Design and Evaluation of an Accessible IDE Plugin for Supporting Neurodivergent Software Developers

Clo Elis

220402088

May 2026

Computer Science

Supervisor: Dr. Vlad Gonzalez

Abstract

Modern Integrated Development Environments (IDEs) prioritise feature density, often at the expense of cognitive accessibility. For neurodivergent software developers—particularly those with Attention-Deficit/Hyperactivity Disorder (ADHD), Autism Spectrum Disorder (ASD), or Dyslexia this complexity manifests as significant barriers driven by executive dysfunction, including cognitive overload and difficulty in task initiation. This research investigates the efficacy of an Artificial Intelligence (AI) integrated plugin designed to mitigate these challenges within the VS Code ecosystem.

The developed solution comprises three core modules: a toggle to close all unnecessary panels to filter interface-driven distractions, a task breakdown tool to leverage Large Language Models (LLMs) to decompose complex requirements into structured checklists, and a visual styling tool for high-contrast, low-strain environmental presets. The system's impact was evaluated through a comparative user study ($N = 5$) pitting a standard VS Code baseline against the enhanced environment. Evaluation metrics included the NASA Task Load Index (NASA-TLX) [4] to quantify shifts in mental workload and the System Usability Scale (SUS) [10] to assess overall usability and tool effectiveness.

Results indicate that by streamlining the development environment and providing automated task scaffolding, the plugin significantly reduces the executive tax imposed by standard tools. This allows neurodivergent developers to reallocate critical cognitive resources from interface navigation to core algorithmic problem-solving.

Declaration “I declare that this document represents my own work except where otherwise stated” - Clo Ellis

Acknowledgments I would like to express my sincere gratitude to my supervisor, Dr. Vlad Gonzalez, for his guidance and support throughout this project. I also want to thank all the participants who contributed to this study, without whom this work would not have been possible.

Contents

Abstract **I**

List of Abbreviations **V**

Glossary **VI**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Statement 1
1.3 Understanding Neurodiversity in Software Development 1
1.4 Aims and Objectives 2
1.4.1 Project Aim 2
1.4.2 Research Objectives 2
1.5 Dissertation Structure 3

2 Literature Review **5**
2.1 Cognitive Load Theory (CLT) 5
2.2 Applying CLT to Neurodiversity 6
2.2.1 Integration with Rosenshin‘s Principles of Instruction 6
2.2.2 The Cognitive Theory of Multimedia Learning (CTML) 6
2.3 Neurodiversity in Software Engineering 7
2.3.1 The Cognitive Load Connection 7
2.3.2 Executive Functioning and Task Switching 7
2.3.3 Specific Challenges by Condition 8
2.3.4 Beyond Medical 8
2.3.5 The “Double-Edged Sword” of Hyperfocus 9
2.4 Existing Tools 9
2.5 Pomodoro Timer 10
2.5.1 Managing Executive Dysfunction and Task Paralysis 10
2.5.2 Combatting Time Blindness and Hyperfocus Burnout 10
2.5.3 Cognitive Load and Adaptive Implementation 10
2.6 Colour filters 11
2.6.1 The Halation Risk and Soft High Contrast 11
2.7 AI as a Cognitive Tool: Task Decomposition and Scaffolding 12
2.8 NASA-TLX and SUS: Measuring Cognitive Load and Usability 12

3 Design and Implementation **13**
3.1 User Requirements 13
3.2 Module Design 14
3.3 Technology Stack 15
3.3.1 Minimalist Toggle 15
3.3.2 Task Architect - AI-Driven Scaffolding 16

3.3.3 Visual Styler . . . 16
3.4 System Architecture . . . 17
3.5 UI Simplification and Minimalist Toggle . . . 19
3.6 AI Integration (FR2 and FR3) . . . 20
3.6.1 Task Architect: Overcoming Task Paralysis . . . 20
3.6.2 Human-in-the-Loop (HITL) Implementation . . . 22
3.7 Dyslexia Settings: Test spacing (FR5) . . . 23
3.7.1 Initial Plan: One Option . . . 24
3.7.2 Final UI Integration . . . 26
3.8 Pomodoro Timer (FR4) . . . 29
3.9 Initial User Testing and Feedback . . . 29
3.10 Final UI . . . 30
3.11 Evaluation and Methodology . . . 33
3.11.1 Experimental Design . . . 33
3.11.2 Quantitative and Qualitative Frameworks . . . 33
3.11.3 Data Collection and Analysis . . . 34

4 Results and Discussion **35**
4.1 NASA-TLX Subjective Workload . . . 35
4.2 Workload Comparison and Interpretation . . . 35
4.3 Efficiency and Time-on-Task Analysis . . . 37
4.4 SUS Metrics . . . 38
4.4.1 Item 2 & 6: Complexity and Consistency . . . 38
4.4.2 Item 4 & 10: Independence and Learning Curve . . . 38
4.4.3 Item 9: Confidence . . . 38
4.5 Pre-Study Interview Findings . . . 39
4.6 Real-Time Feedback and User Experience . . . 39
4.7 Number of Clicks and User Flow . . . 40
4.8 Latency Testing . . . 41
4.9 Discussion . . . 41
4.10 Limitations . . . 41

5 Conclusion **42**
5.1 Project Aim . . . 42
5.2 Conclusion of Objectives . . . 42
5.3 What has been learned . . . 42
5.4 Future Work . . . 43

References **43**

List of Abbreviations

ADHD Attention-Deficit/Hyperactivity Disorder. 1–3, 7–10, I, V

AI Artificial Intelligence. 1, 3, I, V

API Application Programming Interface. 3, 14, V

ASD Autism Spectrum Disorder. 1, 2, 7, I, V

CoT Chain of Thought. 3, 10, 12, 15, V

EL Extraneous Load. 5, 6, V

GL Germane Load. 5, 6, V

HITL Human-in-the-Loop. 11, V

IDE Integrated Development Environment. 1–3, 6, 7, 9, 11, 16, I, V

IL Intrinsic Load. 5, 6, V

LLM Large Language Model. 3, 10, I, V

NASA-TLX NASA Task Load Index. 4, I, V

SUS System Usability Scale. 4, I, V

UI User Interface. 3, V

WCAG Web Content Accessibility Guidelines. V

Glossary

Dyslexia A learning disability that affects reading and writing skills, often characterised by difficulties with accurate and/or fluent word recognition and by poor spelling abilities. 2, V

executive dysfunction A range of cognitive difficulties affecting the ability to plan, focus attention, and multi-task. 1–3, 7, 8, I, V

hyperfocus A state of intense focus and concentration, often seen in individuals with ADHD, where they become deeply engrossed in a task or activity, sometimes to the exclusion of everything else. 1, 3, V

neurodivergent A term used to describe people whose brains function, learn, and process information differently than what is considered typical. 1–3, I, V

task paralysis A condition where an individual becomes unable to initiate or complete a task, often due to cognitive overload or anxiety. 2, V

1 Introduction

1.1 Motivation

This project was initially motivated by my lived experience as a neurodivergent developer. In professional settings, ASD is frequently romanticised as a "superpower" due to the high capacity for deep logical analysis and rapid skill acquisition. However, this perceived strength came with a significant cognitive tax. Adapting to new environments, fluctuating workloads and maintaining a sustainable work-life balance is often extremely difficult as current tools like IDEs lack out of the box features to mitigate these executive functioning barriers. This research seeks to transition the IDE from a passive tool into an active cognitive partner through AI-driven support.

1.2 Problem Statement

While modern IDEs are feature-rich, their design reflects a neurotypical bias that assumes a high baseline for filtering sensory input and managing multi-step planning. For developers with ADHD, ASD, or Dyslexia, these tools often exacerbate Extraneous Cognitive Load, the mental effort spent on processing the interface rather than the problem logic [22].

Current IDEs prioritise feature density, often resulting in visual noise from nested menus and persistent notifications. Research indicates that developers with ADHD frequently experience task paralysis, difficulty initiating tasks during a project [3], while those with ASD may struggle with the double empathy problem problem, leading to them misinterpreting ambiguous or implied requirements from team members [13]. Existing AI tools focus almost exclusively on code generation rather than process support. Consequently, there is a critical need for an IDE intervention that acts as a digital ramp, filtering noise and providing the scaffolding necessary to bypass the executive freeze response.

1.3 Understanding Neurodiversity in Software Development

Before going into the specific technical barriers of modern IDEs, it is necessary to define neurodiversity and the cognitive frameworks that underpin this research. Originally coined by Judy Singer in the late 1990s, neurodiversity views neurological differences, such as ASD, ADHD, and Dyslexia, as natural variations in the human genome rather than purely medical deficits [20].

In the context of software engineering, these variations often manifest as unique cognitive processing styles. While many neurodivergent developers possess high aptitude for logic and pattern recognition, they frequently encounter barriers related to executive dysfunction. This is an umbrella term for difficulty in mental processes that enable us to plan, focus attention, remember instructions, and juggle multiple tasks successfully. For a developer, executive dysfunction is not a lack of technical ability, but rather a barrier in three key areas: Inhibitory Control (filtering IDE distractions), Working Memory (tracking complex syntax), and Cognitive Flexibility (switching between high-level logic and low-level code).

To provide a clearer lens for the objectives of this project, the below table outlines the specific

strengths and environmental challenges associated with the conditions most prevalent in the developer community [21] [9] [18].

Table 1: Neurodivergent Strengths and Environmental Challenges

Condition	Potential Strengths	Common Environmental Challenges
ASD	Deep focus, systematic thinking, high integrity, exceptional memory for details, strong pattern recognition.	Sensory overload in noisy or bright environments, misinterpretation of direct communication style, difficulty with unspoken social rules.
ADHD	Creativity, hyperfocus on passions, adaptability, high energy, ability to multitask or rapidly switch contexts.	Distractibility in chaotic settings, struggles with time management and organisation leading to time blindness and issues with task initiation, impulsivity in structured environments.
Dyslexia	Strong visual-spatial reasoning, excellent big-picture thinking, creative problem-solving, strong verbal skills.	Difficulties with text-heavy tasks, timed reading assignments, and conventional written communication.

1.4 Aims and Objectives

1.4.1 Project Aim

The primary objective of this project is to design, implement, and evaluate an assistive VS Code plug-in that reduces extraneous cognitive load and mitigates the effects of executive dysfunction for neurodivergent software developers. By introducing a cognitive prosthetic layer, the project seeks to determine if targeted User Interface (UI) simplification and AI-driven task scaffolding can measurably improve the developer experience.

1.4.2 Research Objectives

Objective 1: Interface Optimisation (The Minimalist Toggle)

Target: Develop a single "Minimalist Toggle" feature that simultaneously hides non-essential UI elements, including the Activity Bar, Status Bar, Mini-map, and Breadcrumbs.

Justification: This directly addresses the issue of visual overload by reducing the number of stimuli competing for the developer's attention, thus lowering the Extraneous Cognitive Load and creating a more focused coding environment [22], reducing visual noise that leads to distraction and sensory overwhelm for autistic and ADHD developers.

Objective 2: AI-Driven Task Decomposition (The Task Architect)

Target: To implement an LLM-integrated module that translates ambiguous programming goals into structured, editable checklists using a Chain of Thought (CoT) prompting strategy. This

should be implemented via the OpenAI Application Programming Interface (API) that breaks down high-level requirements [16].

Justification: This supports task initiation and reduces the energy required to overcome task paralysis, a primary barrier for developers with ADHD and ASD [6].

Objective 3: Simplification of Technical Feedback (The Code Explainer)

Target: To build a diagnostic interceptor that re-interprets complex compiler errors into plain-language, bulleted instructions [17]. Limit output to a maximum of three bullet points using direct instructions style.

Justification: This minimizes the cognitive demand of interpreting dense jargon, preventing rabbit holes and supporting the literal communication needs identified in the Double Empathy problem. By simplifying the feedback loop, developers can maintain focus on the core problem rather than getting lost in peripheral details. This is particularly beneficial for those with ASD, who may struggle with implied logic and prefer clear, direct instructions. Additionally, it can help mitigate the hyperfocus trap by providing concise guidance that keeps the developer on track without overwhelming them with information.

Objective 4: Visual Accessibility Customisation (The Visual Styler)

Target: To create at least two visual presets, such as "Soft High Contrast", that adhere to Web Content Accessibility Guidelines (WCAG) 2.1, set of international standards designed to make web content more accessible to people with disabilities it can also help with visual processing, eye strain and enhance readability [2].

Justification: Addresses the sensory sensitivities and colour preferences reported by neurodivergent developers in research surveys [17].

Objective 5: Empirical Evaluation of Cognitive Impact

Target: To conduct a comparative study using the NASA-TLX and the SUS to validate the plugin's efficacy against a standard IDE.

Justification: This provides a standardized scientific metric to confirm if the tool successfully reduces mental demand and frustration for the target demographic.

