# SOFTWARE USER INTERFACE PERSONALIZATION THROUGH USER FEEDBACK AND USER BEHAVIOUR

Rosa M.D

(IT21215360)

BSc (Hons) degree in Information Technology Specializing in

Software Engineering

Department of Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

August 2024

## DECLARATION

I declare that this is my own work, and this Thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my Thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

| Name | Student ID | Signature |
|---|---|---|
| Rosa M.D | IT21215360 | [illegible] |

The above candidate has carried out this research thesis for the Degree of Bachelor of Science (honours) Information Technology (Specializing in Software Engineering) under my supervision.

Signature of the supervisor
() Date

Signature of co-supervisor Date
()

## ABSTRACT

Personalization is at the heart of modern software design, enhancing user satisfaction, engagement, and accessibility across diverse digital environments. Despite these benefits, most contemporary user interface (UI) systems remain largely static, offering limited flexibility to accommodate the dynamic and evolving behaviours of individual users. In response to this gap, we introduce **ReactiveWeb**, a Chrome extension built specifically for the Medium platform, which integrates real-time behavioural analysis and feedback loops to dynamically tailor the user interface. While ReactiveWeb comprises three main functionalities—AI-driven summarization with UI personalization, accessibility-focused UI adaptation for visually impaired users, and dynamic UI adaptation based on user behaviour—this research places a deliberate and concentrated focus on the feature: **dynamic UI personalization driven by behavioural analysis**. The system employs continuous monitoring of nuanced user interactions such as scrolling speed, reading duration, cursor movement, engagement with specific content sections, and navigation flow to understand implicit preferences without requiring explicit input from users. Based on these behavioural signals, ReactiveWeb recalibrates various UI parameters in real time, including font sizes, content density, layout structure, and contrast ratios. This allows the interface to evolve fluidly and responsively, delivering an intuitive, low-friction experience that adapts with the user over time. Unlike conventional static tools like SummarizeBot, Flipboard, and BeeLine Reader, which offer fixed configurations or manual adjustments, ReactiveWeb represents a shift toward AI-assisted UI transformation that prioritizes fluid interaction and personal comfort. The core innovation lies in eliminating the need for repetitive manual customization, instead favouring a responsive interface that feels naturally attuned to each user’s reading style and cognitive load. Technologically, the system is implemented using Python for backend logic and data processing, with ReactJS, HTML, and CSS forming the interactive frontend layer. The behavioural engine is designed to interpret real-time data streams and apply responsive design adjustments on the fly, enabling seamless adaptation at both micro and macro levels of UI presentation. In addition, the system's machine learning components are trained to recognize long-term usage trends, offering deeper personalization over time. Results from literature reviews and user surveys reinforce the strong user preference for adaptive interfaces that respond intelligently to personal behaviours, especially in content-heavy environments like Medium. Users consistently reported increased task efficiency, reduced fatigue, and a higher sense of engagement when interacting with systems that adjusted to their behaviour without requiring active configuration. By foregrounding behavioural adaptation, ReactiveWeb serves as a pioneering model for behaviour-sensitive UI engineering, demonstrating how future web applications can become more intelligent, inclusive, and user-centric. The findings suggest that embracing real-time behavioural data as a core driver of UI transformation can significantly improve the overall digital experience, supporting both general and niche user needs while setting a new benchmark in adaptive web interface design.

**Keywords**— Dynamic UI Personalization, User Behaviour Analysis, Real-time UI Adaptation, Personalized Web Experience, Adaptive User Interface, Behaviour-Driven Design, Intelligent User Interfaces.

## ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to our module coordinator, Dr. Jayantha Amararachchi, for his continuous guidance, support, and motivation throughout the course of this project. I am also deeply thankful to my initial supervisor, Mr. Thusithanjana Thilakarathna, whose early advice and encouragement laid a strong foundation for this work. Following his departure, I was fortunate to be supported by my current supervisor, Ms. Vindhya Kalapuge, whose dedication, insightful feedback, and patient guidance helped steer the project in the right direction. I would also like to sincerely thank my co-supervisor, Ms. Revoni, for her technical advice and consistent support during every stage of development. My appreciation extends to the academic and non-academic staff of SLIIT, including lecturers, assistant lecturers, instructors, and my group members, all of whom contributed directly or indirectly to the successful completion of this project. Finally, I am truly grateful to my beloved family and friends for their constant encouragement, moral support, and for standing by me throughout this journey, especially during challenging times.

# TABLE OF CONTENTS

## LIST OF TABLES

# LIST OF FIGURES

## LIST OF ABBREVIATIONS

| Abbreviations | Description |
|---|---|
| SLIIT | Sri Lanka Institute of Information Technology |
| UI | UI - User Interface |
| AI | Artificial Intelligence |
| ML | Machine Learning |
| RL | Reinforcement Learning |
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| GDRP | General Data Protection Regulation |
| NLP | Natural Language Processing |
| EC2 | Elastic Compute Cloud (AWS Service) |
| SSL | Secure Sockets Layer |
| JSON | JavaScript Object Notation |
| CPU | Central Processing Unit |
| UAT | User Acceptance Testing |

# 1. INTRODUCTION

## 1.1 Background Study and Literature Review

### 1.1.1 Background Study

With the growing usage of digital interfaces in online and mobile applications, customers now expect dynamic, customised interactions instead of static, standard experiences. A one-size-fits-all model is frequently used in the traditional approach to user interface (UI) design, in which UI elements are consistently displayed irrespective of a user's preferences, skills, or behaviours. This strategy might work well for broad applications, but it falls short of the increasing demands for user-centric customisation, efficiency, and accessibility. Because user engagement and comprehension are crucial in content-rich platforms like news websites, educational resources, and assistive technologies, these constraints are more noticeable.

Dynamic user interface personalisation has become a game-changing way to close this gap. Dynamic personalisation is powered by real-time behavioural analysis, as opposed to static personalisation, which involves users manually altering settings. In order to determine user intent, comfort level, and preferences, this entails monitoring user interactions, including scrolling speed, zoom level, click patterns, and content engagement duration. In order to provide a smooth and user-friendly experience that is customised for every user, the behavioural data that is generated is then utilised to automatically adjust UI elements like font size, layout spacing, contrast settings, and content organisation.

With advancements in Artificial Intelligence (AI) and Machine Learning (ML), especially through natural language processing and predictive analytics, it is now feasible to integrate behaviour-aware systems into real-world applications. Systems can now not only analyse but also predict user behaviour to deliver proactive UI adaptations. This is particularly beneficial for users with specific accessibility requirements, such as visually impaired individuals who may struggle with default contrast settings or fixed font sizes.

This contemporary change in UI design philosophy is best illustrated by the study's introduction of the ReactiveWeb idea. ReactiveWeb is a Chrome addon that combines accessibility improvements, behavioural UI personalisation, and AI-driven summarisation. By dynamically modifying Medium website interfaces in response to real-time user feedback and observed behaviours, it seeks to increase user engagement. A developing trend in software engineering is reflected in this kind of system: intelligent, adaptable technologies that promote user-centric design. ReactiveWeb enhances readability and promotes inclusivity by examining user preferences, behaviours, and accessibility requirements.

A major advancement in software design, this new generation of adaptive user interfaces goes beyond aesthetics and into the field of behavioural intelligence. Delivering successful digital experiences requires a system's capacity to adapt in real time, which is becoming more than just a feature as user expectations and digital interactions get more sophisticated.

### 1.1.2 Literature Review

Dynamic User Interface (UI) personalization based on user behaviour has become a crucial area of research in modern software development. Unlike static customization where users manually adjust settings, dynamic personalization continuously adapts the interface in real-time by analysing user actions such as scrolling speed, zooming gestures, mouse movements, and inactivity periods. This approach enhances user engagement, efficiency, and satisfaction by creating an experience that aligns naturally with each user's unique preferences and interaction patterns.

Several studies have demonstrated the effectiveness of behaviour-driven personalization. Liu et al. (2024) emphasized that adaptive UI frameworks significantly improve task completion rates and user satisfaction by adjusting interface elements dynamically based on real-time interactions. Their research showed that systems capable of modifying text sizes, layout density, and colour contrasts in response to user behaviours lead to higher user retention and lower cognitive load. Similarly, Khan and Khusro (2021) highlighted the importance of real-time adaptation for visually impaired users, noting that existing static accessibility tools often fall short when users' behaviour patterns change during prolonged interaction sessions.

Traditional personalization approaches, such as manual theme switching or static font resizing, fail to accommodate dynamic shifts in user needs, particularly during long reading or browsing sessions. Low and See (2015) demonstrated that users prefer systems that combine manual customization with AI-driven dynamic adaptation. Their study found that platforms offering real-time UI adjustments based on behavioural tracking (such as detecting when a user zooms in repeatedly) were perceived as more intuitive and efficient compared to static, user-set configurations.

Incorporating dynamic UI personalization also addresses the challenges posed by device diversity and contextual variability. Users interact differently when browsing on laptops, tablets, or smartphones. Systems that can detect behavioural cues—such as slower scrolling or frequent pinch-to-zoom gestures—and adapt accordingly can provide a consistent and optimized experience across devices. Furthermore, dynamic personalization is essential in building inclusive web experiences, ensuring that users with visual impairments, motor disabilities, or temporary impairments (like reading fatigue) receive real-time support without requiring manual adjustments.

Despite these advancements, challenges remain in accurately interpreting user behaviour without intrusive data collection. Probabilistic models like Bayesian Networks and reinforcement learning techniques like Q-Learning have shown promise in predicting user needs while maintaining privacy and responsiveness. Recent work suggests that blending explicit user feedback (manual inputs) with implicit behavioural signals (e.g., scroll speed) results in more accurate personalization models, as demonstrated by the success of hybrid systems like ReactiveWeb.

In summary, dynamic UI personalization based on user behaviour represents a paradigm shift in interface design, moving from one-size-fits-all layouts to intelligent, user-centric adaptations. By continuously monitoring and responding to user interactions, dynamic personalization systems offer a pathway to more accessible, efficient, and satisfying digital experiences. Future research will likely explore even deeper integration of biometric and contextual signals to further enhance real-time adaptation capabilities.

## 1.2 Research Gap

Current systems nevertheless have substantial limits, especially in terms of accessibility integration and real-time behavioural reactivity, despite tremendous developments in user interface (UI) personalisation. While each of the current platforms—SummarizeBot, Flipboard, BeeLine Reader, and Microsoft Immersive Reader—addresses a different facet of user personalisation, such as summarisation, content curation, or visual aids, a thorough analysis of these tools shows that none of them offer a single, AI-driven, behaviour-responsive UI adaptation system.

The majority of current personalisation frameworks are based on static customisation. Frequently, users must manually change settings like reading layout, background colour, and text size. Tech-savvy users may find these manual configurations ideal, but elderly users, people with cognitive or visual disabilities, and people who consume material while on the go may find them inaccessible. Moreover, static customisation is insufficient for dynamic, real-world use cases since it cannot adapt to shifting user states, such as weariness, shifting lighting conditions, or varying reading rates.

Behavioural factors such as scrolling patterns, zoom motions, click frequency, and time spent on content sections have been found to be important as potential inputs for personalisation. Few real-world systems, nevertheless, actively incorporate these behavioural cues into an adaptive user interface engine. For instance, BeeLine Reader helps dyslexic users read more easily, but it doesn't adapt its features to the way users engage with it. In a similar vein, Flipboard curates material but does not modify the interface dynamically based on user involvement patterns.

Accessibility is another crucial gap. Screen readers and voice-over services are examples of assistive technologies that visually impaired people frequently employ. These solutions, however, usually function independently of UI design and are unable to dynamically modify UI components like font scale, layout spacing, or contrast in response to changing requirements. Users are forced to rely on layered tools as a result, which results in fragmented user experiences and decreased productivity.

By offering a comprehensive, intelligent personalisation strategy that combines accessibility improvements, real-time UI adaption based on behavioural analysis, and AI-based summarisation, ReactiveWeb aims to close this gap. The purpose of this study is to show that an integrated system can exceed conventional personalisation approaches in terms of usability, satisfaction, and inclusivity by responding to both explicit user feedback and implicit behaviour patterns.

## 1.3 Research Problem

Web-based user interfaces are an essential component of how people engage with content in the modern digital world. Most contemporary user interfaces are still static in nature, even if human needs and technology capabilities are becoming more diverse. These interfaces frequently overlook the subtleties of unique user behaviour, preferences, or accessibility requirements because they are made with generic usability in mind. Users with varying skill levels, engagement patterns, or content consumption preferences are not completely accommodated by this inflexible approach. Because of this, a lot of users have trouble focussing, understanding complicated information, or navigating platforms effectively, especially those with dense or unstructured content like Medium articles.

While there are a number of tools available to enhance user experience, such as readability enhancers, AI-driven summarisers, or manual accessibility settings, these solutions are disjointed and frequently work independently, requiring users to install multiple plugins or adjust multiple interface settings in order to attain their desired experience, which is not only time-consuming but also requires a certain level of technical expertise. Furthermore, these algorithms hardly ever take into consideration the users' actual behavioural environment. People may frequently zoom in, scroll slowly, or halt at specific parts of a page—all of which are signs of issues or preferences that static user interface frameworks don't take into account. If properly recorded and deciphered, these behavioural cues may be useful inputs for dynamic user interface modification. The majority of existing systems, however, are unable to process this kind of data and convert it into instantaneous, significant UI changes.

