UNIVERSITY OF PAVIA

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL, COMPUTER AND BIOMEDICAL ENGINEERING

MASTER'S DEGREE IN COMPUTER ENGINEERING

MASTER THESIS

Design and Development of an Eye-Tracking-Enhanced Writing Assistant Leveraging Large Language Models

Progettazione e sviluppo di un assistente alla scrittura basato su eye tracking e supportato da modelli linguistici di grandi dimensioni

Candidate: Alireza Khornegah

Supervisor: Prof. Marco Porta

A.A. 2024/25

Abstract

In recent years, writing scientific and academic texts in English has been a challenge for many people, especially non-native English speakers. Problems such as grammatical errors, incorrect choice of words, and lack of appropriate writing style are among the issues that can affect the quality of text. With the development of Large Language Models (LLMs), it has become possible to automatically identify and correct language errors and provide suggestions for improving writing. On the other hand, eye-tracking technology, as one of the new methods of human-computer interaction, allows for analyzing user behavior while reading and writing text. In this thesis, a writing assistant system based on eye tracking and large language models is presented, which aims to help improve the quality of users' writing in English or other languages. In this system, using eye-tracking data, the sentence on which the user's gaze is focused is identified and sent to a language model to be checked grammatically. If there are errors, the system identifies the incorrect part of the sentence and suggests corrections. In addition to grammatical corrections, the system is able to produce a simpler version as well as a more formal version of the sentence to help improve the writing style. The proposed system includes components for receiving eye-tracking data, recognizing the sentence that the user is focused on, linguistic analysis using a large language model, and an interactive user interface for displaying suggestions. The proposed system demonstrates that combining eye-tracking technology with language models can provide an innovative approach to assisted writing systems and help users improve the quality of writing, especially in a foreign language.

Sommario

Negli ultimi anni, la scrittura di testi scientifici e accademici in inglese si è rivelata una sfida per molte persone, soprattutto per i non madrelingua. Problemi come errori grammaticali, scelta inappropriata delle parole e mancanza di uno stile di scrittura adeguato sono tra le problematiche che possono compromettere la qualità del testo. Con lo sviluppo dei Large Language Models (LLM), è diventato possibile identificare e correggere automaticamente gli errori linguistici e fornire suggerimenti per migliorare la scrittura. D'altra parte, la tecnologia dell'eye tracking, come uno dei nuovi metodi di interazione uomo-computer, consente di analizzare il comportamento dell'utente durante la lettura e la scrittura di testi. In questa tesi, viene presentato un sistema di assistenza alla scrittura basato su eye-tracking e LLM che mira a migliorare la qualità della scrittura. In questo sistema, utilizzando i dati di eye tracking, la frase su cui si concentra lo sguardo dell'utente viene identificata e inviata a un modello linguistico per un controllo grammaticale. In caso di errori, il sistema identifica la parte errata della frase e suggerisce le correzioni. Oltre alle correzioni grammaticali, il sistema è in grado di produrre sia una versione semplificata che una più formale della frase, contribuendo così a migliorare lo stile di scrittura. Il sistema proposto include componenti per la ricezione dei dati di eye tracking, il riconoscimento della frase su cui l'utente si concentra, l'analisi linguistica tramite LLM e un'interfaccia utente interattiva per la visualizzazione dei suggerimenti. Il sistema proposto dimostra che la combinazione della tecnologia di eye-tracking con i modelli linguistici può fornire un approccio innovativo ai sistemi di scrittura assistita e aiutare gli utenti a migliorare la qualità della scrittura, soprattutto in una lingua straniera.

Contents

CHAPTER 1 – INTRODUCTION 6

1.1 Background 6
1.2 Problem Statement 7
1.3 Research Objectives 7
1.4 Contributions 8
1.5 Thesis Structure 9

CHAPTER 2 – EYE TRACKING 10

2.1 Introduction to Eye Tracking 10
2.2 Eye Tracking Technologies 11
2.2.1 Infrared 12
2.2.2 Remote Eye Trackers 12
2.2.3 Head-Mounted Eye Trackers 12
2.2.4 Camera-Based Gaze Estimation 13
2.3 Eye Movements 14
2.3.1 Fixation 14
2.3.2 Saccade 15
2.3.3 Smooth Pursuit 16
2.4 Gaze-Based Interaction 18
2.4.1 Gaze as an Input Modality 18
2.4.2 Dwell-Time Interaction 20
2.4.3 Gaze in Human-Computer Interaction 21
2.5 Challenges of Eye Tracking 23
2.5.1 Noise and Jitter 23
2.5.2 Calibration Issues 24
2.5.3 Gaze Instability 26
2.5.4 Accuracy and Precision Limitations 27
2.6 Eye Tracking in Writing and Assistive Interfaces 29
2.7 Chapter Summary 31

CHAPTER 3 – LARGE LANGUAGE MODELS 33

3.1 Introduction to LLMs 33
3.2 Transformer Architecture 35
3.3 LLMs in Writing Assistance 37
3.4 Prompt-Based Interaction with LLMs 39
3.5 Limitations of LLMs 41
3.6 Chapter Summary 43

CHAPTER 4 – RELATED WORK 45

4.1 AI Writing Assistants 45
4.2 Eye-Tracking Interaction Systems 47
4.3 Eye Tracking and AI-Based Systems 49
4.4 Research Gap 50
4.5 Chapter Summary 53

CHAPTER 5 – PROPOSED SYSTEM 54

5.1 System Overview 54
5.2 Design Goals 55
5.3 General Architecture of the Proposed System 58
5.4 Main Components of the System 61

5.4.1 User Interface Module 62
5.4.2 Eye Tracking Module 63
5.4.3 Gaze Interpretation Module 64
5.4.4 Language Model Module 64
5.4.5 Logging and Interaction Recording Module 64
5.4.6 Coordination Between Components 65
5.5 Interaction Workflow 65
5.6 Operating Modes 68
5.6.1 Manual Mode 69
5.6.2 Gaze-Triggered Mode 69
5.6.3 Mouse-as-Gaze Debug Mode 70
5.6.4 Role of Multiple Modes in the Proposed System 70
5.7 Data Flow in the System 71
5.8 Design Rationale 74
5.9 Chapter Summary 76

CHAPTER 6 – SYSTEM IMPLEMENTATION 78

6.1 Implementation Overview 78
6.2 Technology Stack 80
6.3 Project Structure 82
6.4 User Interface Implementation 85
6.5 Gaze Data Processing 88
6.6 Sentence Detection and Fixation Logic 90
6.7 Language Model Integration 93
6.8 Issue Detection and Highlighting 95
6.9 Background Workers and Asynchronous Execution 97
6.10 Logging and Session Recording 99
6.11 Error Handling and Robustness 102
6.12 Chapter Summary 104

CHAPTER 7 – EVALUATION 107

7.1 Evaluation Strategy 107
7.2 System Behavior 109
7.3 Correction, Style, and Simplification Examples 111
7.3.1 Grammar Correction Examples 111
7.3.2 Style and Rewriting Examples 114
7.3.3 Sentence Simplification Examples 116
7.3.4 Overall Analysis of System Output 118
7.4 Interaction Logs Analysis 119
7.4.1 Structure of Logged Data 119
7.4.2 Temporal Interaction Flow 120
7.4.3 Fixation-Triggered Events 121
7.4.4 Role of Logging in Evaluation 121
7.5 Discussion 122

CHAPTER 8 – CONCLUSION 124

8.1 Summary of the Work 124
8.2 Contributions 125
8.2.1 Conceptual Contribution 125
8.2.2 System Design Contribution 126
8.2.3 Implementation Contribution 126
8.2.4 Functional Contribution 127
8.2.5 Empirical Contribution 127

8.2.6 Summary of Contributions 127
8.3 Limitations 128
8.4 Future Work 129
BIBLIOGRAPHY 130

Chapter 1 – Introduction

1.1 Background

In recent years, writing scientific and academic texts in English has become a significant challenge for many students and researchers, especially those whose native language is not English. Grammatical errors, incorrect word choice, and inappropriate sentence structure are among the problems that can reduce the quality of scientific texts and make them more difficult for readers to understand. For this reason, the development of intelligent tools to assist the writing process has attracted much attention in the field of natural language processing in recent years [1].

With the advancement of machine learning methods, and especially in the field of natural language processing, more advanced systems have been developed for analyzing and improving text. In the meantime, Large Language Models have shown significant progress in understanding and producing natural language and are used in applications such as text generation, grammatical correction, and sentence rewriting [2].

These models, which are usually developed based on the Transformer architecture, are able to learn complex language patterns from large amounts of text data and are used in many writing-related applications [3].

On the other hand, in the field of human-computer interaction, eye-tracking technology has been considered as one of the new methods for analyzing user behavior. This technology allows recording and analyzing the user's eye movements and can provide valuable information about how the user pays attention and focuses on different parts of the text or user interface. Eye tracking has been used in various fields such as reading studies, user interface evaluation, and interactive systems [4].

The integration of eye-tracking technology with language model-based systems can provide a new approach to the design of writing assistance tools. Using user gaze data, it is possible to identify parts of the text that have attracted the user's attention and analyze and modify them intelligently. Such an approach can lead to the development of advanced interactive systems that make the process of writing and editing text simpler and more efficient for users.

1.2 Problem Statement

Despite the significant development of assistive writing tools and text correction systems in recent years, many of these systems still operate in a traditional, direct user interaction-based manner. In most of these tools, the user must actively request a text review or select a specific section of text for correction. As a result, these systems are usually unable to intelligently recognize which part of the text the user is focused on or which part of the text is most challenging for them when writing or editing text.

On the other hand, when reading or writing text, users usually focus their visual attention on parts of the text that need to be reviewed, corrected, or better understood. Eye-tracking technology can provide valuable information about how the user's attention is distributed in the text and indicate where the user pauses or focuses most in the text. However, the use of eye-tracking data in the design of assistive writing systems has not yet been widely explored.

Therefore, one of the important challenges in designing intelligent writing assistance tools is to develop systems that can consider the user's behavior when interacting with the text and provide appropriate correction suggestions based on that. Combining eye-tracking technology with large language models can enable the design of new interactive systems that intelligently identify the sentence that the user is focusing on and analyze it in terms of grammar and style.

1.3 Research Objectives

The main objective of this research is to design and implement an interactive writing assistance system that combines eye-tracking technology and large language models in order to support users during text writing and revision. The proposed approach is based on the idea that the user's gaze behavior while reading or writing can provide meaningful information about which parts of the text require attention, revision, or correction.

More specifically, the objectives of this research are as follows:

- To design an interactive framework capable of identifying the sentence or text segment that attracts the user's attention by using eye-tracking data.
- To investigate whether gaze behavior can be used as a meaningful input signal in intelligent writing assistance systems.

- To explore the use of large language models for grammatical analysis and correction suggestion generation in a gaze-aware writing environment.
- To examine the feasibility of improving writing style by generating alternative sentence versions, including simpler and more formal forms.
- To provide a user interface that presents correction and style suggestions in a clear and understandable way.
- To support the analysis of user behavior and system operation through the recording of user-system interactions.

1.4 Contributions

This research makes several contributions to the integration of eye-tracking technology and large language models for writing assistance. Its main contribution is the development of an interactive prototype in which the user's gaze data is used to guide sentence-level linguistic analysis and activate writing support functions. In this way, the system demonstrates that visual attention can be exploited as a meaningful signal for intelligent writing assistance.

The main contributions of this thesis can be summarized as follows:

- A gaze-triggered interactive writing assistant that integrates eye tracking and large language models for writing support.
- A sentence-level attention detection mechanism based on gaze location and fixation information.
- A grammar correction pipeline that analyzes the sentence under attention and generates correction suggestions.
- A style improvement module that produces alternative sentence versions, including simpler and more academic rewritings.
- An interactive user interface for presenting detected issues, correction suggestions, and rewriting options.
- A logging mechanism for recording gaze data and user-system interactions in order to support later analysis and evaluation

1.5 Thesis Structure

This thesis is organized into eight chapters.

In Chapter 1, an introduction to the research topic is provided, along with the research problem, research objectives, and the main contributions of this research.

In Chapter 2, the theoretical foundations of eye-tracking technology are reviewed. In this chapter, the basic concepts of eye tracking, the types of existing eye-tracking technologies and their applications in human-computer interaction are discussed.

In Chapter 3, large language models are introduced. The transformer architecture and the role of large language models in applications such as text generation, grammar correction, and writing assistance systems are also reviewed.

In Chapter 4, a literature review on the research topic is presented. This chapter considers eye-tracking-based systems, intelligent writing assistance tools, and studies exploiting a combination of artificial intelligence and eye tracking.

In Chapter 5, the proposed system is introduced at a functional level. Its overall architecture and its main components, including the eye-tracking module, sentence-detection module, grammar-correction module, and writing-style module, are explained.

In Chapter 6, the details of the system implementation are presented, including the user interface design, the processing of gaze data, and the algorithms used to recognize the user's sentence of interest.

In Chapter 7, the system performance is examined through representative scenarios, sentence correction and rewriting examples, and qualitative analysis of system behavior. The strengths, limitations, and challenges of the current prototype are also discussed.

Finally, Chapter 8 draws some conclusions and provides suggestions for future research.

Chapter 2 – Eye Tracking

2.1 Introduction to Eye Tracking

Eye tracking is a set of methods and technologies used to record eye movements and estimate the user's gaze point. The main goal of this technology is to determine which area of the visual environment the user is paying attention to at any given moment and how their gaze pattern changes over time. For this reason, eye tracking has become an important tool in recent years for studying visual attention, cognitive processing, user behavior, and human-computer interaction [4], [5].

The importance of this technology stems from the fact that eye movements are closely related to human cognitive processes. In many situations, pausing gaze on one area can be a sign of attention, active processing, or difficulty in understanding that area, while rapid gaze shifting between different points may indicate scanning the environment, information search, or attention transfer. Therefore, eye-tracking data are not just raw coordinates, but can be interpreted as behavioral indicators for analyzing user interaction with a system [4].

Eye-tracking technology has been used in many fields, including cognitive psychology, reading studies, education, marketing, usability evaluation, and assistive systems. In the field of human-computer interaction, this technology is used both as an analytical tool to study user behavior and as an input modality in interactive interfaces. In the first case, gaze data is analyzed to better understand user behavior; in the second case, the same data can be used in real time to select, control, or activate system functions [5].

Hardware and software advances in recent years have made eye-tracking systems more accurate, more accessible, and more practical than in the past. In addition to specialized devices, more recent research has moved towards using conventional cameras, appearance-based methods, and machine learning models to estimate gaze. This trend has transformed eye tracking from a largely laboratory tool to a technology with potential for use in real-world environments and intelligent interactive systems [6].

In this thesis, the importance of eye tracking is that gaze data can help the system identify which parts of the text the user is viewing more closely and when further revision, correction, or linguistic assistance is likely to be needed. From this perspective, eye tracking in this research is not just a tool for observing user behavior, but also part of the system's

interaction logic that plays a role in activating linguistic analysis and providing text suggestions [5], [6]. The general working principle of an eye-tracking system is illustrated in Figure 2.1.

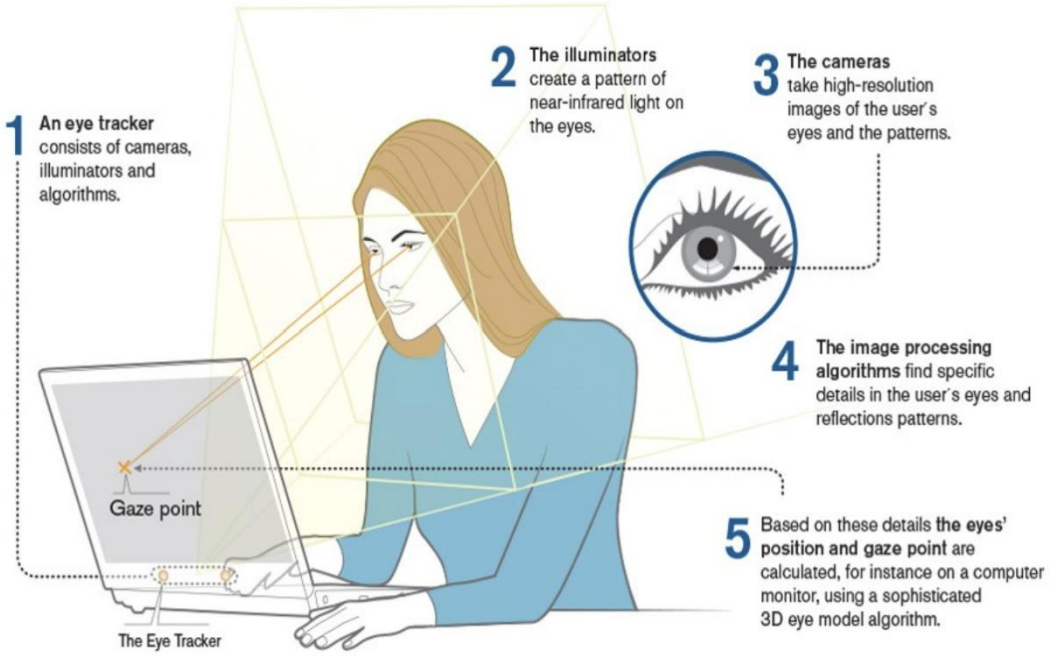

Figure 2.1: General working principle of an eye-tracking system, including infrared illumination, eye image acquisition, feature detection, and gaze estimation. Source: Adopted from Tobii (2024), "How do Tobii eye trackers work?", available at: https://connect.tobii.com/s/article/How-do-Tobii-eye-trackers-work?language=en_US (Accessed: 16 April 2026).

2.2 Eye Tracking Technologies

Eye-tracking technologies can be categorized based on the type of hardware, the way the eye image is captured, the level of contact with the user, and the method of estimating the gaze point. In recent years, most of the common eye-tracking systems have been developed based on capturing an image of the eye and analyzing features such as the pupil, iris, and light reflection from the corneal surface. However, the type of device and the environment of use play an important role in the accuracy, ease of use, and usability of these systems [4], [6].

In general, eye-tracking technologies can be examined in four main groups: infrared eye trackers, remote eye trackers, head-mounted eye trackers, and camera-based gaze estimation systems. These categories overlap somewhat from a technical point of view, but they are very common in the research literature to describe their functional and practical differences [4], [6].

2.2.1 Infrared

A significant portion of modern eye trackers use infrared light to illuminate the eye and extract more stable image features. In these systems, the infrared light irradiation makes the pupil and corneal reflections appear more clearly in the image, allowing tracking algorithms to estimate eye position and gaze direction more accurately. This approach is particularly common in video-based systems and has become one of the standard methods in eye tracking due to its non-invasiveness and reasonable accuracy [4], [9].

The main advantage of infrared systems is the greater stability in detecting eye features and better performance compared to simple visible light-based methods. However, these systems can still be affected by factors such as ambient lighting conditions, unwanted reflections, user head position, and calibration quality [4].

2.2.2 Remote Eye Trackers

Remote eye trackers or screen-based trackers are devices that are usually placed below or above the display and record the user's eye movements without direct contact with the user. These types of systems are usually suitable for desktop scenarios, usability studies, text reading, and screen-based interactions, as the user can work with the system without wearing any special equipment [4], [6].

The most important advantage of this category is the ease of use and the more natural user interaction with the environment. However, remote trackers are sensitive to head movement, user distance from the screen, and calibration quality. For this reason, although they are a good option for many HCI applications, their accuracy may decrease in situations where freedom of movement is high or the user environment is complex [4], [10].

2.2.3 Head-Mounted Eye Trackers

In head-mounted eye trackers, cameras and sensors are mounted on a headset, glasses, or similar wearable device. This structure makes the device less sensitive to head movement and can be used in more dynamic environments, including more naturalistic scenarios, virtual reality, or off-the-shelf behavioral studies. As a result, head-mounted trackers have become particularly important in advanced laboratory studies, immersive environments, and some medical and industrial applications [7], [9].

On the other hand, these types of systems are usually more limited in terms of ease of use, as the user must wear the device on their head or face. Also, higher cost and more complexity in installation and setup are other challenges in this category. Despite these limitations, head-mounted trackers are a very important option in situations where more accurate recording of eye behavior in 3D or moving environments is required [7].

2.2.4 Camera-Based Gaze Estimation

In recent years, considerable attention has been paid to camera-based gaze estimation methods, i.e. methods that attempt to estimate the gaze direction or approximate gaze point without a specialized eye tracker and using only ordinary cameras. This approach has grown particularly with the advancement of machine learning and computer vision, which has increased the possibility of using eye tracking in more common devices, such as laptops and smartphones [6], [10].

The main advantage of these methods is their reduced cost and increased accessibility. For example, recent research has shown that eye tracking can be performed with reasonable accuracy using the front-facing camera of smartphones and machine learning models. However, these methods are generally less accurate, robust, and have less tolerance to environmental changes than specialized systems, and still face challenges such as facial variability, head angle, ambient light, and device differences [6], [8].

Overall, the choice between these technologies depends on the application. If the goal is to study eye behavior in a precise and controlled manner in a desktop environment, remote infrared-based trackers are usually a good choice. If the goal is to record eye behavior in more dynamic or immersive environments, head-mounted systems have more advantages. Conversely, if accessibility, scalability, and lower cost are priorities, camera-based methods can be a more suitable option, although they usually come with a compromise in accuracy [4], [6], [8]. Different types of eye-tracking devices, including both wearable and remote systems, are illustrated in Figure 2.2.

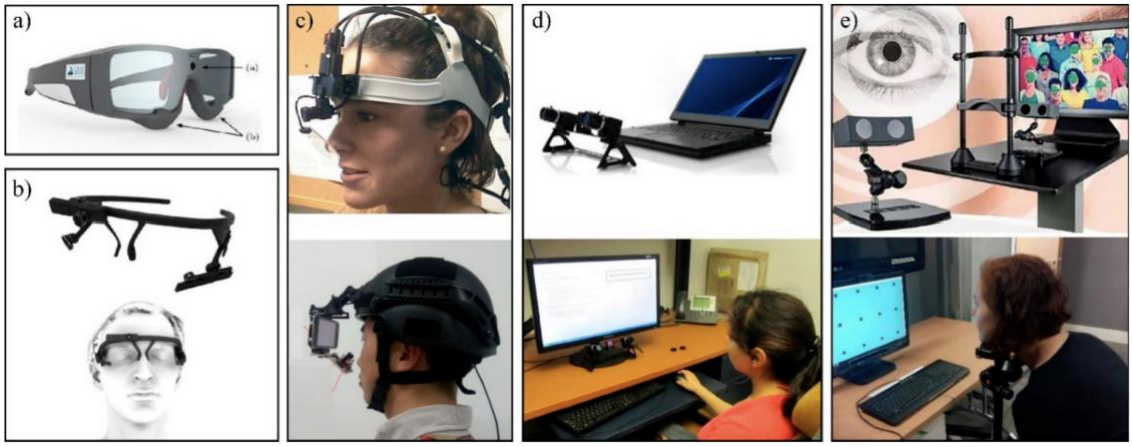

Figure 2.2: Examples of different types of eye-tracking devices: (a) eye-tracking glasses; (b) headband; (c) helmet-mounted; (d) remote/table-mounted; (e) tower-mounted. Source: Adapted from Martinez-Marquez, D., Pingali, S., Panuwatwanich, K., Stewart, R., & Mohamed, S. (2021). Application of Eye Tracking Technology in Aviation, Maritime, and Construction Industries: A Systematic Review. Sensors, 21(13), 4289. https://doi.org/10.3390/s21134289.

2.3 Eye Movements

2.3.1 Fixation

One of the most fundamental concepts in eye tracking data analysis is fixation. A fixation is a period of time during which the user's gaze remains relatively stable on a single point or visual area for a short period of time. In this case, although the eye is never completely still, the changes in gaze position are much more limited compared to rapid eye movements, and it can be assumed that the user's visual attention remains on the same area [4], [11].

The importance of fixation is that in many studies, this event is considered as one of the main indicators of visual attention and cognitive processing. When a user fixes on a piece of text, an image, or a user interface element, this event is usually interpreted as meaning that that part is being viewed, examined, or actively processed. For this reason, fixation analysis plays an important role in research on reading, text comprehension, usability, visual search, and human-computer interaction [4], [5].

The duration of fixation is usually short, but it is considered very valuable for extracting behavioral information. In practice, many eye-tracking analysis methods define fixation as a period of relative gaze stability that can have different lengths depending on the task, the user, and the detection algorithm. For this reason, the definition of fixation at the operational level usually depends on the event detection algorithm, and different studies may use different

temporal and spatial thresholds for its detection [11]. This is important because fixation is not a purely theoretical concept, but in practice must be extracted from noisy and real gaze data.

In interactive environments, fixation is usually linked to the concept of attentional focus. In other words, if the user remains on an area for a relatively long time, the system can interpret this behavior as a sign of interest, attention, or intention to interact. For this reason, many gaze-based systems use fixation or dwell time to activate commands, select a target, or initiate an interactive process [5]. This is also of direct relevance to the present study, since the main logic of the system is based on the assumption that fixation on a sentence can be a signal that the user needs to review or analyze the linguistic content of that sentence.

Despite the importance of fixation, its accurate detection in eye tracking data is not always easy. Device noise, small eye movements, blinking, changes in ambient light, and differences between users can blur the fixation boundary. For this reason, various algorithms have been developed for fixation detection, mainly based on criteria such as the spatial dispersion of gaze points or the speed of eye movement [11]. This challenge shows that fixation is not only an important behavioral concept, but also a technical issue in gaze data processing.

In the context of this thesis, fixation is of particular importance, as it is used as one of the main signals to detect the user's attention to a sentence and to activate linguistic analysis. In fact, the system does not rely solely on the momentary position of the gaze, but also considers the stability of the gaze over time in order to distinguish between a fleeting glance and a meaningful attention. This makes fixation in this research play a role beyond an analytical indicator and become part of the interactive logic of the system [4], [5], [11].

2.3.2 Saccade

A saccade is a rapid and sudden movement of the eye from one fixation point to another. This movement occurs when the visual system wants to shift visual attention from one area to another. In contrast to fixation, in which the gaze remains relatively stable for a short period of time, a saccade is performed at high speed and in a short time interval, and its main purpose is to change the location of the gaze, not to directly extract visual information [4], [11].

In many eye-tracking studies, saccades are considered a link between successive fixations. In other words, the user first fixes on one area, then transfers his gaze to another area with a saccade movement, and a new fixation is formed. Therefore, analyzing the sequence of

fixations and saccades helps researchers to better understand the visual scanning pattern, visual search strategy, and how the user's attention is transferred [5], [12].

A key feature of saccades is that when they occur, the visual system's ability to process visual information accurately is temporarily reduced; for this reason, it is usually assumed that the main part of the information extraction takes place in fixations, not in the saccade itself. From this perspective, saccades play a more transitional role that moves the gaze between different areas of the scene, text, or user interface [12]. This is also important in the analysis of reading behavior, because when reading text, the eye does not slide continuously over words, but rather proceeds through a series of successive fixations and saccades [12].

The pattern of saccades can provide valuable information about the type of task and the user's behavior. For example, in text reading, saccades are usually shorter and more structured, while in visual search or free viewing of a scene, the amplitude and direction of saccades may be more varied. Also, the presence of backward saccades or regressions in the text can be an indication of the user returning to previous sections for rereading or better understanding [12]. Therefore, saccade is not just a simple mechanical movement, but is part of the user's behavioral pattern in interacting with visual information.

Technically, saccade detection in gaze data is usually based on the speed or acceleration of eye movement. Many event detection algorithms identify intervals in which the gaze movement speed exceeds a certain threshold as saccades. For this reason, sampling quality, device noise, and data processing method play an important role in the accuracy of this event detection [11]. This shows that, like fixation, saccade is both a behavioral concept and a processing issue in the analysis of eye tracking data.

In the context of this thesis, saccades are important because they allow the system to distinguish between fleeting glances and relatively sustained attention. Since the goal of the system is to enable linguistic analysis based on the user's attention to a sentence, rapid, transitory eye movements should not be interpreted as a sign of meaningful interaction. Consequently, the distinction between fixation and saccade is essential for the stability of the gaze-based interaction logic in the system [4], [5], [11].

2.3.3 Smooth Pursuit

Smooth pursuit is a type of eye movement that allows a user to continuously follow a moving target. Unlike saccades, which involve rapid, jumping gaze movements from one point to

another, in smooth pursuit the eye attempts to move at a speed close to the target's speed to keep its image stable on the fovea and allow for accurate viewing [4], [13].

The importance of smooth pursuit is that it demonstrates that the human visual system is not only designed to view fixed targets, but can also adjust gaze relatively continuously when confronted with moving objects. This type of movement is important in activities such as following a moving object, watching a video, driving, or interacting with dynamic environments. Therefore, smooth pursuit is an important topic in visual perception studies, neuroscience, and some gaze-based interactive applications [13], [14].

From a behavioral perspective, smooth pursuit occurs when the visual target itself is moving. In such situations, fixation on a fixed point is not sufficient, because the target image is constantly changing its location. Therefore, the eye movement system tries to follow the target path by producing a smooth and continuous movement. For this reason, smooth pursuit can be considered a mechanism for fixing gaze on a moving target, while fixation is more concerned with maintaining gaze on a fixed target [13].

Compared to fixation and saccade, smooth pursuit is more difficult to detect in eye tracking data. On the one hand, its speed is faster than fixation, and on the other hand, it does not reach the very high speeds of saccades. This makes the boundary between pursuit and other ocular events not always completely clear, especially in noisy or low-sampling data. For this reason, some studies consider smooth pursuit to be one of the most complex events to detect algorithmically in eye tracking data [11], [14].

In many HCI and text analysis applications, the role of smooth pursuit is less prominent than fixation and saccade, because interaction with text elements and static interfaces is usually analyzed based on fixation and dwell time. However, familiarity with smooth pursuit is essential for a more complete understanding of eye behavior, as it shows that not all eye movements can be reduced to just two categories: fixation and saccade. Furthermore, in some advanced interaction scenarios, especially in dynamic environments, games, or gaze-based systems based on moving targets, smooth pursuit can directly contribute to interaction design [5], [13].

In the context of this thesis, smooth pursuit is of less practical importance than fixation, since the proposed system mainly deals with text and fixed sentences on the screen. However, its consideration in this chapter is important for two reasons: first, it provides a more complete picture of the classification of eye movements; and second, it clarifies the conceptual boundary

between sustained attention on a fixed area and pursuit of a moving target. This distinction is useful for better understanding the logic of gaze-based interaction in text systems [4], [13]. The main types of eye movements, including fixation, saccade, and smooth pursuit, are illustrated in Figure 2.3.

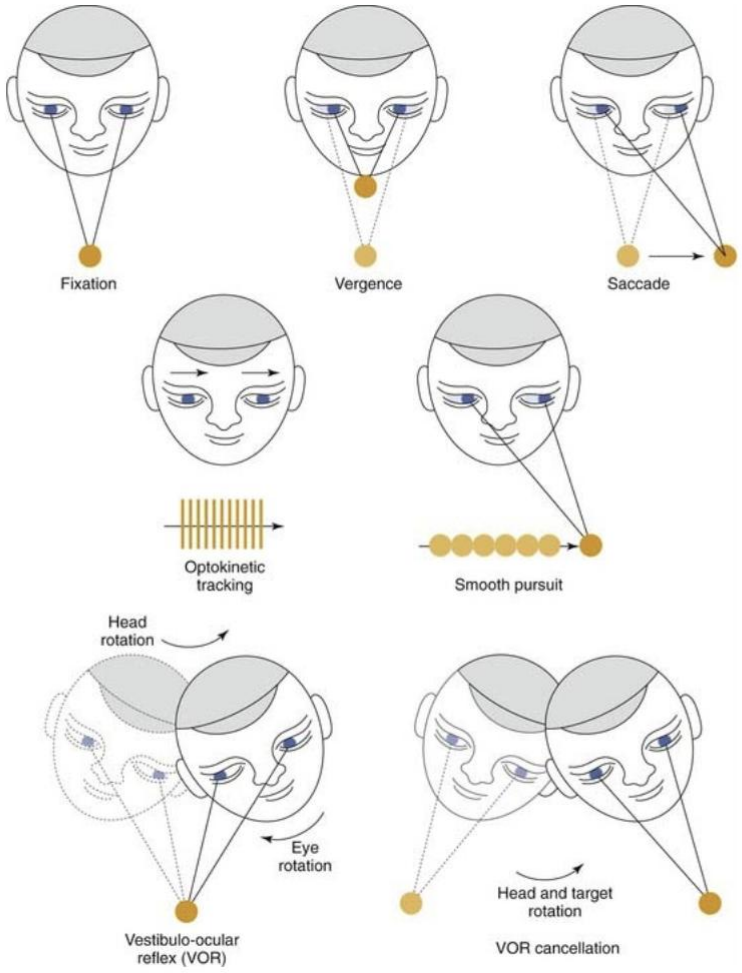

Figure 2.3: Schematic representation of different types of eye movements, including fixation, saccade, and smooth pursuit. Source: Adapted from EntoKey (n.d.), “Evaluation of the Patient with Dizziness”, available at: https://entokey.com/evaluation-of-the-patient-with-dizziness/ (Accessed: 16 April 2026).

2.4 Gaze-Based Interaction

2.4.1 Gaze as an Input Modality

In the field of human-computer interaction, gaze data is not only used to analyze user behavior, but can also be used as an input method. In this case, the system not only records the user's

gaze path, but also uses it to control the interface, select elements, or activate commands. For this reason, gaze has been proposed in many studies as an input modality that can act alongside or instead of traditional inputs such as the mouse and keyboard [5], [15].