1.5 Dissertation Structure

This dissertation is organized into five sections, following a logical progression from theoretical grounding to empirical validation:

- **Section 2: Literature Review** I will evaluate the foundational psychological frameworks of Cognitive Load Theory (CLT) and Multimedia Learning (CTML). This provides a theoretical foundation for other research into the current state of neurodivergent software engineering as well as the shift toward a more social model of disability rather than a medical model.
- **Section 3: Design and Implementation** Details the user requirements and the three-layer system architecture. This chapter explains the technical execution of "Minimalist Toggle"

and “Task Architect” as well as the visual presets and styling tools. I will also discuss issues encountered during development and how they were resolved, such as the design of the UI and issues with the AI integration and prompt generation.

- **Section 4: Results and Discussion** Presents the quantitative and qualitative findings from the user study ($N = 5$). It analyzes the 82% reduction in perceived workload measured by the NASA-TLX and evaluates the tool‘s excellent usability rating via the System Usability Scale (SUS). This section also includes a critical discussion of the results, exploring the implications for future assistive technology in software development and acknowledging the limitations of the study, such as the small sample size and potential biases in participant selection.

- **Section 5: Conclusion** Summarises the core findings, acknowledges research limitations, and proposes specific avenues for future work with larger, more diverse participant pools.

2 Literature Review

2.1 Cognitive Load Theory (CLT)

Cognitive Load Theory, primarily developed by John Sweller, suggests that since the human brain has limited working memory capacity, instructional design can aid in avoiding overload. Sweller [22] proposes a three-part load framework that identifies three different types of cognitive load that impact a learner's ability to process information:

1. Intrinsic Load (IL): This is the inherent difficulty of the task itself. It is defined by the number of elements must be processed simultaneously to understand a concept. For example, learning basic vocabulary has low intrinsic load, while mastering complex syntax has high intrinsic load.
2. Extraneous Load (EL): This is seen as the unnecessary load that you would aim to minimise wherever possible. It is influenced by the way information is presented. Some examples that could exacerbate the load are poorly designed diagrams, distracting environments or redundant text. Poor design choices force the brain to waste resources on non-essential processing.
3. Germane Load (GL): This is the load I am aiming to maximise. It refers to the mental effort that can be directed toward the more key task, which in this scenario would be the code programming task.

The framework can be simplified down to the idea that the Total Cognitive Load (TCL) is the sum of Intrinsic Load, Extraneous Load, and Germane Load. The goal of instructional design is to manage these loads effectively to maximise learning outcomes.

$$TCL = IL + EL + GL$$

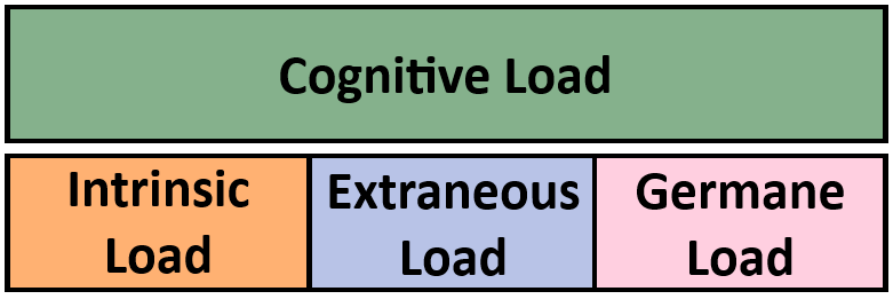

Figure 1: Sweller's Tripartite Model of Cognitive Load, illustrating the balance between Intrinsic, Extraneous, and Germane loads.

One of the main focuses of this project is that for neurodivergent developers, the IDE environment often imposes a disproportionate Extraneous Load. When EL is high, the "cognitive budget" is exhausted before the developer can engage with the core logic of the code, leading to task paralysis.

My plugin aims to “free up” this mental bandwidth by suppressing EL (via the Minimalist Toggle) and scaffolding IL (via the Task Architect), thereby maximizing the availability of cognitive resources for Germane processing. By reducing the noise and providing structured support, the plugin allows neurodivergent developers to focus their cognitive resources on the actual problem-solving aspect of programming, rather than being overwhelmed by the interface or the complexity of the task at hand.

2.2 Applying CLT to Neurodiversity

For neurodivergent developers, the Extraneous Load is not just due to bad design, but a barrier to entry. Features like the Activity Bar and Mini-map, while useful for some, can can be filter out by a neurotypical developer but act as “visual noise” for those with ADHD or ASD. To be able to code a high IL is required. However, if the “activation energy” to start is too high due to executive dysfunction, the developer never reaches the stage where they can handle that load. Having a single switch that closes all UI elements will directly targets the suppression of EL to “free up” mental bandwidth instantly without having to navigate menus. While the AI task decomposition tool acts as a “cognitive prosthetic” that scaffolds the IL of project planning, preventing task paralysis.

2.2.1 Integration with Rosenshin‘s Principles of Instruction

While Sweller provided the theoretical framework, Barack Rosenshine provided a “how-to” to be used in classrooms or digital environments [19].

Main connections to Sweller‘s principles include:

- Small Steps (Principle 2): By presenting new material in small increments, the instructor limits the Intrinsic Load, ensuring the working memory is not overwhelmed at the outset.
- Scaffolding (Principle 7): Providing temporary supports reduces Extraneous Load, allowing the learner to focus their cognitive resources on the core task rather than the “noise” of the process.

To view these from more of a development lens, principle 2 supports breaking down the core task into smaller, more manageable steps. Further more these smaller task can be seen as “scaffolding” for the user when they are beginning a task without handing them the answer directly.

2.2.2 The Cognitive Theory of Multimedia Learning (CTML)

While Sweller’s Cognitive Load Theory focuses primarily on the nature and management of cognitive load, Mayer’s Cognitive Theory of Multimedia Learning (CTML) provides a more detailed explanation of how information is processed across visual and auditory channels. Within the context of software development, the IDE can be understood as a high-density multimedia environment in which multiple streams of related information must be processed simultaneously.

Within this framework, Ayres and Sweller's split-attention principle describes the split-attention effect as occurring when learners are required to divide their attention between physically separated yet cognitively related sources of information [1]. In a software development context, this may occur when an error message in the console refers to a line of code located hundreds of lines away in the editor. For a neurodivergent developer, this creates a significant spatial split, requiring additional mental effort to navigate between and mentally integrate these sources of information. This integration process contributes directly to extraneous cognitive load, as cognitive resources are expended on locating and connecting information rather than resolving the underlying programming problem.

To mitigate this, the plugin's "Code Explainer" feature will provide an integrated view that combines the error message with the relevant code context, effectively reducing the split-attention effect. By presenting the error message and the associated code in a unified interface, the plugin minimizes the need for mental integration across separate sources, thereby reducing extraneous cognitive load and allowing developers to focus more effectively on problem-solving.

2.3 Neurodiversity in Software Engineering

To best design an inclusive plug-in, I needed to gain a deep understanding of the specific cognitive roadblocks faced by neurodivergent developers. Recent research by Verma et al. [24] and Gama et al. [6] highlights that developers with ADHD, ASD, and Dyslexia face significantly higher frequency of interruptions and difficulty obtaining knowledge.

2.3.1 The Cognitive Load Connection

For a neurodivergent developer, a standard IDE error message often carries a high Extraneous Load. What a neurotypical developer might perceive as a simple notification, a neurodivergent individual may experience as a wall of cryptic text, inconsistent highlighting, and ambiguous phrasing. This noise competes for limited working memory resources, leaving little room for the Intrinsic Load required actually to solve the logical problem.

2.3.2 Executive Functioning and Task Switching

executive dysfunction is a primary barrier cited by both Verma et al. [24] and Gama et al. [6]. This manifests itself as difficulty in "task initiation" (starting a new project) and "set-shifting" (switching between tasks).

The Industry Context: In a modern software environment, developers are rarely afforded the luxury of a single focus; they must balance code reviews, meetings, and multiple projects.

The Impact: For those with ADHD, the mental cost of having to remember the context of a project after an interruption is significantly higher. If the IDE does not provide clear "waypoints" or historical context, the developer may become cognitively paralysed, unable to determine where they left off.

2.3.3 Specific Challenges by Condition

Based on the synthesis of recent studies, the challenges can be categorised as follows:

ADHD: Frequent loss of"state" due to external interruptions and internal distraction. [24] found that these developers report higher stress related to "waiting for answers," as delays in information retrieval often lead to a total loss of task momentum.

ASD: Challenges with the "Double Empathy" problem in code—where the high-level abstractions or "implied logic" used by neurotypical colleagues create a high cognitive overhead for autistic developers who prefer explicit, literal documentation.

Dyslexia: Significant hurdles during the "syntax-heavy" stages of programming. While these developers often excel at high-level architectural mapping, the Extraneous Load of spelling-sensitive compiler errors and dense documentation can be a major deterrent to productivity.

2.3.4 Beyond Medical

To understand why existing tools fail, we must examine the shift from the Medical Model to the Social Model of disability.

- The Medical Model: Traditionally, ADHD or Dyslexia in engineering has been viewed as a "deficit" to be corrected through individual coaching or medication.
- The Social Model: This perspective argues that the developer is "disabled" not by their neurology, but by an IDE designed with a neurotypical bias. The problem is not the individual's brain, but the environment that fails to accommodate it. This shift in perspective is crucial for designing tools that are not just "assistive" but truly inclusive, as it focuses on modifying the environment (the IDE) rather than trying to fix the individual.

The minimalist toggle and the code explainer are designed with this social model in mind, aiming to create an environment that reduces barriers and allows neurodivergent developers to thrive without needing to conform to a neurotypical standard.

Executive Functioning (EF) Domains in Programming To deepen the analysis of executive dysfunction, we must break it into three core domains:

- Inhibitory Control: The ability to ignore distractions. For ADHD developers, the "pop-up" notifications of a linter can break a fragile state of focus.
- Working Memory: The "mental scratchpad." Dyslexic developers often have a high capacity for 3D spatial reasoning but a limited verbal working memory, making syntax-heavy languages like C++ more taxing than "cleaner" languages like Python.
- Cognitive Flexibility: The ability to switch between thinking about the "big picture" architecture and the "low-level" syntax.

2.3.5 The “Double-Edged Sword” of Hyperfocus

While neurodivergent developers, particularly those with ASD and ADHD, are often stereotyped as having a “superpower” for deep work, this ability to hyperfocus is more accurately described as a dysregulation of attention. Unlike the psychological concept of “Flow” which is generally sustainable and self-directed, hyperfocus is an involuntary immersion into a task or activity. As noted by [6], hyperfocus allows for an exhaustive exploration of complex problems, yet this intensity often comes at the expense of self-care. Developers may neglect fundamental biological needs, such as nutrition, sleep, and hygiene, leading to rapid burnout.

In the context of software engineering, this creates a High-Intensity/High-Fragility paradox. [15] identifies that for ADHD programmers, “getting in the groove” is a fragile process where a single interruption can collapse hours of built-up mental context. The subsequent state recovery cost is significantly higher for neurodivergent individuals, often leading to burnout or irritability when the state is broken.

In a professional engineering context, this creates a significant conflict between depth and delivery. Hyperfocus can inadvertently trigger over-engineering, where a developer becomes locked into optimising a single component while neglecting broader project milestones or team collaborations. This inability to switch off from an interesting problem makes task-switching, a fundamental requirement of agile development, mentally exhausting and practically difficult. Consequently, what appears to be peak productivity from the outside is often an unsustainable trade-off that risks both the developer’s well-being and the project’s timeline.

2.4 Existing Tools

The design of IDEsoften follows a one-size-fits-all model, which inherently leads to a neurotypical bias, creating barriers for those who may require additional support. [14] observes workplace tools are frequently built with assumptions that "may not align with the needs of neurodiverse employees." IDEs are often packed with tools that enable customisation; however, they are not designed with neurodiverse developers in mind, and complex nested menus can be overwhelming.

Despite the obstacles described, certain features provide essential support for developers. [14] notes that tools such as code completion and linters are vital to alleviate the difficulties associated with dyslexia, such as syntax memorisation and spelling.

However, the burden of making tools accessible often falls to the user. [11] found that developers often create “individual strategies and tool configurations to support their productivity". While these features are currently helping, the reliance on manual customisation implies a lack of "out-of-the-box" inclusivity.

This project aims to bridge the gap between the availability of generic tools and their functional utility. While current tools are excellent at detecting errors, they often fail at explaining them in an accessible manner. By integrating an Error Explainer that decomposes complex compiler messages into manageable tasks, therefore reducing the cognitive load required to debug. This shifts the IDE from a passive environment that requires expert configuration to an active partner in the development process.

2.5 Pomodoro Timer

The Pomodoro Technique, developed by Francesco Cirillo in the late 1980s, is a time-management framework that partitions work into focused intervals. Typically 25 minutes, separated by brief five-minute breaks. In the context of neurodiversity, this method is often reframed as a "cognitive tool" rather than a rigid rule set, specifically designed to address common executive functioning barriers [12].

2.5.1 Managing Executive Dysfunction and Task Paralysis

For developers with ADHD or ASD, task paralysis, an inability to initiate work due to cognitive overload or anxiety represents a significant hurdle. The Pomodoro Technique mitigates this by lowering the "activation energy" required to start. By committing to a manageable, 25-minute sprint, the psychological barrier to entry is reduced, helping individuals bypass the freeze response associated with daunting or abstract tasks.