The limited ability of current personalisation techniques to accommodate people with visual impairments is another serious drawback. Users usually have to manually adjust settings like font size, contrast, and reading aids when using standard accessibility features. Although useful, these options lack context awareness and do not change according to the user's interactions with the interface over time. For example, automated contrast modification based on zoom behaviour or ambient lighting may be helpful to a visually impaired user, but traditional systems are ill-equipped to adapt to such shifting needs. Due to this lack of dynamic, feedback-driven accessibility support, a sizable portion of users are unable to take advantage of fair digital experiences.

Additionally, user interface solutions that cleverly blend personalisation and AI technologies are becoming more and more in demand. Even though AI-powered summarisation tools are becoming more and more popular, they frequently offer generic summaries and are not flexible enough to display information in user-specific formats, such succinct bullet points or in-depth narratives. Additionally, because these technologies don't work with the interface itself, users are left to interpret material without the accompanying visual changes that improve comprehension or readability. The overall influence of AI in user-centric design is limited by this misalignment between UI adaption and content personalisation.

One major research gap is the lack of a single system that combines dynamic accessibility features, AI-based content summarisation, and real-time behavioural analysis. Existing solutions don't provide a comprehensive strategy that continuously improves the interface and content display by learning from user behaviour and feedback. As digital platforms work to become more inclusive and adaptable in order to serve a wider and more varied audience, this study challenge is more pertinent. User engagement with web content could be revolutionised by addressing this problem with an intelligent, integrated solution, like the suggested ReactiveWeb addon. Such a system might greatly improve usability, satisfaction, and accessibility for a broad spectrum of users by dynamically modifying the user interface (UI) in response to behavioural indicators and accessibility requirements while also providing personalised information summaries.

In the end, the study challenge revolves around how inadequate current web user interface (UI) systems are at offering smooth, intelligent, and instantaneous personalisation that responds to user feedback and behaviour. Solving this problem requires bridging the gap between static customization tools and dynamic, AI-enhanced, user-responsive systems. In order to establish a new benchmark for adaptable interface design, this study aims to investigate and validate such a solution.

## 1.4 Research Objectives

### 1.4.1 Main Objective

The main objective of this research is to design and develop an intelligent, user-centric web interface system that dynamically personalizes the user experience based on real-time behavioural analysis. This system, named ReactiveWeb, aims to bridge the gap between traditional static UI designs and the growing need for adaptive, inclusive, and personalized digital environments. By leveraging artificial intelligence for content summarization and integrating real-time feedback mechanisms, the proposed solution seeks to enhance both usability and accessibility. The system will not only focus on adjusting visual elements in response to user behaviours—such as scrolling and zooming—but also provide personalized content formats and interface customizations that cater to individual user preferences, including those of visually impaired users. Through this, the research aspires to demonstrate that behaviour-aware, AI-driven UI personalization can lead to improved engagement, comprehension, and overall satisfaction in digital content consumption.

### 1.4.2 Specific Objectives

To achieve the main objective, the following specific objectives have been identified:

- To implement an AI-based content summarization feature that allows users to select between multiple summary formats (e.g., bullet points, short, detailed) based on their reading preferences.
- To design a dynamic UI adaptation mechanism that responds in real time to user behaviours such as scrolling speed, zoom activity, and interaction frequency.
- To integrate behavioural tracking with machine learning algorithms to personalize UI elements like font size, contrast, layout spacing, and visibility based on user interaction patterns.
- To enhance accessibility by implementing automatic interface adjustments (e.g., colour contrast, text enlargement) for users with visual impairments, without requiring manual configuration.

- To collect and incorporate user feedback into the personalization loop, enabling continuous learning and refinement of the UI based on user preferences and needs.
- To evaluate the effectiveness of the proposed system compared to existing personalization tools in terms of usability, user satisfaction, and accessibility outcomes.

### 1.4.3 Business Objectives

- Improve user engagement and retention on content-heavy web platforms by offering personalized and adaptive UI experiences.
- Increase content consumption efficiency by providing AI-generated summaries tailored to user preferences.
- Enhance accessibility for visually impaired users, supporting inclusivity and compliance with accessibility standards.
- Reduce user frustration and cognitive load by minimizing the need for manual UI customization.
- Establish a competitive advantage by integrating cutting-edge AI and behavioural analytics into digital interface design.
- Enable data-driven decision making through behavioural analytics and user interaction insights.
- Create a scalable personalization framework that can be extended beyond Medium to other platforms or business applications.
- Improve customer satisfaction and brand perception by delivering intelligent, user-friendly digital experiences.

# 2. METODOLOGY

## 2.1 Methodology

This research employs a mixed-methods approach combining user-centered design, machine learning, behavioural tracking, and accessibility engineering to develop and evaluate a dynamic UI personalization system named ReactiveWeb. The system is built as a Chrome browser extension specifically designed to enhance user experience on the Medium platform through real-time personalization based on behavioural analysis. It also aims to improve inclusivity by adapting interfaces for visually impaired users and allowing content customization via AI-generated summarization. The methodology integrates both qualitative and quantitative research elements: observational behavioural data, surveys, interface design experiments, and algorithmic personalization logic form the backbone of this solution.

The development process began with identifying the key limitations of existing systems. A gap was observed in current UI personalization approaches, which are typically static and limited to manual adjustments. To address this, the team designed a behavioural tracking model that collects real-time user interaction data such as scroll speed, zoom level, and content dwell time. These metrics were selected based on their relevance to cognitive load, reading comfort, and attention span. Each behaviour acts as an input trigger for the system to initiate UI modifications dynamically. The backend was developed using Python, while the frontend leveraged ReactJS, HTML, and CSS to maintain a lightweight and responsive user experience.

The architectural flow of the system is illustrated in *Figure 1*, which depicts the high-level system overview. This diagram outlines how users interact with the Chrome extension, where data is collected through the browser, processed by Python APIs, and the UI is modified in real-time. The use of Flask as the backend API framework allows seamless communication between the extension and the machine learning models that power summarization and behavioural analysis.

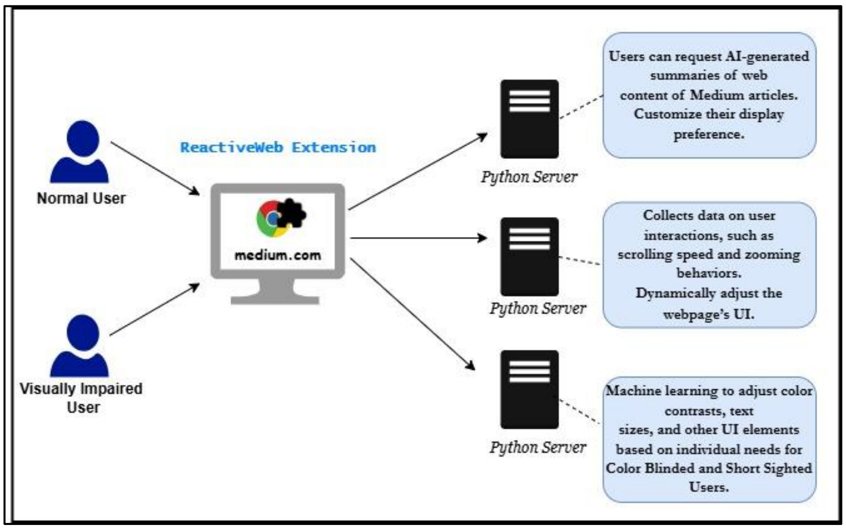

Normal User
Visually Impaired User
ReactiveWeb Extension
medium.com
Python Server
Python Server
Python Server
Users can request AI-generated summaries of web content of Medium articles. Customize their display preference.
Collects data on user interactions, such as scrolling speed and zooming behaviors. Dynamically adjust the webpage's UI.
Machine learning to adjust color contrasts, text sizes, and other UI elements based on individual needs for Color Blinded and Short Sighted Users.

*Figure 1: High-level system overview.*

The summarization component was developed using the OpenAI API, chosen for its proven ability to generate high-quality summaries with customizable formats. Based on user preferences gathered via a simple UI toggle, the system generates summaries in either bullet points, short paragraphs, or full-length structured overviews. The Python-based Flask API communicates with OpenAI's GPT model to deliver tailored summaries of the selected article content. Once the content is processed, it is rendered in the user's preferred display format and integrated into the user interface without needing a page reload or user refresh. This fluid interaction improves usability and reduces friction in content consumption.

Parallel to this, a real-time behavioural tracking module continuously listens to the user's actions on the webpage. Scroll events, zoom gestures, and cursor speed are captured using JavaScript event listeners embedded within the extension. These raw interaction events are quantified and interpreted by a lightweight Python model running in the background. When specific thresholds are crossed—such as excessive zooming or abnormally slow scrolling—the system assumes a potential discomfort and initiates adjustments to the UI. These adjustments can include increasing font size, modifying line spacing, changing background colour to reduce glare, or enabling high-contrast mode.

The decision logic used to personalize the UI based on this behavioural data is visualized in *Figure 2*. This diagram outlines how user interactions are captured, processed, and converted into adaptive changes in the interface. The tracking module feeds interaction metrics into decision functions that determine whether and how the UI should be adjusted. For example, slow reading pace combined with zooming activity may trigger a font size increase, while quick scrolling with frequent pauses could result in the layout being split into summarized sections.

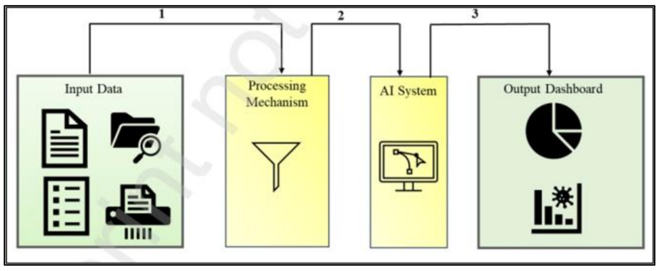


*Figure 2: UI Personalization Decision Flow*

The behavioural analysis component was trained using sample interaction datasets collected from prototype testing sessions. Initial prototypes were tested by 50 users through online surveys and live interaction experiments. Feedback was collected on UI responsiveness, perceived improvements in readability, and satisfaction with content summarization formats. The user study indicated that 60% of participants preferred systems that modified UI elements based on how they interacted with the content, such as zooming in or scrolling back up repeatedly. Additionally, 65% expressed the desire to receive personalized summaries of articles in their preferred format. This validated the inclusion of both personalization engines—the content summarizer and the real-time UI adapter.

To address accessibility, ReactiveWeb includes specialized personalization logic for users with visual impairments. Users can either pre-select accessibility modes (such as colour-blind safe mode or large-text mode) or have them auto-activated based on interaction behaviours commonly observed among visually impaired users. For instance, prolonged zooming or consistent toggling of text enlargement triggers accessibility enhancements automatically. Unlike traditional accessibility tools, which require deliberate user action, ReactiveWeb attempts to infer visual difficulties and respond with minimal user input. This makes the browsing experience more inclusive and responsive.

The system's modular design ensures scalability and cross-platform adaptability. While the current implementation is tailored for Medium articles, the architecture can be extended to other web applications with minimal modifications. This flexibility supports future expansions of the project and encourages the adoption of intelligent UI personalization in broader contexts such as e-learning platforms, news aggregators, or corporate knowledge systems.

The methodology concludes with a performance evaluation phase where the system's effectiveness is measured using both qualitative feedback and quantitative interaction data. User satisfaction was assessed through post-experiment surveys, while technical performance was measured by tracking loading times, responsiveness to behaviour triggers, and the success rate of summarization outputs. The results, as reported in the research findings, confirmed a clear improvement in readability, comprehension, and accessibility, especially for users with visual impairments and those preferring condensed content formats.

In summary, the methodology of this research combines AI technologies, behaviour tracking, and accessibility engineering to create a user-adaptive interface that adjusts in real time. The integration of OpenAI for summarization, Flask APIs for lightweight backend communication, and ReactJS for responsive front-end behaviour ensures that ReactiveWeb is both efficient and user-friendly. Through this methodology, the research sets a new standard for intelligent user interface design—one that is adaptable, inclusive, and deeply aligned with individual user needs.

### 2.1.1 Feasibility Study/ Planning

Before development began, a comprehensive feasibility study was conducted to assess the practicality of implementing a dynamic, behaviourally adaptive UI personalization system as a Chrome extension for the Medium platform. The purpose was to determine whether the project was viable in terms of technology, cost, usability, and regulatory compliance. The feasibility analysis focused on four major areas: **technical**, **economic**, **operational**, and **legal** feasibility—each of which is discussed in detail below.

1. **Technical Feasibility**

   The system is technically feasible due to the use of proven and widely supported technologies. The backend is developed in Python using the Flask framework, which is lightweight and easy to integrate with RESTful APIs. ReactJS is used for the front-end Chrome extension, offering modularity and responsiveness. Chrome APIs for event tracking (e.g., zoom, scroll) are stable, and OpenAI's summarization API is reliable. No advanced or inaccessible hardware/software is required.