Using gaze as input has the advantage of directly reflecting the user's visual attention direction. In many situations, the user first looks at the target before any other conscious action. Therefore, gaze can provide the system with immediate and natural information about the user's point of interest or focus. This feature has made gaze-based interaction attractive in applications such as assistive interfaces, immersive environments, hands-free systems, and some smart interfaces [5], [15].

However, using gaze as input comes with a fundamental challenge: the human eye plays two different roles simultaneously. On the one hand, the eye is the main tool of visual perception, and the user has to look at different parts of the environment to understand it. On the other hand, if the same gaze is interpreted as an input command, the system must be able to distinguish between "seeing" and "interaction intent." This is what is known in the gaze-based interaction literature as the Midas Touch problem; that is, a situation in which each user's gaze may be mistakenly interpreted as an interactive command [15], [16].

To mitigate this problem, in many systems, gaze is not used alone as a direct command, but is accompanied by complementary mechanisms. These mechanisms include dwell time, confirmation clicks, head gestures, combination with voice input, or multimodal use of gaze and other modalities. The goal of all of these approaches is to ensure that the system only responds to the user's gaze when there is sufficient evidence of an interaction intention [15], [16]. Thus, the design of gaze-based interfaces typically requires a balance between interaction speed, selection accuracy, and avoidance of unintended activation.

From an interface design perspective, gaze as an input has characteristics that distinguish it from the mouse and keyboard. Gaze typically moves faster than hand movements toward a target and can be very effective in determining the user's area of interest. However, gaze has typically lower spatial accuracy than tools such as a mouse, and its data is also

sensitive to noise, jitter, and changes in the user's posture. Therefore, in many applications, gaze is used as a complementary input or signal of intent, not necessarily as a complete replacement for all traditional input methods [5], [15].

In the context of this thesis, the user's gaze is used with exactly the same logic. The proposed system does not use gaze to completely replace traditional interaction, but to identify

parts of the text that have attracted the user's attention. In other words, gaze in this study is a meaningful behavioral signal to detect the user's area of interest and activate linguistic analysis. This approach is in line with the common view in HCI that gaze is a powerful modality but requires careful interpretation [5], [15], [16].

2.4.2 Dwell-Time Interaction

In many gaze-based systems, the concept of dwell time is used to ensure that the user's gaze is not directly and immediately interpreted as an interactive command. Dwell time refers to the length of time that the user maintains their gaze on a target, area, or user interface element. If this time exceeds a predetermined threshold, the system interprets it as an indication of interaction intent and performs a specific action [5], [15].

The importance of dwell time is that it provides a relatively simple and practical way to solve part of the Midas Touch problem. Since not all user gazes necessarily imply selection or activation intent, simply placing the gaze on a target cannot be a sufficient basis for executing a command. In contrast, remaining gazed on the same target for a certain period of time provides more evidence of meaningful attention or decision by the user. For this reason, dwell-based interaction is one of the most common approaches in the design of gaze-based interfaces [15], [17].

However, choosing the appropriate threshold for the dwell time is an important design issue. If the threshold is too short, the probability of unintended activation increases, and the system may react to fleeting glances or natural viewing behaviors. Conversely, if the dwell time is chosen too long, the interaction will become slow, tedious, and unnatural. Therefore, the interface designer must balance the speed of interaction, selection accuracy, and user comfort [17], [18]. This balance depends on the type of task, user characteristics, device accuracy, and even the size and location of interface elements.

In practice, the dwell time is usually defined as a fixed time interval, but in some more advanced systems, this value can be adaptive. For example, a longer dwell threshold may be considered for smaller elements or targets that are more likely to be selected incorrectly. Some systems also use visual or temporal feedback to let the user know that a command will be activated if they maintain their gaze for a longer period of time. This feedback can be displayed in the form of a progress bar, color change, countdown, or gradual highlighting, and plays an important role in making the system behavior predictable [15], [18].

Dwell time is widely used in assistive interfaces, eye-type, button selection, menu navigation, and hands-free systems. In such environments, dwell allows the user to make a selection simply by maintaining their gaze on the target, without the need for physical clicking or manual input. However, research has shown that dwell-based selection can lead to fatigue, reduced efficiency, and poor user experience if applied without careful design [17]. Hence, dwell time, although a simple and effective method, its success depends on the appropriate tuning of parameters and the overall interface design.

In the context of this thesis, the interaction logic of the system is directly related to the idea of dwell-based interaction. The system does not rely only on the momentary position of the gaze, but also considers the stability of the user's attention over time. In other words, the system attempts to distinguish between a fleeting glance and a meaningful glance that may indicate the need for revision or analysis of the sentence. This approach enables linguistic exploration to be activated based on sustained attention, rather than simply momentarily passing the gaze over the text [5], [15], [17]. The process of dwell-time interaction is illustrated in Figure 2.4, showing how sustained gaze over a target leads to gradual activation and final selection.

Figure 2.4: Stages of dwell-time interaction, including onset delay, feedback, and final selection based on gaze duration. Source: Adapted from Microsoft (n.d.), “Eye-gaze and dwell”, available at: https://learn.microsoft.com/en-us/windows/mixed-reality/design/gaze-and-dwell-eyes (Accessed: 16 April 2026).

2.4.3 Gaze in Human-Computer Interaction

In the field of human-computer interaction, eye tracking has a dual role: on the one hand, it is used as a tool for analyzing user behavior and evaluating interfaces, and on the other hand, it can be used as an interactive component in the design of new interfaces. This duality has made

gaze in HCI not only a source for observing visual attention and usage patterns, but also a communication channel between the user and the system [5], [15].

In an analytical role, gaze data helps researchers gain a more detailed understanding of how the user interacts with different interface elements. For example, by examining fixations, saccades, and visual scanning patterns, it is possible to determine which parts of the interface the user views most, what points attract their attention, and where there is a possibility of confusion or difficulty. For this reason, eye tracking has been widely used in usability studies, user experience design, education, and cognitive analysis of digital interfaces [5], [19].

In an interactive role, gaze can act as an input to select, guide, or activate system functions. This approach is particularly important in environments where traditional input is difficult, slow, or inappropriate; for example, in hands-free systems, assistive technologies, virtual and augmented reality environments, or interfaces that require rapid response to the user's attention. In such scenarios, gaze can allow the system to adapt its behavior according to the user's visual focus, creating a more natural and less mediated interaction [15], [16].

Despite these advantages, the use of gaze in HCI has always been accompanied by limitations and design considerations. The most important of these is that human gaze is inherently designed for perception, not simply for control. Therefore, if every gaze is interpreted as direct input, the probability of errors and unintended activations increases. Therefore, in many studies, gaze is considered not as a complete replacement for the mouse and keyboard, but as a complementary modality that should be used with appropriate interpretation logic, time thresholds, or other confirmation inputs [5], [15].

HCI research has also shown that the effectiveness of gaze-based interaction depends on several factors, including the accuracy and precision of the device, the quality of calibration, the user's position relative to the screen, the size of the interface targets, and the type of task. Even when the eye tracker performs well, individual differences between users and the variety of application contexts can affect the interactive experience. For this reason, the design of gaze-aware interfaces requires simultaneous consideration of technical, cognitive, and experiential factors [19]. This is especially important when the goal is to build a functional and reliable system for real-world use.

In recent years, gaze has received more and more attention as part of multimodal interaction. In this view, gaze is not solely responsible for the complete control of the system, but is used in conjunction with other modalities such as mouse, keyboard, touch, voice, or user

gestures. Such an approach allows the system to use gaze to identify the area of interest or guess the user's intent, while leaving the final decision or command execution to a complementary mechanism [15]. This approach usually performs better in terms of usability and robustness than purely gaze-based interfaces.

In this thesis, gaze is used exactly in this HCI framework. The proposed system is not designed to completely replace traditional interaction, but to utilize gaze as a signal of attention and interaction intent. The role of gaze here is to identify the sentence or part of the text that has caught the user’s attention and use this information to activate linguistic analysis and provide textual suggestions. Therefore, the use of gaze in this research is consistent with contemporary approaches to HCI, where gaze is considered as a meaningful input but requires intelligent interpretation [5], [15], [16].

2.5 Challenges of Eye Tracking

2.5.1 Noise and Jitter

One of the most important challenges in using eye tracking systems is the presence of noise and jitter in gaze data. Noise refers to unwanted fluctuations and errors that cause the recorded gaze coordinates to not fully match their actual position. Jitter usually refers to small, rapid, and continuous changes in the recorded gaze points that can be observed even when the user is looking at a relatively fixed area [4], [19].

The presence of noise and jitter means that eye-tracking data does not always appear completely smooth, stable, and free of fluctuations in practice. This is especially important in interactive applications, as the system may misinterpret these small changes as actual gaze movements or changes in the user's attention. As a result, if the data is used without proper processing, the probability of erroneous selections, unintended activations, and instability in interaction increases [19], [20].

The origin of these fluctuations can be very diverse. Part of it is related to the limitations of the device itself, such as sensor accuracy, camera quality, sampling rate, and gaze point estimation method. Another part depends on the conditions of use, such as very small head movements, blinking, light reflections, the user's position relative to the display, the use of glasses or lenses, and the ambient lighting conditions [4], [20]. Therefore, noise and jitter are

not just a software problem, but are the result of the interaction between hardware characteristics, the environment, and the user's natural behavior.

This is particularly important in the detection of eye events such as fixation. Since fixation is usually defined based on the relative stability of gaze over a period of time, any additional scatter in the data can cause the fixation boundary to become blurred or the detection algorithm to split a real fixation into several parts. For this reason, many studies have emphasized that data quality and noise have a direct impact on the accuracy of event detection [11], [20].

To reduce the effect of noise and jitter, various strategies have been proposed in the literature. One of the most common methods is the use of filtering or smoothing of the data before analysis. In addition, some fixation and saccade detection algorithms are designed adaptively to be able to adapt to natural variations in data quality. Also, in the design of gaze-based interfaces, temporal thresholds, more tolerable regions, or complementary verification mechanisms are usually used to avoid oversensitivity to minor fluctuations [11], [19].

In the context of HCI, understanding noise and jitter is of great importance, as these two factors indicate that gaze cannot be considered as a perfectly accurate and error-free pointer. In practice, the user's gaze is usually less precise and more volatile than a mouse, and this characteristic makes the design of gaze-aware interfaces require special attention to stability, fault tolerance, and data interpretation methods [19]. This issue is also directly relevant to the present thesis, since the proposed system must be able to distinguish between minor fluctuations of gaze and meaningful attention of the user to a sentence.

In the context of this research, paying attention to noise and jitter is important because gaze data is used as one of the main bases for activating linguistic analysis. If the system is too sensitive to small fluctuations, it may misinterpret transient glances or random changes as meaningful fixations. Therefore, in designing such a system, it is essential to consider gaze stability and use temporal and spatial metrics to increase the robustness of the interaction [4], [11], [19].

2.5.2 Calibration Issues

Calibration is a fundamental step in the use of eye-tracking systems, and its quality has a direct impact on the accuracy of gaze data. In this step, the system attempts to establish a relationship between the image features recorded from the user's eyes and the actual coordinates of their

gaze on the screen. In other words, calibration is the process by which the system learns how to map the raw eye data to the gaze point on the display environment [4], [6].

The importance of calibration stems from the fact that users' physiological and behavioral characteristics are not the same. Differences in eye shape, pupil size, head position, use of glasses or contact lenses, and even the way the user sits in relation to the screen can affect the way the eyes are recorded and, consequently, the gaze estimation. For this reason, many eye tracking systems require a calibration step for each user before they can be used, in order to adjust the gaze estimation model to their individual conditions [4], [21].

However, calibration is not always a perfect process. If the user is not focused enough during calibration, moves his/her head, or changes his/her distance and body position from the display after calibration, the quality of the mapping will decrease. Also, even a successful calibration can drift over time; that is, the accuracy of the system gradually decreases and the registered gaze points move away from the true position [21], [10]. This problem is especially important in interactive applications that depend on spatial accuracy and temporal stability.

From a practical point of view, calibration problems can have significant consequences. In gaze-based systems, if the gaze point is not estimated correctly, the system may select the wrong target, detect fixation on the wrong area, or interpret the user's behavior incorrectly. In analytical studies, poor calibration can also lead to data that cannot be interpreted behaviorally or cognitively [19], [21]. Therefore, calibration is not just a preliminary technical step, but a prerequisite for the reliability of eye tracking data.

To mitigate this problem, various solutions have been proposed in research and practical systems. One of these solutions is to perform a multi-point calibration and check its quality before starting the main task. Some systems also use recalibration or drift correction during use to update the gaze map if the user's conditions change. In addition, gaze-based interfaces usually need to be designed to be somewhat robust to minor calibration errors; for example, by making the selection areas larger, using dwell time, or combining gaze with other confirmation signals [15], [21].

In real-world environments, calibration is one of the important factors that makes everyday use of eye tracking more difficult than in laboratory applications. The more controlled the environment, the more likely it is to achieve a more accurate calibration; however, in real-world environments, changing lighting, user movement, and differences in natural behaviors can reduce this accuracy. For this reason, one of the important directions in recent research is

to develop methods that are less dependent on precise calibration or that can adjust themselves adaptively during interaction [6], [10].

In the context of this thesis, calibration is of great importance because the proposed system relies on relatively accurate detection of the user's attentional region in the text. If the calibration is poor, the user's gaze may be mapped to the wrong sentence or location, and as a result, the fixation-based triggering logic may fail. Therefore, the quality of calibration affects not only gaze registration, but also the entire chain of attention detection, language analysis activation, and suggestion presentation [4], [15], [21]. An example of the eye-tracking calibration process used in this study is shown in Figure 2.5.

Figure 2.5: Calibration interface of the eye-tracking system, showing target points and the detected gaze position during calibration.

2.5.3 Gaze Instability

Gaze instability refers to the relative instability of gaze over time; That is, a state in which, even when the user is apparently focused on a fixed area, the recorded gaze data does not remain completely stable and is subject to fluctuation, local drift, or small shifts. This phenomenon can be caused by natural physiological factors of the eye, device limitations, very

small changes in head position, or environmental conditions. As a result, the recorded gaze point is not always a completely stable representation of the user's attention location and must be interpreted probabilistically and with error tolerance in mind [4], [19], [20].

One important reason why gaze instability is important is that interactive systems typically require a relatively stable mapping between the user's gaze and the target on the screen. If this mapping is unstable, the system may switch between several adjacent targets, misdetect fixation, or reconstruct the attention region in an unstable manner. This is especially prominent in interfaces that deal with text, small buttons, or dense elements, as even small changes in gaze can be perceived as significant by the system [19], [20].

Gaze instability is not limited to device noise. Even with apparent gaze fixation, the human eye is not completely stationary, and very small natural movements can be observed in the gaze pattern. In addition, differences between users, fatigue, calibration quality, ambient light, and changes in sitting distance or angle to the display can also increase the degree of instability. For this reason, gaze instability should be considered from both a physiological and a technical perspective [4], [20].

In the design of gaze-based systems, the presence of this instability makes direct reliance on instantaneous gaze coordinates inappropriate. Instead, many systems use strategies such as data aggregation over short time intervals, the use of more tolerant regions, fixation detection based on relative stability, and smoothing filters. Such approaches help the system interpret the user's overall and more meaningful behavior rather than reacting to every small change [11], [19].

In the context of this thesis, gaze instability has direct practical importance, because the proposed system must be able to distinguish between natural but unstable gazes and the user's actual attention to a sentence. If this instability is not taken into account in the system logic, a sentence may be mistakenly missed, an adjacent sentence may be selected, or the activation of linguistic analysis may occur at an inappropriate moment. Therefore, in such an application, the relative stability of the gaze and its interpretation over time is more important than the raw gaze coordinates themselves [4], [5], [19].

2.5.4 Accuracy and Precision Limitations

In the evaluation of eye-tracking systems, two concepts are considered to be the most important indicators of data quality: accuracy and precision. Accuracy refers to the closeness of the

recorded gaze point to the user's actual gaze position, while precision indicates how stable and low-dispersion the gaze data are when the gaze is fixed. In simple terms, a system may have poor accuracy, but its data are relatively stable, or conversely, the average gaze position may be close to the target, but the data may be recorded with fluctuating precision [4]. The distinction between these two concepts is very important in interactive applications. If accuracy is low, the user's gaze point will be mapped to the wrong area on the screen and the system may select the wrong target. If precision is low, the data will be highly dispersed even when the user is paying steady attention, making it difficult to detect fixation, dwell time, or the boundary between adjacent areas. Therefore, for many gaze-based systems, both components must be considered simultaneously [19], [20].

Empirical studies have shown that accuracy and precision are not constant and uniform in practice, but can vary significantly between users, devices, lighting conditions, and even different screen regions. For example, a study by Feit et al. shows that tracking quality can vary significantly depending on the tracker, lighting condition, and screen region, and that these variations directly affect the interface design and the size of the interactive targets [19]. These findings suggest that using gaze in real-world environments requires an understanding of the practical limitations of the device, rather than relying solely on the manufacturer's stated specifications.

Several factors affect accuracy and precision. The most important of these are the quality of calibration, head movement, user distance from the display, ambient lighting conditions, use of glasses or lenses, camera quality, and sampling rate. In addition, the type of gaze estimation algorithm and data post-processing methods also play a role in the final quality. For this reason, the accuracy and precision limitations should be considered as a combination of hardware, software, environment, and user behavior [4], [20].

In the design of gaze-aware interfaces, these limitations have direct consequences. If the interface targets are very small or their distance from each other is small, even relatively small errors in accuracy and precision can cause incorrect selection. Therefore, many studies recommend that gaze-based interface design should be error-aware; that is, the size of elements, their spacing, time thresholds, and selection logic should be adjusted to be compatible with the actual quality of gaze data [19]. Also, using more tolerant regions, appropriate filters, and dwell time-based selection methods can help reduce the effect of these limitations [15], [19].

In the context of this thesis, the accuracy and precision constraints are of particular importance, since the proposed system relies on detecting the sentence of interest of the user and activating the linguistic analysis based on his gaze. If the gaze quality is low, the system may select the adjacent sentence, detect fixation on the wrong region, or activate the linguistic analysis at an inappropriate time. Therefore, in this research, the user's gaze is not considered as an absolute and precise signal, but rather as meaningful data with uncertainty [4], [19], [20]. The difference between gaze accuracy and precision is shown in Figure 2.6.

Figure 2.6: Gaze accuracy and precision illustrated by the position and dispersion of gaze points. Source: Adapted from Tobii (n.d.), “Signals & Quality”, available at: https://developer.tobii.com/xr/learn/eye-behavior/signals-and-quality/ (Accessed: 16 April 2026).

2.6 Eye Tracking in Writing and Assistive Interfaces

In recent years, eye-tracking technology has not only been used as a tool for studying visual attention or usability evaluation, but has also gradually gained an important place in the design of assistive interfaces and text-based interactive systems. The main reason for this is that gaze data can provide valuable information about the user's focus, text processing, reading difficulties, and possible intention while interacting with a text-based interface [5], [6].

In the field of assistive interfaces, eye tracking is especially important for users who, for motor or physical reasons, cannot fully use traditional inputs such as a mouse and keyboard. In such situations, gaze can serve as an alternative or complementary route for selecting, typing, controlling the interface, and interacting with the digital environment. Various studies have shown that gaze-based systems can play an important role in increasing accessibility and independence for users with motor disabilities, although the effectiveness of these systems depends on the accuracy of the device, appropriate interface design, and gaze interpretation method [15], [16].

One of the most important application areas of eye tracking in assistive environments is text entry and gaze typing. In these types of systems, the user can select letters, words, or text options by fixation, dwell time, or other gaze-based mechanisms. Research has shown that although gaze typing can be very valuable for users with motor limitations, challenges such as fatigue, selection errors, slower speed compared to manual input, and sensitivity to Midas Touch are still important issues in this area [17], [22].

In addition to text entry, eye tracking has also gained importance in writing interfaces and reading environments. In such environments, gaze data can be used to analyze reading behavior, detect difficult areas of text, assess attention span, and even predict where the user is likely to need help or revision. This feature makes eye tracking an attractive tool for designing intelligent systems that can respond not only to text but also to the user's visual behavior [5], [12].

In recent years, more recent research has moved towards combining eye tracking with other technologies, including machine learning models and intelligent language systems. In this approach, gaze is not simply a means of recording attention coordinates, but is used as a meaningful behavioral signal in interpreting user intent and personalizing interactions. For example, in a writing environment, prolonged fixation on a sentence can be a sign of ambiguity, difficulty, or the need for revision, and the system can then suggest corrections or explanations based on this behavior [6], [15].

In this context, eye tracking can play a special role in the development of writing assistants. Traditional writing assistants usually operate only on the basis of the entered text and do not know the actual behavior of the user when reading or reviewing the text. In contrast, a gaze-aware system can use gaze data to identify which parts of the text have attracted the user's attention or where there is a greater likelihood of difficulty. This difference can lead to smarter interaction, more targeted suggestions, and less system interference in parts of the text that do not pose a problem for the user [5], [15].

In conclusion, eye tracking in writing interfaces and assistive systems provides a bridge between visual attention and intelligent interaction. This technology enables the system to make decisions not only based on the explicit content of the text, but also on the implicit behavior of the user while reading or writing. Therefore, for the present study, eye tracking is not just an accessory, but a key part of the logic of the proposed writing assistance system, which aims to use the user's gaze to enable more accurate and meaningful linguistic analysis

[5], [6], [16]. An example of an eye-tracking-based spelling communication system is illustrated in Figure 2.7.

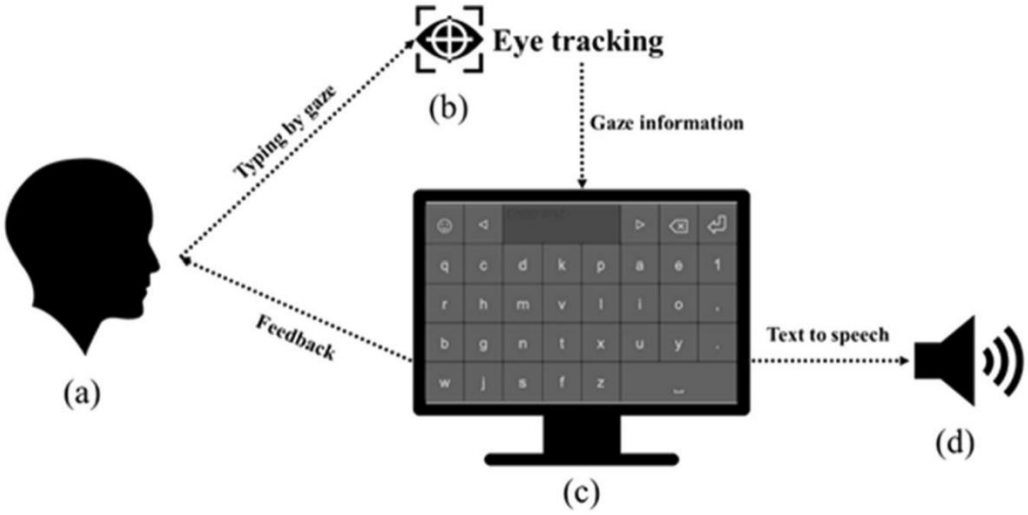

Figure 2.7: Eye-tracking-based spelling communication system, showing how gaze is used to select text on a virtual keyboard and generate output. Source: Adapted from Ngo et al. (2024), "An EEG & eye-tracking dataset of ALS patients & healthy people during eye-tracking-based spelling system usage", Scientific Data, 11, 664. https://doi.org/10.1038/s41597-024-03501-y.

2.7 Chapter Summary

In this chapter, the theoretical foundations of eye tracking technology are reviewed. First, the general concept of eye tracking and its place in the study of visual attention and human-computer interaction are introduced, then the main types of eye tracking technologies are reviewed, including infrared systems, remote trackers, head-mounted trackers, and camera-based methods [4], [6], [10].

Next, the three main categories of eye movements, including fixation, saccade, and smooth pursuit, are explained, and the role of each in the analysis of visual behavior and interaction with the digital environment is discussed [4], [11], [12], [13]. After that, the concept of gaze-based interaction is reviewed, and it is shown that the user's gaze can be used not only as a tool for behavior analysis, but also as an input modality in interactive interfaces. In this regard, the role of dwell time in reducing unwanted activations and transforming gaze into a reliable interactive signal was described [5], [15], [17].

The next part of the chapter was dedicated to the most important challenges of eye tracking. In this section, factors such as noise, jitter, calibration problems, gaze instability, and accuracy and precision limitations were examined, and it was shown that gaze data in real applications is always accompanied by uncertainty and should be used with caution and with appropriate interpretive logic [19], [20], [21]. This point is especially important for the design of gaze-based systems, because the success of interaction depends on the system being able to distinguish between natural fluctuations of the data and meaningful attention of the user [15], [19].

Finally, the place of eye tracking in assistive interfaces and writing environments was examined, and it was clarified that this technology can create a bridge between the user's visual attention and the interactive behavior of the system. This is directly relevant to the purpose of this thesis, as the proposed system is based on the idea that the user's gaze at a sentence can be used as a cue to activate linguistic analysis and provide writing assistance [5], [6], [16], [22].

Accordingly, this chapter provides the necessary theoretical foundation to understand the role of eye tracking in the proposed system. In the next chapter, the focus will shift from eye tracking technology to large-scale language models to investigate their role in grammar correction, text rewriting, and academic writing support [1], [2], [3].

Chapter 3 – Large Language Models

3.1 Introduction to LLMs

Large Language Models (LLMs) are a class of deep learning models that are trained to process, produce, and represent natural language on a very large scale. These models are usually built on the Transformer architecture and learn statistical patterns, semantic relations, and linguistic structures using a huge amount of text data. In recent years, LLMs have become one of the most important areas of research and application in natural language processing due to their high ability to understand and produce text [3], [23].

The importance of LLMs stems from the fact that these models are no longer designed to perform limited linguistic tasks, but can be used in a wide range of tasks such as question answering, summarization, text rewriting, translation, content generation, writing style analysis, and even assisting in research and writing processes. Unlike many traditional NLP systems that required a dedicated design for each task, large language models can be used across multiple tasks with prompting and minor tuning [23], [24].

The development of LLMs is largely tied to the development of the Transformer architecture. The famous paper “Attention Is All You Need” showed that linguistic dependencies could be effectively modelled without relying on recursive or convolutional structures, and using only the attention mechanism. This development paved the way for the emergence of a new generation of language models that were much more powerful in terms of scale, generalizability, and output quality than previous generations [3].

An important feature of LLMs is their ability to not only preserve surface patterns of text, but also to create more complex representations of context, word relations, and sentence structure. This ability does not mean “human understanding” of language, but rather the result of extensive statistical learning from large-scale text data. However, this statistical ability has made LLMs perform remarkably well in many practical applications, including writing assistance systems [23], [25].

In the field of writing, large language models play an increasingly important role in helping users correct grammar, improve vocabulary, rewrite sentences, simplify text, and produce more formal or academic versions. These applications are particularly important for non-native writers, students, and users who need linguistic support while writing. As a result, LLMs have become a key component of a new generation of intelligent writing assistants [24].

Despite these capabilities, the use of LLMs also comes with some limitations. These models may produce linguistically fluent but factually incorrect responses, may lack consistency in some cases, or may produce different outputs depending on the prompting method. Furthermore, many of their practical applications depend on external services, high processing power, and privacy considerations [23], [25].

In the context of this thesis, large language models play the role of the linguistic core of the proposed system. In this research, LLM is used not only to detect and correct grammatical errors, but also to generate simpler and more academic versions of sentences. Therefore, understanding the basics of LLMs is essential to better understand the logic of the proposed system. Accordingly, in the following sections of this chapter, the Transformer architecture, the application of LLMs in writing assistance, and their limitations will be examined in more detail [3], [23], [24]. Figure 3.1 shows the evolution of language models from traditional approaches to modern large language models.

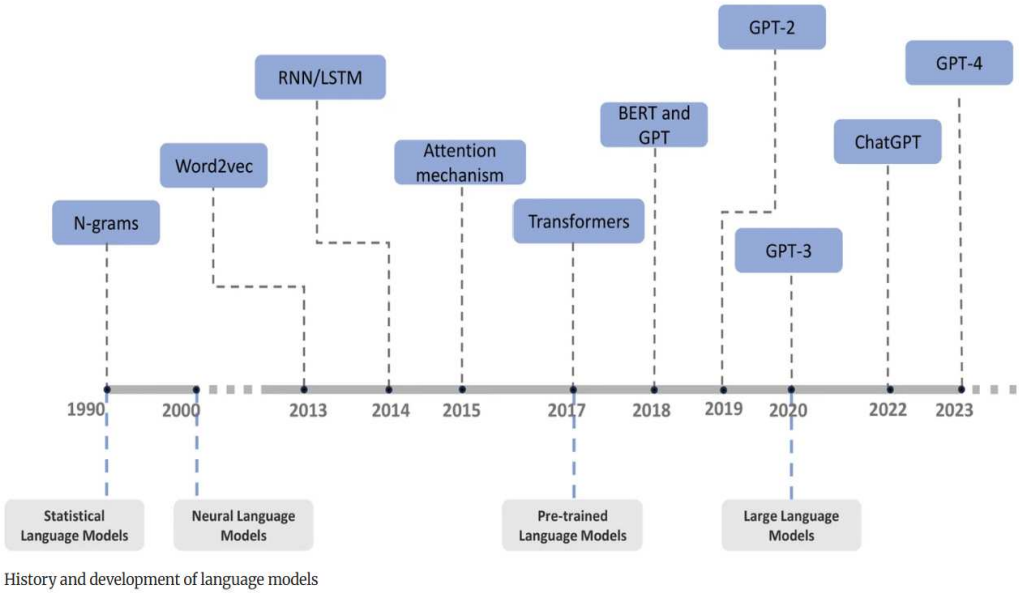

Figure 3.1: Evolution of language models from statistical approaches to large language models.

Source: Adapted from Wang et al. (2025), “History, development, and principles of large language models: an introductory survey”, AI and Ethics. https://doi.org/10.1007/s43681-024-00583-7.

3.2 Transformer Architecture

The Transformer architecture is considered one of the most important milestones in the evolution of large language models. Before the introduction of this architecture, many natural language processing systems relied on recurrent models such as RNN and LSTM. Although these models were effective at their time, they faced serious limitations in modeling long-term dependencies, parallel training, and scalability. By abandoning the recursive structure and relying on the attention mechanism, Transformer showed that dependencies between tokens can be modelled more directly and efficiently [3], [23].

The main idea of Transformer is that when processing a sequence, each token can evaluate the importance of other tokens in the same sequence and, based on this, obtain a richer representation of the context. This is done through self-attention. In self-attention, each token pays attention not only to itself, but also to other input tokens, and the weight of each connection is calculated based on similarity and relevance. In this way, the model can learn short- and long-range relationships without having to traverse the entire sequence sequentially [3].

One of the important advantages of this architecture is parallelization. In recurrent models, sequence processing is usually done sequentially, because the state of each step depends on the previous step. In contrast, Transformer removes this sequential dependency and allows different parts of the sequence to be processed simultaneously. This feature made it more practical to train very large models on big data, and this paved the way for the development of today's LLMs [23], [25].

The classic Transformer architecture consists of two main parts: an encoder and a decoder. The encoder is responsible for transforming the input sequence into contextual representations, while the decoder uses these representations to produce the output sequence. In some models, both parts are present, but in many large contemporary language models, only one of the two is used. For example, the GPT family of models is mostly decoder-only, while BERT is based on an encoder-only architecture [3], [23].

In each layer of the Transformer, two main components are seen: multi-head attention and feed-forward network. In multi-head attention, instead of using a single attention pattern, the model uses multiple heads of attention in parallel to learn different relationships between tokens from different angles. The output of these attentions is then processed by a feed-forward network to enrich the final representation of each token. In addition to these components,

residual connections and layer normalization are also used to ensure training stability and improve gradient descent [3].

Since Transformer lacks the inherent ordinal structure of recursive models, positional encoding is used to preserve information about the order of tokens. This component helps the model know where each token is in the sequence and to take into account the relative or absolute order of words in its analysis. Without such a mechanism, the model would see a set of tokens without knowing their order, and as a result, part of the linguistic structure would be lost [3], [25].

The importance of Transformer in this thesis is that the language models used in the proposed system are built on this architecture or its derivatives. The ability of these models to understand context, rewrite sentences, correct grammatical errors, and produce alternative versions of the text is largely a result of Transformer's ability to model complex linguistic relations. Therefore, understanding this architecture is essential to understanding the technical foundations of the LLMs used in the present help-writing system [23], [24], [25]. The overall structure of the Transformer architecture is illustrated in Figure 3.2, showing the encoder–decoder design and the main internal components such as self-attention and feed-forward networks.