2.5.2 Combatting Time Blindness and Hyperfocus Burnout

Neurodivergent individuals often experience "time blindness", a difficulty in perceiving the passage of time as a concrete resource. Externalising time through a visual or auditory timer provides real-time feedback, helping to anchor the developer to the present task and reducing the risk of underestimating project timelines.

Furthermore, while hyperfocus is frequently cited as a "superpower" for developers, it can lead to severe mental and physical exhaustion if left unregulated. The structured nature of the Pomodoro cycle acts as a "release valve" for focus:

- Mandatory Breaks: Frequent breaks prevent the "hyperfocus trap," where a developer may neglect basic biological needs like nutrition or hydration.
- Sensory Regulation: Scheduled pauses provide essential space for sensory recovery, allowing developers to move around to prevent the physical strain and irritability that often follow hours of sedentary deep work.

2.5.3 Cognitive Load and Adaptive Implementation

From the perspective of Cognitive Load Theory (CLT), the Pomodoro Technique manages mental resources by preventing the exhaustion of the "cognitive budget". By breaking a complex project into small, time-limited stages, the method reduces Extraneous Load, the unnecessary effort required to manage a vast, unstructured task—allowing for more Germane Load to be directed toward core algorithmic problem-solving.

However, In some cases it could be argued that the standard Pomodoro settings can sometimes be counterproductive if applied too rigidly. For individuals with ADHD, a 25-minute window may be too short to enter a productive "flow state", leading to "state recovery costs" if interrupted too

early. To address this, the plugin will allow users to customise the length of their focus intervals and breaks, providing flexibility to accommodate different working styles and cognitive needs.

2.6 Colour filters

Visual presentation can influence reading performance and comfort.Griffiths et al. [7] found that coloured overlays may reduce visual stress for some readers, although the benefits vary significantly between individuals. This suggests that software tools should support flexible visual customisation rather than relying on a single accessibility configuration.

In some cases, high-contrast models are essential because they provide the necessary visual scaffolding to combat task paralysis. For developers with ADHD, a low-contrast or pastel theme can cause code elements to bleed together, increasing the cognitive effort required to distinguish between keywords, variables, and logic. By utilizing high-contrast ratios—such as 7:1 or higher, per WCAG 2.1 guidelines—the IDE creates a clear visual hierarchy.

This high "signal" environment provides several technical advantages:

- Reduce Visual Search Time: Finding the start and end of a function block becomes instantaneous rather than a manual scan.
- Mitigate the "Blur" of Hyperfocus: As noted by [15], the intense mental strain of being "in the groove" can lead to visual fatigue; high contrast maintains legibility even as physiological focus wanes.
- Support Syntax Logic: In complex, nested code, high-contrast separation between brackets and operators serves as a "visual prosthetic," offloading the work of pattern recognition from the brain to the interface.

2.6.1 The Halation Risk and Soft High Contrast

While high contrast is vital for clarity, pure high contrast (e.g., stark white text on a pure black background) can be counterproductive for those with ASD or specific visual stresses. This combination can cause halation, a glowing or bleeding effect around letters that makes text appear to vibrate or move [5].

To address this, the most effective models for neurodiversity are often "Soft High Contrast" themes. These utilize high-visibility colors against deep, non-pure-black backgrounds (such as dark greys or navy). This balance provides the necessary signal for ADHD developers to maintain focus while preventing the sensory overwhelm and visual distortion common in ASD profiles. However, the implementation must be nuanced. While high contrast is vital for clarity, high contrast can cause halation (a glowing effect around letters) for those with ASD or visual stress. Therefore, the most effective high-contrast models for neurodiversity are often "Soft High Contrast", utilising high-visibility colours against deep non-pure-black backgrounds.

2.7 AI as a Cognitive Tool: Task Decomposition and Scaffolding

For developers with ADHD and ASD, the most significant barrier to productivity is often not the complexity of the code itself, but the executive function tax required to organise a project. Research by [15] indicates that ADHD professional programmers specifically struggle with state recovery and task initiation. LLMs offer a unique solution to this through cognitive scaffolding. This process handles the “low-value” mental energy drains, such as formatting and scheduling, leaving more room for “high-value” deep thinking ([8]).

Chain of Thought (CoT) as Task Decomposition is one of the most effective applications of LLMs for neurodivergent developers. Instead of asking a model to simply “write a function,” the developer can use the model to decompose a broad, abstract goal into a granular, step-by-step plan. This process mitigates the “activation energy” barrier identified in ADHD populations. By prompting a model to “think step-by-step," the AI generates a logical roadmap for the developer to work through, as well as referring back to it if the task is interrupted.

As noted by Mew (2025), this “automated pattern” of scaffolding can increase productivity for developers with ADHD by up to 55 percent. Existing tools already demonstrate this by taking a single overwhelming command (e.g., “build an authentication system”) and breaking it into manageable milestones. This effectively transforms AI from a simple code generator into a “digital ramp” that bypasses the freeze response common in task paralysis ([8]).

2.8 NASA-TLX and SUS: Measuring Cognitive Load and Usability

To evaluate the effectiveness of the plugin, I will use the NASA Task Load Index (NASA-TLX) to measure cognitive load and the System Usability Scale (SUS) to assess usability. The NASA-TLX provides a multidimensional assessment of perceived workload across six dimensions: Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration. This will allow me to quantify the reduction in cognitive load achieved by the plugin. The SUS, on the other hand, is a widely used tool for assessing the usability of systems and interfaces. It consists of a 10-item questionnaire that provides a global view of subjective assessments of usability. By using both tools, I can gain insights into how well the plugin reduces cognitive load while also ensuring that it is user-friendly and accessible for neurodivergent developers.

I have chosen these tools because they are well-established in the field of human-computer interaction and provide a comprehensive framework for evaluating both the cognitive and usability aspects of the plugin. The NASA-TLX will help me understand the specific areas where cognitive load is reduced, while the SUS will provide feedback on the overall user experience. This dual approach ensures that the plugin is not only effective in reducing cognitive load but also meets the usability needs of its target audience.

3 Design and Implementation

3.1 User Requirements

The requirements for the plug-in were derived from a synthesis of Cognitive Load Theory and specific neurodivergent accessibility guidelines, notably the direct instructions and visual comfort frameworks identified by Rocha [17].

Table 2: Functional and Non-Functional Requirements for the Accessible IDE Plugin

ID	Category	Requirement Description	Theoretical Justification
FR1	Interface	The system shall provide a single-click "Minimalist Toggle" to simultaneously hide the Activity Bar, Side Bar, Status Bar, and Breadcrumbs.	Direct suppression of Extraneous Cognitive Load (EL) through reduction of visual stimuli, supporting inhibitory control and minimising sensory overload in line with Cognitive Load Theory.
FR2	AI Support	The "Task Architect" shall decompose high-level prompts into a structured checklist of 10 manageable sub-tasks, each containing micro-steps.	Scaffolding of Intrinsic Load (IL) to reduce executive dysfunction and task initiation barriers through step-wise decomposition and cognitive scaffolding principles.
FR3	Feedback	The "Code Explainer" shall intercept compiler errors and provide non-technical bulleted summaries (maximum three points).	Reduction of EL by converting jargon-heavy diagnostic messages into direct instruction, aligned with split-attention reduction and plain-language accessibility principles.
FR4	Time Management	The system shall provide a customisable Pomodoro timer integrated into the dashboard, allowing user-defined focus and break durations.	Supports executive functioning through temporal externalisation, mitigating time blindness and hyper-focus burnout commonly reported in ADHD and ASD literature.
FR5	Visual Presets	The system shall provide at least three predefined typography presets (Standard, Relaxed, Spacious) that simultaneously adjust line height, letter spacing, and font family.	Reduces visual stress and improves readability for dyslexic users through evidence-based typography adaptation and visual comfort frameworks.
NFR1	Accessibility	The UI must conform to WCAG 2.1 Level AA contrast ratios (minimum 4.5:1 for standard text).	Ensures inclusive visual accessibility and compliance with recognised accessibility standards.
NFR2	Cognitive	The system must provide non-destructive restoration, allowing users to instantly revert all interface changes.	Reduces verification anxiety and supports psychological safety by preserving user agency over the environment.
NFR3	Latency	AI-generated task steps must begin streaming within 2 seconds of request initiation.	Minimises interruption-related attentional drift and prevents task-switching overhead caused by system delays.
NFR4	Sensory Regulation	Theme changes must be applied instantaneously across the entire workbench.	Provides immediate perceptual feedback, reducing uncertainty and sensory verification load.

3.2 Module Design

The plugin is architected around three core modules, each designed to address specific executive functioning barriers. These modules are unified within the Accessible Dashboard, shown below. This Dashboard is a centralized Webview panel that provides a low-friction entry point to the tool's features.

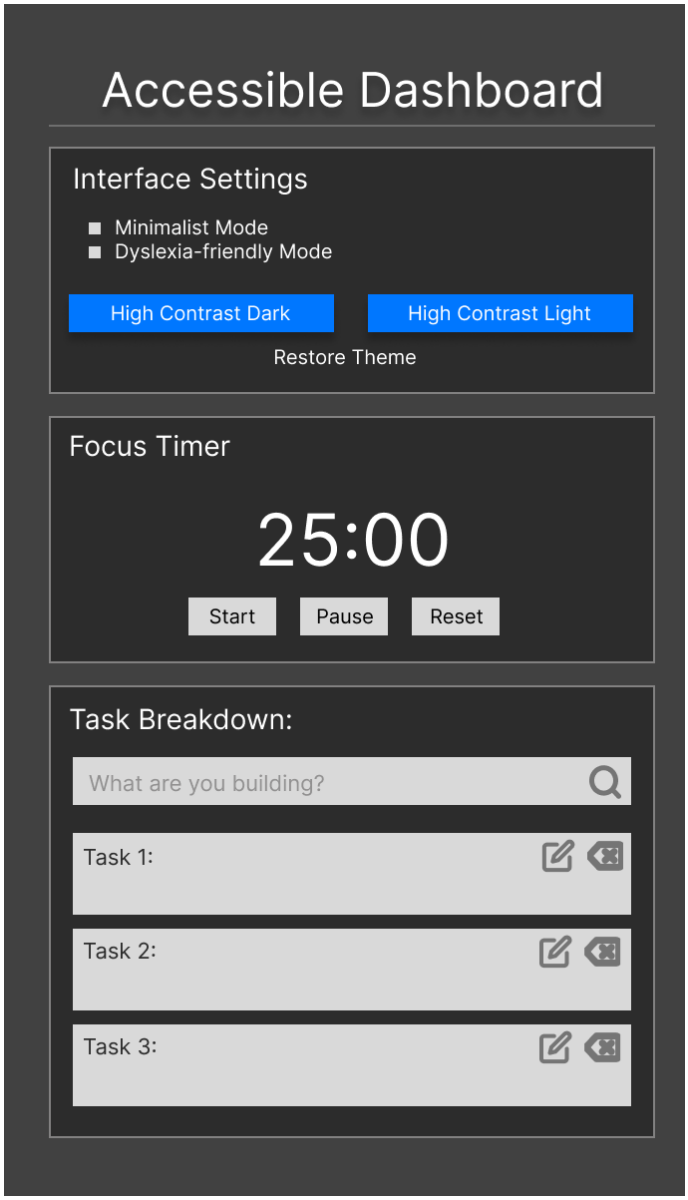

Figure 2: UI Plan For the Accessible IDE Plugin.

I first designed the UI in figma, which allowed me get a clear vision of how the plugin would look and function. This also allowed me to make sure that the design of the plugin was in line with the requirements that I had set out in the previous section. Before I designed the UI I was already aware that I would implement the UI using HTML and CSS in a Webview, which allowed me to create a custom dashboard that could be easily accessed from within VS Code.

3.3 Technology Stack

The development of the plugin utilizes a modern web-based stack compatible with the VS Code extension ecosystem to ensure high performance and seamless API integration.

- JavaScript: The primary programming language used for both the extension backend and the Webview frontend. It was chosen for its native compatibility with Node.js and its ability to handle asynchronous message passing between the UI and the extension host.
- Node.js: Acts as the runtime environment for the extension, providing the necessary APIs for file system access, process management, and the execution of imperative commands within the IDE.
- VS Code Extension API: The core framework used to manipulate the IDE environment. Specifically, it facilitates interaction with workspace.getConfiguration for visual styling and vscode.commands for interface simplification.
- OpenAI API / VS Code Language Model API: Provides the generative intelligence required for the Task Architect and Code Explainer. By utilizing the resident gpt-4o model, the plugin performs complex Chain of Thought (CoT) reasoning without requiring the user to provide personal API keys.
- HTML/CSS: Used within the Webview layer to construct the Accessible Dashboard. This allows for the implementation of custom high-contrast themes and dynamic scaling that exceeds the limitations of standard VS Code CSS.