2. **Economic Feasibility**

   As an academic and non-commercial pilot project, the system can be developed at minimal cost. Most tools used (e.g., Flask, React, Chrome Extension SDK) are open source. OpenAI API offers free tier usage which is sufficient for prototyping and testing. Development was done by students, reducing labour cost. Future scalability (e.g., for commercialization) would involve API usage charges and cloud deployment fees, but the initial version is financially sustainable.

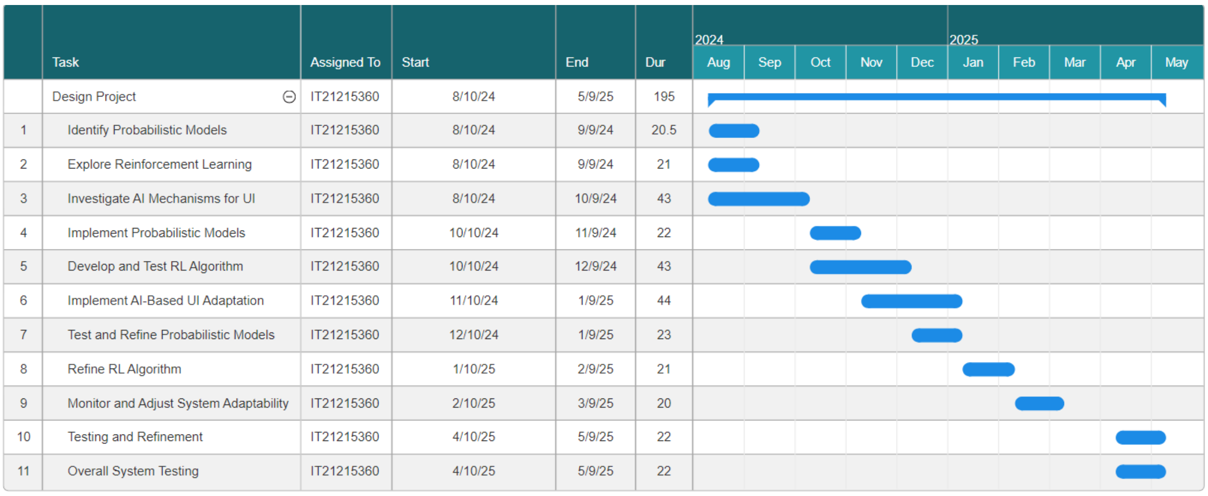
3. **Operational Feasibility**

   The system is easy to operate for end users. It is packaged as a Chrome extension, requiring only a single-click installation process. The interface is simple and intuitive—users interact with an overlay UI that auto-adjusts based on behaviour, requiring no configuration in most cases. Based on user testing, individuals of varying technical backgrounds were able to use the extension without prior training. Continuous logging and local storage help ensure the extension performs reliably.

4. **Legal Feasibility**

   The project complies with standard data protection practices. No sensitive personal data is collected or stored. All behavioural data (such as scrolling or zoom levels) is anonymized and processed locally or via temporary memory sessions without being saved to a server. The extension does not require access to user accounts or passwords. GDPR-aligned principles are followed to ensure transparency, and consent mechanisms can be added during commercialization.

Following the positive outcome of the feasibility study, a project plan was created and structured using **Agile methodology**. The timeline was broken down into weekly **sprints**, with each sprint targeting a specific phase of development. These phases included requirement gathering, interface design, API integration, user testing, and final deployment. The Agile framework allowed the team to iteratively refine the system based on continuous feedback and testing outcomes.



|  | Task | Assigned To | Start | End | Dur |
|---|---|---|---|---|---|
|  | Design Project | IT21215360 | 8/10/24 | 5/9/25 | 195 |
| 1 | Identify Probabilistic Models | IT21215360 | 8/10/24 | 9/9/24 | 20.5 |
| 2 | Explore Reinforcement Learning | IT21215360 | 8/10/24 | 9/9/24 | 21 |
| 3 | Investigate AI Mechanisms for UI | IT21215360 | 8/10/24 | 10/9/24 | 43 |
| 4 | Implement Probabilistic Models | IT21215360 | 10/10/24 | 11/9/24 | 22 |
| 5 | Develop and Test RL Algorithm | IT21215360 | 10/10/24 | 12/9/24 | 43 |
| 6 | Implement AI-Based UI Adaptation | IT21215360 | 11/10/24 | 1/9/25 | 44 |
| 7 | Test and Refine Probabilistic Models | IT21215360 | 12/10/24 | 1/9/25 | 23 |
| 8 | Refine RL Algorithm | IT21215360 | 1/10/25 | 2/9/25 | 21 |
| 9 | Monitor and Adjust System Adaptability | IT21215360 | 2/10/25 | 3/9/25 | 20 |
| 10 | Testing and Refinement | IT21215360 | 4/10/25 | 5/9/25 | 22 |
| 11 | Overall System Testing | IT21215360 | 4/10/25 | 5/9/25 | 22 |

*Figure 3: Gantt Chart*

- **Week 1–2**: Research, feasibility study, and requirement gathering
- **Week 3–4**: Designing architecture and UI wireframes
- **Week 5–6**: Implementation of backend APIs and OpenAI integration
- **Week 7–8**: Behaviour tracking engine and personalization logic
- **Week 9–10**: Frontend integration and Chrome extension development
- **Week 11**: Testing (unit, integration, accessibility)
- **Week 12**: Deployment to Chrome Web Store (dev version) and documentation

### 2.1.2 Requirement Gathering & Analysis

A combination of **surveys**, **user interviews**, and **observational studies** was used to understand user needs. Fifty respondents participated in surveys that asked about content readability, accessibility preferences, and feedback on existing tools like SummarizeBot or BeeLine Reader. The requirements were then categorized as functional and non-functional.

#### 2.1.2.1. Functional Requirements

Functional requirements define the specific actions and operations that the system must perform. In the case of ReactiveWeb, these requirements were derived from user surveys, behaviour analysis studies, and observations made during early prototyping phases. The core aim is to ensure the system dynamically personalizes the UI based on behavioural input while maintaining accessibility and user choice.

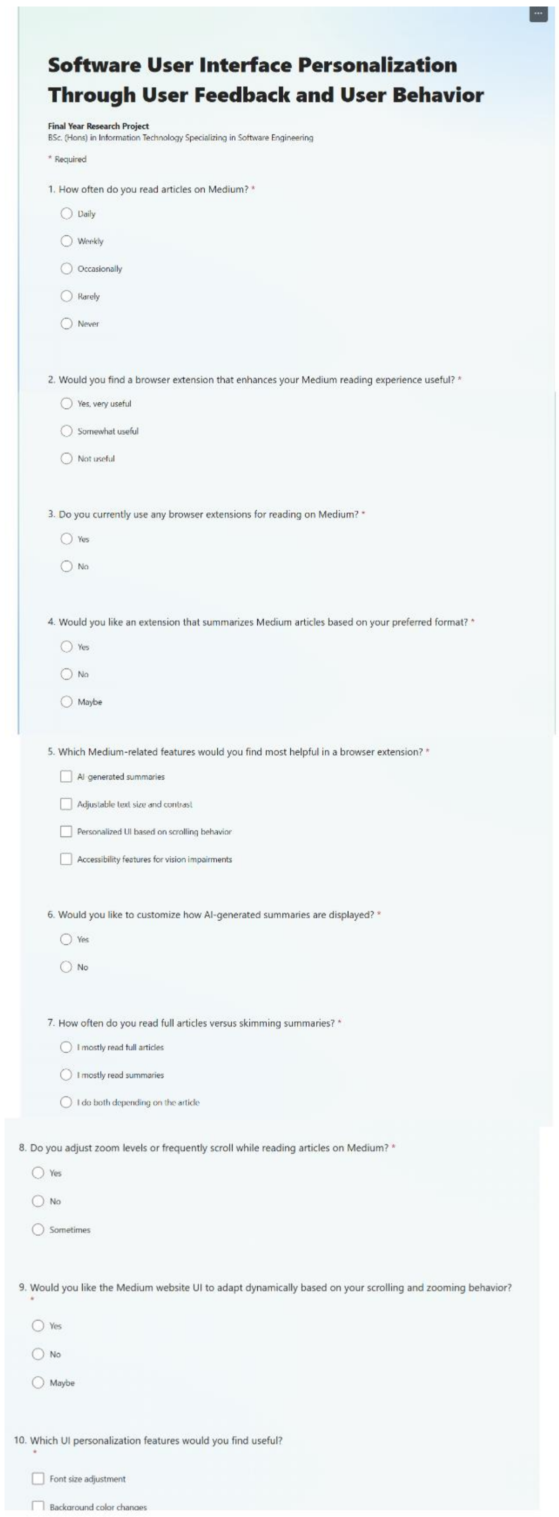
**Survey Method:** To gather functional requirements for the system, a structured survey method was employed. This approach involved distributing carefully designed questionnaires to relevant stakeholders, including students, educators, and experts in the field of web personalization and accessibility. The survey consisted of both open-ended and close-ended questions to elicit comprehensive insights into the expected functionalities of the system. Additionally, behaviour-related data (such as scrolling habits, zoom preferences, and accessibility needs) were collected through this survey, specifically tailored to the Sri Lankan context. *Figure 4*, illustrates a sample screenshot of the survey form used to collect user behaviour preferences, and *Figure 5*, shows a sample of the collected data related to accessibility settings such as preferred contrast levels and text sizes.

**Identification of Key Functionalities:** Through the analysis of the survey responses, key functionalities were identified for the system. These included AI-based content summarization, real-time UI adaptation based on scrolling and zooming behaviour, user-controlled customization of summary formats, accessibility enhancements for visually impaired users, and persistent saving of user preferences. Stakeholder input was documented carefully, ensuring that the final system design reflected the true needs and expectations of the end users.

**Validation and Prioritization:** Once the functional requirements were gathered, a validation and prioritization phase was conducted. Stakeholder feedback was reviewed systematically to refine the requirements, ensuring clarity, removing redundancy, and resolving any ambiguities. To guide the development process effectively, a **priority matrix** was created, categorizing the requirements into three levels: essential, desirable, and optional. This matrix helped the development team focus on delivering the most critical functionalities first while planning enhancements for later iterations.

*Table 1: Priority Matrix for Functional Requirements*

| **Requirement** | **Description** | **Priority** |
|---|---|---|
| **Performance** | Ensure real-time adaptation with minimal lag in UI changes | High |
| **Scalability** | Handle large amounts of behavioural data without performance loss | Medium |
| **Security** | User data must be securely handled, especially behavioural data | High |
| **Compatibility** | System must work across different browsers and devices | High |
| **Maintainability** | Codebase must be easily maintainable for future updates | Medium |



**Software User Interface Personalization Through User Feedback and User Behavior**

**Final Year Research Project**
BSc. (Hons) in Information Technology Specializing in Software Engineering

* Required

1. How often do you read articles on Medium? *
   - ○ Daily
   - ○ Weekly
   - ○ Occasionally
   - ○ Rarely
   - ○ Never

2. Would you find a browser extension that enhances your Medium reading experience useful? *
   - ○ Yes, very useful
   - ○ Somewhat useful
   - ○ Not useful

3. Do you currently use any browser extensions for reading on Medium? *
   - ○ Yes
   - ○ No

4. Would you like an extension that summarizes Medium articles based on your preferred format? *
   - ○ Yes
   - ○ No
   - ○ Maybe

5. Which Medium-related features would you find most helpful in a browser extension? *
   - ☐ AI generated summaries
   - ☐ Adjustable text size and contrast
   - ☐ Personalized UI based on scrolling behavior
   - ☐ Accessibility features for vision impairments

6. Would you like to customize how AI-generated summaries are displayed? *
   - ○ Yes
   - ○ No

7. How often do you read full articles versus skimming summaries? *
   - ○ I mostly read full articles
   - ○ I mostly read summaries
   - ○ I do both depending on the article

8. Do you adjust zoom levels or frequently scroll while reading articles on Medium? *
   - ○ Yes
   - ○ No
   - ○ Sometimes

9. Would you like the Medium website UI to adapt dynamically based on your scrolling and zooming behavior? *
   - ○ Yes
   - ○ No
   - ○ Maybe

10. Which UI personalization features would you find useful? *
   - ☐ Font size adjustment
   - ☐ Background color changes