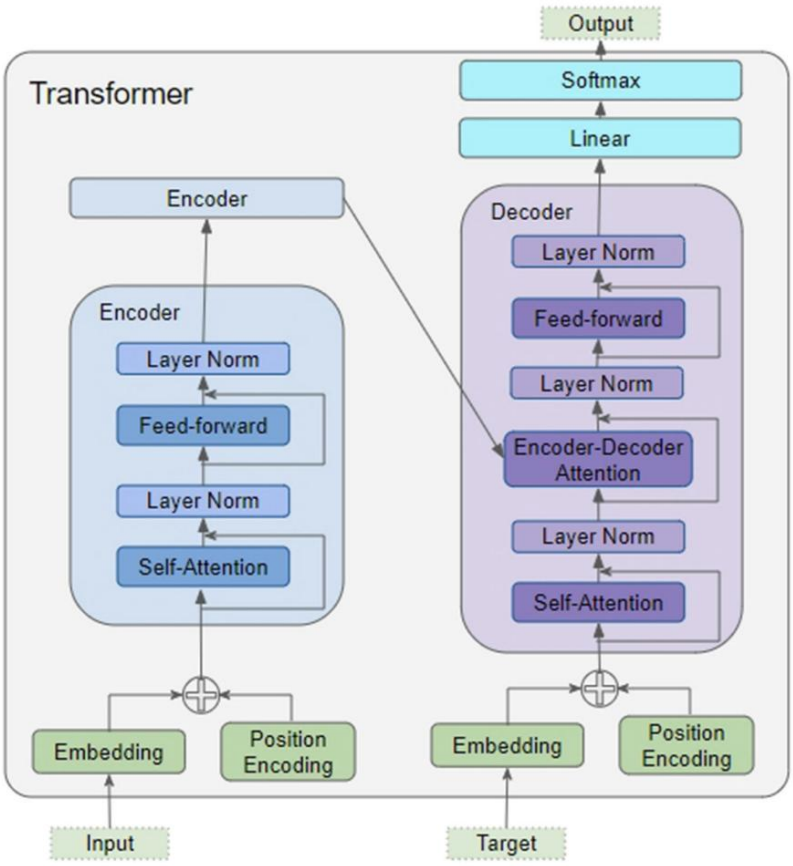

Figure 3.2: Structure of the Transformer architecture showing encoder–decoder components, including self-attention and feed-forward layers. Source: Adapted from Kumar and Solanki (2023), "An abstractive text summarization technique using transformer model with self-attention mechanism", Neural Computing and Applications. https://doi.org/10.1007/s00521-023-08687-7.

3.3 LLMs in Writing Assistance

Large language models have become one of the most important technologies in the development of intelligent writing assistance systems in recent years. These models can perform a wide range of linguistic tasks, including correcting grammatical errors, rewriting sentences, simplifying text, changing the tone of writing, summarizing, and generating stylistic suggestions. This versatility makes LLMs more flexible than many traditional writing assistance tools and can be used in more complex writing environments [26], [27].

In traditional tools, writing assistance was usually based on grammatical rules, predefined patterns, or more limited statistical models. These tools were useful in identifying some superficial errors such as spelling or grammar mistakes, but they usually had limitations in understanding context, detecting stylistic weaknesses, or providing natural and purposeful rewriting. In contrast, LLMs can take sentence-level and paragraph-level context into account

to a large extent and are therefore able to provide suggestions that are not limited to error correction, but also pay attention to clarity, coherence, and stylistic appropriateness [26], [28].

One of the most important applications of LLMs in writing assistance is grammar correction. In this application, the model can not only identify grammatical and lexical errors, but in many cases also provide a short explanation of the cause of the error and a suitable correction suggestion. This feature is especially important for non-native writers, because in addition to text correction, the system can also play a educational role and help the user better understand their error patterns [28].

Another important application is text rewriting. In this case, the LLM can rewrite a sentence or text in a more fluent, simpler, more formal, or more academic way while maintaining the original meaning. This capability is highly valuable in academic and professional settings, as many users not only need "error correction" but also want to improve the expressive quality, stylistic coherence, and tone appropriateness of their writing. Recent research also suggests that LLMs can go beyond grammar correction and play an effective role in revising and polishing text, although their consistency and accuracy are not the same in all cases [28], [29].

In the field of academic writing, LLMs have been particularly noted for producing a more formal tone, helping with sentence structure, simplifying complex expression, and suggesting more appropriate versions for academic text. However, the existing literature emphasizes that their use in academic writing should be accompanied by caution, because although these models can increase the speed and efficiency of writing, they may in some cases produce texts that are overly general, opaque, or factually incorrect. For this reason, LLM in this area should be seen more as a writing support tool than as an independent writer [26], [27].

Another important point in LLM-based writing assistance is the ability of these models to adapt the output to the prompt. The user can ask the model to make the text more formal, shorter, simpler, more academic, or clearer. This prompt-based control makes LLMs more flexible than many traditional tools. However, this dependence on prompts can cause variation in the output; that is, a small change in the input command or context may lead to a different response. Therefore, in the design of writing assistance systems, it is usually necessary to carefully engineer the prompts and process the outputs in a structured or controlled manner [27], [28].

In the context of this thesis, LLM is used in exactly such a role. The proposed system uses the language model both to correct grammatical errors and to produce simpler, more academic versions of sentences. Therefore, the LLM in this study is not simply a text generation tool, but rather the linguistic core of a writing assistance system that needs to be carefully designed both in terms of the quality of the suggestions and how it integrates with the logic of gaze-based interaction [27], [28], [29]. The main applications of AI and LLM-based writing assistance tools are illustrated in Figure 3.3.

Figure 3.3: Main functions of AI-based writing assistants, including grammar checking, content generation, and style suggestions. Source: Adapted from Great Learning Editorial Team (2025), "Top 10 AI Writing Tools of 2026". Available at: https://www.mygreatlearning.com/blog/top-ai-writing-tools/ (accessed April 16, 2026).

3.4 Prompt-Based Interaction with LLMs

One of the important differences between large language models and many traditional NLP systems is that their behavior can be largely controlled and guided by prompts. A prompt, in its simplest definition, is a text that specifies the task, context, constraints, and the type of output expected for the model. For this reason, in many practical applications, the quality of the LLM response depends not only on the model itself, but also on the design of the prompt [23], [30].

In prompt-based interaction, instead of retraining the model, the user or developer tries to guide the model's behavior in a desired direction using language commands, examples, explicit constraints, and appropriate formatting. This feature has allowed LLMs to be used in a variety of tasks such as grammar correction, text rewriting, information extraction, summarization, and question answering without extensive fine-tuning [23], [30].

One of the most common classifications in this field is the distinction between zero-shot prompting, few-shot prompting, and instruction-based prompting. In zero-shot prompting, the model performs the task with a single direct command without examples. In few-shot prompting, a few input-output examples are included in the prompt to make the expected pattern clearer for the model. GPT-3 research has shown that large models can perform significantly better in-context learning with a few examples, and subsequent research has shown that even simple zero-shot prompts can activate the latent capabilities of the model [3], [31].

In many engineering applications, the prompt is not only used to express the task, but also to control the form of the output. For example, in a help-writing system, the model may need to return the error type, a short explanation, and an alternative suggestion in a specific format in addition to the corrected sentence. In such cases, the prompt should clearly specify not only the task goal, but also the structure of the response, the linguistic constraints, and even the type of output keys [30], [32].

For this reason, the concept of structured outputs has gained increasing importance in recent years. In this approach, the developer attempts to constrain the model response to a structured format such as JSON to make the output processing in software systems more reliable. This is especially important in systems where the model output must be fed directly into other pipelines, such as correction, extraction, or recommendation systems. Structured output can reduce the likelihood of missing fields, format inconsistencies, or unwanted explanatory responses [32].

However, prompt-based interaction is not always completely deterministic. Small changes in the wording of the command, the order of the information, the presence or absence of examples, and even the type of linguistic constraints can cause differences in the model response. For this reason, prompt engineering has become an important topic in the development of LLM-based applications. Recent reviews also show that prompt design is in practice an iterative process that usually requires repeated testing, revision, and tuning to achieve more stable behavior of the model [30].

In the context of this thesis, prompt-based interaction plays a very important role, because the proposed system should force the model to produce specific, concise, and actionable outputs. Specifically, in the grammar correction section, the model should provide the corrected sentence, errors, suggestions, and a short explanation in a structured format; and

in the style analysis section, it should return the tone, suitability for academic writing, and simpler and more formal versions of the sentence. Therefore, the success of the system does not only depend on the LLM itself, but also on how the prompt is designed and the output is managed [23], [30], [32]. The process of prompt-based interaction, including prompt engineering and human validation, is illustrated in Figure 3.4, showing how model inputs are structured and how outputs are evaluated and refined.

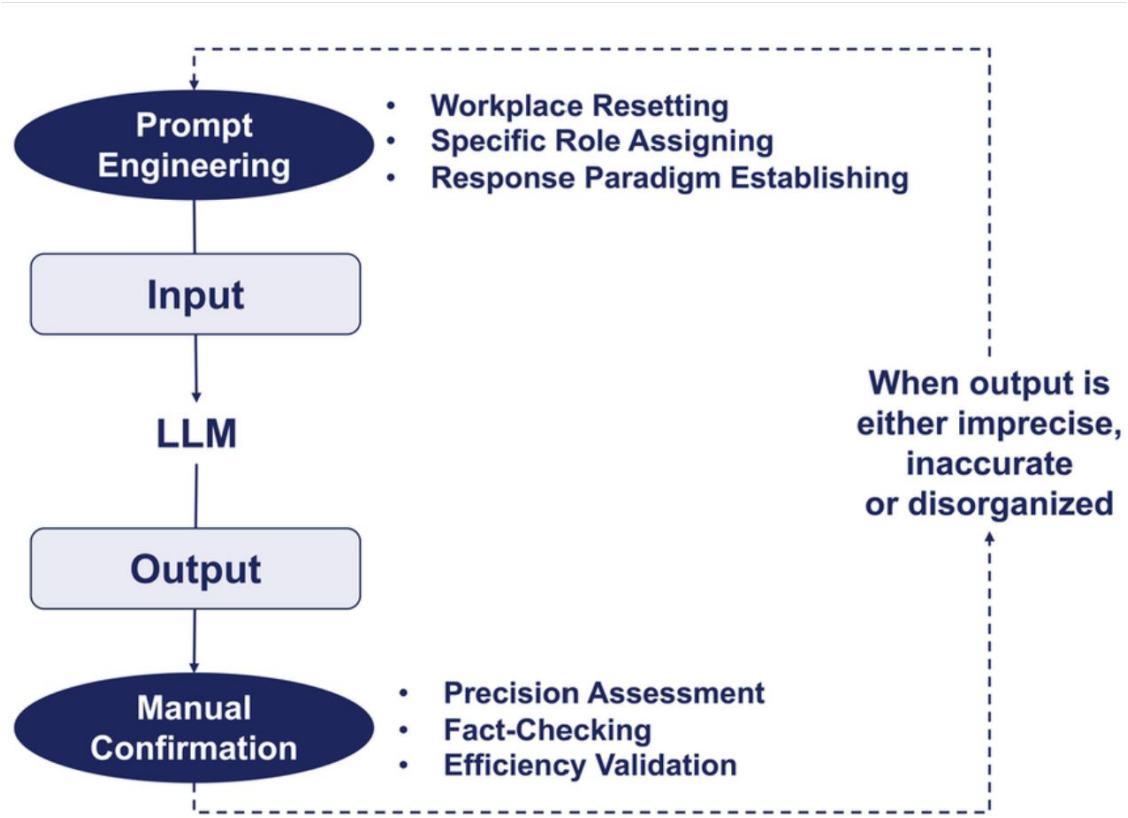

Figure 3.4: Prompt-based interaction flow in large language models, illustrating prompt engineering, model processing, and output validation. Source: Adapted from Shu et al. (2023), "Human-in-the-loop: Human involvement in enhancing medical inquiry performance in large language models", Allergy. https://doi.org/10.1111/all.15976.

3.5 Limitations of LLMs

Despite the high ability of large language models to generate and process text, their use comes with important limitations that must be carefully considered in practical applications, especially in academic environments and writing assistance systems. One of the main limitations is hallucination, that is, the case where the model produces fluent and apparently convincing text, but some of its content is factually incorrect, fabricated, or unverified. Recent reviews also

emphasize that hallucination is one of the fundamental challenges of LLMs and directly affects the reliability of their output [23], [25]. An example of hallucination in large language models is illustrated in Figure 3.5.

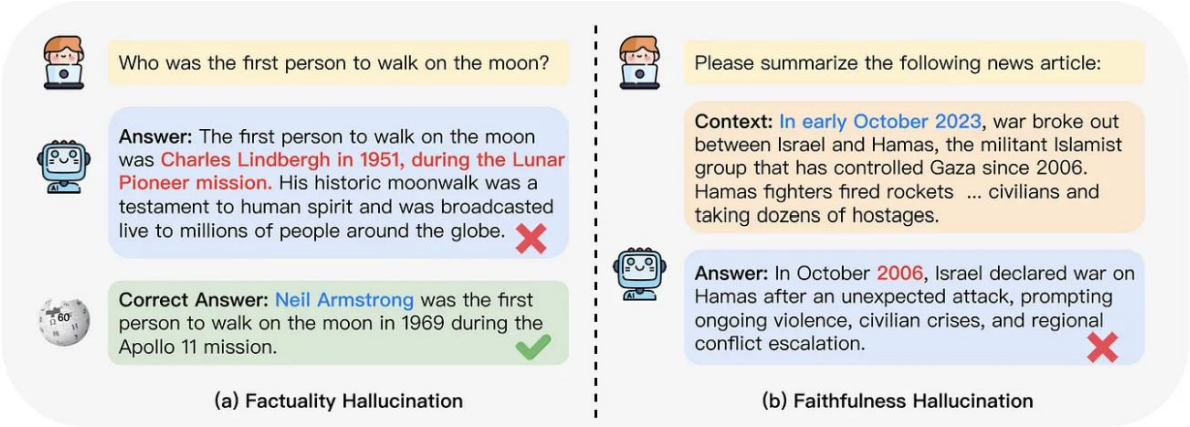

Figure 3.5: Example of LLM hallucination, illustrating incorrect but fluent outputs compared to correct responses. Source: Novita AI, "All You Need to Know about the Limitations of Large Language Models", Medium, 2024. Available at: https://medium.com/@marketing_novita.ai/all-you-need-to-know-about-the-limitations-of-large-language-models-568e15f66809 (accessed April 16, 2026).

This issue is particularly important in writing assistance, because the user may assume that the response is accurate and valid because of its fluency. For example, a language model may rewrite a sentence grammatically well, but make errors in explaining the reason for the error, in providing citations, or in producing academic content. Recent studies on the use of LLMs in academic writing have also shown that these models can make mistakes in generating citations, DOIs, or accurate references, and therefore their output requires human review [26], [27].

Another important limitation is the instability and dependence on prompts. Large language models do not always produce completely deterministic responses, and small changes in the wording of the command, the order of information, or the level of detail of the prompt can lead to different outputs. This is a significant challenge in software systems that require consistent and repeatable behavior. For this reason, prompt design and the use of structured formats for output are an important part of the development of LLM-based systems [30], [32].

On the other hand, LLMs in many cases do not provide explicit and explainable understanding of the language. Although these models can reproduce complex language patterns with high accuracy, this ability does not necessarily translate into human reasoning,

stable conceptual understanding, or transparent explainability. For this reason, in applications such as training, writing assessment, or language feedback, the model output should not be considered equivalent to definitive judgment or human understanding [25], [33].

Another important limitation is security and privacy considerations. Since many LLM applications are based on API services or cloud systems, the user input text may contain sensitive personal, academic, or organizational information. Recent reviews have emphasized risks such as data leakage, educational data mining, membership inference, and misuse of sensitive inputs. As a result, the use of LLMs in real-world environments requires serious attention to data governance, privacy protection, and data retention policies [34], [35]. OpenAI's official documentation also states that data sent to the API is not used to train models by default unless the user explicitly opts in, which shows that data handling is an important component in the design of API-based systems [32].

In addition, many LLM-based systems depend on external services. This dependence can be accompanied by considerations such as latency, usage costs, changes in model versions, API limitations, and the need for a stable Internet connection. In a research prototype, this dependence may be acceptable, but in practical and long-term applications, it should be considered as one of the important constraints of the system architecture [23], [32].

In the context of this thesis, these constraints directly affect the system design. On the one hand, it is necessary to receive the model output in a structured and controlled manner to reduce the instability of the response format. On the other hand, the suggestions generated by the model should not be considered as definitive truths, but rather as auxiliary suggestions that the user can review, reject, or apply. For this reason, in the present proposal system, LLM is not used as a complete replacement for human judgment, but as a linguistic core for providing intelligent and verifiable suggestions [27], [32].

3.6 Chapter Summary

This chapter reviews the theoretical foundations of large language models. First, the general concept of LLMs and their place in the recent evolution of natural language processing are introduced, and it is shown that these models are capable of performing a wide range of linguistic tasks by relying on large text data and deep architectures [23], [25].

Next, the Transformer architecture is explained as the main foundation of many large language models. In this section, the role of self-attention, encoder and decoder, multi-head

attention, and positional encoding in modeling context and efficient processing of linguistic sequences is examined [3], [23]. Then, the application of LLMs in writing assistance systems is reviewed, and it is shown that these models can play an important role in correcting grammatical errors, rewriting sentences, improving writing style, and producing simpler or more academic versions of text [26], [27], [28].

The next part of the chapter is devoted to prompt-based interaction. In this section, it was explained that the behavior of language models depends largely on how the prompt is designed and that the quality of their output is not limited to the model itself. Also, the importance of prompt engineering and the use of structured outputs to increase the reliability and software processing of the model output was discussed [30], [32].

Finally, the most important limitations of LLMs were reviewed. In this section, issues such as hallucination, response instability, prompt sensitivity, explainability limitations, security and privacy concerns, and dependency on external services were raised. These limitations show that although LLMs are powerful tools for writing assistance, their use should be accompanied by careful design, appropriate output control, and the presence of the user in the decision-making loop [25], [33], [34], [35].

In the context of this thesis, the discussions in this chapter are important because the large language model forms the linguistic core of the proposed system. The system in question uses LLM to correct grammar, generate rewriting suggestions, and provide simpler and more formal versions of sentences. Therefore, understanding the capabilities and limitations of LLMs is essential to better understand the design logic of this system. Accordingly, in the next chapter, the focus will shift from the theoretical foundations of eye tracking and LLMs to related research to clarify the position of this thesis among previous studies and the existing research gap [5], [23], [27].

Chapter 4 – Related Work

4.1 AI Writing Assistants

AI writing assistants have become one of the most important branches of language technology applications in recent years. These tools are typically designed to help users correct grammatical errors, improve writing style, rewrite text, and increase clarity or tone. In their early forms, many of these systems relied on relatively fixed linguistic rules and patterns, but newer versions increasingly use machine learning models, especially large language models, to provide more contextual and flexible suggestions [27], [42].

One of the best-known commercial examples in this area is Grammarly. In addition to spelling and grammar correction, this tool also offers features such as tone adjustment, clarity suggestions, paragraph rewrites, and sentence rewrites. According to the official description of Grammarly, the system can improve the clarity and flow of the text and suggest rewrites that match the tone and formality desired by the user [36], [37], [38]. Such features indicate that new writing assistants have gone beyond the level of "pointing out errors" and have moved towards rewriting and text refinement [36], [38].

Another important example is LanguageTool, which is known as a multilingual writing assistant. According to the official information of this tool, in addition to grammar, spelling, and punctuation checking, LanguageTool also provides suggestions related to style and word choice and supports more than 30 languages [39], [40]. This feature has made LanguageTool attractive to users who write in multilingual environments or beyond English, in addition to general applications [39], [41].

In addition to these commercial tools, the research literature has also been focusing on the field of Automated Writing Evaluation (AWE) and writing support for many years. Recent systematic reviews show that AWE systems are commonly used to provide feedback on areas such as grammar, vocabulary, organization, coherence, and mechanics, and have found an important place in second language teaching and writing skill support [42]. However, many of these systems, especially the older generations, have relied more on analytical and rule-based feedback and have been limited in providing contextual rewriting or flexible language interaction [42], [43].

The entry of LLMs into this field has led to the development of writing assistants from purely error-detection tools to systems that can also play the role of co-writer or revision

assistant. Recent research on writing with AI shows that these tools now cover not only grammar correction, but also rewriting, rephrasing, argument refinement, and style adaptation [26], [27], [45]. As a result, the line between traditional grammar checkers and intelligent writing assistants is becoming increasingly blurred [27], [45].

More specialized tools have also emerged in the field of academic writing. For example, recent reviews of Trinka and other intelligent writing evaluation systems show that some tools are designed specifically for academic writing and provide feedback beyond grammar correction, such as context-aware revision and feedback tailored to research texts [44]. This trend suggests that writing assistants are gradually becoming more specialized in specific domains, including academic writing [26], [44].

Despite these advances, existing tools still typically rely on input text and do not directly know the user's actual behavior when reading, hesitating, or revising text. In other words, most existing writing assistants react when text is presented to them, not when the user pays attention to a specific part or pauses on a sentence during the interaction with the text [36], [39], [42]. From this perspective, it can be said that most existing systems do not use user behavioral states, such as gaze behavior, in their core writing assistance logic [27], [42].

In the context of this thesis, the review of AI writing assistant tools is important because it shows that the evolution of these systems has moved from simple grammar checking to context-aware rewriting and style assistance [27], [36], [42]. However, an important gap remains: most of these tools lack a signal of user attention that can determine when and on which part of the text to intervene [27], [39]. Therefore, although existing writing assistants provide an important foundation for the present research, the system proposed in this thesis attempts to go beyond text-only assistance by adding gaze-triggered interaction [27], [45]. The evolution of AI writing assistants is illustrated in Figure 4.1, showing the transition from basic grammar checking tools to modern AI-driven writing systems.

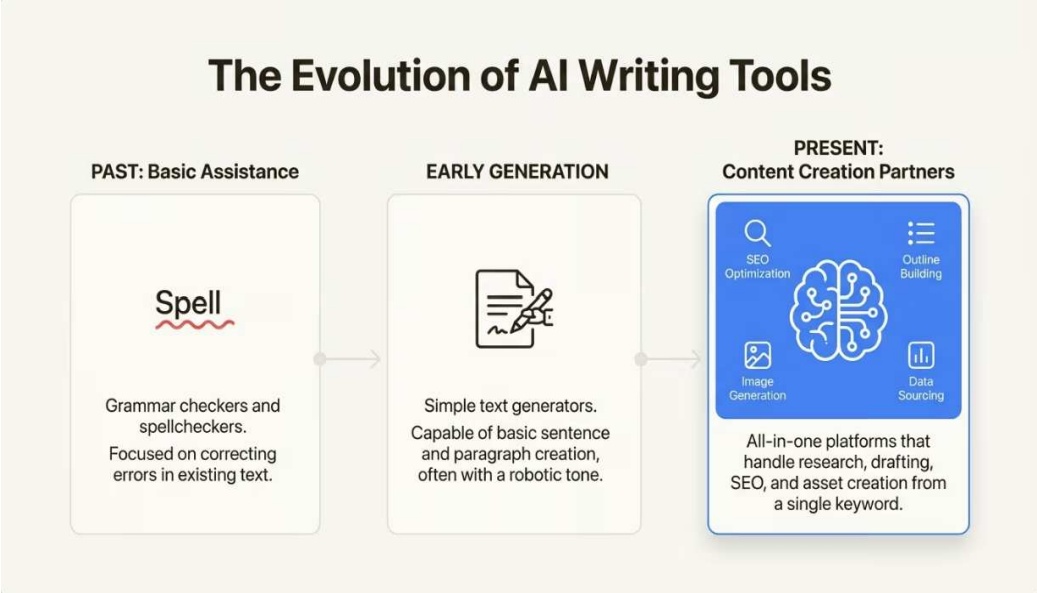

Figure 4.1: Evolution of AI writing assistants from basic grammar checking systems to advanced AI-based content generation tools. Source: eesel.ai, "I tested the 6 best AI writing tools in 2026 (here's my verdict)", 2026. Available at: https://www.eesel.ai/blog/ai-writing (accessed April 16, 2026).

4.2 Eye-Tracking Interaction Systems

Eye-tracking interaction systems are a group of interactive interfaces in which user gaze data is used not only to analyze behavior, but also to control, select, guide, or activate system functions. This area of research lies at the intersection of eye tracking, human-computer interaction, and assistive technologies, and has been considered an important branch of gaze-based interaction for many years [5], [15].

In many early studies, the main goal of these systems was to demonstrate that the user's gaze could act as an independent or complementary input modality. Within this framework, researchers attempted to design mechanisms for selecting objects, controlling the cursor, activating commands, or navigating the interface that use gaze instead of relying entirely on the mouse and keyboard [5], [22]. These studies have formed the theoretical and practical foundation for many subsequent designs in gaze-based HCI.

One of the most well-known approaches in this area is the use of dwell-based selection. In this method, the user selects a target by holding his gaze on it for a certain period of time. Such an approach is particularly useful in assistive systems, hands-free environments, and interfaces where physical clicking is difficult or undesirable [17], [18]. However, as has been repeatedly emphasized in the literature, the effectiveness of this method strongly depends on

the choice of dwell time threshold, the size of the targets, and the quality of the gaze data [15], [18].

A significant part of the research related to eye-tracking interaction systems has focused on assistive interfaces. In this category of systems, gaze plays the role of a vital control channel for users who cannot use traditional inputs due to movement limitations. Recent reviews show that gaze-based control in assistive robots, communication systems, and accessible interfaces is still an active area of research, with issues such as accuracy, reliability, and cognitive load remaining challenging [16].

In the following years, research has moved beyond simple object selection to gaze-aware interfaces; that is, interfaces that not only receive direct commands from the gaze, but also use gaze to interpret attention, predict intent, and adapt the system's behavior to the user's visual focus [15]. In such systems, gaze can be used to highlight information, display relevant content, or change the system's behavior at the appropriate time, even if the user has not directly given an explicit command.

Related research has also shown that in practice, many successful eye-tracking interaction systems are designed to be multimodal. In other words, gaze is usually used alongside other modalities such as mouse, keyboard, voice, or gesture, and not necessarily as the sole interaction channel [15], [22]. This approach is important because human gaze is inherently both a perceptual tool and can act as a signal of intent; therefore, combining it with other inputs often helps to reduce errors, increase robustness, and improve usability [5], [15].

In the context of writing and text, a large part of eye-tracking interaction systems to date have focused on text entry, gaze typing, or reading behavior analysis [17], [22]. In contrast, systems that use gaze to activate intelligent writing assistance or guide linguistic assistance are still relatively limited in the literature. This is important for the present study, because the proposed system precisely seeks to use the user's gaze not for direct typing, but to detect the appropriate time and place for linguistic intervention.

As a result, a review of eye-tracking interaction systems shows that the field has moved from simple selective and assistive interfaces towards more adaptive and attention-aware interfaces [15], [16], [22]. However, there is still a gap between traditional gaze-based systems and systems that combine gaze with language models and writing assistance. This gap is one of the main reasons for the importance of the present study, because the proposed system

attempts to integrate gaze as a signal of attention into the logic of an LLM-based writing assistant [5], [15].

4.3 Eye Tracking and AI-Based Systems

In recent years, some research has moved beyond traditional gaze-based systems to combine eye tracking with artificial intelligence methods. In this approach, gaze data is not only used to directly control or record user behavior, but also as a behavioral signal for interpreting intent, modeling user state, and intelligently adapting system behavior. Recent reviews in HCI and multimodal interaction also show that combining gaze data with machine learning models, adaptive systems, and context-aware approaches has become an active research focus [15], [46], [47].

One important direction in this area is the development of gaze-aware intelligent interfaces; that is, interfaces that use the user's gaze to detect visual focus, predict user needs, or provide timely support. In these systems, gaze is not simply a direct command, but rather a piece of behavioral evidence that helps the system understand what the user is paying attention to and what kind of intervention might be helpful [15], [48]. Such a perspective has become particularly important in adaptive learning systems, assistive interfaces, and context-sensitive interactive environments.

In some studies, eye tracking data has been combined with other modalities such as gesture, voice, or manual inputs to enable multimodal decision making. For example, studies such as EyeTAP have shown that combining gaze with a complementary signal can reduce the Midas Touch problem and improve selection accuracy compared to purely gaze-based interfaces [49]. Similarly, multimodal HCI reviews also emphasize that gaze is often more effective when combined with other modalities than when used alone, both because it better reveals interaction intent and reduces the error burden due to gaze oscillation [15], [46].

Another body of literature has combined eye tracking with machine learning and artificial intelligence algorithms to analyze user state, cognitive behavior, or interaction patterns. In these studies, gaze can be used as input for engagement detection, cognitive state, visual intent, or user behavior modeling. More recent research has even gone so far as to incorporate eye data into human–AI collaboration frameworks and multimodal behavior pattern analysis so that the system can obtain a richer interpretation of user behavior [46], [50].

In the area of language technologies, the number of studies that directly integrate eye tracking with language models or intelligent language systems is still relatively limited, but this direction is rapidly expanding. For example, some studies have used LLMs to interpret reading data, generate adaptive summaries, or improve gaze-based communication systems [51], [52]. More recent work in NLP also shows that gaze signals can be used as valuable behavioral cues in some language tasks and even help improve the alignment of the system with the user's mental process in human-LLM interactions [53]. Despite these advances, most existing systems still fall into one of two categories: either systems that use gaze for interaction control, or systems that use AI to analyze and interpret behavior. The direct integration of gaze with LLM-based writing assistance, especially at the level of sentence-level attention and fixation-triggered linguistic support, has not yet been widely established in the literature [48], [51]. This fact places the present study at the intersection of two lines of research: gaze-aware interaction on the one hand, and LLM-based writing assistance on the other.

In the context of this thesis, the importance of this type of research is that it shows that gaze can be used as a behavioral signal for intelligent system guidance, not simply as a selection tool or a posteriori analysis. The proposed system in this study is based on exactly this idea: using the user's fixation and visual attention to activate linguistic analysis at the right time and place. Therefore, the present work can be considered a step towards bringing eye tracking, interaction intelligence, and LLM-based writing support closer together [15], [46], [51].

4.4 Research Gap

A review of the literature shows that three major research streams have developed relatively independently: first, AI writing assistants that focus on text correction, rewriting, and style improvement; second, eye-tracking interaction systems that use gaze to control the interface, analyze attention, or support assisted interactions; and third, AI-based gaze-aware interfaces that combine eye data with intelligent methods to interpret user behavior [15], [42], [46].

In the first stream, new writing assistant tools, especially those based on LLM, have been able to go beyond traditional grammar checking and provide capabilities such as sentence rewriting, tone adaptation, and academic support [26], [27], [36]. However, most of these systems still rely on text-only input; That is, they decide when and how to intervene based solely on the text itself, not on the user's momentary behavior while reading or reviewing it [27], [36], [39].

In the second stream, eye-tracking interaction systems have shown that the user's gaze can be used as an input modality or signal of attention for selection, control, and assistive interaction [5], [17], [22]. These studies have clearly demonstrated that gaze can play an effective role in assistive interfaces, hands-free systems, and multimodal environments [15], [16]. However, the majority of these systems have focused more on interaction control than on language support or writing assistance [17], [22].

In the third stream, more recent research has moved towards combining eye tracking with AI and multimodal intelligence, showing that gaze can be used in intent interpretation, interface adaptation, or user behavior modeling [46], [50]. Some emerging studies have also investigated the combination of gaze data with LLMs in areas such as adaptive learning, communication support, and human–AI collaboration [51], [52], [53]. However, this work is still limited and has usually either focused on tasks other than writing assistance, or has used gaze for general behavior analysis rather than for activating sentence-level linguistic support [51], [53].

Consequently, the main gap in the literature can be summarized as the fact that there are still few systems that use the user's visual behavior as a meaningful signal to determine when and where to intervene in language [48], [51]. In other words, although writing assistants have become more linguistically intelligent and gaze-based systems have improved in terms of interactivity, the direct combination of the two in the form of a gaze-triggered writing assistant that relies on the user's attention and fixation to invoke linguistic analysis has not yet been widely established in the literature [27], [48], [53].

The second gap concerns the level of granularity of interaction. Many existing systems either provide feedback at the general level of the text, or interpret gaze at the level of object selection and interface control [17], [22], [36]. In contrast, the present work is based on the assumption that sentence-level attention can be a suitable level for activating writing support, because it is neither too general to produce irrelevant feedback, nor too fine-grained to rely on error-prone and unstable word-level selection [15], [48].

The third gap concerns the timing of intervention. In conventional writing assistance tools, feedback is usually either constantly active or provided upon direct user request [36], [39]. In contrast, in the system proposed in this thesis, it is assumed that the user's fixation on a sentence can be a signal of the need for revision and therefore the timing of linguistic

intervention can be determined based on the user's behavior and not just fixed rules of the interface [5], [15], [48].

Accordingly, the present study attempts to bridge the gap between LLM-based writing assistance and gaze-aware interaction. Its main innovation is that it uses eye tracking not for direct typing nor for a posteriori analysis of behavior, but for momentary and contextual guidance of writing assistance. Specifically, the proposed system uses sentence fixation as a signal to activate correction and style assistance, thereby creating a kind of interaction between visual attention and language support that is still underdeveloped in the existing literature [27], [48], [51]. The research gap is illustrated in Figure 4.2, where the proposed system is positioned at the intersection of three previously independent research streams.

Figure 4.2: Conceptual illustration of the research gap, showing the three main research streams: AI writing assistants, eye-tracking interaction systems, and AI-based gaze-aware interfaces. The proposed thesis is positioned at the intersection, introducing a gaze-triggered writing assistant based on sentence-level attention and fixation.

