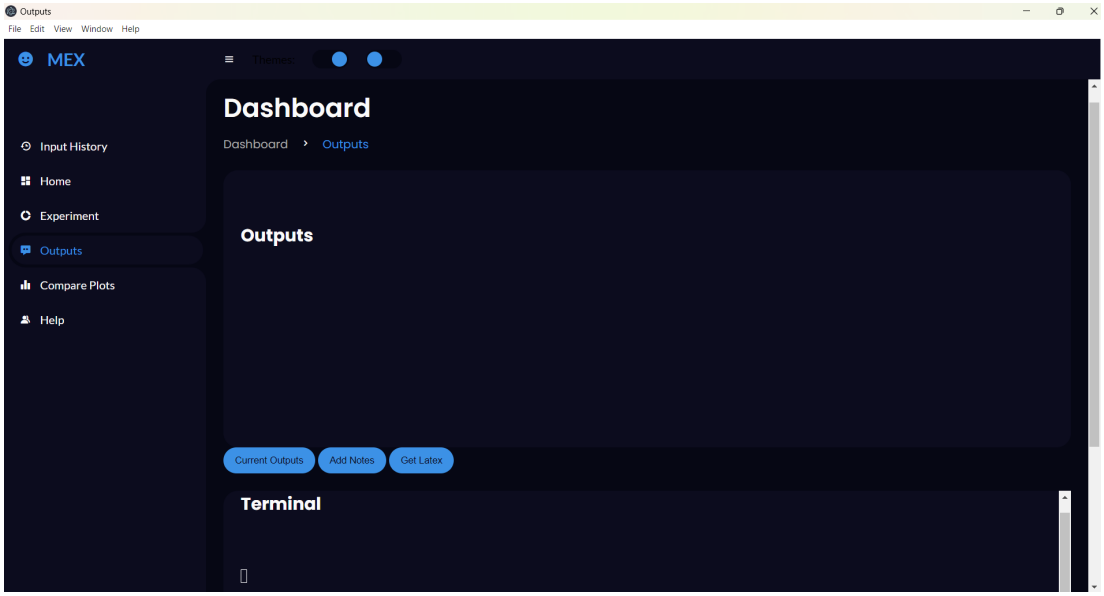
(c) Tooltip design.

(d) Alerts within MEX.

Figure A.3 Design Elements of MEX.



(a) Help documentation for MEX.



(b) Outputs page for MEX.

Figure A.4 More pages of MEX.

Appendix B

Ethics information

Put some explanatory text here to fill this page, using the headings below as a guide. Ensure you include the sample size, so if you had 10 participants you would write $n = 10$. If you are desperately short of material to fill this page, you may choose to repeat the ethics declaration by using `\input{ethics}`.