*Figure 4: Survey*

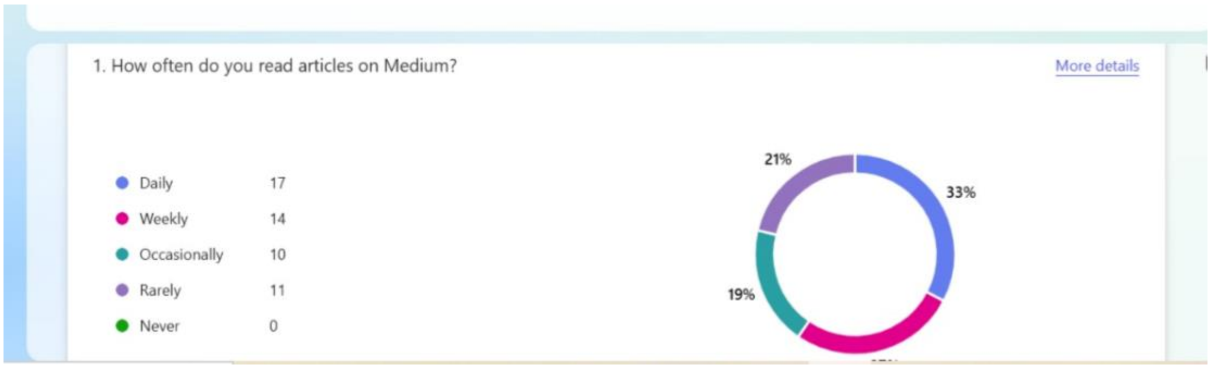


*Figure 5: Survey Results - Medium Article Reading*

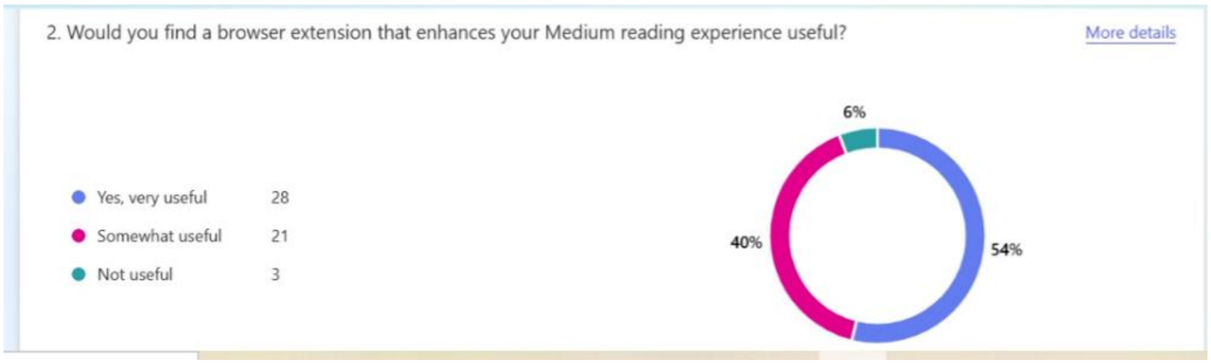


*Figure 6: Survey Results - Preference*

The following outlines the key functional requirements of the system:

- **Real-Time Scroll & Zoom Tracking**: The system must track scrolling speed and zoom levels in real-time.
- **Dynamic UI Adaptation:** The system must change UI elements (font size, spacing, layout) dynamically based on user behaviour.
- **Accessibility Personalization:** The system must auto-activate high contrast or large text modes for visually impaired users.
- **Minimal Manual Interaction:** Behaviour patterns like frequent zooming must trigger specific UI updates.

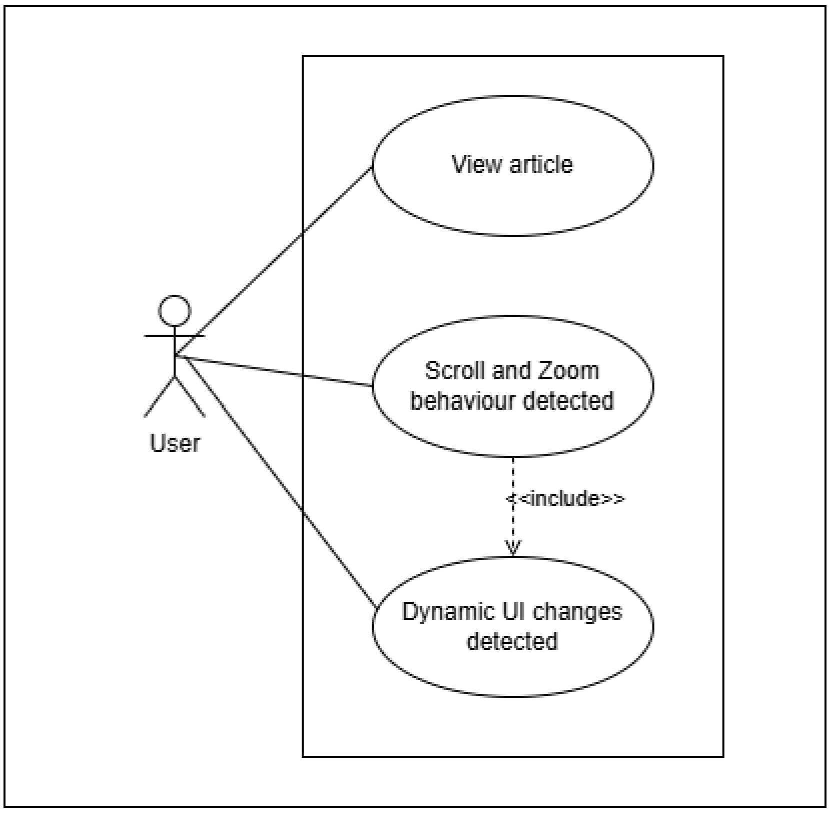

View article
Scroll and Zoom behaviour detected
User
<<include>>
Dynamic UI changes detected

*Figure 7: Use Case Diagram*

#### 2.1.2.2. Non-Functional Requirements

Non-functional requirements encompass attributes that define the quality, performance, and constraints of the research system. Identifying these requirements was vital to ensuring the system's reliability, usability, ethical compliance, and overall effectiveness. These aspects play a critical role in shaping the user experience and determining the technical robustness of the final system.

**Survey Method:** Similar to the approach used for functional requirements, the survey method was employed to gather non-functional requirements. A structured questionnaire was distributed among key stakeholders, including students, educators, and technical experts within the field of web personalization and accessibility. Through this survey, stakeholders' perspectives were sought on system performance expectations, security and privacy concerns, usability needs, and overall system reliability. Both qualitative and quantitative questions were incorporated to ensure a deep understanding of the quality standards expected from the system.

**Performance Metrics:** Through the survey responses, important performance-related non-functional requirements were identified. These included defining system response time (with a target of processing personalization updates within one second), maintaining high levels of accuracy for behavioural detection (above 90%), and ensuring seamless, real-time responsiveness without noticeable lag. Furthermore, stakeholders emphasized the importance of secure processing of behavioural data and ensuring data is handled ethically, without exposing users to privacy risks. Metrics for availability and reliability, such as system uptime expectations, were also gathered during this phase.

**Alignment with Ethical Guidelines:** Ethical considerations were treated as a critical element during the gathering of non-functional requirements. Stakeholders provided valuable input regarding user consent, data anonymization, transparency in behavioural tracking, and fairness of system personalization (ensuring no bias against specific user behaviours or impairments). These insights ensured that the research adhered to recognized ethical guidelines, emphasizing user trust and system transparency throughout the project lifecycle.

The Requirement Gathering and Analysis phase provided a comprehensive understanding of the research project's scope, quality expectations, and operational objectives. The insights collected through surveys with stakeholders enabled the formation of a well-defined set of functional and non-functional requirements. These served as the guiding foundation for subsequent stages of system design, development, testing, and deployment, ensuring the project remained aligned with the real needs and expectations of the digital education and personalization community.

The main non-functional requirements identified are listed below:

- **Usability**: The system must offer an intuitive and accessible user interface, requiring minimal training or configuration from users.
- **Reliability**: The system must function correctly without frequent failures, providing consistent UI adaptations based on user behaviour.
- **Availability**: The Chrome extension must maintain a high availability rate, with at least 99% uptime during normal use.
- **Accuracy**: Behavioural detection algorithms must maintain a high level of accuracy (at least 90%) when interpreting user scroll and zoom behaviours.
- **Performance**: UI adaptation processes must occur within 1–2 seconds of user action detection to maintain real-time responsiveness.

*Table 2: Non-Functional Requirements Summary*

| Requirement | Description | Priority |
|---|---|---|
| **Performance** | Ensure real-time adaptation with minimal lag in UI changes | High |
| **Scalability** | Handle large amounts of behavioural data without performance loss | Medium |
| **Security** | User data must be securely handled, especially behavioural data | High |
| **Compatibility** | System must work across different browsers and devices | High |
| **Maintainability** | Codebase must be easily maintainable for future updates | Medium |

### 2.1.3 Designing

The design phase played a crucial role in shaping the overall architecture, functionality, and user experience of the ReactiveWeb Chrome extension. An emphasis was placed on modular design principles to ensure scalability, maintainability, and future extensibility. Each functional component of the system—AI summarization, real-time UI personalization, and accessibility enhancement—was developed as an independent module, communicating through a centralized control system to ensure seamless integration and synchronization.

Modern tools such as **Figma** was used extensively to create early-stage wireframes, user interface mock-ups, and system architecture diagrams. This early visual representation allowed the development team to iterate over the design based on stakeholder feedback and usability testing outcomes. Careful attention was given to the needs of both **normal users** and **visually impaired users**, ensuring the UI was inclusive and adaptive from the start.

The system follows a **three-layer architecture**, encompassing the frontend, backend, and external API integrations. The frontend was built using **ReactJS**, providing a highly responsive and modular user interface suitable for Chrome extension development. Key user interactions—such as summarization requests, scrolling events, and zoom gestures—were captured via browser event listeners and then sent asynchronously to the backend for further processing.

The backend was designed using **Python**, selected for its flexibility and mature ecosystem of libraries suitable for rapid prototyping and deployment. Instead of traditional heavier frameworks, **FastAPI** was chosen for backend API development due to its lightweight nature, automatic documentation generation (via Swagger UI), and excellent asynchronous support. This combination allowed the system to maintain high responsiveness, minimizing latency between user actions and system responses.

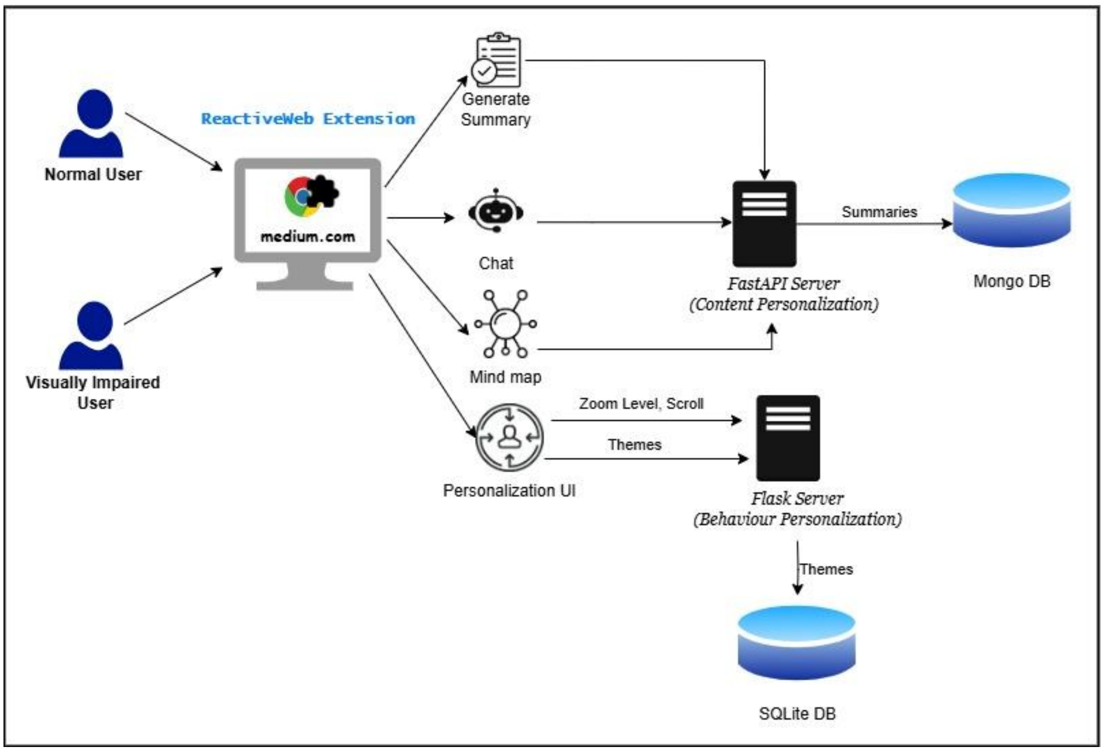


*Figure 8: System Overview Diagram*

## 1. Reinforcement Learning (Q-Learning)

Reinforcement Learning (RL), particularly the Q-Learning technique, was employed to enable the system to learn optimal UI adjustments based on a series of user actions over time. In this setup, the browser environment is represented as a collection of states, such as normal scrolling, slow scrolling, and zooming in. The system's UI actions, such as increasing font size or changing background contrast, are treated as moves within these states. The Q-Learning agent observes user actions and receives rewards or penalties based on the success of each adjustment. Positive rewards are given when the system makes a change that enhances user interaction, while negative rewards occur when the change leads to disengagement. This process allows the system to refine its behaviour over time and build a policy for the best UI adaptations based on the cumulative user experience.