4.5 Chapter Summary

In this chapter, related research is reviewed in three main areas: AI writing assistants, eye-tracking interaction systems, and hybrid systems based on eye tracking and artificial intelligence. The first section shows that writing assistants have moved from traditional grammar checkers to more advanced systems that provide not only error correction but also rewriting, tone adjustment, and support for academic writing [26], [27], [36], [39]. However, most of these tools still rely on the text itself and do not take into account the user's momentary behavior while reading or reviewing the text [27], [42].

The second section reviews eye-tracking systems in the field of human-computer interaction. These studies have shown that gaze can be used both as an input modality and as a signal of attention in interactive interfaces, assistive systems, and multimodal environments [5], [15], [22]. It was also observed that many of these systems focused on selection, control, and assistive interaction, and less on language support or writing assistance [16], [17], [18].

In the third section, research was reviewed that combines eye tracking with AI, multimodal interaction, or intelligent systems. These studies have shown that gaze can be used for intent modeling, user behavior analysis, and the creation of adaptive and context-aware interfaces [46], [50]. Also, some more recent work has moved towards combining gaze with LLMs and language applications, but this path is still in its relatively early stages and is not fully developed, especially in the field of writing assistance [48], [51], [53].

Based on this review, the main research gap was identified as the fact that there are still few systems that use the user's visual behavior as a signal to determine when and where to intervene in language [48], [53]. In particular, the direct combination of sentence-level attention detection with LLM-based writing assistance is still limited in the literature [27], [48], [51]. This gap is precisely where the present study addresses.

As a result, this chapter clarifies the position of the proposed thesis among previous studies and shows that the present work is located at the intersection of three important areas: eye tracking, gaze-aware interaction, and large-scale language model-based assisted writing [15], [27], [48]. Accordingly, in the next chapter, the proposed system of this research and its overall architecture will be introduced in more detail.

Chapter 5 – Proposed System

5.1 System Overview

The proposed system in this study is a gaze-based intelligent writing assistant that combines eye-tracking technology and large-scale language models [15], [23], [27]. The main goal of this system is to transform the process of revising and improving writing from a purely textual process to an interactive and attention-aware process [5], [15], [48]. In this approach, the system does not rely only on the text itself, but also uses the user's visual behavior to identify which parts of the text are likely to require attention, correction, or rewriting [5], [48].

The central idea of the system is based on the assumption that the user's gaze can be a meaningful signal of attention and the possible need for linguistic intervention [5], [15], [19]. In other words, if a user pauses on a sentence while reading or writing and their gaze remains fixed on it for a certain period of time, this behavior can be a sign of hesitation, need for revision, or difficulty in processing that sentence [11], [15], [17]. The proposed system uses this behavioral signal to activate linguistic analysis at the appropriate time and provide relevant writing suggestions [27], [48], [53].

In terms of overall functionality, the system consists of several main components. First, there is a text-based user interface where the user writes or revises their sentences. Second, a gaze tracking module receives gaze data from the eye tracker and converts them into usable coordinates in the interface environment. Third, there is a gaze interpretation layer that determines the user's area of interest, the sentence under gaze, and the fixation state. Fourth, an LLM-based language module is responsible for grammatical analysis of the sentence, generation of correction suggestions, and presentation of rewritten versions. Finally, an interaction logging system is provided to store events and gaze data to analyze user behavior and system performance.

The proposed system has two main types of language functions. In the first case, if the sentence of interest to the user contains grammatical or lexical errors, the system presents a corrected version of the sentence and identifies problematic parts of the text for the user [27], [28]. In the second case, if the sentence is grammatically correct, the system can provide stylistic suggestions instead of correcting the error; for example, producing a simpler version or a more formal version of the same sentence [26], [28], [29]. These two levels of functionality

make the system not just a grammar checker, but also a multi-layered writing assistant [26], [27].

One of the important features of this system is fixation-based activation [11], [15], [17]. In many common tools, text checking is either done continuously or the user has to explicitly press a button to receive feedback [36], [39]. In contrast, in the present system, the main logic is based on the fact that the stability of the gaze on a sentence can be considered as a condition for starting linguistic analysis [15], [48], [53]. This feature allows the system to intervene at times when the user's attention is really focused on a part of the text, avoiding unnecessary reactions or continuous and annoying interventions [17], [19].

In addition to this gaze-based logic, the system also benefits from an interactive design [15]. This means that the output of the language model is not directly and unchangeably applied to the text, but is provided to the user in the form of a suggestion. The user can accept the suggestion, ignore it, or decide to edit the text manually. This shows that the role of the system in this research is not to completely replace the writer, but to intelligently support the writing and revision process [26], [27], [33].

From an architectural perspective, the proposed system can be considered a multilayered and event-driven system. The first layer receives gaze data, the second layer interprets them with sentence-level attention, the third layer performs linguistic analysis via LLM, and the fourth layer displays the results in the form of a user interface and actionable suggestions to the user. This layered organization makes the system modular in design and each part can be developed, modified, or evaluated more independently.

Overall, the proposed system in this thesis is an attempt to link the user's visual attention with an AI-based writing assistant [15], [23], [48]. Its main innovation is that instead of relying on text alone, it uses the user's gaze as a signal to determine the appropriate time and place for linguistic intervention [5], [15], [53]. For this reason, this system can be placed at the intersection of eye tracking, gaze-aware interaction, and LLM-based writing assistance [27], [48], [53].

5.2 Design Goals

The design of the proposed system in this study is based on a set of specific goals that arise from both the limitations of gaze-based systems and the capabilities and challenges of LLM-based writing assistants [15], [19], [23]. These goals are chosen so that the system can, on the

one hand, make meaningful use of the user's gaze data and, on the other hand, provide the user with linguistic feedback in an understandable, unobtrusive, and controllable manner [5], [15], [27].

The first design goal is to make the system aware of the user's attention. In many conventional writing assistance tools, the system's intervention is based solely on the presence of text or general interface rules and is not necessarily coordinated with the user's current attentional state [27], [36], [39]. In contrast, in this research, the goal is for the system to be able to use gaze and fixation to detect at what moment and on which sentence the user is focused, so that linguistic intervention can be performed in a more relevant and meaningful context [5], [15], [48].

The second goal is to reduce unnecessary interventions and avoid disturbing the user. One of the main challenges in gaze-based interaction is that every user's gaze should not be interpreted as a command or a request for help, as such an approach leads to unwanted activations and a poor user experience [15], [17], [18]. For this reason, in the design of the present system, an attempt has been made to activate linguistic analysis only when the user's gaze remains relatively stable on a sentence and there is more evidence of meaningful attention [11], [17], [19].

The third goal is to maintain the user's active role in the writing and revision process. Although LLMs can produce useful language suggestions, limitations such as hallucination, response instability, and the lack of full assurance of correctness or appropriateness of the output make them unsuitable for use without the user being involved in the decision-making process [23], [27], [33]. Accordingly, the proposed system is designed in such a way that the model output is presented as a suggestion and the final decision to accept, reject, or edit it is left to the user [26], [27].

The fourth goal is to provide language support at more than one level. In many traditional tools, the main focus is on correcting superficial errors such as spelling and grammar, while the real needs of users in written environments usually go beyond this level [27], [42]. Therefore, in this research, the goal is that in addition to grammar correction, the system can also provide suggestions related to rewriting, simplification, and formalization of sentences, so that its role can be upgraded from a simple grammar checker to a more comprehensive writing assistant [26], [28], [29].

The fifth goal is to resist fluctuations and uncertainty in gaze data. Since eye tracking data in practice is accompanied by noise, jitter, accuracy and precision limitations, and changes caused by user and environmental conditions, the proposed system should not rely on the instantaneous coordinates of the gaze in an absolute and direct manner [19], [20], [21]. For this reason, the system design should be based on relative gaze stability, temporal thresholds, and sentence-level attention interpretation to be more robust and avoid excessive sensitivity to small fluctuations [11], [15], [19].

The sixth goal is the ability to process the output of the language model in a structured and reliable manner. In systems where LLM forms part of the software pipeline, the output of the model should be produced in a specific and processable format as much as possible so that it can be stably introduced into the next layers of the system [30], [32]. Accordingly, one of the design goals in this research is to interact with the language model not in a free and uncontrolled manner, but with purposeful prompts and structured outputs so that the application of suggestions and the display of feedback in the user interface can be done more reliably [23], [30], [32].

The seventh goal is the modularity of the system architecture. Given that the proposed system includes several different components, including gaze tracking, fixation interpretation, linguistic analysis, user interface, and interaction recording, the modular design allows each part to be developed more independently, modified, or replaced by other methods in the future [15], [23]. Such an approach is not only technically advantageous, but also important for research evaluation, as it allows the performance of different parts of the system to be analyzed separately [5], [22].

Finally, the eighth goal is to support user behavior analysis and system evaluation. Since this research is not limited to producing a prototype but also seeks to better understand the interaction between gaze and writing assistance, it is necessary for the system to be able to record data related to gaze, fixation, and interaction events [5], [22]. This design goal allows the system to not only assist the user at runtime, but also provide data for subsequent analysis of user behavior and performance of various system components [15], [27].

Overall, the design goals of the proposed system can be summarized in four general axes: awareness of user attention, minimizing intrusiveness, maintaining human control over suggestions, and providing reliable, multi-layered language support [15], [23], [27]. These goals form the basis for the architectural and implementation decisions of the system and will

be described in more detail in the following sections of this chapter in terms of the architecture, data flow, and interaction logic of the system [5], [15], [48]. The main design goals of the proposed system are summarized in Figure 5.1, highlighting the key principles guiding the system design.

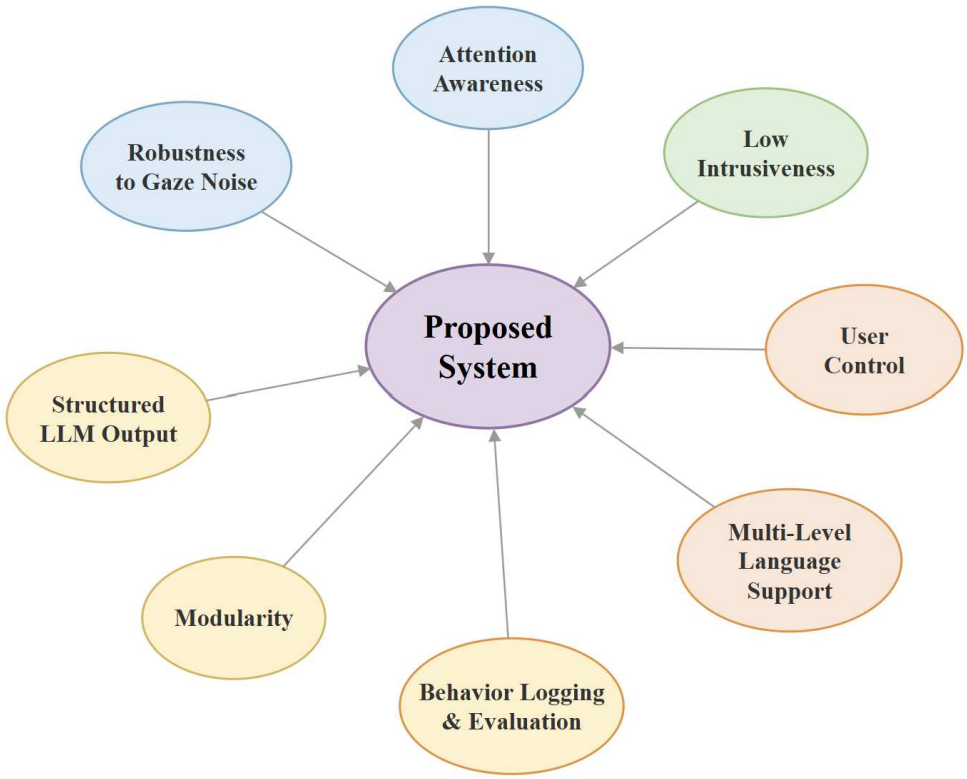

Figure 5.1: Main design goals of the proposed system.

5.3 General Architecture of the Proposed System

The overall architecture of the proposed system is designed to combine user gaze data, fixation-based interaction logic, and LLM-based linguistic analysis into a unified framework [5], [15], [23]. This architecture can be described as a multi-layer, modular, and event-driven system, where each layer is responsible for a specific part of the interaction process and the output of each part serves as the input of the next part [15], [22].

At a general level, the system workflow begins with the user interacting with the text interface. The user enters or revises his text in the editor environment, while his gaze data is recorded by the eye tracker. After being converted into usable coordinates in the interface

environment, this data is transferred to the gaze interpretation layer to determine which area of the text the user's gaze is on and whether their gaze pattern has reached a level of stability that can be considered a meaningful fixation [4], [11], [19].

The first layer of the architecture is the data acquisition layer. This part is responsible for receiving raw gaze data from the eye tracker or input simulation mechanism and converting them into a format usable by the system. In this layer, the data validity status, gaze coordinates, and basic information necessary for synchronization with the user interface are managed [4], [22]. Since gaze data is in practice accompanied by fluctuations, jitter, and measurement errors, this layer plays an important role in preparing the data for subsequent processing [19], [20].

The second layer is the visual attention interpretation layer. In this section, the system transforms gaze position data into more meaningful structures at the interface level; for example, it determines which part of the editor the user's gaze is on, which sentence is under their gaze, and whether gaze stability has reached a level that can be considered fixation [11], [15], [17]. This layer is essentially a bridge between the raw eye tracking data and the interaction logic of the system, as it determines whether the user's visual behavior should be interpreted as transient attention or as a signal that linguistic intervention is needed [5], [15].

The third layer is the interaction logic layer, which decides whether or not to activate linguistic review based on sentence-level attention information. In this section, the system considers not only the instantaneous gaze coordinates, but also the gaze stability over time, the current state of the interaction, and the prevention of repeated or unnecessary activations [15], [17], [19]. In this way, linguistic analysis is only invoked when the conditions for a meaningful intervention are met and the system can help the user with minimal intrusion [15], [48].

The fourth layer is the linguistic analysis layer based on LLM. In this section, the sentence of interest from the user is sent to the linguistic model to be checked for grammar, wording, and style. Depending on the content of the sentence, the output of this layer can include a modified version of the sentence, a list of linguistic problems, a short explanation, as well as simpler or more formal rewritten versions [23], [27], [28]. Since this output has to be processed in a software environment, interaction with the linguistic model in this layer is defined based on purposeful prompts and structured outputs [30], [32].

The fifth layer is the presentation and user interaction layer. In this layer, the results of the linguistic analysis are displayed as visual feedback in the user interface. For example, the system can highlight problematic parts of the sentence, show the revised version in a side panel,

or provide the user with selectable rewriting suggestions. At this stage, the system does not play the role of the final decision-maker, but rather presents its output as a suggestion that the user can accept, reject, or independently edit the text [26], [27], [33]. This feature is in line with the human-in-the-loop approach in AI-based systems [27], [33].

In addition to these layers, the system also has an interaction recording and monitoring module. This section stores data related to gaze, fixation, interaction status, and important system events so that they can be used to analyze user behavior, evaluate system performance, and examine interaction patterns [5], [22]. The existence of this layer allows the system architecture to be not limited to online execution only, but also to enable a posteriori analysis and research interpretation of system and user behavior [15], [27].

One of the important features of the present architecture is the clear separation between sensing, interpretation, language processing, and presentation. This separation allows each part to be developed relatively independently and replaced with other methods or tools if needed [15], [23]. For example, in the future, the eye tracker used can be changed, the fixation detection algorithm can be improved, or another language model can be substituted, without the need for a complete redesign of the entire system. This modularity is an important advantage from both a research and implementation perspective [22], [23].

Overall, the overall architecture of the proposed system shows that the present work is not just a direct connection between eye tracker and LLM, but a multi-layered interaction framework in which user visual attention, intervention logic, and linguistic analysis are purposefully integrated [15], [23], [48]. This architecture provides the necessary context for a more detailed description of the system components in the following sections of this chapter. The general architecture of the proposed system is illustrated in Figure 5.2.

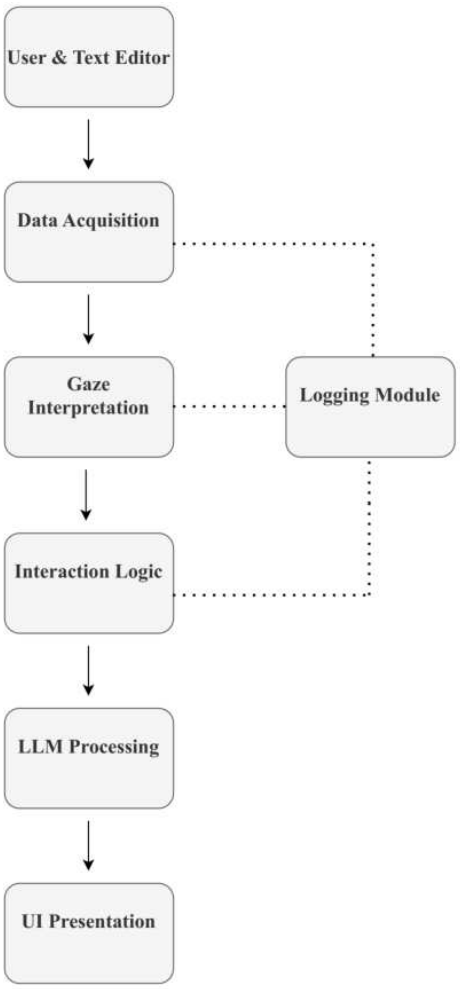

Figure 5.2: General architecture of the proposed gaze-based writing assistant, showing the main components including data acquisition, gaze interpretation, interaction logic, LLM processing, user interface presentation, and the logging module.

5.4 Main Components of the System

The proposed system consists of several main components, each of which plays a specific role in the process of receiving gaze data, interpreting user attention, linguistic analysis, and providing feedback [5], [15], [23]. Separating these components from each other makes the system architecture both conceptually clearer and more modular in terms of implementation and evaluation [15], [22]. In this study, the main components of the system include the user interface, gaze tracking module, gaze interpretation module, language model module, and interaction recording module [23], [48].

5.4.1 User Interface Module

The user interface is the main point of user interaction with the system and provides a platform where writing, reviewing text, viewing suggestions, and receiving system feedback are performed. In assisted writing systems, the user interface is not just a simple display layer, but also plays an important role in the comprehensibility of feedback, avoiding distractions, and maintaining user control over the editing process [27]. For this reason, in the present system, the user interface is designed to be both suitable for text entry and revision and to support gaze-based interaction alongside linguistic feedback [15], [48].

This module includes a text editing environment, an area for displaying suggestions and explanations, and auxiliary elements for controlling the state of the system. In this environment, the user can write their sentence, view correction or style suggestions, and apply or reject them if desired. Such an approach is compatible with the human-in-the-loop perspective in AI-based systems, since the system plays a supporting role and the final decision remains with the user [26], [27], [33]. The user interface of the proposed gaze-based writing assistant is illustrated in Figure 5.3.

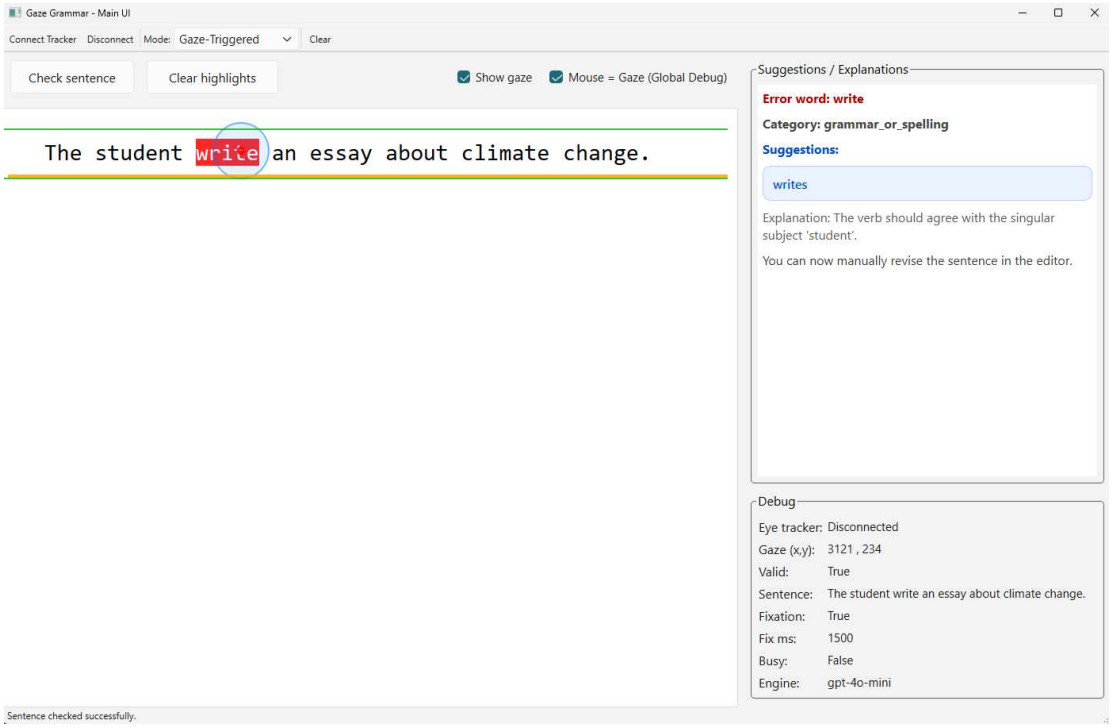

Figure 5.3: User interface of the proposed system, showing gaze-based interaction with a sentence, error detection, and linguistic suggestions provided by the language model.

5.4.2 Eye Tracking Module

The gaze tracking module is responsible for receiving gaze data from the eye tracker and preparing it for use in the system. This module is effectively an interface between the eye tracking hardware and other software components, and is responsible for transmitting gaze coordinates, data validity status, and other information related to eye movement to the system [4], [22]. In gaze-based applications, this layer is of great importance, because any errors or instabilities in the input data can lead to misinterpretation of the user's attention in subsequent layers [19], [20].

In the present system, this module is designed to receive gaze data online so that the system can use the user's visual behavior in real time for decision-making. Given that eye tracking data is usually associated with noise, jitter, and limitations in accuracy and precision, this component does not only act as a raw input, but also provides the basis for the entire gaze-aware logic of the system [19], [20], [21]. The eye-tracking setup used in this study is shown in Figure 5.4.

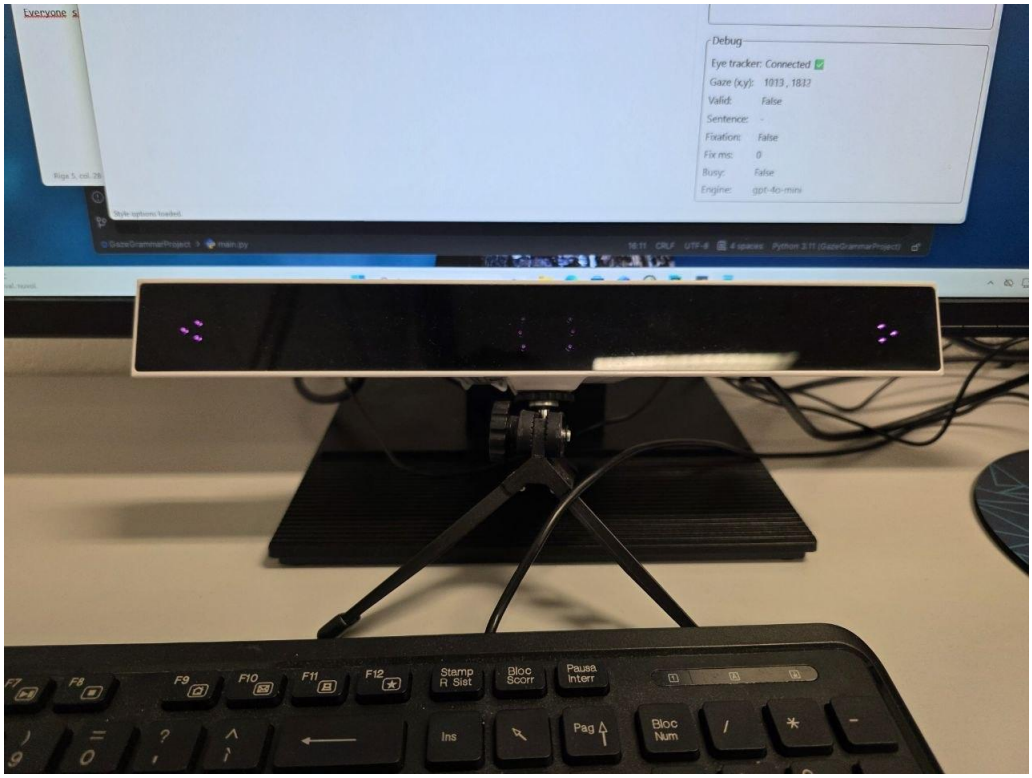

Figure 5.4: Eye tracker used in this study, mounted below the monitor during the experiment.

5.4.3 Gaze Interpretation Module

After receiving the raw gaze data, this information must be transformed into meaningful structures at the interface and text level. This task is the responsibility of the gaze interpretation module. In this module, the system determines which part of the interface the user's gaze is on, which sentence is under his gaze, and whether the gaze stability has reached a level that can be considered a meaningful fixation [11], [15], [17].

The importance of this module is that raw eye tracking data alone is not enough for a writing assistant. What is important for the system is not just the gaze coordinates, but also the sentence-level attention of the user and the appropriate time for intervention [5], [15], [48]. For this reason, this component plays a key role in converting gaze into an interaction signal and is in fact the connecting link between the user's visual behavior and the system's linguistic behavior [15], [53].

5.4.4 Language Model Module

The language model module is responsible for performing the processing related to grammar correction, sentence rewriting, and style assistance. In this section, the sentence of interest of the user is sent to the language model and outputs such as a corrected version, alternative suggestions, a short explanation, or simpler and more academic versions are produced [23], [27], [28]. Since LLMs in writing assistance can operate both at the error correction level and at the text rewriting and refinement level, this component forms the linguistic core of the proposed system [26], [28], [29].

Given the well-known limitations of LLMs, including prompt sensitivity and response format instability, interaction with this module must be done in a controlled manner [30], [32], [33]. For this reason, in the system architecture, this component is not just an open source of text generation, but also part of a structured pipeline whose output must be displayable, interpretable, and applicable to the user interface [23], [30], [32].

5.4.5 Logging and Interaction Recording Module

The interaction recording module is designed to store gaze data, important system events, and some elements related to the user's behavior during the interaction session. In research related to HCI and gaze-based systems, interaction logging is of great importance, as it allows user

behavior and system response to be analyzed later and the quality of interaction to be examined from a research perspective [5], [22].

In the present system, in addition to supporting prototype evaluation, this component helps to better understand intervention times, fixation patterns, and how the user responds to system suggestions. Therefore, logging in this research is not just a technical side tool, but is part of the research design of the system that allows for the analysis of interaction flow and system behavior [15], [27].

5.4.6 Coordination Between Components

Although each of the above components has an independent task, the real efficiency of the system is achieved only when these parts work in coordination. The user interface provides the interaction environment, the eye tracking module receives gaze data, the gaze interpretation module converts it into sentence-level attention, the language module performs the analysis and rewriting, and the logging module records the entire process for later analysis. This multi-part coordination is what distinguishes the present system from a simple editor or a typical grammar checker [15], [23], [48].

In summary, the main components of the system are selected and organized in such a way that they can use the user's visual behavior as meaningful input for intelligent writing assistance [5], [15], [48]. This modular separation provides the basis for a more detailed description of the interaction flow and operational logic of the system in the later sections of the chapter.

5.5 Interaction Workflow

The interaction flow in the proposed system is designed to connect writing, text viewing, gaze tracking, linguistic analysis, and feedback in a continuous and coordinated process [15], [23], [48]. In this approach, the system not only analyzes the written text, but also uses the user's visual behavior to determine the appropriate time for linguistic intervention [5], [15]. For this reason, the system workflow can be considered a multi-stage, event-driven process in which each stage acts as a condition or prelude to the next stage [15], [22].

The first stage of this flow is the user's input or review of the text. The user writes his/her sentence or text in the system's editor and then views, reviews, or corrects the same text. In this stage, the user interface acts as the main interaction space, providing an

environment in which both the writing process and the receipt of linguistic feedback are possible [26], [27]. Unlike some traditional tools that perform text analysis continuously and independently of the user's behavior, in the present system the linguistic analysis is dependent on the user's visual behavior [36], [39].

The second stage is the continuous acquisition of gaze data from the eye tracker. In this stage, the system receives the user's gaze position in real time and converts it into coordinates usable in the interface environment [4], [22]. This data alone does not yet have a specific interactive meaning, but it provides the raw material on which the user's visual attention can be interpreted at the interface and text level [5], [19]. Given that gaze data may be associated with noise, jitter, and accuracy limitations, the system must interpret them with caution and in an appropriate temporal context [19], [20].

The third step is to detect the region of interest and determine the sentence under gaze. In this step, the system associates the user's gaze position with the textual structure of the interface and determines which sentence or section of text the user is currently focused on. This step is of fundamental importance for the present system, since the logic of language intervention is based not on the whole text, but on sentence-level attention [15], [48], [53]. Therefore, converting raw gaze into sentence-level attention is one of the key points in the system's workflow [5], [15].

The fourth step is to check the stability of gaze and detect meaningful fixation. In this step, the system assesses whether the user's gaze has passed over the sentence briefly or has remained on it for a sufficient period of time [11], [17]. Only if the gaze reaches a level of stability that can be considered a meaningful fixation or dwell does the system enter the linguistic analysis phase [15], [17], [18]. This part of the workflow is important to avoid unnecessary interventions and reduce the impact of the Midas Touch problem, since every momentary glance should not be interpreted as a request for help [15], [18].

The fifth step is to activate linguistic analysis. After a sentence of interest and meaningful fixation has been identified, the system sends that sentence to the LLM-based linguistic module to be analyzed for grammar, wording, and style [23], [27], [28]. At this stage, the language model can either return a modified version and linguistic problems, depending on the state of the sentence, or produce simpler and more formal rewritten versions if there are no specific errors [26], [28], [29]. Therefore, the linguistic analysis in the system is not performed in a general and global manner, but in a targeted and attention-driven manner [27], [48].

The sixth stage is the processing and organization of the model output. Since the LLM output must be displayed and used in the software system, the model response is received and interpreted in a structured format [30], [32]. This stage involves extracting the modified sentence, detecting linguistic problems, preparing explanations, and generating items that can be displayed in a clear and usable way in the user interface [23], [32]. In fact, the system workflow is not limited to receiving the response from the model, but also requires translating that response into usable components in the UI [27], [32].

The seventh step is to display feedback to the user. In this step, the system presents the results of the linguistic analysis in the form of visual feedback. For example, if there are specific errors in the sentence, the problematic parts are highlighted and suggested corrections or a revised version are displayed in the side panel [27], [28]. If the sentence is grammatically correct, the system can instead provide suggestions for simplifying or formalizing the sentence [26], [29]. This step of the workflow is very important, because the quality of the user's interaction with the system depends on how this feedback is displayed and understood [27].

The eighth step is the user's decision and action. After receiving feedback, the user can accept the system's suggestion, ignore it, or manually edit the text. This step shows that the system has a supporting role and the LLM output is provided to the user as a suggestion, not as a final decision [26], [27], [33]. In this way, the system's workflow is based on human-in-the-loop interaction; that is, the AI provides feedback, but the final judgment and control remain with the author [27], [33].

In addition to these steps, the system also has a parallel flow for recording data and events. During the interaction, gaze data, fixation state, analysis activation time, and some events related to feedback or suggestion actions are stored [5], [22]. This continuous recording allows the system workflow to be analyzed not only at runtime but also later from a research perspective and to investigate user behavior patterns or the efficiency of different parts of the system [15], [27].

In general, the workflow of the proposed system can be summarized as a logical chain of writing → observation → attention detection → fixation detection → linguistic analysis → feedback display → user decision [15], [23], [48]. This chain forms the interactive core of the system and shows how the user's visual behavior can be purposefully integrated into the assisted writing process [5], [15], [53]. The interaction workflow of the proposed system is illustrated in Figure 5.5.