B.1 Instructions given to participants

B.2 First user study summary

User Studies 1: FeedBack for MEX

10 Responses | 08:53 Average time to complete | Active Status

1. Are you used to using ML Experiment Trackers ?

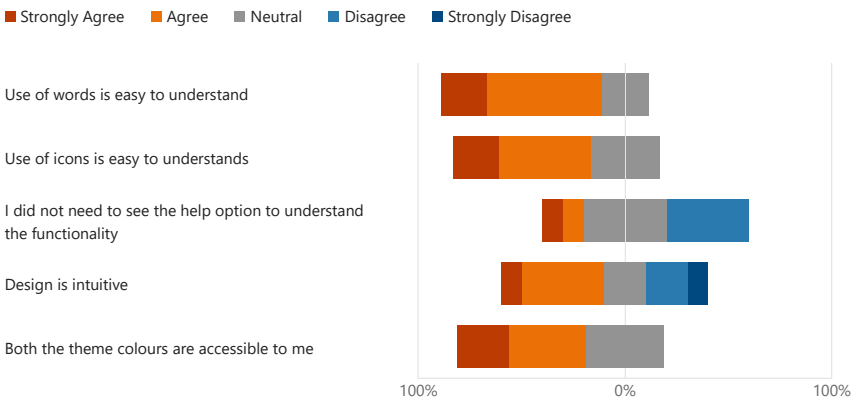
Yes	4
No	6
A Little Bit	0

2. If Yes, Please name a few tools that you usually use

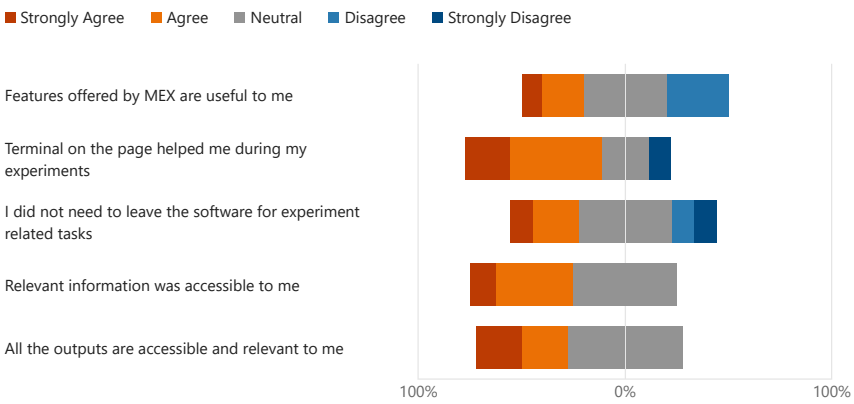
4
Responses

Latest Responses

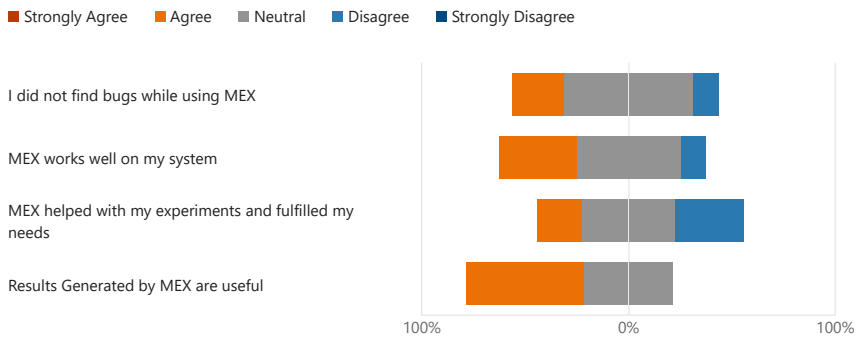
3. Please rate the interface design of MEX



4. Please rate the features of MEX

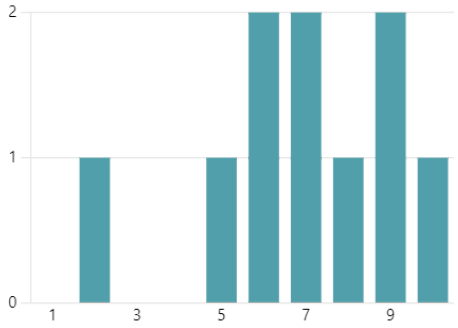


5. Please rate your user experience with MEX



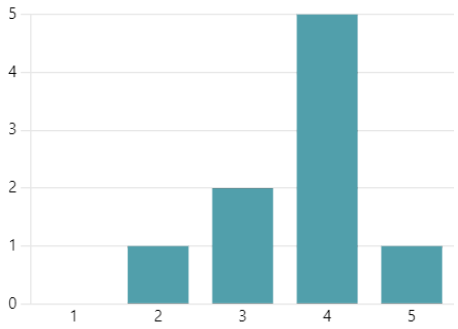
6. How likely are you to continue using MEX

6.90
Average Rating



7. How satisfied are you with the current state of MEX

3.67
Average Rating



8. Are there any Design improvements you would like to suggest for MEX

4
Responses

Latest Responses

9. Are there any features you would like to suggest to be added to MEX

4
Responses

Latest Responses

10. Is there anything you would like to mention that did or did not work about the system or the user studies?

4
Responses

Latest Responses

B.3 First user study summary

User Studies 2: FeedBack for MEX (Second Version)