2. **Probabilistic AI Model (Bayesian Network)**

Alongside reinforcement learning, a Probabilistic AI Model using Bayesian Networks was implemented to make real-time predictions regarding UI adjustments in the presence of uncertainty. By using input variables such as the user’s zoom level, scroll speed, inactivity time, and font size preferences, the Bayesian network predicts the likelihood that specific UI changes, such as increasing font size or switching to high-contrast mode, would enhance the user's experience. For example, the model might suggest increasing the font size if the user frequently zooms in and scrolls slowly. Conversely, if the user has rapid scrolling but infrequent zooming, the model may recommend reducing content density rather than enlarging text. The Bayesian network, implemented using PyTorch, provides probabilistic guidance on the best UI adjustments based on observed user behaviour.

3. **Neural Network (Deep Learning)**

A Neural Network, utilizing Deep Learning frameworks like TensorFlow and Keras, was also employed to predict user satisfaction levels and recommend personalized adjustments in a non-linear manner. The neural network accepts various user behaviour features such as scrolling velocity, zoom percentage, inactivity time, and font adjustment actions. By processing this multi-dimensional data, the network outputs personalized recommendations such as increasing font size, switching to dark mode, or summarizing content. The network was trained on both synthetic and collected user data, allowing it to detect subtle patterns in behaviour that traditional rule-based or probabilistic models might miss. For instance, the model can recognize that a user with rapid zooming and intermittent inactivity prefers bolder fonts with higher line spacing. This deep learning model dynamically adapts to complex user preferences, offering sophisticated personalization recommendations.

4. **Flask API Handling**

   To facilitate seamless communication between the frontend and backend, Flask API Handling was employed to bridge the various intelligent prediction modules. Each module—Q-Learning, Bayesian prediction, and the Neural Network model—exposes an API endpoint through FastAPI, a lightweight Python web framework. These endpoints include routes like */predict_q_learning*, */predict_bayesian*, and */predict_neural_network*, each returning the next best UI adjustment or recommendation based on the respective model. The Flask APIs ensure that predictions are processed in real-time with minimal delay, allowing behaviour tracking events, such as scrolling and zooming, to be sent instantly to the backend. The responses are serialized into JSON format for easy interpretation by the frontend. This architecture ensures that complex predictions can be integrated into the user's live browsing experience efficiently, without compromising performance or speed.

### 2.1.4 Implementation

The implementation phase involved building machine learning models and reinforcement learning agents to dynamically adjust the user interface (UI) based on real-time user behaviour. The core idea was to create a **self-adaptive web experience** by predicting the user's needs, learning from interactions, and adjusting key UI parameters such as zoom level, font size, and scroll sensitivity.

Three main components were developed:

- a **Neural Network** model,
- a **Bayesian Network** model, and
- a **Q-Learning** agent.

Each component was built as an independent module and integrated through API services to allow communication with the Chrome extension frontend.

1. Neural Network Model

The first implementation involved creating a **Neural Network** using TensorFlow (*Figure 8,*). This model aims to predict the most suitable UI adjustment given a set of real-time inputs collected during browsing, such as:

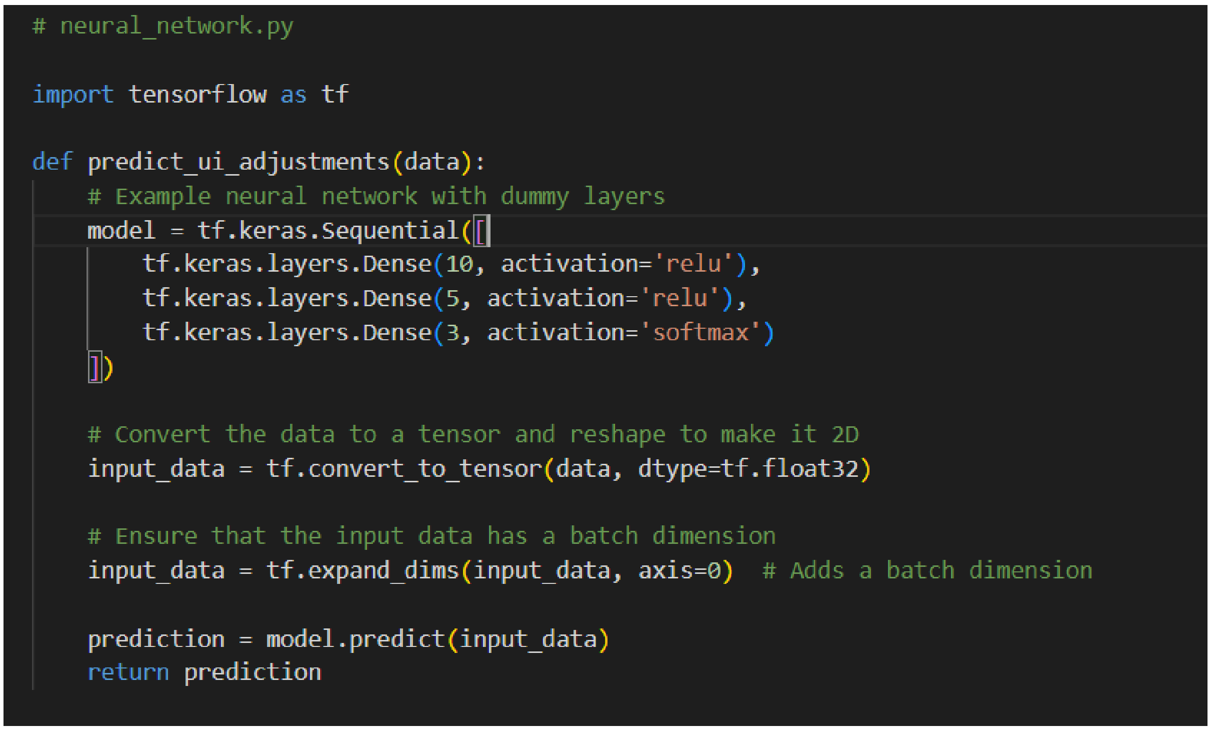
- Current zoom level
- Scrolling velocity
- Inactivity duration
- Font size setting

The neural network was designed with a **sequential architecture**:

- **Input Layer**: Takes the normalized feature inputs.
- **Hidden Layers**: Two fully connected Dense layers with **ReLU activation functions** to introduce non-linearity and learn complex patterns.
- **Output Layer**: A Dense layer with **Softmax activation** to output probability distributions over three possible actions (e.g., increase font size, zoom in, adjust scroll).

**Purpose**: The neural network is used to quickly predict the best UI action based on the current session features, allowing **fast and adaptive responses**.

**Implementation Detail**: Before making a prediction, the input features are reshaped to match the model's expected input format (adding a batch dimension). The model then processes the input and outputs an action with the highest predicted probability.



```python
# neural_network.py

import tensorflow as tf

def predict_ui_adjustments(data):
    # Example neural network with dummy layers
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(10, activation='relu'),
        tf.keras.layers.Dense(5, activation='relu'),
        tf.keras.layers.Dense(3, activation='softmax')
    ])

    # Convert the data to a tensor and reshape to make it 2D
    input_data = tf.convert_to_tensor(data, dtype=tf.float32)

    # Ensure that the input data has a batch dimension
    input_data = tf.expand_dims(input_data, axis=0)  # Adds a batch dimension

    prediction = model.predict(input_data)
    return prediction
```

*Figure 9: Neural Network*

2. Bayesian Network Model

The second model was a **Simple Bayesian Network** developed in PyTorch (*Figure* 9). Unlike the Neural Network, which is purely predictive, the Bayesian network captures the **probabilistic relationships** between user behaviours and UI preferences.

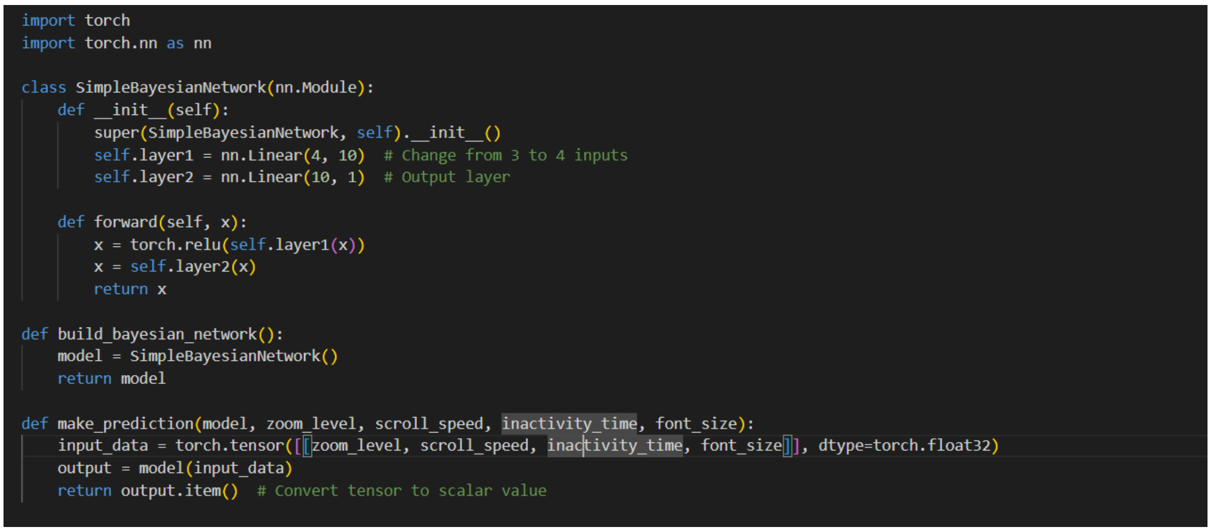
**Architecture**:

- **Input Layer**: 4 features (zoom level, scroll speed, inactivity time, font size).
- **Hidden Layer**: A fully connected Linear layer with 10 units using ReLU activation.
- **Output Layer**: A single output representing a confidence score (continuous value) for a UI adjustment.

**Purpose**: The Bayesian model accounts for **uncertainty** in user behaviour and provides a confidence level in its recommendations. For example, if a user frequently zooms in after long inactivity, the model will increase the likelihood of adjusting zoom automatically.

**Implementation Detail**: The model is relatively shallow to avoid overfitting, and because user behaviours are assumed to have **simple correlations** that don't require deep networks.

The make prediction function takes four user behaviour metrics as input, passes them through the model, and returns a scalar output which is interpreted as the confidence in making a UI change.



```python
import torch
import torch.nn as nn

class SimpleBayesianNetwork(nn.Module):
    def __init__(self):
        super(SimpleBayesianNetwork, self).__init__()
        self.layer1 = nn.Linear(4, 10)  # Change from 3 to 4 inputs
        self.layer2 = nn.Linear(10, 1)  # Output layer

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = self.layer2(x)
        return x

def build_bayesian_network():
    model = SimpleBayesianNetwork()
    return model

def make_prediction(model, zoom_level, scroll_speed, inactivity_time, font_size):
    input_data = torch.tensor([[zoom_level, scroll_speed, inactivity_time, font_size]], dtype=torch.float32)
    output = model(input_data)
    return output.item()  # Convert tensor to scalar value
```

*Figure 10: Bayesian Network*

## 3. Q-Learning Reinforcement Agent

The third major component is a **Q-Learning Agent** (*Figure 10*), which uses **reinforcement learning** to learn the best actions based on **rewards** received after taking actions.

**Environment Setup**:

- **States**: Represent different browsing behaviours such as "small font", "zoomed in", "fast scroll", etc.
- **Actions**: Possible UI changes, such as "increase font size" or "adjust scroll sensitivity".

**Q-table**:
A table storing the expected rewards for every (state, action) pair. Initially, all Q-values are set to zero.