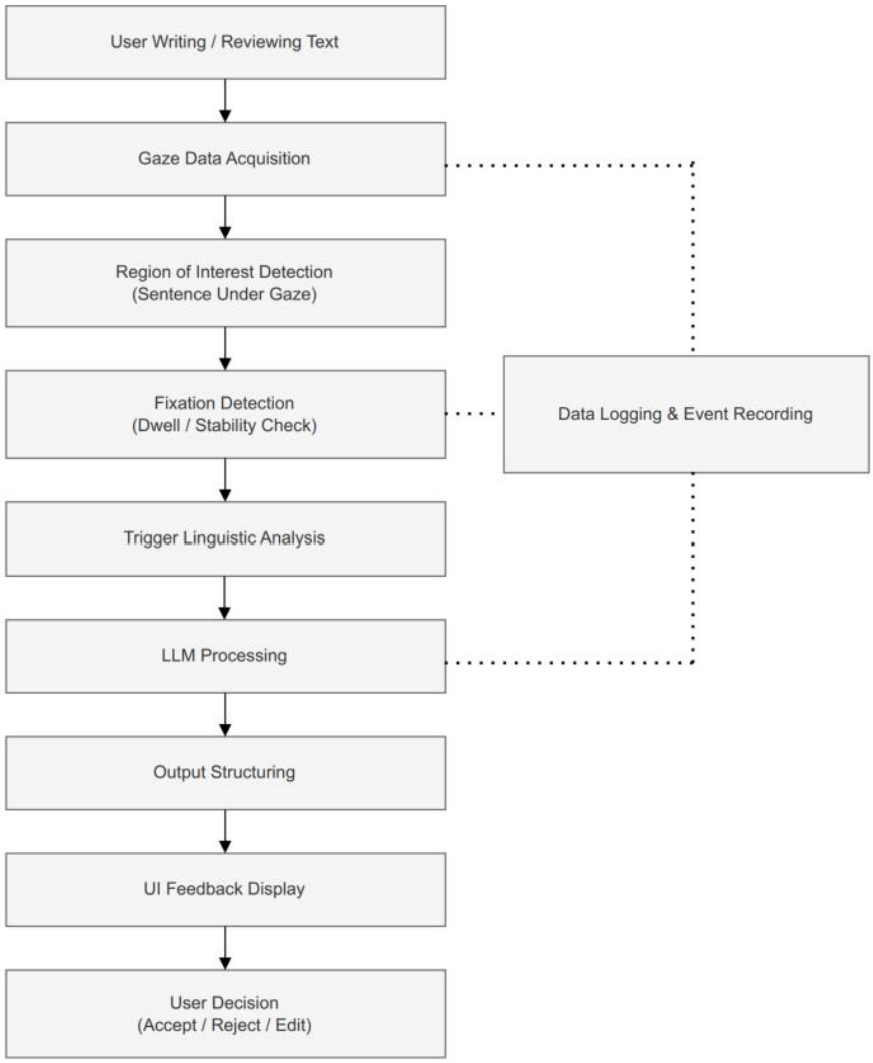

Figure 5.5: Interaction workflow of the proposed gaze-based writing assistant, showing the process from user text input and gaze acquisition to fixation detection, LLM-based linguistic analysis, feedback presentation, and user decision.

5.6 Operating Modes

The proposed system in this study is designed to be used in several different operating modes to be suitable for real-world scenarios of gaze-based interaction and to provide flexibility during the development, testing, and evaluation stages of the system [15], [22]. The existence of several operating modes is also important because gaze-based interaction in practice is not always limited to a fixed condition and may require different ways of activating and controlling the system depending on hardware availability, testing purpose, or user type [5], [15].

In the present system, the operating modes are defined in a way that is compatible with the logic of gaze-aware interaction on the one hand and allows for comparison of the system

behavior under different conditions on the other hand [15], [48]. These modes can be divided into three main categories: Manual Mode, Gaze-Triggered Mode, and Mouse-as-Gaze Debug Mode. Each of these modes plays a different role in the use of the system and represents a different level of dependence on gaze, fixation, and explicit user control.

5.6.1 Manual Mode

In manual mode, the linguistic review process is only activated when the user explicitly requests it. In this mode, although the system can still receive or display gaze data, linguistic analysis and feedback are not directly dependent on fixation, and the user must initiate the correction or style analysis process through interface controls, such as pressing a sentence review button. This type of design is consistent with the way many popular writing assistants operate, in which the user decides when to receive feedback [36], [39].

The importance of Manual Mode is that it provides a baseline interaction mode for the system. In this mode, the system behaves more closely to more traditional writing assistance tools, and the user's gaze does not directly interfere with the timing of the intervention. For this reason, this mode can be useful for conceptual comparisons between gaze-based and explicit request-based interaction [15], [27]. Furthermore, in situations where the user wants to have more complete control over when to receive feedback, the manual mode can be more suitable than the automatic or gaze-triggered modes [26], [27].

5.6.2 Gaze-Triggered Mode

In the gaze-triggered mode, the main logic of the system operates based on the user's visual attention. In this mode, linguistic scanning is automatically initiated when the system detects that the user's gaze has remained on a sentence for a sufficient period of time and that meaningful fixation conditions have been established [11], [15], [17]. This mode is the closest operational mode to the main idea of the thesis, since in it gaze acts not only as auxiliary data, but also as a trigger for linguistic intervention [5], [15], [48].

The main advantage of this mode is that the timing of the system's intervention is coordinated with the user's actual behavior. Instead of constantly scanning the entire text or waiting for an explicit request from the user, the system only intervenes when there is evidence of sustained attention to a sentence [15], [17]. This feature can make feedback more relevant, timely, and less intrusive from an interaction design perspective [15], [48], [53]. However, the

success of this mode depends on the quality of fixation detection, robustness against noise and jitter, and appropriate adjustment of temporal thresholds [19], [20].

Conceptually, Gaze-Triggered Mode represents the main difference between the present system and conventional writing assistants. In conventional tools, the system operates mainly based on text or explicit user requests, whereas here visual attention has become part of the main activation logic [27], [36], [39]. Hence, this mode is not only an execution mode but also represents the interactive innovation of the system [48], [53].

5.6.3 Mouse-as-Gaze Debug Mode

In the mouse-as-Gaze Debug Mode, the mouse cursor position is used as a proxy for gaze data to allow testing and development of the system even without using a real eye tracker. This mode is of research and technical importance, as developing gaze-based systems is usually difficult without a debug or simulation mechanism, and complete dependence on hardware can slow and complicate the development process [22], [46].

In this mode, the system can test parts of the workflow related to gaze interpretation, sentence detection, and activation logic without the need for real eye tracking data. Although mouse behavior is not an exact cognitive and motor equivalent of gaze, it is very useful as a debugging and interface testing tool [15], [22]. This mode allows the researcher or developer to examine the overlay logic, sentence mapping, and activation flow without the need for a full eye tracker setup, and to identify interface or pipeline issues more quickly.

From a design perspective, the presence of Mouse-as-Gaze Debug Mode indicates that the system is designed not only for end-use, but also for incremental development and evaluation. This is of great importance for research prototypes, as many of the challenges of gaze-based interaction can be first explored in a simulated environment and then validated in more realistic conditions with the eye tracker [15], [46].

5.6.4 Role of Multiple Modes in the Proposed System

The presence of multiple operating modes in the present system is not just a technical side feature, but is part of the system design philosophy. Manual Mode serves as a basic and controllable mode, Gaze-Triggered Mode implements the main idea of attention-based interaction, and Mouse-as-Gaze Debug Mode allows for more flexible development and testing of the system [15], [22], [48].

This multi-mode nature has several main advantages. First, it allows the system to be used in different situations, from real-world use to debugging and evaluation. Second, it provides a platform to compare the behavior of the system in different modes and better analyze the real role of gaze-triggered interaction [27], [48]. Third, from a usability perspective, it allows the user or researcher to choose different levels of automation and control depending on their purpose [15], [27].

Overall, the operating modes of the system demonstrate that the proposed architecture of this thesis is not designed for a fixed scenario, but rather to support interaction flexibility, development practicality, and research-oriented evaluation [15], [22], [46]. This feature, along with other system components, provides the basis for a better understanding of the data flow and behavioral logic of the system in the following sections. The operating modes of the proposed system are illustrated in Figure 5.6.

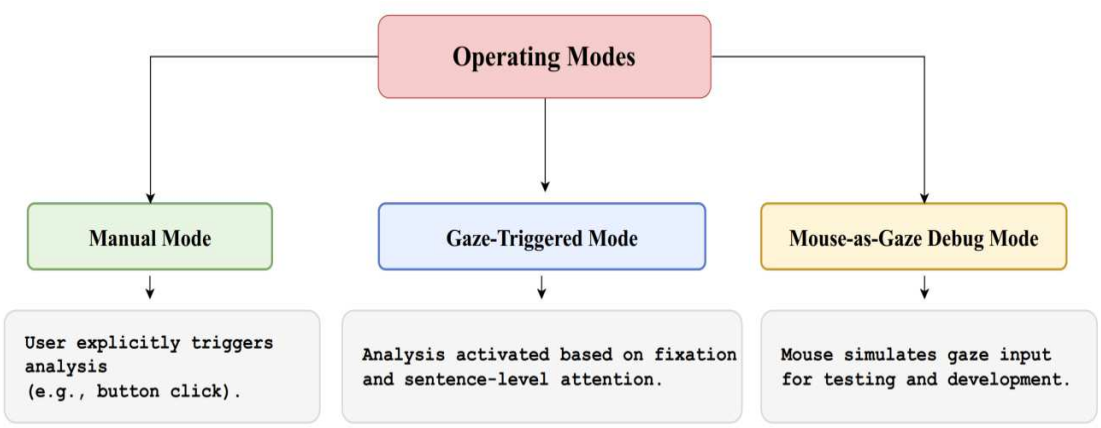

Figure 5.6: Conceptual overview of the operating modes of the proposed gaze-based writing assistant, showing the distinction between manual interaction, fixation-based gaze-triggered activation, and mouse-based simulation for debugging and testing.

5.7 Data Flow in the System

The data flow in the proposed system is designed to connect user behavioral information, text content, and language model output in a coherent, multi-stage chain [5], [15], [23]. This flow starts from the moment of text input and gaze data recording and continues until the feedback display and interaction event recording stage. For this reason, understanding the data flow is essential to understand the logic of the entire system, as it shows how raw eye tracking data is transformed into meaningful linguistic intervention [15], [22], [48].

The starting point of this flow is the user's text in the editor environment. The user enters or revises his/her sentence or text in the interface, and at the same time, the system has the necessary text structure for further processing. This text is the main context on which gaze data must be mapped, since the system can only perform linguistic intervention when a relationship between the user's gaze and a specific part of the text has been established [26], [27].

Simultaneously, gaze data are received from the eye tracker. These data include the gaze position and information related to its validity or registration status and are continuously fed into the system [4], [22]. At this stage, the data is still raw and only represents the approximate gaze position at the interface level. Given that eye tracking data in practice is associated with noise, jitter, and limitations in accuracy and precision, the system must place it in a suitable interactive context before any semantic interpretation [19], [20].

After this stage, the gaze data are passed to the interface mapping and interpretation layer. In this layer, the gaze position is matched with the visual structure of the interface, especially the text editor area, to determine whether the user's gaze is within the textual environment. If the gaze is within the valid area, the system can relate its position to the text structure and identify the sentence or section under the gaze [5], [15], [48]. In this step, the data is upgraded from the raw coordinate level to the text-aware gaze information level.

In the next step, the system enters the sentence-level interpretation stage. In this step, the system determines which sentence the user's gaze is related to and whether this gaze is stable enough to be considered a meaningful fixation or dwell [11], [17], [18]. Thus, an important conceptual transformation occurs in the system's data flow: the raw gaze data is transformed into an attention signal. This transformation forms the basis of the logic of the system, since linguistic intervention should only be initiated when there is sufficient evidence of the user's sustained attention to a sentence [15], [17].

If a meaningful fixation is detected, the corresponding sentence is passed to the language module. In this section, the sentence text, along with the appropriate grammatical context and prompt, is sent to the LLM for linguistic analysis [23], [27], [28]. Depending on the content of the sentence, this analysis can result in the production of a revised version, a list of errors and suggestions, or simpler and more academic rewritten versions [26], [28], [29]. In this way, the data flow of the system moves from a purely perceptual process to a linguistic processing stage.

The output of the language model is not directly displayed to the user after being received, but rather enters a post-processing and structuring stage. In this step, the system transforms the model response into usable components for the interface; for example, the corrected sentence, identified errors, text positions, and short explanations are extracted from each other [30], [32]. This part of the data flow is of great importance because in LLM-based systems, the model output must be organized in a way that is both understandable for the UI and allows the user to apply or reject the suggestions [23], [32].

The data is then fed into the feedback layer. At this step, the system highlights problematic parts of the text, displays correction or rewriting suggestions in a side panel, and provides the user with additional explanations if necessary [27], [28]. This layer is where the system's data flow transitions from internal processing to interactive feedback. The quality of this transformation is crucial for the user experience, as the model output must be presented in a way that is both understandable and maintains a sense of control for the user [26], [27], [33].

Alongside this main flow, a parallel data logging flow also exists throughout the system. During the interaction process, gaze data, fixation status, analysis activation events, and user interactions with the system's suggestions are stored [5], [22]. This information allows the system to not only act in the moment, but also to be examined later from a research and analytical perspective. Hence, logging is part of the system's data flow, not a separate and discrete component [15], [27].

An important point about the data flow of this system is that data does not move in a simple one-way, linear manner, but rather flows in an interactive cycle between the user and the system. The user writes the text, his gaze is recorded and interpreted, linguistic analysis is activated, feedback is displayed, and then the user's decision or edit can change the text again and start the cycle again. Therefore, the data flow of the system is iterative and interactive in nature, and each cycle can be the basis for the next cycle [15], [27], [48].

In summary, the data flow of the proposed system can be summarized as follows: text input → gaze acquisition → gaze-to-text mapping → fixation detection → LLM-based analysis → output structuring → user feedback → interaction logging [15], [23], [32]. This chain shows how the system uses a combination of behavioral data and language processing to create a writing assistant that is aware of the user's attention [5], [15], [48]. The data flow of the proposed system is illustrated in Figure 5.7.

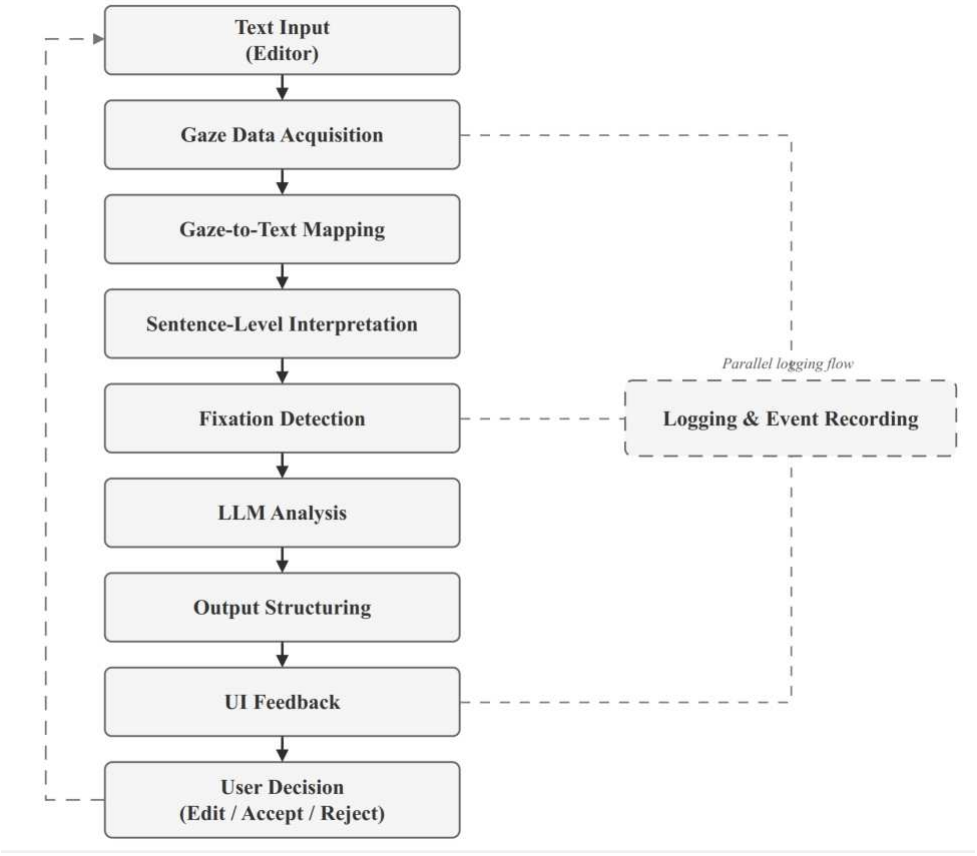

Figure 5.7: Data flow of the proposed system, showing the multi-stage transformation from user text input and gaze acquisition to gaze-to-text mapping, sentence-level attention detection, fixation-based activation of LLM analysis, output structuring, feedback presentation, and iterative user interaction with integrated logging and event recording.

5.8 Design Rationale

The design logic of the proposed system is based on the principle that linguistic intervention should be both content-relevant and temporally appropriate. In many common assisted writing tools, feedback is either permanently active or only provided upon explicit request by the user [36], [39]. In contrast, this study assumes that the user's visual attention can be a behavioral cue to detect the appropriate time for intervention, and therefore the system should intervene at a point where the user is actually pausing on a piece of text and is more likely to need revision [5], [15], [48].

One of the key design decisions was to choose fixation-based triggering rather than reacting to every momentary gaze situation. The reason for this choice is that eye tracking data in practice is subject to noise, jitter, and natural fluctuations, and if the system reacts to every small gaze shift, the probability of unwanted activations increases significantly [11], [19], [20].

The gaze-based interaction literature also shows that dwell time and relative gaze stability provide a more appropriate basis for inferring user intent than instantaneous gaze responses [15], [17], [18].

Another important decision was to use sentence-level attention as the main unit of intervention. If the granularity of the system is too general, feedback may become irrelevant and scattered; and if it is too fine-grained and defined at the word level, the system is highly susceptible to errors due to the limitations of gaze accuracy and precision [19]. For this reason, sentence-level selection was chosen as a balance point; A level that is both linguistically meaningful and interactively more robust to gaze noise than word-level interaction [15], [48], [53].

The design logic of the system also emphasizes minimizing intrusion. One well-known problem in gaze-based systems is Midas Touch, where every gaze is mistakenly interpreted as a request for interaction [15]. For this reason, several design constraints are considered in the present system: requiring stable fixation, limiting the intervention to the sentence under the gaze, and avoiding repeated invocations in short intervals. These decisions are in line with the recommendations of the HCI literature on the design of gaze-aware interfaces, which emphasize error-aware design and reducing intrusiveness [15], [19].

In the linguistic aspect, the system design is based on the premise that feedback should be multi-layered. Users in written environments do not always encounter only grammatical errors; Sometimes the text is grammatically correct, but needs to be rewritten for clarity, simplicity, or formality [26], [27], [28]. Therefore, the system is designed to provide corrections if there is an obvious error, and to move towards style-oriented assistance if the sentence is grammatically correct, suggesting simpler or more academic versions [26], [28], [29]. This decision allows the system to go beyond a simple grammar checker and play the role of a more flexible writing assistant [27], [28].

Another design consideration is to maintain the active role of the user in decision-making. Despite the high capabilities of LLMs, these models still face limitations such as hallucination, prompt sensitivity, and output instability [23], [25], [33]. For this reason, the present system is designed on a human-in-the-loop basis; That is, the model output is presented as a suggestion, and it is up to the user to accept or reject it. This decision is not only important from a reliability perspective, but also in line with the recent literature on human–AI

collaborative writing, which emphasizes the role of the writer as the ultimate decision-maker [27], [33].

From a technical perspective, the use of purposeful prompts and structured output is also part of the system design logic. In systems where LLM enters a software pipeline, the model response should be processable, stable, and as limited as possible to the expected format [30], [32]. Accordingly, the system design is done in such a way that the language model does not produce free and unconstrained output, but rather responses in a specified format for correction, issue listing, and style suggestions. This decision is directly supported by the literature on prompt engineering and structured output support in LLM-based systems [30], [32].

Finally, the choice of a modular and event-driven architecture was also part of the design rationale. Since the present system involves several different layers, including gaze tracking, fixation interpretation, language analysis, and feedback display, separating these components from each other provides an important advantage in terms of development, testing, and evaluation [15], [22], [23]. This structure allows each part to be improved more independently and replaced in the future, for example, with an eye tracker or another language model, without requiring a complete redesign of the entire system [15], [23].

In summary, the rationale for the system design can be summarized in a few principles: attention-based intervention, reduced distraction, sentence-level selection as the appropriate level of interaction, maintaining human control, structuring the LLM output, and modular architecture [15], [23], [27], [48]. These principles not only form the basis for the design of the different parts of the system, but also explain why this system can provide a different type of interaction compared to purely textual writing assistants.

5.9 Chapter Summary

In this chapter, the proposed system of this research is introduced from a conceptual and architectural perspective. First, it is shown that this system is a gaze-based intelligent writing assistant designed by combining eye tracking technology and large-scale language models, and its goal is to transform the text review process from a purely textual process to an interactive and attention-aware process [5], [15], [23], [48].

Next, the design goals of the system are explained and it is revealed that the overall logic of the system is based on several principles: awareness of the user's visual attention,

reduction of unnecessary interventions, maintaining the user's active role in decision-making, provision of linguistic support at multiple levels, and reliable structuring of the language model output [15], [19], [23], [27], [32]. These objectives demonstrated that the system design was not based solely on connecting an eye tracker to a language model, but rather on a set of interactional and technical decisions [15], [22], [48].

The overall architecture of the system was then described as a multi-layered, modular, and event-driven structure. In this architecture, gaze data is received, interpreted at the sentence-level, and if a meaningful fixation is detected, linguistic analysis is activated by the LLM and the results are displayed as understandable feedback in the user interface [11], [15], [23], [32]. Also, the main components of the system, including the user interface, gaze tracking module, gaze interpretation module, language model module, and interaction recording module, were introduced and the role of each in the overall system performance was determined [4], [22], [27], [28].

In the next section, the workflow of the system was examined and it was shown that the interaction flow in this system follows a specific chain: text input, gaze acquisition, gaze mapping to sentence, fixation detection, language analysis activation, model output structuring, feedback display, and user decision [15], [23], [30], [48]. This process indicates that the present system relies not only on the text content but also on the user's visual behavior and uses this data to determine the appropriate time and place for language intervention [5], [15], [53].

Also, different operating modes of the system were introduced, including Manual Mode, Gaze-Triggered Mode, and Mouse-as-Gaze Debug Mode. This section showed that the system is designed for both real-world gaze-based use, for comparison with explicit request-based interaction, and for system development and testing in simulated environments [15], [22], [27], [48]. This multi-state nature increases the flexibility of the system in terms of application and research [15], [46].

Overall, this chapter has shown that the system proposed in this thesis can be considered as an integrated framework in which user visual attention, interaction logic, and LLM-based language processing are purposefully combined [15], [23], [48]. Accordingly, the next chapter will be dedicated to the implementation of the system to show how this architecture and design logic are realized at the software level and in the actual system components [22], [27].

Chapter 6 – System Implementation

6.1 Implementation Overview

The proposed system is implemented as a modular desktop application that aims to integrate the written user interface, gaze data acquisition, fixation interpretation, and language processing based on large language models in a single environment. In this system, the implementation is done in such a way that the different components of the system, including the text editing environment, gaze tracking layer, attention detection logic, language processing modules, and interaction recording layer, operate relatively independently but in harmony with each other. This type of organization is chosen to maintain the readability of the software structure, facilitate incremental development, and support research evaluation of the system [54], [55].

Technologically, the user interface core is implemented based on Qt for Python / PyQt6. This choice was appropriate because the Qt framework provides the necessary facilities for building graphical desktop interfaces, event management, widget control, and organizing concurrent processes [54], [55]. In the current implementation, the system's writing environment is designed based on a simple text editor that is integrated with eye tracking logic and displaying linguistic feedback. The use of a plain-text text editor is also consistent with the prototype nature of the system, because this type of widget is optimized for editing and displaying plain text and performs well in responding to user input [54].

The gaze tracking part of this system is designed based on TCP/XML communication with Gazepoint. According to the OpenGaze API documentation, the Gazepoint server provides data to the client via TCP/IP in XML format, and the client can activate the type of data it needs by sending specific commands. In the current implementation, the gaze acquisition logic is considered as a separate component that receives the data received from the eye tracker, extracts the required features such as gaze coordinates and validity status, and passes them to the interface and gaze interpretation layer [56].

To prevent the user interface from slowing down, the time-consuming or continuous parts of the system are organized into threads and worker objects. The Qt documentation specifies that QThread is used to manage threads and moveToThread() is used to execute worker objects in separate threads. In the current system, the gaze tracking logic and language model-based processing are separated from the main interface flow to maintain system

responsiveness and the UI does not lock up while receiving data or waiting for the model to respond [55]. This decision was necessary from an implementation perspective for an interactive system, since both gaze-based interaction and linguistic feedback require fast and consistent responsiveness.

At the software organization level, the system implementation is based on the separation of responsibilities between modules. In general, the application structure includes modules related to data and shared models, the LLM communication engine, the gaze tracking module, user interface components, and background workers. This separation allows each part of the system to be responsible for a specific role; for example, one layer is responsible for defining the data structure, another layer is responsible for communicating with the linguistic model, and another layer is responsible for graphical interaction and event management. From a software engineering perspective, such a structure helps to better understand the system and its gradual development, especially in prototypes that combine several different technologies in a single framework [55].

From a practical point of view, the implementation of the system was not limited to building an interface or connecting to an eye tracker, but also included developing an intermediate logic that transforms the raw gaze data into meaningful structures at the text level. As a result, the system should be able to match the gaze coordinates to the editor environment, recognize the sentence following the gaze, check the stability of the gaze over time, and only activate linguistic analysis if the necessary conditions are met. It is this middle layer that distinguishes the system from a conventional editor or a conventional grammar checker, and its implementation is one of the central parts of the project.

In addition, the system implementation also uses an interaction logging layer to store key events and gaze data during the session. This decision is not only useful for debugging and development, but also important for research analysis of system and user behavior. As a result, the system implementation was not designed to be purely functional from the start, but also to allow for subsequent analysis of interactions.

Overall, the present system implementation can be considered as a software realization of a multi-modular, interactive, and gaze-aware architecture, in which Qt/PyQt6 is used for the interface and event management, OpenGaze/Gazepoint for receiving eye tracking data, and the LLM communication module for language processing are integrated [32], [54], [55], [56]. In the following sections of this chapter, the main components of the implementation will be

examined in more detail; including the project structure, user interface, gaze data processing, fixation detection logic, and how the language module is integrated into the system. The implementation structure of the proposed system is illustrated in Figure 6.1.

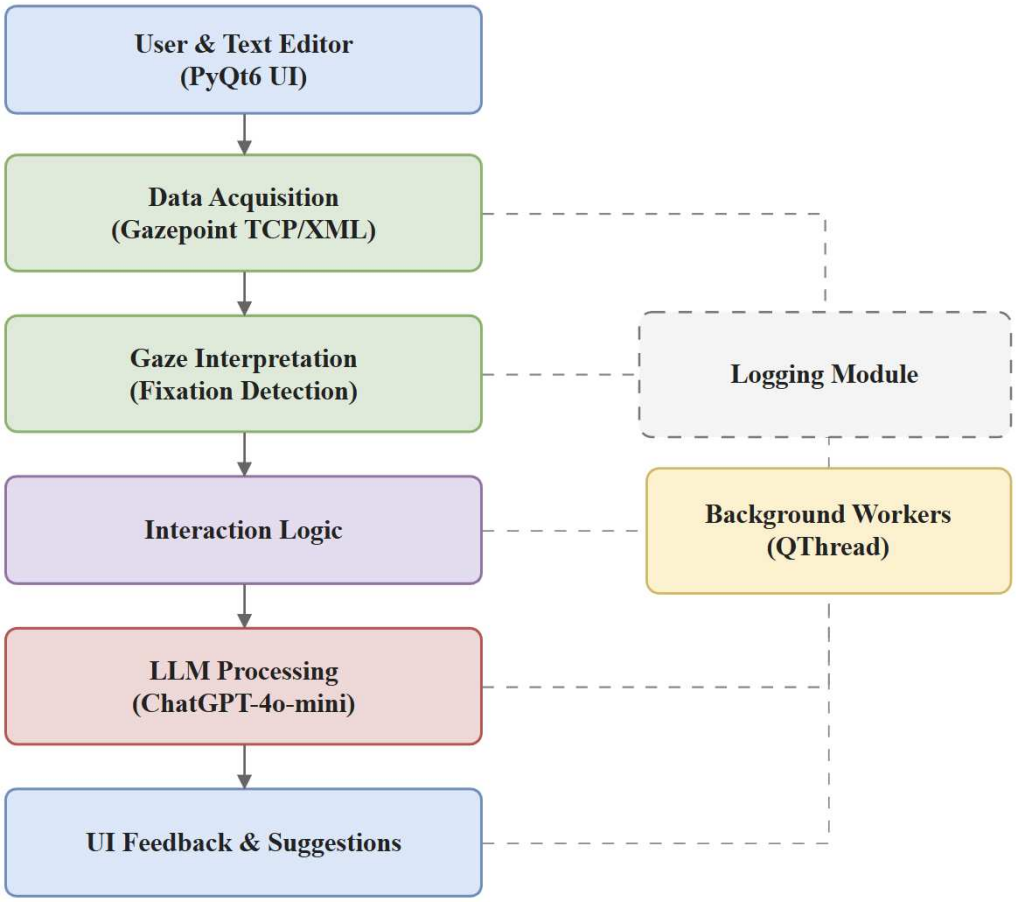

Figure 6.1: Implementation overview of the proposed system, illustrating the integration of the PyQt6 user interface, gaze data acquisition, gaze interpretation, interaction logic, LLM processing (ChatGPT-4o-mini), and user feedback components. The architecture also includes a background worker layer for time-consuming processes and a logging module for recording gaze data and interaction events.

6.2 Technology Stack

The implementation of the proposed system in this study is based on a multi-layer technology stack, each part of which plays a specific role in realizing interactive capabilities, receiving gaze data, language processing, and recording system events [32], [54], [55], [56]. These technologies were selected with the aim of ensuring that the system can be responsive in terms of user interface, reliable in terms of communication with the eye tracker, and also provide integration with the language model and structured processing of the output [32], [54], [56].

In the programming layer, the system is developed using Python. Python was chosen because it is a very common and efficient language for building research prototypes, quickly integrating different modules, and connecting to various software libraries. In this project, Python provides a common platform for implementing the user interface, gaze tracking logic, data processing, and communication with the language model. Since the present system is multi-modular and research in nature, it is a good idea to use a high-level language that facilitates incremental development.

For the user interface layer, PyQt6 / Qt for Python is used. The Qt framework provides the necessary facilities for building graphical desktop interfaces, event management, widget control, and organizing visual components [54]. In the present system, the text editor, suggestion panel, help information panel, and main interactive components are all implemented in this framework. The choice of Qt is also important because it is a mature and reliable option for building event-driven interfaces and integrating them with real-time inputs, including gaze data [54], [55].

In the text editing section, the use of plain-text editing widgets was consistent with the system prototype logic. These types of widgets are suitable for entering and displaying plain text, while allowing for customization of behavior, cursor control, and overlaying with visual assistive elements [54]. In the present system, this choice made the implementation of the text layer simpler and at the same time sufficient for the sentence-level writing assistance scenario.

To manage concurrent processes and prevent the user interface from locking, QThread and the worker object pattern were used. The Qt documentation explicitly indicates that executing time-consuming tasks in separate threads and using moveToThread() for workers is a good way to maintain interface responsiveness. In the present system, this approach is used for processes such as continuously receiving gaze data and executing language requests to keep the system stable and responsive during user interaction [55].

In the gaze tracking layer, the system relies on the Gazepoint / Open Gaze API. According to the official Gazepoint documentation, eye tracking data is transferred to the client via TCP/IP in the form of XML messages, and the client can activate or manage the data flow it needs by sending appropriate commands. This communication model is suitable for the present prototype, because it allows the online receipt of gaze coordinates and other gaze-related fields in a relatively simple and processable structure [56]. In the implementation of the

system, the same structure is used to receive data, parse the XML, and transfer them to the gaze interpretation layer.

In the language processing layer, the system is integrated with the OpenAI API. This layer is responsible for sending targeted prompts to the language model and receiving responses related to grammar correction and style assistance. In this project, the use of API was chosen because the language model can function as an independent service but integrated with the software logic of the system. Also, considering the system's need for processable outputs, the use of structured outputs and controlled formats in this layer becomes important [32].

In addition to these core technologies, the system also uses standard Python libraries and modules for tasks such as socket communication, XML parsing, JSON handling, and file-based logging. Although these components are less prominent in terms of architecture compared to Qt, Gazepoint, and OpenAI, they are essential for achieving stable system performance; because communication with the eye tracker, receiving and interpreting gaze data, and storing interactions would not be possible without these supporting layers.

From a general perspective, the technology stack of the system can be summarized in four levels: the general language and logic layer (Python), the interface and event management layer (PyQt6 / Qt), the eye data acquisition layer (Gazepoint / Open Gaze API), and the language processing layer (OpenAI API) [32], [54], [55], [56]. This combination has enabled the system to use real-time gaze data, perform advanced language analysis, and provide feedback to the user in the form of an interactive interface in a single environment.

In general, the selection of technologies used in this project was based on three main criteria: the possibility of modular development, appropriate responsiveness in real-time interaction, and compatibility with the research needs of the prototype [54], [55], [56]. This technology stack provides a platform on which the implementation of the main components of the system will be explained in more detail in the following sections of the chapter.

6.3 Project Structure

The implemented project structure is organized in a modular manner to separate the different components of the system, including the user interface, gaze tracking, language processing, and interaction recording, and to simplify the development and maintenance of the system [54], [55]. This type of organization is consistent with the nature of the present system, since the

project simultaneously includes several different layers: the user interaction layer, the gaze data acquisition layer, the interaction logic layer, and the language processing layer [32], [55].

At a general level, the project consists of several main file groups and modules. One section contains files related to the application startup and configuration; another section is dedicated to data models and event recording; one is intended for communication with the language model; one is responsible for receiving eye tracker data; and finally, a set of files implements the user interface, visual overlay, and background workers. This separation allows each part of the system to have a specific role and the dependencies between components to be controlled as much as possible [54], [55].