10 Responses 01:15 Average time to complete Active Status

1. Were you part of the first user studies ?

Yes	10
No	0

2. Are you used to using ML Experiment Trackers ?

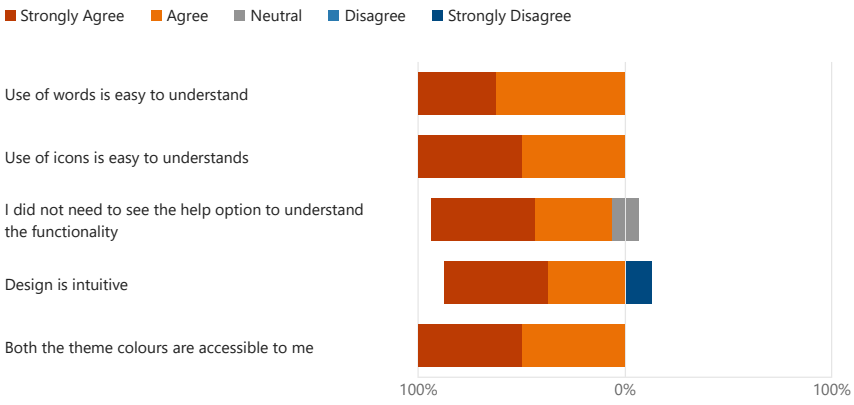
Yes	2
No	7
A Little Bit	1

3. If Yes, Please name a few tools that you usually use

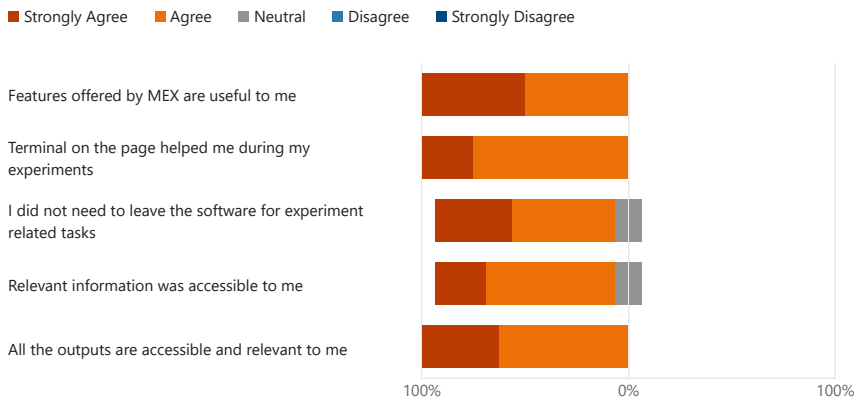
2
Responses

Latest Responses

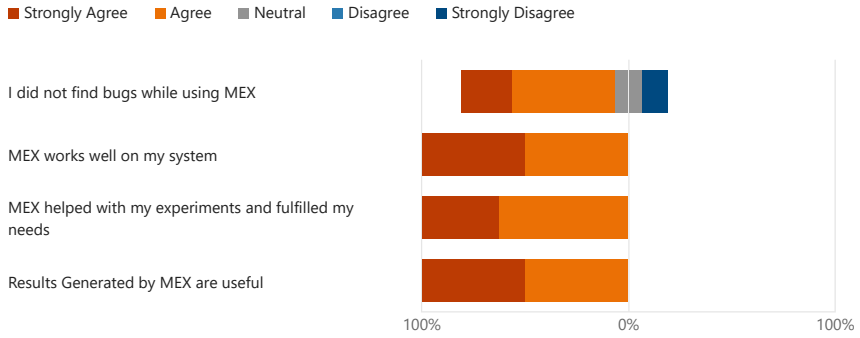
4. Please rate the interface design of MEX



5. Please rate the features of MEX

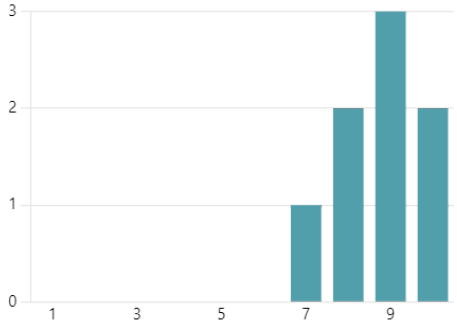


6. Please rate your user experience with MEX



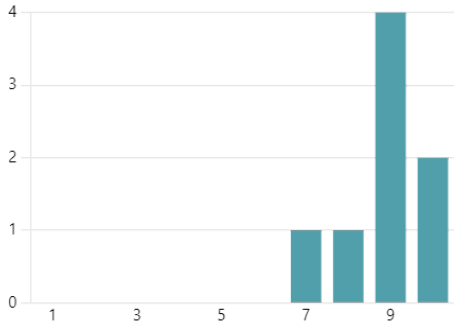
7. How likely are you to continue using MEX