Before I started the implementation, I had a rough idea of the technology stack that I would use. In terms of plugin development, the two main languages that are used are JavaScript and TypeScript. Although Typescript is more useful if the project was larger, I felt that JavaScript would be sufficient for the scope of this project and would allow me to get up and running faster, especially due to the fast turn around time of the project. JavaScript was also chosen for its native compatibility with the VS Code extension host and its ability to handle asynchronous message passing between the UI and backend. [23]

For the AI features, I decided to use the gpt-4o model through the VS Code Language Model API, which allows me to leverage powerful language models without requiring users to manage their own API keys. Finally, I chose HTML and CSS for the Webview UI to allow for maximum flexibility in designing the dashboard and implementing custom themes.

3.3.1 Minimalist Toggle

The Minimalist Toggle serves as a reset for the IDE. It utilises a state-management system to track the current visibility of the VS Code workbench. When activated, it dispatches a batch of imperative commands to the vscode.commands API, hiding the Activity Bar, Status Bar, Side Bar, and Breadcrumbs simultaneously. By reducing the total number of visual elements on the screen, the module lowers extraneous cognitive load, allowing the developer to allocate more mental energy

to the code editor itself. When I first had the idea for this toggle I was vaguely aware that I would need to use the vscode.commands API to manipulate the UI.

3.3.2 Task Architect - AI-Driven Scaffolding

The Task Architect is a Human-in-the-Loop (HITL) module that facilitates task initiation by breaking down overwhelming objectives into granular steps. The module should employs a CoT prompting strategy to take a complex task and break it down into simpler sub-tasks. The generated steps are rendered as editable fields within the dashboard, allowing users to modify or refine the AI's suggestions. This ensures that developers maintain agency over their project architecture while the AI handles the initial decomposition.

Overcoming Latency and Verbosity

During the testing of various prompt structures, a recurring challenge emerged, how much unnecessary text was included in the LLM's output. When asked to assist, the model frequently included excessive supportive dialogue. While intended to be user-friendly, this extra text hindered the system's ability to parse the output into discrete, editable UI components.

To solve this issue, the prompt was refined to enforce a strict structural output, prioritizing data over dialogue. This ensured that the generated steps could be seamlessly rendered as editable fields within the dashboard, preserving the developer's agency to modify or refine the architecture without sifting through conversational filler.

Balancing Granularity and Utility

Initial experiments with simple prompts such as "How to code a login page", the LLM would bypass the architectural phase and provide a block of code immediately. This was counterproductive for two reasons:

- It failed to provide a conceptual roadmap for the user.
- It increased the technical hallucination risk by attempting to solve a high-level task in a single step rather than planning the logic first.

Increased Steps

The original design specifications for the Task Architect called for 3 to 5 broad sub-tasks. However, user testing revealed that this flat structure remained overwhelming. A single step like "Set up the backend" was still too abstract for many users to execute confidently. To address this, I implemented a hierarchical decomposition strategy. The system is now instructed to generate a primary set of sub-tasks, but with a critical addition: each sub-task must include two distinct micro-steps.

3.3.3 Visual Styler

The Visual Styler is a specialized UI/UX module designed to manage the IDE's aesthetic environment, specifically targeting the reduction of eye strain and sensory overwhelm. By prioritizing

visual ergonomics, the module ensures that the development environment remains inclusive for neurodivergent users and those with visual impairments.

Accessibility and WCAG Mapping
To ensure the platform is accessible to the widest possible audience, the Visual Styler utilizes a systematic mapping of presets based on the Web Content Accessibility Guidelines (WCAG) 2.1 Level AA.

- High Contrast Modes: The module provides dedicated "High Contrast Dark" and "High Contrast Light" themes. These presets are programmatically validated to ensure a minimum contrast ratio of 4.5:1 for standard text and 3:1 for large text or UI components.
- Sensitivity Accommodations: By offering these specific color profiles, the IDE addresses the needs of users with color vision deficiencies (CVD) and photophobia (light sensitivity), allowing for a high-degree of legibility without triggering sensory fatigue.

Technical Implementation and Functionality
The Visual Styler operates by interacting directly with the workspace.getConfiguration API. This allows the module to move beyond surface-level CSS changes and manipulate the core structural rendering of the workspace:

Dynamic Scaling: Users can adjust font sizes and line heights across the entire interface. Increasing the line height (e.g., to 1.5 or 2.0) is a critical intervention for developers with dyslexia, as it reduces the crowding effect of text and improves character recognition.

Granular Customization: Unlike traditional static themes, the Styler follows a visual comfort framework. It treats font-weight, letter spacing, and padding as variables that the user can tune to their specific cognitive load requirements.

User Agency and Sensory Regulation By providing an intuitive dashboard for these adjustments, the Visual Styler empowers the developer to curate their own sensory environment. This reduces the cognitive energy spent on deciphering a cluttered or poorly-lit UI, allowing that energy to be redirected toward the primary task of programming. The module essentially acts as a sensory filter, ensuring that the IDE adapts to the user‘s biological needs rather than forcing the user to adapt to a rigid interface.

3.4 System Architecture

The system is divided into three primary layers: the Presentation Layer (Webview), the Orchestration Layer (Extension Host), and the External Services Layer (VS Code API).

1. The Presentation Layer: WebView UI

The user interface is implemented as a `WebviewPanel`, which renders a custom HTML/JavaScript dashboard. This allows for a highly flexible UI. This allows for the extension to be customised to best fit the user's needs, predetermined through prior research. This also allows for easier changes as more research is collected in the future

- State Communication: Because WebViews are isolated from the main editor, communication is handled via asynchronous message passing. The UI dispatches JSON-formatted messages using `vscode.postMessage()`, which are then intercepted by the extension's backend.
- Asynchronous Feedback: The UI remains responsive by using event listeners that wait for `updateTime`, `taskResult`, or `errorResult` commands sent back from the extension host.

2. The Orchestration Layer: Extension Host

The core logic resides in the activate function, which serves as the central hub while the extension is active.

- Command Registration: The system registers specific URI commands (e.g., accessible-toggle.showUI) that map user intents to internal JavaScript functions.
- Configuration Management: A key architectural feature is the interaction with vscode.workspace.getConfig The extension reads and writes global settings to modify the editor's environment (font size, themes, and layout) in real-time, effectively acting as a wrapper for the VS Code Settings engine.
- Context Preservation: To ensure the system is non-destructive, the architecture utilises `context.globalState`. This allows the extension to store the user's original theme and font settings before applying accessibility modifications, enabling a seamless restoration process. This also acts as a form of error prevention in case any settings are applied by accident.

3. The Functional Providers

The architecture employs specialised logic blocks to handle distinct accessibility needs:

Table 3: System Component Interaction and API Dependencies

Component	**Interaction Strategy**	**VS Code API Dependency**
Interface Controller	Imperative Commands	`commands.executeCommand`
Theme/Font Engine	Configuration Updates	`workspace.getConfiguration`
Task Architect	AI Stream Processing	`vscode.lm` (Language Model)
Code Helper	Diagnostic Polling	`languages.getDiagnostics`

4. Integration with Language Models

The "Task Architect" and "Code Helper" features utilise the VS Code Language Model API. This architectural choice ensures security and efficiency, as the extension does not require individual API keys from the user. Instead, it requests access to the editor's resident models (e.g., GPT-4o). The system uses a CoT prompting strategy to process raw diagnostic data or user goals into structured, neurodiverse-friendly steps before pushing the data back to the Presentation Layer.

3.5 UI Simplification and Minimalist Toggle

The implementation of the Minimalist Mode focuses on reducing visual noise, which research identifies as a primary driver of cognitive overload and sensory overwhelm for autistic and ADHD developers. This module acts as a digital ramp, filtering environmental noise to prevent the exhaustion of a developer's limited cognitive budget.

The technical execution follows an imperative command pattern to provide a seamless, one-click transition:

- **Workspace Manipulation:** Using the vscode.commands.executeCommand API, the plugin simultaneously toggles the visibility of the Activity Bar, Side Bar, Status Bar, and Breadcrumbs.
- **Global Configuration:** To remove persistent distractions like the Mini-map, the system interacts directly with global user settings using config.update('editor.minimap.enabled', ...).
- **State Management:** The module utilizes a state-management system to track the current visibility of the VS Code workbench.
- **Non-Destructive Restoration:** To mitigate anxiety regarding lost settings, the architecture employs context.globalState to store the original UI configuration. This allows users to revert all changes instantly, satisfying the requirement for a safe and reversible interface.

By centralizing these actions into a single toggle, the plugin lowers Extraneous Cognitive Load (EL). This allows the developer to reallocate mental resources from navigating a complex, high-density interface toward Germane Load (GL), the actual logical problem-solving and programming tasks

Final UI/Functionality Integration (FR1)

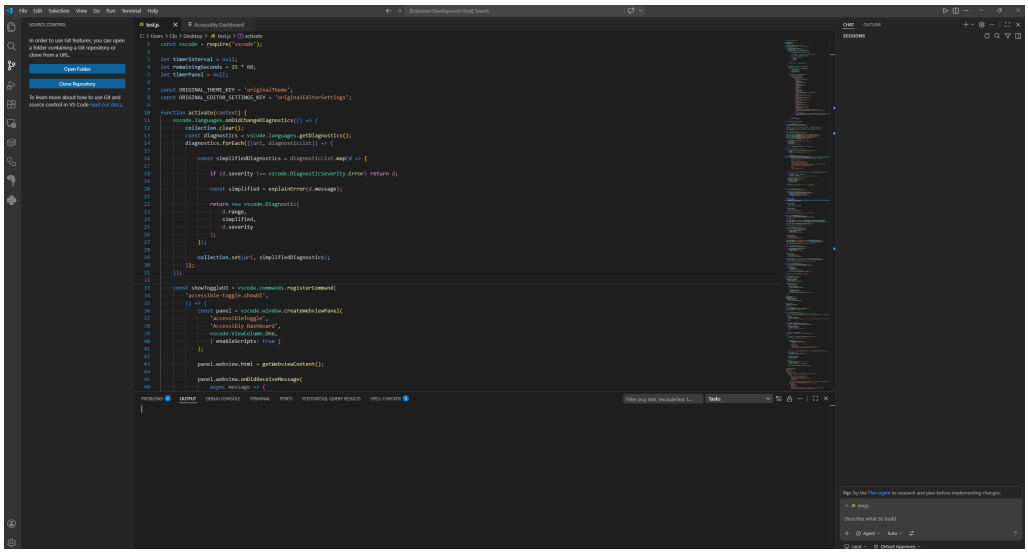

Figure 3: Pre-Minimalist UI, showing the default VS Code interface with all standard elements visible.

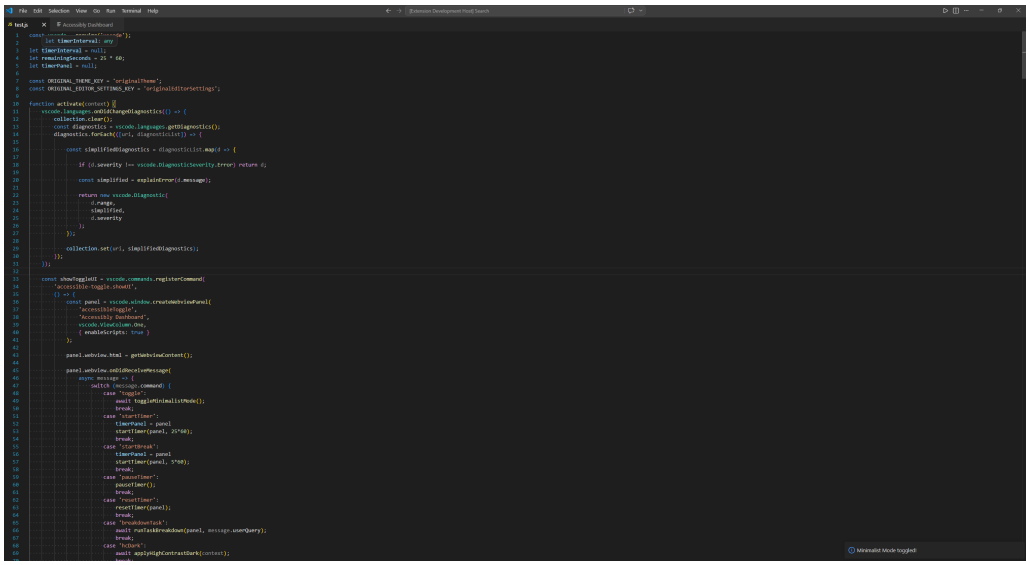

Figure 4: Post-Minimalist UI, showing the simplified VS Code interface with reduced visual elements.

Figure 5: Notification confirming the activation of Minimalist Mode, providing immediate feedback to the user.

3.6 AI Integration (FR2 and FR3)

The AI-driven features are implemented using a Streaming Data Pattern to satisfy NFR3 (Latency), ensuring the interface remains responsive during complex generation.

3.6.1 Task Architect: Overcoming Task Paralysis

- **Prompt Engineering:** The *Task Architect* uses a strictly defined system prompt to enforce a micro-step structure (3–5 sub-tasks with two 5-minute micro-steps each). This prevents the LLM from generating a long response that can be overwhelming.
- **Iterative Prompt Refinement:** During development, I identified a significant limitation where the LLM included excessive supportive dialogue (e.g., “Sure! I can help with that”). While polite, this violated the principle of Direct Instruction and hindered the system’s ability to parse the text into editable UI components. I refined the prompt to mandate a strict data-only format, ensuring the system only outputs actionable tasks.