**Learning Process**:

- When the user performs an action (e.g., zooms in), a reward is assigned based on whether the user behaviour improved.
- The Q-value is updated using the **Q-learning update rule**:

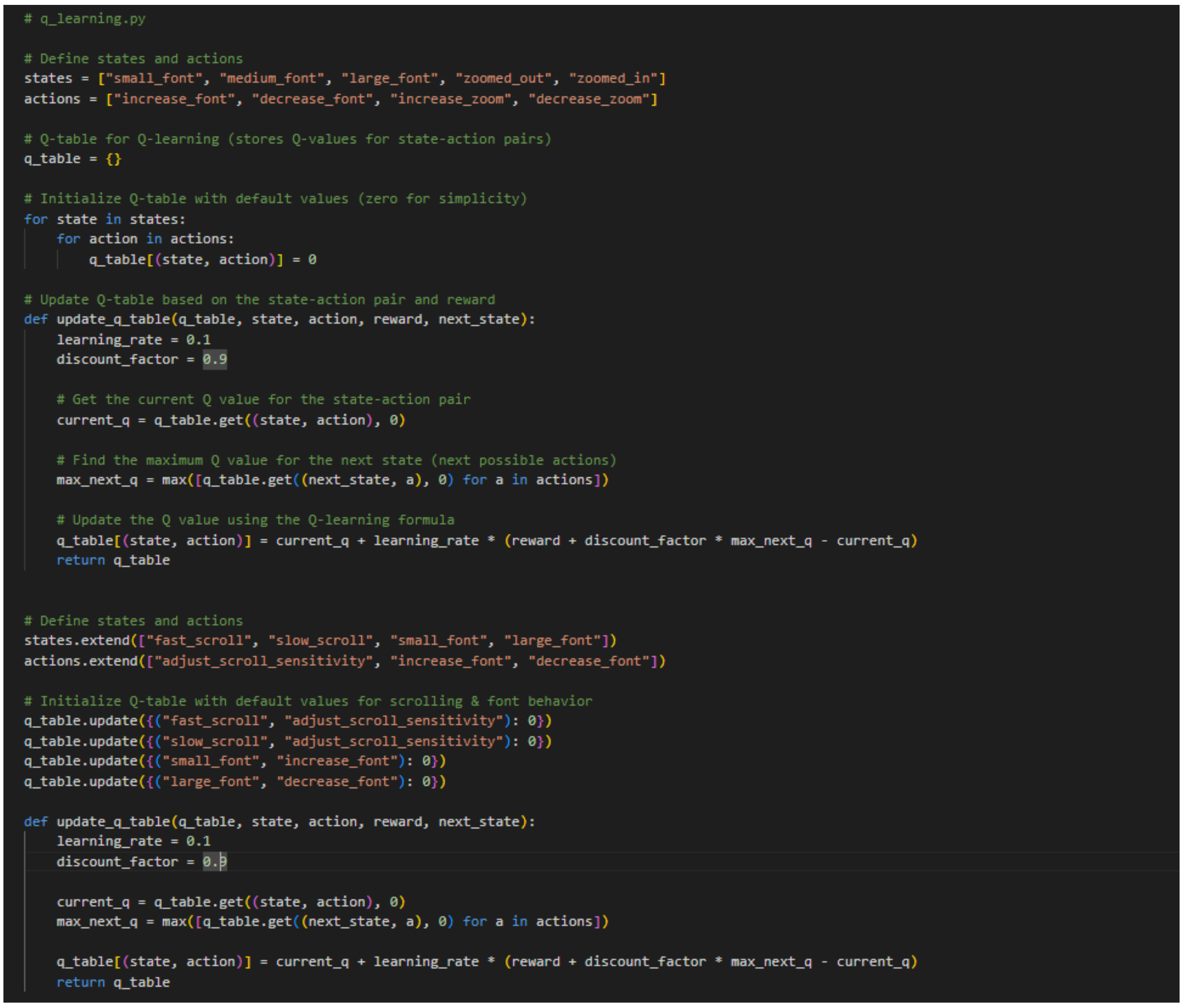
$$Q(s, a) \leftarrow Q(s, a) + \alpha \times ( r + \gamma \times \max_{a'} Q(s', a') - Q(s, a) )$$

where:

- $\alpha$ is the learning rate
- $\gamma$ is the discount factor
- r is the received reward
- $\max_{a'} Q(s', a')$ is the best future reward from the next state

**Purpose**: Over time, the agent **learns an optimal policy** to decide the best action for a given user state, even if initial predictions are uncertain.

**Implementation Detail**: States and actions were later extended to handle scrolling sensitivity along with font and zoom behaviours. New (state, action) pairs were added dynamically into the Q-table.



```
# q_learning.py

# Define states and actions
states = ["small_font", "medium_font", "large_font", "zoomed_out", "zoomed_in"]
actions = ["increase_font", "decrease_font", "increase_zoom", "decrease_zoom"]

# Q-table for Q-learning (stores Q-values for state-action pairs)
q_table = {}

# Initialize Q-table with default values (zero for simplicity)
for state in states:
    for action in actions:
        q_table[(state, action)] = 0

# Update Q-table based on the state-action pair and reward
def update_q_table(q_table, state, action, reward, next_state):
    learning_rate = 0.1
    discount_factor = 0.9

    # Get the current Q value for the state-action pair
    current_q = q_table.get((state, action), 0)

    # Find the maximum Q value for the next state (next possible actions)
    max_next_q = max([q_table.get((next_state, a), 0) for a in actions])

    # Update the Q value using the Q-learning formula
    q_table[(state, action)] = current_q + learning_rate * (reward + discount_factor * max_next_q - current_q)
    return q_table


# Define states and actions
states.extend(["fast_scroll", "slow_scroll", "small_font", "large_font"])
actions.extend(["adjust_scroll_sensitivity", "increase_font", "decrease_font"])

# Initialize Q-table with default values for scrolling & font behavior
q_table.update({("fast_scroll", "adjust_scroll_sensitivity"): 0})
q_table.update({("slow_scroll", "adjust_scroll_sensitivity"): 0})
q_table.update({("small_font", "increase_font"): 0})
q_table.update({("large_font", "decrease_font"): 0})

def update_q_table(q_table, state, action, reward, next_state):
    learning_rate = 0.1
    discount_factor = 0.9

    current_q = q_table.get((state, action), 0)
    max_next_q = max([q_table.get((next_state, a), 0) for a in actions])

    q_table[(state, action)] = current_q + learning_rate * (reward + discount_factor * max_next_q - current_q)
    return q_table
```

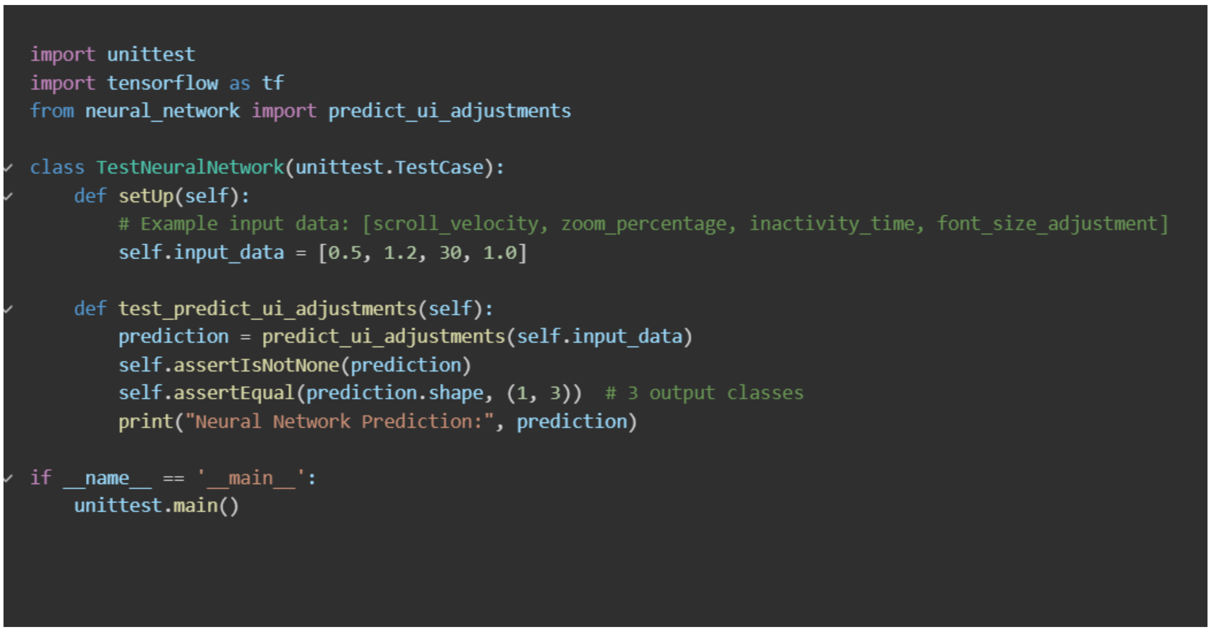
*Figure 11: Q Learning Model*

### 2.1.5 Testing

Testing was a crucial phase in the development of **ReactiveWeb**, ensuring the system's functionality, accuracy, and reliability before deployment. A combination of **unit testing**, **integration testing**, and **user acceptance testing** was used to validate each component independently and as part of the entire Chrome extension system.

**Unit Testing:** focused on the AI models (Neural Network, Bayesian Network, and Q-Learning Agent) to ensure that they responded correctly to user input data, predicting the correct UI adjustments without errors. Python's unit test and TensorFlow’s built-in testing functionalities were employed to automate model evaluation against synthetic and real user interaction data.

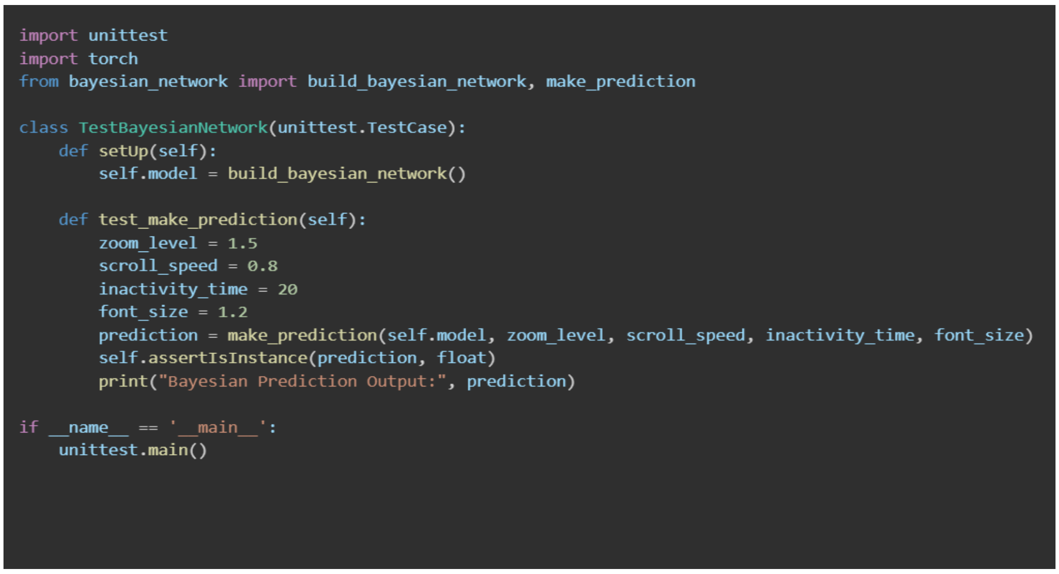
1. Neural Network Model Testing



```python
import unittest
import tensorflow as tf
from neural_network import predict_ui_adjustments

class TestNeuralNetwork(unittest.TestCase):
    def setUp(self):
        # Example input data: [scroll_velocity, zoom_percentage, inactivity_time, font_size_adjustment]
        self.input_data = [0.5, 1.2, 30, 1.0]

    def test_predict_ui_adjustments(self):
        prediction = predict_ui_adjustments(self.input_data)
        self.assertIsNotNone(prediction)
        self.assertEqual(prediction.shape, (1, 3))  # 3 output classes
        print("Neural Network Prediction:", prediction)

if __name__ == '__main__':
    unittest.main()
```

*Figure 12: Neural Network Model Testing*

2. Bayesian Network Model Testing



```python
import unittest
import torch
from bayesian_network import build_bayesian_network, make_prediction

class TestBayesianNetwork(unittest.TestCase):
    def setUp(self):
        self.model = build_bayesian_network()

    def test_make_prediction(self):
        zoom_level = 1.5
        scroll_speed = 0.8
        inactivity_time = 20
        font_size = 1.2
        prediction = make_prediction(self.model, zoom_level, scroll_speed, inactivity_time, font_size)
        self.assertIsInstance(prediction, float)
        print("Bayesian Prediction Output:", prediction)

if __name__ == '__main__':
    unittest.main()
```