The entry point of the program is in the main executable file, which is responsible for creating the application object and launching the main window. This structure is common in Qt-based applications, because QApplication manages the event loop and the main interface cycle, and the main window acts as a place to collect system components. In the present system, this file also plays the role of bootstrap, and the main logic of the system is not located there, but is delegated to the main window and more specialized modules [54].

The project configuration section contains the settings file, which keeps the main system values, such as the language model used, fixation and dwell time thresholds, gaze display radius, and the log storage path in a centralized location. This type of design makes it possible to change the behavior of the system without directly manipulating multiple files, and is very useful for research prototypes that need to adjust parameters. Having a centralized configuration layer also increases the readability and reproducibility of the system.

The data model section contains the structures used to exchange information between different modules. For example, data related to gaze samples, language issues, and correction results are maintained in a structured manner for use both in the internal logic of the system and in communication between workers, the interface, and the language module. Using clear data structures makes the dependencies between components rely less on raw strings and scattered variables, and the implementation remains more organized.

The logging section is implemented as a separate module that stores important system events and gaze data in separate files. This separation allows logging to remain an independent part of the interface or language logic and can be used for both debugging and research analysis. In practice, separating logging from the rest of the components helps reduce the complexity of the main window and allows for more independent development of this part.

The language model communication module is placed in a separate section that is responsible for creating the prompt, sending the request, and initial interpretation of the response. This separation prevents the API-related logic and model output processing from mixing with the interface or eye tracking logic. In the present system, this section is used for both correction and style analysis and is therefore considered one of the central components of the project structure [32].

The gaze tracking section includes a module that manages the communication with the eye tracker and receives and interprets gaze data via TCP/XML. This module is kept separate from the UI and language modules to reduce the UI's direct dependence on low-level hardware interactions. This is an important architectural decision, as changes to the source of gaze data or how it is received do not necessarily require extensive changes to other parts of the system if this separation is maintained [56].

The UI itself is made up of several separate components. The main window is responsible for overall component coordination and high-level event handling, while the text editor, visual overlay, and suggestions or information panel each have a specific task at the UI level. This separation is useful in Qt-based projects, as it creates reusable and more readable components rather than cramming all the visual and interaction logic into a single file [54], [55].

Worker modules are also an important part of the project structure. These workers are placed in separate threads to perform time-consuming tasks, such as queries to the language model, to maintain interface responsiveness [55]. In addition to the technical advantage, separating the correction worker and style worker in the project structure is also useful in terms of organizing the system logic, since each worker remains responsible for a specific type of language task [32], [55].

In general, the project structure can be summarized in a few main clusters: setup and configuration, data models and logging, communication with LLM, gaze tracking, user interface, and background workers. This organization has made the system implementation, despite its multifaceted nature, understandable and relatively extensible. Therefore, the project structure in this study was not just a way to organize files, but also part of the system implementation strategy to maintain modularity and clarity [32], [54], [55]. The project structure of the implemented system is illustrated in Figure 6.2, showing the organization of the main software modules and their relationships.

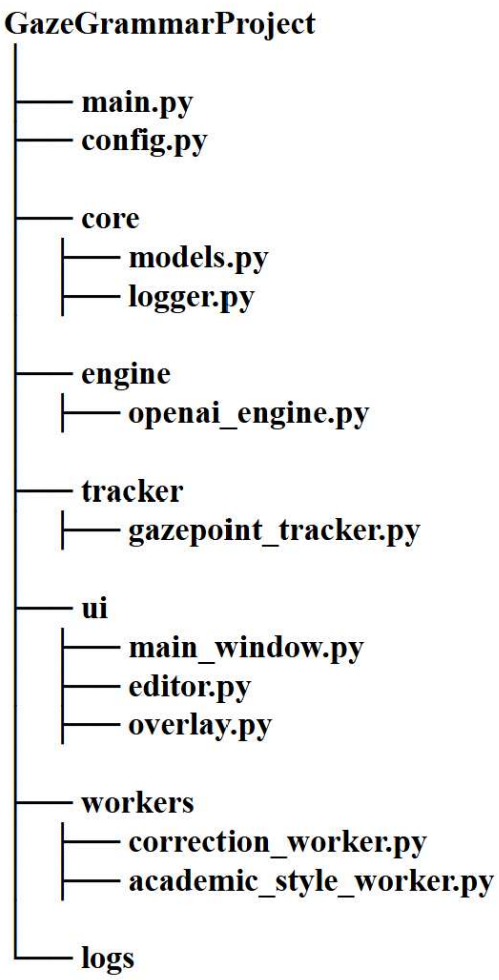

Figure 6.2: Project structure of the implemented system, showing the hierarchical organization of the main files and modules, including configuration, core components, LLM engine, gaze tracking, user interface, background workers, and logging.

6.4 User Interface Implementation

The user interface of the proposed system is implemented based on Qt Widgets in PyQt6 / Qt for Python. This choice is appropriate because the Qt framework provides the necessary facilities for creating main windows, text editors, scrollable areas, custom widgets, and interactive event management in a unified desktop environment [54], [57]. In the present system, the user interface not only plays the role of displaying text, but also forms the main platform for user interaction, gaze data, and linguistic feedback.

The core of the user interface in this system is implemented based on a main window. In Qt, the QMainWindow class provides a standard framework for building application interfaces and allows the programmer to use the central area, toolbar, and status bar. In the

present implementation, the main window is the place where the various interface components are integrated, and the text editing sections, the suggestions panel, and the help information are organized in this same environment [57].

In the central area of the interface, a simple text editor is used for entering and reviewing sentences. This section is developed based on QPlainTextEdit, because according to the Qt documentation, this class is optimized for displaying and editing plain text and is faster in responding to user input and managing text documents than richer alternatives. This choice is consistent with the prototype nature of the system, since the project focuses on sentence-level writing assistance and there is no need for the full capabilities of a rich text editor [54].

To adapt the editor to the logic of the system, its behavior has been developed in a customized way. In this section, controls are provided for managing the editable area, displaying placeholders, limiting the cursor position, and facilitating text replacements. Since QPlainTextEdit allows for finer control of the cursor, text, and input events, it has been a good basis for developing an interactive editor that is compatible with the gaze-aware logic of the system [54].

One of the important elements of the user interface is the visual overlay layer that is placed on top of the editor viewport. This layer is implemented as a custom QWidget. According to the Qt documentation, QWidget is the base class for all user interface components and can be used to create custom widgets, receive events, and perform paint operations [58]. In the current system, the overlay is used to display the gaze circle, sentence area rectangle, and fixation marks on the editor, thus connecting the user interface to the user's visual behavior [58].

The implementation of this overlay uses the paint and resize events to keep the visual overlay in sync with the editor viewport. This approach is natural in the Qt framework, as custom widgets can adjust their visual behavior to interface changes by overriding the paint and resize events [58]. In the present system, this decision has allowed the display of gaze and fixation to be done in real time and in sync with the interface changes.

To display suggestions and descriptions, a scrollable area is used, built on top of QScrollArea. The Qt documentation shows that QScrollArea is used to display a child widget inside a scrollable frame and is useful when the content of the area may exceed the size of the frame. In the present system, this feature has been suitable for displaying suggestion cards,

error descriptions, rewritten versions, and help information, since the amount of this content can vary depending on the model output and sentence type [59].

The user interface is also organized based on the signal-slot pattern in Qt. The Qt and Qt for Python documentation emphasize that signals and slots are the main communication mechanisms between objects in this framework, allowing different components of the system to interact with each other in a loosely coupled manner. In the current implementation, this pattern is used to connect events such as text changes, clicks on interface elements, receiving gaze data, and completing background workers. This feature helps to keep the interface logic, language processing logic, and gaze tracking logic structurally separate from each other, while still being able to exchange events with each other at the appropriate time [60], [61].

Another important feature of the interface implementation is the use of event-driven interaction. In this system, much of the UI behavior is based on events, timers, and signals sent between components rather than heavy polling. Qt is inherently designed for this type of programming, and its central event loop enables the coordinated execution of widgets, threads, and events [57], [60]. This model is suitable for the present system, because the interaction between gaze, fixation detection, and feedback display must be continuous but smooth and controlled.

From a design perspective, the user interface of this system is organized into two columns: a main area for writing and viewing text, and a side area for displaying suggestions, explanations, and auxiliary information. This structure is useful in terms of usability, as it allows the user to view the main text and the system feedback at the same time, without one replacing the other or disrupting the writing experience. Also, the presence of a status bar and auxiliary information sections has helped to better understand the system state and debug during development [57].

Overall, the user interface implementation in this project is based on a widget-based, event-driven, and extensible framework [54], [57], [60]. The use of QMainWindow for the overall interface organization, QPlainTextEdit for the writing environment, a custom QWidget for the visual overlay, QScrollArea for the feedback area, and the signal-slot pattern for communication between components provides a platform where gaze-aware interaction and writing assistance can be integrated into a single, controllable environment [54], [57], [58], [59], [61].

6.5 Gaze Data Processing

Gaze data processing is one of the key implementation layers in the proposed system, as it is responsible for transforming the raw eye tracker output data into information that can be used at the interface and interaction logic level of the system [4], [19], [56]. In practice, eye tracking data directly and without processing is not sufficient for a writing assistant, because the system must be able to determine from the raw gaze coordinates which area of the interface the user is on, whether his gaze is valid, and whether this gaze can be the basis for linguistic intervention [5], [15], [22].

In the input layer, gaze data is received via the Gazepoint / Open Gaze API. According to the official documentation of this API, communication with the eye tracker is done via TCP/IP and the data is sent to the client application in XML format. In the present system, a dedicated component is also considered to receive these messages, which is responsible for reading XML records continuously, extracting gaze-related fields, and converting them into a format suitable for the higher layer [56]. This design keeps the eye data reception separate from other parts of the interface and system logic, and possible changes in the protocol or device handling have less impact on other components.

In the next step, the received data is checked for validity and usability. In eye tracking applications, not all gaze samples are necessarily valid; for example, blinking, gaze departure from the trackable area, or momentary device instability can make some samples unsuitable for interactive interpretation. For this reason, the present system does not rely only on recording raw coordinates, but also considers the validity status of the data in subsequent processing to avoid misinterpretation of uncertain gazes [19], [20].

One of the important steps in gaze processing is the transformation of normalized eye tracker coordinates into user interface coordinates. In many eye trackers, gaze position is reported as normalized or device-space-dependent values, while the system requires usable window and viewport-level coordinates to display the overlay and interpret the region of interest [4], [22], [56]. Therefore, in the present implementation, gaze data is first mapped to the global display space and then to the local coordinates of the text editor. This spatial transformation is one of the main prerequisites for sentence-level attention detection, since without it, the relationship between eye data and the text in the interface cannot be established [5], [15].

After this spatial mapping, the gaze data enters the interface-oriented phase; that is, the system checks whether the user's gaze is within the text editor's range or not. This phase is of great importance because in a desktop environment, the user's gaze may be focused on any part of the screen and only a part of it is meaningful for the text-assistance logic [15], [19]. In the present system, gaze data is only interpreted when it is in the text-related region. This decision helps to reduce irrelevant activations and limit the interaction domain to the text [15], [17].

In addition to the real eye tracker data, the system also supports a mouse-simulated gaze mode. In this mode, the mouse cursor position is considered as a surrogate for gaze to allow testing of the interface, overlay, and sentence-level mapping logic even without the presence of a real eye tracker [22], [46]. Although mouse behavior is not a complete cognitive and dynamic equivalent of gaze, it is very useful for debugging and testing the gaze processing pipeline, as it allows for rapid inspection of interaction logic in a controlled environment [15], [22].

Another important component of this layer is the real-time representation of gaze in the form of an overlay. After the gaze coordinates have been transformed into the local interface space, the system can visually display them on the editor environment; for example, using a circle indicating the gaze, the center point of the gaze, and an approximate uncertainty range [19]. This representation is not only visual, but also has practical importance in the development and evaluation process of the system, as it helps to better understand and verify the accuracy of the gaze mapping to the interface and the sentence detection behavior [22], [58].

From a processing perspective, gaze data in this system is considered as streaming input, not as individual and independent data. In other words, the system does not react to each gaze instance in isolation and without temporal context, but interprets it in the context of a temporal sequence of instances [11], [15]. This feature is essential for the transition from gaze positioning to fixation detection, since many of the system's interactive decisions, such as attention stability detection, are only possible by considering the continuity of gaze over time [11], [17], [18].

In summary, gaze data processing in the present system involves several main steps: data acquisition from the eye tracker, validation, spatial mapping to the interface environment, restriction to the text area, support for simulation mode, and data preparation for the attention and fixation interpretation layers [4], [15], [56]. This layer essentially creates a bridge between

the eye tracking hardware and the interaction logic of the system, and without it, meaningful use of gaze for writing assistance would not be possible [5], [22], [48].

6.6 Sentence Detection and Fixation Logic

One of the most important parts of the proposed system implementation is the logic for detecting the sentence that the user is interested in and determining whether his gaze has reached a level of stability that can be considered a meaningful fixation. This part is in effect the layer that elevates gaze data from the raw coordinate level to the interaction-relevant textual attention level and allows the system to decide when to activate linguistic analysis [5], [11], [15].

In this system, sentence recognition is based on a direct mapping between gaze coordinates in the editor environment and the text position. After the gaze is converted to local editor coordinates, the system uses the facilities available in the editor to determine the cursor position corresponding to that point. This decision is appropriate because in a QPlainTextEdit-based environment, the text and its locations are accessible in a structured way in the form of blocks and cursors, and therefore the user's gaze can be associated with the actual structure of the text [54].

After determining the approximate position of the gaze in the text, the system enters the sentence boundary detection stage. In this stage, the goal is to extract the sentence boundary on which the user's gaze is located. In the present system, this is done using simple punctuation-based segmentation rules; that is, sentence boundaries are determined mainly based on signs such as periods, question marks, and exclamation marks. This approach, despite its simplicity, was suitable for the present prototype, because the main goal of the system is to detect sentence-level regions in a controlled text environment, not full syntactic analysis or discourse-level segmentation [15], [48].

After determining the sentence boundaries, the system must check whether the user's gaze is actually within the acceptable region of that sentence. Since gaze data is always subject to fluctuation and uncertainty, relying on a single point for decision-making is not appropriate [19], [20]. For this reason, in the present implementation, an extended attention region is considered for each sentence, which extends slightly beyond the direct visual boundary of the text lines. This decision helps to ensure that small fluctuations in gaze or minor spatial mapping errors do not cause sentence recognition to be lost [15], [19].

In this architecture, sentence recognition is not only performed instantaneously, but also involves temporal stability logic. In other words, the system does not just check which sentence the user's gaze is on at a given moment, but also checks whether this sentence–gaze relationship remains stable over time [11], [17]. This feature is essential for the transition from sentence detection to fixation detection, because in a gaze-aware system, reacting to any instantaneous gaze change is neither desirable nor defensible from a usability perspective [15], [18].

The fixation logic in this system is designed based on gaze persistence on a specific sentence key. The sentence key is the implicit identifier derived from the beginning and end of a sentence in the text. When the system detects that the gaze remains on the same sentence, a timer is activated for that region. If the gaze leaves the sentence before reaching the required time threshold, the process is reset; however, if the gaze persistence continues long enough, the system considers it a meaningful fixation [11], [17], [18]. This approach is conceptually similar to fixation detection algorithms based on persistence or dwell logic, although here the main unit is not a single point but a sentence-level region [11], [21].

One reason for choosing this logic is that in text environments, the user's attention is usually focused not on a single pixel or a small target, but on a meaningful range of text. If the system were to define fixation solely based on instantaneous coordinates or very small targets, it would be both more sensitive to noise and less useful in terms of writing [19]. In contrast, choosing sentence-level fixation allows linguistic intervention to be performed on a unit of text that is also more logical in terms of grammar correction and rewriting [15], [48], [53].

In the present implementation, the fixation logic is also accompanied by several control conditions to avoid unnecessary activations. For example, if the user is still actively typing or their gaze only briefly passes over the sentence, linguistic analysis should not be initiated immediately. For this reason, in addition to the fixation threshold, the system uses auxiliary logic to pause typing and to prevent repeated re-triggering on a sentence within a short interval. This type of control is in line with the recommendations of the gaze-based interaction literature on reducing intrusiveness and managing Midas Touch [15], [17], [19].

Another part of this logic concerns issue-level attention detection. After sentence-level fixation has triggered linguistic analysis and a result containing issues has been obtained, the system can also check at a lower level whether the user's gaze is focused on one of the problematic parts of the text. This design allows the system to move from a sentence-level intervention to a finer interaction to display suggestions and explanations. Thus, the attention

logic in the present system is not only used for the initial trigger, but also plays a role at a later level to guide interaction with feedback [15], [48], [53].

From an implementation perspective, the sentence detection and fixation logic must be compatible with both the text structure in the editor and the non-ideal properties of gaze data. This has made the design of this part not just a simple algorithmic problem, but rather a kind of interface between UI semantics and gaze interpretation. In fact, the system must be able to both use the features of the text cursor and block geometry and take into account the natural uncertainty of gaze data in decision-making [19], [54], [58].

In general, the sentence detection and fixation logic in this system is based on three principles: mapping gaze to text, determining sentence-level regions, and assessing gaze stability over time [11], [15], [17]. Together, these three parts form the same mechanism that makes the idea of gaze-triggered writing assistance possible at the implementation level. Without this layer, the system could only register gaze, but was unable to convert it into a meaningful signal to activate linguistic analysis [5], [15], [48]. An example of the system behavior during sentence-level gaze interaction is shown in Figure 6.3, where the user's gaze is mapped onto the text and triggers linguistic feedback for the attended sentence.

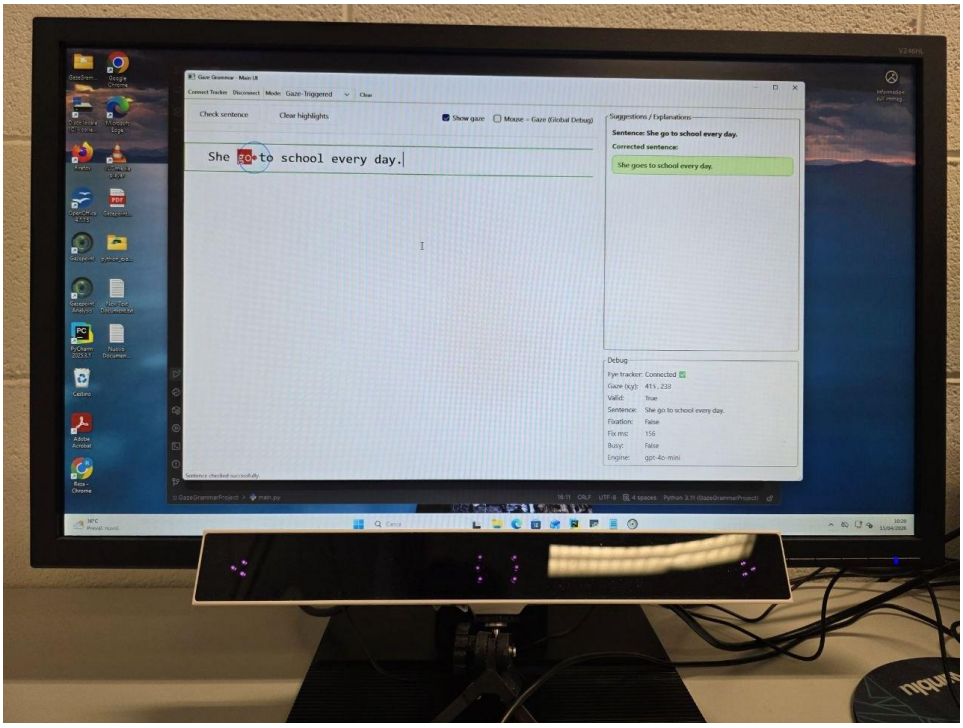

Figure 6.3: Example of sentence-level gaze interaction in the implemented system, showing the mapping of gaze position to text, visual overlay of gaze, and the activation of linguistic feedback for the attended sentence.

6.7 Language Model Integration

The integration of the language model in the proposed system is done in such a way that the language module acts as an independent component but coordinated with the interface logic and gaze-based interaction. In this implementation, the language model is not directly integrated into the UI core, but is called through an interface layer to manage requests, prompts, and received outputs in a controlled manner [23], [30], [32]. The official OpenAI documentation and the official Python library also indicate that interaction with the models is done programmatically and modularly through the Responses API and the openAI library, and this API is currently the main way to interact with the models in the official library [32], [57].

In the present system, a dedicated module is considered for communication with the LLM, which is responsible for making requests, sending prompts, receiving responses, and preparing output for subsequent layers of the system. This separation is important from an implementation perspective, as the API logic, error handling, and model response parsing should not be mixed with the user interface logic or gaze interpretation logic [30], [32]. This approach is also consistent with the recommended structure in the official OpenAI library documentation, where the API client and the model invocation logic are organized independently and reusable [32], [57].

At the functional level, the integration of the language model in this system supports two main paths: grammar correction and style-oriented analysis. In the first path, the selected sentence is sent to the model to detect grammatical, lexical, and punctuation errors, and the expected output consists of the corrected sentence and a list of issues [26], [27], [28]. In the second path, the system asks the model to evaluate the tone of the sentence and provide simpler or more formal alternatives to take writing assistance beyond grammar checking [26], [28], [29]. Since these two types of processing have different output structures, the current implementation considers appropriate prompt and parsing logic for each path [30], [32].

A key component of this integration is the design of purposeful prompts. As explained in Chapter 3, the quality of LLM output does not depend solely on the model, but also on the way the task is stated, the constraints, and the format of the response play a decisive role [23], [30]. In the current system, the prompts are designed so that the model produces concise, task-specific, and software-structured output instead of free-form responses. This logic is consistent with the prompt engineering literature, which emphasizes the importance of task specificity, limited response scope, and explicit definition of the output format [30].

In order to make the model output more reliable in the system pipeline, the LLM integration in this project relies on structured output. The official OpenAI documentation states that the Structured Outputs feature is designed to force the model to conform to a specific format, such as JSON schema [32]. In the current system, the logic for communicating with the model is also arranged in such a way that the model response is received in a structured format whenever possible; this makes the extraction of fields such as corrected sentence, issue list, category, explanation, tone, and rewriting options simpler and more robust [30], [32]. The use of structured output is especially important in interactive systems, since the UI and suggestion application logic require responses with a predictable form [32].

However, since the model output is not always fully deterministic, the present implementation also includes a post-processing and validation layer. In this layer, the model response is checked upon receipt to ensure that the overall structure is usable and that the expected fields are present. If the response is not valid as a direct JSON, the system attempts to extract its structured part and then map it to the system's internal data models. This approach is consistent with the practical reality of using LLMs, since even in the presence of structured prompting, robust parsing and additional validation are sometimes still required [30], [32].

From a scheduling perspective, requests to the language model in the present system are executed in the background so that the user interface does not lock up while waiting for the API response. This decision is especially necessary in an interactive system, since both gaze input and text interaction require appropriate responsiveness [54], [55]. Therefore, LLM integration is not just a matter of API integration, but also depends on concurrency management, system busy state control, and coordination between workers and UI [32], [55].

In this system, the model output is passed to the UI layer after validation to be displayed as a highlight, corrected sentence, suggestion card, or style option, depending on the type of response. This step shows that the integration of LLM in the project is not limited to just text generation, but also includes the transformation of the model response into interactive elements understandable to the user [26], [27], [28]. Also, because the system maintains the human-in-the-loop role, the model output remains as a suggestion and is not directly imposed on the text without the user's decision [27], [33]. This is important from the perspective of reliability and usability, since the limitations of the language model still require the user to be the final decision-maker [23], [33].

In summary, the language model integration in this project can be considered a combination of API-based interaction, prompt engineering, structured output handling, robust post-processing, and UI-oriented rendering [30], [32]. This layer is one of the most important parts of the system implementation, as it is the point where gaze-triggered interaction becomes practical writing assistance and the language model elevates its role from an external service to an active part of the system logic [23], [27], [48].

6.8 Issue Detection and Highlighting

One of the important parts of the system implementation is the layer for recognizing language issues and visually representing them in the text. This layer acts as an interface between the output of the language model and the user's interactive experience, as the model response must be transformed from a text or JSON-like structure into components that can be displayed and acted upon in the editor environment [27], [30], [32]. In practice, this layer determines how the system extracts errors, suggestions, and explanations from the model response and how it provides them to the user in the form of highlights and feedback [26], [28], [29].

In the first stage, the language model response, after initial validation, is transformed into a set of issue items. Each issue typically consists of a problematic text fragment, an alternative suggestion, an error category, and a short explanation [27], [28]. This type of structure is compatible with the logic of writing assistance, because the system does not only produce a final modified sentence, but also tries to make the problematic parts recognizable to the user so that the relationship between the original text and the system's suggestions remains clear [26], [27].

One of the main challenges in this area is the mapping of the linguistic output of the model to its actual position in the text. The linguistic model usually returns issues in the form of text or conceptual structure, but to display them in the editor it is necessary to specify which text interval each issue exactly corresponds to in the original sentence. In the present system, this is done by comparing the content of the issue with the text of the sentence under analysis and finding the appropriate offsets in the same sentence. This step of the implementation is very important, because if the issue position is not determined correctly, highlights and suggestions will also be displayed in the wrong place and the usability of the system will be reduced [27], [29].

In this system, a text-based offset resolution logic is used to map issues to text. In this approach, the system first tries to find the error_text section directly, or if necessary, case-insensitively and based on token matching, in the original sentence. If a suitable position is found, its start and end range is stored and the issue is prepared for rendering. This type of approach is suitable for the present prototype, because the model output is usually limited to the sentence level and the identified issues are often related to short and specific ranges in the text [27], [28].

However, in some cases, the language model may report issues as rewritten phrases or with different wording from the original text. In such situations, direct mapping is not always sufficient. For this reason, the present system also uses a fallback mechanism that operates based on the difference between the original sentence and the modified sentence. In this step, by comparing the word sequences or tokens of the original text and the modified version, the changed parts are extracted as candidate issues. This approach allows the system to roughly identify the changed areas even when the offset model does not provide an explicit or direct match [28], [29].

After determining the location of the issues, the system enters the highlight rendering stage. In this step, the problematic areas of the text are highlighted in the editor using visual formatting so that the user can quickly identify them in the text. In the Qt framework, this behavior is usually done using the selection and text formatting mechanisms on the editor. In the present system, a text range is selected for each issue and is visually distinguished from other parts of the sentence by appropriate formatting, including changing the background or text color [54].

The importance of highlighting is that effective writing assistance is not limited to presenting suggestions in a side panel, but must also establish a clear visual connection between the main text and the system's feedback. If the user cannot quickly identify which part of the sentence the suggestion relates to, the system's efficiency decreases and the interaction flow is disrupted. For this reason, highlighting is considered part of the system's usability, not just a visual choice [27].

In addition to in-text highlights, the present system also uses a suggestions and explanations panel. In this area, the system displays the selected issue or issue of interest along with its category, suggestion, and explanation. This design allows feedback to be provided at two levels: first, in the text itself via a highlight; and second, in a side panel via a description

and actionable options. Such a design is in line with common patterns in writing assistants, as it both relies on context within the text and allows for explicit interaction with suggestions in a separate area [26], [27], [28].

Another important part of this implementation is the user interaction with issues. In the present system, issues are not just static elements, but can be opened interactively and their suggestions applied. For example, if the user clicks on a highlighted area or focuses their gaze meaningfully on it, the system can display details about that issue in the feedback panel. This feature transforms feedback from a passive state to a more active interaction layer [15], [48], [53].

At the implementation level, applying suggestions to the text is also part of this layer. When the user decides to accept a suggestion, the system must replace the issue-related interval in the original text with the suggestion and then update the interface. This process requires coordination between the issue data model, the cursor structure in the editor, and the highlight update logic [54]. For this reason, the issue detection and highlighting layer is not limited to extracting errors only, but also forms part of the complete identify → display → select → apply cycle [27], [28].

In general, this part of the system implementation can be considered as a layer that translates the linguistic response of the model into interactive and visual feedback for the user [27], [30], [32]. Without this layer, the LLM output remains a mere text response; However, suggestions, explanations, and suggested changes are integrated neatly into the text and user interface, making writing assistance more usable and controllable [26], [27].

6.9 Background Workers and Asynchronous Execution

In the proposed system, some of the features operate in such a way that if they are executed in the same thread as the main interface, they can reduce responsiveness or even temporarily lock the interface. Since this system both receives gaze data continuously and sends language requests to an external service, its implementation required a suitable mechanism for asynchronous and background execution [32], [55], [57]. In the Qt framework, it is also explicitly recommended that time-consuming or blocking tasks be separated from the main interface thread so that the interface event loop can still respond to events [54], [55].

In the present implementation, this requirement is met through the use of QThread and the worker object pattern. The Qt documentation explains that QThread provides a platform-

independent way to manage threads and also emphasizes that work objects can be moved to a separate thread using moveToThread() so that their slots are executed on the same thread. The same approach is used in the present system to keep time-consuming tasks separate from the UI layer and to keep the user interaction with the editor and overlay smooth and stable [55].

One of the most important uses of background execution in this system is for language model requests. Invoking the language model API can inherently involve network latency, server-side processing time, and response parsing time, and if this operation is performed on the main thread, the user interface will be interrupted while waiting for the response [32], [57]. For this reason, in the present system, separate workers are used for the correction and style analysis tasks to communicate with the language model asynchronously and the result is returned to the interface when ready [32], [55].

Another important use of background execution in this project is related to the receipt and processing of gaze data. Since the eye scanner continuously produces data, the gaze receipt logic must be able to read the data stream and pass it to the interface layer without interrupting the interface. In Qt, the existence of an event loop in separate threads and the use of signal-slot communication between threads allows background objects to produce or process data and the interface to be updated only when necessary via signals [55], [60], [61]. This feature is crucial for the present system, because gaze-based interaction directly depends on the stability and continuity of the input data [55].

In the Qt architecture, one of the important points is that the QThread object itself lives in its creator thread, not in a new worker thread; therefore, to execute operational logic in a separate thread, using worker objects and moving them to the thread via moveToThread() is a more suitable pattern than directly placing worker slots in QThread subclasses. The official Qt documentation also recommends this worker-object approach for many practical scenarios. The current system implementation is also designed with this logic in mind, so that correction and style analysis tasks are executed in separate worker objects and the interface is notified of their completion via signals [55].

For communication between workers and the interface, the system uses the signal-slot pattern. The Qt and Qt for Python documentation explain that signals and slots are the main mechanisms for exchanging events between objects, and that, especially in the case of inter-thread communication, queued connections allow data and events to be passed safely from one thread to another [60], [61]. The same logic is used in the present system: workers send the

appropriate signal after completing their task, and the user interface updates the busy state, receives results, and refreshes panels or highlights via the corresponding slots [55], [60].

One advantage of this organization is that the main interface thread remains responsible only for displaying and managing the interaction. As a result, operations related to rendering and responding to user events do not compete directly with heavier tasks [54], [55]. This is especially important in the current system, because if the user is typing, gaze is registered, and a language request is being executed, the user experience remains good only if the UI thread is kept as light and free as possible [55].

In addition to starting threads, ending them safely is also an important part of an asynchronous implementation. The Qt documentation recommends using quit() and then wait() to terminate the thread, and freeing objects created in the thread by hooking the finished() signal to deleteLater(). This approach is also important in the current system, because language workers and their corresponding threads must be freed after finishing their work to avoid resource leaks and object lifecycle complexities [55].

From an implementation perspective, asynchronous execution in this project is not just a technical decision to increase speed, but also part of the interactive logic of the system. Because writing assistance in this system must be activated in response to user fixation, any excessive delay or interface locking can disrupt the continuity of the interaction flow. Therefore, background workers and asynchronous execution help the system to remain both technically stable and user-experience-friendly by providing feedback in a timely manner without disrupting the writing process [15], [27], [55].

In general, the implementation of background workers and asynchronous execution in this system is based on three principles: separating time-consuming tasks from the interface thread, using the worker-object pattern for controlled execution, and leveraging signal-slot communication for safe coordination between components [55], [60], [61]. This layer is a key element in the implementation of a gaze-aware system, because without it, the simultaneous combination of the written interface, gaze data flow, and API-based language processing could not be done with appropriate interactive quality [32], [55].

6.10 Logging and Session Recording

Logging and storing session data in the proposed system is an important part of the implementation, because the system is not only designed to provide real-time feedback, but

also to allow for analysis of user behavior and evaluation of system performance [5], [22]. In gaze-aware environments, logging interaction data is of particular importance, since many system decisions are based on temporal events, gaze persistence, and user responses to suggestions [15], [27]. For this reason, in this project, the logging layer is designed independently to store key session data in a structured manner [62].

In the present implementation, data logging is performed in a session-based manner. This means that for each system execution, a session time identifier is created and the output files associated with the same session are named accordingly [63], [64]. Using timestamps to name files and separate sessions allows each session's data to remain independent from other sessions and makes it easier to track them in subsequent analysis steps [63], [64].

The logging structure in this system is based on two separate types of files: event log and gaze log. The first file is used to record important interaction events such as program start, sentence review start, suggestion opening, or session end; while the second file is used to store gaze samples, gaze validity status, recorded coordinates, and fixation-related indices [5], [15], [62]. This separation is logical because events and gaze samples differ in their occurrence rate, meaning, and analytical utility, and keeping them in two separate files simplifies subsequent processing and analysis [62].