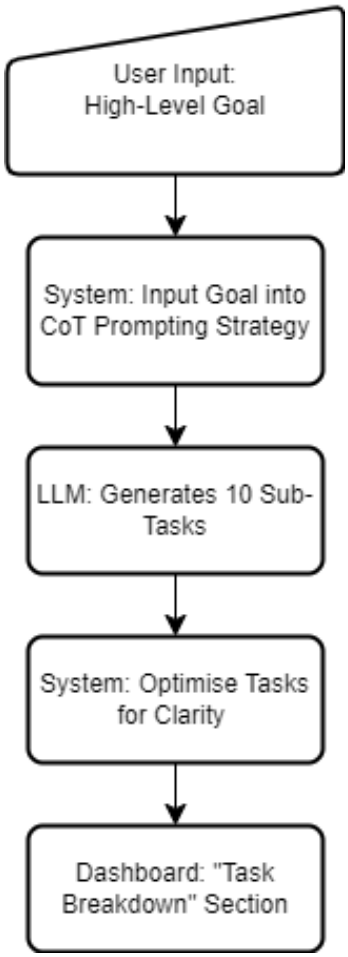

Figure 6: AI Flow for the Task Architect Module.

The Figure 6 demonstrates the process of the Task Architect module. When a user enters a high-level goal (e.g., “Implement a search bar”), the system instructs the LLM to output 3-5 actionable sub-tasks with 2 additional tasks from each of the sub-tasks.

```
const messages = [
    vscode.LanguageModelChatMessage.User(
        `Act as a Task Architect for a neurodiverse developer.
        The goal is to provide 'Task Initiation' support.
        Please simply state list with no other explanations, preambles, or encouragement.

        USER GOAL: "${userQuery}"

        INSTRUCTIONS (Chain of Thought Strategy):
        1. Identify the high-level objective.
        2. Break this down into 3-5 major sub-tasks.
        3. For each sub-task, provide 2 very small 'micro-steps' that take less than 5 minutes.
        4. DO NOT provide any code snippets or technical syntax.
        5. Use clear, encouraging, and actionable language.

        FORMAT:
        - Use Markdown bold for sub-tasks.
        - Use [ ] for micro-steps.`
    )
];
```

Figure 7: Code written to generate tasks

Not only does the this prompt ask the LLM to break down the task, but it also instructs it to format the output in a specific way. So that the response can be easily broken down and cleaned for the output the prompt includes instructions to make sure the subtasks in bold and the micro-steps are preceded by "[ ]".

3.6.2 Human-in-the-Loop (HITL) Implementation

As shown in Figures 8 and 9, the generated tasks are rendered in the dashboard as editable fields. This is a critical design choice to preserve User Agency. Neurodivergent developers often have unique workflows so by allowing the user to edit or delete AI-generated steps, the tool moves from being a prescriptive authority to a flexible partner. The graying-out of completed tasks provides immediate visual feedback, supporting working memory by externalizing task progress.

```
--- PLAIN AI OUTPUT START ---
**Objective: Create a login page**

**Sub-task 1: Set up the project environment**
- [ ] Open Visual Studio Code and create a new folder for the project.
- [ ] Initialize the project with a basic file structure (e.g., HTML, CSS, JS files).

**Sub-task 2: Design the login form layout**
- [ ] Sketch or outline the form fields (e.g., username, password) on paper or a digital tool.
- [ ] Add placeholders or labels for each field in the HTML file.

**Sub-task 3: Style the login form**
- [ ] Open the CSS file and define basic styles for the form (e.g., width, padding).
- [ ] Adjust the alignment and spacing of the form elements.

**Sub-task 4: Add functionality to the form**
- [ ] Write a script to handle the form submission event.
- [ ] Add basic validation for the input fields (e.g., check if fields are empty).

**Sub-task 5: Test the login page**
- [ ] Open the page in a browser and test the form layout and functionality.
- [ ] Use the integrated terminal to check for any errors in the console.
--- PLAIN AI OUTPUT END ---
```

Figure 8: Example of the output generated by the Task Architect module, showing the micro-steps for a given task.

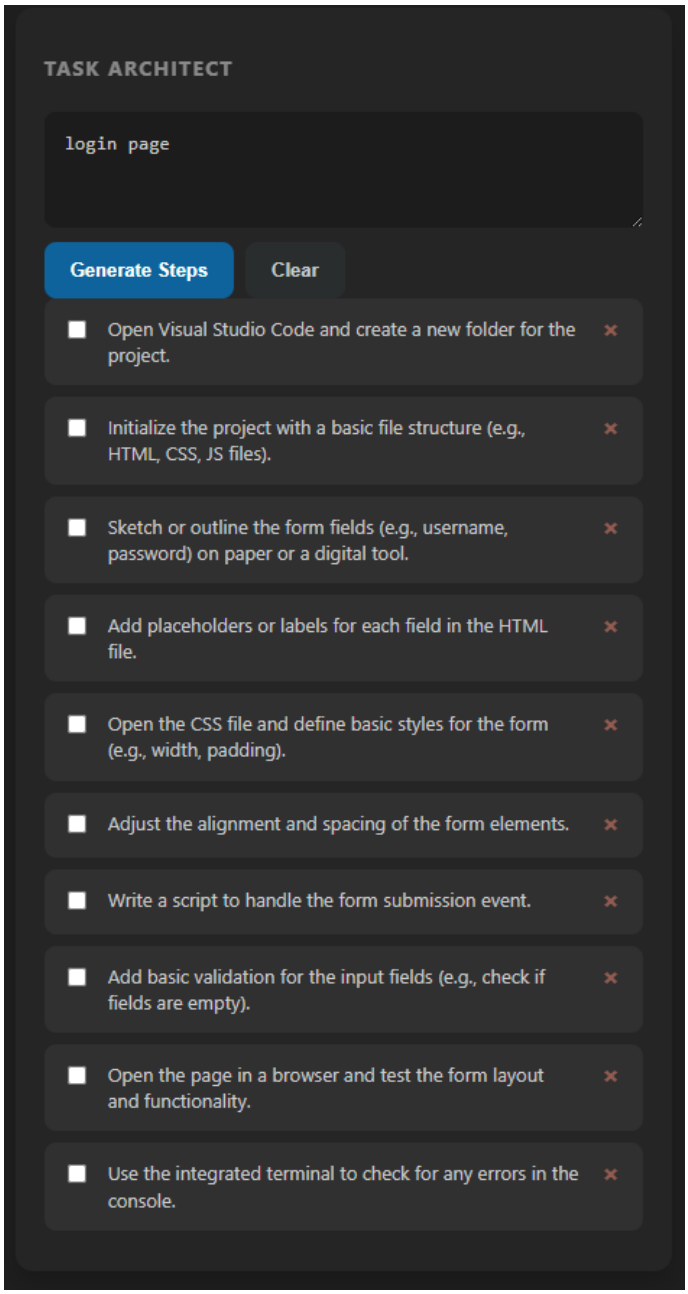

Figure 9: Task Architect Module in the Dashboard, showing the generated tasks.

3.7 Dyslexia Settings: Test spacing (FR5)

The Visual Styler module allows users to adjust the spacing between characters and line height in the IDE. This is particularly beneficial for developers with dyslexia, as increasing letter spacing and vertical padding can significantly reduce the crowding effect in dense code, improving character recognition and overall readability.

To balance customisability with cognitive ease, the plugin provides a set of predefined spacing presets, Standard, Relaxed, and Spacious, for the users to choose from in the dashboard. This allows for quick adjustments without overwhelming the user with too many options, while still providing the necessary customization to accommodate different cognitive needs.

The presets implement a hierarchical increase in visual breathing room:

- Standard: Acts as a soft reset to default editor settings ($lineHeight$: 0, $letterSpacing$: 0) using the system's standard font family.
- Relaxed: Increases the $lineHeight$ to 30 and $letterSpacing$ to 0.8, introducing specialized fonts such as Lexend and OpenDyslexic.
- Spacious: Offers the maximum level of separation with a $lineHeight$ of 40 and $letterSpacing$ of 1.5. This preset prioritizes OpenDyslexic as the primary font to further enhance character distinction.

The inclusion of OpenDyslexic and Lexend is a critical design choice; these fonts use weighted bottoms or unique shapes to help the brain distinguish between similar characters (e.g., "b" vs "d"), which is a common hurdle for dyslexic readers. By offering these as a unified "Spacious" preset, the plugin provides a high-signal environment that accommodates diverse cognitive needs with minimal interaction cost.

3.7.1 Initial Plan: One Option

Initially, the plugin was programmed with fixed values for line height and letter spacing. However, user testing revealed that a one-size-fits-all approach was insufficient, as individual cognitive needs regarding text density vary significantly.

The figures below demonstrate the visual impact of these adjustments. Figure 10 displays the standard VS Code environment, while Figure 11 illustrates the enhanced readability achieved by increasing the line height to 1.5 and the letter spacing to 0.5em.

```
1   const vscode = require('vscode');
2
3   let timerInterval = null;
4   let remainingSeconds = 25 * 60;
5   let timerPanel = null;
6
7   const ORIGINAL_THEME_KEY = 'originalTheme';
8   const ORIGINAL_EDITOR_SETTINGS_KEY = 'originalEditorSettings';
9
10  function activate(context) {
11      vscode.languages.onDidChangeDiagnostics(() => {
12          collection.clear();
13          const diagnostics = vscode.languages.getDiagnostics();
14          diagnostics.forEach(([uri, diagnosticList]) => {
15
16              const simplifiedDiagnostics = diagnosticList.map(d => {
17
18                  if (d.severity !== vscode.DiagnosticSeverity.Error) return d;
19
20                  const simplified = explainError(d.message);
21
22                  return new vscode.Diagnostic(
23                      d.range,
24                      simplified,
25                      d.severity
26                  );
27              });
28
29              collection.set(uri, simplifiedDiagnostics);
30          });
31      });
32
```

Figure 10: Example of the difference in spacing between the default settings and the original dyslexia settings, where the line height is set to 1.5 and the letter spacing is set to 0.5em.

```
const vscode = require('vscode');

let timerInterval = null;
let remainingSeconds = 25 * 60;
let timerPanel = null;

const ORIGINAL_THEME_KEY = 'originalTheme';
const ORIGINAL_EDITOR_SETTINGS_KEY = 'originalEditorSettings';

function activate(context) {
    vscode.languages.onDidChangeDiagnostics(() => {
        collection.clear();
        const diagnostics = vscode.languages.getDiagnostics();
        diagnostics.forEach(([uri, diagnosticList]) => {

            const simplifiedDiagnostics = diagnosticList.map(d => {

                if (d.severity !== vscode.DiagnosticSeverity.Error) return d;

                const simplified = explainError(d.message);

                return new vscode.Diagnostic(
                    d.range,
                    simplified,
                    d.severity
                );
            });

            collection.set(uri, simplifiedDiagnostics);
        });
    });

```

Figure 11: Example of the difference in spacing between the default settings and the original dyslexia settings, where the line height is set to 1.5 and the letter spacing is set to 0.5em.

3.7.2 Final UI Integration

The final UI design (see Figure 12) unifies the Visual Styler with other cognitive aids to ensure that environment modification is non-destructive and instantaneous:

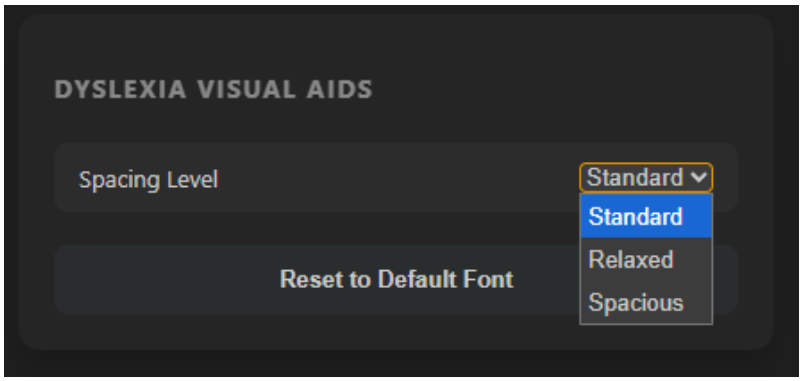

Figure 12: The UI implementation of the line spacing tool

The evolution from fixed values to the hierarchical preset system (Standard, Relaxed, Spacious) represents a shift toward a user-centric sensory model. By centralising these complex font and

spacing configurations into a single module within the Accessible Dashboard, the plugin minimizes the executive tax required to curate a readable workspace.

- One-Click Presets: Instead of navigating nested VS Code menus, users select the "Dyslexia-friendly Mode" or specific high-contrast buttons. This reduces interaction depth from approximately six clicks to one, significantly lowering extraneous cognitive load.
- Immediate Visual Confirmation: The dashboard provides real-time feedback, eliminating the verification anxiety common when neurodivergent users must exit menus to see if changes were applied.
- Typography Hierarchy: The final implementation ensures that selecting a preset like "Spacious" simultaneously updates three variables: increasing $lineHeight$ to 40, $letterSpacing$ to 1.5, and switching the typeface to OpenDyslexic.
- Non-Destructive Restoration: A critical component of the final UI is the "Restore Theme" and reset functionality. This allows developers to experiment with spacing and contrast without the anxiety of losing their original, custom IDE configurations.