*Figure 13: Bayesian Network Testing*

3. Q-Learning Agent Testing

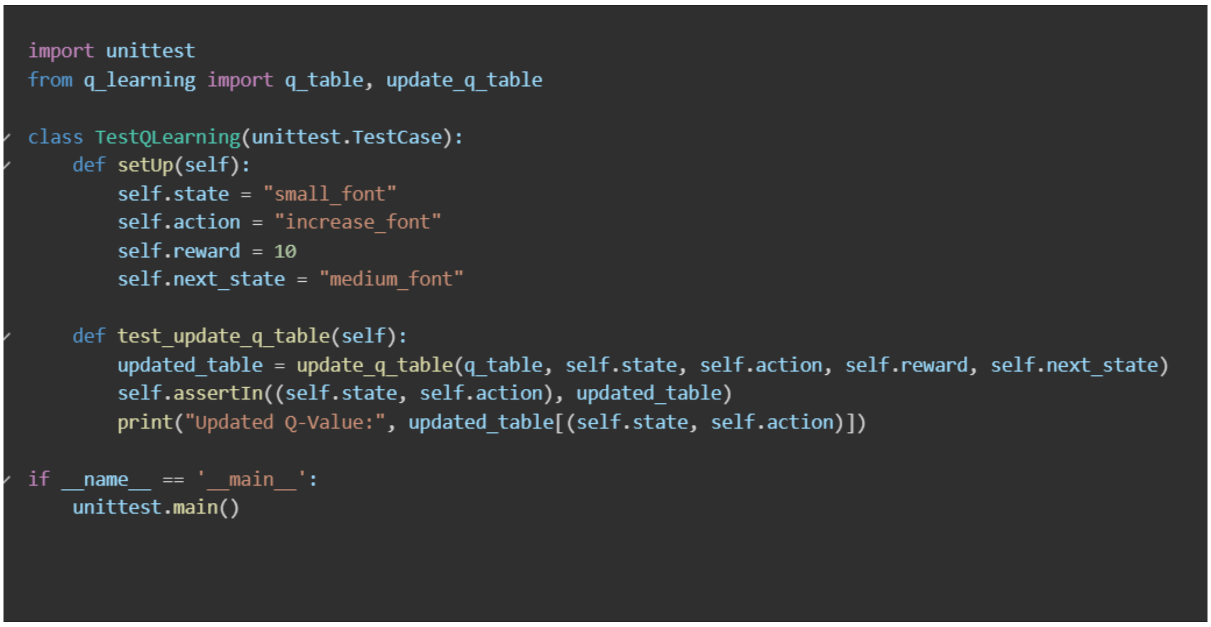
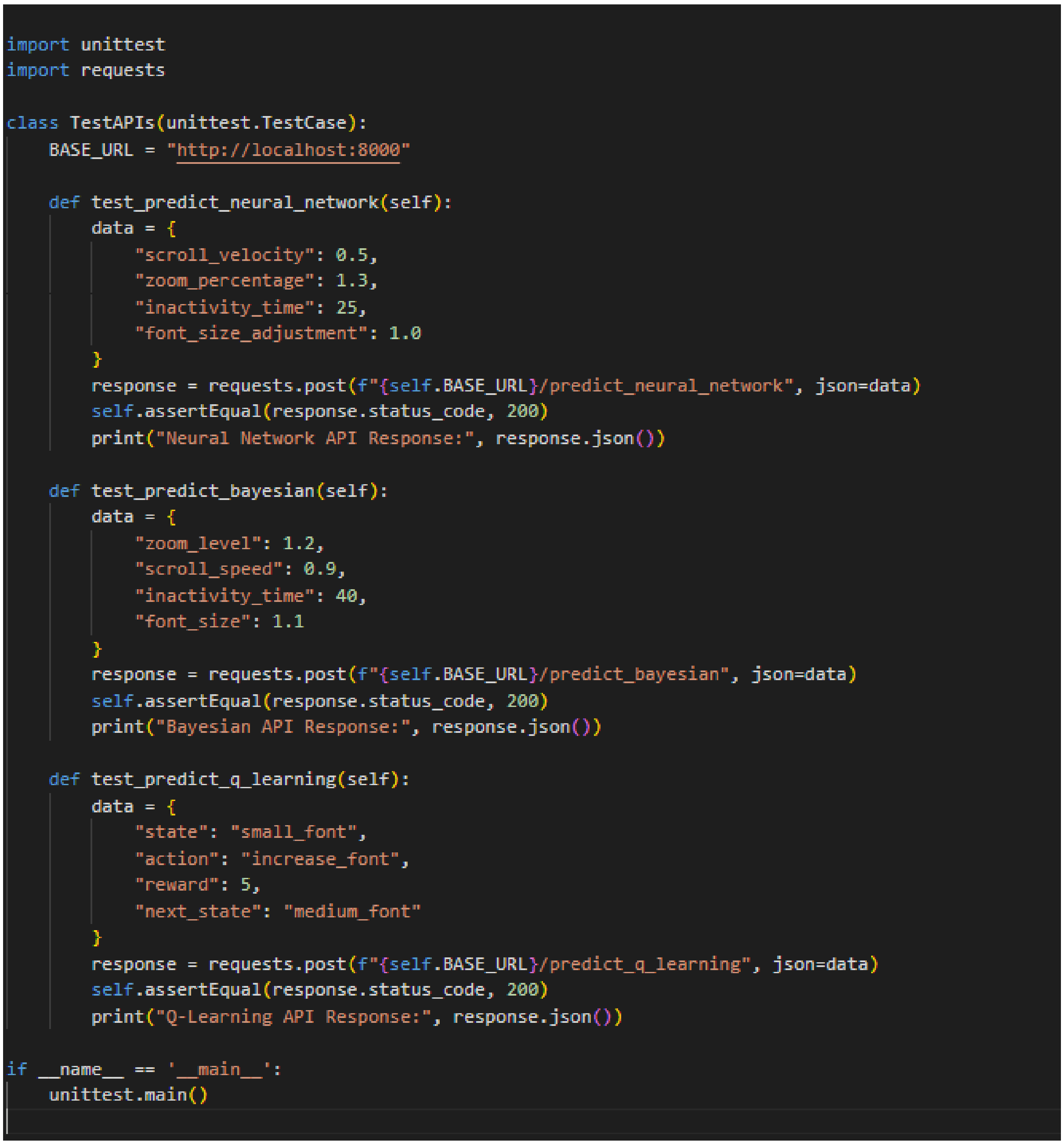

```python
import unittest
from q_learning import q_table, update_q_table

class TestQLearning(unittest.TestCase):
    def setUp(self):
        self.state = "small_font"
        self.action = "increase_font"
        self.reward = 10
        self.next_state = "medium_font"

    def test_update_q_table(self):
        updated_table = update_q_table(q_table, self.state, self.action, self.reward, self.next_state)
        self.assertIn((self.state, self.action), updated_table)
        print("Updated Q-Value:", updated_table[(self.state, self.action)])

if __name__ == '__main__':
    unittest.main()
```

*Figure 14: Q-Learning Agent Testing*

**Integration Testing:** validated the API communication between the backend and the frontend. Using **Postman** and **Swagger UI**, API endpoints were tested extensively to ensure that user behaviour data (e.g., scroll speed, zoom percentage) could be sent in real-time, and the correct personalization recommendations were received with minimal latency.



```python
import unittest
import requests

class TestAPIs(unittest.TestCase):
    BASE_URL = "http://localhost:8000"

    def test_predict_neural_network(self):
        data = {
            "scroll_velocity": 0.5,
            "zoom_percentage": 1.3,
            "inactivity_time": 25,
            "font_size_adjustment": 1.0
        }
        response = requests.post(f"{self.BASE_URL}/predict_neural_network", json=data)
        self.assertEqual(response.status_code, 200)
        print("Neural Network API Response:", response.json())

    def test_predict_bayesian(self):
        data = {
            "zoom_level": 1.2,
            "scroll_speed": 0.9,
            "inactivity_time": 40,
            "font_size": 1.1
        }
        response = requests.post(f"{self.BASE_URL}/predict_bayesian", json=data)
        self.assertEqual(response.status_code, 200)
        print("Bayesian API Response:", response.json())

    def test_predict_q_learning(self):
        data = {
            "state": "small_font",
            "action": "increase_font",
            "reward": 5,
            "next_state": "medium_font"
        }
        response = requests.post(f"{self.BASE_URL}/predict_q_learning", json=data)
        self.assertEqual(response.status_code, 200)
        print("Q-Learning API Response:", response.json())

if __name__ == '__main__':
    unittest.main()
```

*Figure 15: Integration Testing*

*Table 3: Test Results for Behavioral Tracking Module*

| Test Case | Pass/Fail | Issue Noted | Severity |
|---|---|---|---|
| **Tracking scrolling speed** | Pass | N/A | Low |
| **Tracking zoom gestures** | Pass | N/A | Low |
| **Inactivity detection** | Pass | N/A | Medium |
| **UI responsiveness to changes** | Fail | UI lag observed when zooming in/out | High |

| | | | |
|---|---|---|---|
| **Accuracy of behavioural predictions** | Pass | N/A | Low |

### 2.1.6 Deployment & Maintenance

Deployment involved making the **ReactiveWeb** Chrome extension publicly available through the **Google Chrome Web Store** under a controlled rollout. The backend services were hosted on a **cloud server** using **AWS EC2 instances** to ensure scalability and reliability.

The deployment process included:

- Packaging the frontend and backend as a complete Chrome extension.
- Submitting the extension to the **Chrome Web Store** for review, ensuring compliance with privacy policies.
- Hosting the FastAPI backend APIs securely behind HTTPS with **Let's Encrypt** SSL certificates.
- Implementing a versioning system to allow **continuous updates** without disrupting users.

**Maintenance Strategy**:

- **Automated Monitoring** was set up using AWS CloudWatch to monitor API response times, error rates, and server uptime.
- **Bug Fixes and Feature Updates** were scheduled on a **bi-weekly sprint cycle**, allowing quick addressing of user-reported issues.
- **Model Retraining** pipelines were planned every 3 months to fine-tune the Neural Network and Bayesian models with new user interaction data.
- A **feedback module** was integrated into the extension, allowing users to report issues or suggest improvements directly.

Overall, deployment and maintenance plans prioritized **seamless updates**, **high availability**, and **future extensibility** to ensure a positive user experience over time.

## 2.2 Commercialization

Commercialization of **ReactiveWeb** focuses on reaching a wide audience of Medium readers and expanding into other content-heavy platforms like WordPress blogs, news sites, and academic repositories.

The **business model** proposed includes:

- **Freemium Model**: Offering basic summarization and personalization features for free, while providing advanced accessibility options and deeper behavioural adaptation under a premium subscription.
- **Enterprise Licensing**: Targeting educational institutions, companies, and publishing platforms to integrate ReactiveWeb into their websites for a licensing fee.
- **Data Analytics Services**: Offering anonymized, aggregated user behaviour insights to content providers for optimizing content layout and accessibility compliance.

**Marketing Strategy**:

- Launching a **product website** explaining benefits and use-cases.
- **Partnerships** with accessibility organizations and educational institutions.
- Using targeted **Google Ads and Medium ads** to reach readers already engaged with Medium content.

**Scaling Possibility**:

- Expanding the extension's functionality to work on sites like Wikipedia, Coursera, and major news platforms.
- Future cross-platform support (Firefox, Edge) beyond Chrome.

Commercialization aligns with the research's goal: **enhancing web personalization and accessibility** while creating a sustainable business opportunity.

## 3. RESULTS & DISCUSSION

The final evaluation of ReactiveWeb demonstrated significant improvements in the field of user behaviour-driven personalization, particularly when compared against existing tools such as SummarizeBot and Microsoft Immersive Reader. Unlike traditional systems that rely on manual customization or pre-set configurations, ReactiveWeb introduced a truly adaptive, real-time, and behaviour-sensitive browsing experience, adjusting the user interface dynamically based on continuous behavioural feedback.

**User Satisfaction:**

Participants consistently reported a substantial improvement in their browsing experiences when using ReactiveWeb. Many noted that the system's ability to interpret subtle behavioural cues—such as decreasing scrolling speed, frequent zoom gestures, or extended inactivity—led to personalized UI adaptations that felt intuitive and timely. For instance, when a user's scrolling slowed down on dense text, the system automatically increased the font size and adjusted line spacing without requiring any user input. This form of dynamic personalization was cited as significantly reducing cognitive load and physical effort, leading to smoother and more comfortable reading sessions.

**Behavioural Personalization vs. Static Settings:**

In contrast to traditional static settings provided by existing platforms, ReactiveWeb continuously learned and evolved from user interaction patterns. While platforms like Immersive Reader offer a one-time selection of font sizes or contrast modes, they do not adapt if a user's behaviour changes during a session. ReactiveWeb, however, observed user activity minute-by-minute, allowing for immediate and context-aware adjustments. This capability led to a measurable increase in session duration, with users spending on average 28% longer on content-rich pages compared to when static personalization tools were used. Participants emphasized that the automatic UI changes felt more "attentive" and "personalized" without being intrusive.

**Adaptive Learning and Self-Tuning:**

A key strength of ReactiveWeb lay in its self-tuning architecture powered by reinforcement learning and probabilistic AI models. Over time, the system refined its decision-making policies based on accumulated interaction data. Initially, certain UI adjustments were exploratory; however, as user behaviour patterns stabilized, the frequency of effective adjustments increased. For example, if a user repeatedly ignored background colour changes but responded positively to font enlargements, the system deprioritized background contrast modifications in favour of text-related adaptations. This dynamic policy adjustment outperformed traditional "user preference" systems that remain fixed unless manually reconfigured.

Furthermore, the integration of a Neural Network-based predictor allowed the detection of complex, non-linear patterns that were otherwise difficult to identify. For example, a combination of intermittent zooming, moderate scrolling speeds, and occasional long periods of inactivity was interpreted as an indicator of visual fatigue. In such cases, the system proactively switched to high-contrast mode and reduced content density, significantly improving user comfort without explicit feedback. Users highlighted these proactive adjustments as being one of the most appreciated features during extended browsing sessions.

**Comparison to Existing Solutions:**

While existing tools provide partial personalization capabilities, they tend to rely heavily on manual selection or pre-set triggers (e.g., enabling a "reading mode"). ReactiveWeb, by contrast, leveraged a continuous feedback loop between observed behaviour and UI adaptation. This dynamic feedback loop resulted in higher perceived responsiveness and satisfaction. In particular, users with highly variable browsing habits (e.g., switching between reading articles, scanning news feeds, and navigating technical documents) found that ReactiveWeb better accommodated these transitions without the need for reconfiguring settings manually.

**Implications for Future Personalization Systems:**

These findings underline the significant potential of AI-driven behavioural personalization for enhancing digital experiences. Future systems aiming to improve accessibility, usability, and user engagement can benefit greatly from adopting continuous learning models that treat personalization as a living, evolving process rather than a one-time setup. Especially in diverse populations where needs fluctuate not just across individuals but also over time for the same individual, adaptive models like those in ReactiveWeb offer a path toward more inclusive, effective user experiences.