The CSV format is used to store these data. The CSV format is widely used for tabular data and structured logs, and the standard Python csv module provides the necessary tools for writing rows of data and managing writer objects. The official Python documentation also recommends that CSV files be opened with newline=' ' when writing to properly control the behavior of writing rows. This format is appropriate for the present project, as it is both easy to quickly view the data and remains usable for later analysis with statistical tools or spreadsheets [62].

In the event log file, each line represents a meaningful system-level event. These events can include the system mode state, the type of engine used, the range of the analyzed sentence, the sentence text, the problem word, the suggested suggestion, and helpful notes. From a research perspective, such information is of great importance, as it allows the system workflow to be reconstructed over time, and it is possible to determine at what moments the system intervened, what kind of feedback was produced, and what the user's reaction was to it [5], [15], [27].

In the gaze log file, data is recorded at the level of gaze instances. This file usually includes the relative time from the start of the session, gaze coordinates at the global and editor level, instance validity, and fixation-related indices [5], [22]. The importance of this type of data is that it can later be analyzed for gaze temporal trends, attentional stability, and the relationship between fixations and linguistic events [15], [27]. As a result, the gaze log is not just a technical output for debugging, but also part of the research data of the system [5], [22].

One of the important points in this implementation is the use of relative session time instead of relying on the absolute system time of each event. In this approach, an initial time reference is defined at the start of each session, and all events and gaze samples are then measured in milliseconds from that point [64]. This approach is very useful for interaction flow analysis, as it simplifies direct comparisons between the timing of events, fixations, and system responses, and reduces the dependence of the analysis on the absolute time of the system clock [15], [64].

At the file implementation level, using pathlib.Path to create the logs folder and generate file paths makes path management clearer and more robust. The official pathlib documentation shows that this module provides an object-oriented interface for managing filesystem paths and transparently supports the creation of directories with options such as parents=True and exist_ok=True. This choice is appropriate for the present project, because the system must ensure that the log storage folder exists at the beginning of each run and then create files related to that session in it [63].

From a research design perspective, the presence of this logging layer also has another important advantage: the possibility of post-interaction analysis. Without recording events and gaze data, the evaluation of the system would be limited to observing its behavior at the moment; but with structured storage of events and gaze samples, it is possible to later examine where fixations occurred, which sentences were most frequently triggered, what types of suggestions were most frequently displayed, and under what circumstances the user engaged with the system’s feedback [5], [15], [27]. This is especially important for the evaluation chapter, because part of the research argument can rely on the data recorded in this layer [5], [22].

In summary, the implementation of logging and session recording in this system can be considered to involve several main steps: creating a storage folder, generating a timestamp for the session, creating separate CSV files for events and gaze, recording the relative time of each

row, and flushing the data to maintain file integrity during execution [62], [63], [64]. This layer is an important component of the prototype, as it provides a structured view of system and user behavior for both debugging and research analysis [5], [22], [27].

6.11 Error Handling and Robustness

In the proposed system, error handling is not just a minor technical consideration, but an essential part of the implementation, as the present system simultaneously deals with several potential sources of error, including communication with the eye tracker, processing gaze data, executing background threads, and API-based requests to the language model [32], [55], [56]. Since exception handling in Python is done through the try/except construct and unhandled errors are propagated to higher callers, the system is designed to contain and report errors as much as possible at the appropriate layer [65], [66].

One of the first critical points for error handling is the communication layer with the eye tracker. In this section, problems such as Gazepoint server unavailability, socket disconnection, incomplete data reception, or invalid XML can disrupt the gaze flow [56]. For this reason, in the current implementation, the data acquisition logic is placed in a separate component and communication errors are handled in the same layer so that a failure in the gaze source does not directly cause the entire interface to fail. This approach is architecturally important, since a failure in the hardware or network connection should not lead to a UI lockup or a complete loss of the session [55], [56].

Another critical point is the integration with the language model. API requests may encounter errors such as request failures, rate limits, validation problems, or server-side errors. The official OpenAI documentation for the API and the official Python library also explicitly describe a set of error codes and failure modes for these scenarios [32], [67], [68]. For this reason, in the present system, language model calls are executed in protected blocks so that in case of an error, the system can report the status to the interface in a controlled manner instead of stopping abruptly and maintain the interaction flow as much as possible [32], [67].

In this system, error handling in the LLM layer is not limited to catching exceptions, but also includes validating the response structure. Even if the API request completes successfully, the model response may not exactly match the system's expectations in terms of format or content. Since the system requires structured outputs for correction and style analysis, the received response should be checked for the presence of necessary fields, overall format,

and parsing capability [30], [32]. The Structured Outputs documentation also emphasizes the importance of structured formatting to reduce subsequent errors in the software layer [32]. Therefore, the robustness of this section depends on the combination of prompt design, robust parsing, and fallback handling, not just the success of the API call [30], [32].

From a user interface perspective, one of the important principles in this project was that errors should be understandable to the user, but not destructively interfere with their workflow. For this reason, in the current implementation, errors related to correction or style analysis are reflected through UI mechanisms such as status messages or suggestions areas, rather than by stopping the application or abruptly closing the window. This decision is consistent with the logic of interactive systems, because in a writing assistant, a failure in a request should not disable the entire writing environment [27], [33].

Error handling in the background threads and workers section is also of great importance. In Qt, it is recommended to run worker objects in separate threads and safely clean them up after they are finished using the finished() and deleteLater() signals [55], [69], [70]. In the present system, the same pattern is used for language workers and their corresponding threads, both to avoid blocking the execution of background UI tasks and to safely manage the lifecycle of objects [55], [69]. This is part of the robustness of the system, since many practical failures do not arise from the task itself, but from improper thread management and improper cleanup of objects [55], [70].

Another aspect of robustness in this project is tolerance to incomplete or unstable gaze data. As explained in the previous sections, eye tracking data is always accompanied by noise and invalid samples [19], [20]. For this reason, the system implementation is done in such a way that invalid gaze samples are not directly used as the basis for interaction, and the sentence detection and fixation logic is activated only in the presence of valid and relatively stable data [11], [19]. This decision can be considered a kind of error handling at the data level, because instead of assuming every input is valid, the system screens the input before using it [19], [20].

The logging layer also plays a role in increasing robustness. Recording important events and states of the system helps to reconstruct the sequence of events in the event of a failure and to understand in which part of the pipeline the error occurred [5], [62]. Therefore, logging in the present system is not only useful for analyzing user behavior, but also for debugging and monitoring the health of the system execution. Such an approach is consistent with common best practices in developing interactive prototypes, because in multi-part systems, the ability

to observe the internal behavior of the system is an important prerequisite for debugging and improving stability [62], [68].

In summary, the error handling and robustness strategy in this system can be summarized in a few principles: error containment at the appropriate layer, isolation of communication failures from the UI, validation of model responses, safe cleanup of threads and workers, neutralization of invalid gaze data, and structured logging of events for later analysis [32], [55], [62], [67]. These principles make the system more robust against common failures at the interface, eye tracker, and API levels while providing a more consistent and reliable user experience [27], [33], [68].

6.12 Chapter Summary

In this chapter, the implementation of the proposed system is examined from a software and implementation architecture perspective. First, it is shown that the system is implemented as a modular desktop application in which the user interface, gaze data acquisition, interaction logic, and language processing based on large language models are integrated into a single environment. It is also explained that the selection of the main technologies of the system, including Python, PyQt6/Qt for Python, Gazepoint/Open Gaze API, and OpenAI API, was made with the aim of supporting real-time interaction, modular development, and the possibility of implementing a research prototype [32], [54], [55], [56].

Next, the project structure and the organization of the main files and modules are described. In this section, it was shown that the project was designed to separate components related to configuration, data models, interaction recording, gaze tracking, user interface, and language processing from each other to improve readability, extensibility, and maintainability of the system [32], [54], [55]. This type of organization showed that the system implementation was not based on simply collecting several features in a single file, but rather on separating responsibilities and reducing dependencies between components [32], [55].

In the user interface section, it was explained how QMainWindow, QPlainTextEdit, custom QWidget, QScrollArea, and the signal-slot mechanism in Qt were used to create an interactive environment suitable for writing, gaze observation, and receiving language feedback [54], [57], [58], [59], [60], [61]. This section showed that the UI of the system is not just a simple presentation layer, but also forms the main platform for interaction between the user, gaze, and system suggestions [54], [57].

Next, the gaze data processing layer and the sentence and fixation recognition logic were examined. This section explained how raw eye tracker data is received, validated, and mapped to interface coordinates, and then interpreted at the sentence-level [4], [19], [20], [56]. It was also shown that fixation detection in this system is based on gaze persistence on a meaningful text area and not just the instantaneous gaze position; a decision that is consistent with the inherent limitations of eye tracking data and the usability requirements of gaze-based interaction [11], [15], [17], [18], [21].

In the section on language model integration, it was explained how the system uses an interface layer to construct the prompt, send the request to the API, receive the response, validate the output structure, and prepare it for display in the interface [23], [30], [32], [57], [67]. It was also shown that the model output is processed in a structured manner so that corrections, issues, and style suggestions can be reliably applied in the system pipeline [30], [32]. This section showed that the integration of LLM into the system is not limited to calling an external service, but also includes careful design of prompts, robust parsing, and interactive rendering of the output [26], [27], [28], [29], [32].

Next, the issue detection and highlighting layer, as well as the background worker and asynchronous execution mechanisms, were described. In these sections, it was shown how the system converts the model response into writable issues in the text, highlights them in the editor, and displays relevant suggestions in the feedback panel [27], [28], [29], [54]. It was also explained that to prevent the user interface from locking up, time-consuming processes such as language requests and some continuous tasks are executed in separate threads via QThread and worker objects, and their communication with the UI is managed via signal-slot [55], [60], [61], [70].

In the final sections, logging and session recording, as well as error handling and robustness strategies, were examined. In this section, it was shown that the system records gaze data and key interaction events in CSV files on a session basis to enable both debugging and exploratory analysis of system and user behavior [62], [63], [64]. It was also shown that to increase robustness, errors related to the eye tracker, API, output parsing, and threads are contained in appropriate layers, and the system attempts to report the status to the interface in a controlled manner instead of stopping completely in the event of a failure [32], [55], [56], [65], [66], [67], [68].

Overall, this chapter showed that the proposed system is realized as a multi-modular, event-driven, and gaze-aware system not only in terms of concept but also in terms of implementation [32], [54], [55]. This implementation provides the necessary framework for evaluating system behavior, analyzing user interaction with language suggestions, and investigating the role of gaze in activating writing assistance [15], [27], [48]. Accordingly, in the next chapter, the system performance, output samples, and its behavior in different scenarios will be examined to show how this architecture and implementation work in practice [5], [22], [27].

Chapter 7 – Evaluation

7.1 Evaluation Strategy

The evaluation of the proposed system in this study is designed based on a behavioral and qualitative evaluation strategy, given the nature of the system prototype and the current scope of its development. The aim of this chapter is not to evaluate the system in the form of a large-scale usability experiment or a statistical study with a large number of participants, but rather to show how the system behaves in practice, whether the gaze-triggered interaction logic works correctly, and whether the language outputs produced are meaningful and usable for the target scenarios [27], [48].

In many research prototypes in the field of HCI and AI-based systems, especially in the early stages of development, the evaluation can rely on system behavior, interaction traces, and illustrative examples to clarify the basic functionality of the system before conducting more extensive usability studies [5], [15], [22]. This approach is also appropriate in this thesis, because the present system is at the intersection of eye tracking, interaction design, and LLM-based writing assistance, and before any extensive evaluation, it must be shown that its main chain of performance from fixation detection to linguistic feedback operates coherently [15], [23], [48].

Accordingly, the evaluation strategy of this research is based on four main axes. The first axis is to examine the overall behavior of the system in practical scenarios; that is, whether the system can receive text, interpret the user's gaze at the sentence level, activate linguistic analysis at the appropriate time, and display feedback in an understandable manner [15], [22], [48]. The second axis is to analyze correction and rewriting samples to examine the quality and type of linguistic outputs of the system and to clarify in which cases the system provides correction, suggestion, or style assistance [26], [27], [28]. The third axis is the analysis of logs and recorded interaction data to reconstruct the actual flow of interaction between the user and the system from a temporal and behavioral perspective [5], [27], [62]. The fourth axis is a critical discussion of the strengths and limitations so that the system performance is not presented in a purely descriptive manner, but also its current limitations are clearly stated [23], [33].

An important feature of this evaluation strategy is that it performs the evaluation at the interaction loop level, not just at the level of a single module. In other words, the goal is not

just to check the accuracy of a fixation detection algorithm or the correctness of a linguistic response in isolation, but to determine whether the entire cycle gaze → sentence detection → LLM analysis → feedback presentation → user response functions properly as an integrated system [15], [23], [48]. This view is consistent with the nature of the system itself, because the main value of the system lies in the coordination between the components and the overall interaction experience, not simply in the separate performance of each component [5], [22].

In this study, the evaluation is carried out as a combination of qualitative analysis and behavioral observation of the system. Qualitative analysis is suitable for examining the nature of linguistic feedback, the clarity of suggestions, and the fit of the feedback provided to the sentence of interest to the user [26], [27], [29]. On the other hand, behavioral observation of the system and examination of logs help to examine the timing of intervention, the sequence of events, and the role of fixation in activating analysis more objectively [5], [15], [62]. This combination is suitable for the present prototype, because on the one hand it pays attention to the linguistic quality of the outputs and, on the other hand, it does not ignore the interaction logic of the system [27], [48].

Another important point is that this evaluation chapter is designed based on realistic claims. That is, the goal is not to prove that the system is better than all existing writing assistants or that its performance is statistically generalizable to all users and contexts. Rather, the goal is to show that the present prototype is conceptually and practically feasible, that its main components work in harmony, and that gaze-based interaction can be used as a basis for intelligent writing assistance [15], [27], [33]. Such an approach is more appropriate and honest for a thesis that focuses on the design and implementation of an innovative system than extensive empirical claims without sufficient support [23], [33].

Also, the present evaluation strategy is designed with respect to the data available in the system itself. Since the system generates event logs and gaze logs and can record real examples of corrections and suggestions, the evaluation chapter can rely on evidence obtained directly from the prototype's performance [27], [62]. This allows the evaluation to be based on real artifacts from the system's implementation, rather than on general descriptions; artifacts such as recorded events, examples of gaze-triggered activation, and real examples of linguistic feedback [5], [27], [62].

Accordingly, the structure of this chapter will be as follows: first, the general behavior of the system is described, then examples of correction and style assistance output are

examined, then the interaction logs and traces are analyzed, and finally, a critical discussion of the system's limitations and capabilities is presented [27], [48]. This structure helps the evaluation go beyond the level of simple observation and cover both the technical aspect of the interaction and the linguistic aspect of the outputs [15], [23].

In summary, the evaluation strategy of this research can be described as a prototype-oriented evaluation framework that relies on real system behavior, practical output examples, and recorded interaction data [5], [22], [62]. Rather than being limited to quantitative criteria, this framework attempts to show how the proposed system performs in terms of interaction design, integration of gaze, and writing assistance, and to what extent it implements the main idea of the thesis in practice [15], [27], [48].

7.2 System Behavior

In this section, the overall behavior of the proposed system is examined in practical scenarios to show how the different components of the system work together and whether the main gaze-based interaction cycle is implemented coherently. The aim of this section is to provide a behavioral analysis of the system, rather than a numerical or statistical evaluation of it, and to focus on how the system reacts in practice and what kind of interactions it forms [5], [15], [22].

In a typical scenario, the user enters their text in the editor and the system simultaneously receives and processes gaze data. At this stage, the system continuously monitors the user's gaze position relative to the text, but does not intervene until the gaze reaches a level of stability (fixation). This behavior indicates that the system is non-intrusive by default and only enters the analysis phase if there is a meaningful indication of the user's attention [15], [17], [18].

When the user holds their gaze on a sentence for a certain period of time, the system identifies it as a fixation and enters the linguistic analysis phase. At this moment, the sentence in question is sent to the language model module and, after receiving a response, the results are displayed as highlights in the text and suggestions in the side panel. This process indicates the successful activation of the gaze-triggered interaction cycle, in which the user's gaze acts as the main trigger for activating writing assistance [15], [48], [53].

One of the important features of the system's behavior is that interventions are performed in a context-sensitive manner. In other words, the system only analyzes the sentence that the user is actually focused on and refrains from analyzing the entire text or providing

broad, untargeted suggestions. This behavior is consistent with the design principles of gaze-based systems, since reducing unnecessary interventions is one of the main strategies to prevent the Midas Touch phenomenon and increase usability [15], [19].

Examining the behavior of the system, it is observed that the timing of the intervention plays a key role in the user experience. If the fixation threshold is set correctly, the system reacts neither too quickly nor too late. This balance makes the user feel that the system understands his attention, without disrupting the natural writing interaction. Conversely, if the thresholds are not set correctly, the system may either over-activate or not react in time; Therefore, the behavior of the system is directly dependent on its temporal settings [11], [17], [21].

The behavior of the system in the face of rapid gaze changes is also noteworthy. When the user shifts his gaze rapidly between different sentences, the system usually refrains from activating linguistic analysis, because the gaze stability condition is not met. This behavior indicates that the system has been able to distinguish between a fleeting glance and meaningful attention and avoids reacting to noisy or unstable signals [11], [19], [20].

At the presentation level, the system attempts to provide feedback simultaneously in the text and in the side panel. In-text highlighting helps the user to quickly identify the location of the problem, while the suggestion panel provides more detailed explanations and correction options. This combination allows the user to interact with the system's feedback without leaving the original context of the text [27], [28].

Another important aspect of the system's behavior is the maintenance of the active role of the user (human-in-the-loop). Even after providing linguistic suggestions, the system does not automatically apply any changes to the text, and the final decision always remains with the user. This behavior is important in terms of usability and reliability, because the limitations of linguistic models require that their output be considered as suggestions, not definitive decisions [27], [33].

Differences in behavior are also observed in the different operating modes of the system. In manual mode, the user explicitly requests analysis, while in gaze-triggered mode, this process is performed automatically and based on visual attention. Comparing these two modes shows that the system can function both as a traditional writing assistant tool and as a gaze-based system, which increases its flexibility [15], [22].

Overall, the analysis of the system behavior shows that the main interaction cycle including attention detection, analysis activation, feedback generation, and user interaction with suggestions operates coherently. This behavior indicates that the main idea of the system, namely using gaze as a stimulus for writing assistance, has been practically realized at the implementation level [15], [48], [53]. However, as will be shown in the following sections, the performance of the system still depends on factors such as the quality of gaze data, parameter tuning, and the accuracy of the language model output, and this will be discussed in the subsequent analyses [19], [23].

7.3 Correction, Style, and Simplification Examples

In this section, examples of real system outputs are presented and analyzed to show what kind of linguistic feedback the system produces in practice and to what extent these feedbacks are aligned with the goal of a gaze-based writing assistance system. Unlike the previous section, which focused on the overall behavior of the system, this section focuses on the nature, quality, and variety of linguistic outputs [27], [28], [29].

The proposed system provides three main types of feedback:

- Grammar correction
- Style refinement/rewriting
- Sentence simplification

These three capabilities together demonstrate that the system is not just a grammar checker, but a multi-purpose writing assistant that can improve the text in various aspects [26], [28].

7.3.1 Grammar Correction Examples

In this section, the system's performance in correcting grammatical errors at different levels is examined to show how the system deals with simple to relatively complex errors.

Simple Example

Original sentence:

He go home.

System output:

He goes home.

Identified issue:

subject–verb agreement

This example shows that the system is able to correctly identify and correct basic and common errors in simple sentences. Such errors are very common among language learners and their accurate correction plays an important role in the usability of the system [26], [27]. A simple example of grammar correction is shown in Figure 7.1.

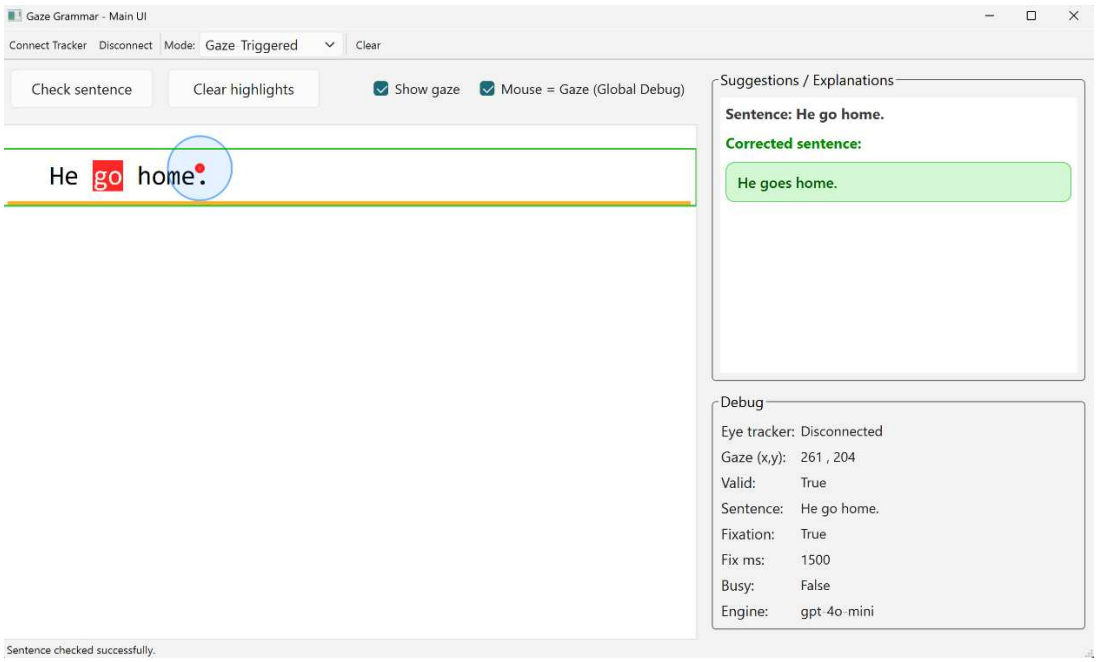

Figure 7.1: Simple example of grammar correction in the system interface.

Intermediate Example

Original sentence:

I have done my homework yesterday.

System output:

I did my homework yesterday.

Identified issue:

incorrect verb form

In this example, the system has identified an error related to incorrect verb tense and converted the sentence to the correct form. This type of correction shows that the system can handle temporal relations and more complex structures to some extent [27], [29]. An intermediate example of grammar correction is shown in Figure 7.2.

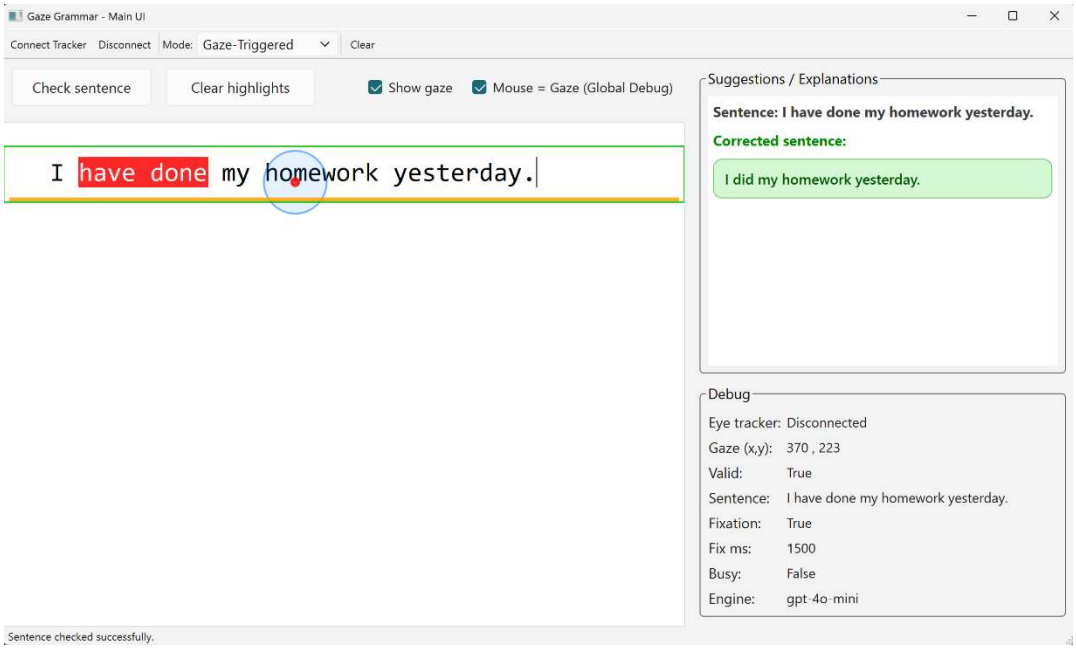

Figure 7.2: Intermediate example of grammar correction in the system interface.

Common Usage Example

Original sentence:

She goes to university every day.

System output:

She goes to university every day.

This example shows that the system performs reliably and consistently when faced with sentences commonly used in everyday writing [26], [28]. An example of correct sentence handling is shown in Figure 7.3.

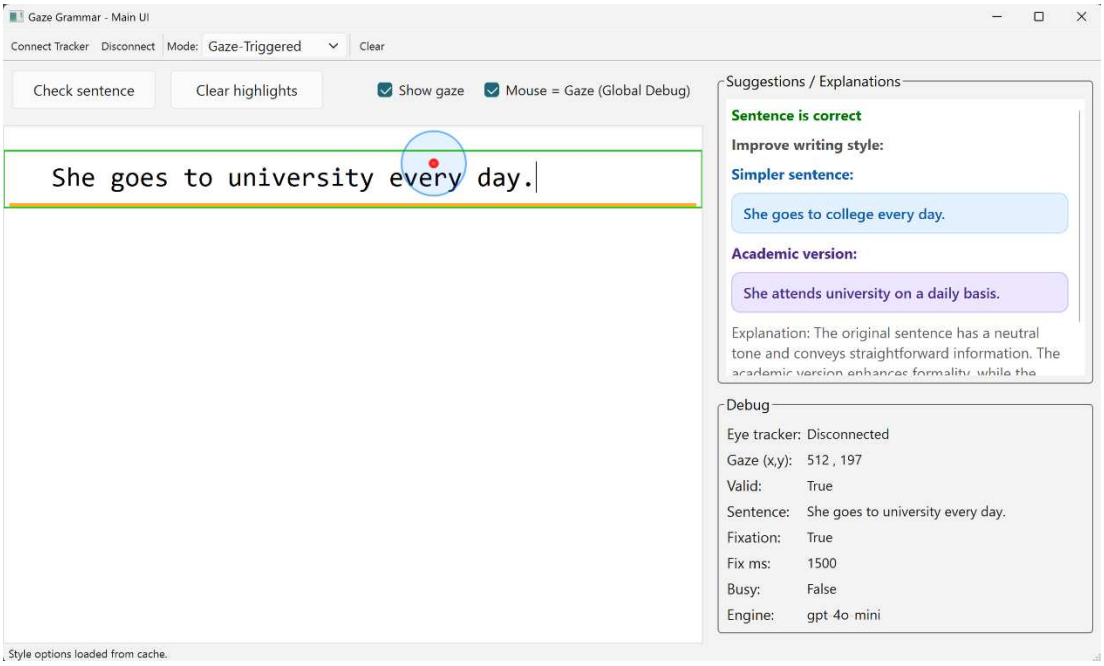

Figure 7.3: Example of correct sentence handling and style suggestions in the system interface.

7.3.2 Style and Rewriting Examples

In this section, the system's ability to improve writing style is examined. In this case, the system's goal is not simply to correct errors, but to transform the sentence into a more formal, precise, or academic form [28], [29].

Example 1 (Formalization)

Original sentence:

Many people think this method is not very good.

System output:

A significant number of individuals perceive that this method is suboptimal.

In this example, the system has transformed a vague and colloquial sentence into a more formal and precise expression. This type of rewriting is of great importance for scientific texts [28], [29]. An example of sentence formalization is shown in Figure 7.4.

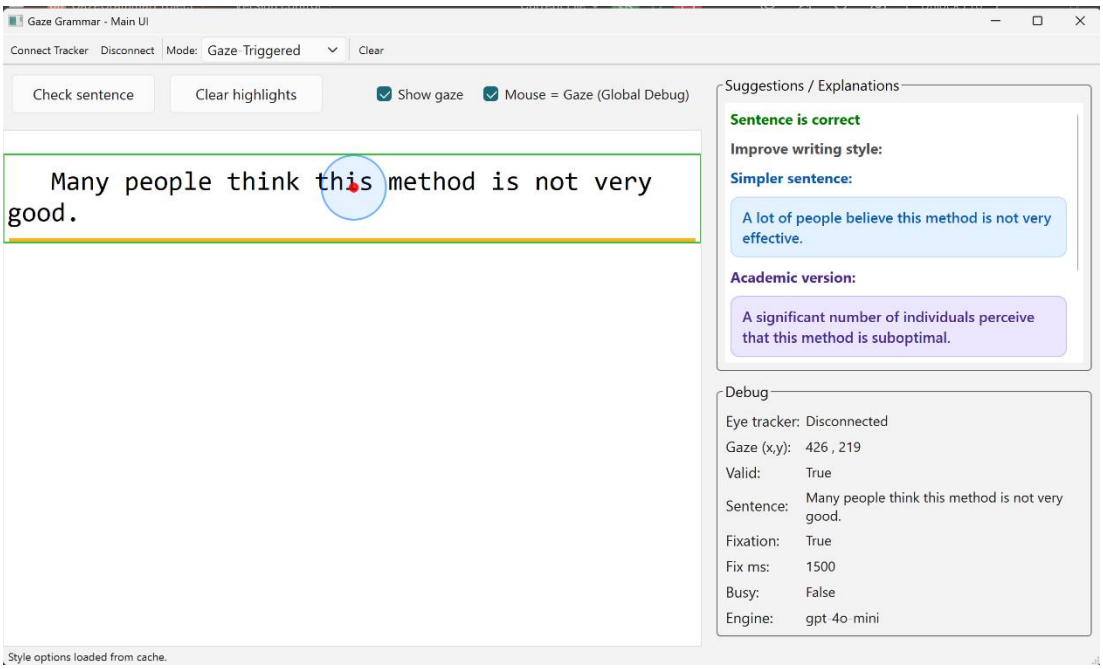

Figure 7.4: Example of sentence formalization in the system interface, showing the transformation of an informal sentence into a more formal and precise expression.

Example 2 (Tone Improvement)

Original sentence:

I think this method is quite useful.

System output:

This method demonstrates significant utility.

In this example, the system transforms an informal and uncertain tone into a more professional and objective form suitable for scientific writing [26], [28]. These examples demonstrate that the system is capable of handling not only grammatical corrections, but also higher-level writing improvements such as tone adjustment and formalization. An example of tone improvement is shown in Figure 7.5.

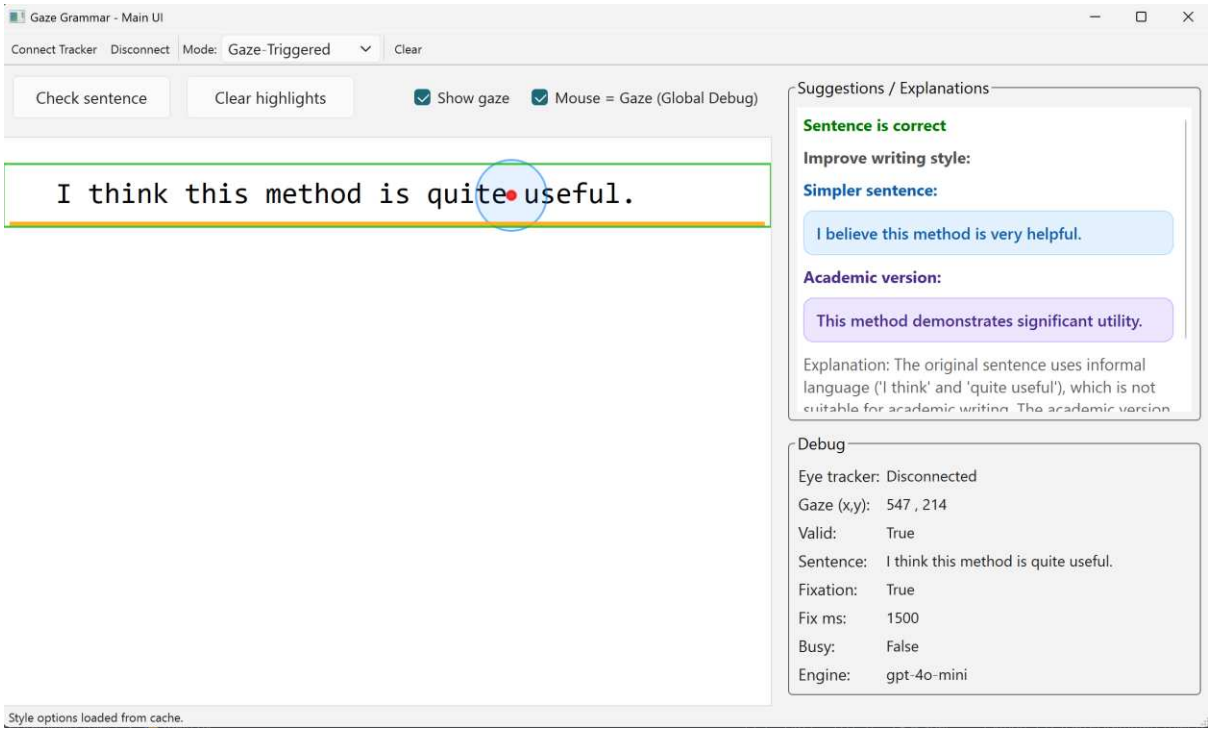

Figure 7.5: Example of tone improvement in the system interface, showing the transformation of an informal and subjective sentence into a more formal expression.