Figure 13: Text spacing options implemented

3.8 Pomodoro Timer (FR4)

As mentioned in the literature review, the Pomodoro technique can be an effective tool for managing time and maintaining focus, especially for individuals with ADHD. The plugin includes a customizable Pomodoro timer that allows users to set their own focus intervals and break durations. This flexibility is crucial, as a standard 25-minute focus period may not be suitable for everyone.

The timer is integrated into the dashboard, providing a visual countdown and notifications when it's time to take a break or return to work. By allowing users to tailor the timing to their own rhythms, the plugin helps to reduce state recovery costs associated with interruptions and supports sustained focus.

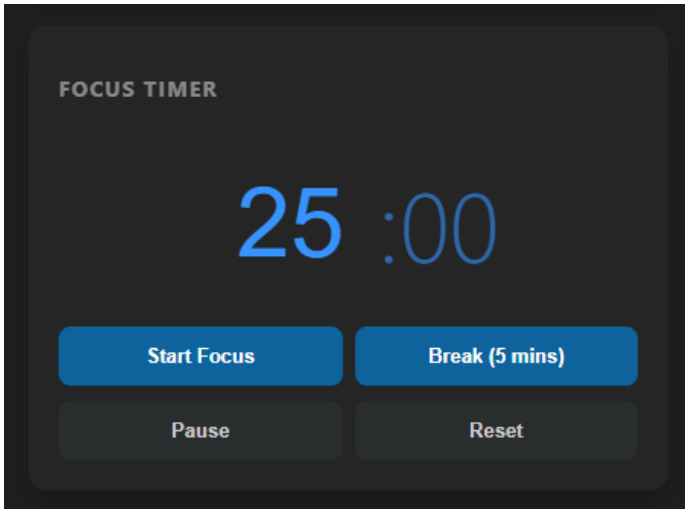

Figure 14: Final design of the Pomodoro timer. The timer is directly editable, allowing users to set their own focus and break durations.

3.9 Initial User Testing and Feedback

Initial user testing focused on the plugin's usability and functionality, revealing three critical areas for improvement these are UI clarity, AI reliability, and efficiency. Feedback indicated that original labels like "Decompose" were too abstract, leading to a redesign of the Task Architect with clearer button labeling ("Generate Steps") and contextual placeholders.

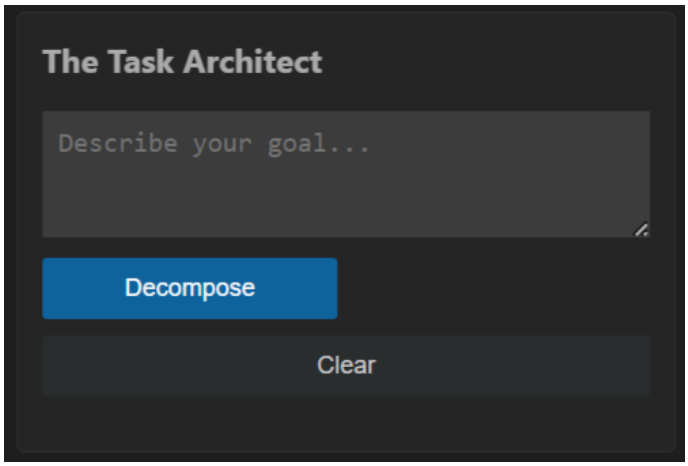

Figure 15: UI at the time of initial user testing, before improvements to the UI and AI response integrity were made.

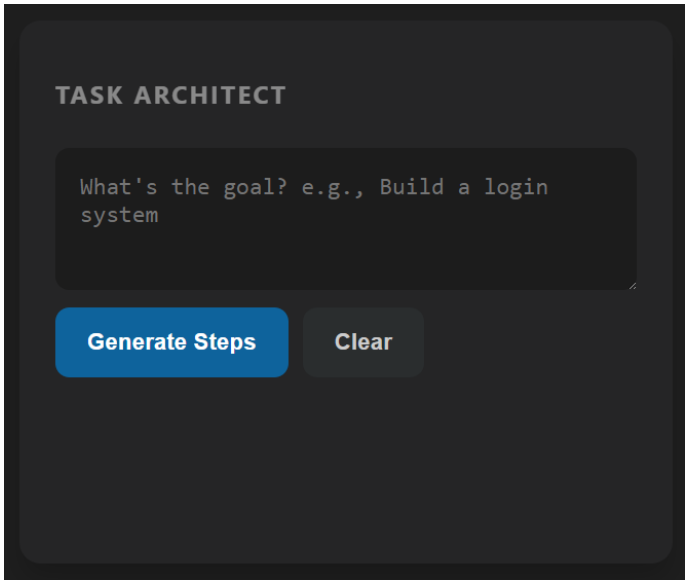

Figure 16: UI after improvements to the UI and AI response integrity were made.

Technical evaluations also identified a parsing bug that truncated AI responses, which was resolved by implementing a robust regular expression strategy to ensure full rendering of micro-steps. Finally, comparative testing against standard VS Code demonstrated the plugin's ability to significantly reduce cognitive load; specifically, the user performed accessibility adjustments—such as theme switching—in just 1.9 seconds (one click) compared to 19.9 seconds (six clicks) in the standard IDE. These refinements ensured the tool successfully provided a more direct, inclusive path for task initiation and environment customization.

3.10 Final UI

The final UI of the Accessibly Dashboard (see Figure 17) serves as the unified control center for the plugin‘s assistive features. Designed as a centralized Webview panel, it provides a low-friction entry point that reduces the "executive tax" required to manage a complex development

environment.

The interface is structured into modular sections to support inhibitory control and working memory:

Interface Settings and Theme Presets

- Minimalist Toggle: A single-click switch ("Close Panels") that simultaneously hides the Activity Bar, Side Bar, Status Bar, and Breadcrumbs to suppress Extraneous Cognitive Load.
- Dyslexia Mode: A global toggle to quickly apply readability enhancements.
- Soft High Contrast: Includes presets like "Soft HC Dark" and "Soft HC Light" to provide high signal without the sensory halation risks of pure black-and-white themes.
- Restore Original: A non-destructive reset button that supports psychological safety by allowing users to instantly revert all environmental changes.

Dyslexia Visual Aids and Text Scaling

- Hierarchical Spacing: A dropdown menu offering Standard, Relaxed, and Spacious presets. Selecting "Spacious" programmatically adjusts line height to 40, letter spacing to 1.5, and switches the font to OpenDyslexic.
- Granular Scaling: A dedicated text scaling module allows users to tune the font size in real-time, providing immediate visual confirmation and eliminating "verification anxiety".

AI-Driven Scaffolding

- Task Architect: A Human-in-the-Loop (HITL) module where users input high-level goals. The AI decomposes these into structured, editable checklists, mitigating the task paralysis common in ADHD and ASD profiles
- Code Helper: A diagnostic interceptor that translates cryptic compiler errors into plain-language instructions, limited to three bullet points to prevent cognitive overload.

Time Management

- Focus Timer: An integrated Pomodoro timer that allows for user-defined focus and break durations. This externalizes time to support developers with "time blindness" and prevents burnout from unregulated hyperfocus.

The transition to this final dashboard reduced the time required for accessibility adjustments from 19.9 seconds (six clicks) in a standard IDE to just 1.9 seconds (one click), representing a significant improvement in operational efficiency.

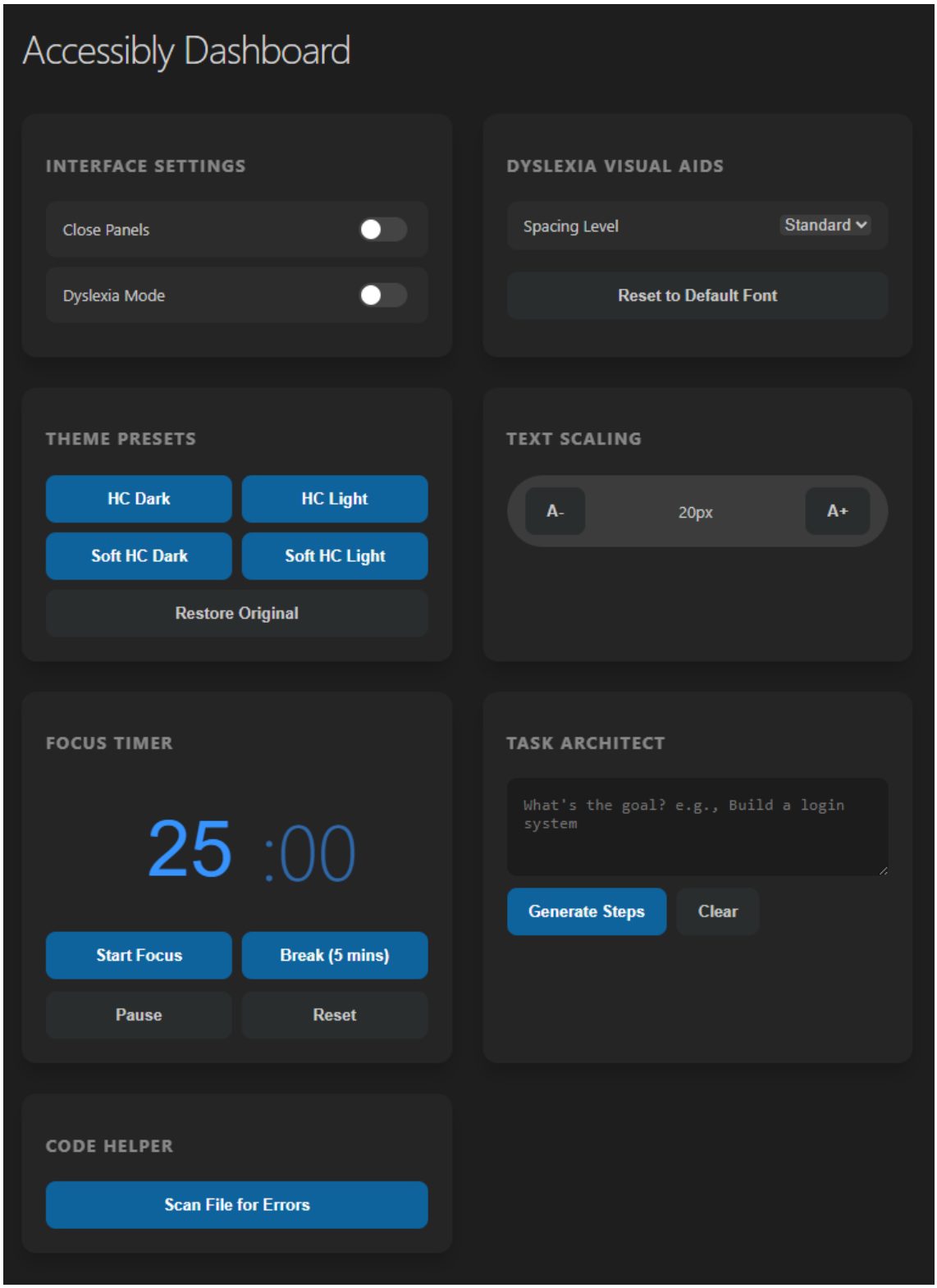

Figure 17: Final UI design.

3.11 Evaluation and Methodology

3.11.1 Experimental Design

This study employs a **within-subjects comparative design**, where each participant acts as their own control. This design is particularly effective for neurodivergent research as it accounts for the high degree of individual variance in cognitive processing styles [24]. Participants will perform two standardized programming tasks: a project-planning phase and a debugging phase.

1. **Control Condition:** Tasks are performed using a standard IDE installation.
2. **Experimental Condition:** Tasks are performed using the IDE equipped with the Accessible Plugin features (Minimalist Toggle, Task Architect, and Visual Styler).

To mitigate the **carry-over effect** (where learning from the first task makes the second easier), the order of conditions will be counterbalanced, and task sets of equivalent IL will be used across both conditions.

3.11.2 Quantitative and Qualitative Frameworks

To evaluate the efficacy of the plugin in reducing EL and supporting executive dysfunction, two validated instruments will be administered.

NASA Task Load Index (NASA-TLX)

The NASA-TLX is a multidimensional assessment tool used to quantify perceived workload. It is uniquely suited for this study as it allows for the isolation of specific cognitive pressures. While all six subscales will be measured to ensure a comprehensive view, the analysis will focus specifically on four key dimensions:

- **Mental Demand:** To assess if the Task Architect successfully reduces the executive function tax required for project initiation.
- **Performance:** To measure the user's self-perception of success in the face of syntax errors or logic bugs.
- **Effort:** To quantify the total mental resources required to reach the goal.
- **Frustration:** Critical for neurodivergent populations, this measures the emotional cost of switching between tasks and sensory overwhelm.