8.75
Average Rating



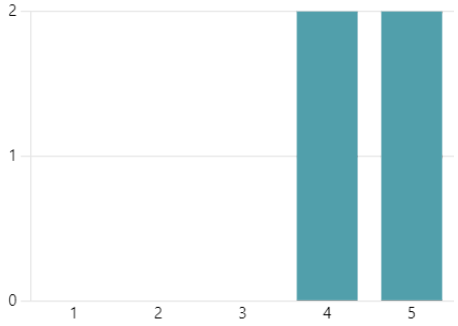
8. If you were also part of the first user studies, please rate the overall improvements the system has had

8.88
Average Rating



9. How satisfied are you with the current state of MEX

4.50
Average Rating



10. Are there any Design improvements you would like to suggest for MEX

2
Responses

Latest Responses

11. Are there any features you would like to suggest to be added to MEX

2
Responses

Latest Responses

12. Is there anything you would like to mention that did or did not work about the system or the user studies?

1
Responses

Latest Responses

B.4 Participants' information sheet

Participant Information Sheet

Project title:	System for managing experiments, data, and results
Principal investigator:	Brian Mitchell
Researcher collecting data:	Shivay Sharma
Funder (if applicable):	

This study was certified according to the Informatics Research Ethics Process, reference number 409739. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

There are two researchers a MSc Computer Science Student, Shivay Sharma, who is working on this project as his MSc Final year Project and his supervisor, Brian Mitchell, who will be supervising his work and advising where necessary.

What is the purpose of the study?

The goal of the study is to develop a system for managing data, experiments and results for machine learning. While running such experiments heaps of data is produces and the experiments are run several times with different datasets and hyperparameters. The project aims to streamline this process and help with managing these changes, for better interpretation of the results.

Why have I been asked to take part?

Participants would ideally be someone involved with running machine learning experiments, and I would like for them to use the system for their experiments, and I would work with their feedback as to what could be improved. The feedback would help me to introduce changes participants recommend and improve the functionality of the project.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, up until the submission of feedback on the project without giving a

reason. After this point, it will no longer be possible to withdraw because we are not collecting any data that would allow us to identify you.

What will happen if I decide to take part?

Only the data regarding the feedback on the software would be collected (such as user experience or possible improvements) and no other information would be kept. The data can be collected through the means of an online form, mail or an interview (Any option that would best suit the candidate will be used). The session will be of maximum 30 minutes and would be held at least once(it can be held again if the participant has more to add to their previous feedback) after the user feels that they want to give feedback on the current state of the system

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

No, There is no compensation for taking part.

What data are you collecting about me?

The data we collect for our research is completely anonymous: We are not collecting any information that could, in our assessment, allow anyone to identify you. Your signed participant consent form will be kept separately from your responses and destroyed after 23rd August 2024 (End of dissertation period).

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Your anonymised data may be published and can also be used for future research.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Shivay Sharma(s2504637@ed.ac.uk).

If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on http://web.inf.ed.ac.uk/infweb/research/study-updates.

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Shivay Sharma(s2504637@ed.ac.uk).

Participant number:_____

B.5 Participants' consent form

Participant Consent Form

Project title:	System for managing experiments, data, and results
Principal investigator (PI):	Brian Mtichell
Researcher:	Shivay Sharma
PI contact details:	brian.x.mitchell@ed.ac.uk

By participating in the study you agree that: you will use the system developed by the researcher for machine learning experiments and give feedback on its current state which will be used to improve the system and add new improvements. I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction.

- My participation is voluntary, and that I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights.
- I consent to my anonymised data being used in academic publications and presentations.
- I understand that my anonymised data will be stored for the duration outlined in the Participant Information Sheet.

Please tick yes or no for each of these statements.

1. I allow my data to be used in future ethically approved research.

Yes	**No**

2. I agree to take part in this study.

Yes	**No**

Name of person giving consent	Date dd/mm/yy	Signature

Name of person taking consent	Date dd/mm/yy	Signature