7.3.3 Sentence Simplification Examples

In addition to formal correction and rewriting, the system is also able to provide a simpler and more understandable version of the sentence. This feature is especially important for increasing readability and helping users with lower language levels [28], [29].

Example 1 (Simplification)

Original sentence:

The implementation of this method is significantly complicated.

Simplified output:

Using this method is quite complicated.

In this example, the system simplifies a complex sentence structure into a more direct and readable form. An example of sentence simplification is shown in Figure 7.6.

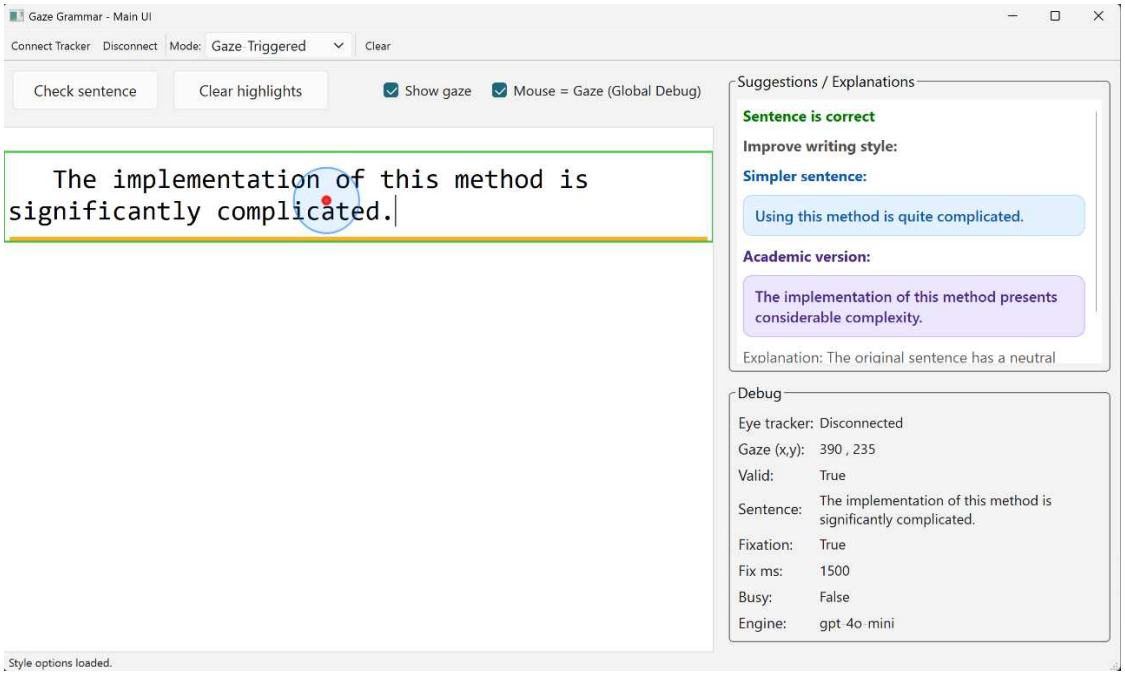

Figure 7.6: Example of sentence simplification in the system interface, illustrating the transformation of a complex sentence into a simpler and more readable form.

Example 2 (Simplification)

Original sentence:

It is essential to take into account all possible factors.

Simplified output:

You need to think about all the possible factors.

In this example, the system simplifies a longer and more complex expression into a shorter and more natural form, improving readability and fluency. Another example of sentence simplification is shown in Figure 7.7.

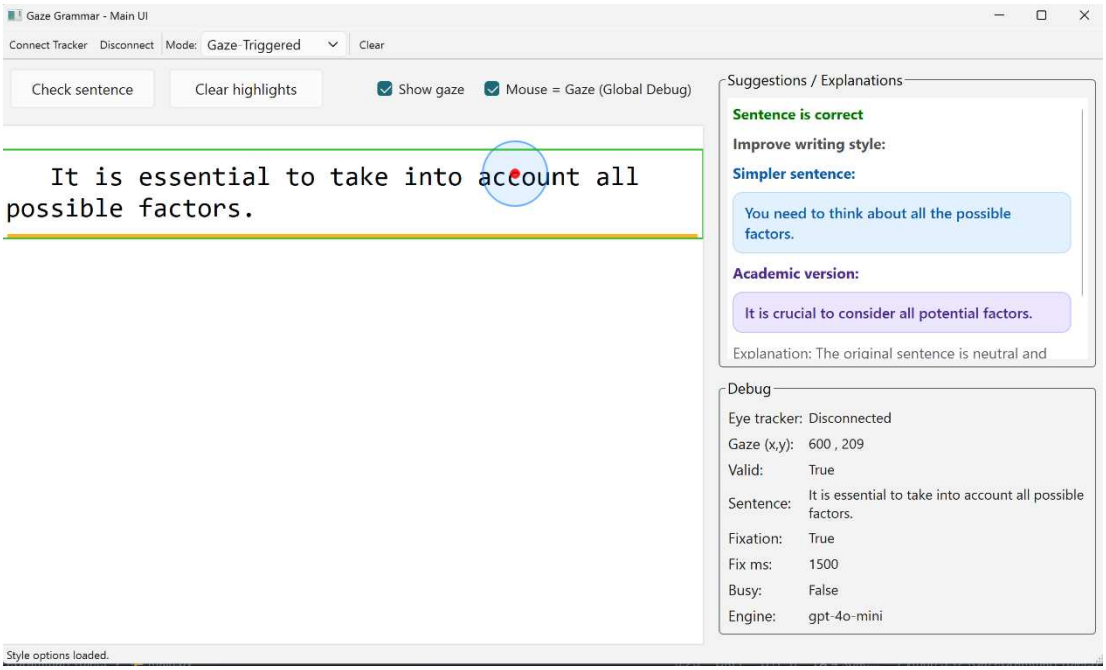

Figure 7.7: Example of sentence simplification in the system interface, illustrating the transformation of a longer and more complex sentence into a shorter and more natural expression.

7.3.4 Overall Analysis of System Output

By examining the examples provided, several key features of the system output can be observed.

First, the system performs reasonably well in identifying and correcting common grammatical errors, especially in simple sentences and common structures. This feature indicates its suitability for basic writing assistance applications [27], [29].

Second, in the style refinement section, the system is able to increase the level of formality and clarity of the text. This is especially important for formal writing, since precise and professional expression is one of the main requirements of this type of text [28], [29].

Third, the simplification capability shows that the system can help improve readability in addition to correction and rewriting. This feature expands the scope of the system and makes it a suitable tool for users with different needs [26], [28].

However, the system outputs still depend on the nature of the language models and may in some cases:

- make minor changes in meaning
- or change the level of sentence certainty

which is a well-known limitation of LLMs [23], [33].

Taken together, these examples show that the proposed system is able to provide three complementary types of linguistic feedback:

correction, refinement, and simplification, which together provide a relatively complete writing assistance experience at the sentence level [27], [28], [29].

7.4 Interaction Logs Analysis

In this section, the recorded data of the system, including the event log and gaze log, are analyzed to objectively examine the actual behavior of the system and how the user interacts with it. Unlike the previous sections that focused on selected examples, this section uses real data recorded during the execution of the system to reconstruct the interaction flow more accurately [5], [15], [62].

The goal of this analysis is to answer the following questions:

- Does the system activate at the right time?
- Do fixations actually trigger the analysis?
- What is the sequence of interaction between the user and the system?

This type of analysis is of great importance for gaze-based systems, because many key behaviors of the system can only be observed through recorded and temporal data [15], [22].

7.4.1 Structure of Logged Data

As explained in the previous chapter, the system records two main types of data:

Event Log

Including high-level events such as:

- Session start and end
- Fixation detection
- Request sent to the language model
- Response received
- Suggestion display

Gaze Log

Including continuous gaze data:

- Time (relative time)
- Gaze coordinates
- Validity status (valid / invalid)
- Fixation-related information

These two types of data together allow for the simultaneous analysis of user behavior and system response [5], [62].

7.4.2 Temporal Interaction Flow

One of the most important results of log analysis is the reconstruction of the interaction time sequence. A simple example of this sequence is as follows:

1. User starts typing
2. User's gaze is fixed on a sentence
3. System detects fixation
4. Request is sent to language model
5. Response is received
6. Suggestion is displayed in UI

This sequence shows that the system has been able to implement the full attention → analysis → feedback cycle in practice [15], [48].

Log analysis shows that the time interval between different steps (especially between fixation and response display) plays an important role in the user experience. If this interval is too long, the interaction flow will be disrupted; and if it is too short, the system may appear overactive [11], [17].

7.4.3 Fixation-Triggered Events

One of the main goals of this analysis is to investigate whether fixations actually lead to meaningful events in the system.

By examining the logs, it is observed that:

- Not all gazes trigger the system
- Only gazes that pass a time threshold are considered fixations
- Only these fixations trigger the analysis

This behavior indicates that the system is able to distinguish between:

- glance (passive glance)
- attention (real attention)

which is one of the main challenges in gaze-based systems [11], [19], [20].

7.4.4 Role of Logging in Evaluation

The existence of log data allows the evaluation of a system to be based on:

- Real evidence
- Not just subjective descriptions

For example, it is possible to examine:

- How many fixations led to analysis
- How long it took for the answer to be displayed
- Under what circumstances did the user ignore suggestions

This type of analysis is important for interactive systems, because many of their features can only be observed over time and in real context [5], [27], [62].

7.5 Discussion

In this section, the performance of the proposed system is critically examined in relation to the results presented in this chapter. The aim of this section is to analyze the behavior of the system and interpret the results obtained, rather than simply to replicate them, in order to determine how the system works in practice and what factors affect its performance [27], [33].

One of the main findings of this research is that the use of gaze as an interaction signal can lead to a natural and context-sensitive interaction. As observed in the analysis of the system behavior and interaction logs, the system is activated only when the user focuses his attention on a sentence, avoiding unnecessary interventions. This feature makes the system compatible with the natural flow of writing and causes less disturbance to the user [15], [48].

However, log analysis shows that the system performance is highly dependent on the timing of the thresholds and the quality of the gaze data. In cases where the gaze data is unstable or fixation is not detected correctly, the system may activate later than expected or miss some intervention opportunities. This indicates that gaze processing is a critical part of the system and plays an important role in the user experience [11], [19].

From the interaction perspective, it was observed that the system was able to distinguish between fleeting glances and meaningful attention, which is important to prevent unwanted activations. This behavior indicates that using fixation as an activation criterion was a suitable choice for the interaction design in this system [11], [20].

In the linguistic output section, the analysis of the examples also showed that the system is able to provide feedback at three different levels: grammatical correction, style improvement, and sentence simplification. This variety in feedback allows the system to cover different needs of users. However, the quality of these outputs is not uniform in all cases and depends on the sentence structure and its complexity [27], [28].

Also, in some cases it is observed that the provided rewrites may cause slight changes in the meaning or level of certainty of the sentence. This indicates the influence of the probabilistic nature of language models on the system output and highlights the importance of the user's role in the final decision-making [23], [33].

Another notable aspect is the impact of the system's response time (latency) on the user experience. Interaction flow analysis shows that the delay between fixation detection and suggestion display can affect the continuity of the interaction. If this delay is too long, the user's mental connection with the sentence in question may decrease, which highlights the importance of optimizing response time [17], [32].

Overall, the analysis of the results in this chapter shows that the proposed system has been able to implement a complete cycle of attention detection → language analysis → feedback in a practical and coherent manner. However, its performance depends on a set of factors, including the quality of gaze data, parameter tuning, and the behavior of the language model.

These findings indicate that although the gaze-triggered writing assistance approach is conceptually and practically feasible, the design of such systems requires careful attention to the details of interaction, timing, and input data quality to achieve a desirable user experience [15], [48].

Chapter 8 – Conclusion

8.1 Summary of the Work

This thesis designs, implements, and evaluates an innovative writing assistance system based on gaze interaction and large-scale language models. The main goal of this research was to investigate whether the user's visual attention can be used as an implicit signal to activate language analysis, thereby creating a more natural and less intrusive interactive experience in the writing process [15], [27], [48].

In the initial chapters, the theoretical foundations related to eye tracking, human-computer interaction, and large-scale language models were reviewed and it was shown that each of these areas has significant capabilities on its own, but their combination can create a new direction for the design of intelligent interactive systems [5], [22], [23]. Accordingly, this study presented a conceptual framework for gaze-triggered writing assistance, in which the user's gaze is considered as the main trigger for text analysis [15], [48].

Subsequently, this conceptual framework was transformed into an operational system (prototype). The system implementation included the development of a PyQt-based writing environment, a module for receiving and processing gaze data from the eye tracker, and an integration layer with language models for correction, rewriting, and simplification. In this architecture, the different components of the system were designed in a modular manner to manage the interaction between the user interface, gaze data, and language processing in a coherent manner [32], [54], [55].

One of the main innovations of this system was the use of sentence-level fixation detection to activate language analysis. In this approach, the system only enters the analysis phase when the user's gaze is stably focused on a sentence. This design decision allowed the system to avoid unnecessary interventions and provide an experience more consistent with the user's natural behavior [11], [17], [19].

In the implementation section, it was also shown how the use of background workers and asynchronous execution allows for maintaining the responsiveness of the user interface, and how the logging structure and interaction data recording can be used to analyze system and user behavior [55], [62]. These features allowed the system to function as a real interactive system, not just a conceptual model.

In the evaluation section, the system performance was studied through system behavior analysis, output sample review, and interaction log analysis. The results showed that the system is able to:

- Detect user attention at the sentence level
- Activate linguistic analysis at the appropriate time
- And provide understandable and useful feedback

Also, the examples presented showed that the system can produce three complementary types of feedback:

Grammar correction, style improvement, and sentence simplification, which together provide a relatively complete writing assistance experience [27], [28], [29].

Overall, this research showed that the combination of eye tracking and language models can lead to the creation of a new type of interaction in writing systems; one in which the system uses implicit signals of the user's attention to provide assistance, rather than waiting for an explicit command. This approach could be a step towards designing smarter, more adaptable, and more user-centric systems in the field of human-computer interaction [15], [48].

8.2 Contributions

This research presents a set of contributions in the fields of human-computer interaction, gaze-based systems, and the application of language models to writing assistance. These contributions can be summarized in four main categories.

8.2.1 Conceptual Contribution

The first and most important contribution of this research is the presentation of a new conceptual framework for gaze-triggered writing assistance. In this framework, instead of using explicit inputs such as clicks or direct commands, the user's visual attention is considered as the main signal to activate linguistic analysis [15], [48].

This idea creates an important change in the way interactive systems are designed, as interaction is shifted from explicit to implicit interaction. Such an approach can lead to the design of systems that are more natural, less intrusive, and more consistent with the real behavior of users [15].

8.2.2 System Design Contribution

The second contribution of this research is the design of an integrated architecture to combine:

- gaze data acquisition and processing
- sentence-level attention detection logic
- and language processing based on large language models

In this architecture, the different components of the system are organized in a modular manner to manage the interaction between them in a controllable and scalable manner [32], [55].

One of the important aspects of this design is the use of sentence-level fixation detection as the main mechanism for activating the analysis, which provides a balance between accuracy, simplicity, and usability [11], [19].

8.2.3 Implementation Contribution

The third contribution of this research is the development of an operational and implementable prototype of the proposed system. This system consists of:

- an interactive writing environment
- a gaze tracking module
- and an integration layer with language models

which together form a complete writing assistance system.

In this implementation, techniques such as:

- asynchronous execution
- the use of background workers
- and structured logging

are used to create a stable and responsive system [55], [62].

This prototype shows that the proposed idea can be implemented and used not only at the conceptual level, but also in practice.

8.2.4 Functional Contribution

The fourth contribution of this research is the presentation of a multi-purpose writing assistance system that provides three main types of linguistic feedback:

- Grammar correction
- Style refinement (formal rewriting)
- Sentence simplification

This combination makes the system go beyond a grammar correction tool and become a more comprehensive writing assistant that can help improve the quality, clarity, and readability of the text [26], [28].

8.2.5 Empirical Contribution

Finally, this research provides a set of empirical evidence based on the system's behavior that shows how the system works in practice. This evidence includes:

- System behavior analysis
- Examination of output samples
- And analysis of interaction logs

which together show that the gaze-triggered interaction cycle is feasible in practice [27], [48].

Although this evaluation was conducted on a limited scale, the results demonstrate the feasibility and applicability of this approach in future interactive systems [27].

8.2.6 Summary of Contributions

In summary, the main contributions of this research can be summarized as follows:

- Proposing a new paradigm for gaze-based interaction in writing assistance
- Designing an integrated architecture for combining gaze and language models
- Developing an operational and workable prototype
- Providing a multipurpose system for text correction, rewriting, and simplification
- Providing empirical evidence of the system's performance in real-world scenarios

These contributions collectively demonstrate that using gaze as an interactive signal can open a new path for designing intelligent and user-centered systems in the writing domain [15], [48].

8.3 Limitations

Despite the positive results obtained, the proposed system has limitations that should be considered in the interpretation of the results and future applications.

First, the performance of the system is directly dependent on the quality of the gaze data. Since eye tracking data is inherently noisy and uncertain, the accuracy of fixation detection and, consequently, the timing of the intervention may vary under different conditions [19], [20]. This can affect the user experience and reliability of the system.

Second, the system in its current version operates at the sentence-level processing level and the broader context of the text, such as the paragraph or the overall structure of the document, is not considered in the linguistic analysis. This limitation can in some cases reduce the accuracy or fit of the suggestions to the overall meaning of the text [28], [29].

Third, the linguistic part of the system is completely dependent on large language models. These models are probabilistic in nature and may in some cases produce incorrect, ambiguous, or unintended changes in meaning [23], [33]. Therefore, the suggestions presented should always be considered as suggestions, not definitive decisions.

Fourth, the dependence on external services (APIs) can cause response delays or access restrictions. This can affect the flow of interaction, especially in environments with unstable network connectivity [32], [68].

Finally, the evaluation conducted in this research is mainly qualitative and based on system behavior analysis and does not include extensive user studies or statistical analyses. Therefore, the results presented are more indicative of the feasibility and initial performance of the system, and their generalization requires further empirical studies [27].

Overall, these limitations indicate that the proposed system should be considered as an expandable research prototype that paves the way for future developments and evaluations.

8.4 Future Work

Given the results of this research and the limitations mentioned, there are several paths for developing and extending the proposed system that can improve its performance, accuracy, and usability.

One of the most important future directions is to develop the system towards context-aware processing. In the current version, linguistic analysis is performed at the sentence level, while considering the context at the paragraph or whole text level can lead to more accurate and meaningful suggestions, especially in academic texts or complex structures [28], [29].

Another path is to conduct user studies on a larger scale. Such studies can help to quantitatively assess the usability of the system, the level of user acceptance, and its impact on the quality of writing. Also, examining user behavior in real-world scenarios can provide valuable information for optimizing interaction design [15], [27].

From a technical perspective, improving the accuracy and robustness of gaze data is also an important direction for future work. Using more advanced algorithms to detect fixation or combining gaze with other interactive signals (such as mouse or keyboard activity) can help reduce errors and increase the robustness of the system [11], [19].

In addition, using more advanced or fine-tuned language models can improve the quality of language outputs and allow the system to be adapted to specific user needs, such as scientific writing or language learning [23], [28].

Another development path is to extend the system towards multilingual support. This capability can increase the scope of the system and make it more usable for users with different languages, especially in international environments [26].

Finally, we can point to the development of the system towards adaptive interactions, in which the system learns from the user's behavior and adjusts fixation thresholds, feedback type, and intervention timing in a personalized manner. Such an approach can significantly improve the user experience and make the system more intelligent and user-centric [15], [48].

Overall, the system presented in this research can be considered a starting point for the development of a new generation of writing assistants based on implicit and intelligent interaction. The proposed future directions show that this area has a high potential for research and development and can lead to the creation of systems that support users in the writing process more naturally and effectively.

Bibliography

[1] Dale, R., & Kilgarriff, A. (2011). Helping Our Own: The HOO 2011 Pilot Shared Task. Proceedings of the 13th European Workshop on Natural Language Generation. https://aclanthology.org/W11-2815

[2] Brown, T. B., et al. (2020). Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems. https://arxiv.org/abs/2005.14165

[3] Vaswani, A., et al. (2017). Attention Is All You Need. Advances in Neural Information Processing Systems. https://arxiv.org/abs/1706.03762

[4] Duchowski, A. T. (2017). Eye Tracking Methodology: Theory and Practice (3rd ed.). Springer. https://doi.org/10.1007/978-3-319-57883-5

[5] Jacob, R. J. K., & Karn, K. S. (2003). Eye tracking in human–computer interaction and usability research. In The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research. Elsevier. https://doi.org/10.1016/B978-044451020-4/50031-1

[6] Klaib, A. F., Alsrehin, N. O., Melhem, W. Y., Bashtawi, H. O., & Magableh, A. A. (2021). Eye tracking algorithms, techniques, tools, and applications with an emphasis on machine learning and Internet of Things technologies. Expert Systems with Applications, 166, 114037. https://doi.org/10.1016/j.eswa.2020.114037

[7] Aizenman, A. M., Koulieris, G. A., Gibaldi, A., Sehgal, V., Levi, D. M., & Banks, M. S. (2023). The Statistics of Eye Movements and Binocular Disparities during VR Gaming: Implications for Headset Design. ACM Transactions on Graphics, 42(1). https://doi.org/10.1145/3549529

[8] Valliappan, N., et al. (2020). Accelerating eye movement research via accurate and affordable smartphone eye tracking. Nature Communications, 11(1), 4553. https://doi.org/10.1038/s41467-020-18360-5

[9] Sigut, J., & Sidha, S.-A. (2011). Iris Center Corneal Reflection Method for Gaze Tracking Using Visible Light. IEEE Transactions on Biomedical Engineering, 58(2), 411–419. https://doi.org/10.1109/TBME.2010.2087330

[10] Zhang, X., Park, S., & Feit, A. M. (2021). Eye Gaze Estimation and Its Applications. In Artificial Intelligence for Human Computer Interaction: A Modern Approach. Springer. https://link.springer.com/chapter/10.1007/978-3-030-82681-9_4

[11] Nyström, M., & Holmqvist, K. (2010). An adaptive algorithm for fixation, saccade, and glissade detection in eyetracking data. Behavior Research Methods, 42(1), 188–204. https://doi.org/10.3758/BRM.42.1.188

[12] Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. Psychological Bulletin, 124(3), 372–422. https://doi.org/10.1037/0033-2909.124.3.372

[13] Kowler, E. (2019). Predictive smooth pursuit eye movements. Annual Review of Vision Science, 5, 223–246. https://pubmed.ncbi.nlm.nih.gov/31283450/

[14] Lisberger, S. G. (2010). Visual guidance of smooth-pursuit eye movements. Neuron, 66(4), 477–491. https://pubmed.ncbi.nlm.nih.gov/20510853/

[15] Qvarfordt, P. (2017). Gaze-informed multimodal interaction. In The Handbook of Multimodal-Multisensor Interfaces (pp. 365–402). ACM and Morgan & Claypool. https://doi.org/10.1145/3015783.3015794

[16] Fischer-Janzen, A., et al. (2024). A scoping review of gaze and eye tracking-based control in assistive robots. Frontiers in Robotics and AI, 11, 1326670. https://doi.org/10.3389/frobt.2024.1326670

[17] Majaranta, P., et al. (2009). Fast gaze typing with an adjustable dwell time. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 357–360. https://doi.org/10.1145/1518701.1518758

[18] Huckauf, A., & Urbina, M. H. (2008). On object selection in gaze controlled environments. Journal of Eye Movement Research, 2(4), 1–7. https://doi.org/10.16910/jemr.2.4.4

[19] Feit, A. M., et al. (2017). Toward everyday gaze input: Accuracy and precision of eye tracking and implications for design. Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, 1118–1130. https://doi.org/10.1145/3025453.3025599

[20] Ooms, K., & Krassanakis, V. (2018). Measuring the spatial noise of a low-cost eye tracker to enhance fixation detection. Journal of Imaging, 4(8), 96. https://doi.org/10.3390/jimaging4080096

[21] Blignaut, P. (2009). Fixation identification: The optimum threshold for a dispersion algorithm. Attention, Perception, & Psychophysics, 71(4), 881–895. https://doi.org/10.3758/APP.71.4.881

[22] Majaranta, P., & Bulling, A. (2014). Eye tracking and eye-based human-computer interaction. In Advances in Physiological Computing (pp. 39–65). Springer. https://doi.org/10.1007/978-1-4471-6392-3_3

[23] Zhao, W. X., et al. (2023). A Survey of Large Language Models. arXiv. https://arxiv.org/abs/2303.18223

[24] Bekker, M. (2024). Large language models and academic writing: Five tiers of engagement. South African Journal of Science, 120(1/2). https://doi.org/10.17159/sajs.2024/17147

[25] Naveed, H., et al. (2023). A Comprehensive Overview of Large Language Models. arXiv. https://arxiv.org/abs/2307.06435

[26] Nayak, P., & Narayan, K. A. (2024). Large language models and the future of academic writing. Journal of Oral and Maxillofacial Pathology, 28(Suppl 1), S95–S101. https://pmc.ncbi.nlm.nih.gov/articles/PMC11160983/

[27] Mugaanyi, J., Sserunkuma, R., & Batte, A. (2024). Evaluation of Large Language Model Performance and Reliability as an Academic Writing Assistant: Usability Study. JMIR AI, 3, e52935. https://www.jmir.org/2024/1/e52935/

[28] Du, Z., et al. (2025). AcademiCraft: Transforming Writing Assistance for English for Academic Purposes With Large Language Models and Multi-Agent Systems. Information, 16(4), 254. https://www.mdpi.com/2078-2489/16/4/254

[29] Du, Z., et al. (2025). Exploring sentence-level revision capabilities of large language models in academic writing assistance. International Journal of Educational Technology in Higher Education. https://link.springer.com/article/10.1186/s40862-025-00334-z

[30] Sahoo, P., et al. (2024). A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. arXiv. https://arxiv.org/abs/2402.07927

[31] Kojima, T., et al. (2022). Large Language Models are Zero-Shot Reasoners. arXiv. https://arxiv.org/abs/2205.11916

[32] OpenAI. (2024). Structured model outputs. OpenAI API Documentation. https://developers.openai.com/api/docs/guides/structured-outputs/

[33] Cuskley, C., et al. (2024). The limitations of large language models for understanding human language and cognition. Frontiers in Artificial Intelligence, 7, 1416490. https://pmc.ncbi.nlm.nih.gov/articles/PMC11370970/

[34] Chen, K., et al. (2025). A Survey on Privacy Risks and Protection in Large Language Models. arXiv. https://arxiv.org/abs/2505.01976

[35] Das, B. C., Amini, M. H., & Wu, Y. (2024). Security and Privacy Challenges of Large Language Models: A Survey. arXiv. https://arxiv.org/abs/2402.00888

[36] Grammarly. (2026). Product features. https://www.grammarly.com/features

[37] Grammarly. (2026). AI writing assistant. https://www.grammarly.com/ai-writing-assistant

[38] Grammarly. (2026). AI sentence rewriter. https://www.grammarly.com/ai/ai-writing-tools/sentence-rewriter

[39] LanguageTool. (2026). Free AI grammar checker. https://languagetool.org/

[40] LanguageTool. (2026). Multilingual grammar checker: Languages supported. https://languagetool.org/insights/post/product-languages-supported/

[41] LanguageTool. (2026). LanguageTool for Teams / business features. https://languagetool.org/business

[42] Zhang, J., et al. (2024). Bibliometrically and systematically analyzing automated writing evaluation for English as a second language. Forum for Linguistic Studies. https://journals.bilpubgroup.com/index.php/fls/article/view/6463

[43] Lin, Y., et al. (2025). Elucidating university students' intentions to seek assistance from automated writing evaluation tools. Humanities and Social Sciences Communications. https://www.nature.com/articles/s41599-024-03861-1

[44] Barrot, J. S. (2025). Trinka: Facilitating academic writing through an intelligent writing evaluation system. Computer Assisted Language Learning. https://www.sciencedirect.com/science/article/abs/pii/S1075293525000406

[45] Deep, P. D., et al. (2025). Impacts on writing skills, critical thinking, and integrity in higher education of AI writing assistants. Sociologies, 15(9), 247. https://www.mdpi.com/2075-4698/15/9/247

[46] Dritsas, E., & Trigka, M. (2025). Multimodal Interaction, Interfaces, and Communication. Multimodal Technologies and Interaction, 9(1), 6. https://www.mdpi.com/2414-4088/9/1/6

[47] Korkmaz, A., et al. (2026). Mapping Eye-Tracking Research in Human–Computer Interaction: A Scientometric Review. PMC. https://pmc.ncbi.nlm.nih.gov/articles/PMC12921980/

[48] Langner, M., Toreini, P., & Maedche, A. (2023). Leveraging Eye Tracking Technology for a Situation-Aware Writing Assistant. Proceedings of the 2023 Symposium on Eye Tracking Research and Applications (ETRA '23). https://doi.org/10.1145/3588015.3590121

[49] Parisay, M., et al. (2021). EyeTAP: Introducing a multimodal gaze-based technique for point-and-select tasks. International Journal of Human-Computer Studies. https://www.sciencedirect.com/science/article/abs/pii/S107158192100094X

[50] Guo, D., et al. (2025). Multimodal Behavioral Patterns Analysis with Eye-Tracking in Human–AI Collaboration. ACM Digital Library. https://dl.acm.org/doi/full/10.1145/3716553.3750787

[51] Santhosh, J., et al. (2024). Gaze-Driven Adaptive Learning System With ChatGPT-Generated Summaries. DFKI / published PDF. https://www.dfki.de/fileadmin/user_upload/import/15571_Gaze-Driven_Adaptive_Learning_System_With_ChatGPT-Generated_Summaries.pdf

[52] Cai, S., et al. (2024). Using large language models to accelerate communication for eye gaze typing users with ALS. Nature Communications, 15(1). https://doi.org/10.1038/s41467-024-53873-3

[53] Singh, A., et al. (2025). Using Lookback Fixations to Enhance Human-LLM Alignment for Text Completion. Proceedings of BEA 2025. https://doi.org/10.18653/v1/2025.bea-1.61

[54] Qt Documentation. (2026). QPlainTextEdit Class / Qt for Python Documentation. Qt 6 Documentation. https://doc.qt.io/qt-6/qplaintextedit.html

https://doc.qt.io/qtforpython-6/

[55] Qt Documentation. (2026). QThread Class and QObject Thread Affinity. Qt 6 Documentation. https://doc.qt.io/qt-6/qthread.html

https://doc.qt.io/qt-6/qobject.html

[56] Gazepoint. Open Gaze API v2.0. Gazepoint API Documentation. https://www.gazept.com/dl/Gazepoint_API_v2.0.pdf

[57] Qt Documentation. (2026). QMainWindow Class. Qt 6 Documentation. https://doc.qt.io/qt-6/qmainwindow.html

https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/QMainWindow.html

[58] Qt Documentation. (2026). QWidget Class. Qt 6 Documentation. https://doc.qt.io/qt-6/qwidget.html

[59] Qt Documentation. (2026). QScrollArea Class. Qt 6 Documentation. https://doc.qt.io/qt-6/qscrollarea.html

[60] Qt Documentation. (2026). QObject Class / Signals and Slots Overview. Qt 6 Documentation. https://doc.qt.io/qt-6/qobject.html

https://doc.qt.io/qtforpython-6/overviews/qtcore-signalsandslots.html

[61] Qt for Python Documentation. (2026). Signals and Slots Tutorial. Qt for Python Documentation. https://doc.qt.io/qtforpython-6/tutorials/basictutorial/signals_and_slots.html

[62] Python Software Foundation. (2026). csv — CSV File Reading and Writing. Python Standard Library Documentation. https://docs.python.org/3/library/csv.html

[63] Python Software Foundation. (2026). pathlib — Object-oriented filesystem paths. Python Standard Library Documentation. https://docs.python.org/3/library/pathlib.html

[64] Python Software Foundation. (2026). datetime — Basic date and time types. Python Standard Library Documentation. https://docs.python.org/3/library/datetime.html

[65] Python Software Foundation. (2026). Errors and Exceptions. Python Documentation. https://docs.python.org/3/tutorial/errors.html

[66] Python Software Foundation. (2026). Built-in Exceptions. Python Standard Library Documentation. https://docs.python.org/3/library/exceptions.html

[67] OpenAI. (2026). Error codes. OpenAI API Documentation. https://developers.openai.com/api/docs/guides/error-codes/

[68] OpenAI. (2026). Production best practices / Rate limits. OpenAI API Documentation. https://developers.openai.com/api/docs/guides/production-best-practices/

https://developers.openai.com/api/docs/guides/rate-limits/

[69] Qt Documentation. (2026). QCoreApplication Class. Qt 6 Documentation. https://doc.qt.io/qt-6/qcoreapplication.html

[70] Qt for Python Documentation. (2026). PySide6.QtCore.QThread. Qt for Python Documentation. https://doc.qt.io/qtforpython-6/PySide6/QtCore/QThread.html