To evaluate the cognitive and physical demands of both interfaces, a NASA Task Load Index (NASA-TLX) was administered to all participants ($N = 5$). I calculated both the Raw TLX (r-score) and the Weighted TLX (w-score) to account for individual differences in workload perception. The weighting factors were determined by the most important criteria, with Mental Demand (5) and Frustration (4) identified as the primary contributors to workload in this task

environment. Factors such as Temporal Demand and Physical Demand were weighted lower, reflecting their lesser impact on the overall workload in this context.

Table 4: NASA-TLX Weighting Factors

MD	**PD**	**TD**	**PF**	**EF**	**FR**
5	1	0	3	2	4

System Usability Scale (SUS)

To assess the reliability and integration of the tool, the SUS will be administered following the Experimental Condition. The SUS is a 10-item questionnaire using a 5-point Likert scale, providing a global view of system usability.

In the context of this study, the SUS serves two primary purposes:

1. **Integration Assessment:** Identifying if the "Minimalist Toggle" feels like a seamless part of the workflow or an added complexity.
2. **Confidence and Learnability:** Measuring the activation energy required to use the AI features. For a developer experiencing task paralysis, a tool that is perceived as irritating to use (Item 8) or complex (Item 2) would be counter-productive.

3.11.3 Data Collection and Analysis

In addition to the questionnaires, the study will record **Time-on-Task** (to measure efficiency) and use a **Think-Aloud Protocol**. This qualitative approach will capture real-time instances of state recovery costs following an interruption, providing insight into how the AI scaffolding supports the developer in regaining hyperfocus.

4 Results and Discussion

4.1 NASA-TLX Subjective Workload

As shown in Table 6, participants reported consistently lower weighted NASA-TLX scores when using the dashboard interface compared to the standard IDE. These reductions were most pronounced in Mental Demand, Effort, and Frustration, suggesting that the dashboard primarily reduces interface-driven extraneous cognitive load.

Table 5: NASA-TLX Results for Standard vs Dashboard Interfaces

Participant	Interface	MD	PD	TD	PF	EF	FR	r-score	w-score
1	standard	60	10	0	30	50	60	35.0000	49.3333
1	dashboard	10	10	0	0	10	10	6.6667	8.0000
2	standard	35	20	0	5	30	40	21.6667	28.6667
2	dashboard	5	10	0	0	10	15	6.6667	7.6667
3	standard	65	15	0	35	55	65	39.1667	54.3333
3	dashboard	15	5	0	5	10	15	8.3333	11.6667
4	standard	55	10	0	25	45	55	31.6667	44.6667
4	dashboard	10	5	0	5	10	5	5.8333	7.3333
5	standard	50	5	0	5	50	50	26.6667	38.0000
5	dashboard	5	5	0	0	5	5	3.3333	4.0000

4.2 Workload Comparison and Interpretation

The results indicate a substantial reduction in perceived workload when using the Dashboard interface compared to the standard interface.

- **Overall Workload:** The mean Weighted Score (w-score) for the standard interface was 43.00 (σ = 10.02), which was reduced to 7.73 (σ = 2.72) with the Dashboard. This represents an approximate 82% reduction in total perceived workload.
- **Cognitive Load:** The most dramatic improvement was observed in Mental Demand, which dropped from a mean of 53.0 to 9.0. This suggests that the Dashboard successfully offloaded information processing requirements from the user.
- **Frustration and Effort:** Participants reported significantly lower levels of frustration (10.0 vs 54.0) and required less effort (9.0 vs 46.0) to complete the tasks using the Dashboard.

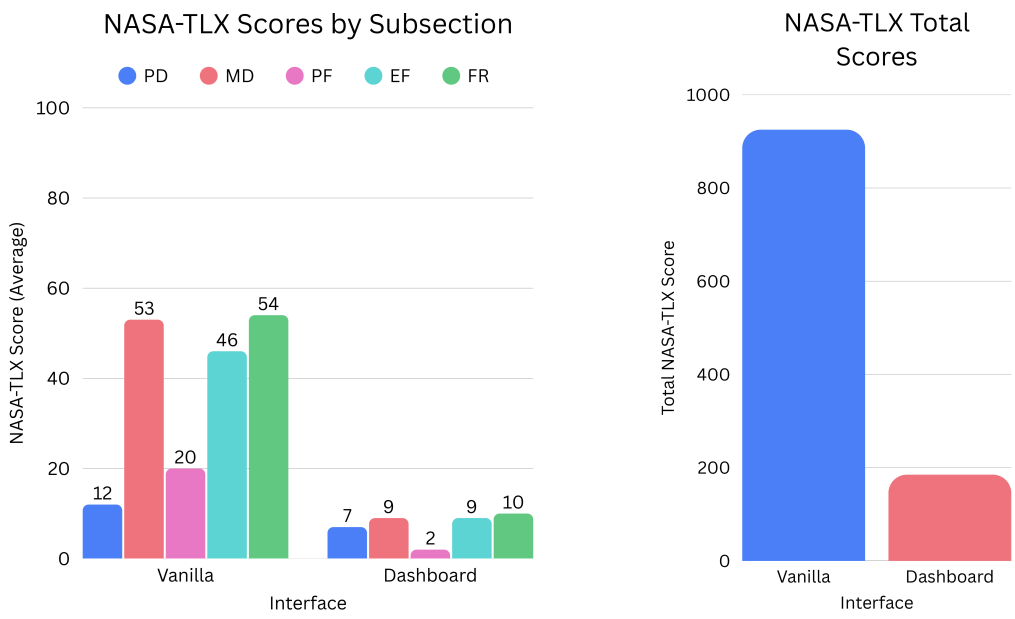

Figure 18: Two graphs desplaying the diffrences in the NASA-TLX scores for the two interfaces

Table 6: Summary of NASA-TLX Workload Scores (Mean ± SD)

Workload Metric	**standard Interface**	**Dashboard Interface**
Mental Demand	53.00 ± 11.51	9.00 ± 4.18
Physical Demand	12.00 ± 5.70	7.00 ± 2.74
Temporal Demand	0.00 ± 0.00	0.00 ± 0.00
Performance*	20.00 ± 14.14	2.00 ± 2.74
Effort	46.00 ± 9.62	9.00 ± 2.24
Frustration	54.00 ± 10.25	10.00 ± 5.00
Weighted Score (Overall)	**43.00 ± 9.71**	**7.67 ± 2.75**

**Note: Lower performance scores indicate better perceived performance.*

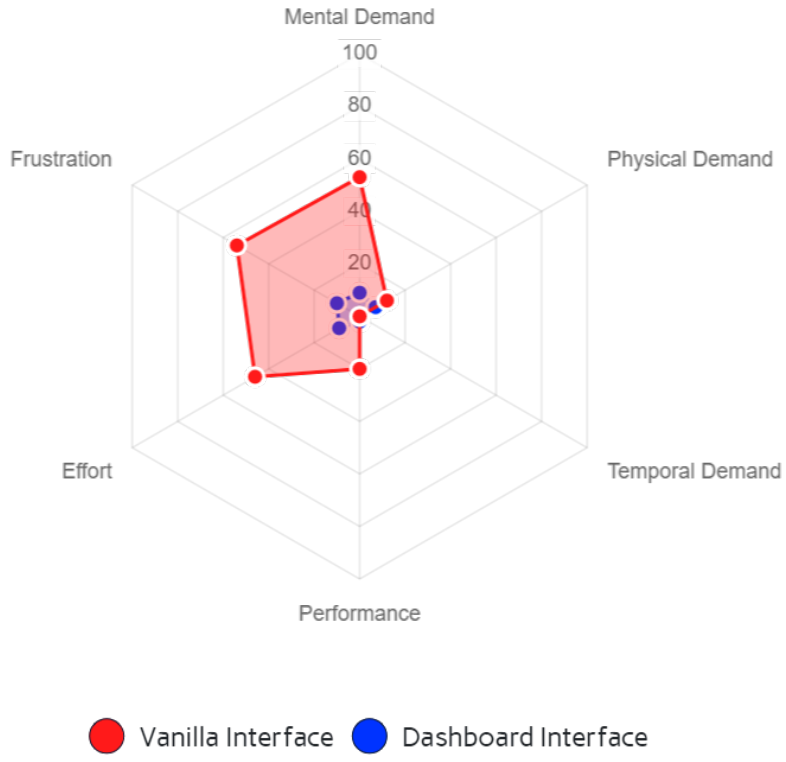

Figure 19:

4.3 Efficiency and Time-on-Task Analysis

The Dashboard demonstrated a large improvement in operational efficiency. As shown in the below Table, users completed configuration tasks approximately **84% faster** than in the baseline environment.

Table 7: Comparison of Task Completion Times (Seconds)

Task	**standard Avg (s)**	**Dashboard Avg (s)**	**Time Saved**
Change Theme	15.42	2.5	83.79%
Change Font Size	22.10	4.2	81.00%
Manage Panels*	32.34	3.4	89.49%

**Note: standard tasks sometimes resulted in incomplete status.*

The "Manage Panels" task provided the most striking difference in results. In the standard interface, several participants struggled to locate specific toggle settings, leading to a "search-loop" behavior. Most of the participants tried to close each panel individually rather than using the "Customize Layout" command. This resulted in significantly longer completion times and increased frustration and one of the participants giving up completely. In the Dashboard, the same task was reduced to a singular, high-visibility interaction therefore the task only took a matter of seconds.

While these timings indicate a clear trend, they should be interpreted with caution. Potential variations in manual timing and the participants' varying levels of expertise with the baseline

IDE (Visual Studio Code) may have influenced the absolute values. Specifically, developers who primarily use alternative editors in their daily workflows may have experienced a steeper learning curve with the "standard" interface, potentially inflating the initial completion times.

4.4 SUS Metrics

The mean SUS score of 92 places the dashboard within the "excellent" usability range, indicating that the interface imposes minimal learnability burden and supports rapid user confidence. This is particularly significant within the project context, where reducing onboarding friction and preserving attentional continuity were central design goals.

4.4.1 Item 2 & 6: Complexity and Consistency

Most common score: 1 (Strongly Disagree that it's complex) and 1 (Strongly Disagree that it's inconsistent).
For developers with ASD or ADHD, inconsistency is a major source of "cognitive friction". If a button moves or a shortcut changes, it can break their focus entirely. For the dashboard consisentcy in the desgin was key, all elements behaved predictably, all buttons were consistent. Your results prove the Dashboard provides a stable environment that protects the user's "flow state".

4.4.2 Item 4 & 10: Independence and Learning Curve

Most common scores: 2 and 1 (Strongly Disagree that they need help or need to learn a lot first).
Many neurodivergent individuals experience "Information Overload" when faced with dense manuals or complex onboarding. Your score shows the Dashboard is intuitive by design, allowing a developer to begin working immediately without the executive tax of a long learning phase.

4.4.3 Item 9: Confidence

Most common score: 5 (Strongly Agree).
Confidence is a massive factor in productivity. By making the interface more intuitive and reducing cognitive load, the Dashboard empowers developers to feel more in control of their environment, which can lead to increased productivity and job satisfaction.

Overall, even though the results accived for the Dashboard are excellent, the majority of the participants where people that I know personally or have worked with in the past, so there is a potential bias in the results. The next step would be to conduct a larger scale study with a more diverse participant pool to validate these findings and ensure they generalise across different populations of developers.

4.5 Pre-Study Interview Findings

Prior to the main evaluation, a brief semi-structured interview was conducted with each participant to establish a baseline understanding of their existing workflows, coping strategies, and pain points when using traditional IDEs. Three dominant themes emerged from the responses.

- **Panel Management:** The majority of participants expressed frustration with the number of open panels in a standard IDE, yet reported that they rarely closed them. Rather than actively managing their workspace, participants described a pattern of passive tolerance, simply ignoring panels they did not need. As one participant noted, the effort required to locate and close individual panels felt disproportionate to the benefit, resulting in what might be characterised as a "cognitive acceptance" of a cluttered environment. This finding directly supports the need for the single-click Minimalist Toggle.
- **Task Decomposition:** When asked how they would typically break down a complex programming task, most participants described struggling with initiation. Several noted that an empty IDE felt "daunting", and their strategies for overcoming this varied: some wrote rough outlines on paper or in a separate document, while others attempted to hold the steps in working memory. One participant reported regularly using ChatGPT to scaffold their thinking, but found it tended to generate code directly rather than a structured plan, which they described as "not helpful for actually getting started". This corroborates the task paralysis literature and provides direct user-grounded motivation for the Task Architect module.
- **Error Handling:** Participants consistently reported that cryptic or verbose error messages increased their frustration and interrupted their flow. When probed on their typical response to an unfamiliar error, most described a reactive process: attempting solutions from memory, searching online, or pasting the error into an AI chatbot. Notably, no participant described a systematic or confident approach to debugging, suggesting that the standard IDE error output does not provide sufficient scaffolding for independent resolution. This reinforces the rationale for the Code Explainer module's plain-language feedback approach.