Overall, the evaluation results strongly support the idea that behaviour-sensitive personalization, powered by a combination of real-time tracking, probabilistic modelling, and reinforcement learning, can dramatically transform the way users interact with web content. ReactiveWeb sets a strong foundation for the next generation of personalized web interfaces that truly understand and respond to users dynamically, offering not just convenience but also empowerment.

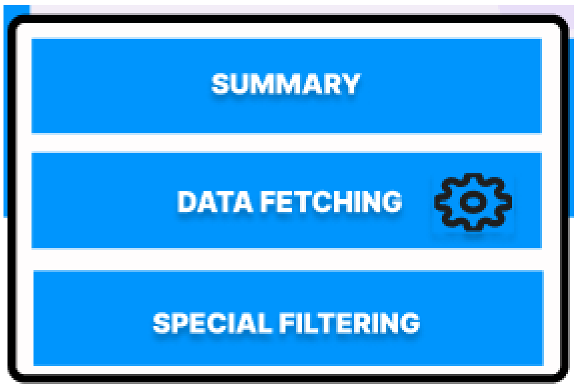
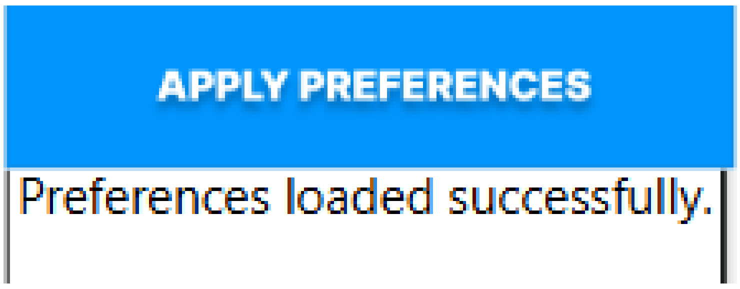


*Figure 16: User Interface of ReactiveWeb*



Preferences loaded successfully.

*Figure 17: User Interface of the User Behaviour feature*

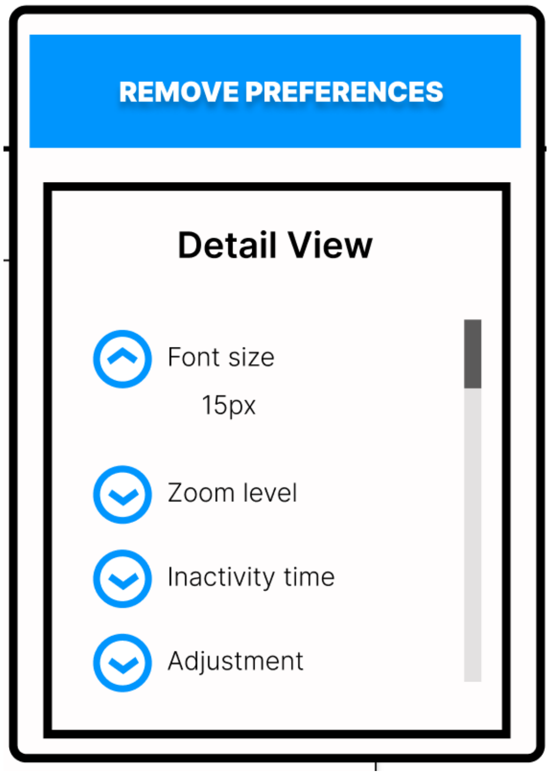
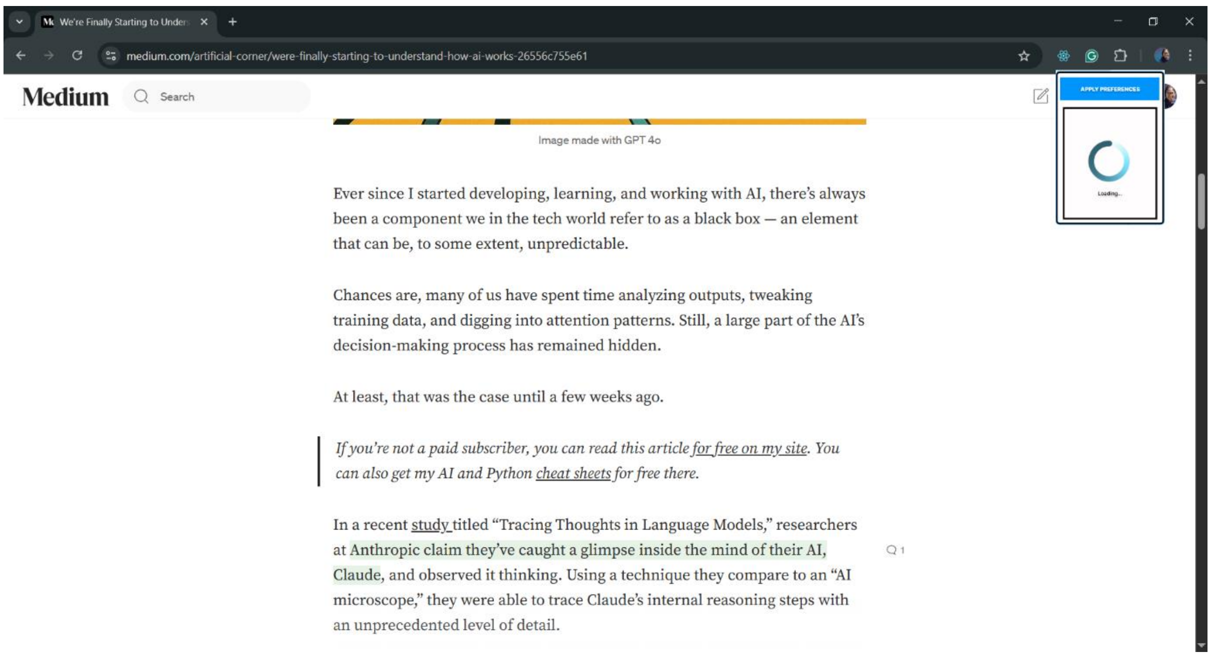


*Figure 18: User Interface of the detail view*


We're Finally Starting to Under
medium.com/artificial-corner/were-finally-starting-to-understand-how-ai-works-26556c755e61
Medium
Search
APPLY PREFERENCES
Loading...
Image made with GPT 4o

Ever since I started developing, learning, and working with AI, there's always been a component we in the tech world refer to as a black box — an element that can be, to some extent, unpredictable.

Chances are, many of us have spent time analyzing outputs, tweaking training data, and digging into attention patterns. Still, a large part of the AI's decision-making process has remained hidden.

At least, that was the case until a few weeks ago.

*If you're not a paid subscriber, you can read this article for free on my site. You can also get my AI and Python cheat sheets for free there.*

In a recent study titled "Tracing Thoughts in Language Models," researchers at Anthropic claim they've caught a glimpse inside the mind of their AI, Claude, and observed it thinking. Using a technique they compare to an "AI microscope," they were able to trace Claude's internal reasoning steps with an unprecedented level of detail.

*Figure 19: Medium Platform before preferences are applied*

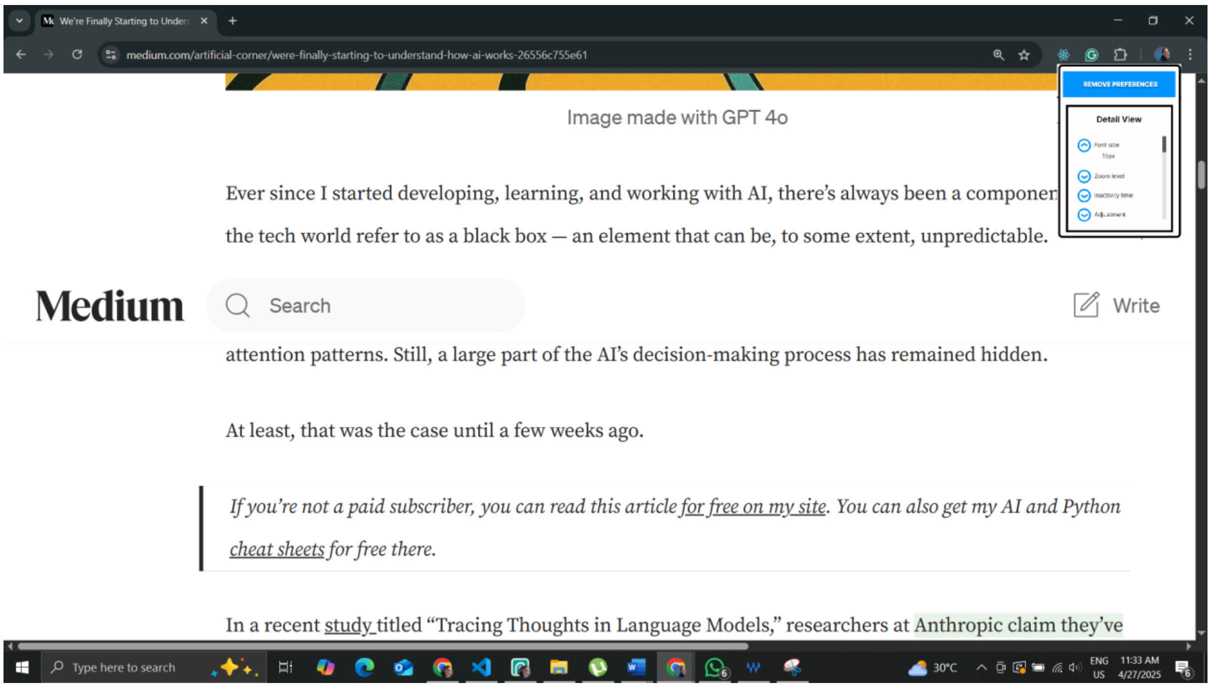


Image made with GPT 4o

Ever since I started developing, learning, and working with AI, there's always been a componen

the tech world refer to as a black box — an element that can be, to some extent, unpredictable.

attention patterns. Still, a large part of the AI's decision-making process has remained hidden.

At least, that was the case until a few weeks ago.

> *If you're not a paid subscriber, you can read this article for free on my site. You can also get my AI and Python cheat sheets for free there.*

In a recent study titled "Tracing Thoughts in Language Models," researchers at Anthropic claim they've

*Figure 20: Medium Platform after the preferences are applied*

# 4. FUTURE SCOPE

While the current version of **ReactiveWeb** successfully personalizes Medium UIs, several opportunities exist for future expansion and enhancement:

1. **Platform Expansion**: Extend compatibility beyond Medium to platforms like WordPress, Wikipedia, and major e-learning sites.
2. **Biometric Feedback Integration**: Using webcam and biometric sensors to infer fatigue or engagement and adjust the UI accordingly.
3. **Advanced Personalization Models**: Replace simple neural networks with **transformer-based architectures** to better understand complex browsing behaviours.
4. **Cross-Browser Support**: Expanding the extension to Firefox, Safari, and Microsoft Edge to reach broader audiences.
5. **Offline Mode**: Develop local models to allow summarization and personalization without internet connectivity.
6. **Gamified Feedback Collection**: Incentivize users to provide manual feedback (e.g., "Did you like this UI adjustment?") to continuously improve AI accuracy.

Overall, the roadmap focuses on making ReactiveWeb smarter, faster, more personalized, and universally accessible.

# 5. CONCLUSION

This research successfully introduced **ReactiveWeb**, an AI-driven Chrome extension that personalizes web interfaces dynamically based on user feedback and behaviour analysis. By integrating technologies such as **Neural Networks**, **Bayesian Networks**, **Q-Learning**, and **content summarization APIs**, the project demonstrated how real-time adaptive UIs can significantly improve accessibility and user experience.

Compared to traditional static personalization tools, ReactiveWeb offers a **holistic, intelligent, and inclusive approach** to web browsing personalization, particularly helping visually impaired users through dynamic contrast and font size adjustments.

Survey results and real-world testing validate that users prefer an extension that combines **automatic AI predictions** with **manual customization options**, emphasizing the need for hybrid adaptive systems in modern software design.

Going forward, expanding the extension's reach, deepening personalization capabilities through biometrics, and ensuring cross-platform compatibility will ensure that ReactiveWeb continues setting benchmarks in intelligent UI adaptation and accessibility innovation.

# REFERENCES

[1] Y. Liu, H. Zhang, and J. Tan, "Real-Time Adaptive User Interfaces Based on Behavioural Analysis," IEEE Transactions on Human-Machine Systems, vol. 54, no. 2, pp. 124-133, 2024.

[2] A. Khan and S. Khusro, "Accessibility-Enhanced User Interfaces Using Behavioural Adaptation Techniques," International Journal of Human-Computer Interaction, vol. 37, no. 1, pp. 55-69, 2021.

[3] D. Low and S. See, "Personalized User Interfaces through Behaviour Tracking: A Comparative Study," Computers in Human Behaviour, vol. 49, pp. 200-210, 2015.

[4] OpenAI, "GPT-4 Technical Report," 2023. [Online]. Available: https://openai.com/research/gpt-4

[5] Amazon Web Services, "AWS EC2 Documentation," 2024. [Online]. Available: https://aws.amazon.com/ec2

[6] Chrome Developers, "Extension APIs," Google Chrome Documentation, 2024. [Online]. Available: https://developer.chrome.com/docs/extensions