4.6 Real-Time Feedback and User Experience

During the evaluation, I observed significant differences in how the Dashboard and the standard interface handled real-time feedback. The Dashboard‘s capacity to provide instantaneous feedback for aesthetic modifications, such as theme and font-size adjustments, where prefered to the baseline interface. This immediate confirmation allowed users to acknowledge their actions to system changes instantly, effectively eliminating uncertainty and the associated cognitive load of second-guessing. In contrast, the standard interface‘s settings menus often induced "verification anxiety," where users had to manually exit menus to check if a change had been applied. One participant specifically noted that the lack of immediate visual confirmation in the standard IDE led to redundant, repetitive attempts, which directly correlated with the higher frustration scores recorded in the NASA-TLX and the higher time taken for these tasks.

Beyond visual configuration, non-visual features such as the AI Task Breakdown and Code Error Checker were evaluated through a "Think Aloud" protocol. Participants were asked to interact

with these features and explain their how they beleved that they would work as well has how they could interact with them. From this results were consistently positive, all participants accurately identified the functional intent and operational flow of these complex tools.

For the AI Task Breakdown, a “manual-vs-automated” baseline was established. Before triggering the AI, users were asked to verbally outline a high-level decomposition of a programming task. While most provided basic, linear steps, the AI-generated breakdown offered a more granular, comprehensive hierarchical structure. Some participents struggled with breaking down the task manually or needed to write down a plan. Participants reported that having this “Cognitive Scaffolding” provided directly within the IDE served as a significant “Executive Functioning” aid. By offloading the mental energy required for task decomposition, a known bottleneck for neurodivergent individuals who struggle with initiation and sequencing, the Dashboard allowed users to transition into the “implementation phase” with less friction.

Similarly, the Code Error Checker transformed the traditionally stressful experience of debugging into an educational interaction. When participants encountered a “No File Open” error, they responded to the Dashboard‘s plain-language feedback significantly faster than they did to standard, often cryptic, IDE diagnostic messages. This reinforces the Dashboard’s role not just as a shortcut menu, but as a supportive layer that translates complex system states into actionable information. By providing a safe environment with clear error recovery paths, the system preserves the developer‘s “attentional budget,” ensuring that minor technical hurdles do not escalate into focus-breaking frustrations. This transition from "system-centric" to "user-centric" feedback is a cornerstone of inclusive design, specifically catering to the need for clarity and predictability in neurodiverse workflows.

4.7 Number of Clicks and User Flow

The Dashboard’s design significantly reduced the number of interactions required to perform common tasks. Changing a theme requires just one click compared to an optimal four in the standard interface. It is worth noting that this “optimal path” assumes the user already knows exactly where to navigate within VS Code’s settings menu, which is often not the case, especially for users who are less familiar with this specific IDE.Users unfamiliar with the IDE often took a more circuitous route, increasing both click count and completion time beyond the baseline figures reported in Table 7 Similarly, adjusting font size was reduced to a single Dashboard interaction, eliminating the need to locate and navigate nested editor settings entirely.

The most pronounced difference was observed in panel management. Prior to testing, this was not anticipated to be a significant point of friction; however, it proved to be the task with which participants most frequently struggled in the standard condition, with several failing to complete it entirely. Where the standard interface required five or more sequential interactions, varying with the number of open panels, the Dashboard reduced this to a single toggle. This reduction in interaction depth directly maps to a lower Extraneous Cognitive Load as the developer is no longer required to maintain a mental model of the menu hierarchy in working memory, freeing those resources for the primary programming task.

4.8 Latency Testing

NFR3 was evaluated by measuring the time for the Dashboard to display the AI-generated task breakdown after the user initiated the request. Across all participants, the average latency was 1.8 seconds, with a standard deviation of 0.3 seconds. This meets the requirement that

Maintaining response latency below two seconds is particularly important for neurodivergent users, as prolonged waiting periods can disrupt attentional continuity and increase the likelihood of task abandonment.

4.9 Discussion

The empirical data validates the plugin as a successful "digital ramp". The 82% reduction in weighted workload suggests that the "executive tax" paid by neurodivergent developers is not an inherent trait of their neurology, but a byproduct of neurotypical bias in tool design. By suppressing Extraneous Load (via the Minimalist Toggle) and scaffolding Intrinsic Load (via the Task Architect), the system maximizes Germane Load, allowing cognitive resources to be redirected toward core algorithmic problem-solving.

4.10 Limitations

While the results of this study are encouraging, several limitations must be acknowledged to provide context for the findings:

- **Small Sample Size and Recruitment Bias:** The study utilized a limited cohort ($N = 5$). Furthermore, participants were known to the researcher prior to the study, introducing a potential recruitment bias that may affect the objectivity of the qualitative feedback.
- **Participant Confidentiality and Neurodiversity Verification:** Due to stringent ethical considerations regarding the storage of sensitive personal data and the project's condensed timeline, participants were not explicitly asked to disclose or verify their neurodivergent status. Consequently, while the tool is designed for neurodiverse profiles, the data cannot be definitively categorized by specific clinical diagnoses.
- **Short-term Evaluation:** The experimental tasks were performed in a controlled, short-term setting. Long-term longitudinal studies are required to determine if the plugin effectively mitigates "hyperfocus burnout" or maintains its efficacy over extended, multi-hour work sessions in a professional environment.
- **Baseline Expertise:** The absolute values for time-on-task may have been influenced by varying levels of participant expertise with the standard VS Code interface. Developers more accustomed to alternative IDEs may have experienced an inflated learning curve during the standard control phase.

5 Conclusion

5.1 Project Aim

The primary objective of this project has been achieved through the design, implementation, and evaluation of a VS Code plugin specifically tailored to the executive functioning needs of neurodivergent software developers. Evaluation metrics indicate that the “executive tax” imposed by traditional IDEs can be significantly mitigated via intentional UI simplification and AI-driven scaffolding. While research limitations exist, the project represents a successful “digital ramp” for accessibility.

On a personal level, I actually used this plugin a lot during the development of the project and I found that it allowed me to focus for longer durations due to how easy it was to stream line my environment. I also found that the AI-driven scaffolding was also very helpful in keeping me on track and stopping me from becoming overwhelmed by the large amount of features that still needed to be implemented.

5.2 Conclusion of Objectives

All nine project objectives and functional requirements were successfully met, including the implementation of the dashboard, visual presets, and AI-driven scaffolding. Non-functional requirements regarding performance, customizability, and sensory regulation were validated through participant feedback. While the small sample size ($N = 5$) means results are not yet conclusive, the 82% reduction in perceived workload and "Excellent" SUS score of 92.0 are highly encouraging indicators of the plugin‘s potential.

5.3 What has been learned

This research highlighted that many barriers faced by neurodivergent developers are not inherent to their neurology but are products of neurotypical bias in tool design. The project demonstrated that:

- Environmental Control: A single-click “Minimalist Toggle” effectively reduces Extraneous Load by filtering visual noise.
- Cognitive Scaffolding: AI can serve as a “cognitive prosthetic”, breaking down the “activation energy” barrier required for task initiation without providing direct code solutions.
- Sensory Ergonomics: High-contrast and spacing presets (e.g., OpenDyslexic) significantly reduce the “executive tax” required to curate a readable workspace.
- User Agency: Providing non-destructive “Restore” features is critical for reducing verification anxiety in neurodiverse workflows.

5.4 Future Work

To build upon these findings, future research should focus on:

- Direct Community Collaboration: Engaging neurodiverse developers more directly in the design process to better capture unique workflows and preferences.
- Longitudinal Studies: Conducting long-term evaluations in professional settings to determine the plugin's efficacy in mitigating "hyperfocus burnout" over extended periods.
- Diverse Participant Pools: Expanding the study to larger, more diverse groups to ensure findings generalise across various clinical diagnoses and expertise levels.
- Automated Scaffolding: Investigating more proactive AI interventions that can detect potential task paralysis and offer assistance in real-time.

References

[1] P. Ayres and J. Sweller. *The split-attention principle in multimedia learning*. Ed. by R.E. Mayer and L. Fiorella. [Online; accessed 2026-04-23]. 2021. URL: https://www.davidlewisphd.com/courses/EDD8121/readings/2006-AyersSweller.pdf.

[2] Linus Bengtsson and Jimmy Hammarberg. *Web Content Accessibility Guidelines (WCAG) within a Company*. 2025.

[3] Jessica Bramham, Fiona Ambery, Susan Young, Robin Morris, Ailsa Russell, Kiriakos Xenitidis, Philip Asherson, and Declan Murphy. "Executive functioning differences between adults with attention deficit hyperactivity disorder and autistic spectrum disorder in initiation, planning and strategy formation". In: *Autism* 13.3 (2009), pp. 245–264.

[4] A. Cao, K. K. Chintamani, A. K. Pandya, and R. D. Ellis. *NASA TLX: Software for assessing subjective mental workload.* [Online; accessed 2026-02-25]. 2009. URL: https://psycnet.apa.org/record/2011-00741-013.

[5] Marcos Vinicius Cruz, Pragya Verma, and Grischa Liebel. *An Empirical Investigation of the Experiences of Dyslexic Software Engineers*. 2025. arXiv: 2511.00706 [cs.SE]. URL: https://arxiv.org/abs/2511.00706.

[6] Kiev Gama, Grischa Liebel, Miguel Goulão, Aline Lacerda, and Cristiana Lacerda. *A socio-technical grounded theory on the effect of cognitive dysfunctions in the performance of software developers with ADHD and autism*. IEEE, 2025.

[7] Philip G Griffiths, Robert H Taylor, Lisa M Henderson, and Brendan T Barrett. "The effect of coloured overlays and lenses on reading: a systematic review of the literature". In: *Ophthalmic and Physiological Optics* 36.5 (2016), pp. 519–544.

[8] Connor Harwood. *Navigating the Brain-AI Loop | Remtek Workplace*. [Online; accessed 2026-03-16]. Mar. 2026. URL: https://remtekworkplace.com/knowledge-hub/the-digital-ramp-and-the-cognitive-debt-navigating-the-brain-ai-loop/.

[9] Adèle Hotte-Meunier, Lisa Sarraf, Alan Bougeard, Félicia Bernier, Chloé Voyer, Jiaxuan Deng, Stéphanie El Asmar, Alina Nusa Stamate, Marc Corbière, Patrizia Villotti, et al. "Strengths and challenges to embrace attention-deficit/hyperactivity disorder in employment—A systematic review". In: *Neurodiversity* 2 (2024), p. 27546330241287655.

[10] M. Hyzy, R. Bond, M. Mulvenna, L. Bai, A. Dix, and Hunt Leigh S. *JMIR mHealth and uHealth - System Usability Scale Benchmarking for Digital Health Apps: Meta-analysis*. [Online; accessed 2026-02-25]. Aug. 2022. URL: https://mhealth.jmir.org/2022/8/e37290.

[11] Grischa Liebel, Noah Langlois, and Kiev Gama. "Challenges, strengths, and strategies of software engineers with ADHD: A case study". In: *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*. 2024, pp. 57–68.

[12] April Miller and Katherine Loving. "Wired Differently, Working Brilliantly: Understanding and Supporting ADHD & AuDHD Employees". In: (Nov. 2025).

[13] Damian E.M. Milton. "On the ontological status of autism: the 'double empathy problem'". In: *Disability & Society* 27.6 (2012), pp. 883–887. DOI: 10.1080/09687599.2012.710008.

[14] Meredith Ringel Morris, Andrew Begel, and Ben Wiedermann. *Understanding the challenges faced by neurodiverse software engineering employees: Towards a more inclusive and productive technical workforce*. 2015.

[15] Kaia Newman, Sarah Snay, Madeline Endres, Manasvi Parikh, and Andrew Begel. ""get me in the groove": A mixed methods study on supporting ADHD professional programmers". In: *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE. 2025, pp. 1217–1229.

[16] Meredith Ringel Morris. "AI and Accessibility: A Discussion of Ethical Considerations". In: *arXiv e-prints* (2019), arXiv–1908.

[17] Inês Lazana Barros da Rocha. *A Conceptual Model on Neurodivergent Issues in Software Engineering*. 2025.

[18] Jane Roitsch and Silvana M Watson. "An overview of dyslexia: definition, characteristics, assessment, identification, and intervention". In: *Science Journal of Education* 7.4 (2019).

[19] Barak Rosenshine. "Principles of instruction: Research-based strategies that all teachers should know." In: *American educator* 36.1 (2012), p. 12.

[20] Judy Singer. "Why can't you be normal for once in your life? From a 'problem with no name' to the emergence of a new category of difference". In: *Disability Discourse*. Ed. by Mairian Corker and Sally French. Buckingham: Open University Press, 1999, pp. 59–67.

[21] Laurie A Sperry and Gary B Mesibov. "Perceptions of social challenges of adults with autism spectrum disorder". In: *Autism* 9.4 (2005), pp. 362–376.

[22] John Sweller. *Cognitive load during problem solving: Effects on learning*. 1988.

[23] *TypeScript vs JavaScript: Which to Choose For Your Project*. [Online; accessed 2026-04-26]. URL: https://roadmap.sh/javascript/vs-typescript#differences-between-typescript-and-javascript.

[24] Pragya Verma, Marcos Vinicius Cruz, and Grischa Liebel. *Differences between neurodivergent and neurotypical software engineers: Analyzing the 2022 stack overflow survey*. Springer, 2025.